aws

개발자 가이드

Amazon Lookout for Vision

Amazon Lookout for Vision: 개발자 가이드

Table of Contents

지원 종료 공지: 2025 AWS 년 10월 31일에는 Amazon Lookout for Vision에 대한 지원을 중단할 예정입니다. 2025년 10월 31일 이후에는 Lookout for Vision 콘솔 또는 Lookout for Vision 리소스에 더 이상 액세스할 수 없습니다. 자세한 내용은이 블로그 게시물을 참조하십시오.

기계 번역으로 제공되는 번역입니다. 제공된 번역과 원본 영어의 내용이 상충하는 경우에는 영어 버전이 우선합니다.

Amazon Lookout for Vision이란 무엇입니까?

Amazon Lookout for Vision을 사용하여 산업용 제품의 시각적 결함을 대규모로 정확하게 찾아낼 수 있습니다. 컴퓨터 비전을 사용하여 산업용 제품의 누락된 구성 요소, 차량 또는 구조물의 손상, 생산 라인의 불규칙성, 실리콘 웨이퍼의 미세한 결함까지 식별합니다. 인쇄 회로 기판의 커패시터 누락과 같이 품질이 중요한 기타 물리적 품목도 식별할 수 있습니다.

주요 이점

Amazon Lookout for Vision은 다음과 같은 이점을 제공합니다.

- 프로세스를 빠르고 효율적으로 개선 — Amazon Lookout for Vision을 사용하여 산업 프로세스에서 컴퓨터 비전 기반 검사를 대규모로 빠르고 효율적으로 구현할 수 있습니다. 최소 30개의 기본 양호한 이미지를 제공할 수 있으며 Lookout for Vision은 결함 감지를 위한 사용자 지정 ML 모델을 자동으로 구축할 수 있습니다. 그런 다음 IP 카메라의 이미지를 일괄 또는 실시간으로 처리하여 찌그러짐, 균열, 스크래치와 같은 이상 현상을 빠르고 정확하게 식별할 수 있습니다.
- 생산 품질을 빠르게 향상 — Amazon Lookout for Vision을 사용하면 생산 프로세스의 결함을 실시간으로 줄일 수 있습니다. 대시보드에서 시각적 이상을 식별하고 보고하므로 더 많은 결함이 발생하지 않도록 신속하게 조치를 취하여 생산 품질을 높이고 비용을 절감할 수 있습니다.
- 운영 비용 절감 — Amazon Lookout for Vision은 결함률이 가장 높은 프로세스를 식별하거나 결함의 최근 변동을 표시하는 등 시각적 검사 데이터의 추세를 보고합니다. 이 정보를 사용하여 비용이 많이 드는 예상치 못한 가동 중지 시간이 발생하기 전에 프로세스 라인에서 유지 관리 일정을 잡을지 아니면 생산 경로를 다른 기계로 변경할지 결정할 수 있습니다.

Amazon Lookout for Vision 최종 사용자를 처음 사용하시나요?

Amazon Lookout for Vision를 처음 사용하시는 경우 다음 단원을 순서대로 읽는 것을 추천드립니다.

1. Amazon Lookout for Vision 설정하기 – 이 단원에서는 계정을 설정합니다.
2. Amazon Lookout for Vision을 시작하기— 이 섹션에서는 첫 번째 Amazon Lookout for Vision 모델을 만드는 방법에 대해 알아봅니다.

Amazon Lookout for Vision 설정하기

이 섹션에서는 AWS 계정에 가입하고 Amazon Lookout for Vision을 설정합니다.

Amazon Lookout for Vision을 지원하는 AWS 리전에 대한 자세한 내용은 Amazon Lookout for Vision 엔드포인트 및 할당량을 참조하세요.

주제

1단계: AWS 계정 생성

이 단계에서는 AWS 계정에 가입하고 관리 사용자를 생성합니다.

주제

에 가입 AWS 계정

이 없는 경우 다음 단계를 AWS 계정완료하여 생성합니다.

에 가입하려면 AWS 계정

1. https://portal.aws.amazon.com/billing/signup을 엽니다.
2. 온라인 지시 사항을 따릅니다.

   등록 절차 중 전화 또는 텍스트 메시지를 받고 전화 키패드로 확인 코드를 입력하는 과정이 있습니다.

   에 가입하면 AWS 계정AWS 계정 루트 사용자이 생성됩니다. 루트 사용자에게는 계정의 모든 AWS 서비스 및 리소스에 액세스할 권한이 있습니다. 보안 모범 사례는 사용자에게 관리 액세스

권한을 할당하고, 루트 사용자만 사용하여 루트 사용자 액세스 권한이 필요한 작업을 수행하는 것입니다.

AWS 는 가입 프로세스가 완료된 후 확인 이메일을 보냅니다. 언제든지 https://aws.amazon.com/으로 이동하고 내 계정을 선택하여 현재 계정 활동을 보고 계정을 관리할 수 있습니다.

관리자 액세스 권한이 있는 사용자 생성

에 가입한 후 일상적인 작업에 루트 사용자를 사용하지 않도록 관리 사용자를 AWS 계정보호 AWS IAM Identity Center, AWS 계정 루트 사용자활성화 및 생성합니다.

보안 AWS 계정 루트 사용자

1. 루트 사용자를 선택하고 AWS 계정 이메일 주소를 입력하여 계정 소유자AWS Management Console로에 로그인합니다. 다음 페이지에서 비밀번호를 입력합니다.

   루트 사용자를 사용하여 로그인하는 데 도움이 필요하면 AWS 로그인 User Guide의 루트 사용자로 로그인을 참조하세요.
2. 루트 사용자의 다중 인증(MFA)을 활성화합니다.

   지침은 IAM 사용 설명서의 AWS 계정 루트 사용자(콘솔)에 대한 가상 MFA 디바이스 활성화를 참조하세요.

관리자 액세스 권한이 있는 사용자 생성

1. IAM Identity Center를 활성화합니다.

   지침은 AWS IAM Identity Center 사용 설명서의 AWS IAM Identity Center설정을 참조하세요.
2. IAM Identity Center에서 사용자에게 관리 액세스 권한을 부여합니다.

   를 자격 증명 소스 IAM Identity Center 디렉터리 로 사용하는 방법에 대한 자습서는 사용 AWS IAM Identity Center 설명서의 기본값으로 사용자 액세스 구성을 IAM Identity Center 디렉터리 참조하세요.

관리 액세스 권한이 있는 사용자로 로그인

- IAM IDentity Center 사용자로 로그인하려면 IAM Identity Center 사용자를 생성할 때 이메일 주소로 전송된 로그인 URL을 사용합니다.

IAM Identity Center 사용자를 사용하여 로그인하는 데 도움이 필요하면 AWS 로그인 사용 설명서의 AWS 액세스 포털에 로그인을 참조하세요.

추가 사용자에게 액세스 권한 할당

1. IAM Identity Center에서 최소 권한 적용 모범 사례를 따르는 권한 세트를 생성합니다.

   지침은AWS IAM Identity Center 사용 설명서의 Create a permission set를 참조하세요.
2. 사용자를 그룹에 할당하고, 그룹에 Single Sign-On 액세스 권한을 할당합니다.

   지침은 AWS IAM Identity Center 사용 설명서의 Add groups를 참조하세요.

2단계: 권한 설정

Amazon Lookout for Vision을 사용하려면 Lookout for Vision 콘솔, AWS SDK 작업 및 모델 훈련에 사용하는 Amazon S3 버킷에 대한 액세스 권한이 필요합니다.

> **Note**
>
> AWS SDK 작업만 사용하는 경우 AWS SDK 작업으로 범위가 지정된 정책을 사용할 수 있습니다. 자세한 내용은 SDK 권한 설정 단원을 참조하십시오.

주제

- AWS 관리형 정책을 사용하여 콘솔 액세스 설정
- Amazon S3 버킷 권한 설정
- 권한 할당

AWS 관리형 정책을 사용하여 콘솔 액세스 설정

다음 AWS 관리형 정책을 사용하여 Amazon Lookout for Vision 콘솔 및 SDK 작업에 적절한 액세스 권한을 적용합니다.

- AmazonLookoutVisionConsoleFullAccess - Amazon Lookout for Vision 콘솔 및 SDK 작업에 대한 전체 액세스를 허용합니다. 콘솔 버킷을 생성하려면 `AmazonLookoutVisionConsoleFullAccess` 권한이 필요합니다. 자세한 내용은 3단계: 콘솔 버킷 생성 단원을 참조하십시오.

- AmazonLookoutVisionConsoleReadOnlyAccess - Amazon Lookout for Vision 콘솔 및 SDK 작업에 대한 읽기 전용 액세스를 허용합니다.

권한을 할당하려면 권한 할당 항목을 참조하세요.

AWS 관리형 정책에 대한 자세한 내용은 AWS 관리형 정책을 참조하세요.

Amazon S3 버킷 권한 설정

Amazon Lookout for Vision은 Amazon S3 버킷을 사용하여 다음 파일을 저장합니다.

- 데이터세트 이미지 — 모델 학습에 사용되는 이미지. 자세한 내용은 데이터 세트를 생성합니다. 단원을 참조하십시오.
- Amazon SageMaker AI Ground Truth 형식 매니페스트 파일. 예를 들어 SageMaker AI GroundTruth 작업의 매니페스트 파일 출력입니다. 자세한 내용은 Amazon SageMaker AI Ground Truth 매니페스트 파일을 사용하여 데이터 세트 생성 단원을 참조하십시오.
- 모델 학습의 결과입니다.

콘솔을 사용하는 경우 Lookout for Vision은 프로젝트를 관리하기 위한 Amazon S3 버킷(콘솔 버킷)을 생성합니다. `LookoutVisionConsoleReadOnlyAccess` 및 `LookoutVisionConsoleFullAccess` 관리형 정책에는 콘솔 버킷에 대한 Amazon S3 액세스 권한이 포함됩니다.

콘솔 버킷을 사용하여 데이터세트 이미지와 SageMaker AI Ground Truth 형식 매니페스트 파일을 저장할 수 있습니다. 다른 Amazon S3 버킷을 사용할 수도 있습니다. 버킷은 AWS 계정이 소유해야 하며 Lookout for Vision을 사용하는 AWS 리전에 있어야 합니다.

다른 버킷을 사용하려면 다음 정책을 원하는 사용자 또는 그룹에 추가합니다. `amzn-s3-demo-bucket`를 대상 버킷의 이름으로 바꿉니다. IAM 정책 생성에 대한 자세한 내용은 IAM 정책 생성을 참조하세요.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "LookoutVisionS3BucketAccessPermissions",
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation",
```

```
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket"
            ]
        },
        {
            "Sid": "LookoutVisionS3ObjectAccessPermissions",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket/*"
            ]
        }
    ]
}
```

권한을 할당하려면 권한 할당 항목을 참조하세요.

권한 할당

액세스 권한을 제공하려면 사용자, 그룹 또는 역할에 권한을 추가하세요:

- 의 사용자 및 그룹 AWS IAM Identity Center:

  권한 세트를 생성합니다. AWS IAM Identity Center 사용 설명서의 권한 세트 생성의 지침을 따릅니다.

- 보안 인증 공급자를 통해 IAM에서 관리되는 사용자:

  ID 페더레이션을 위한 역할을 생성합니다. IAM 사용 설명서의 Create a role for a third-party identity provider (federation)의 지침을 따릅니다.

- IAM 사용자:
  - 사용자가 맡을 수 있는 역할을 생성합니다. IAM 사용 설명서에서 Create a role for an IAM user의 지침을 따릅니다.
  - (권장되지 않음)정책을 사용자에게 직접 연결하거나 사용자를 사용자 그룹에 추가합니다. IAM 사용 설명서에서 사용자(콘솔)에 권한 추가의 지침을 따르세요.

3단계: 콘솔 버킷 생성

Amazon Lookout for Vision 콘솔을 사용하려면 콘솔 버킷으로 알려진 Amazon S3 버킷이 필요합니다. 콘솔 버킷에는 다음이 저장됩니다.

- 콘솔을 사용하여 데이터세트에 업로드하는 이미지.
- 콘솔에서 시작하는 모델 학습의 학습 결과.
- 평가판 탐지 결과.
- 콘솔을 사용하여 S3 버킷의 이미지에 레이블을 자동으로 지정하여 데이터 세트를 생성할 때 콘솔이 생성하는 임시 매니페스트 파일입니다. 콘솔은 매니페스트 파일을 삭제하지 않습니다.

새 AWS 리전에서 Amazon Lookout for Vision 콘솔을 처음 열면 Lookout for Vision이 사용자를 대신하여 콘솔 버킷을 생성합니다. AWS SDK 작업 또는 데이터 세트 생성과 같은 콘솔 작업에 버킷 이름을 사용해야 할 수 있으므로 콘솔 버킷 이름을 기록해 둡니다.

대신, Amazon S3 콘솔을 사용하여 콘솔 버킷을 생성할 수 있습니다. Amazon S3 버킷 정책에 따라 Amazon Lookout for Vision 콘솔에서 콘솔 버킷을 성공적으로 생성할 수 없는 경우 이 접근 방식을 사용하세요. Amazon S3 버킷의 자동 생성을 허용하지 않는 정책을 예시로 들 수 있습니다.

> **Note**
>
> Lookout for Vision 콘솔이 아닌 AWS SDK만 사용하는 경우 콘솔 버킷을 생성할 필요가 없습니다. 원하는 이름을 가진 다른 S3 버킷을 사용할 수 있습니다.

콘솔 버킷 이름의 형식은 `lookoutvision-<region>-<random value>`입니다. 임의 값은 버킷 이름 간에 충돌이 발생하지 않도록 합니다.

주제

- Amazon Lookout for Vision 콘솔을 사용하여 콘솔 버킷 생성
- Amazon S3로 콘솔 버킷을 생성합니다.
- 콘솔 버킷 설정

Amazon Lookout for Vision 콘솔을 사용하여 콘솔 버킷 생성

다음 절차에 따라 Amazon Lookout for Vision 콘솔을 사용하여 AWS 리전에 대한 콘솔 버킷을 생성합니다. 활성화한 S3 버킷 설정에 대한 자세한 내용은 콘솔 버킷 설정을 참조하십시오.

Amazon Lookout for Vision 콘솔을 사용하여 콘솔 버킷을 만들려면

1. 사용 중인 사용자 또는 그룹에 `AmazonLookoutVisionConsoleFullAccess` 권한이 있는지 확인하십시오. 자세한 내용은 2단계: 권한 설정 단원을 참조하십시오.
2. https://console.aws.amazon.com/lookoutvision/에서 Amazon Lookout for Vision을 엽니다.
3. 탐색 바에서 지역 선택을 선택합니다. 그런 다음 콘솔 버킷을 생성할 AWS 리전을 선택합니다.
4. Get started를 선택합니다.
5. 현재 AWS 리전에서 콘솔을 처음 연 경우 최초 설정 대화 상자가 표시됩니다.

   a. 표시된 Amazon S3 버킷의 이름을 복사합니다. 나중에 이 정보가 필요합니다.

   b. Amazon Lookout for Vision이 사용자를 대신하여 콘솔 버킷을 생성하도록 하려면 S3 버킷 생성을 선택합니다.

   현재 AWS 리전의 콘솔 버킷이 이미 있는 경우 최초 설정 대화 상자가 표시되지 않습니다.
6. 브라우저 창을 닫습니다.

Amazon S3로 콘솔 버킷을 생성합니다.

Amazon S3을 사용하여 콘솔 버킷을 생성할 수 있습니다. Amazon S3 버전 관리를 활성화한 상태로 버킷을 생성해야 합니다. Amazon S3 수명 주기 구성을 사용하여 객체의 최신 버전이 아닌 (이전) 버전을 제거하고 완료되지 않은 멀티파트 업로드를 삭제하는 것이 좋습니다. 객체의 최신 버전을 삭제하는 수명 주기 구성은 권장하지 않습니다. Amazon Lookout for Vision 콘솔을 사용하여 생성한 콘솔 버킷에 대해 활성화하는 S3 버킷 설정에 대한 자세한 내용은 콘솔 버킷 설정을 참조하십시오.

1. 콘솔 버킷을 생성할 AWS 리전을 결정합니다. 지원되는 지역에 대한 자세한 내용은 Amazon Lookout for Vision 엔드포인트 및 할당량을 참조하십시오.
2. 버킷 생성의 S3 콘솔 지침을 사용하여 버킷을 생성합니다. 다음을 수행합니다.

   a. 3단계에서는 앞에 `lookoutvision-`*`region`*`-`*`your-identifier`*가 붙은 버킷 이름을 지정합니다. `region`을 이전 단계에서 선택한 지역 코드로 변경합니다. 선택한 고유 식별자로 `your-identifier`을 변경하십시오.

b. 4단계에서 사용할 AWS 리전을 선택합니다.

3. 버킷 버전 관리 활성화의 S3 콘솔 지침에 따라 버킷의 버전 관리를 활성화하십시오.
4. (선택 사항) 버킷의 수명 주기 구성 설정의 S3 콘솔 지침에 따라 버킷의 수명 주기 구성을 지정합니다. 객체의 최신 버전이 아닌 (이전) 버전을 제거하고 미완료 멀티파트 업로드를 삭제하려면 다음을 수행하십시오. 6, 8, 9, 10단계는 수행할 필요가 없습니다.

   a. 5단계에서는 버킷의 모든 객체에 적용을 선택합니다.

   b. 7단계에서는 최신이 아닌 버전의 객체 영구 삭제 및 만료된 객체 삭제 마커 또는 미완료 멀티파트 업로드 삭제를 선택합니다.

   c. 11단계의 경우 객체의 최신 버전이 아닌 버전을 삭제하기 전에 대기할 일 수를 입력합니다.

   d. 12단계에서는 완료되지 않은 멀티파트 업로드를 삭제하기 전까지 기다릴 일수를 입력합니다.

콘솔 버킷 설정

Amazon Lookout for Vision 콘솔을 사용하여 콘솔 버킷을 생성하는 경우 콘솔 버킷에서 다음 설정을 활성화합니다.

- 콘솔 버킷의 객체 버전 관리.
- 콘솔 버킷에 있는 객체의 서버 측 암호화.
- 최신이 아닌 객체 삭제 (30일) 및 미완료 멀티파트 업로드 (3일) 를 위한 수명 주기 구성.
- 콘솔 버킷에 대한 퍼블릭 액세스를 차단합니다.

4단계: AWS CLI 및 AWS SDKs 설정

다음 단계에서는 AWS Command Line Interface (AWS CLI) 및 AWS SDKs를 설치하는 방법을 보여줍니다. 이 설명서의 예제에서는 AWS CLI Python 및 AWS SDKs 사용합니다.

주제

- AWS SDKS 설치
- 프로그래밍 방식 액세스 권한 부여
- SDK 권한 설정
- Amazon Lookout for Vision 작업 호출

AWS SDKS 설치

단계에 따라 AWS SDKs를 다운로드하고 구성합니다.

AWS CLI 및 AWS SDKs를 설정하려면

- 및 사용하려는 AWS SDKs를 다운로드AWS CLI하여 설치합니다. 이 가이드에서는 AWS CLI, Java 및 Python에 대한 예제를 제공합니다. SDK 설치 AWS SDKs. https://aws.amazon.com/tools/

프로그래밍 방식 액세스 권한 부여

이 안내서의 AWS CLI 및 코드 예제는 로컬 컴퓨터 또는 Amazon Elastic Compute Cloud 인스턴스와 같은 기타 AWS 환경에서 실행할 수 있습니다. 예제를 실행하려면 예제에서 사용하는 AWS SDK 작업에 대한 액세스 권한을 부여해야 합니다.

주제

- 로컬 컴퓨터에서 코드 실행
- AWS 환경에서 코드 실행

로컬 컴퓨터에서 코드 실행

로컬 컴퓨터에서 코드를 실행하려면 단기 자격 증명을 사용하여 사용자에게 AWS SDK 작업에 대한 액세스 권한을 부여하는 것이 좋습니다. 로컬 컴퓨터에서 AWS CLI 및 코드 예제를 실행하는 방법에 대한 자세한 내용은 섹션을 참조하세요로컬 컴퓨터에서 프로필 사용.

사용자는 AWS 외부에서와 상호 작용하려는 경우 프로그래밍 방식으로 액세스해야 합니다 AWS Management Console. 프로그래밍 방식 액세스를 부여하는 방법은에 액세스하는 사용자 유형에 따라 다릅니다 AWS.

사용자에게 프로그래밍 방식 액세스 권한을 부여하려면 다음 옵션 중 하나를 선택합니다.

프로그래밍 방식 액세스가 필요한 사용자는 누구인가요?	To	액세스 권한을 부여하는 사용자
작업 인력 ID (IAM Identity Center가 관리하는 사용자)	임시 자격 증명을 사용하여 AWS CLI, AWS SDKs 또는 AWS APIs.	사용하고자 하는 인터페이스에 대한 지침을 따릅니다.

프로그래밍 방식 액세스가 필요한 사용자는 누구인가요?	To	액세스 권한을 부여하는 사용자
		• 자세한 AWS CLI내용은 AWS Command Line Interface 사용 설명서의 AWS CLI 를 사용하도록 구성을 AWS IAM Identity Center 참조하세요. • AWS SDKs, 도구 및 AWS APIs 경우 SDK 및 도구 참조 안내서의 IAM Identity Center 인증을 참조하세요. AWS SDKs
IAM	임시 자격 증명을 사용하여 AWS CLI, AWS SDKs 또는 AWS APIs.	IAM 사용 설명서의 AWS 리소스에서 임시 자격 증명 사용의 지침을 따릅니다.
IAM	(권장되지 않음) 장기 자격 증명을 사용하여 AWS CLI, AWS SDKs 또는 AWS APIs.	사용하고자 하는 인터페이스에 대한 지침을 따릅니다. • 자세한 AWS CLI내용은 AWS Command Line Interface 사용 설명서의 IAM 사용자 자격 증명을 사용한 인증을 참조하세요. • AWS SDKs 및 도구의 경우 SDK 및 도구 참조 안내서의 장기 자격 증명을 사용하여 인증을 참조하세요. AWS SDKs • AWS APIs 경우 IAM 사용 설명서의 IAM 사용자의 액세스 키 관리를 참조하세요.

로컬 컴퓨터에서 프로필 사용

에서 생성한 단기 자격 증명을 사용하여이 설명서의 AWS CLI 및 코드 예제를 실행할 수 있습니다로컬 컴퓨터에서 코드 실행. 보안 인증 정보 및 기타 설정 정보를 가져오려면, 예제에서는 `lookoutvision-access` 이름의 프로필을 사용합니다. 예를 들어:

```
session = boto3.Session(profile_name='lookoutvision-access')
lookoutvision_client = session.client("lookoutvision")
```

프로파일이 나타내는 사용자는 Lookout for Vision SDK 작업 및 예제에 필요한 기타 AWS SDK 작업을 호출할 수 있는 권한이 있어야 합니다. 자세한 내용은 SDK 권한 설정 단원을 참조하십시오. 권한을 할당하려면 권한 할당 항목을 참조하세요.

AWS CLI 및 코드 예제와 함께 작동하는 프로파일을 생성하려면 다음 중 하나를 선택합니다. 생성한 프로필의 이름이 `lookoutvision-access`인지 확인하세요.

- IAM으로 관리하는 사용자 — IAM 역할로 전환(AWS CLI)의 지침을 따르세요.
- 작업 인력 자격 증명(사용자 관리형 AWS IAM Identity Center) - 사용할 AWS CLI 구성의 AWS IAM Identity Center 지침을 따릅니다. 코드 예제의 경우 IAM ID 센터를 통한 인증을 지원하는 AWS 툴킷을 지원하는 통합 개발 환경(IDE)을 사용하는 것이 좋습니다. Java 예제는 Java로 구축 시작을 참조하세요. Python 예제는 Python으로 구축 시작을 참조하세요. 자세한 정보는 IAM Identity Center 보안 인증 정보를 참조하세요.

> **Note**
>
> 코드를 사용하여 단기 보안 인증 정보를 얻을 수 있습니다. 자세한 내용은 IAM 역할로 전환(API)을 참조하세요. IAM Identity Center의 경우 CLI 액세스를 위한 IAM 역할 자격 증명 가져오기의 지침에 따라 역할에 대한 단기 보안 인증 정보를 받으세요.

AWS 환경에서 코드 실행

AWS Lambda 함수에서 실행되는 프로덕션 코드와 같은 AWS 환경에서 사용자 자격 증명을 사용하여 AWS SDK 호출에 서명해서는 안 됩니다. 대신 코드에 필요한 권한을 정의하는 역할을 구성합니다. 그런 다음 코드가 실행되는 환경에 역할을 연결합니다. 역할을 연결하고 임시 보안 인증 정보를 사용할 수 있게 하는 방법은 코드가 실행되는 환경에 따라 달라집니다.

- AWS Lambda 함수 - Lambda 함수의 실행 역할을 수임할 때 Lambda가 함수에 자동으로 제공하는 임시 자격 증명을 사용합니다. 보안 인증 정보는 Lambda 환경 변수에서 사용할 수 있습니다. 프로필을 지정할 필요가 없습니다. 자세한 내용을 알아보려면 Lambda 실행 역할을 참조하세요.
- Amazon EC2 - Amazon EC2 인스턴스 메타데이터 엔드포인트 보안 인증 인증 정보를 사용합니다. 공급자는 Amazon EC2 인스턴스에 연결한 Amazon EC2 인스턴스 프로파일을 사용하여 자동으로 보안 인증 정보를 생성하고 새로 고칩니다. 자세한 내용은 IAM 역할을 사용하여 Amazon EC2 인스턴스에서 실행되는 애플리케이션에 권한 부여를 참조하세요.
- Amazon Elastic 컨테이너 서비스 - 컨테이너 보안 인증 정보 공급자를 사용하세요. Amazon ECS는 메타데이터 엔드포인트로 보안 인증 정보를 전송하고 새로 고칩니다. 지정한 작업 IAM 역할은 애플리케이션이 사용하는 보안 인증 정보를 관리하기 위한 전략을 제공합니다. 자세한 내용은 AWS 서비스와 상호 작용을 참조하세요.
- 그린그라스 디바이스 — TLS 상호 인증 프로토콜을 사용하여 AWS IoT Core에 연결할 때 X.509 인증서를 사용할 수 있습니다. 이러한 인증서를 통해 디바이스는 AWS 자격 증명 없이 AWS IoT와 상호 작용할 수 있습니다. 이 AWS IoT 자격 증명 공급자는 X.509 인증서를 사용하여 디바이스를 인증하고 제한된 권한의 임시 보안 토큰 형태로 AWS 보안 인증을 발급합니다. 자세한 내용은 AWS 서비스와 상호 작용을 참조하세요.

보안 인증 정보 공급자에 대한 자세한 정보는 표준화된 보안 인증 정보 공급자를 참조하세요.

SDK 권한 설정

Amazon Lookout for Vision SDK 작업을 사용하려면 Lookout for Vision API 및 모델 교육에 사용되는 Amazon S3 버킷에 대한 액세스 권한이 필요합니다.

주제

- SDK 작업 권한 부여
- Amazon S3 버킷 권한 부여
- 권한 할당

SDK 작업 권한 부여

작업을 수행하는 데 필요한 권한만 부여하는 것을 권장합니다(최소 권한 권한). 예를 들어 DetectAnomalies를 호출하려면 `lookoutvision:DetectAnomalies`를 수행할 수 있는 권한이 필요합니다. 작업에 대한 권한을 찾으려면 API 참조를 확인하세요.

애플리케이션을 막 시작하는 경우 필요한 특정 권한을 모를 수 있으므로 더 광범위한 권한으로 시작할 수 있습니다. AWS 관리형 정책은 시작하는 데 도움이 되는 권한을 제공합니다.

- AmazonLookoutvisionFullAccess — Amazon Lookout for Vision SDK 작업에 대한 전체 액세스를 허용합니다.
- AmazonLookOutVisionReadOnlyAccess — 읽기 전용 SDK 작업에 대한 액세스를 허용합니다.

콘솔의 관리형 정책은 SDK 작업에 대한 액세스 권한도 제공합니다. 자세한 내용은 2단계: 권한 설정 단원을 참조하십시오.

AWS 관리형 정책에 대한 자세한 내용은 AWS 관리형 정책을 참조하세요.

애플리케이션에 필요한 권한을 알고 있는 경우, 사용자에게 필요한 고객 관리형 정책을 정의하여 권한을 줄이세요. 자세한 내용은 고객 관리형 정책을 참조하세요.

> **Note**
>
> 시작 지침에는 `s3:PutObject` 권한이 필요합니다. 자세한 내용은 1단계: 매니페스트 파일 생성 및 이미지 업로드 단원을 참조하십시오.

권한을 할당하려면 권한 할당 항목을 참조하세요.

Amazon S3 버킷 권한 부여

모델을 학습시키려면 이미지, 매니페스트 파일 및 학습 결과를 저장할 수 있는 적절한 권한이 있는 Amazon S3 버킷이 필요합니다. 버킷은 AWS 계정이 소유해야 하며 Amazon Lookout for Vision을 사용하는 AWS 리전에 위치해야 합니다.

SDK 전용 관리형 정책 (`AmazonLookoutVisionFullAccess` 및 `AmazonLookoutVisionReadOnlyAccess`) 에는 Amazon S3 버킷 권한이 포함되지 않으므로 기존 콘솔 버킷을 포함하여 사용하는 버킷에 액세스하려면 다음 권한 정책을 적용해야 합니다.

콘솔 관리형 정책 (`AmazonLookoutVisionConsoleFullAccess` 및 `AmazonLookoutVisionConsoleReadOnlyAccess`)에는 콘솔 버킷에 대한 액세스 권한이 포함됩니다. SDK 작업으로 콘솔 버킷에 액세스하고 콘솔 관리형 정책 권한이 있는 경우 다음 정책을 사용할 필요가 없습니다. 자세한 내용은 2단계: 권한 설정 단원을 참조하십시오.

작업 권한 결정

다음 정보를 사용하여 수행하려는 작업에 필요한 권한을 결정하십시오.

데이터 세트 생성

CreateDataset으로 데이터 세트를 생성하려면 다음 권한이 필요합니다.

- `s3:GetBucketLocation`— Lookout for Vision에서 버킷이 Lookout for Vision을 사용하고 있는 지역과 동일한 지역에 있는지 확인할 수 있습니다.
- `s3:GetObject`— `DatasetSource` 입력 파라미터에 지정된 매니페스트 파일에 액세스할 수 있습니다. 매니페스트 파일의 정확한 S3 객체 버전을 지정하려면 매니페스트 파일에 대한 `s3:GetObjectVersion`도 필요합니다. 자세한 내용은 S3 버킷에서 버전 관리 사용을 참조하세요.

모델 생성

CreateModel을 사용하여 모델을 생성하려면 다음과 같은 권한이 필요합니다.

- `s3:GetBucketLocation`— Lookout for Vision에서 버킷이 Lookout for Vision을 사용하고 있는 지역과 동일한 지역에 있는지 확인할 수 있습니다.
- `s3:GetObject`— 프로젝트의 교육 및 테스트 데이터 세트에 지정된 이미지에 액세스할 수 있습니다.
- `s3:PutObject`— 훈련 결과를 지정된 버킷에 저장할 수 있는 권한을 허용합니다. `OutputConfig` 파라미터에 출력 버킷 위치를 지정합니다. 선택적으로, `S3Location` 입력 필드의 `Prefix` 필드에 지정된 객체 키로만 권한 범위를 좁힐 수 있습니다. 자세한 내용은 OutputConfig을 참조하세요.

이미지, 매니페스트 파일 및 교육 출력에 액세스

Amazon S3 버킷 권한은 Amazon Lookout for Vision 작업 응답을 보는 데 필요하지 않습니다. 작업 응답에서 참조되는 이미지, 매니페스트 파일 및 교육 출력에 액세스하려면 `s3:GetObject` 권한이 필요합니다. 버전이 지정된 Amazon S3 객체에 액세스하는 경우 `s3:GetObjectVersion` 권한이 필요합니다.

Amazon S3 버킷 정책 설정

다음 정책을 사용하여 데이터 세트 (`CreateDataset`) 생성, 모델 생성 (`CreateModel`), 이미지, 매니페스트 파일 및 학습 출력에 액세스하는 데 필요한 Amazon S3 버킷 권한을 지정할 수 있습니다. *`amzn-s3-demo-bucket`*의 값을 사용하려는 버킷의 이름으로 변경합니다.

정책을 필요에 맞게 조정할 수 있습니다. 자세한 내용은 작업 권한 결정 단원을 참조하십시오. 원하는 사용자에게 정책을 추가합니다. 자세한 내용은 IAM 정책 생성을 참조하십시오.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "LookoutVisionS3BucketAccess",
            "Effect": "Allow",
            "Action": "s3:GetBucketLocation",
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket"
            ],
            "Condition": {
                "Bool": {
                    "aws:ViaAWSService": "true"
                }
            }
        },
        {
            "Sid": "LookoutVisionS3ObjectAccess",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket/*"
            ],
            "Condition": {
                "Bool": {
                    "aws:ViaAWSService": "true"
                }
            }
        }
    ]
}
```

권한을 할당하려면 권한 할당 항목을 참조하세요.

권한 할당

액세스 권한을 제공하려면 사용자, 그룹 또는 역할에 권한을 추가하세요:

- 의 사용자 및 그룹 AWS IAM Identity Center:

  권한 세트를 생성합니다. AWS IAM Identity Center 사용 설명서의 권한 세트 생성의 지침을 따릅니다.
- 보안 인증 공급자를 통해 IAM에서 관리되는 사용자:

  ID 페더레이션을 위한 역할을 생성합니다. IAM 사용 설명서의 Create a role for a third-party identity provider (federation)의 지침을 따릅니다.
- IAM 사용자:
  - 사용자가 맡을 수 있는 역할을 생성합니다. IAM 사용 설명서에서 Create a role for an IAM user의 지침을 따릅니다.
  - (권장되지 않음)정책을 사용자에게 직접 연결하거나 사용자를 사용자 그룹에 추가합니다. IAM 사용 설명서에서 사용자(콘솔)에 권한 추가의 지침을 따르세요.

Amazon Lookout for Vision 작업 호출

다음 코드를 실행하여 Amazon Lookout for Vision API를 호출할 수 있는지 확인합니다. 코드는 현재 AWS 리전의 AWS 계정에 있는 프로젝트를 나열합니다. 이전에 프로젝트를 생성하지 않은 경우 응답은 비어 있지만 `ListProjects` 작업을 호출할 수 있다는 확인은 표시됩니다.

일반적으로 예제 함수를 호출하려면 AWS SDK Lookout for Vision 클라이언트 및 기타 필수 파라미터가 필요합니다. AWS SDK Lookout for Vision 클라이언트는 기본 함수에서 선언됩니다.

코드가 실패하면 사용하는 사용자에게 올바른 권한이 있는지 확인합니다. 또한 Amazon Lookout for Vision을 일부 AWS 리전에서 사용할 수 없으므로 사용하는 AWS 리전을 확인합니다.

Amazon Lookout for Vision 작업을 호출하려면

1. 아직 설치하지 않은 경우 및 AWS SDKs AWS CLI 를 설치하고 구성합니다. 자세한 내용은 4단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.
2. 다음 예제 코드를 사용하여 프로젝트를 확인하십시오.

   CLI

   `list-projects` 명령어를 사용하여 계정에 있는 프로젝트를 나열합니다.

```
aws lookoutvision list-projects \
--profile lookoutvision-access
```

Python

이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져온 것입니다. 전체 예제는 여기에서 확인하세요.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0


from botocore.exceptions import ClientError
import boto3

class GettingStarted:

    @staticmethod
    def list_projects(lookoutvision_client):
        """
        Lists information about the projects that are in in your AWS account
        and in the current AWS Region.

        :param lookoutvision_client: A Boto3 Lookout for Vision client.
        """
        try:
            response = lookoutvision_client.list_projects()
            for project in response["Projects"]:
                print("Project: " + project["ProjectName"])
                print("ARN: " + project["ProjectArn"])
                print()
            print("Done!")
        except ClientError:
            raise
def main():
    session = boto3.Session(profile_name='lookoutvision-access')
    lookoutvision_client = session.client("lookoutvision")

    GettingStarted.list_projects(lookoutvision_client)
```

```
if __name__ == "__main__":
    main()
```

Java V2

이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져온 것입니다. 전체 예제는 여기에서 확인하세요.

```
/*
   Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
   SPDX-License-Identifier: Apache-2.0
*/

package com.example.lookoutvision;

import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.services.lookoutvision.LookoutVisionClient;
import software.amazon.awssdk.services.lookoutvision.model.ProjectMetadata;
import
 software.amazon.awssdk.services.lookoutvision.paginators.ListProjectsIterable;
import software.amazon.awssdk.services.lookoutvision.model.ListProjectsRequest;
import
 software.amazon.awssdk.services.lookoutvision.model.LookoutVisionException;

import java.util.ArrayList;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;

public class GettingStarted {

    public static final Logger logger =
 Logger.getLogger(GettingStarted.class.getName());

    /**
     * Lists the Amazon Lookoutfor Vision projects in the current AWS account
 and
     * AWS Region.
     *
     * @param lfvClient   An Amazon Lookout for Vision client.
     * @return List<ProjectMetadata> Metadata for each project.
     */
```

```
   public static List<ProjectMetadata> listProjects(LookoutVisionClient
lfvClient)
           throws LookoutVisionException {

       logger.log(Level.INFO, "Getting projects:");
       ListProjectsRequest listProjectsRequest = ListProjectsRequest.builder()
               .maxResults(100)
               .build();

       List<ProjectMetadata> projectMetadata = new ArrayList<>();

       ListProjectsIterable projects =
lfvClient.listProjectsPaginator(listProjectsRequest);

       projects.stream().flatMap(r -> r.projects().stream())
               .forEach(project -> {
                   projectMetadata.add(project);
                   logger.log(Level.INFO, project.projectName());
               });

       logger.log(Level.INFO, "Finished getting projects.");

       return projectMetadata;

   }

   public static void main(String[] args) throws Exception {

       try {

           // Get the Lookout for Vision client.
           LookoutVisionClient lfvClient = LookoutVisionClient.builder()

.credentialsProvider(ProfileCredentialsProvider.create("lookoutvision-access"))
                   .build();

           List<ProjectMetadata> projects = Projects.listProjects(lfvClient);

           System.out.printf("Projects%n--------%n");

           for (ProjectMetadata project : projects) {
               System.out.printf("Name: %s%n", project.projectName());
               System.out.printf("ARN: %s%n", project.projectArn());
```

```
                System.out.printf("Date: %s%n%n",
 project.creationTimestamp().toString());
            }

        } catch (LookoutVisionException lfvError) {
            logger.log(Level.SEVERE, "Could not list projects: {0}: {1}",
                    new Object[] { lfvError.awsErrorDetails().errorCode(),
                            lfvError.awsErrorDetails().errorMessage() });
            System.out.println(String.format("Could not list projects: %s",
 lfvError.getMessage()));
            System.exit(1);
        }

    }

}
```

5단계: (선택 사항) 고유한 AWS Key Management Service 키 사용

AWS KMS(키 관리 서비스)를 사용하여 Amazon S3 버킷에 저장하는 입력 이미지의 암호화를 관리할 수 있습니다.

기본적으로 이미지는 AWS가 소유하고 관리하는 키로 암호화됩니다. 자체 AWS Key Management Service (KMS) 키를 사용하도록 선택할 수도 있습니다. 자세한 내용은 AWS Key Management Service 개념을 참조하십시오.

자체 KMS 키를 사용하려면 다음 정책을 사용하여 KMS 키를 지정하십시오. *kms_key_arn*을 사용하려는 KMS 키 (또는 KMS 별칭 ARN)의 ARN으로 변경합니다. 또는 아무 KMS 키나 사용하도록 *을 지정할 수도 있습니다. IAM 정책에 대한 자세한 내용은 사용 설명서의 정책 생성을 참조하세요.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "LookoutVisionKmsDescribeAccess",
            "Effect": "Allow",
            "Action": "kms:DescribeKey",
            "Resource": "kms_key_arn"
        },
        {
            "Sid": "LookoutVisionKmsCreateGrantAccess",
```

```
            "Effect": "Allow",
            "Action": "kms:CreateGrant",
            "Resource": "kms_key_arn",
            "Condition": {
                "StringLike": {
                    "kms:ViaService": "lookoutvision.*.amazonaws.com"
                },
                "Bool": {
                    "kms:GrantIsForAWSResource": "true"
                }
            }
        }
    ]
}
```

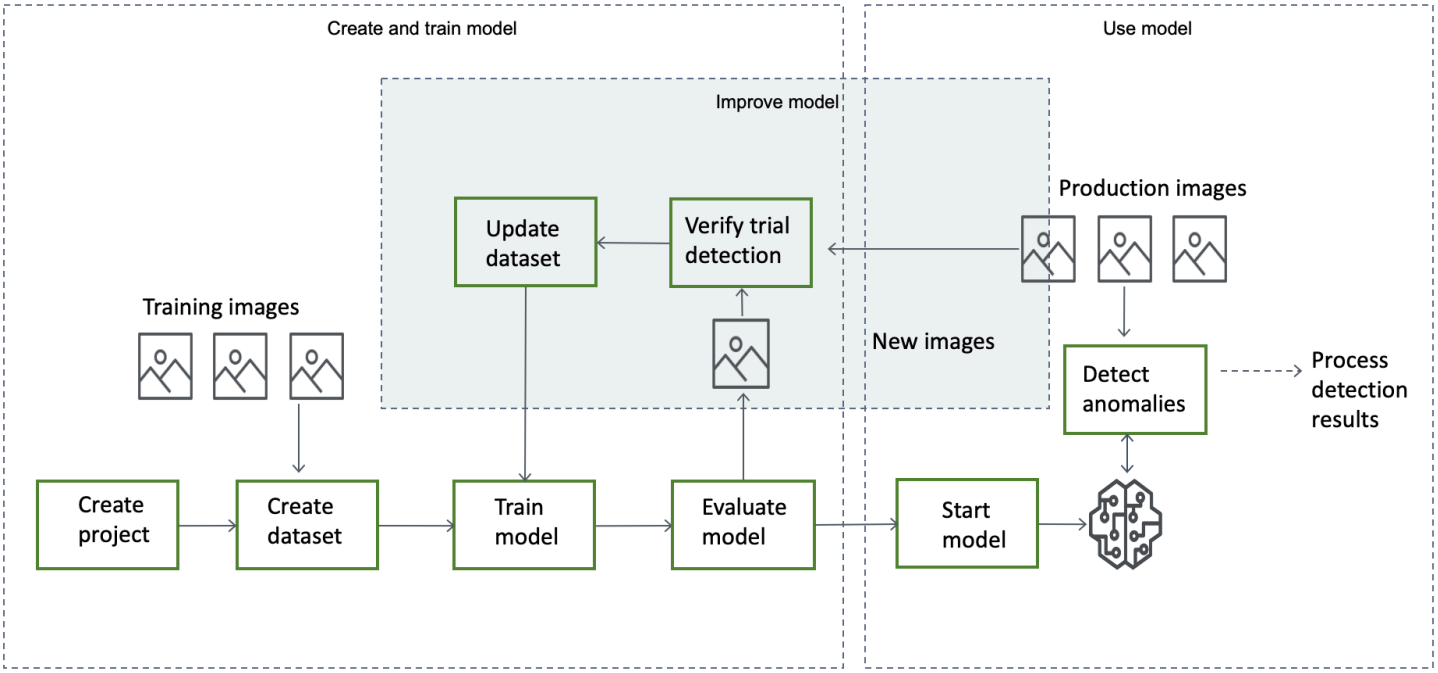
Amazon Lookout for Vision 이해

Amazon Lookout for Vision을 사용하면 다음과 같은 작업에서 산업용 제품의 시각적 결함을 대규모로 정확하게 찾아낼 수 있습니다.

- 손상된 부품 감지 — 제조 및 조립 과정에서 제품 표면 품질, 색상 및 모양에 대한 손상을 찾아냅니다.
- 누락된 구성 요소 식별 — 물체의 부재, 존재 또는 위치를 기준으로 누락된 구성 요소를 확인합니다. 인쇄 회로 기판의 커패시터 분실을 예로 들 수 있습니다.
- 공정 문제 파악 — 실리콘 웨이퍼의 동일한 지점에서 반복되는 스크래치와 같이 패턴이 반복되는 결함을 감지합니다.

Lookout for Vision을 사용하면 이미지에 이상이 있는지 예측하는 컴퓨터 비전 모델을 만들 수 있습니다. Amazon Lookout for Vision에서 모델을 학습시키고 테스트하는 데 사용하는 이미지를 제공합니다. Amazon Lookout for Vision은 학습된 모델을 평가하고 개선하는 데 사용할 수 있는 지표를 제공합니다. AWS 클라우드에서 훈련된 모델을 호스팅하거나 엣지 디바이스에 모델을 배포할 수 있습니다. 간단한 API 작업을 통해 모델이 예측한 결과를 반환합니다.

모델을 만들고, 평가하고, 사용하는 일반적인 워크플로는 다음과 같습니다.



주제

- 모델 유형 선택

- 모델 생성
- 모델 평가
- 모델 사용
- 엣지 디바이스에서 모델 사용
- 대시보드 사용

모델 유형 선택

모델을 생성하려면 먼저 원하는 모델 유형을 결정해야 합니다. 이미지 분류와 이미지 세분화이라는 두 가지 유형의 모델을 만들 수 있습니다. 사용 사례를 바탕으로 생성할 모델 유형을 결정합니다.

이미지 분류 모델

이미지에 예외 항목이 포함되어 있는지 여부만 알면 되고 위치는 알 필요가 없다면 이미지 분류 모델을 만들어 보세요. 이미지 분류 모델은 이미지에 이상이 있는지 여부를 예측합니다. 예측에는 예측 정확도에 대한 모델의 신뢰도가 포함됩니다. 모델은 이미지에서 발견된 이상 현상의 위치에 대한 정보를 제공하지 않습니다.

이미지 세분화 모델

스크래치 위치와 같은 이상 현상의 위치를 알아야 하는 경우 이미지 세분화 모델을 만드십시오. Amazon Lookout for Vision 모델은 의미 체계 분할을 사용하여 이미지에서 결함 유형 (예: 스크래치 또는 누락된 부분) 이 있는 픽셀을 식별합니다.

> **Note**
>
> 의미 체계 분할 모델은 다양한 유형의 예외 항목을 찾습니다. 개별 이상 현상에 대한 인스턴스 정보는 제공하지 않습니다. 예를 들어 이미지에 움푹 들어간 곳이 두 개 있는 경우 Lookout for Vision은 움푹 들어간 곳 이상 유형을 나타내는 단일 엔티티의 두 움푹 들어간 곳에 대한 정보를 반환합니다.

Amazon Lookout for Vision 분할 모델은 다음을 예측합니다.

분류

모델은 분석된 이미지에 대한 분류 (정상/변칙)를 반환하며, 여기에는 예측에 대한 모델의 신뢰도가 포함됩니다. 분류 정보는 분할 정보와 별도로 계산되므로 둘 사이의 관계가 있다고 가정해서는 안 됩니다.

분할

모델은 이미지에서 이상이 발생하는 픽셀을 표시하는 이미지 마스크를 반환합니다. 데이터셋의 예외 항목 레이블에 할당된 색상에 따라 다양한 유형의 예외 항목이 색상으로 구분됩니다. 예외 항목 레이블은 예외 항목의 유형을 나타냅니다. 예를 들어, 다음 이미지의 파란색 마스크는 자동차에서 발견된 스크래치 이상 유형의 위치를 표시합니다.

모델은 마스크에 있는 각 예외 항목 레이블의 색상 코드를 반환합니다. 또한 모델은 예외 항목 레이블의 이미지 커버링 비율도 반환합니다.

Lookout for Vision 분할 모델을 사용하면 다양한 기준을 사용하여 모델의 분석 결과를 분석할 수 있습니다. 예시:

- 예외 위치 — 변칙 위치를 알아야 하는 경우 분할 정보를 사용하여 변칙을 커버하는 마스크를 확인하십시오.
- 예외 유형 — 분할 정보를 사용하여 이미지에 허용되는 개수 이상의 예외 유형이 포함되어 있는지 확인할 수 있습니다.
- 적용 범위 — 분할 정보를 사용하여 예외 유형이 영상의 허용 가능한 영역 이상을 포함하는지 여부를 결정할 수 있습니다.
- 이미지 분류 — 이상 위치를 알 필요가 없는 경우 분류 정보를 사용하여 이미지에 이상이 있는지 확인하십시오.

예제 코드는 이미지에서 이상 탐지 항목을 참조하세요.

원하는 모델 유형을 결정한 후 모델을 관리할 프로젝트와 데이터 세트를 생성합니다. 레이블을 사용하여 이미지를 정상 또는 이상 이미지로 분류할 수 있습니다. 또한 레이블은 마스크 및 예외 유형과 같은 분할 정보를 식별합니다. 데이터 세트의 이미지에 레이블을 지정하는 방법에 따라 Lookout for Vision에서 생성하는 모델 유형이 결정됩니다.

이미지 분할 모델에 레이블을 지정하는 것은 이미지 분류 모델에 레이블을 지정하는 것보다 더 복잡합니다. 분할 모델을 훈련시키려면 훈련 이미지를 정상 또는 변칙 이미지로 분류해야 합니다. 또한 각 변칙 이미지에 대해 변칙 마스크와 변칙 유형을 정의해야 합니다. 분류 모델에서는 훈련 이미지를 정상 또는 변칙 이미지로 식별하기만 하면 됩니다.

모델 생성

모델을 만드는 단계는 프로젝트 생성, 데이터세트 생성, 모델 학습입니다.

프로젝트 생성

프로젝트를 생성하여 생성한 데이터 세트와 모델을 관리하세요. 프로젝트는 단일 사용 사례 (예: 단일 유형의 기계 부품에서 이상을 감지하는 경우)에만 사용해야 합니다.

대시보드를 사용하여 프로젝트 개요를 파악할 수 있습니다. 자세한 내용은 Amazon Lookout for Vision 대시보드 사용 단원을 참조하십시오.

추가 정보: 프로젝트 만들기.

데이터 세트 생성

모델을 학습시키려면 Amazon Lookout for Vision은 사용 사례에 맞는 정상 및 변칙 물체의 이미지가 필요합니다. 이러한 이미지를 데이터 세트로 제공합니다.

데이터세트는 해당 이미지를 설명하는 이미지와 레이블의 집합입니다. 이미지는 이상 현상이 발생할 수 있는 단일 유형의 객체를 나타내야 합니다. 자세한 내용은 데이터 세트용 이미지 준비 단원을 참조하십시오.

Amazon Lookout for Vision을 사용하면 단일 데이터 세트를 사용하는 프로젝트 또는 별도의 교육 및 테스트 데이터 세트가 있는 프로젝트를 만들 수 있습니다. 교육, 테스트 및 성능 튜닝을 세부적으로 제어하려는 경우가 아니라면 단일 데이터 세트 프로젝트를 사용하는 것이 좋습니다.

이미지를 가져와서 데이터 세트를 생성합니다. 이미지를 가져오는 방법에 따라 이미지에 레이블이 지정될 수도 있습니다. 그렇지 않은 경우 콘솔을 사용하여 이미지에 레이블을 지정합니다.

이미지 가져오기

Lookout for Vision 콘솔에서 데이터세트를 생성하면, 다음 중 한 가지 방법으로 이미지를 가져올 수 있습니다.

- 로컬 컴퓨터에서 이미지를 가져옵니다. 이미지에는 레이블이 지정되지 않습니다.
- S3 버킷에서 이미지를 가져옵니다. Amazon Lookout for Vision은 이미지가 포함된 폴더 이름을 사용하여 이미지를 분류할 수 있습니다. 일반 이미지에 `normal`를 사용합니다. 변칙 이미지에 `anomaly`를 사용합니다. 분할 레이블은 자동으로 할당할 수 없습니다.
- Amazon SageMaker AI Ground Truth 매니페스트 파일을 가져옵니다. 매니페스트 파일의 이미지에는 레이블이 지정됩니다. 매니페스트 파일을 직접 만들고 가져올 수 있습니다. 이미지가 많은 경우 SageMaker AI Ground Truth 레이블 지정 서비스를 사용하는 것이 좋습니다. 그런 다음 Amazon SageMaker AI Ground Truth 작업에서 출력 매니페스트 파일을 가져옵니다.

이미지 레이블 지정

레이블은 데이터세트의 이미지를 설명합니다. 레이블은 이미지가 정상인지 비정상(분류)인지를 지정합니다. 또한 레이블은 이미지 상의 예외 위치 (분할)를 설명합니다.

이미지에 레이블이 지정되지 않은 경우 콘솔을 사용하여 이미지에 레이블을 지정할 수 있습니다.

데이터 세트의 이미지에 할당하는 레이블에 따라 Lookout for Vision이 생성하는 모델 유형이 결정됩니다.

이미지 분류

이미지 분류 모델을 만들려면 Lookout for Vision 콘솔을 사용하여 데이터 세트의 이미지를 정상 또는 이상 이미지로 분류합니다.

`CreateDataset` 작업을 사용하여 분류 정보가 포함된 매니페스트 파일에서 데이터세트를 만들 수도 있습니다.

이미지 분할

이미지 분할 모델을 만들려면 Lookout for Vision 콘솔을 사용하여 데이터 세트의 이미지를 정상 또는 이상 이미지로 분류합니다. 또한 이미지의 이상 영역 (있는 경우)에는 픽셀 마스크를 지정하고 개별 예외 마스크에는 예외 항목 레이블을 지정할 수 있습니다.

`CreateDataset`작업을 사용하여 세분화 및 분류 정보가 포함된 매니페스트 파일에서 데이터셋을 만들 수도 있습니다.

프로젝트에 별도의 훈련 및 테스트 데이터 세트가 있는 경우 Lookout for Vision은 훈련 데이터 세트를 사용하여 모델 유형을 학습하고 결정합니다. 테스트 데이터 세트의 이미지에도 같은 방식으로 레이블을 지정해야 합니다.

추가 정보: 데이터세트 만들기.

모델 학습

학습을 통해 모델이 생성되고 이미지에 이상이 있는지 예측하도록 모델을 학습시킵니다. 학습할 때마다 모델의 새 버전이 생성됩니다.

Amazon Lookout for Vision은 학습을 시작할 때 모델 학습에 가장 적합한 알고리즘을 선택합니다. 모델을 학습시킨 다음 테스트합니다. Amazon Lookout for Vision을 시작하기에서 단일 데이터세트 프로젝트를 학습시키는 경우 데이터세트를 내부적으로 분할하여 학 데이터세트와 테스트 데이터세트를 만듭니다. 학습 데이터 세트와 테스트 데이터 세트가 분리된 프로젝트를 만들 수도 있습니다. 이 구성에서 Amazon Lookout for Vision은 학습 데이터 세트를 사용하여 모델을 학습시키고 테스트 데이터 세트로 모델을 테스트합니다.

> **Important**
>
> 모델을 성공적으로 학습하는 시간을 기준으로 요금이 부과됩니다.

추가 정보: 모델 학습하기.

모델 평가

테스트 중에 만든 성능 지표를 사용하여 모델의 성능을 평가하십시오.

성능 지표를 사용하면 학습된 모델의 성능을 더 잘 이해하고 프로덕션 환경에서 사용할 준비가 되었는지 결정할 수 있습니다.

추가 정보: 모델 개선.

성능 지표에서 개선이 필요한 것으로 나타나면 새 이미지로 시험 탐지 작업을 실행하여 훈련 데이터를 더 추가할 수 있습니다. 작업이 완료되면 결과를 확인하고 검증된 이미지를 훈련 데이터 세트에 추가할 수 있습니다. 또는 새 훈련 이미지를 데이터 세트에 직접 추가할 수도 있습니다. 다음으로 모델을 재학습시키고 성능 지표를 다시 확인합니다.

추가 정보: 시험 탐지 작업을 통한 모델 검증.

모델 사용

AWS 클라우드에서 모델을 사용하기 전에 StartModel 작업으로 모델을 시작합니다. 콘솔에서 모델의 `StartModel` CLI 명령을 가져올 수 있습니다.

추가 정보: 모델 시작.

훈련된 Amazon Lookout for Vision 모델은 입력 이미지에 정상 콘텐츠 또는 비정상 콘텐츠가 포함되어 있는지 예측합니다. 모델이 분할 모델인 경우 예측에는 예외가 발견된 픽셀을 표시하는 예외 항목 마스크가 포함됩니다.

모델을 사용하여 예측하려면 DetectAnomalies 작업을 호출하고 로컬 컴퓨터의 입력 이미지를 전달하십시오. 콘솔에서 `DetectAnomalies`을 호출하는 CLI 명령을 가져올 수 있습니다.

추가 정보: 이미지에서 이상을 감지하십시오.

> **Important**
>
> 모델이 실행되는 시간만큼 요금이 부과됩니다.

모델을 더 이상 사용하지 않는 경우 StopModel 작업을 사용하여 모델을 중지하십시오. CLI 명령은 콘솔에서 가져올 수 있습니다.

추가 정보: 모델 중지.

엣지 디바이스에서 모델 사용

AWS IoT Greengrass Version 2 코어 디바이스에서 Lookout for Vision 모델을 사용할 수 있습니다.

추가 정보: 엣지 디바이스에서 Amazon Lookout for Vision 모델 사용하기.

대시보드 사용

대시보드를 사용하여 모든 프로젝트의 개요와 개별 프로젝트에 대한 개요 정보를 얻을 수 있습니다.

추가 정보: 대시보드 사용.

Amazon Lookout for Vision을 시작하기

시작하기 지침 전에 Amazon Lookout for Vision 이해 항목을 먼저 읽어 보는 것이 좋습니다.

시작하기 지침에서는 예제 이미지 분할 모델을 만드는 방법을 보여줍니다. 예제 이미지 분류 모델을 만들려면 이미지 분류 데이터 세트을 참조하십시오.

예제 모델을 빠르게 사용해 보려는 경우 예제 훈련 이미지와 마스크 이미지를 제공합니다. 또한 이미지 분할 매니페스트 파일을 생성하는 Python 스크립트도 제공합니다. 매니페스트 파일을 사용하여 프로젝트용 데이터 세트를 만들면 데이터 세트의 이미지에 레이블을 지정할 필요가 없습니다. 자체 이미지로 모델을 만들 때는 데이터 세트의 이미지에 레이블을 지정해야 합니다. 자세한 내용은 데이터 세트를 생성합니다. 단원을 참조하십시오.

제공하는 이미지는 정상 쿠키와 비정상 쿠키입니다. 비정상 쿠키는 쿠키 모양 전체에 균열이 있습니다. 이미지를 사용하여 학습시킨 모델은 다음 예제와 같이 분류 (정상 또는 비정상)를 예측하고 비정상 쿠키의 균열 영역 (마스크)을 찾습니다.

주제

1단계: 매니페스트 파일 생성 및 이미지 업로드

이 절차에서는 Amazon Lookout for Vision 설명서 리포지토리를 컴퓨터에 복제합니다. 그런 다음 Python (버전 3.7 이상) 스크립트를 사용하여 매니페스트 파일을 만들고 학습 이미지와 마스크 이미지를 지정한 Amazon S3 위치에 업로드합니다. 매니페스트 파일을 사용하여 모델을 생성합니다. 나중에 로컬 저장소의 테스트 이미지를 사용하여 모델을 시험해 볼 수 있습니다.

매니페스트 파일을 생성하고 이미지를 업로드하려면

1. Amazon Lookout for Vision 설정의 지침에 따라 Amazon Lookout for Vision을 설정하십시오. AWS Python용 SDK를 설치해야 합니다.
2. Lookout for Vision을 사용하려는 AWS 리전에서 S3 버킷을 생성합니다.
3. Amazon S3 버킷에서 이름이 `getting-started`인 폴더를 생성합니다.
4. Amazon S3 URI와 Amazon 리소스 이름 (ARN)을 폴더에 기록해 둡니다. 이를 사용하여 권한을 설정하고 스크립트를 실행할 수 있습니다.
5. 스크립트를 호출하는 사용자에게 `s3:PutObject` 작업을 호출할 수 있는 권한이 있는지 확인하십시오. 다음 정책을 사용할 수 있습니다. 권한을 할당하려면 권한 할당 항목을 참조하세요.

   ```
   {
    "Version": "2012-10-17",
    "Statement": [{
     "Sid": "Statement1",
     "Effect": "Allow",
     "Action": [
      "s3:PutObject"
     ],
     "Resource": [
      "arn:aws:s3::: ARN for S3 folder in step 4/*"
     ]
    }]
   }
   ```

6. 이름이 `lookoutvision-access`로 지정된 로컬 프로필이 있고 프로필 사용자에게 이전 단계의 권한이 있는지 확인하십시오. 자세한 내용은 로컬 컴퓨터에서 프로필 사용 단원을 참조하십시오.
7. zip 파일인 getting-started.zip를 다운로드하십시오. zip 파일에는 시작 데이터세트와 설정 스크립트가 포함되어 있습니다.
8. `getting-started.zip` 파일의 압축을 풉니다.
9. 명령 프롬프트에서 다음을 실행합니다.

a. `getting-started` 폴더로 이동합니다.

b. 다음 명령을 실행하여 매니페스트 파일을 생성하고 학습 이미지와 이미지 마스크를 4단계에서 기록한 Amazon S3 경로에 업로드합니다.

```
python getting_started.py S3-URI-from-step-4
```

c. 스크립트가 완료되면 스크립트가 `Create dataset using manifest file:` 다음에 표시하는 `train.manifest` 파일의 경로를 기록해 둡니다. 경로는 `s3://path to getting started folder/manifests/train.manifest`과 유사해야 합니다.

2단계:모델 생성

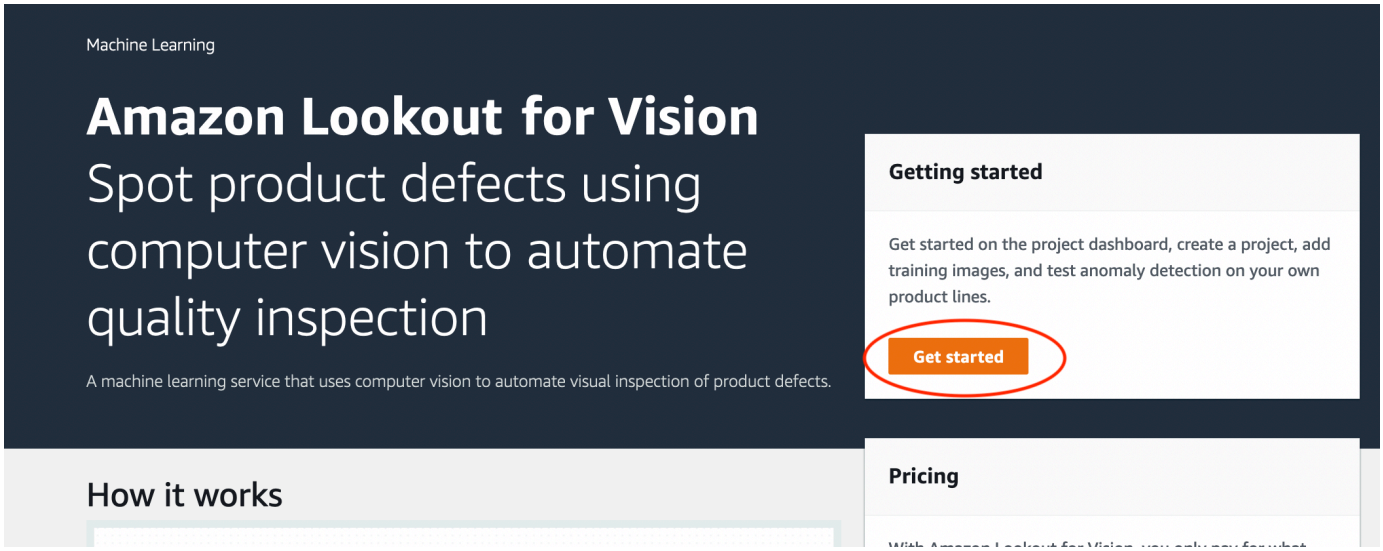
이 절차에서는 이전에 Amazon S3 버킷에 업로드한 이미지와 매니페스트 파일을 사용하여 프로젝트와 데이터 세트를 생성합니다. 그런 다음 모델을 생성하고 모델 학습의 평가 결과를 확인합니다.

시작하기 매니페스트 파일에서 데이터 세트를 생성하므로 데이터 세트 이미지에 레이블을 지정할 필요가 없습니다. 자체 이미지로 데이터 세트를 만들 때는 이미지에 레이블을 지정해야 합니다. 자세한 내용은 이미지 레이블 지정 단원을 참조하십시오.

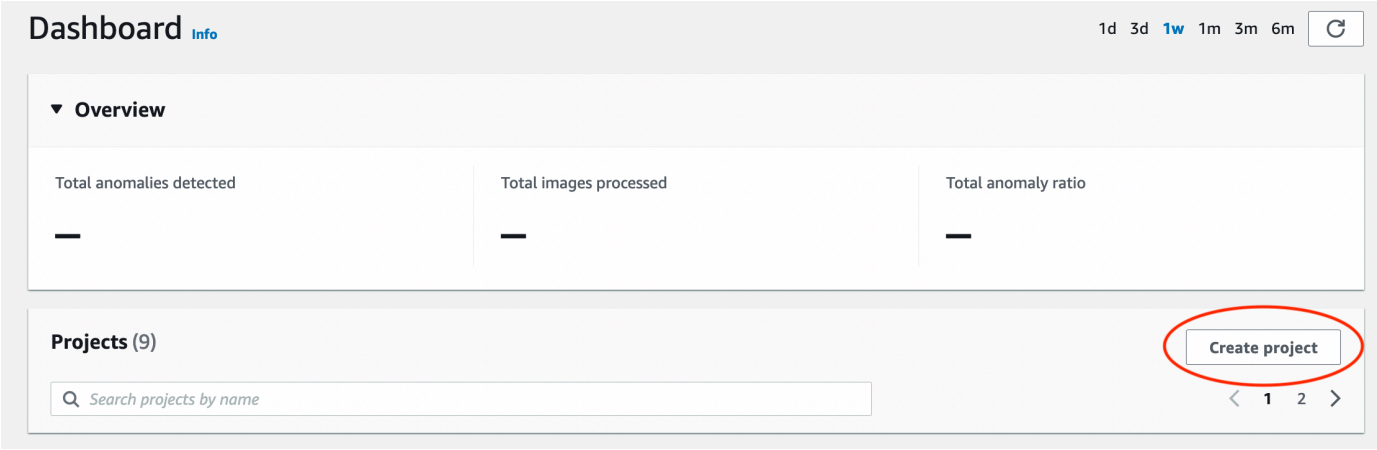
> **Important**
>
> 모델을 성공적으로 학습시키는 데에만 비용이 부과됩니다.

모델을 생성하는 방법

1. https://console.aws.amazon.com/lookoutvision/에서 Amazon Lookout for Vision을 엽니다.
2. 에서 Amazon S3 버킷을 생성한 리전과 동일한 AWS 리전에 있어야 합니다1단계: 매니페스트 파일 생성 및 이미지 업로드. 지역을 변경하려면 탐색바에서 현재 표시된 지역의 이름을 선택합니다. 그런 다음 전환하려는 지역을 선택합니다.
3. Get started를 선택합니다.



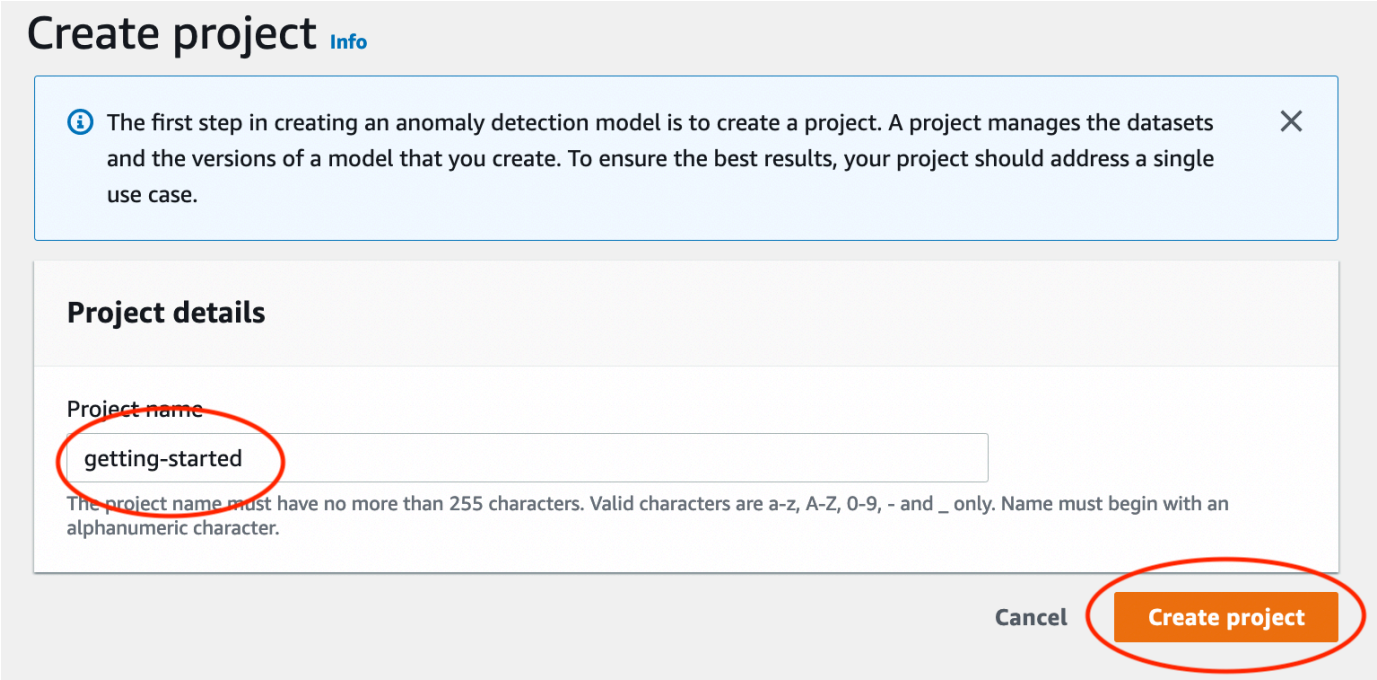
4. 프로젝트 섹션에서 프로젝트 만들기를 선택합니다.



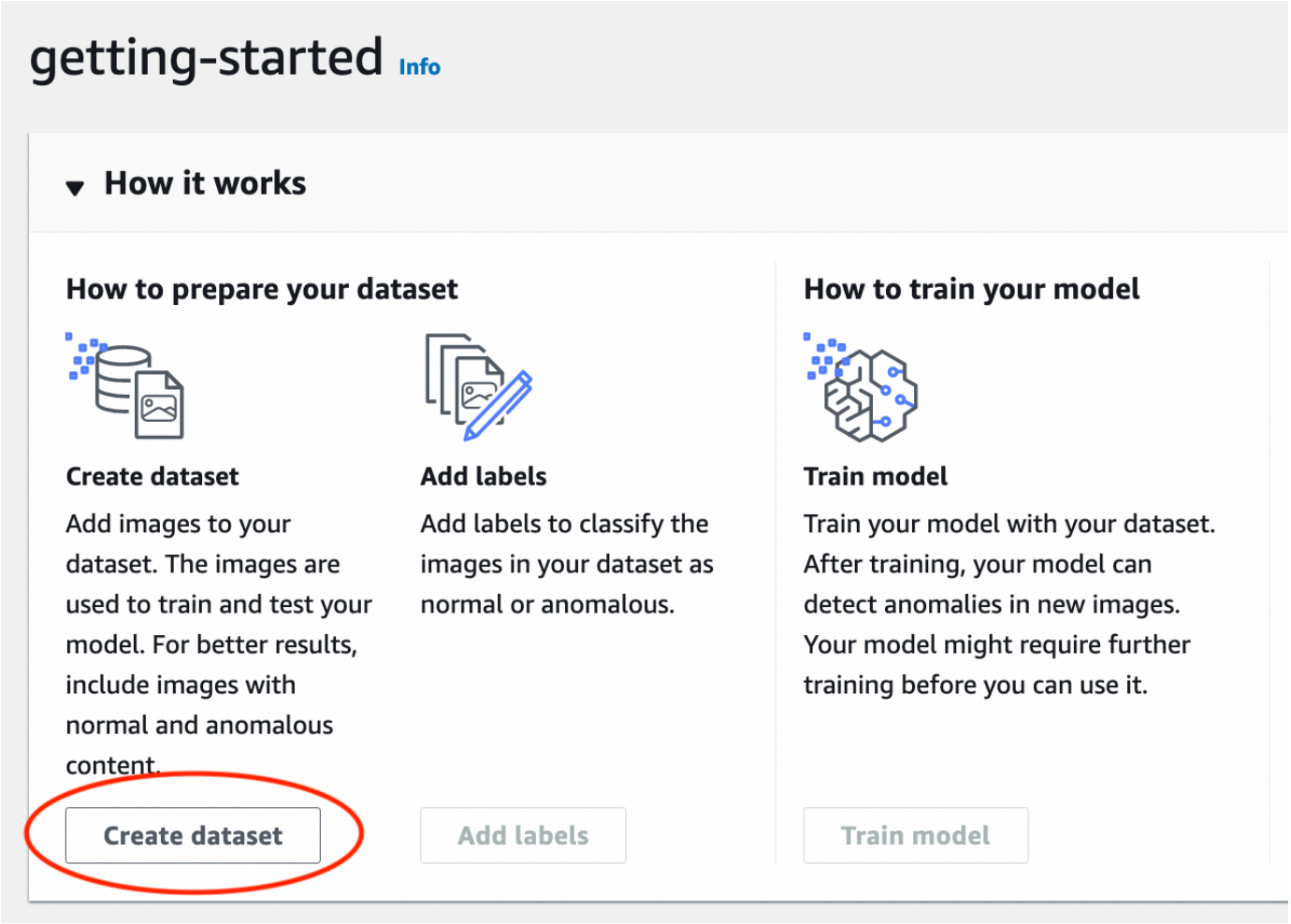
5. 프로젝트 생성 페이지에서 다음을 수행합니다.

   a. 프로젝트 이름에 `getting-started`을 입력합니다.

   b. 프로젝트 생성을 선택합니다.



6. 프로젝트 페이지의 작동 방식 섹션에서 데이터 세트 생성을 선택합니다.



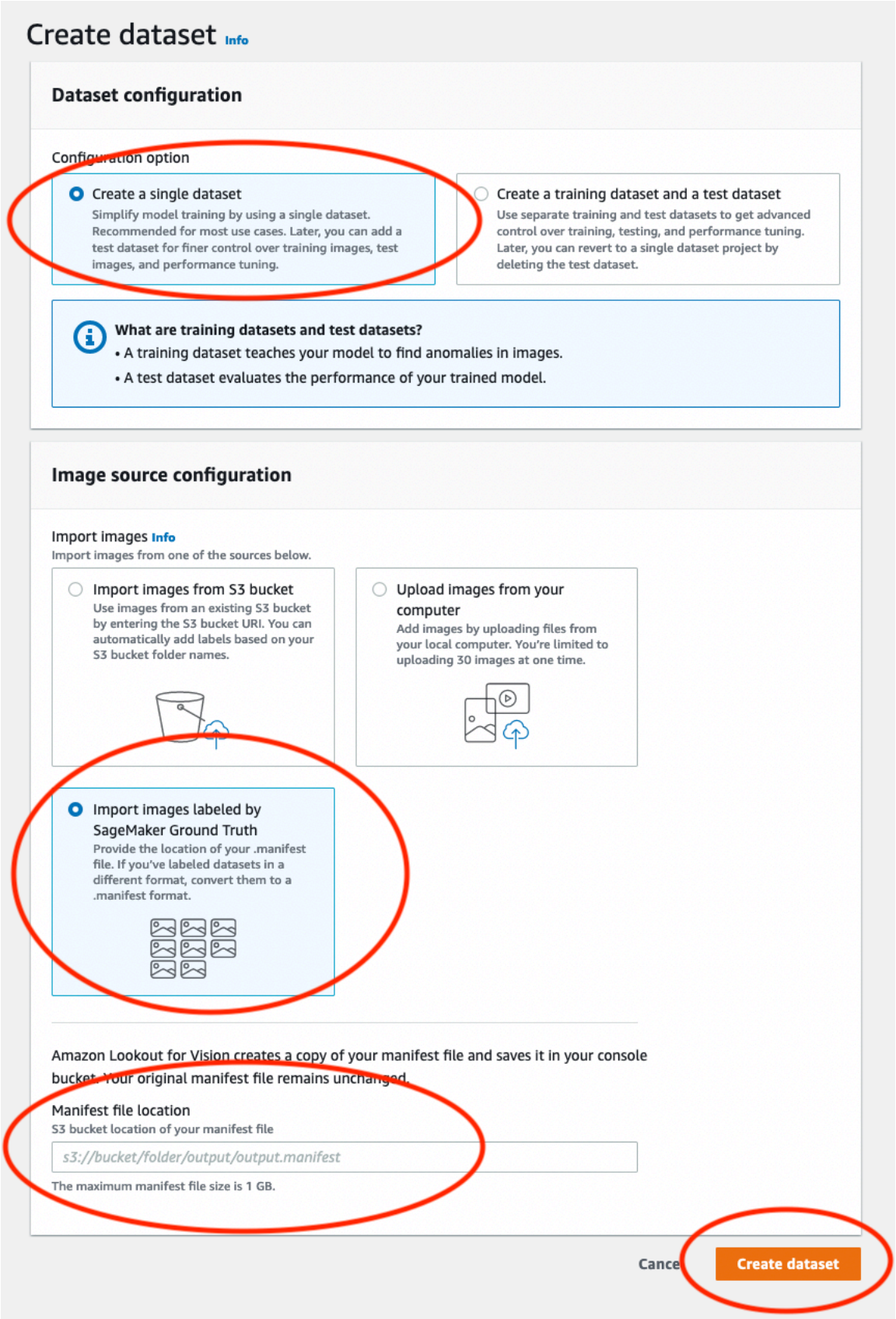
7. 데이터베이스 생성 페이지에서 다음을 수행합니다.

   a. 단일 데이터 세트 만들기를 선택합니다.

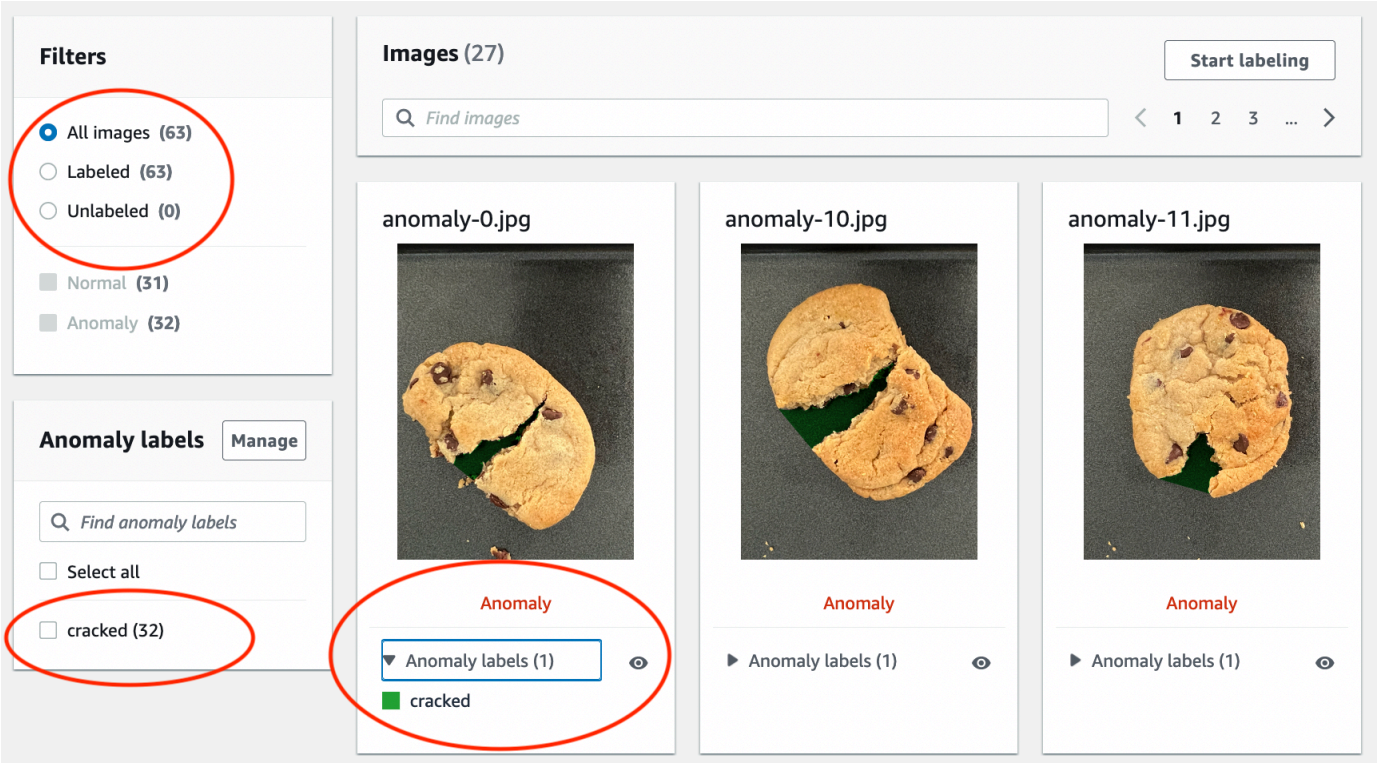
   b. 이미지 원본 구성 섹션에서 SageMaker Ground Truth로 레이블이 지정된 이미지 가져오기를 선택합니다.

   c. .manifest 파일 위치에는 1단계: 매니페스트 파일 생성 및 이미지 업로드의 6.c단계에서 기록해 둔 매니페스트 파일의 Amazon S3 위치를 입력합니다. Amazon S3 위치는 `s3://path to getting started folder/manifests/train.manifest`과 비슷해야 합니다.

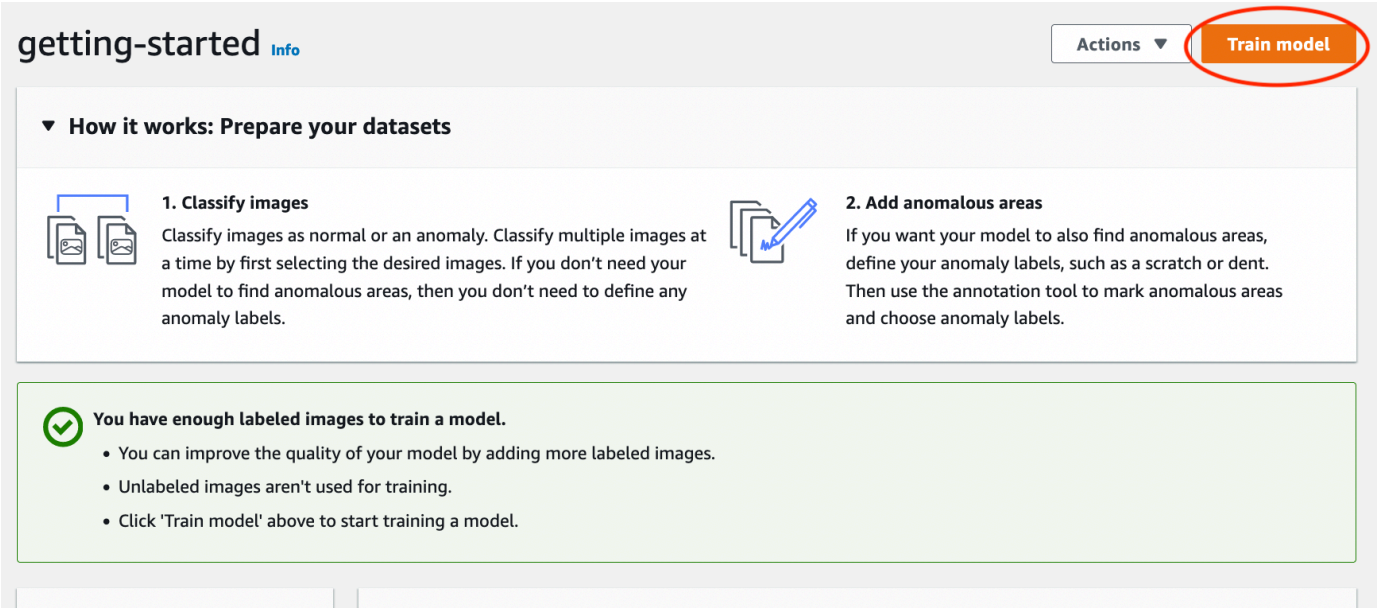
   d. 데이터 세트 생성을 선택합니다.


Create dataset Info
Dataset configuration
Configuration option
Create a single dataset
Simplify model training by using a single dataset. Recommended for most use cases. Later, you can add a test dataset for finer control over training images, test images, and performance tuning.
Create a training dataset and a test dataset
Use separate training and test datasets to get advanced control over training, testing, and performance tuning. Later, you can revert to a single dataset project by deleting the test dataset.
What are training datasets and test datasets?
• A training dataset teaches your model to find anomalies in images.
• A test dataset evaluates the performance of your trained model.
Image source configuration
Import images Info
Import images from one of the sources below.
Import images from S3 bucket
Use images from an existing S3 bucket by entering the S3 bucket URI. You can automatically add labels based on your S3 bucket folder names.
Upload images from your computer
Add images by uploading files from your local computer. You're limited to uploading 30 images at one time.
Import images labeled by SageMaker Ground Truth
Provide the location of your .manifest file. If you've labeled datasets in a different format, convert them to a .manifest format.
Amazon Lookout for Vision creates a copy of your manifest file and saves it in your console bucket. Your original manifest file remains unchanged.
Manifest file location
S3 bucket location of your manifest file
s3://bucket/folder/output/output.manifest
The maximum manifest file size is 1 GB.
Cancel
Create dataset

8. 프로젝트 세부 정보 페이지의 이미지 섹션에서 데이터 세트 이미지를 확인합니다. 각 데이터세트 이미지에 대한 분류 및 이미지 분할 정보 (마스크 및 예외 항목 레이블)를 볼 수 있습니다. 이미지를 검색하거나, 레이블 지정 상태 (레이블 지정/레이블 없음) 별로 이미지를 필터링하거나, 이미지에 할당된 예외 항목 레이블을 기준으로 이미지를 필터링할 수도 있습니다.

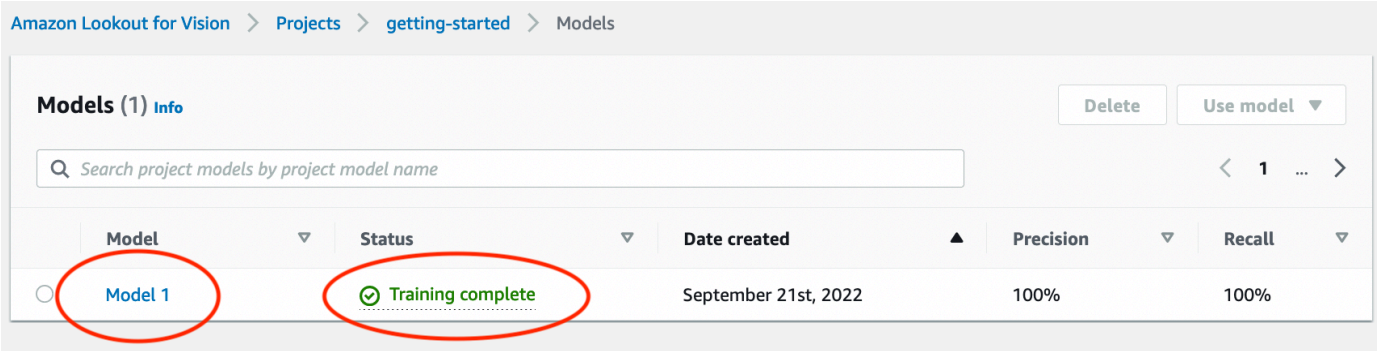


9. 프로젝트 세부 정보 페이지에서 모델 학습을 선택합니다.



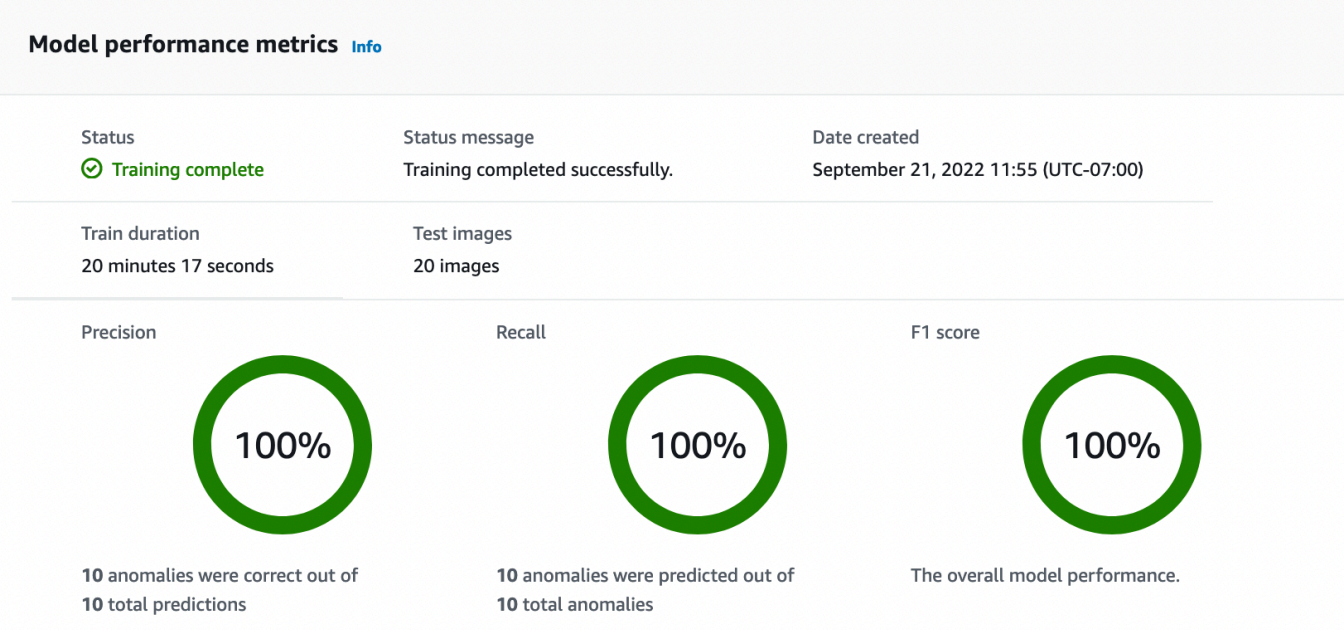
10. 모델 훈련 페이지에서 모델 훈련을 선택합니다.

11. 모델을 훈련하고 싶으신가요? 대화 상자에서 모델 훈련을 선택합니다.
12. 프로젝트 모델 페이지에서 학습이 시작된 것을 확인할 수 있습니다. 모델 버전의 상태 열을 확인하여 현재 상태를 확인하십시오. 모델 학습을 완료하는 데 최소 30분 걸립니다. 상태가 학습 완료로 변경되면 학습이 성공적으로 완료된 것입니다.
13. 훈련이 끝나면 모델 페이지에서 모델 1을 선택합니다.

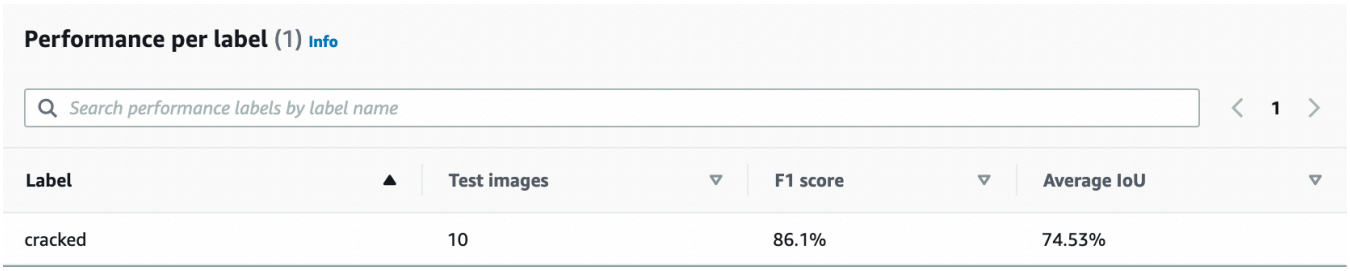


14. 모델의 세부 정보 페이지에서 성능 지표 탭의 평가 결과를 확인하십시오. 다음에 사용할 수 있는 지표는 다음과 같습니다.

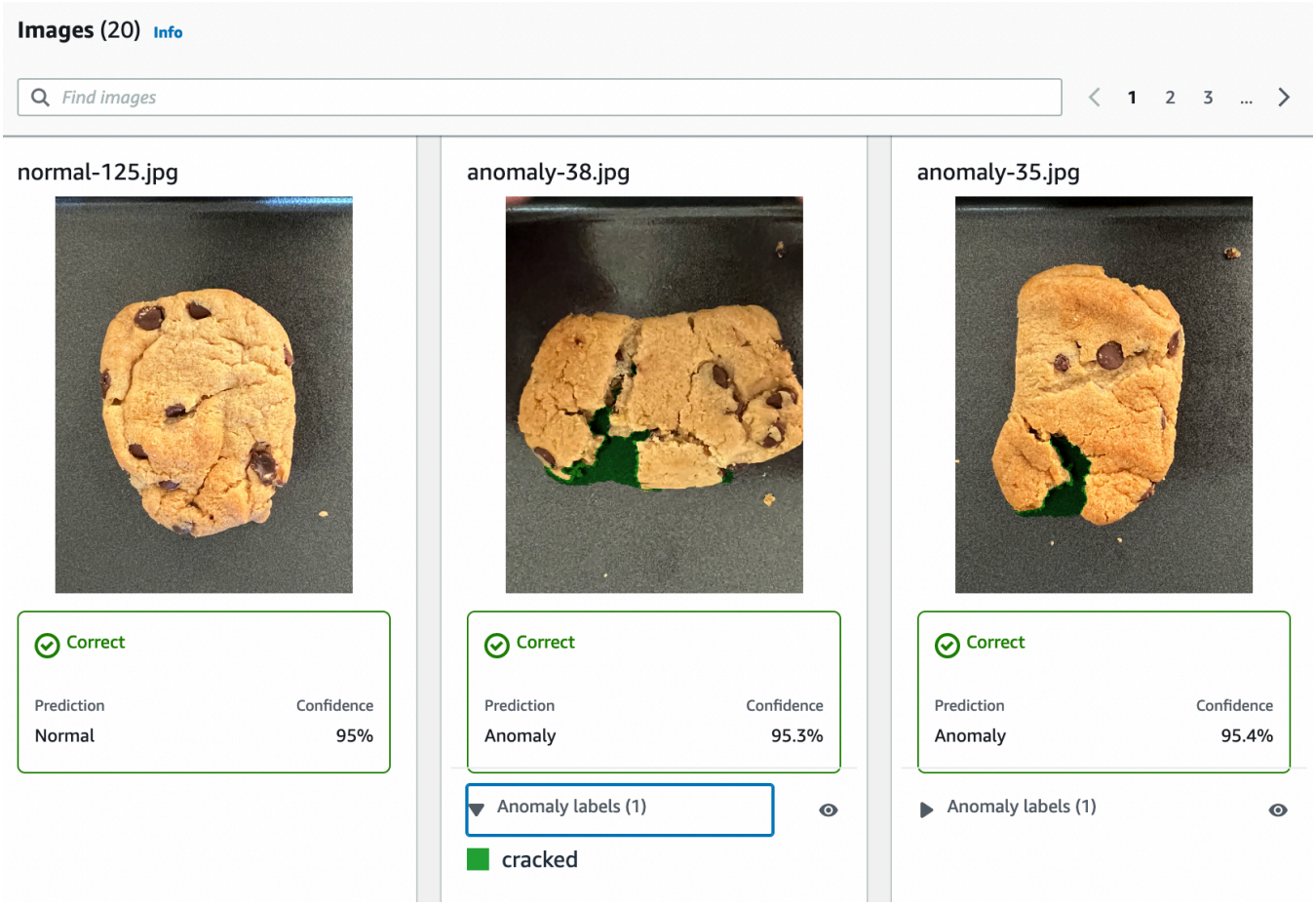
    - 모델에서 수행한 분류 예측에 대한 전체 모델 성능 지표 (정밀도, 재현율, F1 점수)



    - 테스트 이미지에서 발견된 예외 항목 레이블에 대한 성능 지표 (평균 IOU, F1 점수)



- 테스트 이미지에 대한 예측 (분류, 분할 마스크, 예외 항목 레이블)



모델 학습은 비결정적이므로 평가 결과가 이 페이지에 표시된 결과와 다를 수 있습니다. 자세한 내용은 Amazon Lookout for Vision 모델 개선 단원을 참조하십시오.

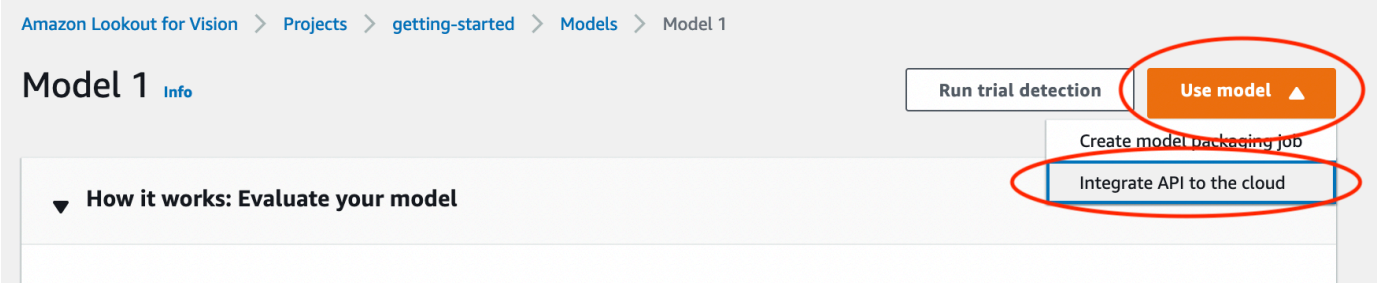
3단계: 모델 시작

이 단계에서는 이미지를 분석할 준비가 되도록 모델 호스팅을 시작합니다. 자세한 내용은 훈련된 Amazon Lookout for Vision 모델 실행 단원을 참조하십시오.

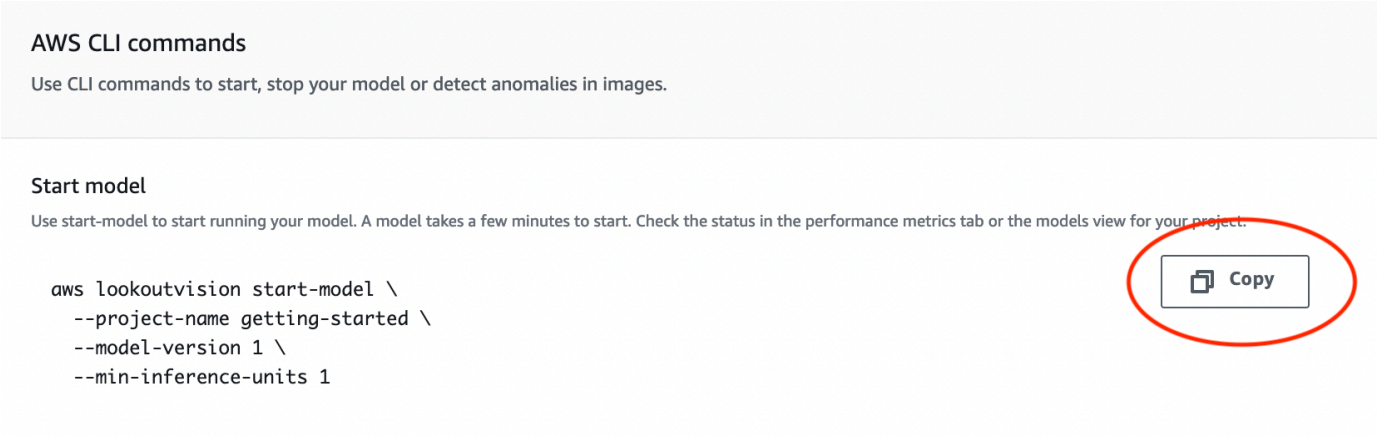
> **Note**
>
> 모델을 실행하는 시간에 따라 요금이 청구됩니다. 5단계: 모델 중지에서 모델을 정지시키면 됩니다.

모델을 시작하려면

1. 모델의 세부 정보 페이지에서 모델 사용을 선택한 다음 클라우드에 API 통합을 선택합니다.



2. AWS CLI 명령 섹션에서 `start-model` AWS CLI 명령을 복사합니다.



3. AWS CLI 가 Amazon Lookout for Vision 콘솔을 사용하는 리전과 동일한 AWS 리전에서 실행되도록 구성되어 있는지 확인합니다. 에서 AWS CLI 사용하는 AWS 리전을 변경하려면 섹션을 참조하세요AWS SDKS 설치.
4. 명령 프롬프트에서 `start-model` 명령을 입력하여 모델을 시작합니다. `lookoutvision` 프로필을 사용하여 자격 증명을 가져오려면 `--profile lookoutvision-access` 매개 변수를 추가하십시오. 예시:

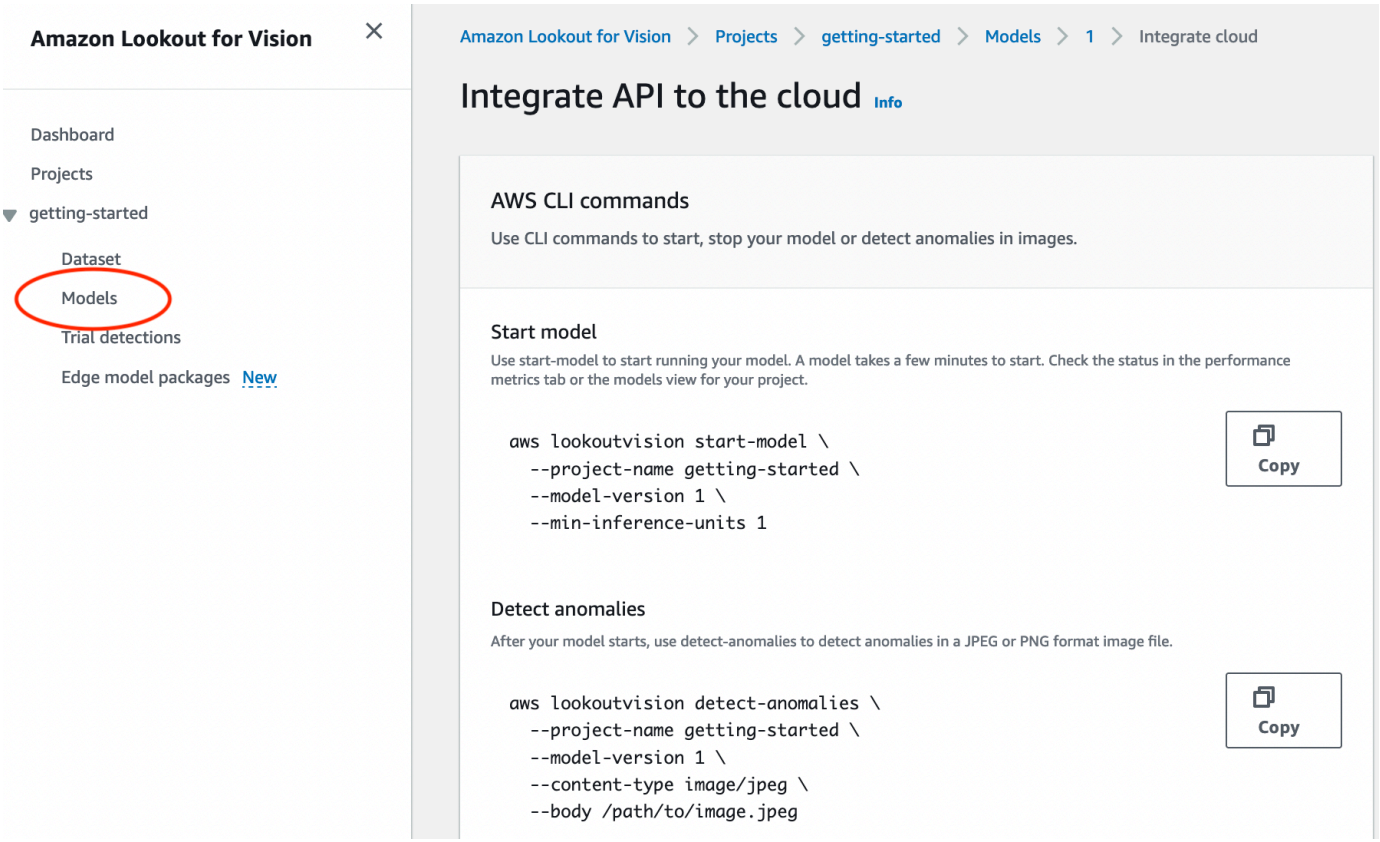
```
aws lookoutvision start-model \
  --project-name getting-started \
  --model-version 1 \
  --min-inference-units 1 \
```

```
  --profile lookoutvision-access
```

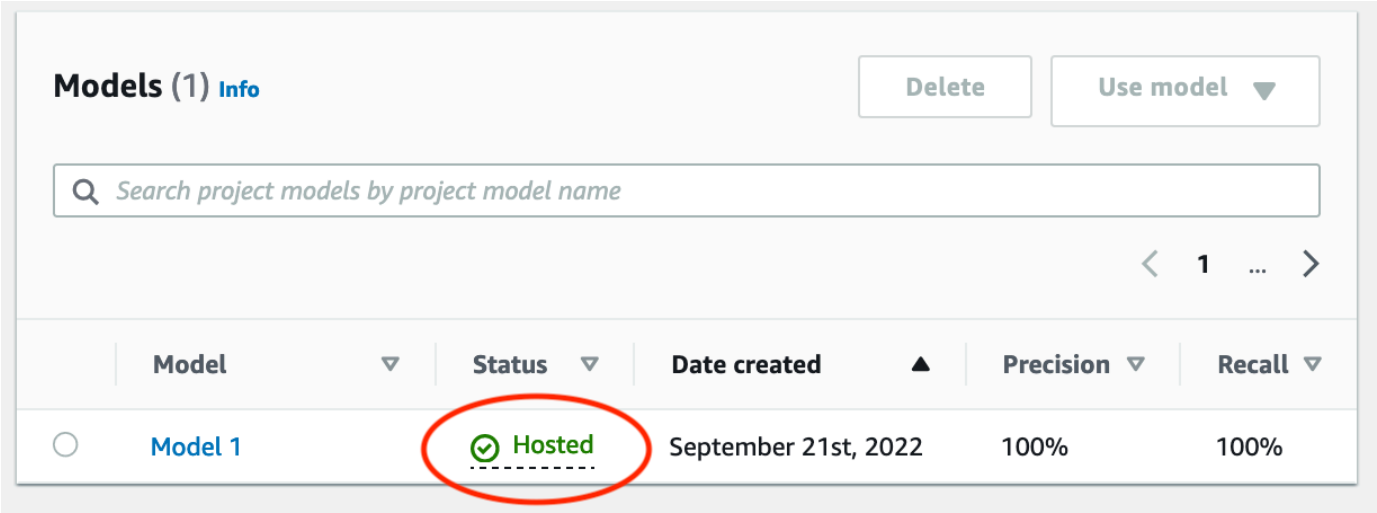
직접 호출이 성공하면 다음 출력이 표시됩니다.

```
{
    "Status": "STARTING_HOSTING"
}
```

5. 콘솔로 돌아가 탐색 창에서 모델을 선택합니다.



6. 상태 열의 모델 (모델 1) 상태가 호스팅됨으로 표시될 때까지 기다리십시오. 프로젝트에서 이전에 모델을 학습시킨 적이 있다면 최신 모델 버전이 완료될 때까지 기다리세요.

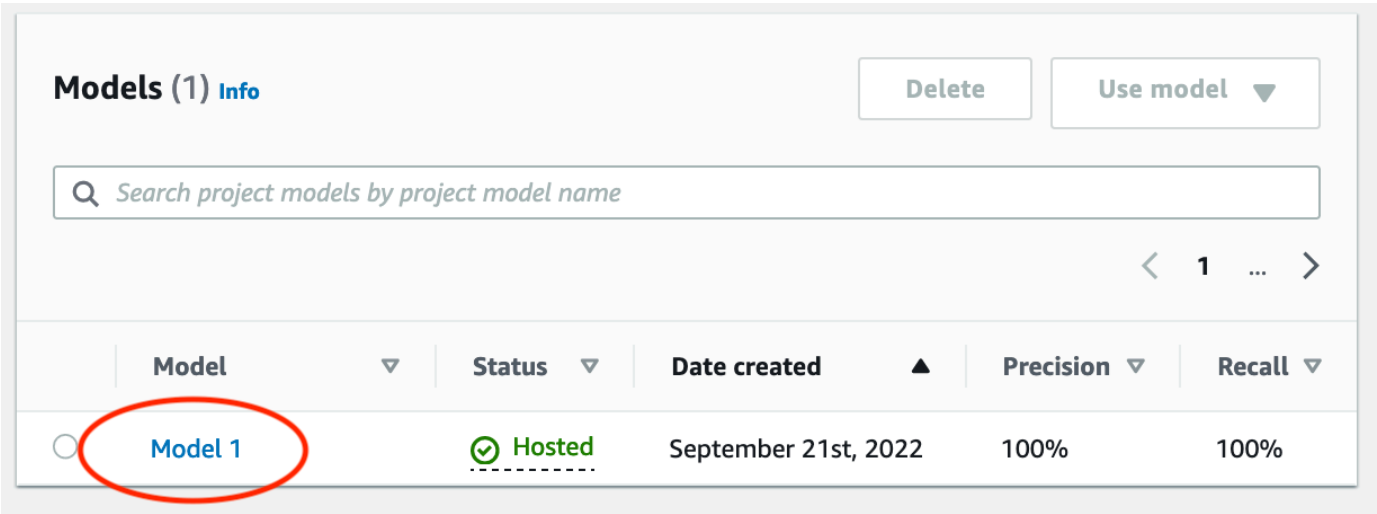


4단계: 이미지 분석

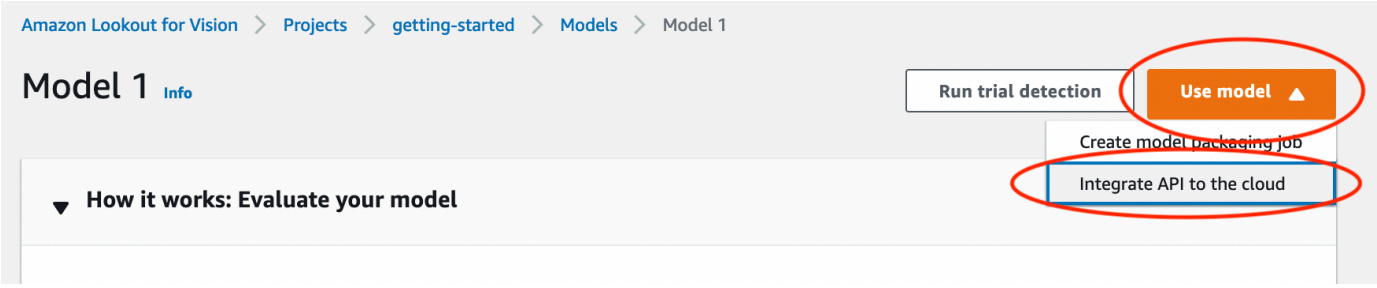
이 단계에서는 모델을 사용하여 이미지를 분석합니다. 컴퓨터의 Lookout for Vision 설명서 리포지토리에 있는 시작하기 `test-images` 폴더에서 사용할 수 있는 예제 이미지를 제공합니다. 자세한 내용은 이미지에서 이상 탐지 단원을 참조하십시오.

이미지를 분석하려면

1. 모델 페이지에서 모델 1을 선택합니다.



2. 모델의 세부 정보 페이지에서 모델 사용을 선택한 다음 클라우드에 API 통합을 선택합니다.



3. AWS CLI 명령 섹션에서 `detect-anomalies` AWS CLI 명령을 복사합니다.

4. 명령 프롬프트에서 이전 단계의 `detect-anomalies` 명령을 입력하여 비정상 이미지를 분석합니다. 컴퓨터의 시작하기 `test-images` 폴더에서 변칙 이미지를 `--body` 매개 변수에 지정하십시오. `lookoutvision` 프로필을 사용하여 자격 증명을 가져오려면 `--profile lookoutvision-access` 매개 변수를 추가하십시오. 예시:

```
aws lookoutvision detect-anomalies \
  --project-name getting-started \
  --model-version 1 \
  --content-type image/jpeg \
  --body /path/to/test-images/test-anomaly-1.jpg \
  --profile lookoutvision-access
```

다음과 같이 출력됩니다

```
{
    "DetectAnomalyResult": {
        "Source": {
            "Type": "direct"
        },
        "IsAnomalous": true,
        "Confidence": 0.983975887298584,
        "Anomalies": [
            {
                "Name": "background",
```

```
                "PixelAnomaly": {
                    "TotalPercentageArea": 0.9818974137306213,
                    "Color": "#FFFFFF"
                }
            },
            {
                "Name": "cracked",
                "PixelAnomaly": {
                    "TotalPercentageArea": 0.018102575093507767,
                    "Color": "#23A436"
                }
            }
        ],
        "AnomalyMask": "iVBORw0KGgoAAAANSUhEUgAAAkAAAAMACA......"
    }
}
```

5. 다음은 이 출력에 대한 유의 사항입니다.

   - `IsAnomalous`은 예측된 분류에 대한 부울입니다. 이미지가 비정상이면 `true`를 실행하고, 그렇지 않으면 `false`을 실행하십시오.
   - `Confidence`은 Amazon Lookout for Vision이 예측에 대해 갖는 신뢰도를 나타내는 부동 소수점 값입니다. 0은 가장 낮은 신뢰도이고 1은 가장 높은 신뢰도입니다.
   - `Anomalies`은 이미지에서 발견된 이상 현상의 목록입니다. `Name`는 예외 항목 레이블입니다. `PixelAnomaly`은 예외 항목의 총 백분율 면적 (`TotalPercentageArea`) 과 예외 항목 레이블의 색상 (`Color`)을 포함합니다. 목록에는 이미지에서 발견된 예외 항목 이외의 영역을 포함하는 “배경” 예외 항목도 포함되어 있습니다.
   - `AnomalyMask`은 분석된 영상에서 비정상 현상의 위치를 보여주는 마스크 이미지입니다.

   다음 예제와 같이 응답의 정보를 사용하여 분석된 이미지와 예외 마스크를 혼합하여 표시할 수 있습니다. 예제 코드는 분류 및 세분화 정보 표시 항목을 참조하세요.


Classification:
Prediction: Anomalous
Confidence: 99.9%
Segmentation:
Anomaly: cracked. Area: 6.2%

6. 명령 프롬프트에서 시작하기 test-images 폴더의 일반 이미지를 분석합니다. lookoutvision 프로필을 사용하여 자격 증명을 가져오려면 --profile lookoutvision-access 매개 변수를 추가하십시오. 예시:

```
aws lookoutvision detect-anomalies \
  --project-name getting-started \
  --model-version 1 \
  --content-type image/jpeg \
  --body /path/to/test-images/test-normal-1.jpg \
  --profile lookoutvision-access
```

다음과 같이 출력됩니다

```
{
    "DetectAnomalyResult": {
        "Source": {
            "Type": "direct"
        },
        "IsAnomalous": false,
        "Confidence": 0.9916400909423828,
        "Anomalies": [
            {
                "Name": "background",
                "PixelAnomaly": {
                    "TotalPercentageArea": 1.0,
                    "Color": "#FFFFFF"
                }
            }
        ],
        "AnomalyMask": "iVBORw0KGgoAAAANSUhEUgAAAkAAAA....."
    }
}
```

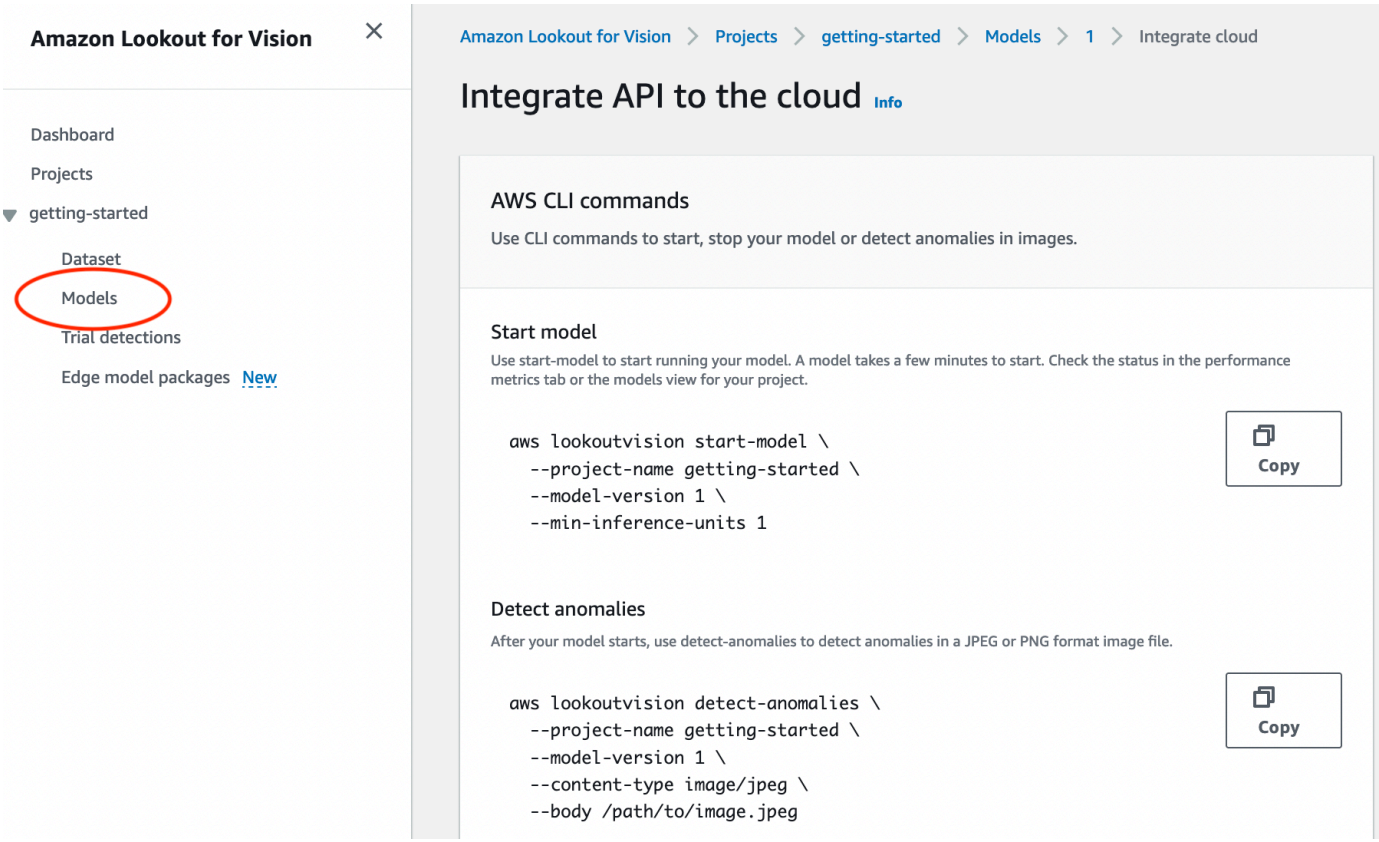
7. 출력에서 IsAnomalous의 false 값은 이미지를 비정상이 없는 것으로 분류한다는 점에 유의하십시오. Confidence을 사용하면 분류에 대한 신뢰도를 결정하는 데 도움이 됩니다. 또한 Anomalies 배열에는 background 예외 항목 레이블만 있습니다.

5단계: 모델 중지

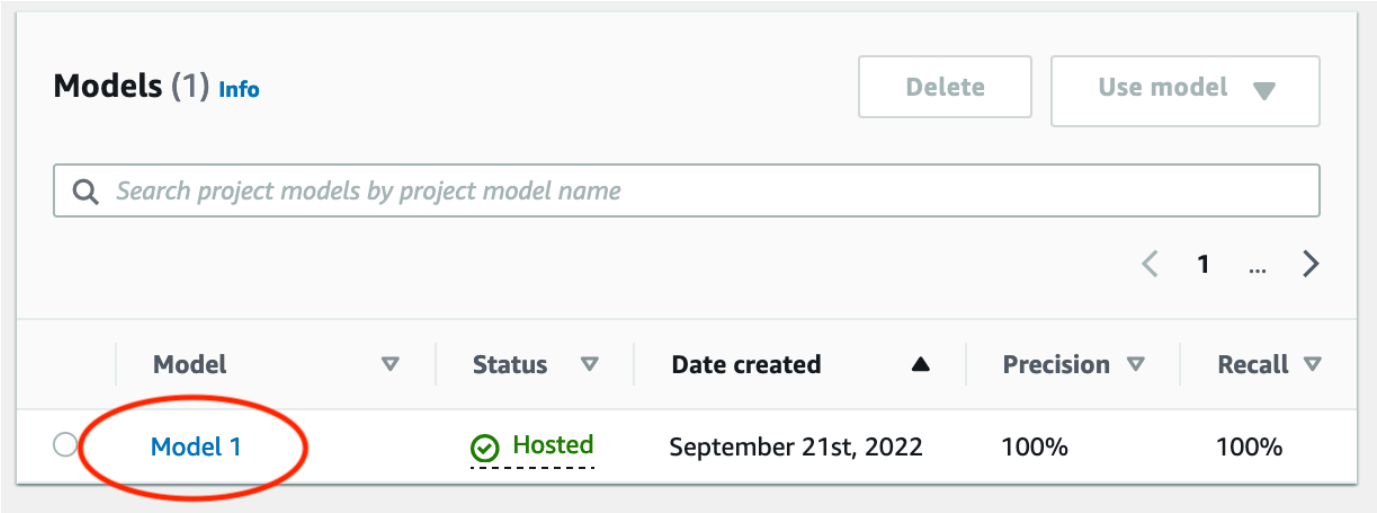
이 단계에서는 모델 호스팅을 중지합니다. 모델이 실행되는 시간만큼 요금이 부과됩니다. 이 모델을 사용하지 않는 경우 이를 중지해야 합니다. 다음 번에 필요할 때 모델을 다시 시작할 수 있습니다. 자세한 내용은 Amazon Lookout for Vision 모델 시작하기 단원을 참조하십시오.

모델을 중지하려면

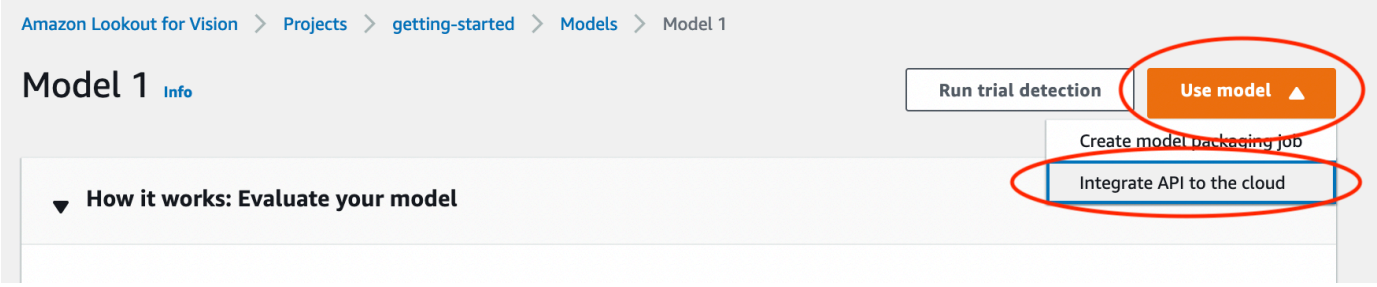
1. 탐색 창에서 모델을 선택합니다.



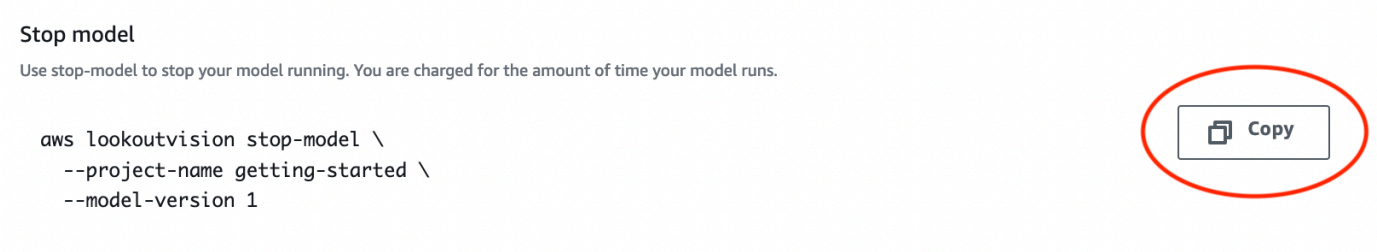
2. 모델 페이지에서 모델 1을 선택합니다.



3. 모델의 세부 정보 페이지에서 모델 사용을 선택한 다음 클라우드에 API 통합을 선택합니다.



4. AWS CLI 명령 섹션에서 `stop-model` AWS CLI 명령을 복사합니다.



5. 명령 프롬프트에서 이전 단계의 `stop-model` AWS CLI 명령을 입력하여 모델을 중지합니다. `lookoutvision` 프로필을 사용하여 자격 증명을 가져오려면 `--profile lookoutvision-access` 매개 변수를 추가하십시오. 예시:

```
aws lookoutvision stop-model \
  --project-name getting-started \
  --model-version 1 \
  --profile lookoutvision-access
```

직접 호출이 성공하면 다음 출력이 표시됩니다.

```
{
    "Status": "STOPPING_HOSTING"
}
```

6. 콘솔로 돌아가서 왼쪽 탐색 페이지에서 모델을 선택합니다.
7. 상태 열의 모델 상태가 학습 완료일 때 모델이 중지된 것입니다.

다음 단계

직접 만든 이미지로 모델을 만들 준비가 되면 먼저 프로젝트를 생성합니다.의 지침을 따르세요. 지침에는 Amazon Lookout for Vision 콘솔과 AWS SDK를 사용하여 모델을 생성하는 단계가 포함되어 있습니다.

다른 예제 데이터세트를 사용해 보려면 예제 코드 및 데이터세트을 참조하십시오.

Amazon Lookout for Vision 모델 생성

Amazon Lookout for Vision 모델은 모델 학습에 사용된 이미지에서 패턴을 찾아 새 이미지에 이상이 있는지 예측하는 기계 학습 모델입니다. 이 단원에서는 모델을 생성하고 교육하는 방법을 보여 줍니다. 모델을 교육한 후 모델의 성능을 평가합니다. 자세한 내용은 Amazon Lookout for Vision 모델 개선 단원을 참조하십시오.

첫 번째 모델을 생성하기 전에 Amazon Lookout for Vision 이해 및 Amazon Lookout for Vision을 시작하기을 읽어보는 것이 좋습니다. AWS SDK를 사용하는 경우 Amazon Lookout for Vision 작업 호출를 읽어보세요.

주제

- 프로젝트를 생성합니다.
- 데이터 세트를 생성합니다.
- 이미지 레이블 지정
- 모델 학습
- 모델 학습 문제 해결

프로젝트를 생성합니다.

Amazon Lookout for Vision 프로젝트는 Lookout for Vision 모델을 생성하고 관리하는 데 필요한 리소스를 그룹화한 것입니다. 프로젝트는 다음을 관리합니다.

- 데이터세트 — 모델 학습에 사용되는 이미지 및 이미지 레이블. 자세한 내용은 데이터 세트를 생성합니다. 단원을 참조하십시오.
- 모델 — 이상을 탐지하도록 학습시키는 소프트웨어. 여러 버전의 모델을 생성할 수 있습니다. 자세한 내용은 모델 학습 단원을 참조하십시오.

단일 사용 사례 (예: 단일 유형의 기계 부품에서 이상을 감지하는 경우)에는 프로젝트를 사용하는 것이 좋습니다.

> **Note**
>
> AWS CloudFormation 를 사용하여 Amazon Lookout for Vision 프로젝트를 프로비저닝하고 구성할 수 있습니다. 자세한 내용은 다음을 사용하여 Amazon Lookout for Vision 리소스 만들기 AWS CloudFormation 단원을 참조하십시오.

프로젝트를 보려면 프로젝트 보기를 참조하거나 Amazon Lookout for Vision 대시보드 사용를 여십시오. 모델을 삭제하려면 모델 삭제을 참조하십시오.

주제

- 프로젝트 생성(콘솔)
- 프로젝트 생성(SDK)

프로젝트 생성(콘솔)

다음 절차는 콘솔에서 사용자 지정 지표를 생성하는 방법을 보여줍니다.

프로젝트 생성(콘솔)

1. https://console.aws.amazon.com/lookoutvision/에서 Amazon Lookout for Vision 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 프로젝트를 선택합니다.
3. 프로젝트 생성을 선택합니다.
4. 프로젝트 이름에 프로젝트의 이름을 입력합니다.
5. 프로젝트 생성을 선택합니다. 프로젝트 세부 정보 페이지가 열립니다.
6. 데이터 세트를 생성하려면 데이터 세트를 생성합니다. 섹션의 절차를 따르십시오.

프로젝트 생성(SDK)

프로젝트 생성 작업을 사용하여 Amazon Lookout for Vision 프로젝트를 생성합니다. `CreateProject` 응답에는 프로젝트의 Amazon 리소스 이름(ARN)이 포함되어 있습니다. 그런 다음 CreateDataset를 호출하여 학습 및 테스트 데이터세트를 프로젝트에 추가합니다. 자세한 내용은 매니페스트 파일 (SDK)로 데이터세트 만들기 단원을 참조하십시오.

프로젝트에서 만든 프로젝트를 보려면 `ListProjects`를 호출하세요. 자세한 내용은 프로젝트 보기 단원을 참조하십시오.

프로젝트 (SDK) 생성

1. 아직 설치하지 않은 경우 및 AWS SDKs AWS CLI 를 설치하고 구성합니다. 자세한 내용은 4단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.
2. 다음 예제 코드를 실행하여 모델을 생성합니다.

   CLI

   `project-name`의 값을 프로젝트에 사용할 이름으로 변경합니다.

   ```
   aws lookoutvision create-project --project-name project name \
     --profile lookoutvision-access
   ```

   Python

   이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져온 것입니다. 전체 예제는 여기에서 확인하세요.

   ```
       @staticmethod
       def create_project(lookoutvision_client, project_name):
           """
           Creates a new Lookout for Vision project.

           :param lookoutvision_client: A Boto3 Lookout for Vision client.
           :param  project_name: The name for the new project.
           :return project_arn: The ARN of the new project.
           """
           try:
               logger.info("Creating project: %s", project_name)
               response =
   lookoutvision_client.create_project(ProjectName=project_name)
               project_arn = response["ProjectMetadata"]["ProjectArn"]
               logger.info("project ARN: %s", project_arn)
           except ClientError:
               logger.exception("Couldn't create project %s.", project_name)
               raise
           else:
               return project_arn
   ```

Java V2

이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져온 것입니다. 전체 예제는 여기에서 확인하세요.

```
/**
 * Creates an Amazon Lookout for Vision project.
 *
 * @param lfvClient   An Amazon Lookout for Vision client.
 * @param projectName The name of the project that you want to create.
 * @return ProjectMetadata Metadata information about the created project.
 */
public static ProjectMetadata createProject(LookoutVisionClient lfvClient,
 String projectName)
                throws LookoutVisionException {

        logger.log(Level.INFO, "Creating project: {0}", projectName);
        CreateProjectRequest createProjectRequest =
 CreateProjectRequest.builder().projectName(projectName)
                        .build();

        CreateProjectResponse response =
 lfvClient.createProject(createProjectRequest);

        logger.log(Level.INFO, "Project created. ARN: {0}",
 response.projectMetadata().projectArn());

        return response.projectMetadata();

}
```

3. 데이터 세트를 생성하려면 Amazon SageMaker AI Ground Truth 매니페스트 파일을 사용하여 데이터 세트 생성 섹션의 절차를 따르십시오.

데이터 세트를 생성합니다.

데이터세트에는 모델을 학습시키고 테스트하는 데 사용하는 이미지와 할당된 레이블이 포함됩니다. Amazon Lookout for Vision 콘솔 또는 CreateDataset 작업을 사용하여 프로젝트에 대한 데이터 세트를 생성합니다. 생성하려는 모델 유형 (이미지 분류 또는 이미지 분할)에 따라 데이터 세트 이미지에 레이블을 지정해야 합니다.

주제

데이터 세트용 이미지 준비

데이터 세트를 만들려면 이미지 컬렉션이 필요합니다. 이미지는 PNG 또는 JPEG 형식 파일이어야 합니다. 필요한 이미지의 수와 유형은 프로젝트에 단일 데이터 세트가 있는지 아니면 훈련 및 테스트 데이터 세트가 분리되어 있는지에 따라 달라집니다.

단일 데이터 세트

이미지 분류 모델을 만들려면 학습을 시작하기 위해 다음이 필요합니다.

- 일반 물체의 이미지가 20개 이상 있어야 합니다.
- 최소 10개 이상의 변칙 물체 이미지

이미지 분할 모델을 만들려면 학습을 시작하기 위해 다음이 필요합니다.

- 각 이상 유형의 이미지가 20개 이상 있어야 합니다.
- 각 변칙 이미지 (이상 유형이 있는 이미지)에는 한 가지 유형의 이상 항목만 있어야 합니다.
- 일반 물체의 이미지가 20개 이상 있어야 합니다.

별도의 학습 및 테스트 데이터세트 프로젝트

이미지 분류 모델을 생성하려면 다음이 필요합니다.

- 학습 데이터 세트에 있는 정상 물체의 이미지가 10개 이상 있어야 합니다.
- 테스트 데이터 세트에 있는 정상 물체의 이미지가 10개 이상 있어야 합니다.
- 테스트 데이터 세트에 있는 이상 물체의 이미지가 10개 이상 있어야 합니다.

이미지 분할 모델을 생성하려면 다음이 필요합니다.

- 각 데이터 세트에는 각 이상 유형의 이미지가 10개 이상 필요합니다.
- 각 변칙 이미지 (이상 유형이 있는 이미지)에는 한 가지 유형의 이상 항목만 포함되어야 합니다.
- 각 데이터 세트에는 정상 개체의 이미지가 10개 이상 있어야 합니다.

더 높은 품질의 모델을 만들려면 최소 이미지 수보다 많은 이미지를 사용하세요. 분할 모델을 만드는 경우 여러 예외 유형이 있는 이미지를 포함하는 것이 좋지만, 이러한 이미지는 Lookout for Vision이 학습을 시작하는 데 필요한 최소 수에는 포함되지 않습니다.

이미지는 단일 유형의 객체여야 합니다. 또한 카메라 위치, 조명, 물체 포즈 등 이미지 캡처 조건이 일정해야 합니다.

훈련 데이터 세트와 테스트 데이터 세트의 모든 이미지는 크기가 같아야 합니다. 나중에 학습된 모델로 분석하는 이미지는 훈련 및 테스트 데이터 세트 이미지와 크기가 같아야 합니다. 자세한 내용은 이미지에서 이상 탐지 단원을 참조하십시오.

모든 훈련 영상과 테스트 영상은 고유한 영상이어야 하며, 가급적이면 고유한 객체를 포함하는 영상이어야 합니다. 일반 이미지는 분석 대상 물체의 일반적인 변화를 캡처해야 합니다. 변칙 이미지는 다양한 변칙 샘플을 캡처해야 합니다.

Amazon Lookout for Vision은 사용자가 사용할 수 있는 예시 이미지를 제공합니다. 자세한 내용은 이미지 분류 데이터 세트 단원을 참조하십시오.

이미지 제한에 대한 내용은 할당량을 참조하십시오.

데이터 세트 생성

프로젝트용 데이터 세트를 만들 때 프로젝트의 초기 데이터 세트 구성을 선택합니다. 또한 Lookout for Vision에서 이미지를 가져오는 위치도 선택할 수 있습니다.

프로젝트의 데이터 세트 구성 선택

프로젝트에서 첫 번째 데이터 세트 생성 시 다음 데이터 세트 구성 중 하나를 선택합니다.

- 단일 데이터 세트 — 단일 데이터 세트 프로젝트는 단일 데이터 세트를 사용하여 모델을 학습하고 테스트합니다. 단일 데이터 세트를 사용하면 Amazon Lookout for Vision에서 교육 및 테스트 이미지를 선택할 수 있으므로 교육이 간소화됩니다. Amazon Lookout for Vision은 교육 중에 데이터 세트를 내부적으로 교육 데이터 세트와 테스트 데이터 세트로 분할합니다. 이 분할 데이터 세트에 액세스할 수 없습니다. 대부분의 시나리오에서는 단일 데이터 세트 프로젝트를 사용하는 것이 좋습니다.
- 학습 및 테스트 데이터 세트 분리 - 학습, 테스트, 성능 튜닝을 더 세밀하게 제어하려면 별도의 학습 및 테스트 데이터 세트를 포함하도록 프로젝트를 구성할 수 있습니다. 테스트에 사용되는 이미지를

제어하려는 경우 또는 사용하려는 이미지의 벤치마크 세트가 이미 있는 경우 별도의 테스트 데이터 세트를 사용하십시오.

기존 단일 데이터 세트 프로젝트에 테스트 데이터 세트를 추가할 수 있습니다. 그러면 단일 데이터 세트가 학습 데이터 세트가 됩니다. 학습 데이터 세트와 테스트 데이터 세트가 분리된 프로젝트에서 테스트 데이터 세트를 제거하면 프로젝트는 단일 데이터 세트 프로젝트가 됩니다. 자세한 내용은 데이터 세트 삭제 단원을 참조하십시오.

이미지 가져오기

데이터 세트를 만들 때 이미지를 가져올 위치를 선택합니다. 이미지를 가져오는 방법에 따라 이미지에 이미 레이블이 지정되어 있을 수 있습니다. 데이터세트를 만든 후 이미지에 레이블이 지정되지 않은 경우 이미지 레이블 지정을 참조하십시오.

다음 방법 중 하나로 데이터세트를 생성하고 이미지를 가져옵니다.

- 로컬 컴퓨터에서 이미지 가져오기 이미지에는 레이블이 지정되지 않습니다. Lookout for Vision 콘솔을 사용하여 레이블을 추가합니다.
- S3 버킷에서 이미지를 가져옵니다. Amazon Lookout for Vision은 폴더 이름을 사용하여 이미지에 레이블을 지정함으로써 이미지를 분류할 수 있습니다. 일반 이미지에 `normal`를 사용합니다. 변칙 이미지에 `anomaly`를 사용합니다. 분할 레이블은 자동으로 할당할 수 없습니다.
- 레이블이 지정된 이미지가 포함된 Amazon SageMaker AI Ground Truth 매니페스트 파일을 가져옵니다. 자체 매니페스트 파일을 생성하고 가져올 수 있습니다. 이미지가 많은 경우 SageMaker AI Ground Truth 레이블 지정 서비스를 사용하는 것이 좋습니다. 그런 다음 Amazon SageMaker AI Ground Truth 작업에서 출력 매니페스트 파일을 가져옵니다. 필요에 따라 Lookout for Vision 콘솔을 사용하여 레이블을 추가하거나 변경할 수 있습니다.

AWS SDK를 사용하는 경우 Amazon SageMaker AI Ground Truth 매니페스트 파일을 사용하여 데이터 세트를 생성합니다. 자세한 내용은 Amazon SageMaker AI Ground Truth 매니페스트 파일을 사용하여 데이터 세트 생성 단원을 참조하십시오.

데이터 세트를 생성한 후 이미지에 레이블이 지정되면 모델을 학습시킬 수 있습니다. 이미지에 레이블이 지정되지 않은 경우 만들려는 모델 유형에 따라 레이블을 추가하세요. 자세한 내용은 이미지 레이블 지정 단원을 참조하십시오.

기존 데이터 세트에 이미지를 더 추가할 수 있습니다. 자세한 내용은 데이터 세트에 이미지 추가 단원을 참조하십시오.

로컬 컴퓨터에 저장된 이미지를 사용하여 데이터 세트 만들기

컴퓨터에서 직접 로드한 이미지를 사용하여 데이터 세트를 만들 수 있습니다. 한 번에 최대 30개의 이미지를 업로드할 수 있습니다. 이 절차에서는 단일 데이터 세트 프로젝트 또는 별도의 학습 및 테스트 데이터 세트가 있는 프로젝트를 만들 수 있습니다.

> **Note**
>
> 방금 프로젝트를 생성합니다.을 완료했다면 콘솔에 프로젝트 대시보드가 표시되므로 1~3단계를 수행할 필요가 없습니다.

로컬 컴퓨터의 이미지를 사용하여 데이터 세트를 만들려면 (콘솔)

1. https://console.aws.amazon.com/lookoutvision/에서 Amazon Lookout for Vision 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 프로젝트를 선택합니다.
3. 프로젝트 페이지에서 데이터 세트에 추가하려는 프로젝트를 선택합니다.
4. 프로젝트 세부 정보 페이지에서 데이터 세트 생성 을 선택합니다.
5. 단일 데이터 세트 탭 또는 학습 및 테스트 데이터 세트 분리 탭을 선택하고 단계를 따르세요.

   Single dataset

   a. 데이터 세트 구성 섹션에서 단일 데이터 세트 만들기를 선택합니다.
   b. 이미지 소스 구성 섹션에서 컴퓨터에서 이미지 업로드를 선택합니다.
   c. 데이터 세트 생성을 선택합니다.
   d. 데이터 세트 페이지에서 이미지 추가를 선택합니다.
   e. 컴퓨터 파일에서 데이터 세트에 업로드할 이미지를 선택합니다. 이미지를 드래그하거나 로컬 컴퓨터에서 업로드할 이미지를 선택할 수 있습니다.
   f. 이미지 업로드를 선택합니다.

   Separate training and test datasets

   a. 데이터 세트 구성 섹션에서 학습 데이터 세트 및 테스트 데이터 세트 만들기를 선택합니다.
   b. 학습 데이터 세트 세부 정보 섹션에서 컴퓨터에서 이미지 업로드를 선택합니다.
   c. 테스트 데이터 세트 세부정보 섹션에서 컴퓨터에서 이미지 업로드를 선택합니다.

> **Note**
>
> 훈련 데이터 세트와 테스트 데이터 세트에는 서로 다른 이미지 소스가 있을 수 있습니다.

d. 데이터 세트 생성을 선택합니다. 각 데이터 세트에 대한 훈련 탭과 테스트 탭이 있는 데이터 세트 페이지가 나타납니다.

e. 작업을 선택한 다음 훈련 데이터 세트에 이미지 추가를 선택합니다.

f. 데이터 세트에 업로드하려는 이미지를 선택합니다. 이미지를 드래그하거나 로컬 컴퓨터에서 업로드할 이미지를 선택할 수 있습니다.

g. 이미지 업로드를 선택합니다.

h. 5e~5g단계를 반복합니다. 5e단계에서는 작업을 선택한 다음 테스트 데이터 세트에 이미지 추가를 선택합니다.

6. 이미지 레이블 지정에 나온 단계에 따라 이미지에 레이블을 지정합니다.
7. 모델 학습에 나온 단계에 따라 모델을 훈련하세요.

Amazon S3 버킷에 저장된 이미지를 사용하여 데이터 세트를 생성합니다.

Amazon S3 버킷에 저장된 이미지를 사용하여 데이터 세트를 생성할 수 있습니다. 이 옵션을 사용하면 Amazon S3 버킷의 폴더 구조를 사용하여 이미지를 자동으로 분류할 수 있습니다. 이미지는 콘솔 버킷 또는 계정의 다른 Amazon S3 버킷에 저장할 수 있습니다.

자동 레이블 지정을 위한 폴더 설정

데이터 세트를 만드는 동안 이미지가 포함된 폴더의 이름을 기반으로 이미지에 레이블 이름을 할당할 수 있습니다. 폴더는 데이터 세트를 생성할 때 S3 URI에 지정한 Amazon S3 폴더 경로의 하위 폴더여야 합니다.

다음은 시작하기 예제 이미지의 `train` 폴더입니다. Amazon S3 폴더 위치를 `S3-bucket/circuitboard/train/`로 지정하면 `normal` 폴더의 이미지에 레이블 `Normal`이 할당됩니다. 폴더의 이미지에는 `anomaly` 레이블이 할당됩니다`Anomaly`. 더 깊은 하위 폴더의 이름은 이미지에 레이블을 지정하는 데 사용되지 않습니다.

```
S3-bucket
  ### circuitboard
```

```
### train
    ### anomaly
        ### train-anomaly_1.jpg
        ### train-anomaly_2.jpg
        ### .
        ### .
    ### normal
        ### train-normal_1.jpg
        ### train-normal_2.jpg
        ### .
        ### .
```

Amazon S3 버킷의 이미지를 사용하여 데이터 세트를 생성합니다.

다음 절차는 Amazon S3 버킷에 저장된 분류 예제 이미지를 사용하여 데이터 세트를 생성합니다. 자체 이미지를 사용하려면 자동 레이블 지정을 위한 폴더 설정에 설명된 폴더 구조를 생성하십시오.

또한 이 절차에서는 단일 데이터 세트 프로젝트 또는 별도의 교육 및 테스트 데이터 세트를 사용하는 프로젝트를 만드는 방법도 보여줍니다.

이미지에 자동으로 레이블을 지정하지 않는 경우 데이터 세트를 만든 후 이미지에 레이블을 지정해야 합니다. 자세한 내용은 이미지 분류 (콘솔) 단원을 참조하십시오.

> **Note**
>
> 방금 프로젝트를 생성합니다.을 완료했다면 콘솔에 프로젝트 대시보드가 표시되므로 1~4단계를 수행할 필요가 없습니다.

Amazon S3 버킷에 저장된 이미지를 사용하여 데이터 세트를 생성할 수 있습니다.

1. 아직 업로드하지 않은 경우 Amazon S3 버킷에 시작 이미지를 업로드합니다. 자세한 내용은 이미지 분류 데이터 세트 단원을 참조하십시오.
2. https://console.aws.amazon.com/lookoutvision/에서 Amazon Lookout for Vision 콘솔을 엽니다.
3. 왼쪽 탐색 창에서 프로젝트를 선택합니다.
4. 프로젝트 페이지에서 데이터 세트에 추가하려는 프로젝트를 선택합니다. 프로젝트 세부 정보 페이지가 열립니다.
5. 데이터 세트 생성을 선택합니다. 데이터세트 생성 페이지가 표시됩니다.

> **Tip**
>
> 시작하기 안내를 따르는 경우 교육 데이터 세트와 테스트 데이 터세트 만들기를 선택하세요.

6. 단일 데이터 세트 탭 또는 학습 및 테스트 데이터 세트 분리 탭을 선택하고 단계를 따르세요.

   Single dataset

   a. 데이터 세트 구성 섹션에서 단일 데이터 세트 만들기를 선택합니다.

   b. 이미지 소스 구성 섹션에 7~9단계의 정보를 입력합니다.

   Separate training and test datasets

   a. 데이터 세트 구성 섹션에서 학습 데이터 세트 및 테스트 데이터 세트 만들기를 선택합니다.

   b. 학습 데이터 세트의 경우 학습 데이터 세트 세부 정보 섹션에 7~9단계의 정보를 입력합니다.

   c. 테스트 데이터 세트의 경우 테스트 데이터 세트 세부 정보 섹션에 7~9단계의 정보를 입력합니다.

   > **Note**
   >
   > 학습 데이터 세트와 테스트 데이터 세트에는 서로 다른 이미지 소스가 있을 수 있습니다.

7. Amazon S3 버킷에서 이미지 가져오기를 선택합니다.

8. S3 URI에 Amazon S3 버킷 위치 및 폴더 경로를 입력합니다. `bucket`을(를) Amazon S3 버킷의 이름으로 변경합니다.

   a. 단일 데이터 세트 프로젝트 또는 학습 데이터 세트를 생성하는 경우 다음을 입력합니다.

   ```
   s3://bucket/circuitboard/train/
   ```

   b. 테스트 데이터 세트를 만들려면 다음을 입력하세요.

   ```
   s3://bucket/circuitboard/test/
   ```

9. 폴더를 기반으로 이미지에 레이블 자동 첨부를 선택합니다.

10. 데이터 세트 생성을 선택합니다. 레이블이 지정된 이미지가 포함된 데이터 세트 페이지가 열립니다.
11. 모델 학습에 나온 단계에 따라 모델을 훈련하세요.

Amazon SageMaker AI Ground Truth 매니페스트 파일을 사용하여 데이터 세트 생성

매니페스트 파일에는 모델을 학습시키고 테스트하는 데 사용할 수 있는 이미지 및 이미지 레이블에 대한 정보가 들어 있습니다. Amazon S3 버킷에 매니페스트 파일을 저장하고 이를 사용하여 데이터 세트를 생성할 수 있습니다. 자체 매니페스트 파일을 생성하거나 Amazon SageMaker AI Ground Truth 작업의 출력과 같은 기존 매니페스트 파일을 사용할 수 있습니다.

주제

- Amazon SageMaker Ground Truth 작업 사용하기
- 매니페스트 파일 만들기

Amazon SageMaker Ground Truth 작업 사용하기

이미지에 레이블을 지정하는 데는 상당한 시간이 걸릴 수 있습니다. 예를 들어, 예외 항목 주위에 마스크를 정확하게 그리는 데 10초가 걸릴 수 있습니다. 이미지가 100개라면 레이블을 지정하는 데 몇 시간이 걸릴 수 있습니다. 이미지에 직접 레이블을 지정하는 대신 Amazon SageMaker Ground Truth를 사용하는 것을 고려해 보십시오.

Amazon SageMaker AI Ground Truth를 사용하면 선택한 공급업체인 Amazon Mechanical Turk 또는 내부 프라이빗 작업 인력의 작업자를 사용하여 레이블이 지정된 이미지 세트를 생성할 수 있습니다. 자세한 내용은 Amazon SageMaker AI Ground Truth를 사용하여 데이터 레이블 지정을 참조하세요.

Amazon Mechanical Turk를 사용하는 데는 비용이 부과됩니다. 또한 Amazon Ground Truth 라벨 제작 작업을 완료하는 데 며칠이 걸릴 수 있습니다. 비용이 문제가 되거나 모델을 빠르게 학습시켜야 하는 경우 Amazon Lookout for Vision 콘솔을 사용하여 이미지에 레이블을 지정하는 것이 좋습니다.

Amazon SageMaker AI Ground Truth 레이블 지정 작업을 사용하여 이미지 분류 모델 및 이미지 분할 모델에 적합한 이미지에 레이블을 지정할 수 있습니다. 작업이 완료되면 출력 매니페스트 파일을 사용하여 Amazon Lookout for Vision 데이터세트를 생성합니다.

이미지 분류

이미지 분류 모델용 이미지에 라벨을 지정하려면 이미지 분류 (단일 레이블) 작업에 대한 라벨링 작업을 생성하십시오.

이미지 세분화

이미지 세분화 모델용 이미지에 레이블을 지정하려면 이미지 분류 (단일 레이블) 작업을 위한 레이블 지정 작업을 생성하십시오. 그런 다음 작업을 연계하여 이미지 의미 체계 세분화 작업을 위한 레이블 지정 작업을 생성하십시오.

레이블 지정 작업을 사용하여 이미지 세분화 모델의 부분 매니페스트 파일을 만들 수도 있습니다. 예를 들어, 이미지 분류 (단일 레이블) 작업으로 이미지를 세분화할 수 있습니다. 작업 출력으로 Lookout for Vision 데이터세트를 생성한 후 Amazon Lookout for Vision 콘솔을 사용하여 데이터세트 이미지에 세분화 마스크와 예외 항목 레이블을 추가합니다.

Amazon SageMaker AI Ground Truth로 이미지 레이블 지정

다음 절차에서는 Amazon SageMaker AI Ground Truth 이미지 레이블 지정 작업으로 이미지에 레이블을 지정하는 방법을 보여줍니다. 이 절차는 이미지 세분화 매니페스스트 파일을 생성하고 선택적으로 이미지 레이블 지정 작업을 연결하여 이미지 분할 매니페스트 파일을 생성합니다. 프로젝트에 별도의 테스트 데이터 세트를 만들려면 이 절차를 반복하여 테스트 데이터 세트용 매니페스트 파일을 만드세요.

Amazon SageMaker AI Ground Truth로 이미지에 레이블을 지정하려면(콘솔)

1. 라벨링 작업 생성 (콘솔)의 지침에 따라 이미지 분류 (단일 레이블) 작업을 위한 Ground Truth 작업을 생성하십시오.

   a. 10단계에서는 작업 범주 드롭다운 메뉴에서 이미지를 선택하고 작업 유형으로 이미지 분류 (단일 레이블)를 선택합니다.

   b. 16단계의 경우 이미지 분류 (단일 레이블) 레이블 지정 도구 섹션에서 정상 및 이상 레이블이라는 두 개의 레이블을 추가합니다.
2. 직원이 이미지 분류를 완료할 때까지 기다리십시오.
3. 이미지 세분화 모델용 데이터세트를 만드는 경우 다음을 수행하세요. 4단계로 이동합니다.

   a. Amazon SageMaker AI Ground Truth 콘솔에서 레이블 지정 작업 페이지를 엽니다.

   b. 이전에 생성한 작업을 선택합니다. 그러면 작업 메뉴가 활성화됩니다.

   c. 작업 메뉴에서 Chain(연결)을 선택합니다. 작업 상세 페이지가 열립니다.

   d. 작업 유형에서 의미 체계 세분화를 선택합니다.
   e. 다음을 선택합니다.
   f. 의미 체계 세분화 라벨링 도구 섹션에서 모델에서 찾고자 하는 각 예외 유형에 대해 예외 항목 레이블을 추가합니다.
   g. 생성(Create)을 선택합니다.
   h. 직원이 이미지에 라벨을 붙일 때까지 기다리세요.
4. Ground Truth 콘솔을 열고 라벨링 작업 페이지를 엽니다.
5. 이미지 분류 모델을 생성하는 경우 1단계에서 생성한 작업을 선택합니다. 이미지 세분화 모델을 만드는 경우 3단계에서 만든 작업을 선택하십시오.
6. 라벨링 작업 요약의 출력 데이터 세트 위치에서 S3 위치를 엽니다. 매니페스트 파일 위치를 기록해 둡니다. `s3://output-dataset-location/manifests/output/output.manifest`위치가 맞아야 합니다.
7. 테스트 데이터 세트에 대한 매니페스트 파일을 만들려면 이 절차를 반복하세요. 그렇지 않으면 데이터 세트 생성의 지침에 따라 매니페스트 파일로 데이터 세트를 만드세요.

데이터 세트 생성

이 절차를 사용하여 Amazon SageMaker AI Ground Truth로 이미지 레이블 지정의 6단계에서 기록해 둔 매니페스트 파일을 사용하여 Lookout for Vision 프로젝트에서 데이터세트를 만들 수 있습니다. 매니페스트 파일은 단일 데이터 세트 프로젝트에 대한 교육 데이터 세트를 만듭니다. 프로젝트에 별도의 테스트 데이터 세트를 포함하려면 다른 Amazon SageMaker AI Ground Truth 작업을 실행하여 테스트 데이터 세트에 대한 매니페스트 파일을 생성할 수 있습니다. 또는 매니페스트 파일을 직접 생성할 수도 있습니다. Amazon S3 버킷 또는 로컬 컴퓨터에서 테스트 데이터 세트로 이미지를 가져올 수도 있습니다. (모델을 학습시키기 전에 이미지에 레이블이 필요할 수 있습니다.)

이 절차에서는 프로젝트에 데이터 세트가 없다고 가정합니다.

Lookout for Vision을 사용하여 데이터세트를 만들려면 (콘솔)

1. https://console.aws.amazon.com/lookoutvision/에서 Amazon Lookout for Vision 콘솔을 엽니다.
2. Get started를 선택합니다.
3. 왼쪽 탐색 창에서 프로젝트를 선택합니다.
4. 매니페스트 파일에 추가할 프로젝트를 선택합니다.
5. 작동 방식 섹션에서 데이터세트 생성을 선택합니다.
6. 단일 데이터 세트 탭 또는 학습 및 테스트 데이터 세트 분리 탭을 선택하고 단계를 따르세요.

Single dataset

1. 단일 데이터 세트 만들기를 선택합니다.
2. 이미지 원본 구성 섹션에서 SageMaker Ground Truth로 레이블이 지정된 이미지 가져오기를 선택합니다.
3. .manifest 파일 위치의 경우 Amazon SageMaker AI Ground Truth로 이미지 레이블 지정의 6단계에서 기록해 둔 매니페스트 파일의 위치를 입력합니다.

Separate training and test datasets

1. 교육 데이터세트 및 테스트 데이터세트 만들기를 선택합니다.
2. 교육 데이터세트 세부 정보 섹션에서 SageMaker Ground Truth로 레이블이 지정된 이미지 가져오기를 선택합니다.
3. .manifest 파일 위치에서는 Amazon SageMaker AI Ground Truth로 이미지 레이블 지정의 6단계에서 기록해 둔 매니페스트 파일의 위치입니다.
4. 테스트 데이터세트 세부정보 섹션에서 SageMaker Ground Truth로 레이블이 지정된 이미지 가져오기를 선택합니다.
5. .manifest 파일 위치에서는 Amazon SageMaker AI Ground Truth로 이미지 레이블 지정의 6단계에서 기록해 둔 매니페스트 파일의 위치입니다. 테스트 데이터 세트에는 별도의 매니페스트 파일이 필요하다는 점을 기억하세요.

7. 제출을 선택합니다.
8. 모델 학습에 나온 단계에 따라 모델을 훈련하세요.

매니페스트 파일 만들기

SageMaker AI Ground Truth 형식 매니페스트 파일을 가져와서 데이터 세트를 생성할 수 있습니다. 이미지에 SageMaker AI Ground Truth 매니페스트 파일이 아닌 형식으로 레이블이 지정된 경우 다음 정보를 사용하여 SageMaker AI Ground Truth 형식 매니페스트 파일을 생성합니다.

매니페스트 파일은 JSON 라인 형식이며, 각 라인은 이미지의 레이블 정보를 나타내는 완전한 JSON 객체입니다. 이미지 분류 및 이미지 분할에는 다양한 형식이 있습니다. 퍼센트 인코딩은 UTF-8 인코딩과 호환됩니다.

> **Note**
>
> 이 섹션의 JSON 라인 예제는 가독성을 위해 형식이 지정되었습니다.

매니페스트 파일이 참조하는 이미지는 동일한 Amazon S3 버킷에 있어야 합니다. 매니페스트는 다른 버킷에 있을 수 있습니다. JSON 라인의 `source-ref` 필드에 이미지 위치를 지정합니다.

코드를 사용하여 매니페스트 파일을 만들 수 있습니다. Amazon Lookout for Vision Lab Python 노트북은 회로 기판 예제 이미지에 대한 이미지 분류 매니페스트 파일을 생성하는 방법을 보여줍니다. 또는 코드 예제 리포지토리의 데이터 세트 예제 AWS 코드를 사용할 수 있습니다. CSV (쉼표로 분리된 값) 파일을 사용하여 매니페스트 파일을 쉽게 만들 수 있습니다. 자세한 내용은 CSV 파일로 분류 매니페스트 파일 생성 단원을 참조하십시오.

주제

- 이미지 분류를 위한 JSON 라인 정의
- 이미지 분할을 위한 JSON 라인 정의
- CSV 파일로 분류 매니페스트 파일 생성
- 매니페스트 파일로 데이터세트 만들기 (콘솔)
- 매니페스트 파일 (SDK)로 데이터세트 만들기

이미지 분류를 위한 JSON 라인 정의

Amazon Lookout for Vision 매니페스트 파일에서 사용하려는 각 이미지에 대해 JSON 라인을 정의합니다. 분류 모델을 만들려면 JSON 라인에 정상 또는 이상 이미지 분류가 포함되어야 합니다. JSON 라인은 SageMaker AI Ground Truth 분류 작업 출력 형식입니다. 매니페스트 파일은 가져오려는 각 이미지당 하나씩, 하나 이상의 JSON 라인으로 구성됩니다.

분류된 이미지에 대한 매니페스트 파일을 만들려면

1. 빈 텍스트 파일 생성
2. 가져올 각 이미지에 JSON 라인을 추가합니다. 각 JSON 라인은 다음과 비슷한 모습이어야 합니다.

```
{"source-ref":"s3://lookoutvision-console-us-east-1-nnnnnnnnnn/gt-job/manifest/
IMG_1133.png","anomaly-label":1,"anomaly-label-metadata":{"confidence":0.95,"job-
name":"labeling-job/testclconsolebucket","class-name":"normal","human-
```

```
annotated":"yes","creation-date":"2020-04-15T20:17:23.433061","type":"groundtruth/
image-classification"}}
```

3. 파일을 저장합니다.

> **Note**
>
> .manifest 확장을 사용할 수 있지만 필수는 아닙니다.

4. 생성한 매니페스트 파일을 사용하여 데이터 세트를 생성하세요. 자세한 내용은 매니페스트 파일 만들기 단원을 참조하십시오.

분류 JSON 라인

이 섹션에서는 이미지를 정상 또는 비정상 이미지로 분류하는 JSON 라인을 만드는 방법을 알아봅니다.

이상 JSON 라인

다음 JSON 라인은 예외 항목으로 표시된 이미지를 보여줍니다. 참고로 class-name의 값은 anomaly입니다.

```
{
    "source-ref": "s3: //bucket/image/anomaly/abnormal-1.jpg",
    "anomaly-label-metadata": {
        "confidence": 1,
        "job-name": "labeling-job/auto-label",
        "class-name": "anomaly",
        "human-annotated": "yes",
        "creation-date": "2020-11-10T03:37:09.600",
        "type": "groundtruth/image-classification"
    },
    "anomaly-label": 1
}
```

정상 JSON 라인

다음 JSON 라인은 정상이라는 라벨이 붙은 이미지를 보여줍니다. 참고로 class-name의 값은 normal입니다.

```
{
    "source-ref": "s3: //bucket/image/normal/2020-10-20_12-14-55_613.jpeg",
    "anomaly-label-metadata": {
        "confidence": 1,
        "job-name": "labeling-job/auto-label",
        "class-name": "normal",
        "human-annotated": "yes",
        "creation-date": "2020-11-10T03:37:09.603",
        "type": "groundtruth/image-classification"
    },
    "anomaly-label": 0
}
```

JSON 라인 키 및 값

다음 정보는 Amazon Lookout for Vision JSON 라인의 키와 값을 설명합니다.

소스 참조

(필수) 이미지의 Amazon S3 위치입니다. 형식은 "s3://*BUCKET*/*OBJECT_PATH*"입니다. 가져온 데이터세트의 이미지는 동일한 Amazon S3 버킷에 저장되어야 합니다.

이상 레이블

(필수) 레이블 속성 키 anomaly-label 또는 선택한 다른 키 이름을 사용하십시오. Amazon Lookout for Vision에서는 키 값 (위 예시 0)이 필요하지만 사용되지는 않습니다. Amazon Lookout for Vision에서 생성한 출력 매니페스트는 비정상 이미지의 경우 값을 1로 변환하고 일반 이미지의 경우 값을 0로 변환합니다. class-name의 값에 따라 이미지가 정상인지 비정상인지가 결정됩니다.

-metadata가 추가된 필드 이름으로 식별되는 상응하는 메타데이터가 있어야 합니다. 예를 들어 "anomaly-label-metadata"입니다.

예외 레이블-메타데이터

(필수) 레이블 속성에 대한 메타데이터 필드 이름은 -metadata가 추가된 레이블 속성과 동일해야 합니다.

confidence

(선택) Amazon Lookout for Vision이 사용할 수 없습니다. 값을 지정하는 경우 1의 값을 사용하십시오.

job-name

(선택 사항) 이미지를 처리하는 작업에 원하는 이름을 붙이세요.

class-name

(필수) 이미지에 일반 내용이 포함된 경우 `normal`을 지정하고, 그렇지 않으면 `anomaly`를 지정하십시오. `class-name`의 값이 다른 값인 경우 이미지는 레이블이 지정되지 않은 이미지로 데이터세트에 추가됩니다. 이미지에 레이블을 지정하려면 데이터 세트에 이미지 추가을 참조하십시오.

human-annotated

(필수) 사람이 주석을 완성했으면 `"yes"` 항목을 지정하세요. 아닌 경우에는 `"no"`로 지정합니다.

creation-date

(선택 사항) 레이블이 생성된 협정 세계시 (UTC) 날짜 및 시간입니다.

type

(필수) 이미지에 적용해야 하는 처리 유형입니다. 이미지 수준 예외 레이블의 경우 값은 `"groundtruth/image-classification"`입니다.

이미지 분할을 위한 JSON 라인 정의

Amazon Lookout for Vision 매니페스트 파일에서 사용하려는 각 이미지에 대해 JSON 라인을 정의합니다. 세분화 모델을 만들려면 JSON 라인에 이미지에 대한 세분화 및 분류 정보가 포함되어야 합니다. 매니페스트 파일은 가져오려는 각 이미지당 하나씩, 하나 이상의 JSON 라인으로 구성됩니다.

분류된 이미지를 위한 매니페스트 파일을 만들려면

1. 빈 텍스트 파일 생성
2. 가져올 각 이미지에 JSON 라인을 추가합니다. 각 JSON 라인은 다음과 비슷한 모습이어야 합니다.

```
{"source-ref":"s3://path-to-image","anomaly-label":1,"anomaly-label-metadata":
{"class-name":"anomaly","creation-date":"2021-10-12T14:16:45.668","human-
annotated":"yes","job-name":"labeling-job/classification-job","type":"groundtruth/
image-classification","confidence":1},"anomaly-mask-ref":"s3://path-to-
image","anomaly-mask-ref-metadata":{"internal-color-map":{"0":{"class-
name":"BACKGROUND","hex-color":"#ffffff","confidence":0.0},"1":{"class-
name":"scratch","hex-color":"#2ca02c","confidence":0.0},"2":{"class-
name":"dent","hex-color":"#1f77b4","confidence":0.0}},"type":"groundtruth/
```

```
semantic-segmentation","human-annotated":"yes","creation-
date":"2021-11-23T20:31:57.758889","job-name":"labeling-job/segmentation-job"}}
```

3. 파일을 저장합니다.

> **Note**
>
> .manifest 확장을 사용할 수 있지만 필수는 아닙니다.

4. 생성한 매니페스트 파일을 사용하여 데이터 세트를 생성하세요. 자세한 내용은 매니페스트 파일 만들기 단원을 참조하십시오.

세분화 JSON 라인

이 섹션에서는 이미지에 대한 세분화 및 분류 정보가 포함된 JSON 라인을 만드는 방법을 알아봅니다.

다음 JSON 라인은 분할 및 분류 정보가 포함된 이미지를 보여줍니다. anomaly-label-metadata에는 분류 정보가 들어 있습니다. anomaly-mask-ref 및 anomaly-mask-ref-metadata은 세분화 정보를 포함합니다.

```
{
    "source-ref": "s3://path-to-image",
    "anomaly-label": 1,
    "anomaly-label-metadata": {
        "class-name": "anomaly",
        "creation-date": "2021-10-12T14:16:45.668",
        "human-annotated": "yes",
        "job-name": "labeling-job/classification-job",
        "type": "groundtruth/image-classification",
        "confidence": 1
    },
    "anomaly-mask-ref": "s3://path-to-image",
    "anomaly-mask-ref-metadata": {
        "internal-color-map": {
            "0": {
                "class-name": "BACKGROUND",
                "hex-color": "#ffffff",
                "confidence": 0.0
            },
            "1": {
                "class-name": "scratch",
```

```
                "hex-color": "#2ca02c",
                "confidence": 0.0
            },
            "2": {
                "class-name": "dent",
                "hex-color": "#1f77b4",
                "confidence": 0.0
            }
        },
        "type": "groundtruth/semantic-segmentation",
        "human-annotated": "yes",
        "creation-date": "2021-11-23T20:31:57.758889",
        "job-name": "labeling-job/segmentation-job"
    }
}
```

JSON 라인 키 및 값

다음 정보는 Amazon Lookout for Vision JSON 라인의 키와 값을 설명합니다.

소스 참조

(필수) 이미지의 Amazon S3 위치입니다. 형식은 "s3://*BUCKET*/*OBJECT_PATH*"입니다. 가져온 데이터세트의 이미지는 동일한 Amazon S3 버킷에 저장되어야 합니다.

이상 레이블

(필수) 레이블 속성 키 anomaly-label 또는 선택한 다른 키 이름을 사용하십시오. Amazon Lookout for Vision에서는 키 값 (위 예시 1)이 필요하지만 사용되지는 않습니다. Amazon Lookout for Vision에서 생성한 출력 매니페스트는 비정상 이미지의 경우 값을 1로 변환하고 일반 이미지의 경우 값을 0로 변환합니다. class-name의 값에 따라 이미지가 정상인지 비정상인지가 결정됩니다.

-metadata가 추가된 필드 이름으로 식별되는 상응하는 메타데이터가 있어야 합니다. 예를 들어 "anomaly-label-metadata"입니다.

예외 레이블-메타데이터

(필수) 레이블 속성에 대한 메타데이터 분류 정보가 들어 있습니다. 필드 이름은 -metadata가 추가된 레이블 속성과 동일해야 합니다.

confidence

(선택) Amazon Lookout for Vision이 사용할 수 없습니다. 값을 지정하는 경우 1의 값을 사용하십시오.

job-name

(선택 사항) 이미지를 처리하는 작업에 원하는 이름을 붙이세요.

class-name

(필수) 이미지에 일반 내용이 포함된 경우 `normal`을 지정하고, 그렇지 않으면 `anomaly`를 지정하십시오. `class-name`의 값이 다른 값인 경우 이미지는 레이블이 지정되지 않은 이미지로 데이터세트에 추가됩니다. 이미지에 레이블을 지정하려면 데이터 세트에 이미지 추가을 참조하십시오.

human-annotated

(필수) 사람이 주석을 완성했으면 `"yes"` 항목을 지정하세요. 아닌 경우에는 `"no"`로 지정합니다.

creation-date

(선택 사항) 레이블이 생성된 협정 세계시 (UTC) 날짜 및 시간입니다.

type

(필수) 이미지에 적용해야 하는 처리 유형입니다. `"groundtruth/image-classification"`값을 사용합니다.

anomaly-mask-ref

(필수) 마스크 이미지의 Amazon S3 위치입니다. 키 이름에는 `anomaly-mask-ref`를 사용하거나 원하는 키 이름을 사용하십시오. 키는 `-ref`로 끝나야 합니다. 마스크 이미지에는 각 예외 유형 `internal-color-map`에 대한 컬러 마스크가 포함되어야 합니다. 형식은 `"s3://`*`BUCKET`*`/`*`OBJECT_PATH`*`"`입니다. 가져온 데이터세트의 이미지는 동일한 Amazon S3 버킷에 저장되어야 합니다. 마스크 이미지는 이동식 네트워크 그래픽 (PNG) 형식 이미지여야 합니다.

anomaly-mask-ref-metadata

(필수) 이미지의 세분화 메타데이터. 키 이름에는 `anomaly-mask-ref-metadata`를 사용하거나 원하는 키 이름을 사용하십시오. 키 이름은 `-ref-metadata`로 끝나야 합니다.

internal-color-map

(필수) 개별 예외 유형에 매핑되는 색상 맵. 색상은 마스크 이미지의 색상과 일치해야 합니다 (`anomaly-mask-ref`).

키

(필수) 맵의 키. `0` 항목에는 이미지에서 예외 항목 이외의 영역을 나타내는 클래스 이름 BACKGROUND가 포함되어야 합니다.

class-name

(필수) 스크래치 또는 덴트와 같은 예외 유형의 이름입니다.

헥스 컬러

(필수) 예외 유형의 헥스 컬러 (예:`#2ca02c`). 색상은 `anomaly-mask-ref`의 색상과 일치해야 합니다. `BACKGROUND`예외 유형의 값은 항상 `#ffffff`입니다.

confidence

(필수) 현재 Amazon Lookout for Vision에서는 사용하지 않지만 부동 소수점 값이 필요합니다.

human-annotated

(필수) 사람이 주석을 완성했으면 `"yes"` 항목을 지정하세요. 아닌 경우에는 `"no"`로 지정합니다.

creation-date

(선택 사항) 세분화 정보가 생성된 협정 세계시 (UTC) 날짜 및 시간입니다.

type

(필수) 이미지에 적용해야 하는 처리 유형입니다. `"groundtruth/semantic-segmentation"`값을 사용합니다.

CSV 파일로 분류 매니페스트 파일 생성

이 예제 Python 스크립트는 CSV (쉼표로 구분된 값) 파일을 사용하여 이미지에 레이블을 지정함으로써 분류 매니페스트 파일을 간단하게 만들 수 있습니다. CSV 파일을 만들려면

매니페스트 파일은 모델 학습에 사용되는 이미지를 설명합니다. 매니페스트 파일은 하나 이상의 JSON 라인으로 구성됩니다. 각 JSON 라인은 단일 이미지를 설명합니다. 자세한 내용은 이미지 분류를 위한 JSON 라인 정의 단원을 참조하십시오.

CSV 파일은 텍스트 파일의 여러 행에 대한 표 형식 데이터를 나타냅니다. 행의 필드는 쉼표로 구분합니다. 자세한 내용은 comma separated values를 참조하세요. 이 스크립트의 경우 CSV 파일의 각 행에는 이미지의 S3 위치와 이미지의 예외 항목 분류 (`normal` 또는`anomaly`)가 포함됩니다. 각 행은 매니페스트 파일의 JSON 라인에 매핑됩니다.

예를 들어, 다음 CSV 파일은 예제 이미지의 일부 이미지를 설명합니다.

```
s3://s3bucket/circuitboard/train/anomaly/train-anomaly_1.jpg,anomaly
s3://s3bucket/circuitboard/train/anomaly/train-anomaly_10.jpg,anomaly
s3://s3bucket/circuitboard/train/anomaly/train-anomaly_11.jpg,anomaly
s3://s3bucket/circuitboard/train/normal/train-normal_1.jpg,normal
s3://s3bucket/circuitboard/train/normal/train-normal_10.jpg,normal
s3://s3bucket/circuitboard/train/normal/train-normal_11.jpg,normal
```

스크립트는 각 행에 대해 JSON 라인을 생성합니다. 예를 들어, 다음은 첫 번째 행 (`s3://s3bucket/circuitboard/train/anomaly/train-anomaly_1.jpg,anomaly`)의 JSON 라인입니다.

```
{"source-ref": "s3://s3bucket/csv_test/train_anomaly_1.jpg","anomaly-label":
 1,"anomaly-label-metadata": {"confidence": 1,"job-name": "labeling-job/anomaly-
classification","class-name": "anomaly","human-annotated": "yes","creation-date":
 "2022-02-04T22:47:07","type": "groundtruth/image-classification"}}
```

CSV 파일에 이미지의 Amazon S3 경로가 포함되어 있지 않은 경우 `--s3-path` 명령줄 인수를 사용하여 이미지에 대한 Amazon S3 경로를 지정하십시오.

매니페스트 파일을 생성하기 전에 스크립트는 CSV 파일의 중복 이미지와 `normal`또는 `anomaly` 이외의 이미지 분류가 있는지 확인합니다. 중복된 이미지 또는 이미지 분류 오류가 발견되면 스크립트는 다음을 수행합니다.

- 모든 이미지에 대한 첫 번째 유효한 이미지 항목을 중복 제거된 CSV 파일에 기록합니다.
- 오류 파일에 이미지가 중복된 경우를 기록합니다.
- 오류 파일에 `normal`나 `anomaly`가 없는 이미지 분류를 기록합니다.
- 매니페스트 파일을 생성하지 마세요.

오류 파일에는 입력 CSV 파일에서 중복 이미지 또는 분류 오류가 발견된 줄 번호가 포함됩니다. 오류 CSV 파일을 사용하여 입력 CSV 파일을 업데이트한 다음 스크립트를 다시 실행합니다. 또는 errors CSV 파일을 사용하여 이미지 분류 오류가 없는 고유한 이미지 항목과 이미지만 포함하는 중복 제거된 CSV 파일을 업데이트하십시오. 업데이트된 중복 제거된 CSV 파일을 사용하여 스크립트를 다시 실행합니다.

입력 CSV 파일에 중복이나 오류가 없는 경우 스크립트는 중복 제거된 이미지 CSV 파일과 오류 파일이 비어 있으므로 삭제합니다.

이 절차에서는 CSV 파일을 만들고 Python 스크립트를 실행하여 매니페스트 파일을 만듭니다. 이 스크립트는 Python 버전 3.7로 테스트되었습니다.

CSV 파일에서 매니페스트 파일을 생성하려면

1. 각 행에 다음 필드를 포함하는 CSV 파일을 생성합니다 (이미지당 한 행). CSV 파일에 헤더 행을 추가하지 마세요.

필드 1	필드 2
이미지 이름 또는 Amazon S3 이미지 경로 예를 들어 `s3://s3bucket/circuitboard/train/anomaly/train-anomaly_10.jpg` 입니다. Amazon S3 경로가 있는 이미지와 그렇지 않은 이미지를 혼합할 수는 없습니다.	이미지의 예외 항목 분류 (`normal`또는`anomaly`).

   예: `s3://s3bucket/circuitboard/train/anomaly/image_10.jpg,anomaly` 또는 `image_11.jpg,normal`
2. CSV 파일을 저장합니다.
3. 다음 Python 스크립트를 실행합니다. 다음 인수를 제공하세요.
   - `csv_file`: 1단계에서 생성한 CSV 파일
   - (선택 사항) `--s3-path` *`s3://path_to_folder/`* — 이미지 파일 이름에 추가할 Amazon S3 경로 (필드 1). 필드 1의 이미지에 아직 S3 경로가 포함되어 있지 않은 경우 `--s3-path` 항목을 사용합니다.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier:  Apache-2.0
"""
Purpose
Shows how to create an Amazon Lookout for Vision manifest file from a CSV file.
The CSV file format is image location,anomaly classification (normal or anomaly)
For example:
s3://s3bucket/circuitboard/train/anomaly/train_11.jpg,anomaly
s3://s3bucket/circuitboard/train/normal/train_1.jpg,normal
```

```
If necessary, use the bucket argument to specify the Amazon S3 bucket folder for
 the images.
"""

from datetime import datetime, timezone
import argparse
import logging
import csv
import os
import json

logger = logging.getLogger(__name__)


def check_errors(csv_file):
    """
    Checks for duplicate images and incorrect classifications in a CSV file.
    If duplicate images or invalid anomaly assignments are found, an errors CSV
 file
    and deduplicated CSV file are created. Only the first
    occurrence of a duplicate is recorded. Other duplicates are recorded in the
 errors file.
    :param csv_file: The source CSV file
    :return: True if errors or duplicates are found, otherwise false.
    """

    logger.info("Checking %s.", csv_file)

    errors_found = False
    errors_file = f"{os.path.splitext(csv_file)[0]}_errors.csv"
    deduplicated_file = f"{os.path.splitext(csv_file)[0]}_deduplicated.csv"

    with open(csv_file, 'r', encoding="UTF-8") as input_file,\
            open(deduplicated_file, 'w', encoding="UTF-8") as dedup,\
            open(errors_file, 'w', encoding="UTF-8") as errors:

        reader = csv.reader(input_file, delimiter=',')
        dedup_writer = csv.writer(dedup)
        error_writer = csv.writer(errors)
        line = 1
        entries = set()
        for row in reader:

            # Skip empty lines.
```

```
            if not ''.join(row).strip():
                continue

            # Record any incorrect classifications.
            if not row[1].lower() == "normal" and not row[1].lower() == "anomaly":
                error_writer.writerow(
                    [line, row[0], row[1], "INVALID_CLASSIFICATION"])
                errors_found = True

            # Write first image entry to dedup file and record duplicates.
            key = row[0]
            if key not in entries:
                dedup_writer.writerow(row)
                entries.add(key)
            else:
                error_writer.writerow([line, row[0], row[1], "DUPLICATE"])
                errors_found = True
            line += 1

    if errors_found:
        logger.info("Errors found check %s.", errors_file)
    else:
        os.remove(errors_file)
        os.remove(deduplicated_file)

    return errors_found


def create_manifest_file(csv_file, manifest_file, s3_path):
    """
    Read a CSV file and create an Amazon Lookout for Vision classification manifest
 file.
    :param csv_file: The source CSV file.
    :param manifest_file: The name of the manifest file to create.
    :param s3_path: The Amazon S3 path to the folder that contains the images.
    """
    logger.info("Processing CSV file %s.", csv_file)

    image_count = 0
    anomalous_count = 0

    with open(csv_file, newline='', encoding="UTF-8") as csvfile,\
        open(manifest_file, "w", encoding="UTF-8") as output_file:
```

```
        image_classifications = csv.reader(
            csvfile, delimiter=',', quotechar='|')

        # Process each row (image) in the CSV file.
        for row in image_classifications:
            # Skip empty lines.
            if not ''.join(row).strip():
                continue

            source_ref = str(s3_path) + row[0]
            classification = 0

            if row[1].lower() == 'anomaly':
                classification = 1
                anomalous_count += 1

           # Create the JSON line.
            json_line = {}
            json_line['source-ref'] = source_ref
            json_line['anomaly-label'] = str(classification)

            metadata = {}
            metadata['confidence'] = 1
            metadata['job-name'] = "labeling-job/anomaly-classification"
            metadata['class-name'] = row[1]
            metadata['human-annotated'] = "yes"
            metadata['creation-date'] = datetime.now(timezone.utc).strftime('%Y-%m-
%dT%H:%M:%S.%f')
            metadata['type'] = "groundtruth/image-classification"

            json_line['anomaly-label-metadata'] = metadata

            output_file.write(json.dumps(json_line))
            output_file.write('\n')
            image_count += 1

    logger.info("Finished creating manifest file %s.\n"
                "Images: %s\nAnomalous: %s",
                manifest_file,
                image_count,
                anomalous_count)
    return image_count, anomalous_count
```

```
def add_arguments(parser):
    """
    Add command line arguments to the parser.
    :param parser: The command line parser.
    """

    parser.add_argument(
        "csv_file", help="The CSV file that you want to process."
    )

    parser.add_argument(
        "--s3_path", help="The Amazon S3 bucket and folder path for the images."
        " If not supplied, column 1 is assumed to include the Amazon S3 path.",
 required=False
    )


def main():

    logging.basicConfig(level=logging.INFO,
                        format="%(levelname)s: %(message)s")

    try:

        # Get command line arguments.
        parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
        add_arguments(parser)
        args = parser.parse_args()
        s3_path = args.s3_path
        if s3_path is None:
            s3_path = ""

        csv_file = args.csv_file
        csv_file_no_extension = os.path.splitext(csv_file)[0]
        manifest_file = csv_file_no_extension + '.manifest'

        # Create manifest file if there are no duplicate images.
        if check_errors(csv_file):
            print(f"Issues found. Use {csv_file_no_extension}_errors.csv "\
                "to view duplicates and errors.")
            print(f"{csv_file}_deduplicated.csv contains the first"\
                "occurrence of a duplicate.\n"
                  "Update as necessary with the correct information.")
```

```
            print(f"Re-run the script with
 {csv_file_no_extension}_deduplicated.csv")
        else:
            print('No duplicates found. Creating manifest file.')

            image_count, anomalous_count = create_manifest_file(csv_file,
 manifest_file, s3_path)

            print(f"Finished creating manifest file: {manifest_file} \n")

            normal_count = image_count-anomalous_count
            print(f"Images processed: {image_count}")
            print(f"Normal: {normal_count}")
            print(f"Anomalous: {anomalous_count}")

    except FileNotFoundError as err:
        logger.exception("File not found.:%s", err)
        print(f"File not found: {err}. Check your input CSV file.")

if __name__ == "__main__":
    main()
```

4. 중복 이미지가 발생하거나 분류 오류가 발생하는 경우:

   a. 오류 파일을 사용하여 중복 제거된 CSV 파일 또는 입력 CSV 파일을 업데이트합니다.

   b. 업데이트된 중복 제거 CSV 파일 또는 업데이트된 입력 CSV 파일을 사용하여 스크립트를 다시 실행합니다.

5. 테스트 데이터세트를 사용하려는 경우 1~4단계를 반복하여 테스트 데이터 세트용 매니페스트 파일을 만드세요.

6. 필요한 경우 CSV 파일의 열 1에서 지정한(또는 `--s3-path` 명령줄에서 지정한) Amazon S3 버킷 경로에 이미지를 복사합니다. 이미지를 복사하려면 명령 프롬프트에서 다음 명령을 입력합니다.

```
aws s3 cp --recursive your-local-folder/ s3://your-target-S3-location/
```

7. 매니페스트 파일로 데이터세트 만들기 (콘솔)의 지침에 따라 데이터 세트를 생성합니다. AWS SDK를 사용하는 경우 섹션을 참조하세요매니페스트 파일 (SDK)로 데이터세트 만들기.

매니페스트 파일로 데이터세트 만들기 (콘솔)

다음 절차에서는 Amazon S3 버킷에 저장된 SageMaker AI 형식 매니페스트 파일을 가져와서 훈련 또는 테스트 데이터 세트를 생성하는 방법을 보여줍니다.

데이터세트를 생성한 후 데이터세트에 이미지를 더 추가하거나 이미지에 레이블을 추가할 수 있습니다. 자세한 내용은 데이터 세트에 이미지 추가 단원을 참조하십시오.

SageMaker AI Ground Truth 형식 매니페스트 파일을 사용하여 데이터 세트를 생성하려면(콘솔)

1. 기존 Amazon Lookout for Vision 호환 SageMaker AI Ground Truth 형식 매니페스트 파일을 생성하거나 사용합니다. 자세한 내용은 매니페스트 파일 만들기 단원을 참조하십시오.
2. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/s3/ Amazon S3 콘솔을 엽니다.
3. Amazon S3 버킷에서 매니페스트 파일을 보관할 폴더를 생성합니다.
4. 방금 생성한 폴더에 매니페스트 파일을 업로드합니다.
5. Amazon S3 버킷에 이미지를 저장할 폴더를 생성할 수도 있습니다.
6. 방금 생성한 폴더에 이미지를 업로드합니다.

   > **Important**
   >
   > 각 JSON 라인의 `source-ref` 필드 값은 폴더의 이미지에 매핑되어야 합니다.

7. https://console.aws.amazon.com/lookoutvision/에서 Amazon Lookout for Vision 콘솔을 엽니다.
8. Get started를 선택합니다.
9. 왼쪽 탐색 창에서 프로젝트를 선택합니다.
10. 매니페스트 파일에 추가할 프로젝트를 선택합니다.
11. 작동 방식 섹션에서 데이터세트 생성을 선택합니다.
12. 단일 데이터 세트 탭 또는 학습 및 테스트 데이터 세트 분리 탭을 선택하고 단계를 따르세요.

    Single dataset

    1. 단일 데이터 세트 만들기를 선택합니다.
    2. 이미지 원본 구성 섹션에서 SageMaker Ground Truth로 레이블이 지정된 이미지 가져오기를 선택합니다.
    3. .manifest 파일 위치의 경우 매니페스트 파일의 위치를 입력합니다.

Separate training and test datasets

1. 교육 데이터세트 및 테스트 데이터세트 만들기를 선택합니다.
2. 교육 데이터세트 세부 정보 섹션에서 SageMaker Ground Truth로 레이블이 지정된 이미지 가져오기를 선택합니다.
3. .manifest 파일 위치에 교육 매니페스트 파일의 위치를 입력합니다.
4. 테스트 데이터세트 세부정보 섹션에서 SageMaker Ground Truth로 레이블이 지정된 이미지 가져오기를 선택합니다.
5. .manifest 파일 위치에 테스트 매니페스트 파일의 위치를 입력합니다.

> **Note**
>
> 학습 데이터 세트와 테스트 데이터 세트에는 서로 다른 이미지 소스가 있을 수 있습니다.

13. 제출을 선택합니다.
14. 모델 학습에 나온 단계에 따라 모델을 훈련하세요.

Amazon Lookout for Vision은 Amazon S3 버킷 `datasets`폴더에 데이터세트를 생성합니다. 원본 `.manifest` 파일은 변경되지 않습니다.

매니페스트 파일 (SDK)로 데이터세트 만들기

CreateDataset 작업을 사용하여 Amazon Lookout for Vision 프로젝트와 연결된 데이터세트를 생성합니다.

학습 및 테스트에 단일 데이터세트를 사용하려면 `DatasetType`값이 `train`로 설정된 단일 데이터세트를 생성하십시오. 훈련 중에는 데이터세트를 내부적으로 분할하여 훈련 및 테스트 데이터 세트로 만듭니다. 데이터 세트를 생성할 수 없습니다. 별도의 테스트 데이터세트를 원하는 경우 `DatasetType` 값 세트`test`를 사용하여 `CreateDataset`를 다시 호출하세요. 훈련 중에는 훈련 및 테스트 데이터세트를 사용하여 모델을 훈련하고 테스트합니다.

선택적으로 `DatasetSource` 파라미터를 사용하여 데이터 세트를 채우는 데 사용되는 SageMaker AI Ground Truth 형식 매니페스트 파일의 위치를 지정할 수 있습니다. 이 경우 `CreateDataset`에 대한 호출은 비동기식입니다. 현재 상태를 확인하려면 `DescribeDataset` 항목을 호출하세요. 자세한 내

용은 데이터 세트 보기 단원을 참조하십시오. 가져오는 동안 검증 오류가 발생하는 경우 Status의 값은 CREATE_FAILED로 설정되고 상태 메시지 (StatusMessage) 가 설정됩니다.

> **Tip**
>
> 시작하기 예제 데이터셋으로 데이터세트를 만드는 경우 스크립트가 1단계: 매니페스트 파일 생성 및 이미지 업로드에 생성하는 매니페스트 파일 (getting-started/dataset-files/manifests/train.manifest)을 사용하세요.
> 회로판 예제 이미지로 데이터세트를 만드는 경우 다음 두 가지 옵션이 있습니다.
>
> 1. 코드를 사용하여 매니페스트 파일을 생성합니다. Amazon Lookout for Vision Lab Python 노트북은 회로 기판 예제 이미지에 대한 매니페스트 파일을 만드는 방법을 보여줍니다. 또는 코드 예제 리포지토리의 데이터 세트 예제 AWS 코드를 사용합니다.
> 2. 이미 Amazon Lookout for Vision 콘솔을 사용하여 회로판 예제 이미지가 포함된 데이터 세트를 생성한 경우 Amazon Lookout for Vision에서 생성한 매니페스트 파일을 다시 사용하십시오. 교육 및 테스트 매니페스트 파일 위치는 s3://*bucket*/datasets/*project name*/*train or test*/manifests/output/output.manifest과 같습니다.

DatasetSource를 지정하지 않으면 빈 데이터 세트가 생성됩니다. 이 경우 CreateDataset에 대한 호출은 동기식입니다. 나중에 UpdateDataSetEntries를 호출하여 데이터세트에 이미지에 레이블을 지정할 수 있습니다. 예제 코드는 더 많은 이미지 추가 (SDK) 항목을 참조하세요.

데이터세트를 바꾸려면 먼저 DeleteDataset으로 기존 데이터세트를 삭제한 다음, CreateDataset를 호출하여 동일한 데이터세트 유형의 새 데이터세트를 만드세요. 자세한 내용은 데이터 세트 삭제 단원을 참조하십시오.

데이터 집합을 만든 후 모델을 만들 수 있습니다. 자세한 내용은 모델 학습 (SDK) 단원을 참조하십시오.

ListDataSetEntries를 호출하여 데이터세트 내에서 레이블이 지정된 이미지 (JSON 라인)를 볼 수 있습니다. UpdateDatasetEntries를 호출하여 레이블이 있는 이미지를 추가할 수 있습니다.

테스트 및 훈련 데이터세트에 대한 정보를 보려면 데이터 세트 보기을 참조하십시오.

데이터세트 (SDK)를 만들려면

1. 아직 설치하지 않은 경우 및 AWS SDKs AWS CLI 를 설치하고 구성합니다. 자세한 내용은 4단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.

2. 다음 예제 코드를 사용하여 데이터 세트를 생성하세요.

CLI

다음 값을 변경합니다.

- `project-name`은 데이터세트와 연결하려는 프로젝트의 이름입니다.
- `dataset-type`은 만들려는 데이터세트 유형 (`train`또는`test`)
- `dataset-source`은 Amazon S3 매니페스트 파일의 위치입니다.
- `Bucket`은 매니페스트 파일을 포함할 Amazon S3 버킷의 이름입니다.
- `Key`은 Amazon S3 버킷에 있는 매니페스트 파일의 경로와 파일 이름입니다.

```
aws lookoutvision create-dataset --project-name project\
  --dataset-type train or test\
  --dataset-source '{ "GroundTruthManifest": { "S3Object": { "Bucket": "bucket",
 "Key": "manifest file" } } }' \
  --profile lookoutvision-access
```

Python

이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져온 것입니다. 전체 예제는 여기에서 확인하세요.

```
    @staticmethod
    def create_dataset(lookoutvision_client, project_name, manifest_file,
 dataset_type):
        """
        Creates a new Lookout for Vision dataset

        :param lookoutvision_client: A Lookout for Vision Boto3 client.
        :param project_name: The name of the project in which you want to
                             create a dataset.
        :param bucket: The bucket that contains the manifest file.
        :param manifest_file: The path and name of the manifest file.
        :param dataset_type: The type of the dataset (train or test).
        """
        try:
            bucket, key = manifest_file.replace("s3://", "").split("/", 1)
            logger.info("Creating %s dataset type...", dataset_type)
            dataset = {
```

```
                "GroundTruthManifest": {"S3Object": {"Bucket": bucket, "Key":
 key}}
            }
            response = lookoutvision_client.create_dataset(
                ProjectName=project_name,
                DatasetType=dataset_type,
                DatasetSource=dataset,
            )
            logger.info("Dataset Status: %s", response["DatasetMetadata"]
["Status"])
            logger.info(
                "Dataset Status Message: %s",
                response["DatasetMetadata"]["StatusMessage"],
            )
            logger.info("Dataset Type: %s", response["DatasetMetadata"]
["DatasetType"])

            # Wait until either created or failed.
            finished = False
            status = ""
            dataset_description = {}
            while finished is False:
                dataset_description = lookoutvision_client.describe_dataset(
                    ProjectName=project_name, DatasetType=dataset_type
                )
                status = dataset_description["DatasetDescription"]["Status"]

                if status == "CREATE_IN_PROGRESS":
                    logger.info("Dataset creation in progress...")
                    time.sleep(2)
                elif status == "CREATE_COMPLETE":
                    logger.info("Dataset created.")
                    finished = True
                else:
                    logger.info(
                        "Dataset creation failed: %s",
                        dataset_description["DatasetDescription"]
["StatusMessage"],
                    )
                    finished = True

            if status != "CREATE_COMPLETE":
                message = dataset_description["DatasetDescription"]
["StatusMessage"]
```

```
                logger.exception("Couldn't create dataset: %s", message)
                raise Exception(f"Couldn't create dataset: {message}")

        except ClientError:
            logger.exception("Service error: Couldn't create dataset.")
            raise
```

Java V2

이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져온 것입니다. 전체 예제는 여기에서 확인하세요.

```
/**
 * Creates an Amazon Lookout for Vision dataset from a manifest file.
 * Returns after Lookout for Vision creates the dataset.
 *
 * @param lfvClient    An Amazon Lookout for Vision client.
 * @param projectName  The name of the project in which you want to create a
 *                     dataset.
 * @param datasetType  The type of dataset that you want to create (train or
 *                     test).
 * @param bucket       The S3 bucket that contains the manifest file.
 * @param manifestFile The name and location of the manifest file within the S3
 *                     bucket.
 * @return DatasetDescription The description of the created dataset.
 */
public static DatasetDescription createDataset(LookoutVisionClient lfvClient,
                String projectName,
                String datasetType,
                String bucket,
                String manifestFile)
                throws LookoutVisionException, InterruptedException {

        logger.log(Level.INFO, "Creating {0} dataset for project {1}",
                        new Object[] { projectName, datasetType });

        // Build the request. If no bucket supplied, setup for empty dataset
 creation.
        CreateDatasetRequest createDatasetRequest = null;

        if (bucket != null && manifestFile != null) {
```

```
                InputS3Object s3Object = InputS3Object.builder()
                                .bucket(bucket)
                                .key(manifestFile)
                                .build();

                DatasetGroundTruthManifest groundTruthManifest =
DatasetGroundTruthManifest.builder()
                                .s3Object(s3Object)
                                .build();

                DatasetSource datasetSource = DatasetSource.builder()
                                .groundTruthManifest(groundTruthManifest)
                                .build();

                createDatasetRequest = CreateDatasetRequest.builder()
                                .projectName(projectName)
                                .datasetType(datasetType)
                                .datasetSource(datasetSource)
                                .build();
        } else {
                createDatasetRequest = CreateDatasetRequest.builder()
                                .projectName(projectName)
                                .datasetType(datasetType)
                                .build();
        }

        lfvClient.createDataset(createDatasetRequest);

        DatasetDescription datasetDescription = null;

        boolean finished = false;

        // Wait until dataset is created, or failure occurs.
        while (!finished) {

                datasetDescription = describeDataset(lfvClient, projectName,
datasetType);

                switch (datasetDescription.status()) {
                        case CREATE_COMPLETE:
                                logger.log(Level.INFO, "{0}dataset created for
project {1}",
                                                new Object[] { datasetType,
projectName });
```

```
                                finished = true;
                                break;
                        case CREATE_IN_PROGRESS:
                                logger.log(Level.INFO, "{0} dataset creating for
project {1}",
                                                new Object[] { datasetType,
projectName });

                                TimeUnit.SECONDS.sleep(5);

                                break;

                        case CREATE_FAILED:
                                logger.log(Level.SEVERE,
                                                "{0} dataset creation failed for
project {1}. Error {2}",
                                                new Object[] { datasetType,
projectName,

datasetDescription.statusAsString() });
                                finished = true;
                                break;
                        default:
                                logger.log(Level.SEVERE, "{0} error when
creating {1} dataset for project {2}",
                                                new Object[] { datasetType,
projectName,

datasetDescription.statusAsString() });
                                finished = true;
                                break;

                }
        }

        logger.log(Level.INFO, "Dataset info. Status: {0}\n Message: {1} }",
                        new Object[] { datasetDescription.statusAsString(),
                                        datasetDescription.statusMessage() });

        return datasetDescription;

}
```

3. 모델 학습 (SDK)의 단계에 따라 모델을 학습시키십시오.

이미지 레이블 지정

Amazon Lookout for Vision 콘솔을 사용하여 데이터세트의 이미지에 할당된 라벨을 추가하거나 수정할 수 있습니다. SDK를 사용하는 경우 레이블은 CreateDataset에 제공하는 매니페스트 파일의 일부입니다. UpdateDataSetEntries를 호출하여 이미지의 라벨을 업데이트할 수 있습니다. 예제 코드는 더 많은 이미지 추가 (SDK) 항목을 참조하세요.

모델 유형 선택

이미지에 지정하는 레이블에 따라 Lookout for Vision이 생성하는 모델 유형이 결정됩니다. 프로젝트에 별도의 테스트 데이터세트가 있는 경우 같은 방식으로 이미지에 라벨을 지정하세요.

이미지 분류 모델

이미지 분류 모델을 만들려면 각 이미지를 정상 또는 비정상 이미지로 분류합니다. 각 이미지에 대해 이미지 분류 (콘솔)을 수행하십시오.

이미지 세분화 모델

이미지 세분화 모델을 만들려면 각 이미지를 정상 또는 변칙 이미지로 분류합니다. 또한 각 변칙 이미지에 대해 영상의 각 변칙 영역에 대한 픽셀 마스크와 픽셀 마스크 내의 예외 항목 유형에 대한 예외 항목 레이블을 지정합니다. 예를 들어, 다음 이미지의 파란색 마스크는 자동차에 있는 스크래치 이상 유형의 위치를 표시합니다. 이미지에 두 가지 이상의 예외 항목 레이블을 지정할 수 있습니다. 각 이미지에 대해 이미지 세분화 (콘솔)을 수행하십시오.

이미지 분류 (콘솔)

Lookout for Vision 콘솔을 사용하여 데이터 세트의 이미지를 정상 또는 예외 이미지로 분류합니다. 분류되지 않은 이미지는 모델을 학습시키는 데 사용되지 않습니다.

이미지 분할 모델을 만드는 경우에는 이 절차를 건너뛰고 이미지 분류 단계가 포함된 이미지 세분화 (콘솔) 절차를 수행하세요.

> **Note**
>
> 방금 데이터 세트를 생성합니다.을 완료했다면 콘솔에 현재 모델 대시보드가 표시되므로 1~4 단계를 수행할 필요가 없습니다.

이미지 분류하기 (콘솔)

1. https://console.aws.amazon.com/lookoutvision/에서 Amazon Lookout for Vision 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 프로젝트를 선택합니다.
3. 프로젝트 페이지에서 사용할 프로젝트를 선택합니다.
4. 프로젝트의 왼쪽 탐색 창에서 데이터 세트를 선택합니다.
5. 학습 데이터세트와 테스트 데이터세트가 분리되어 있는 경우 사용하려는 데이터세트의 탭을 선택하세요.
6. 라벨링 시작을 선택합니다.
7. 이 페이지에 있는 모든 이미지 선택을 선택합니다.
8. 이미지가 정상이면 정상으로 분류를 선택하고 그렇지 않으면 예외 항목으로 분류를 선택합니다. 각 사진 아래에 레이블이 표시됩니다.
9. 이미지의 라벨을 변경해야 하는 경우 다음과 같이 하세요:
   a. 이미지 아래에서 [예외 항목] 또는 [정상]을 선택합니다.
   b. 이미지의 올바른 라벨을 결정할 수 없는 경우 갤러리에서 이미지를 선택하여 이미지를 확대하세요.

   > **Note**
   >
   > 필터 섹션에서 원하는 레이블 또는 레이블 상태를 선택하여 이미지 레이블을 필터링할 수 있습니다.

10. 데이터 세트의 모든 이미지에 레이블이 올바르게 지정될 때까지 필요에 따라 각 페이지에서 7~9 단계를 반복합니다.
11. 변경 사항 저장을 선택합니다.

12. 이미지 레이블링을 마쳤으면 모델을 학습시킬 수 있습니다.

이미지 세분화 (콘솔)

이미지 분할 모델을 만드는 경우 이미지를 정상 또는 이상 영상으로 분류해야 합니다. 또한 변칙 이미지에 분할 정보를 추가해야 합니다. 분할 정보를 지정하려면 먼저 모형이 찾고자 하는 각 유형의 예외 항목 (예: 찌그러짐, 긁힘) 에 대해 예외 항목 레이블을 지정합니다. 그런 다음 데이터 세트의 변칙 이미지에 있는 각 예외 항목에 대해 예외 항목 마스크와 예외 항목 레이블을 지정합니다.

> **Note**
>
> 이미지 분류 모델을 생성하는 경우 이미지를 분할하거나 예외 항목 레이블을 지정할 필요가 없습니다.

주제

- 이상 레이블 지정
- 이미지 레이블 지정
- 주석 도구를 사용하여 이미지 세분화하기

이상 레이블 지정

데이터세트 이미지에 있는 각 유형의 예외 항목에 대해 예외 항목 레이블을 정의합니다.

이상 레이블 지정

1. https://console.aws.amazon.com/lookoutvision/에서 Amazon Lookout for Vision 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 프로젝트를 선택합니다.
3. 프로젝트 페이지에서 사용할 프로젝트를 선택합니다.
4. 프로젝트의 왼쪽 탐색 창에서 데이터 세트를 선택합니다.
5. 예외 항목 레이블에서 예외 항목 레이블 추가를 선택합니다. 이전에 예외 항목 레이블을 추가한 경우 관리를 선택합니다.
6. 대화 상자에서 다음을 수행합니다.

    a. 추가하려는 예외 항목 레이블을 입력하고 예외 항목 레이블 추가를 선택합니다.

b. 모델에서 찾으려는 모든 예외 항목 레이블을 입력할 때까지 이전 단계를 반복합니다.

c. (선택 사항) 편집 아이콘을 선택하여 레이블 이름을 변경합니다.

d. (선택 사항) 새 예외 항목 레이블을 삭제하려면 삭제 아이콘을 선택합니다. 데이터세트에서 현재 사용 중인 예외 유형은 삭제할 수 없습니다.

7. 확인을 선택하여 데이터세트에 새 예외 항목 레이블을 추가합니다.

예외 항목 레이블을 지정한 후에는 이미지 레이블 지정 작업을 수행하여 이미지에 레이블을 지정합니다.

이미지 레이블 지정

이미지 세분화를 위해 이미지에 레이블을 지정하려면 이미지를 정상 또는 이상 영상으로 분류하십시오. 그런 다음 주석 도구를 사용하여 이미지에 나타나는 각 유형의 예외 영역을 촘촘하게 덮는 마스크를 그려 이미지를 분할합니다.

이미지에 레이블을 지정하려면

1. 학습 데이터세트와 테스트 데이터세트가 분리되어 있는 경우 사용하려는 데이터세트의 탭을 선택하세요.
2. 아직 지정하지 않았다면 이상 레이블 지정을 수행하여 데이터세트의 예외 유형을 지정합니다.
3. 라벨링 시작을 선택합니다.
4. 이 페이지에 있는 모든 이미지 선택을 선택합니다.
5. 이미지가 정상이면 정상으로 분류를 선택하고 그렇지 않으면 예외 항목으로 분류를 선택합니다.
6. 단일 이미지의 레이블을 변경하려면 이미지 아래에서 정상 또는 예외을 선택합니다.

> **Note**
>
> 필터 섹션에서 원하는 레이블 또는 레이블 상태를 선택하여 이미지 레이블을 필터링할 수 있습니다. 정렬 옵션 섹션에서 신뢰도 점수를 기준으로 정렬할 수 있습니다.

7. 각 변칙 이미지에 대해 이미지를 선택하여 주석 도구를 엽니다. 주석 도구를 사용하여 이미지 세분화하기를 하면 세그멘테이션 정보를 추가할 수 있습니다.
8. 변경 사항 저장을 선택합니다.
9. 이미지 레이블링을 마쳤으면 모델을 학습시킬 수 있습니다.

주석 도구를 사용하여 이미지 세분화하기

주석 도구를 사용하면 마스크로 이상 영역을 표시하여 이미지를 세분화할 수 있습니다.

주석 도구를 사용하여 이미지를 세분화하려면

1. 데이터세트 갤러리에서 이미지를 선택하여 주석 도구를 엽니다. 필요한 경우 레이블 지정 시작을 선택하여 레이블 지정 모드로 전환합니다.
2. 예외 항목 레이블 섹션에서 표시하려는 예외 항목 레이블을 선택합니다. 필요한 경우 예외 항목 레이블 추가를 선택하여 새 예외 항목 레이블을 추가합니다.
3. 페이지 하단에 있는 그리기 도구를 선택하고 예외 항목 레이블의 변칙 영역을 촘촘하게 덮는 마스크를 그립니다. 다음 이미지는 예외 항목을 촘촘하게 가리는 마스크의 예입니다.

다음은 이상 현상을 제대로 가리지 못하는 불량 마스크의 예제입니다.

4. 분할할 이미지가 더 많으면 [다음] 을 선택하고 2단계와 3단계를 반복합니다.
5. 제출 및 닫기를 선택하여 이미지 세분화를 완료합니다.

모델 학습

데이터세트를 만들고 이미지에 레이블을 지정한 후 모델을 학습시킬 수 있습니다. 학습 프로세스의 일환으로 테스트 데이터세트가 사용됩니다. 단일 데이터 세트 프로젝트가 있는 경우 훈련 프로세스의 일환으로 데이터세트의 이미지가 테스트 데이터세트와 훈련 데이터 세트로 자동 분할됩니다. 프로젝트에 훈련 데이터 세트와 테스트 데이터 세트가 있는 경우 이를 사용하여 데이터 세트를 별도로 훈련하고 테스트합니다.

학습이 완료되면 모델의 성능을 평가하고 필요한 부분을 개선할 수 있습니다. 자세한 내용은 Amazon Lookout for Vision 모델 개선 단원을 참조하십시오.

모델을 학습시키기 위해 Amazon Lookout for Vision은 소스 교육 및 테스트 이미지의 사본을 만듭니다. 기본적으로 복사된 이미지는 AWS가 소유하고 관리하는 키로 암호화됩니다. 자체 AWS KMS (AWS Management Service) 키를 선택하여 사용할 수도 있습니다. 자세한 내용은 AWS Key Management Service 개념을 참조하십시오. 소스 이미지는 영향을 받지 않습니다.

리소스에 태그 형태로 메타데이터를 지정할 수 있습니다. 자세한 내용은 모델 태깅 단원을 참조하십시오.

모델을 학습시킬 때마다 모델의 새 버전이 생성됩니다. 모델의 버전이 더 이상 필요하지 않으면 삭제할 수 있습니다. 자세한 내용은 모델 삭제 단원을 참조하십시오.

모델을 성공적으로 훈련하는 데 걸리는 시간만큼 요금이 부과됩니다. 자세한 정보는 훈련 시간 섹션을 참조하세요.

프로젝트의 기존 모델을 보려면모델 보기.

> **Note**
>
> 방금 데이터 세트를 생성합니다. 데이터 세트에 이미지 추가 완료했거나 현재 콘솔에 모델 대시보드가 표시되므로 1~4단계를 수행할 필요가 없습니다.

주제

- 모델 교육 (콘솔)
- 모델 학습 (SDK)

모델 교육 (콘솔)

다음 절차에서는 콘솔을 사용하여 모델을 학습하는 방법을 보여 줍니다.

모델 학습하기 (콘솔)

1. https://console.aws.amazon.com/lookoutvision/에서 Amazon Lookout for Vision 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 프로젝트를 선택합니다.
3. 프로젝트 페이지에서 학습시키려는 모델이 포함된 프로젝트를 선택합니다.
4. 프로젝트 세부 정보 페이지에서 모델 학습을 선택합니다. 모델 학습에 필요한 레이블이 지정된 이미지가 충분하면 모델 학습 버튼을 사용할 수 있습니다. 버튼을 사용할 수 없는 경우 레이블이 지정된 이미지가 충분히 나올 때까지 이미지를 더 추가하세요.
5. (선택 사항) 자체 AWS KMS 암호화 키를 사용하려면 다음을 수행하십시오.
   a. 이미지 데이터 암호화에서 암호화 설정 사용자 지정 (고급)을 선택합니다.
   b. 암호화.aws_kms_key에서 키의 Amazon Resource Name (ARN)을 입력하거나 기존 AWS KMS 키를 선택합니다. 새 키를 생성하려면 AWS IMS 키 생성을 선택합니다.
6. (선택 사항) 태그를 모델에 추가하려면 다음을 수행합니다.

a. 태그 항목에서 새로운 태그 추가를 선택합니다.

b. 다음을 입력합니다.

   i. 키에 있는 키의 이름.

   ii. 값에 있는 키의 값

c. 태그를 더 추가하려면 6a단계와 6b단계를 반복합니다.

d. (선택 사항) 태그를 제거하려면 제거할 태그 옆의 제거를 선택합니다. 이전에 저장한 태그를 제거하는 경우 변경 내용을 저장하면 해당 태그가 제거됩니다.

7. 모델 학습을 선택합니다.
8. 모델을 훈련하고 싶으신가요? 대화 상자에서 모델 훈련을 선택합니다.
9. 모델 보기에서 학습이 시작된 것을 확인할 수 있으며 모델 버전의 Status 열을 보면 현재 상태를 확인할 수 있습니다. 모델 훈련은 완료하는 데 다소 시간이 걸립니다.
10. 학습이 끝나면 성능을 평가할 수 있습니다. 자세한 내용은 Amazon Lookout for Vision 모델 개선 단원을 참조하십시오.

모델 학습 (SDK)

CreateModel 작업을 사용하여 모델의 학습, 테스트 및 평가를 시작합니다. Amazon Lookout for Vision은 프로젝트와 관련된 교육 및 테스트 데이터 세트를 사용하여 모델을 학습합니다. 자세한 내용은 프로젝트 생성(SDK) 단원을 참조하십시오.

CreateModel을 호출할 때마다 모델의 새 버전이 생성됩니다. CreateModel의 응답에는 모델 버전이 포함됩니다.

모델 훈련을 성공적으로 완료할 때 비용이 청구됩니다. ClientToken 입력 파라미터를 사용하면 사용자의 불필요하거나 우발적인 모델 훈련 반복으로 인한 요금을 방지할 수 있습니다. ClientToken는 가 특정 파라미터 세트에 대해 한 CreateModel 번만 완료되도록 하는 멱등성 입력 파라미터입니다. ClientToken 값이 동일한 CreateModel에 대한 반복 호출은 훈련이 반복되지 않도록 합니다. ClientToken값을 제공하지 않으면 사용 중인 AWS SDK가 값을 삽입합니다. 이렇게 하면 네트워크 오류가 발생한 후 재시도하여 여러 교육 작업을 시작하지 못하게 되지만, 사용 사례에 맞게 값을 직접 입력해야 합니다. 자세한 내용은 CreateModel 항목을 참조하십시오.

학습을 완료하는 데 시간이 좀 걸립니다. 현재 상태를 확인하려면 DescribeModel을 호출하고 (호출에서 CreateProject로 지정) 프로젝트 이름과 모델 버전을 전달하십시오. status필드는 모델 학습의 현재 상태를 나타냅니다. 예제 코드는 모델 보기 (SDK) 항목을 참조하세요.

학습이 성공적이면 모델을 평가할 수 있습니다. 자세한 내용은 Amazon Lookout for Vision 모델 개선 단원을 참조하십시오.

프로젝트에서 만든 모델을 보려면 `ListModels`를 호출하십시오. 예제 코드는 모델 보기 항목을 참조하세요.

모델 (SDK) 학습시키기

1. 아직 설치하지 않은 경우 및 AWS SDKs AWS CLI 를 설치하고 구성합니다. 자세한 내용은 4단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.
2. 다음 예제 코드를 사용하여 모델을 학습시키십시오.

   CLI

   다음 값을 변경합니다.

   - `project-name`은 생성하려는 모델이 포함된 프로젝트 이름입니다.
   - `output-config`은 학습 결과를 저장하려는 위치입니다. 다음 값을 교체합니다.
     - `output bucket`은 Amazon Lookout for Vision이 훈련 결과를 저장하는 Amazon S3 버킷의 이름으로 표시됩니다.
     - `output folder`은 학습 결과를 저장할 폴더의 이름과 함께 표시됩니다.
     - `Key`은 태그의 키 이름입니다.
     - `Value`은 `tag_key`와 연관시킨 값입니다.

   ```
   aws lookoutvision create-model --project-name "project name"\
     --output-config '{ "S3Location": { "Bucket": "output bucket", "Prefix":
    "output folder" } }'\
     --tags '[{"Key":"Key","Value":"Value"}]' \
     --profile lookoutvision-access
   ```

   Python

   이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져온 것입니다. 전체 예제는 여기에서 확인하세요.

   ```
       @staticmethod
       def create_model(
           lookoutvision_client,
   ```

```
        project_name,
        training_results,
        tag_key=None,
        tag_key_value=None,
    ):
        """
        Creates a version of a Lookout for Vision model.

        :param lookoutvision_client: A Boto3 Lookout for Vision client.
        :param project_name: The name of the project in which you want to create
a
                             model.
        :param training_results: The Amazon S3 location where training results
are stored.
        :param tag_key: The key for a tag to add to the model.
        :param tag_key_value - A value associated with the tag_key.
        return: The model status and version.
        """
        try:
            logger.info("Training model...")
            output_bucket, output_folder = training_results.replace("s3://",
"").split(
                "/", 1
            )
            output_config = {
                "S3Location": {"Bucket": output_bucket, "Prefix": output_folder}
            }
            tags = []
            if tag_key is not None:
                tags = [{"Key": tag_key, "Value": tag_key_value}]

            response = lookoutvision_client.create_model(
                ProjectName=project_name, OutputConfig=output_config, Tags=tags
            )

            logger.info("ARN: %s", response["ModelMetadata"]["ModelArn"])
            logger.info("Version: %s", response["ModelMetadata"]
["ModelVersion"])
            logger.info("Started training...")

            print("Training started. Training might take several hours to
complete.")

            # Wait until training completes.
```

```
            finished = False
            status = "UNKNOWN"
            while finished is False:
                model_description = lookoutvision_client.describe_model(
                    ProjectName=project_name,
                    ModelVersion=response["ModelMetadata"]["ModelVersion"],
                )
                status = model_description["ModelDescription"]["Status"]

                if status == "TRAINING":
                    logger.info("Model training in progress...")
                    time.sleep(600)
                    continue

                if status == "TRAINED":
                    logger.info("Model was successfully trained.")
                else:
                    logger.info(
                        "Model training failed: %s ",
                        model_description["ModelDescription"]["StatusMessage"],
                    )
                finished = True
        except ClientError:
            logger.exception("Couldn't train model.")
            raise
        else:
            return status, response["ModelMetadata"]["ModelVersion"]
```

Java V2

이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져온 것입니다. 전체 예제는 여기에서 확인하세요.

```
/**
 * Creates an Amazon Lookout for Vision model. The function returns after model
 * training completes. Model training can take multiple hours to complete.
 * You are charged for the amount of time it takes to successfully train a
 model.
 * Returns after Lookout for Vision creates the dataset.
 *
 * @param lfvClient   An Amazon Lookout for Vision client.
 * @param projectName The name of the project in which you want to create a
```

```
     *                    model.
     * @param description A description for the model.
     * @param bucket      The S3 bucket in which Lookout for Vision stores the
     *                    training results.
     * @param folder      The location of the training results within the S3
     *                    bucket.
     * @return ModelDescription The description of the created model.
     */
    public static ModelDescription createModel(LookoutVisionClient lfvClient, String
     projectName,
                    String description, String bucket, String folder)
                    throws LookoutVisionException, InterruptedException {

            logger.log(Level.INFO, "Creating model for project: {0}.", new Object[]
     { projectName });

            // Setup input parameters.
            S3Location s3Location = S3Location.builder()
                            .bucket(bucket)
                            .prefix(folder)
                            .build();

            OutputConfig config = OutputConfig.builder()
                            .s3Location(s3Location)
                            .build();

            CreateModelRequest createModelRequest = CreateModelRequest.builder()
                            .projectName(projectName)
                            .description(description)
                            .outputConfig(config)
                            .build();

            // Create and train the model.
            CreateModelResponse response =
     lfvClient.createModel(createModelRequest);

            String modelVersion = response.modelMetadata().modelVersion();
            boolean finished = false;
            DescribeModelResponse descriptionResponse = null;

            // Wait until training finishes or fails.

            do {
```

```
                DescribeModelRequest describeModelRequest =
DescribeModelRequest.builder()
                                .projectName(projectName)
                                .modelVersion(modelVersion)
                                .build();

                descriptionResponse =
lfvClient.describeModel(describeModelRequest);

                switch (descriptionResponse.modelDescription().status()) {
                        case TRAINED:
                                logger.log(Level.INFO, "Model training completed
for project {0} version {1}.",
                                                new Object[] { projectName,
modelVersion });
                                finished = true;
                                break;

                        case TRAINING:
                                logger.log(Level.INFO,
                                                "Model training in progress for
project {0} version {1}.",
                                                new Object[] { projectName,
modelVersion });
                                TimeUnit.SECONDS.sleep(60);

                                break;

                        case TRAINING_FAILED:
                                logger.log(Level.SEVERE,
                                                "Model training failed for for
project {0} version {1}.",
                                                new Object[] { projectName,
modelVersion });
                                finished = true;
                                break;

                        default:
                                logger.log(Level.SEVERE,
                                                "Unexpected error when training
model project {0} version {1}: {2}.",
                                                new Object[] { projectName,
modelVersion,
```

```
descriptionResponse.modelDescription()

.status() });
                                finished = true;
                                break;

                }
        } while (!finished);

        return descriptionResponse.modelDescription();

}
```

3. 학습이 끝나면 성능을 평가할 수 있습니다. 자세한 내용은 Amazon Lookout for Vision 모델 개선 단원을 참조하십시오.

모델 학습 문제 해결

매니페스트 파일 또는 학습 이미지에 문제가 있으면 모델 학습이 실패할 수 있습니다. 모델을 재학습시키기 전에 다음과 같은 잠재적 문제를 확인하세요.

예외 항목 레이블 색상이 마스크 이미지의 예외 항목 색상과 일치하지 않습니다.

이미지 분할 모델을 훈련시키는 경우 매니페스트 파일의 예외 항목 레이블 색상이 마스크 이미지의 색상과 일치해야 합니다. 매니페스트 파일에 있는 이미지의 JSON 줄에는 Amazon Lookout for Vision에 어떤 색상이 예외 레이블에 해당하는지 알려주는 메타데이터 (`internal-color-map`) 가 있습니다. 예를 들어, 다음 JSON 줄에 있는 `scratch` 예외 항목 레이블의 색상은 `#2ca02c`입니다.

```
{
    "source-ref": "s3://path-to-image",
    "anomaly-label": 1,
    "anomaly-label-metadata": {
        "class-name": "anomaly",
        "creation-date": "2021-10-12T14:16:45.668",
        "human-annotated": "yes",
        "job-name": "labeling-job/classification-job",
        "type": "groundtruth/image-classification",
        "confidence": 1
```

```
    },
    "anomaly-mask-ref": "s3://path-to-image",
    "anomaly-mask-ref-metadata": {
        "internal-color-map": {
            "0": {
                "class-name": "BACKGROUND",
                "hex-color": "#ffffff",
                "confidence": 0.0
            },
            "1": {
                "class-name": "scratch",
                "hex-color": "#2ca02c",
                "confidence": 0.0
            },
            "2": {
                "class-name": "dent",
                "hex-color": "#1f77b4",
                "confidence": 0.0
            }
        },
        "type": "groundtruth/semantic-segmentation",
        "human-annotated": "yes",
        "creation-date": "2021-11-23T20:31:57.758889",
        "job-name": "labeling-job/segmentation-job"
    }
}
```

마스크 이미지의 색상이 해당 hex-color값과 일치하지 않으면 학습이 실패하므로 매니페스트 파일을 업데이트해야 합니다.

매니페스트 파일의 색상 값을 업데이트하려면

1. 텍스트 편집기를 사용하여 데이터세트를 만드는 데 사용한 매니페스트 파일을 엽니다.
2. 각 JSON 라인 (이미지) 에 대해 internal-color-map 필드 내 색상 (hex-color)이 마스크 이미지의 예외 항목 레이블 색상과 일치하는지 확인합니다.

   anomaly-mask-ref필드에서 마스크 이미지의 위치를 가져올 수 있습니다. 이미지를 컴퓨터에 다운로드하고 다음 코드를 사용하여 이미지의 색상을 가져옵니다.

```
from PIL import Image
```

```
img = Image.open('path to local copy of mask file')
colors = img.convert('RGB').getcolors() #this converts the mode to RGB
for color in colors:
    print('#%02x%02x%02x' % color[1])
```

3. 색상이 잘못 지정된 각 이미지에 대해 이미지의 JSON 라인에서 `hex-color` 필드를 업데이트하십시오.
4. 매니페스트 파일을 저장합니다.
5. 프로젝트에서 기존 데이터세트를 삭제합니다.
6. 업데이트된 매니페스트 파일을 사용하여 프로젝트에 새 데이터세트를 생성합니다.
7. 모델을 학습시키세요.

또는 5단계와 6단계에서는 UpdateDataSetEntries 작업을 호출하고 업데이트하려는 이미지에 대해 업데이트된 JSON 라인을 제공하여 데이터세트의 개별 이미지를 업데이트할 수 있습니다. 예제 코드는 더 많은 이미지 추가 (SDK) 항목을 참조하세요.

마스크 이미지는 PNG 형식이 아닙니다.

이미지 분할 모델을 훈련시키는 경우 마스크 이미지는 PNG 형식이어야 합니다. 매니페스트 파일에서 데이터 세트를 만드는 경우 *anomaly-mask*-ref에서 참조하는 마스크 이미지가 PNG 형식인지 확인하세요. 마스크 이미지가 PNG 형식이 아닌 경우 PNG 형식으로 변환해야 합니다. 이미지 파일의 확장명 이름을 .png로 바꾸는 것만으로는 충분하지 않습니다.

Amazon Lookout for Vision 콘솔에서 또는 SageMaker AI Ground Truth 작업을 사용하여 생성한 마스크 이미지는 PNG 형식으로 생성됩니다. 이러한 설정을 변경할 필요는 없습니다.

매니페스트 파일에서 PNG가 아닌 형식의 마스크 이미지를 수정하려면

1. 텍스트 편집기를 사용하여 데이터세트를 만드는 데 사용한 매니페스트 파일을 엽니다.
2. 각 JSON 라인 (이미지)에 대해 이미지가 PNG 형식 이미지를 *anomaly-mask*-ref 참조하는지 확인하세요. 자세한 내용은 매니페스트 파일 만들기 단원을 참조하십시오.
3. 매니페스트 파일을 선택합니다.
4. 프로젝트에서 기존 데이터세트를 삭제합니다.
5. 업데이트된 매니페스트 파일을 사용하여 프로젝트에 새 데이터세트를 생성합니다.
6. 모델을 학습시키세요.

세분화 또는 분류 레이블이 부정확하거나 누락되었습니다.

레이블이 누락되거나 부정확하면 훈련이 실패하거나 성능이 떨어지는 모델을 만들 수 있습니다. 데이터세트의 모든 이미지에 레이블을 지정하는 것이 좋습니다. 모든 이미지에 레이블을 지정하지 않아 모델 학습에 실패하거나 모델 성능이 저조한 경우 이미지를 더 추가하세요.

다음을 확인하세요.

- 분할 모델을 만드는 경우 마스크가 데이터셋 이미지의 이상 징후를 철저하게 가려야 합니다. 데이터세트의 마스크를 확인하려면 프로젝트의 데이터세트 갤러리에서 이미지를 확인하세요. 필요한 경우 이미지 마스크를 다시 그리세요. 자세한 내용은 이미지 세분화 (콘솔) 단원을 참조하십시오.

- 데이터 세트 이미지의 변칙 이미지가 분류되었는지 확인하세요. 이미지 분할 모델을 만드는 경우 변칙 이미지에 예외 항목 레이블과 이미지 마스크가 있는지 확인하세요.

  어떤 유형의 모델 (세분화 또는 분류) 을 만들고 있는지 기억하는 것이 중요합니다. 분류 모델에는 변칙 이미지에 대한 이미지 마스크가 필요하지 않습니다. 분류 모델용 데이터세트 이미지에는 마스크를 추가하지 마세요.

  누락된 라벨을 업데이트하려면

  1. 프로젝트의 데이터세트 갤러리를 엽니다.
  2. 라벨이 없는 이미지를 필터링하여 라벨이 없는 이미지를 확인할 수 있습니다.
  3. 다음 중 하나를 수행합니다.
     - 이미지 분류 모델을 만드는 경우 레이블이 지정되지 않은 각 이미지를 분류하십시오.
     - 이미지 분할 모델을 만드는 경우 레이블이 지정되지 않은 각 이미지를 분류하고 세분화하십시오.
  4. 이미지 분할 모델을 만드는 경우 마스크가 없는 분류된 모든 변칙 영상에 마스크를 추가하십시오.
  5. 모델을 학습시키세요.

불량하거나 누락된 레이블을 수정하지 않으려면 레이블이 지정된 이미지를 더 추가하거나 데이터세트에서 영향을 받는 이미지를 제거하는 것이 좋습니다. 이 작업은 콘솔 또는 UpdateDatasetEntries 작업을 사용하여 수행할 수 있습니다. 자세한 내용은 데이터 세트에 이미지 추가 단원을 참조하십시오.

이미지를 제거하기로 선택한 경우 데이터세트에서 이미지를 삭제할 수 없으므로 영향을 받은 이미지가 없는 데이터세트를 다시 만들어야 합니다. 자세한 내용은 데이터 세트에서 이미지 삭제하기 단원을 참조하십시오.

Amazon Lookout for Vision 모델 개선

학습 중에 Lookout for Vision은 테스트 데이터 세트로 모델을 테스트하고 그 결과를 사용하여 성능 지표를 생성합니다. 성능 지표를 사용하여 모델의 성능을 평가할 수 있습니다. 필요한 경우 데이터 세트를 개선하기 위한 조치를 취한 다음 모델을 재학습할 수 있습니다.

모델의 성능이 만족스러우면 사용을 시작할 수 있습니다. 자세한 내용은 훈련된 Amazon Lookout for Vision 모델 실행 단원을 참조하십시오.

주제

- 1단계: 모델의 성능 평가
- 2단계: 모델 개선
- 성능 지표 보기
- 평가판 확인 작업을 통한 모델 검증

1단계: 모델의 성능 평가

콘솔과 DescribeModel 작업에서 성능 지표에 액세스할 수 있습니다. Amazon Lookout for Vision은 테스트 데이터 세트에 대한 요약 성능 지표와 모든 개별 이미지에 대한 예측 결과를 제공합니다. 모델이 분할 모델인 경우 콘솔에는 각 예외 항목 레이블에 대한 요약 지표도 표시됩니다.

콘솔에서 성능 지표와 테스트 이미지 예측을 보려면 성능 지표 보기 (콘솔)을 참조하십시오. `DescribeModel` 작업을 통한 성능 지표 및 테스트 이미지 예측에 액세스하는 방법에 대한 자세한 내용은 성능 지표 보기 (SDK)을 참조하십시오.

이미지 분류 지표

Amazon Lookout for Vision은 테스트 중에 모델이 생성하는 분류에 대해 다음과 같은 요약 지표를 제공합니다.

- 정밀도
- 재현율
- F1 점수

이미지 분할 모델 지표

모델이 이미지 분할 모델인 경우 Amazon Lookout for Vision은 각 예외 항목 레이블에 대한 요약 이미지 분류 지표와 요약 성능 지표를 제공합니다.

- F1 점수
- Union (IoU)에 대한 평균 Intersection

정밀도

정밀도 지표는 다음과 같은 질문에 대한 답을 제공합니다. 모델이 이미지에 이상 징후가 포함되어 있다고 예측하면 예측이 얼마나 자주 정확할까요?

정밀도는 오탐으로 인한 비용이 많이 드는 상황에 유용한 지표입니다. 예를 들어, 조립된 기계에서 결함이 없는 기계 부품을 제거하는 데 드는 비용.

Amazon Lookout for Vision은 전체 테스트 데이터 세트에 대한 요약 정밀도 지표 값을 제공합니다.

정밀도는 모든 예측된 변칙 (참 긍정 및 거짓 긍정)에 비해 정확하게 예측된 예외 항목 (참 긍정)의 비율입니다. 정밀도 공식은 다음과 같습니다.

정밀도 값 = 참 긍정/ (참 긍정+거짓 긍정)

가능한 정밀도 값의 범위는 0—1입니다. Amazon Lookout for Vision 콘솔은 정밀도를 백분율 값 (0—100)으로 표시합니다.

정밀도 값이 높을수록 예측된 예외 항목 중 더 많은 부분이 정확하다는 것을 나타냅니다. 예를 들어, 모델이 100개의 이미지가 비정상이라고 예측한다고 가정해 보겠습니다. 예측값 중 85개가 맞고 (참 긍정) 15개가 틀린 경우 (거짓 긍정) 정밀도는 다음과 같이 계산됩니다.

참 긍정 85개/ (참 긍정 85개+거짓 긍정 15개) = 0.85 정밀도 값

그러나 모델이 100개의 예외 항목 예측 중 40개 이미지만 올바르게 예측하는 경우 결과 정밀도 값은 0.40 (즉, 40/(40 + 60) = 0.40)에서 더 낮습니다. 이 경우 모형은 올바른 예측보다 잘못된 예측을 더 많이 합니다. 이 문제를 해결하려면 모델을 개선해 보세요. 자세한 내용은 2단계: 모델 개선 단원을 참조하십시오.

자세한 내용은 정밀도 및 재현율을 참조하세요.

재현율

재현율 지표는 다음과 같은 질문에 대한 답을 제시합니다. 테스트 데이터 세트에 있는 총 비정상 이미지 수 중 비정상으로 올바르게 예측되는 영상의 수는 몇 개입니까?

재현율 지표는 오탐으로 인한 비용이 많이 드는 상황에 유용합니다. 결함 부품을 제거하지 않는 데 드는 비용이 많이 드는 경우를 예로 들 수 있습니다. Amazon Lookout for Vision은 전체 테스트 데이터 세트에 대한 요약 재현율 지표 값을 제공합니다.

재현율은 비정상적인 테스트 이미지 중 올바르게 탐지된 비율을 나타냅니다. 이는 테스트 데이터 세트의 이미지에 비정상 이미지가 실제로 존재할 때 모델이 얼마나 자주 이를 정확하게 예측할 수 있는지를 나타내는 척도입니다. 재현율 공식은 다음과 같이 계산됩니다.

재현율 값 = 참 긍정/ (참 긍정+오탐)

재현율 범위는 0—1입니다. Amazon Lookout for Vision 콘솔은 재현율을 백분율 값 (0—100)으로 표시합니다.

재현율 값이 높을수록 더 많은 비정상 이미지가 올바르게 식별되었음을 나타냅니다. 예를 들어 테스트 데이터 세트에 100개의 비정상 이미지가 포함되어 있다고 가정해 보겠습니다. 모델이 100개의 비정상 이미지 중 90개를 올바르게 탐지한 경우 재현율은 다음과 같습니다.

참 긍정 90개/ (참 긍정 90개+오탐 10개) = 0.90 재현율

재현율 값이 0.90이면 모델이 테스트 데이터 세트에 있는 대부분의 비정상 이미지를 올바르게 예측하고 있음을 나타냅니다. 모델이 변칙 이미지 중 20개만 정확하게 예측하는 경우 재현율은 0.20 (즉, 20/(20 + 80) = 0.20)로 낮아집니다.

이 경우 모델을 개선하는 것을 고려해야 합니다. 자세한 내용은 2단계: 모델 개선 단원을 참조하십시오.

자세한 내용은 정밀도 및 재현율을 참조하세요.

F1 점수

Amazon Lookout for Vision은 테스트 데이터 세트에 대한 평균 모델 성능 점수를 제공합니다. 특히, 예외 항목 분류를 위한 모델 성능은 정밀도 및 재현율 점수의 조화 평균인 F1 점수 지표로 측정됩니다.

F1 점수는 정밀도와 재현율을 모두 고려한 집계 측정값입니다. 모델 성능 점수는 0에서 1 사이의 값입니다. 값이 높을수록 재현율과 정밀도 모두에서 모델의 성능이 향상됩니다. 예를 들어 정밀도가 0.9이고 재현율이 1.0인 모델의 경우 F1 점수는 0.947입니다.

예를 들어 정밀도가 0.30이고 재현율이 1.0인 경우와 같이 모델의 성능이 좋지 않은 경우 F1 점수는 0.46입니다. 마찬가지로 정밀도가 높고(0.95) 재현율이 낮으면(0.20) F1 점수는 0.33입니다. 두 경우 모두 F1 점수가 낮으며 이는 모델에 문제가 있음을 나타냅니다.

자세한 내용은 F1 점수를 참조하세요.

Union (IoU)에 대한 평균 Intersection

테스트 이미지의 예외 항목 마스크와 모델이 테스트 이미지에 대해 예측하는 예외 항목 마스크 간의 평균 백분율 겹침 Amazon Lookout for Vision은 각 예외 항목 레이블에 대한 평균 IOU를 반환하며 이미지 분할 모델에서만 반환됩니다.

백분율 값이 낮으면 모델이 라벨의 예측 마스크를 테스트 이미지의 마스크와 정확하게 일치시키지 않는다는 것을 나타냅니다.

다음 이미지는 IOU가 낮습니다. 주황색 마스크는 모델에서 예측한 것으로, 테스트 이미지의 마스크를 나타내는 파란색 마스크를 완전히 덮지는 않습니다.

다음 이미지는 IOU가 더 높습니다. 파란색 마스크 (테스트 이미지)는 주황색 마스크 (예측 마스크)로 완전히 가려져 있습니다.

결과 테스트

테스트 중에 모델은 테스트 데이터 세트의 각 테스트 이미지에 대한 분류를 예측합니다. 다음과 같이 각 예측 결과를 해당 테스트 영상의 레이블 (정상 또는 변칙)과 비교합니다.

- 이미지가 비정상이라고 정확하게 예측하는 것은 참 긍정으로 간주됩니다.
- 이미지가 비정상이라고 잘못 예측하는 것은 false positive로 간주됩니다.
- 이미지가 정상이라고 정확하게 예측하는 것은 true negative로 간주됩니다.
- 이미지가 정상이라고 잘못 예측하는 것은 false negative로 간주됩니다.

모델이 분할 모델인 경우 모델은 테스트 영상의 이상 위치에 대한 마스크와 예외 항목 레이블도 예측합니다.

Amazon Lookout for Vision은 비교 결과를 사용하여 성능 지표를 생성합니다.

2단계: 모델 개선

성능 지표는 모델을 개선할 수 있는 것으로 나타날 수 있습니다. 예를 들어, 모델이 테스트 데이터 세트에서 모든 예외를 감지하지 못하면 모델의 재현율이 낮습니다 (즉, 재현율 지표의 값이 낮음). 모델을 개선해야 하는 경우 다음을 고려하세요.

- 학습 및 테스트 데이터 세트 이미지에 레이블이 제대로 지정되어 있는지 확인하세요.
- 조명 및 물체 포즈와 같은 이미지 캡처 조건의 가변성을 줄이고 동일한 유형의 객체를 기반으로 모델을 학습시키십시오.
- 이미지에 필요한 내용만 표시되는지 확인하십시오. 예를 들어 프로젝트에서 기계 부품의 이상을 감지하는 경우 이미지에 다른 물체가 없는지 확인하세요.
- 레이블이 지정된 이미지를 학습 및 테스트 데이터세트에 더 추가하세요. 테스트 데이터 세트의 재현율과 정밀도가 뛰어나지만 모델을 배포했을 때 성능이 좋지 않은 경우 테스트 데이터 세트를 충분히 대표하지 못하므로 확장해야 할 수 있습니다.
- 테스트 데이터 세트의 재현율 및 정밀도가 떨어지는 경우 학습 데이터 세트와 테스트 데이터 세트의 이상 현상과 이미지 캡처 조건이 얼마나 잘 일치하는지 생각해 보세요. 학습 이미지는 예상된 이상 및 조건을 대표하지 않지만 테스트 이미지의 이미지는 그렇지 않은 경우, 학습 데이터 세트에 예상 이상 및 조건을 포함하는 이미지를 추가하십시오. 테스트 데이터 세트 이미지는 예상한 조건에 맞지 않지만 학습 이미지는 예상한 조건에 속하지 않는 경우 테스트 데이터 세트를 업데이트하세요.

  자세한 내용은 더 많은 이미지 추가 단원을 참조하십시오. 레이블이 지정된 이미지를 학습 데이터 세트에 추가하는 또 다른 방법은 시험 탐지 작업을 실행하고 결과를 확인하는 것입니다. 그런 다음 검

증된 이미지를 학습 데이터 세트에 추가할 수 있습니다. 자세한 내용은 평가판 확인 작업을 통한 모델 검증 단원을 참조하십시오.

- 학습 및 테스트 데이터 세트에 충분히 다양한 정상 이미지와 비정상 이미지가 있는지 확인하세요. 이미지는 모델에서 보게 될 정상 이미지와 비정상 이미지의 유형을 나타내야 합니다. 예를 들어, 회로 기판을 분석할 때 일반 이미지는 저항기 및 트랜지스터와 같은 구성 요소의 위치 및 납땜의 변화를 나타내야 합니다. 비정상적인 이미지는 구성 요소가 잘못 배치되거나 누락되는 등 시스템에서 발생할 수 있는 다양한 유형의 이상을 나타내야 합니다.
- 탐지된 이상 유형에 대한 모델의 평균 IOU가 낮은 경우 분할 모델의 마스크 출력을 확인하십시오. 스크래치와 같은 일부 사용 사례의 경우 모델은 테스트 이미지의 실측 스크래치와 매우 비슷하지만 픽셀 중첩은 적은 스크래치를 출력할 수 있습니다. 1픽셀 간격으로 떨어져 있는 두 개의 평행선을 예로 들 수 있습니다. 이러한 경우 평균 IOU는 예측 성공을 측정하는 신뢰할 수 없는 지표입니다.
- 이미지 크기가 작거나 이미지 해상도가 낮으면 더 높은 해상도로 이미지를 캡처하는 것을 고려해 보십시오. 이미지 크기는 64x64픽셀에서 최대 4096픽셀 X 4096픽셀까지 다양합니다.
- 예외 크기가 작은 경우 이미지를 별도의 타일로 나누고 바둑판식으로 배열된 이미지를 학습 및 테스트에 사용하는 것이 좋습니다. 이렇게 하면 모델이 이미지에서 더 큰 크기의 결함을 확인할 수 있습니다.

학습 및 테스트 데이터 세트를 개선한 후에는 모델을 재학습하고 재평가하십시오. 자세한 내용은 모델 학습 단원을 참조하십시오.

측정치를 통해 모델의 성능이 만족스러운 것으로 나타나면 평가판 확인 작업의 결과를 테스트 데이터 세트에 추가하여 모델의 성능을 확인할 수 있습니다. 재학습 후에는 성능 지표가 이전 학습의 성과 지표를 확인해야 합니다. 자세한 내용은 평가판 확인 작업을 통한 모델 검증 단원을 참조하십시오.

성능 지표 보기

콘솔에서 그리고 `DescribeModel` 작업을 호출하여 성능 지표를 가져올 수 있습니다.

주제

- 성능 지표 보기 (콘솔)
- 성능 지표 보기 (SDK)

성능 지표 보기 (콘솔)

학습이 완료되면 콘솔에 성능 지표가 표시됩니다.

Amazon Lookout for Vision 콘솔은 테스트 중에 수행한 분류에 대한 다음과 같은 성능 지표를 보여줍니다.

- 정밀도
- 재현율
- F1 점수

모델이 분할 모델인 경우 콘솔에는 각 예외 항목 레이블에 대해 다음과 같은 성능 지표도 표시됩니다.

- 예외 항목 레이블이 발견된 테스트 이미지 수입니다.
- F1 점수
- Union (IoU)에 대한 평균 Intersection

테스트 결과 개요 섹션에는 테스트 데이터 세트의 이미지에 대한 총 정답 및 오답 예측이 표시됩니다. 또한 테스트 데이터 세트의 개별 이미지에 대한 예측 및 실제 레이블 할당을 확인할 수 있습니다.

다음 절차에서는 프로젝트의 모델 목록 보기에서 성능 지표를 가져오는 방법을 보여 줍니다.

성능 지표 (콘솔)를 보려면

1. https://console.aws.amazon.com/lookoutvision/에서 Amazon Lookout for Vision 콘솔을 엽니다.
2. Get started를 선택합니다.
3. 왼쪽 탐색 창에서 프로젝트를 선택합니다.
4. 프로젝트 보기에서 확인하고자 하는 모델의 버전이 포함된 프로젝트를 선택합니다.
5. 왼쪽 탐색 창의 프로젝트 이름에서 모델을 선택합니다.
6. 모델 목록 보기에서 확인하고자 하는 모델의 버전을 선택합니다.
7. 모델 세부 정보 페이지의 성능 지표 탭에서 성능 지표를 확인하세요.
8. 다음 사항에 유의하세요.

    a. 모델 성능 지표 섹션에는 모델이 테스트 이미지에 대해 만든 분류 예측에 대한 전체 모델 지표 (정밀도, 재현율, F1 점수)가 포함되어 있습니다.

    b. 모델이 이미지 분할 모델인 경우 레이블당 성능 섹션에는 예외 레이블이 발견된 테스트 이미지 수가 포함됩니다. 또한 각 예외 항목 레이블에 대한 지표 (F1 점수, 평균 IOU)도 볼 수 있습니다.

c. 테스트 결과 개요 섹션에서는 Lookout for Vision이 모델을 평가하는 데 사용하는 각 테스트 이미지에 대한 결과를 제공합니다. 여기에는 다음이 포함됩니다.

- 모든 테스트 이미지에 대한 정답 (true positive) 및 오답 (false negative) 분류 예측 (정상 또는 비정상)의 총 수입니다.
- 각 테스트 이미지에 대한 분류 예측. 이미지 아래에 정정이 표시되면 예측된 분류가 영상의 실제 분류와 일치합니다. 그렇지 않으면 모델이 이미지를 올바르게 분류하지 못한 것입니다.
- 이미지 분할 모델을 사용하면 모델이 이미지에 할당한 예외 항목 레이블과 예외 항목 레이블의 색상과 일치하는 이미지 마스크가 표시됩니다.

성능 지표 보기 (SDK)

DescribeModel 작업을 사용하여 모델에 대한 요약 성능 지표 (분류), 평가 매니페스트, 모델에 대한 평가 결과를 가져올 수 있습니다.

요약 성능 지표 가져오기

테스트 중에 모델에서 수행한 분류 예측에 대한 요약 성능 지표 (정밀도, 재현율 및F1 점수)는 `DescribeModel`를 호출하여 반환된 `Performance` 필드에 반환됩니다.

```
        "Performance": {
            "F1Score": 0.8,
            "Recall": 0.8,
            "Precision": 0.9
        },
```

분할 모델에서 반환한 예외 항목 레이블 성능 지표는 `Performance` 필드에 포함되지 않습니다. `EvaluationResult` 필드에서 가져올 수 있습니다. 자세한 내용은 평가 결과 검토 단원을 참조하십시오.

요약 성능 지표에 대한 자세한 내용은 1단계: 모델의 성능 평가을 참조하십시오. 예제 코드는 모델 보기 (SDK) 항목을 참조하세요.

평가 매니페스트 사용

평가 매니페스트는 모델을 테스트하는 데 사용된 개별 이미지에 대한 테스트 예측 지표를 제공합니다. 테스트 데이터세트의 각 이미지에 대한 JSON 라인에는 원본 테스트 (그라운드 트루스) 정보와 이미지

에 대한 모델의 예측이 포함됩니다. Amazon Lookout for Vision은 평가 매니페스트를 Amazon S3 버킷에 저장합니다. `DescribeModel` 작업의 응답 내 `EvaluationManifest` 필드에서 위치를 가져올 수 있습니다.

```
"EvaluationManifest": {
            "Bucket": "lookoutvision-us-east-1-nnnnnnnnnn",
            "Key": "my-sdk-project-model-output/EvaluationManifest-my-sdk-
project-1.json"
        }
```

파일 이름 형식은 `EvaluationManifest-`*`project name`*`.json`입니다. 예제 코드는 모델 보기 항목을 참조하세요.

다음 샘플 JSON 줄에서 `class-name`는 이미지 내용에 대한 기본 정보입니다. 이 예시에서는 이미지에 예외 항목이 포함되어 있습니다. 이 `confidence` 필드는 Amazon Lookout for Vision이 예측에 대해 갖고 있는 확신을 보여줍니다.

```
{
    "source-ref"*: "s3://customerbucket/path/to/image.jpg",
    "source-ref-metadata": {
        "creation-date": "2020-05-22T21:33:37.201882"
    },

    // Test dataset ground truth
    "anomaly-label": 1,
    "anomaly-label-metadata": {
        "class-name": "anomaly",
        "type": "groundtruth/image-classification",
        "human-annotated": "yes",
        "creation-date": "2020-05-22T21:33:37.201882",
        "job-name": "labeling-job/anomaly-detection"
    },
    // Anomaly label detected by Lookout for Vision
    "anomaly-label-detected": 1,
    "anomaly-label-detected-metadata": {
        "class-name": "anomaly",
        "confidence": 0.9,
        "type": "groundtruth/image-classification",
        "human-annotated": "no",
        "creation-date": "2020-05-22T21:33:37.201882",
        "job-name": "training-job/anomaly-detection",
```

```
        "model-arn": "lookoutvision-some-model-arn",
        "project-name": "lookoutvision-some-project-name",
        "model-version": "lookoutvision-some-model-version"
    }
}
```

평가 결과 검토

평가 결과에는 전체 테스트 이미지 세트에 대한 다음과 같은 집계 성능 지표 (분류)가 있습니다.

- 정밀도
- 재현율
- ROC 곡선 (콘솔에는 표시되지 않음)
- 평균 정밀도 (콘솔에는 표시되지 않음)
- F1 점수

평가 결과에는 모델 학습 및 테스트에 사용된 이미지 수도 포함됩니다.

모델이 분할 모델인 경우 평가 결과에는 테스트 데이터 세트에서 발견된 각 예외 항목 레이블에 대한 다음 지표도 포함됩니다.

- 정밀도
- 재현율
- F1 점수
- Union (IoU)에 대한 평균 Intersection

Amazon Lookout for Vision은 평가 결과를 Amazon S3 버킷에 저장합니다. `DescribeModel` 작업에서 응답 내 `EvaluationResult`의 값을 확인하여 위치를 가져올 수 있습니다.

```
"EvaluationResult": {
            "Bucket": "lookoutvision-us-east-1-nnnnnnnnnn",
            "Key": "my-sdk-project-model-output/EvaluationResult-my-sdk-project-1.json"
        }
```

파일 이름 형식은 `EvaluationResult-`*`project name`*`.json`입니다. 예시는 모델 보기에서 확인하십시오.

다음 스키마는 평가 결과를 보여줍니다.

```
   {
       "Version": 1,
       "EvaluationDetails":
       {
           "ModelArn": "string", // The Amazon Resource Name (ARN) of the model
version.
           "EvaluationEndTimestamp": "string",   // The UTC date and time that
evaluation finished.
           "NumberOfTrainingImages": int,        // The number of images that were
successfully used for training.
           "NumberOfTestingImages": int          // The number of images that were
successfully used for testing.
       },
       "AggregatedEvaluationResults":
       {
           "Metrics":
           {                       //Classification metrics.
               "ROCAUC": float,            // ROC area under the curve.
               "AveragePrecision": float,  // The average precision of the model.
               "Precision": float,         // The overall precision of the model.
               "Recall": float,            // The overall recall of the model.
               "F1Score": float,           // The overal F1 score for the model.

               "PixelAnomalyClassMetrics":     //Segmentation metrics.
               [
                   {
                       "Precision": float,       // The precision for the anomaly
label.
                       "Recall": float,          // The recall for the anomaly label.
                       "F1Score": float,         // The F1 score for the anomaly
label.
                       "AIOU" : float,           // The average Intersection Over
Union for the anomaly label.
                       "ClassName": "string"   // The anomaly label.
                   }
               ]
           }
       }
   }
```

평가판 확인 작업을 통한 모델 검증

모델의 품질을 확인하거나 개선하려는 경우 평가판 확인 작업을 실행할 수 있습니다. 평가판 확인 작업은 사용자가 제공하는 새 이미지에서 이상을 탐지합니다.

확인 결과를 확인하고 검증된 이미지를 데이터 세트에 추가할 수 있습니다. 학습 데이터 세트와 테스트 데이터 세트가 분리되어 있는 경우 검증된 이미지가 학습 데이터세트에 추가됩니다.

로컬 컴퓨터의 이미지 또는 Amazon S3 버킷에 있는 이미지를 확인할 수 있습니다. 검증된 이미지를 데이터세트에 추가하려면 S3 버킷에 있는 이미지가 데이터세트의 이미지와 동일한 S3 버킷에 있어야 합니다.

> **Note**
>
> 평가판 확인 작업을 실행하려면 S3 버킷에 버전 관리가 활성화되어 있어야 합니다. 자세한 내용은 버전 관리 사용을 참조하십시오. 콘솔 버킷은 버전 관리가 활성화된 상태로 생성됩니다.

기본적으로 이미지는 AWS가 소유하고 관리하는 키로 암호화됩니다. 자체 AWS Key Management Service (KMS) 키를 사용하도록 선택할 수도 있습니다. 자세한 내용은 AWS Key Management Service 개념을 참조하십시오.

주제

- 평가판 확인 작업 실행
- 평가판 확인 결과 확인
- 주석 도구를 사용하여 분할 레이블 수정하기

평가판 확인 작업 실행

평가판 확인 작업을 실행하려면 다음 단계를 수행하세요.

평가판 확인을 실행하려면 (콘솔)

1. https://console.aws.amazon.com/lookoutvision/에서 Amazon Lookout for Vision 콘솔을 엽니다.
2. Get started를 선택합니다.
3. 왼쪽 탐색 창에서 프로젝트를 선택합니다.
4. 프로젝트 보기에서 확인하고자 하는 모델의 버전이 포함된 프로젝트를 선택합니다.

5. 왼쪽 탐색 창의 프로젝트 이름에서 평가판 확인을 선택합니다.
6. 평가판 확인 보기에서 평가판 확인 실행을 선택합니다.
7. 평가판 확인 실행 페이지의 작업 이름에 평가판 확인 작업 이름을 입력합니다.
8. 모델 선택에서 사용하고자 하는 모델의 버전을 선택합니다.
9. 다음과 같이 이미지 소스에 따라 이미지를 가져옵니다.
   - Amazon S3 버킷에서 소스 이미지를 가져오는 경우 S3 URI를 입력합니다.

     > **Tip**
     >
     > 시작하기 예제 이미지를 사용하는 경우 extra_images 폴더를 사용하십시오. Amazon S3 URI는 `s3://your bucket/circuitboard/extra_images`입니다.

   - 컴퓨터에서 이미지를 업로드하는 경우, 예외 항목 감지를 선택한 후 이미지를 추가하십시오.
10. (선택 사항) 자체 AWS KMS 암호화 키를 사용하려면 다음을 수행하십시오.
    a. 이미지 데이터 암호화의 경우 암호화 설정 사용자 지정 (고급)을 선택합니다.
    b. 암호화.aws_kms_key에서 키의 Amazon 리소스 이름 (ARN)을 입력하거나 기존 AWS KMS 키를 선택합니다. 새 키를 생성하려면 AWS IMS 키 생성을 선택합니다.
11. 이상 탐지를 선택한 다음 평가판 확인 실행을 선택하여 평가판 확인 작업을 시작합니다.
12. 평가판 확인 보기에서 현재 상태를 확인하세요. 평가판 확인을 완료하는 데 시간이 걸릴 수 있습니다.

평가판 확인 결과 확인

평가판 확인 결과를 검증하면 모델을 개선하는 데 도움이 될 수 있습니다.

성능 지표가 좋지 않은 경우, 평가판 확인을 실행하여 모델을 개선한 다음 검증된 이미지를 데이터 세트 (별도의 데이터 세트가 있는 경우 학습 데이터 세트)에 추가하세요.

모델의 성능 지표는 양호하지만 평가판 확인 결과가 좋지 않은 경우 데이터세트 (학습 데이터세트) 에 검증된 이미지를 추가하여 모델을 개선할 수 있습니다. 별도의 테스트 데이터 세트가 있는 경우 테스트 데이터 세트에 이미지를 더 추가하는 것을 고려해 보세요.

검증된 이미지를 데이터세트에 추가한 후에는 모델을 재학습하고 재평가하세요. 자세한 내용은 모델 학습 단원을 참조하십시오.

평가판 확인 결과를 검증하기 위해

1. https://console.aws.amazon.com/lookoutvision/에서 Amazon Lookout for Vision 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 프로젝트를 선택합니다.
3. 프로젝트 페이지에서 사용할 프로젝트를 선택합니다. 프로젝트의 대시보드가 표시됩니다.
4. 왼쪽 탐색 창에서 평가판 확인을 선택합니다.
5. 확인하고자 하는 평가판 확인을 선택하십시오.
6. 평가판 확인 페이지에서 기계 예측 확인을 선택합니다.
7. 이 페이지에 있는 모든 이미지 선택을 선택합니다.
8. 예측이 정확하면 올바른 것으로 확인을 선택합니다. 그렇지 않으면 틀린 것으로 확인을 선택합니다. 예측 및 예측 신뢰도 점수는 각 이미지 아래에 표시됩니다.
9. 이미지의 레이블을 변경해야 하는 경우 다음을 수행하세요.

    a. 이미지 아래에서 정정 또는 틀림을 선택합니다.

    b. 이미지의 올바른 라벨을 결정할 수 없는 경우 갤러리에서 이미지를 선택하여 이미지를 확대하세요.

> **Note**
>
> 필터 섹션에서 원하는 레이블 또는 레이블 상태를 선택하여 이미지 레이블을 필터링할 수 있습니다. 정렬 옵션 섹션에서 신뢰도 점수를 기준으로 정렬할 수 있습니다.

10. 모델이 분할 모델이고 이미지의 마스크 또는 예외 항목 레이블이 잘못된 경우 이미지 아래에서 비정상 영역을 선택하고 주석 도구를 여십시오. 주석 도구를 사용하여 분할 레이블 수정하기를 수행하여 분할 정보를 업데이트하십시오.
11. 모든 이미지가 확인될 때까지 필요에 따라 각 페이지에서 7~10단계를 반복합니다.
12. 데이터세트에 검증된 이미지 추가를 선택합니다. 별도의 데이터 세트가 있는 경우 이미지가 학습 데이터 세트에 추가됩니다.
13. 모델 재학습하기 자세한 내용은 the section called “모델 학습” 단원을 참조하십시오.

주석 도구를 사용하여 분할 레이블 수정하기

주석 도구를 사용하면 마스크로 이상 영역을 표시하여 이미지를 분할할 수 있습니다.

주석 도구를 사용하여 이미지의 분할 레이블을 수정하려면

1. 데이터세트 갤러리의 이미지 아래에 있는 비정상 영역을 선택하여 주석 도구를 엽니다.
2. 마스크의 예외 항목 레이블이 올바르지 않은 경우 마스크를 선택한 다음 예외 항목 레이블에서 올바른 예외 항목 레이블을 선택합니다. 필요한 경우 예외 항목 레이블 추가를 선택하여 새 예외 항목 레이블을 추가합니다.
3. 마스크가 올바르지 않으면 페이지 하단에 있는 그리기 도구를 선택하고 예외 항목 레이블의 이상 영역을 촘촘하게 덮는 마스크를 그리십시오. 다음 이미지는 예외 항목을 촘촘하게 가리는 마스크의 예입니다.

다음은 이상 현상을 완전히 가리지 않는 불량 마스크의 예제입니다.

4. 수정해야 할 이미지가 더 있으면 다음을 선택하고 2단계와 3단계를 반복합니다.
5. 제출 및 닫기를 선택하여 이미지 업데이트를 완료합니다.

훈련된 Amazon Lookout for Vision 모델 실행

모델로 이미지의 이상을 감지하려면 먼저 StartModel 작업을 통해 모델을 시작해야 합니다. Amazon Lookout for Vision 콘솔은 모델을 시작하고 중지하는 데 사용할 수 있는 AWS CLI 명령을 제공합니다. 이 섹션에는 사용할 수 있는 예제 코드가 포함되어 있습니다.

모델이 시작된 후 `DetectAnomalies` 작업을 사용하여 이미지의 이상을 감지할 수 있습니다. 자세한 내용은 이미지에서 이상 탐지 단원을 참조하십시오.

주제

추론 단위

모델을 시작하면 Amazon Lookout for Vision은 추론 단위라고 하는 최소 하나의 컴퓨팅 리소스를 프로비저닝합니다. `StartModel` API의 `MinInferenceUnits` 입력 파라미터에 사용할 추론 단위 수를 지정합니다. 모델의 기본 할당은 추론 단위 1개입니다.

> **Important**
>
> 모델 실행 구성 방법에 따라 모델이 실행되는 시간 및 모델이 실행되는 동안 사용하는 추론 단위 수에 대한 요금이 부과됩니다. 예를 들어 두 개의 추론 단위로 모델을 시작하고 8시간 동안 모델을 사용하면 추론 시간 16시간 (실행 시간 8시간* 추론 단위 2개)에 대한 요금이 부과됩니다. 자세한 내용은 Amazon Lookout for Vision 요금을 참조하세요. StopMode을 호출하여 명시적으로 모델을 중지하지 않으면 모델로 이미지를 적극적으로 분석하지 않더라도 요금이 부과됩니다.

단일 추론 단위가 지원하는 초당 트랜잭션 수 (TPS)는 다음의 영향을 받습니다.

- Lookout for Vision에서 모델을 훈련하는 데 사용하는 알고리즘입니다. 모델을 학습시키면 여러 모델이 학습됩니다. Lookout for Vision은 데이터 세트의 크기와 정상 및 비정상 이미지의 구성을 기반으로 최상의 성능을 가진 모델을 선택합니다.

- 해상도가 높은 이미지는 분석에 더 많은 시간을 필요로 합니다.
- 크기가 작은 이미지 (MB 단위) 는 큰 이미지보다 빠르게 분석됩니다.

추론 단위를 사용한 처리량 관리

애플리케이션의 요구에 따라 모델의 처리량을 늘리거나 줄일 수 있습니다. 처리량을 늘리려면 추가 추론 단위를 사용하십시오. 추론 단위가 추가될 때마다 처리 속도가 추론 단위당 1씩 빨라집니다. 필요한 추론 단위 수를 계산하는 방법에 대한 자세한 내용은 Amazon Rekognition Custom Labels 및 Amazon Lookout for Vision 모델의 추론 단위 계산을 참조하세요. 모델의 지원되는 처리량을 변경하려면 다음 두 가지 옵션이 있습니다.

수동으로 추론 유닛을 추가하거나 제거합니다.

모델을 중지한 다음 필요한 수의 추론 단위로 다시 시작합니다. 이 접근 방식의 단점은 모델을 다시 시작하는 동안 요청을 받을 수 없고 급증하는 수요를 처리하는 데 사용할 수 없다는 것입니다. 모델의 처리량이 안정적이고 사용 사례가 10~20분의 가동 중지 시간을 견딜 수 있는 경우 이 접근 방식을 사용하세요. 예를 들어 주간 일정을 사용하여 모델과의 통화를 일괄 처리하려는 경우를 들 수 있습니다.

추론 단위 자동 규모 조정

모델이 수요 급증을 수용해야 하는 경우 Amazon Lookout for Vision은 모델에서 사용하는 추론 단위의 수를 자동으로 조정할 수 있습니다. 수요가 증가하면 Amazon Lookout for Vision은 모델에 추가 추론 단위를 추가하고 수요가 감소하면 이를 제거합니다.

Lookout for Vision에서 모델의 추론 단위를 자동으로 조정하도록 하려면 모델을 시작하고 `MaxInferenceUnits` 매개변수를 사용하여 사용할 수 있는 최대 추론 단위 수를 설정합니다. 최대 추론 단위 수를 설정하면 모델에 사용할 수 있는 추론 단위 수를 제한하여 모델 실행 비용을 관리할 수 있습니다. 최대 단위 수를 지정하지 않으면 Lookout for Vision은 처음에 사용한 추론 단위 수만 사용하여 모델의 크기를 자동으로 조정하지 않습니다. 최대 추론 단위 수에 대한 자세한 내용은 Service Quotas를 참조하십시오.

`MinInferenceUnits` 파라미터를 사용하여 최소 추론 단위 수를 지정할 수도 있습니다. 이를 통해 모델의 최소 처리량을 지정할 수 있습니다. 여기서 단일 추론 단위는 1시간의 처리 시간을 나타냅니다.

> **Note**
>
> Lookout for Vision 콘솔에서는 최대 추론 단위 수를 설정할 수 없습니다. 대신 `StartModel` 작업에 `MaxInferenceUnits` 입력 파라미터를 지정하십시오.

Lookout for Vision은 모델의 현재 자동 조정 상태를 확인하는 데 사용할 수 있는 다음과 같은 Amazon CloudWatch Logs 지표를 제공합니다.

지표	설명
`DesiredInferenceUnits`	Lookout for Vision이 확대 또는 축소되는 추론 단위의 수입니다.
`InServiceInferenceUnits`	모델이 사용하는 추론 단위의 수.

`DesiredInferenceUnits=InServiceInferenceUnits`인 경우 Lookout for Vision은 현재 추론 단위 수를 조정하지 않습니다.

`DesiredInferenceUnits>InServiceInferenceUnits`인 경우 Lookout for Vision은 `DesiredInferenceUnits`의 값까지 확장되고 있습니다.

`DesiredInferenceUnits<InServiceInferenceUnits` 인 경우 Lookout for Vision은 `DesiredInferenceUnits`의 값으로 축소되고 있습니다.

Lookout for Vision에서 반환되는 지표 및 필터링 차원에 대한 자세한 내용은 Amazon CloudWatch를 사용한 Lookout for Vision 모니터링을 참조하십시오.

모델에 대해 요청한 최대 추론 단위 수를 확인하려면 DescribeModel을 호출하고 응답의 `MaxInferenceUnits`필드를 확인하십시오.

가용 영역

Amazon Lookout for Vision; 은 AWS 리전 내 여러 가용 영역에 추론 단위를 분산하여 가용성을 높입니다. 자세한 내용은 가용 영역을 참조하세요. 가용 영역 중단 및 추론 단위 장애로부터 프로덕션 모델을 보호하려면 최소 두 개의 추론 단위로 프로덕션 모델을 시작하십시오.

가용 영역이 중단되면 가용 영역의 모든 추론 유닛을 사용할 수 없게 되고 모델 용량이 감소합니다. DetectAnomalies에 대한 호출은 나머지 추론 단위에 재분배됩니다. 이러한 호출은 나머지 추론 단위의 지원되는 초당 트랜잭션 수 (TPS) 를 초과하지 않으면 성공합니다. 가 가용 영역을 AWS 복구하면 추론 단위가 다시 시작되고 전체 용량이 복원됩니다.

단일 추론 단위에 장애가 발생하는 경우 Amazon Lookout for Vision은 동일한 가용 영역에서 새 추론 단위를 자동으로 시작합니다. 새 추론 단위가 시작되기 전까지는 모델 용량이 줄어듭니다.

Amazon Lookout for Vision 모델 시작하기

Amazon Lookout for Vision 모델을 사용하여 이상 현상을 탐지하려면 먼저 모델을 시작해야 합니다. StartModel API를 호출하고 다음을 전달하여 모델을 시작합니다.

- ProjectName — 시작하려는 모델이 포함된 프로젝트 이름입니다.
- ModelVersion — 시작하려는 모델의 버전입니다.
- MinInferenceUnits — 추론 단위의 최소 수입니다. 자세한 내용은 추론 단위 단원을 참조하십시오.
- (선택 사항) MaxInferenceUnits — Amazon Lookout for Vision에서 모델을 자동으로 확장하는 데 사용할 수 있는 최대 추론 단위 수입니다. 자세한 내용은 추론 단위 자동 규모 조정 단원을 참조하십시오.

Amazon Lookout for Vision에서는 모델을 시작하고 중지하는 데 사용할 수 있는 예제 코드를 제공합니다.

> **Note**
>
> 모델이 실행되는 시간에 따라 요금이 부과됩니다. 모델 실행을 중지하려면 Amazon Lookout for Vision 모델 중지하기 항목을 참조하세요.
> AWS SDK를 사용하여 Lookout for Vision을 사용할 수 있는 모든 AWS 리전에서 실행 중인 모델을 볼 수 있습니다. 예제 코드는 find_running_models.py 를 참조하십시오.

주제

- 모델 시작 (콘솔)
- Amazon Lookout for Vision 모델 (SDK) 시작하기

모델 시작 (콘솔)

Amazon Lookout for Vision 콘솔은 모델을 시작하는 데 사용할 수 있는 AWS CLI 명령을 제공합니다. 모델이 시작되면 이미지의 이상 탐지를 시작할 수 있습니다. 자세한 내용은 이미지에서 이상 탐지 단원을 참조하십시오.

모델을 시작하려면 (콘솔)

1. 아직 설치하지 않은 경우 및 AWS SDKs AWS CLI 를 설치하고 구성합니다. 자세한 내용은 4단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.
2. https://console.aws.amazon.com/lookoutvision/에서 Amazon Lookout for Vision 콘솔을 엽니다.
3. Get started를 선택합니다.
4. 왼쪽 탐색 창에서 프로젝트를 선택합니다.
5. 프로젝트 리소스 페이지에서 시작하려는 학습된 모델이 포함된 프로젝트를 선택합니다.
6. 모델 항목에서 시작할 모델을 선택합니다.
7. 모델의 세부 정보 페이지에서 모델 사용을 선택한 다음 클라우드에 API 통합을 선택합니다.

> **Tip**
>
> 모델을 에지 기기에 배포하려면 모델 패키징 작업 생성을 선택합니다. 자세한 내용은 Amazon Lookout for Vision 모델 패키징 단원을 참조하십시오.

8. AWS CLI 명령에서를 호출하는 AWS CLI 명령을 복사합니다`start-model`.
9. 명령 프롬프트에서 이전 단계에서 복사한 `start-model` 명령을 입력합니다. `lookoutvision`프로필을 사용하여 자격 증명을 가져오려면 `--profile lookoutvision-access` 매개 변수를 추가하십시오.
10. 콘솔의 왼쪽 탐색 창에서 모델을 선택합니다.
11. 상태 열에서 모델의 현재 상태를 확인합니다. 상태가 호스팅인 경우 모델을 사용하여 이미지의 이상을 감지할 수 있습니다. 자세한 내용은 이미지에서 이상 탐지 단원을 참조하십시오.

Amazon Lookout for Vision 모델 (SDK) 시작하기

StartModel 작업을 호출하여 모델을 시작합니다.

모델을 시작하는 데 시간이 걸릴 수 있습니다. DescribeModel을 호출하여 현재 상태를 확인할 수 있습니다. 자세한 내용은 모델 보기 단원을 참조하십시오.

모델 (SDK) 을 시작하려면

1. 아직 설치하지 않은 경우 및 AWS SDKs AWS CLI 를 설치하고 구성합니다. 자세한 내용은 4단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.
2. 다음 예제 코드를 사용하여 모델을 시작합니다.

CLI

다음 값을 변경합니다.

- `project-name`은 시작하려는 모델이 포함된 프로젝트 이름으로 이동합니다.
- `model-version`은 시작하려는 모델의 버전으로 이동합니다.
- `--min-inference-units`은 사용하려는 추론 단위 수입니다.
- (선택 사항) `--max-inference-units`은 Amazon Lookout for Vision에서 모델을 자동으로 스케일링하는 데 사용할 수 있는 최대 추론 단위 수까지 늘립니다.

```
aws lookoutvision start-model --project-name "project name"\
  --model-version model version\
  --min-inference-units minimum number of units\
  --max-inference-units max number of units \
  --profile lookoutvision-access
```

Python

이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져옵니다. 전체 예제는 여기에서 확인하세요.

```
    @staticmethod
    def start_model(
            lookoutvision_client, project_name, model_version,
min_inference_units, max_inference_units = None):
        """
        Starts the hosting of a Lookout for Vision model.

        :param lookoutvision_client: A Boto3 Lookout for Vision client.
        :param project_name:  The name of the project that contains the version
of the
                              model that you want to start hosting.
        :param model_version: The version of the model that you want to start
hosting.
        :param min_inference_units: The number of inference units to use for
hosting.
        :param max_inference_units: (Optional) The maximum number of inference
units that
        Lookout for Vision can use to automatically scale the model.
```

```
        """
        try:
            logger.info(
                "Starting model version %s for project %s", model_version,
project_name)

            if max_inference_units is None:
                lookoutvision_client.start_model(
                    ProjectName = project_name,
                    ModelVersion = model_version,
                    MinInferenceUnits = min_inference_units)

            else:
                lookoutvision_client.start_model(
                    ProjectName = project_name,
                    ModelVersion = model_version,
                    MinInferenceUnits = min_inference_units,
                    MaxInferenceUnits = max_inference_units)

            print("Starting hosting...")

            status = ""
            finished = False

            # Wait until hosted or failed.
            while finished is False:
                model_description = lookoutvision_client.describe_model(
                    ProjectName=project_name, ModelVersion=model_version)
                status = model_description["ModelDescription"]["Status"]

                if status == "STARTING_HOSTING":
                    logger.info("Host starting in progress...")
                    time.sleep(10)
                    continue

                if status == "HOSTED":
                    logger.info("Model is hosted and ready for use.")
                    finished = True
                    continue

                logger.info("Model hosting failed and the model can't be used.")
                finished = True

            if status != "HOSTED":
```

```
                logger.error("Error hosting model: %s", status)
                raise Exception(f"Error hosting model: {status}")
        except ClientError:
            logger.exception("Couldn't host model.")
            raise
```

Java V2

이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져옵니다. 전체 예제는 여기에서 확인하세요.

```
/**
 * Starts hosting an Amazon Lookout for Vision model. Returns when the model has
 * started or if hosting fails. You are charged for the amount of time that a
 * model is hosted. To stop hosting a model, use the StopModel operation.
 *
 * @param lfvClient   An Amazon Lookout for Vision client.
 * @param projectName The name of the project that contains the model that you
 *                    want to host.
 * @modelVersion The version of the model that you want to host.
 * @minInferenceUnits The number of inference units to use for hosting.
 * @maxInferenceUnits The maximum number of inference units that Lookout for
 *                    Vision can use for automatically scaling the model. If the
 *                    value is null, automatic scaling doesn't happen.
 * @return ModelDescription The description of the model, which includes the
 *         model hosting status.
 */
public static ModelDescription startModel(LookoutVisionClient lfvClient, String
 projectName, String modelVersion,
        Integer minInferenceUnits, Integer maxInferenceUnits) throws
 LookoutVisionException, InterruptedException {

    logger.log(Level.INFO, "Starting Model version {0} for project {1}.",
            new Object[] { modelVersion, projectName });

    StartModelRequest startModelRequest = null;

    if (maxInferenceUnits == null) {

        startModelRequest =
 StartModelRequest.builder().projectName(projectName).modelVersion(modelVersion)
                .minInferenceUnits(minInferenceUnits).build();
    } else {
```

```
        startModelRequest =
StartModelRequest.builder().projectName(projectName).modelVersion(modelVersion)

.minInferenceUnits(minInferenceUnits).maxInferenceUnits(maxInferenceUnits).build();
    }

    // Start hosting the model.
    lfvClient.startModel(startModelRequest);

    DescribeModelRequest describeModelRequest =
DescribeModelRequest.builder().projectName(projectName)
            .modelVersion(modelVersion).build();

    ModelDescription modelDescription = null;

    boolean finished = false;
    // Wait until model is hosted or failure occurs.
    do {

        modelDescription =
lfvClient.describeModel(describeModelRequest).modelDescription();

        switch (modelDescription.status()) {

        case HOSTED:
            logger.log(Level.INFO, "Model version {0} for project {1} is
running.",
                    new Object[] { modelVersion, projectName });
            finished = true;
            break;

        case STARTING_HOSTING:
            logger.log(Level.INFO, "Model version {0} for project {1} is
starting.",
                    new Object[] { modelVersion, projectName });

            TimeUnit.SECONDS.sleep(60);

            break;
        case HOSTING_FAILED:
            logger.log(Level.SEVERE, "Hosting failed for model version {0} for
project {1}.",
                    new Object[] { modelVersion, projectName });
            finished = true;
```

```
            break;

        default:
            logger.log(Level.SEVERE, "Unexpected error when hosting model
 version {0} for project {1}: {2}.",
                    new Object[] { projectName, modelVersion,
 modelDescription.status() });
            finished = true;
            break;

        }

    } while (!finished);

    logger.log(Level.INFO, "Finished starting model version {0} for project {1}
 status: {2}",
            new Object[] { modelVersion, projectName,
 modelDescription.statusMessage() });

    return modelDescription;

}
```

3. 코드 출력이 `Model is hosted and ready for use`인 경우 모델을 사용하여 이미지의 이상을 감지할 수 있습니다. 자세한 내용은 이미지에서 이상 탐지 단원을 참조하십시오.

Amazon Lookout for Vision 모델 중지하기

실행 중인 모델을 중지하려면 `StopModel` 작업을 호출하고 다음을 전달합니다.

- 프로젝트 — 중지하려는 모델이 포함된 프로젝트 이름입니다.
- ModelVersion — 중지하려는 모델의 버전입니다.

Amazon Lookout for Vision 콘솔은 모델을 중지하는 데 사용할 수 있는 예제 코드를 제공합니다.

> **Note**
>
> 모델이 실행되는 시간에 따라 요금이 부과됩니다.

주제

모델 중지하기 (콘솔)

콘솔을 사용하여 탑재 대상을 생성하려면 다음 절차의 단계를 수행합니다.

모델을 중지하려면 (콘솔)

1. 아직 설치하지 않은 경우 및 AWS SDKs AWS CLI 를 설치하고 구성합니다. 자세한 내용은 4단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.
2. https://console.aws.amazon.com/lookoutvision/에서 Amazon Lookout for Vision 콘솔을 엽니다.
3. Get started를 선택합니다.
4. 왼쪽 탐색 창에서 프로젝트를 선택합니다.
5. 프로젝트 리소스 페이지에서 중지하려는 실행 중인 모델이 포함된 프로젝트를 선택합니다.
6. 모델 섹션에서 중지하려는 모델을 선택합니다.
7. 모델의 세부 정보 페이지에서 모델 사용을 선택한 다음 클라우드에 API 통합을 선택합니다.
8. AWS CLI 명령에서를 호출하는 AWS CLI 명령을 복사합니다`stop-model`.
9. 명령 프롬프트에서 이전 단계에서 복사한 `stop-model` 명령을 입력합니다. `lookoutvision`프로필을 사용하여 자격 증명을 가져오려면 `--profile lookoutvision-access` 매개 변수를 추가하십시오.
10. 콘솔의 왼쪽 탐색 창에서 모델을 선택합니다.
11. 상태 열에서 모델의 현재 상태를 확인하십시오. 상태 열 값이 학습 완료일 때 모델이 중지된 것입니다.

Amazon Lookout for Vision 모델 (SDK) 중지

StopMode 작업을 호출하여 모델을 중지합니다.

모델을 중지하는 데 시간이 걸릴 수 있습니다. 현재 상태를 확인하려면 `DescribeModel` 항목을 사용하세요.

모델 (SDK) 을 중지하려면

1. 아직 설치하지 않은 경우 및 AWS SDKs AWS CLI 를 설치하고 구성합니다. 자세한 내용은 4단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.
2. 다음 예제 코드를 사용하여 모델 실행을 중지합니다.

   CLI

   다음 값을 변경합니다.

   - `project-name`은 중지하려는 모델이 포함된 프로젝트 이름으로 이동합니다.
   - `model-version`은 중지하려는 모델의 버전으로 이동합니다.

```
aws lookoutvision stop-model --project-name "project name"\
    --model-version model version \
  --profile lookoutvision-access
```

   Python

   이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져옵니다. 전체 예제는 여기에서 확인하세요.

```
    @staticmethod
    def stop_model(lookoutvision_client, project_name, model_version):
        """
        Stops a running Lookout for Vision Model.

        :param lookoutvision_client: A Boto3 Lookout for Vision client.
        :param project_name: The name of the project that contains the version
of
                             the model that you want to stop hosting.
        :param model_version:  The version of the model that you want to stop
hosting.
        """
        try:
            logger.info("Stopping model version %s for %s", model_version,
project_name)
            response = lookoutvision_client.stop_model(
                ProjectName=project_name, ModelVersion=model_version
            )
            logger.info("Stopping hosting...")
```

```
            status = response["Status"]
            finished = False

            # Wait until stopped or failed.
            while finished is False:
                model_description = lookoutvision_client.describe_model(
                    ProjectName=project_name, ModelVersion=model_version
                )
                status = model_description["ModelDescription"]["Status"]

                if status == "STOPPING_HOSTING":
                    logger.info("Host stopping in progress...")
                    time.sleep(10)
                    continue

                if status == "TRAINED":
                    logger.info("Model is no longer hosted.")
                    finished = True
                    continue

                logger.info("Failed to stop model: %s ", status)
                finished = True

            if status != "TRAINED":
                logger.error("Error stopping model: %s", status)
                raise Exception(f"Error stopping model: {status}")
        except ClientError:
            logger.exception("Couldn't stop hosting model.")
            raise
```

Java V2

이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져옵니다. 전체 예제는 여기에서 확인하세요.

```
/**
 * Stops the hosting an Amazon Lookout for Vision model. Returns when model has
 * stopped or if hosting fails.
 *
 * @param lfvClient   An Amazon Lookout for Vision client.
 * @param projectName The name of the project that contains the model that you
```

```
 *                    want to stop hosting.
 * @modelVersion The version of the model that you want to stop hosting.
 * @return ModelDescription The description of the model, which includes the
 *         model hosting status.
 */

public static ModelDescription stopModel(LookoutVisionClient lfvClient, String
 projectName,
                String modelVersion) throws LookoutVisionException,
 InterruptedException {

        logger.log(Level.INFO, "Stopping Model version {0} for project {1}.",
                        new Object[] { modelVersion, projectName });

        StopModelRequest stopModelRequest = StopModelRequest.builder()
                        .projectName(projectName)
                        .modelVersion(modelVersion)
                        .build();

        // Stop hosting the model.

        lfvClient.stopModel(stopModelRequest);

        DescribeModelRequest describeModelRequest =
 DescribeModelRequest.builder()
                        .projectName(projectName)
                        .modelVersion(modelVersion)
                        .build();

        ModelDescription modelDescription = null;

        boolean finished = false;
        // Wait until model is stopped or failure occurs.
        do {

                modelDescription =
 lfvClient.describeModel(describeModelRequest).modelDescription();

                switch (modelDescription.status()) {

                        case TRAINED:
                                logger.log(Level.INFO, "Model version {0} for
 project {1} has stopped.",
```

```
                                                    new Object[] { modelVersion,
projectName });
                                finished = true;
                                break;

                        case STOPPING_HOSTING:
                                logger.log(Level.INFO, "Model version {0} for
project {1} is stopping.",
                                                    new Object[] { modelVersion,
projectName });

                                TimeUnit.SECONDS.sleep(60);

                                break;

                        default:
                                logger.log(Level.SEVERE,
                                                    "Unexpected error when stopping
model version {0} for project {1}: {2}.",
                                                    new Object[] { projectName,
modelVersion,

modelDescription.status() });
                                finished = true;
                                break;

                }

        } while (!finished);

        logger.log(Level.INFO, "Finished stopping model version {0} for project
{1} status: {2}",
                        new Object[] { modelVersion, projectName,
modelDescription.statusMessage() });

        return modelDescription;

}
```

이미지에서 이상 탐지

훈련된 Amazon Lookout for Vision 모델을 사용하여 이미지에서 이상 현상을 탐지하려면 DetectAnomalies 작업을 호출합니다. DetectAnomalies의 결과에는 이미지를 하나 이상의 예외가 포함된 것으로 분류하는 부울 예측과 예측에 대한 신뢰값이 포함됩니다. 모델이 영상 세분화 모델인 경우 결과에는 다양한 유형의 이상 현상의 위치를 보여주는 컬러 마스크도 포함됩니다.

DetectAnomalies에 제공하는 이미지의 너비와 높이 치수는 모델을 학습시키는 데 사용한 이미지와 같아야 합니다.

DetectAnomalies은 이미지를 PNG 또는 JPG 형식 이미지로 받아들입니다. 이미지는 모델 학습에 사용된 것과 동일한 인코딩 및 압축 형식을 사용하는 것이 좋습니다. 예를 들어, PNG 형식 이미지로 모델을 학습시키는 경우 PNG 형식 이미지로 DetectAnomalies을 호출하십시오.

DetectAnomalies을 호출하기 전에 먼저 StartModel 작업과 함께 모델을 시작해야 합니다. 자세한 내용은 Amazon Lookout for Vision 모델 시작하기 단원을 참조하십시오. 모델이 실행되는 시간(분) 과 모델에서 사용하는 이상 감지 단위 수에 따라 요금이 부과됩니다. 모델을 사용하지 않는 경우 StopModel 작업을 사용하여 모델을 중지하십시오. 자세한 내용은 Amazon Lookout for Vision 모델 중지하기 단원을 참조하십시오.

주제

- DetectAnomalies 호출하기
- DetectAnomalies 응답 이해
- 이미지가 변칙적인지 여부 확인
- 분류 및 세분화 정보 표시
- AWS Lambda 함수를 사용하여 이상 찾기

DetectAnomalies 호출하기

DetectAnomalies 항목을 호출하려면 다음을 지정해야 합니다.

- 프로젝트 — 사용하고자 하는 모델이 포함된 프로젝트 이름.
- ModelVersion — 사용하려는 모델의 버전입니다.
- ContentType — 분석하려는 이미지 유형입니다. 유효한 값은 image/png (PNG 형식 이미지) 및 image/jpeg (JPG 형식 이미지) 입니다.

- 본문 — 이미지를 나타내는 인코딩되지 않은 2진 바이트입니다.

  이미지는 모델 학습에 사용된 이미지와 크기가 같아야 합니다.

다음 예에는 DetectAnomalies를 직접적으로 호출하는 방법이 나와 있습니다. Python 및 Java 예제의 함수 응답을 사용하여 이미지가 변칙적인지 여부 확인에서 함수를 호출할 수 있습니다.

AWS CLI

이 AWS CLI 명령은 DetectAnomalies CLI 작업에 대한 JSON 출력을 표시합니다. 다음 입력 파라미터의 값을 변경합니다.

- project name: 사용하려는 프로젝트의 이름
- model version: 사용하려는 모델의 버전
- content type: 사용하려는 이미지 형식 유효한 값은 image/png (PNG 형식 이미지) 및 image/jpeg (JPG 형식 이미지) 입니다.
- file name: 사용하려는 이미지의 경로 및 파일 이름을 입력합니다. 파일 유형이 content-type의 값과 일치하는지 확인하십시오.

```
aws lookoutvision detect-anomalies  --project-name project name\
    --model-version model version\
    --content-type content type\
    --body file name \
    --profile lookoutvision-access
```

Python

전체 코드 예제는 GitHub를 참조하십시오.

```
    def detect_anomalies(lookoutvision_client, project_name, model_version, photo):
        """
        Calls DetectAnomalies using the supplied project, model version, and image.
        :param lookoutvision_client: A Lookout for Vision Boto3 client.
        :param project: The project that contains the model that you want to use.
        :param model_version: The version of the model that you want to use.
        :param photo: The photo that you want to analyze.
        :return: The DetectAnomalyResult object that contains the analysis results.
        """
```

```
        image_type = imghdr.what(photo)
        if image_type == "jpeg":
            content_type = "image/jpeg"
        elif image_type == "png":
            content_type = "image/png"
        else:
            logger.info("Invalid image type for %s", photo)
            raise ValueError(
                f"Invalid file format. Supply a jpeg or png format file: {photo}")


        # Get images bytes for call to detect_anomalies
        with open(photo, "rb") as image:
            response = lookoutvision_client.detect_anomalies(
                ProjectName=project_name,
                ContentType=content_type,
                Body=image.read(),
                ModelVersion=model_version)

        return response['DetectAnomalyResult']
```

Java V2

```
  public static DetectAnomalyResult detectAnomalies(LookoutVisionClient lfvClient,
String projectName,
            String modelVersion,
            String photo) throws IOException, LookoutVisionException {
        /**
         * Creates an Amazon Lookout for Vision dataset from a manifest file.
         * Returns after Lookout for Vision creates the dataset.
         *
         * @param lfvClient    An Amazon Lookout for Vision client.
         * @param projectName  The name of the project in which you want to create a
         *                     dataset.
         * @param modelVersion The version of the model that you want to use.
         *
         * @param photo        The photo that you want to analyze.
         *
         * @return DetectAnomalyResult The analysis result from DetectAnomalies.
         */

        logger.log(Level.INFO, "Processing local file: {0}", photo);
```

```
        // Get image bytes.

        InputStream sourceStream = new FileInputStream(new File(photo));
        SdkBytes imageSDKBytes = SdkBytes.fromInputStream(sourceStream);
        byte[] imageBytes = imageSDKBytes.asByteArray();

        // Get the image type. Can be image/jpeg or image/png.
        String contentType = getImageType(imageBytes);

        // Detect anomalies in the supplied image.
        DetectAnomaliesRequest request =
DetectAnomaliesRequest.builder().projectName(projectName)
                .modelVersion(modelVersion).contentType(contentType).build();

        DetectAnomaliesResponse response = lfvClient.detectAnomalies(request,
                RequestBody.fromBytes(imageBytes));

        /*
         * Tip: You can also use the following to analyze a local file.
         * Path path = Paths.get(photo);
         * DetectAnomaliesResponse response = lfvClient.detectAnomalies(request,
path);
         */
        DetectAnomalyResult result = response.detectAnomalyResult();

        String prediction = "Prediction: Normal";

        if (Boolean.TRUE.equals(result.isAnomalous())) {
            prediction = "Prediction: Anomalous";
        }

        // Convert confidence to percentage.
        NumberFormat defaultFormat = NumberFormat.getPercentInstance();
        defaultFormat.setMinimumFractionDigits(1);
        String confidence = String.format("Confidence: %s",
defaultFormat.format(result.confidence()));

        // Log classification result.
        String photoPath = "File: " + photo;
        String[] imageLines = { photoPath, prediction, confidence };
        logger.log(Level.INFO, "Image: {0}\nAnomalous: {1}\nConfidence {2}",
imageLines);

        return result;
```

```
    }

    // Gets the image mime type. Supported formats are image/jpeg and image/png.
    private static String getImageType(byte[] image) throws IOException {

        InputStream is = new BufferedInputStream(new ByteArrayInputStream(image));
        String mimeType = URLConnection.guessContentTypeFromStream(is);

        logger.log(Level.INFO, "Image type: {0}", mimeType);

        if (mimeType.equals("image/jpeg") || mimeType.equals("image/png")) {
            return mimeType;
        }
        // Not a supported file type.
        logger.log(Level.SEVERE, "Unsupported image type: {0}", mimeType);
        throw new IOException(String.format("Wrong image type. %s format isn't
supported.", mimeType));
    }
```

DetectAnomalies 응답 이해

DetectAnomalies로부터의 응답은 학습시키는 모델의 유형 (분류 모델 또는 분할 모델) 에 따라 달라집니다. 두 경우 모두 응답은 DetectAnomalyResult 객체입니다.

분류 모델

모델이 이미지 분류 모델 인 경우 DetectAnomalies로부터의 응답에는 다음이 포함됩니다.

- IsAnomalous — 이미지에 하나 이상의 예외가 포함되어 있음을 나타내는 부울 표시기입니다.
- 신뢰도 — Amazon Lookout for Vision이 이상 항목 예측의 정확성(IsAnomalous)에 대해 갖고 있다는 확신입니다.Confidence은 0과 1 사이의 부동 소수점 값입니다. 값이 높을수록 신뢰도가 높습니다.
- 소스 — DetectAnomalies에 전달된 이미지에 대한 정보입니다.

```
{
    "DetectAnomalyResult": {
        "Source": {
            "Type": "direct"
```

```
        },
        "IsAnomalous": true,
        "Confidence": 0.9996867775917053
    }
}
```

IsAnomalous필드를 확인하고 Confidence 값이 필요에 맞게 충분히 높은지 확인하여 이미지가 비정상적인지 판단합니다.

DetectAnomalies에서 반환된 신뢰값이 너무 낮다고 생각되면 모델을 재훈련해 보세요. 예제 코드는 분류 항목을 참조하세요.

세분화 모델

모델이 이미지 세분화 모델인 경우 응답에는 분류 정보와 이미지 마스크, 예외 유형 등의 분할 정보가 포함됩니다. 분류 정보는 세분화 정보와 별도로 계산되므로 이들 간의 관계가 있다고 가정해서는 안 됩니다. 응답에서 분할 정보를 가져오지 못하면 최신 버전의 AWS SDK가 설치되어 있는지 확인합니다 (AWS Command Line Interface를 사용하는 경우 AWS CLI). 예제 코드는 세분화 및 분류 및 세분화 정보 표시 단원을 참조하세요.

- iSanomalous (분류) — 이미지를 정상 또는 비정상 이미지로 분류하는 부울 표시기입니다.
- 신뢰도 (분류) — Amazon Lookout for Vision이 이미지 분류의 정확성(IsAnomalous)에 대해 갖고 있다는 확신입니다.Confidence은 0과 1 사이의 부동 소수점 값입니다. 값이 높을수록 신뢰도가 높습니다.
- 소스 — DetectAnomalies에 전달된 이미지에 대한 정보입니다.
- AnomalyMask (세분화) — 분석된 이미지에서 발견된 이상 현상을 덮는 픽셀 마스크입니다. 이미지에 여러 예외가 있을 수 있습니다. 마스크 맵의 색상은 예외 유형을 나타냅니다. 마스크 색상은 훈련 데이터 세트의 예외 유형에 할당된 색상에 매핑됩니다. 마스크 색상에서 예외 유형을 찾으려면 Anomalies 목록에서 반환된 각 예외 항목의 PixelAnomaly 필드의 Color를 확인하세요. 예제 코드는 분류 및 세분화 정보 표시 항목을 참조하세요.
- 예외 항목 (분할) — 이미지에서 발견된 예외 항목 목록입니다. 각 예외 항목에는 예외 유형 (Name)과 픽셀 정보 (PixelAnomaly)가 포함됩니다.TotalPercentageArea은 예외 항목이 포함하는 이미지의 백분율 영역입니다. Color는 예외 항목의 마스크 색상입니다.

  목록의 첫 번째 요소는 항상 이미지 배경 (BACKGROUND)을 나타내는 예외 유형이므로 예외 항목으로 간주해서는 안 됩니다. Amazon Lookout for Vision은 응답에 배경 이상 유형을 자동으로 추가합니다. 데이터세트에서 배경 이상 유형을 선언할 필요 없습니다.

```
{
    "DetectAnomalyResult": {
        "Source": {
            "Type": "direct"
        },
        "IsAnomalous": true,
        "Confidence": 0.9996814727783203,
        "Anomalies": [
            {
                "Name": "background",
                "PixelAnomaly": {
                    "TotalPercentageArea": 0.998999834060669,
                    "Color": "#FFFFFF"
                }
            },
            {
                "Name": "scratch",
                "PixelAnomaly": {
                    "TotalPercentageArea": 0.0004034999874420464,
                    "Color": "#7ED321"
                }
            },
            {
                "Name": "dent",
                "PixelAnomaly": {
                    "TotalPercentageArea": 0.0005966666503809392,
                    "Color": "#4DD8FF"
                }
            }
        ],
        "AnomalyMask": "iVBORw0....."
    }
}
```


이미지가 변칙적인지 여부 확인

다양한 방법으로 이미지가 이상인지 보기 선택하는 방법은 사용 사례와 모델 유형에 따라 다릅니다. 잠재적 해결 방법은 다음과 같습니다.

주제

- 분류

- 세분화

분류

IsAnomalous은 이미지를 변칙으로 분류하고 Confidence 필드를 사용하여 이미지가 실제로 변칙적인지 판단할 수 있습니다. 값이 높을수록 신뢰도가 높아집니다. 예를 들어 신뢰도가 80% 를 초과하는 경우에만 제품에 결함이 있다고 판단할 수 있습니다. 분류 모델 또는 이미지 분할 모델로 분석된 이미지를 분류할 수 있습니다.

Python

전체 코드 예제는 GitHub를 참조하십시오.

```
   def reject_on_classification(image, prediction, confidence_limit):
       """
       Returns True if the anomaly confidence is greater than or equal to
       the supplied confidence limit.
       :param image: The name of the image file that was analyzed.
       :param prediction: The DetectAnomalyResult object returned from
DetectAnomalies
       :param confidence_limit: The minimum acceptable confidence. Float value
between 0 and 1.
       :return: True if the error condition indicates an anomaly, otherwise False.
       """

       reject = False

       logger.info("Checking classification for %s", image)

       if prediction['IsAnomalous'] and prediction['Confidence'] >=
confidence_limit:
           reject = True
           reject_info=(f"Rejected: Anomaly confidence
({prediction['Confidence']:.2%}) is greater"
                   f" than limit ({confidence_limit:.2%})")
           logger.info("%s", reject_info)

       if not reject:
           logger.info("No anomalies found.")
       return reject
```

Java V2

```
   public static boolean rejectOnClassification(String image, DetectAnomalyResult
prediction, float minConfidence) {
       /**
        * Rejects an image based on its anomaly classification and prediction
        * confidence
        *
        * @param image         The file name of the analyzed image.
        * @param prediction    The prediction for an image analyzed with
        *                      DetectAnomalies.
        * @param minConfidence The minimum acceptable confidence for the prediction
        *                      (0-1).
        *
        * @return boolean True if the image is anomalous, otherwise False.
        */

       Boolean reject = false;

       logger.log(Level.INFO, "Checking classification for {0}", image);

       String[] logParameters = { prediction.confidence().toString(),
String.valueOf(minConfidence) };

       if (Boolean.TRUE.equals(prediction.isAnomalous()) && prediction.confidence()
>= minConfidence) {
           logger.log(Level.INFO, "Rejected: Anomaly confidence {0} is greater than
confidence limit {1}",
                   logParameters);
           reject = true;
       }
       if (Boolean.FALSE.equals(reject))
           logger.log(Level.INFO, ": No anomalies found.");

       return reject;

   }
```

세분화

모델이 이미지 세분화 모델인 경우 세분화 정보를 사용하여 이미지에 이상이 있는지 확인할 수 있습니다. 이미지 세분화 모델을 사용하여 이미지를 분류할 수도 있습니다. 이미지 마스크를 가져와서 표시하는 예제 코드는 분류 및 세분화 정보 표시을 참조하십시오.

이상 영역

이미지에 있는 예외 항목의 백분율 범위 (TotalPercentageArea) 를 사용하십시오. 예를 들어, 이상 영역이 이미지의 1% 보다 크면 제품에 결함이 있다고 판단할 수 있습니다.

Python

전체 코드 예제는 GitHub를 참조하십시오.

```
   def reject_on_coverage(image, prediction, confidence_limit, anomaly_label,
coverage_limit):
       """
       Checks if the coverage area of an anomaly is greater than the coverage limit
and if
       the prediction confidence is greater than the confidence limit.
       :param image: The name of the image file that was analyzed.
       :param prediction: The DetectAnomalyResult object returned from
DetectAnomalies
       :param confidence_limit: The minimum acceptable confidence (float 0-1).
       :anomaly_label: The anomaly label for the type of anomaly that you want to
check.
       :coverage_limit: The maximum acceptable percentage coverage of an anomaly
(float 0-1).
       :return: True if the error condition indicates an anomaly, otherwise False.
       """

       reject = False

       logger.info("Checking coverage for %s", image)

       if prediction['IsAnomalous'] and prediction['Confidence'] >=
confidence_limit:
           for anomaly in prediction['Anomalies']:
               if (anomaly['Name'] == anomaly_label and
                       anomaly['PixelAnomaly']['TotalPercentageArea'] >
(coverage_limit)):
                   reject = True
```

```
                    reject_info=(f"Rejected: Anomaly confidence
 ({prediction['Confidence']:.2%}) "
                        f"is greater than limit ({confidence_limit:.2%}) and
 {anomaly['Name']} "
                        f"coverage ({anomaly['PixelAnomaly']
['TotalPercentageArea']:.2%}) "
                        f"is greater than limit ({coverage_limit:.2%})")

                    logger.info("%s", reject_info)

        if not reject:
            logger.info("No anomalies found.")

        return reject
```

Java V2

```
   public static Boolean rejectOnCoverage(String image, DetectAnomalyResult
 prediction, float minConfidence,
            String anomalyType, float maxCoverage) {
        /**
         * Rejects an image based on a maximum allowable coverage area for an
 anomaly
         * type.
         *
         * @param image         The file name of the analyzed image.
         * @param prediction    The prediction for an image analyzed with
         *                      DetectAnomalies.
         * @param minConfidence The minimum acceptable confidence for the prediction
         *                      (0-1).
         * @param anomalyTypes  The anomaly type to check.
         * @param maxCoverage   The maximum allowable coverage area of the anomaly
 type.
         *                      (0-1).
         *
         * @return boolean True if the coverage area of the anomaly type exceeds the
         *         maximum allowed, otherwise False.
         */

        Boolean reject = false;

        logger.log(Level.INFO, "Checking coverage for {0}", image);
```

```
       if (Boolean.TRUE.equals(prediction.isAnomalous()) && prediction.confidence()
>= minConfidence) {
           for (Anomaly anomaly : prediction.anomalies()) {

               if (Objects.equals(anomaly.name(), anomalyType)
                       && anomaly.pixelAnomaly().totalPercentageArea() >=
maxCoverage) {

                   String[] logParameters = { prediction.confidence().toString(),
                           String.valueOf(minConfidence),

String.valueOf(anomaly.pixelAnomaly().totalPercentageArea()),
                           String.valueOf(maxCoverage) };
                   logger.log(Level.INFO,
                           "Rejected: Anomaly confidence {0} is greater than
confidence limit {1} and " +
                                   "{2} anomaly type coverage is higher than
coverage limit {3}\n",
                           logParameters);
                   reject = true;

               }
           }
       }

       if (Boolean.FALSE.equals(reject))
           logger.log(Level.INFO, ": No anomalies found.");

       return reject;
   }
```

예외 유형 수

이미지에 있는 다양한 예외 유형 수 (Name)를 사용하십시오. 예를 들어, 두 가지 이상의 이상 유형이 있는 경우 제품에 결함이 있다고 판단할 수 있습니다.

Python

전체 코드 예제는 GitHub를 참조하십시오.

```
   def reject_on_anomaly_types(image, prediction, confidence_limit,
anomaly_types_limit):
       """
       Checks if the number of anomaly types is greater than than the anomaly types
       limit and if the prediction confidence is greater than the confidence limit.
       :param image: The name of the image file that was analyzed.
       :param prediction: The DetectAnomalyResult object returned from
DetectAnomalies
       :param confidence: The minimum acceptable confidence. Float value between 0
and 1.
       :param anomaly_types_limit: The maximum number of allowable anomaly types
(Integer).
       :return: True if the error condition indicates an anomaly, otherwise False.
       """

       logger.info("Checking number of anomaly types for %s",image)

       reject = False

       if prediction['IsAnomalous'] and prediction['Confidence'] >=
confidence_limit:

           anomaly_types = {anomaly['Name'] for anomaly in prediction['Anomalies']\
               if anomaly['Name'] != 'background'}

           if len (anomaly_types) > anomaly_types_limit:
                   reject = True
                   reject_info = (f"Rejected: Anomaly confidence
({prediction['Confidence']:.2%}) "
                   f"is greater than limit ({confidence_limit:.2%}) and "
                   f"the number of anomaly types ({len(anomaly_types)-1}) is "
                   f"greater than the limit ({anomaly_types_limit})")

                   logger.info("%s", reject_info)

       if not reject:
           logger.info("No anomalies found.")
       return reject
```

Java V2

```
   public static Boolean rejectOnAnomalyTypeCount(String image, DetectAnomalyResult
prediction,
```

```
            float minConfidence, Integer maxAnomalyTypes) {

        /**
         * Rejects an image based on a maximum allowable number of anomaly types.
         *
         * @param image           The file name of the analyzed image.
         * @param prediction      The prediction for an image analyzed with
         *                        DetectAnomalies.
         * @param minConfidence   The minimum acceptable confidence for the
predictio
         *                        (0-1).
         * @param maxAnomalyTypes The maximum allowable number of anomaly types.
         *
         * @return boolean True if the image contains more than the maximum allowed
         *         anomaly types, otherwise False.
         */

        Boolean reject = false;

        logger.log(Level.INFO, "Checking coverage for {0}", image);

        Set<String> defectTypes = new HashSet<>();

        if (Boolean.TRUE.equals(prediction.isAnomalous()) && prediction.confidence()
>= minConfidence) {
            for (Anomaly anomaly : prediction.anomalies()) {
                defectTypes.add(anomaly.name());
            }
            // Reduce defect types by one to account for 'background' anomaly type.
            if ((defectTypes.size() - 1) > maxAnomalyTypes) {
                String[] logParameters = { prediction.confidence().toString(),
                        String.valueOf(minConfidence),
                        String.valueOf(defectTypes.size()),
                        String.valueOf(maxAnomalyTypes) };
                logger.log(Level.INFO, "Rejected: Anomaly confidence {0} is >=
minimum confidence {1} and " +
                        "the number of anomaly types {2} > the allowable number of
anomaly types {3}\n", logParameters);
                reject = true;
            }

        }

        if (Boolean.FALSE.equals(reject))
```

```
            logger.log(Level.INFO, ": No anomalies found.");

        return reject;
    }
```

분류 및 세분화 정보 표시

이 예제는 분석된 이미지를 보여주고 분석 결과를 오버레이합니다. 응답에 예외 항목 마스크가 포함된 경우 마스크는 관련 예외 유형의 색상으로 표시됩니다.

이미지 분류 및 이미지 세분화 정보를 표시하려면

1. 아직 수행하지 않은 경우 다음 작업을 수행하십시오.
   a. 아직 설치하지 않은 경우 및 AWS SDKs AWS CLI 를 설치하고 구성합니다. 자세한 내용은 4 단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.
   b. 모델 훈련
   c. 모델을 시작합니다.
2. DetectAnomalies을 호출한 사용자가 사용하려는 모델 버전에 액세스할 수 있는지 확인하세요. 자세한 내용은 SDK 권한 설정 단원을 참조하십시오.
3. 다음 코드를 사용합니다.

   Python

   다음 예제 코드는 제공한 이미지에서 이상을 감지합니다. 명령줄에서 다음 옵션이 허용됩니다.

   - project: 사용하려는 프로젝트의 이름
   - version— 프로젝트 내에서 사용하고자 하는 모델 버전.
   - image— 로컬 이미지 파일 (JPEG 또는 PNG 형식)의 경로 및 파일.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
"""
Shows how to detect and show anomalies in an image using a trained Amazon
 Lookout
for Vision model. The script displays the analysed image and overlays mask and
 analysis
output.
```

```
"""

import argparse
import logging
import io
import boto3
from PIL import Image, ImageDraw, ImageFont

from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)


class ShowAnomalies:
    """
    Class to detect and show anomalies in an image analyzed by detect_anomalies.
    """



    @staticmethod
    def draw_line(draw, text, fnt,  y_coordinate, color):
        """
        Draws a line of text on the supplied drawing surface.
        :param draw: The surface on which to draw the text.
        :param text: The text to draw in the drawing surface.
        :param fnt: The font for the text.
        :param y_coordinate: The y position for the text.
        :param color: The color for the text.
        :returns The y coordinate for the next line of text.
        """
        text_width, text_height = draw.textsize(text, fnt)
        draw.rectangle([(10, y_coordinate), (text_width + 10,
                                             y_coordinate + text_height)],
 fill="black")
        draw.text((10, y_coordinate), text, fill=color, font=fnt)

        y_coordinate += text_height

        return y_coordinate

    @staticmethod
    def draw_analysis_text(image, analysis):
        """
        Draws classification and segmentation info onto supplied image
```

```
        overlay analysis results on an image analyzed by detect_anomalies.
        :param analysis: The response from a call to detect_anomalies.
        :returns Nothing
        """

        ## Calculate a reasonable font size based on image width.
        font_size = int(image.size[0]/32)

        fnt = ImageFont.truetype('/Library/Fonts/Tahoma.ttf', font_size)

        draw = ImageDraw.Draw(image)

        y_coordinate = 0

        # Draw classification information.
        prediction = "Anomalous" if analysis["DetectAnomalyResult"]
["IsAnomalous"] \
            else "Normal"

        confidence = analysis["DetectAnomalyResult"]["Confidence"]
        found_anomalies = analysis["DetectAnomalyResult"]['Anomalies']
        segmentation_info = False

        logger.info("Prediction: %s", format(prediction))
        logger.info("Confidence: %s", format(confidence))

        y_coordinate = 0
        y_coordinate = ShowAnomalies.draw_line(
            draw, "Classification", fnt, y_coordinate, "white")
        y_coordinate = ShowAnomalies.draw_line(
            draw, f"  Prediction: {prediction}", fnt, y_coordinate, "white")
        y_coordinate = ShowAnomalies.draw_line(
            draw, f"  Confidence: {confidence:.2%}", fnt, y_coordinate, "white")

        # Draw segmentation information, if present.
        if (len(found_anomalies)) > 1:
            logger.info("Anomalies:")

            y_coordinate = ShowAnomalies.draw_line(
                draw, "Segmentation:", fnt, y_coordinate, "white")
            for i in range(1, len(found_anomalies)):

                # Only display info if more than 0% coverage found.
```

```
                percent_coverage = found_anomalies[i]['PixelAnomaly']
['TotalPercentageArea']
                if percent_coverage > 0:
                    segmentation_info = True
                    logger.info("  %s", found_anomalies[i]['Name'])
                    logger.info("    Color: %s",
                                found_anomalies[i]['PixelAnomaly']['Color'])
                    logger.info("    Area: %s", percent_coverage)
                    y_coordinate = ShowAnomalies.draw_line(
                        draw,
                        f"  Anomaly: {found_anomalies[i]['Name']}. Area:
 {percent_coverage:.2%}",
                        fnt,
                        y_coordinate,
                        found_anomalies[i]['PixelAnomaly']['Color'])

            if not segmentation_info:
                y_coordinate = ShowAnomalies.draw_line(
                    draw, "No segmentation information found.", fnt,
 y_coordinate, "white")




    @staticmethod
    def show_anomaly_prediction(lookoutvision_client, project_name,
 model_version, photo):
        """
        Detects anomalies in an image (jpg/png) by using your Amazon Lookout for
Vision
        model. Displays the image and overlays prediction information text.
        :param lookoutvision_client: An Amazon Lookout for Vision Boto3 client.
        :param project_name: The name of the project that contains the model
that
        you want to use.
        :param model_version: The version of the model that you want to use.
        :param photo: The path and name of the image in which you want to detect
        anomalies.
        """
        try:

            logger.info("Detecting anomalies in %s", photo)

            image = Image.open(photo)
```

```
            image_type = Image.MIME[image.format]

            # Check that image type is valid.
            if image_type not in ("image/jpeg", "image/png"):
                logger.info("Invalid image type for %s", photo)
                raise ValueError(
                    f"Invalid file format. Supply a jpeg or png format file:
 {photo}"
                )

            # Get images bytes for call to detect_anomalies.
            image_bytes = io.BytesIO()
            image.save(image_bytes, format=image.format)
            image_bytes = image_bytes.getvalue()

            # Analyze the image.
            response = lookoutvision_client.detect_anomalies(
                ProjectName=project_name,
                ContentType=image_type,
                Body=image_bytes,
                ModelVersion=model_version
            )

            # Overlay mask onto analyzed image.
            image_mask_bytes = response["DetectAnomalyResult"]["AnomalyMask"]
            image_mask = Image.open(io.BytesIO(image_mask_bytes))

            final_img = Image.blend(image, image_mask, 0.5) \
                if response["DetectAnomalyResult"]["IsAnomalous"] else image

            # Overlay analysis output on image.
            ShowAnomalies.draw_analysis_text(final_img, response)

            final_img.show()

        except ClientError as err:
            logger.info(format(err))
            raise


def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
```

```
    """

    parser.add_argument(
        "project", help="The project containing the model that you want to use."
    )
    parser.add_argument(
        "version", help="The version of the model that you want to use."
    )
    parser.add_argument(
        "image",
        help="The file that you want to analyze. "
        "Supply a local file path.",
    )


def main():
    """
    Entrypoint for anomaly detection example.
    """

    try:
        logging.basicConfig(level=logging.INFO,
                            format="%(levelname)s: %(message)s")

        session = boto3.Session(
            profile_name='lookoutvision-access')

        lookoutvision_client = session.client("lookoutvision")

        parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)

        add_arguments(parser)

        args = parser.parse_args()

        # Analyze the image and show results.
        ShowAnomalies.show_anomaly_prediction(
            lookoutvision_client, args.project, args.version, args.image
        )

    except ClientError as err:
        print("A service error occured: " +
              format(err.response["Error"]["Message"]))
    except FileNotFoundError as err:
```

```
        print("The supplied file couldn't be found: " + err.filename)
    except ValueError as err:
        print("A value error occured. " + format(err))
    else:
        print("Successfully completed analysis.")


if __name__ == "__main__":
    main()
```

Java 2

다음 예제 코드는 제공한 이미지에서 이상을 감지합니다. 명령줄에서 다음 옵션이 허용됩니다.

- `project`: 사용하려는 프로젝트의 이름
- `version`— 프로젝트 내에서 사용하고자 하는 모델 버전.
- `image`— 로컬 이미지 파일 (JPEG 또는 PNG 형식)의 경로 및 파일.

```
/*
   Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
   SPDX-License-Identifier: Apache-2.0
*/


package com.example.lookoutvision;

import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.core.sync.RequestBody;
import software.amazon.awssdk.services.lookoutvision.LookoutVisionClient;
import software.amazon.awssdk.services.lookoutvision.model.Anomaly;
import
 software.amazon.awssdk.services.lookoutvision.model.DetectAnomaliesRequest;
import
 software.amazon.awssdk.services.lookoutvision.model.DetectAnomaliesResponse;
import software.amazon.awssdk.services.lookoutvision.model.DetectAnomalyResult;
import
 software.amazon.awssdk.services.lookoutvision.model.LookoutVisionException;

import java.io.BufferedInputStream;
import java.io.ByteArrayInputStream;
```

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.net.URLConnection;

import java.text.NumberFormat;
import java.awt.*;
import java.awt.font.LineMetrics;
import java.awt.image.BufferedImage;
import javax.imageio.ImageIO;
import javax.swing.*;

import java.util.logging.Level;
import java.util.logging.Logger;

// Finds anomalies on a supplied image.
public class ShowAnomalies extends JPanel {
/**
 * Finds and displays anomalies on a supplied image.
 */

    private static final long serialVersionUID = 1L;
    private transient BufferedImage image;
    private transient BufferedImage maskImage;
    private transient Dimension dimension;
    public static final Logger logger =
 Logger.getLogger(ShowAnomalies.class.getName());

    // Constructor. Finds anomalies in a local image file.
    public ShowAnomalies(LookoutVisionClient lfvClient, String projectName,
 String modelVersion,
            String photo) throws IOException, LookoutVisionException {

        logger.log(Level.INFO, "Processing local file: {0}", photo);

        maskImage = null;

        // Get image bytes and buffered image.
        InputStream sourceStream = new FileInputStream(new File(photo));
        SdkBytes imageSDKBytes = SdkBytes.fromInputStream(sourceStream);
        byte[] imageBytes = imageSDKBytes.asByteArray();
```

```
        ByteArrayInputStream inputStream = new
ByteArrayInputStream(imageSDKBytes.asByteArray());
        image = ImageIO.read(inputStream);

        // Get the image type. Can be image/jpeg or image/png.
        String contentType = getImageType(imageBytes);

        // Set the size of the window that shows the image.
        setWindowDimensions();

        // Detect anomalies in the supplied image.
        DetectAnomaliesRequest request =
DetectAnomaliesRequest.builder().projectName(projectName)
                .modelVersion(modelVersion).contentType(contentType).build();

        DetectAnomaliesResponse response = lfvClient.detectAnomalies(request,
                RequestBody.fromBytes(imageBytes));

        /*
         * Tip: You can also use the following to analyze a local file.
         * Path path = Paths.get(photo);
         * DetectAnomaliesResponse response = lfvClient.detectAnomalies(request,
path);
         */
        DetectAnomalyResult result = response.detectAnomalyResult();


        if (result.anomalyMask() != null){
            SdkBytes maskSDKBytes = result.anomalyMask();

            ByteArrayInputStream maskInputStream = new
ByteArrayInputStream(maskSDKBytes.asByteArray());
            maskImage = ImageIO.read(maskInputStream);
        }

        drawImageInfo(result);

    }

    // Sets window dimensions to 1/2 screen size, unless image is smaller.
    public void setWindowDimensions() {
        dimension = java.awt.Toolkit.getDefaultToolkit().getScreenSize();

        dimension.width = (int) dimension.getWidth() / 2;
```

```
        dimension.height = (int) dimension.getHeight() / 2;

        if (image.getWidth() < dimension.width || image.getHeight() <
dimension.height) {
            dimension.width = image.getWidth();
            dimension.height = image.getHeight();
        }
        setPreferredSize(dimension);

    }

    private int drawLine(Graphics2D g2d, String line, FontMetrics metrics, int
yPos, Color color) {
    /**
    * Draws a line of text at the spsecified y position and color.
    * confidence
    *
    * @param g2D The Graphics2D object for the image.
    * @param line The line of text to draw.
    * @param metrics The font information to use.
    * @param yPos The y position for the line of text.
    *
    * @return  The yPos for the next line of text.
    */

        int indent = 10;

        // Get text height, width, and descent.
        int textWidth = metrics.stringWidth(line);
        LineMetrics lm = metrics.getLineMetrics(line, g2d);
        int textHeight = (int) lm.getHeight();
        int descent = (int) lm.getDescent();

        int y2Pos = (yPos + textHeight) - descent;

        // Draw black rectangle.
        g2d.setColor(Color.BLACK);
        g2d.fillRect(indent, yPos, textWidth, textHeight);

        // Draw text.
        g2d.setColor(color);
        g2d.drawString(line, indent, y2Pos);

        yPos += textHeight;
```

```
        return yPos;

    }

    public void drawImageInfo(DetectAnomalyResult result) {
    /**
     * Draws the results from DetectAnomalies onto the output image.
     *
     * @param result The response from a call to
     *               DetectAnomalies.
     *
     */

        // Set up drawing.
        Graphics2D g2d = image.createGraphics();

        if (result.anomalyMask() != null){
            Composite composite = g2d.getComposite();
            g2d.setComposite(AlphaComposite.SrcOver.derive(0.5f));
            int x = (image.getWidth() - maskImage.getWidth()) / 2;
            int y = (image.getHeight() - maskImage.getHeight()) / 2;
            g2d.drawImage(maskImage, x, y, null);
            // Set composite for overlaying text.
            g2d.setComposite(composite);
        }

        //Calculate font size based on arbitary 32 pixel image width.
        int fontSize = (image.getWidth() / 32);


        g2d.setFont(new Font("Tahoma", Font.PLAIN, fontSize));
        Font font = g2d.getFont();
        FontMetrics metrics = g2d.getFontMetrics(font);

        // Get classification information.

        String prediction = "Prediction: Normal";

        if (Boolean.TRUE.equals(result.isAnomalous())) {
            prediction = "Prediction: Anomalous";
        }

        // Convert prediction to percentage.
```

```
        NumberFormat defaultFormat = NumberFormat.getPercentInstance();
        defaultFormat.setMinimumFractionDigits(1);
        String confidence = String.format("Confidence: %s",
defaultFormat.format(result.confidence()));

        // Draw classification information.
        int yPos = 0;

        yPos = drawLine(g2d, "Classification:", metrics, yPos, Color.WHITE);
        yPos = drawLine(g2d, prediction, metrics, yPos, Color.WHITE);
        yPos = drawLine(g2d, confidence, metrics, yPos, Color.WHITE);

        // Draw segmentation info.
        yPos = drawLine(g2d, "Segmentation:", metrics, yPos, Color.WHITE);

        // Ignore background label, so size must be > 1
        if (result.anomalies().size() > 1) {
            for (Anomaly anomaly : result.anomalies()) {
                if (anomaly.name().equals("background"))
                    continue;
                String label = String.format("Anomaly: %s. Area: %s",
anomaly.name(),

defaultFormat.format(anomaly.pixelAnomaly().totalPercentageArea()));
                Color anomalyColor =
Color.decode((anomaly.pixelAnomaly().color()));
                yPos = drawLine(g2d, label, metrics, yPos, anomalyColor);

            }

        } else {
            drawLine(g2d, "None found.", metrics, yPos, Color.WHITE);

        }

        g2d.dispose();

    }

    @Override
    public void paintComponent(Graphics g)
    /**
     * Draws the image and analysis results.
     *
```

```
     * @param g The Graphics context object for drawing.
     *          DetectAnomalies.
     *
     */
    {

        Graphics2D g2d = (Graphics2D) g; // Create a Java2D version of g.

        // Draw the image.
        g2d.drawImage(image, 0, 0, dimension.width, dimension.height, this);

    }

    // Gets the image mime type. Supported formats are image/jpeg and image/png.

    private String getImageType(byte[] image) throws IOException
    /**
     * Gets the file type of a supplied image. Raises an exception if the image
     * isn't compatible with with Amazon Lookout for Vision.
     *
     * @param image The image that you want to check.
     *
     * @return String The type of the image.
     */

    {
        InputStream is = new BufferedInputStream(new
ByteArrayInputStream(image));
        String mimeType = URLConnection.guessContentTypeFromStream(is);

        logger.log(Level.INFO, "Image type: {0}", mimeType);

        if (mimeType.equals("image/jpeg") || mimeType.equals("image/png")) {
            return mimeType;
        }
        // Not a supported file type.
        logger.log(Level.SEVERE, "Unsupported image type: {0}", mimeType);
        throw new IOException(String.format("Wrong image type. %s format isn't
supported.", mimeType));
    }

    public static void main(String[] args) throws Exception {

        String photo = null;
```

```
        String projectName = null;
        String modelVersion = null;

        final String USAGE = "\n" +
                "Usage:\n" +
                "    DetectAnomalies <project> <version> <image> \n\n" +
                "Where:\n" +
                "    project - The Lookout for Vision project.\n\n" +
                "    version - The version of the model within the project.\n\n"
+
                "    image - The path and filename of a local image. \n\n";

        try {

            if (args.length != 3) {
                System.out.println(USAGE);
                System.exit(1);
            }

            projectName = args[0];
            modelVersion = args[1];
            photo = args[2];
            ShowAnomalies panel = null;

            // Get the Lookout for Vision client.
            LookoutVisionClient lfvClient = LookoutVisionClient.builder()

.credentialsProvider(ProfileCredentialsProvider.create("lookoutvision-access"))
                .build();

            // Create frame and panel.
            JFrame frame = new JFrame(photo);
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

            panel = new ShowAnomalies(lfvClient, projectName, modelVersion,
photo);

            frame.setContentPane(panel);
            frame.pack();
            frame.setVisible(true);


        } catch (LookoutVisionException lfvError) {
```

```
            logger.log(Level.SEVERE, "Lookout for Vision client error: {0}:
 {1}",
                    new Object[] { lfvError.awsErrorDetails().errorCode(),
                            lfvError.awsErrorDetails().errorMessage() });
            System.out.println(String.format("lookout for vision client error:
 %s", lfvError.getMessage()));
            System.exit(1);

        } catch (FileNotFoundException fileError) {
            logger.log(Level.SEVERE, "Could not find file: {0}",
 fileError.getMessage());
            System.out.println(String.format("Could not find file: %s",
 fileError.getMessage()));
            System.exit(1);

        } catch (IOException ioError) {
            logger.log(Level.SEVERE, "IO error {0}", ioError.getMessage());
            System.out.println(String.format("IO error: %s",
 ioError.getMessage()));
            System.exit(1);
        }

    }
}
```

4. 모델을 계속 사용할 계획이 없다면 모델을 중지하세요.

AWS Lambda 함수를 사용하여 이상 찾기

AWS Lambda 는 서버를 프로비저닝하거나 관리하지 않고도 코드를 실행할 수 있는 컴퓨팅 서비스입니다. 예를 들어 애플리케이션 코드를 호스팅할 서버를 만들지 않고도 모바일 애플리케이션에서 제출된 이미지를 분석할 수 있습니다. 다음 지침은 DetectCustomLabels를 호출하는 Lambda 함수를 Python에서 생성하는 방법을 보여줍니다. 이 함수는 제공된 이미지를 분석하고 해당 이미지에 이상이 있는지 분류하여 반환합니다. 지침에는 Amazon S3 버킷의 이미지 또는 로컬 컴퓨터에서 제공된 이미지를 사용하여 Lambda 함수를 호출하는 방법을 보여주는 예제 Python 코드가 포함되어 있습니다.

주제

- 1단계: AWS Lambda 함수 생성(콘솔)
- 2단계: (선택 사항) 계층 생성 (콘솔)

- 3단계: Python 코드 추가 (콘솔)
- 4단계: Lambda 함수 테스트

1단계: AWS Lambda 함수 생성(콘솔)

이 단계에서는 AWS 함수가 `DetectAnomalies` 작업을 호출할 수 있도록 빈 함수와 IAM 실행 역할을 생성합니다. 또한 분석용 이미지를 저장하는 Amazon S3 버킷에 대한 액세스 권한도 부여합니다. 또한 다음에 대한 환경 변수를 지정합니다.

- Lambda 함수에서 사용하려는 Amazon Lookout for Vision 프로젝트 및 모델 버전.
- 모델에서 사용하려는 신뢰 한도.

나중에 Lambda 함수에 소스 코드와 계층(선택 사항)을 추가합니다.

AWS Lambda 함수를 생성하려면(콘솔)

1. 에 로그인 AWS Management Console 하고 https://console.aws.amazon.com/lambda/ AWS Lambda 콘솔을 엽니다.
2. 함수 생성(Create function)을 선택합니다. 자세한 내용은 콘솔로 Lambda 함수 생성을 참조하세요.
3. 다음과 같은 옵션을 선택하세요.
   - 새로 작성을 선택합니다.
   - 함수 이름의 값을 입력합니다.
   - 런타임에는 Python 3.10을 선택합니다.
4. 함수 생성을 선택하여 AWS Lambda 함수를 생성합니다.
5. 함수 페이지에서 구성 탭을 선택합니다.
6. 환경 변수 창에서 편집을 선택합니다.
7. 다음 환경 변수를 추가합니다. 각 변수에 대해 환경 변수 추가를 선택한 다음 변수 키와 값을 입력합니다.

키	값
PROJECT_NAME	사용하려는 모델이 들어 있는 Lookout for Vision 프로젝트입니다.

키	값
MODEL_VERSION	사용하려는 모델의 버전입니다.
CONFIDENCE	예측이 비정상적이라는 모델 신뢰도의 최소값 (0-100) 입니다. 신뢰도가 낮으면 분류가 정상으로 간주됩니다.

8. 저장을 선택하여 환경 변수를 저장합니다.
9. 권한 창의 실행 역할에서 IAM 콘솔의 역할을 열 실행 역할을 선택합니다.
10. 권한 탭에서 권한 추가, 그다음 인라인 정책 생성을 선택합니다.
11. JSON을 선택하고 기존 정책을 다음 정책으로 바꿉니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": "lookoutvision:DetectAnomalies",
            "Resource": "*",
            "Effect": "Allow",
            "Sid": "DetectAnomaliesAccess"
        }
    ]
}
```

12. Next(다음)를 선택합니다.
13. 정책 세부 정보에 정책 이름 (예: DetectAnomalies-access)을 입력합니다.
14. 정책 생성을 선택합니다.
15. Amazon S3 버킷에 분석용 이미지를 저장하는 경우 10~14단계를 반복합니다.

    a. 11단계에서는 다음 정책을 사용하십시오. *##/## ##*를 Amazon S3 버킷 및 분석하려는 이미지의 폴더 경로로 바꿉니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "S3Access",
            "Effect": "Allow",
            "Action": "s3:GetObject",
```

```
            "Resource": "arn:aws:s3:::bucket/folder path/*"
        }
    ]
}
```

b. 13단계에서는 S3Bucket-access와 같은 다른 정책 이름을 선택합니다.

2단계: (선택 사항) 계층 생성 (콘솔)

이 단계는 예제를 실행하는 데 필수 사항이 아닙니다. `DetectAnomalies` 작업은 AWS SDK for Python(Boto3)의 일부로 기본 Lambda Python 환경에 포함됩니다. Lambda 함수의 다른 부분에 기본 Lambda Python 환경에 없는 최신 AWS 서비스 업데이트가 필요한 경우이 단계를 수행하여 최신 Boto3 SDK 릴리스를 함수에 계층으로 추가합니다.

먼저 Boto3 SDK가 포함된 .zip 파일 아카이브를 생성합니다. 그런 다음 계층을 생성하고 .zip 파일 아카이브를 계층에 추가합니다. 자세한 내용은 Lambda 함수에서 계층 사용을 참조하세요.

계층을 생성하고 추가하려면(콘솔)

1. 명령 프롬프트를 열고 다음 명령을 입력합니다.

```
pip install boto3 --target python/.
zip boto3-layer.zip -r python/
```

2. zip 파일(boto3-layer.zip)의 이름을 기록해 두세요. 이 절차의 6단계에서 필요합니다.
3. https://console.aws.amazon.com/lambda/ AWS Lambda 콘솔을 엽니다.
4. 탐색 창에서 계층을 선택합니다.
5. 계층 생성을 선택합니다.
6. Name(이름)과 Description(설명)을 입력합니다.
7. .zip 파일 업로드를 선택한 후 업로드를 선택합니다.
8. 대화 상자에서 이 절차의 1단계에서 만든.zip 파일 아카이브 (boto3-layer.zip)를 선택합니다.
9. 호환되는 런타임은 Python 3.9를 선택하세요.
10. 생성을 선택하여 계층을 생성합니다.
11. 탐색 창 메뉴 아이콘을 선택합니다.
12. 탐색 창에서 함수를 선택합니다.
13. 리소스 목록에서 1단계: AWS Lambda 함수 생성(콘솔) 항목에서 생성한 함수를 선택합니다.

14. 코드 탭을 선택합니다.
15. 계층 영역에서 계층 추가를 선택합니다.
16. 사용자 지정 계층을 선택합니다.
17. 사용자 지정 계층에서 6단계에서 입력한 계층 이름을 선택합니다.
18. 버전에서 계층 버전을 선택합니다. 계층 버전은 1이어야 합니다.
19. 추가를 선택합니다.

3단계: Python 코드 추가 (콘솔)

이 단계에서는 Lambda 콘솔 코드 편집기를 사용하여 Lambda 함수에 Python 코드를 추가합니다. 코드는 제공된 이미지를 `DetectAnomalies`로 분석하여 분류를 반환합니다 (이미지가 비정상이면 true, 이미지가 정상이면 false). 제공된 이미지는 Amazon S3 버킷에 있거나 byte64로 인코딩된 이미지 바이트로 제공될 수 있습니다.

Python 코드를 추가하려면 (콘솔)

1. Lambda 콘솔을 사용하지 않는 경우 다음을 수행합니다.
   a. https://console.aws.amazon.com/lambda/ AWS Lambda 콘솔을 엽니다.
   b. 1단계: AWS Lambda 함수 생성(콘솔)에서 생성한 Lambda 함수를 엽니다.
2. 코드 탭을 선택합니다.
3. 코드 소스에서 lambda_function.py의 코드를 다음과 같이 바꾸세요.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

"""
Purpose
An AWS lambda function that analyzes images with an Amazon Lookout for Vision
 model.
"""
import base64
import imghdr
from os import environ
from io import BytesIO
import logging
import boto3
```

```
from botocore.exceptions import ClientError


logger = logging.getLogger(__name__)


# Get the model and confidence.
project_name = environ['PROJECT_NAME']
model_version = environ['MODEL_VERSION']
min_confidence = int(environ.get('CONFIDENCE', 50))

lookoutvision_client = boto3.client('lookoutvision')


def lambda_handler(event, context):
    """
    Lambda handler function
    param: event: The event object for the Lambda function.
    param: context: The context object for the lambda function.
    return: The labels found in the image passed in the event
    object.
    """

    try:

        file_name = ""

        # Determine image source.
        if 'image' in event:
            # Decode the encoded image
            image_bytes = event['image'].encode('utf-8')
            img_b64decoded = base64.b64decode(image_bytes)
            image_type = get_image_type(img_b64decoded)
            image = BytesIO(img_b64decoded)
            file_name = event['filename']

        elif 'S3Object' in event:
            bucket = boto3.resource('s3').Bucket(event['S3Object']['Bucket'])
            image_object = bucket.Object(event['S3Object']['Name'])
            image = image_object.get().get('Body').read()
            image_type = get_image_type(image)
            file_name = f"s3://{event['S3Object']['Bucket']}/{event['S3Object']
['Name']}"
```

```
        else:
            raise ValueError(
                'Invalid image source. Only base 64 encoded image bytes or images
in S3 buckets are supported.')

        # Analyze the image.
        response = lookoutvision_client.detect_anomalies(
            ProjectName=project_name,
            ContentType=image_type,
            Body=image,
            ModelVersion=model_version)

        reject = reject_on_classification(
            response['DetectAnomalyResult'],
confidence_limit=float(environ['CONFIDENCE'])/100)

        status = "anomalous" if reject else "normal"

        lambda_response = {
            "statusCode": 200,
            "body": {
                "Reject": reject,
                "RejectMessage": f"Image {file_name} is {status}."
            }
        }

    except ClientError as err:
        error_message = f"Couldn't analyze {file_name}. " + \
            err.response['Error']['Message']

        lambda_response = {
            'statusCode': 400,
            'body': {
                "Error": err.response['Error']['Code'],
                "ErrorMessage": error_message,
                "Image": file_name
            }
        }
        logger.error("Error function %s: %s",
                     context.invoked_function_arn, error_message)

    except ValueError as val_error:

        lambda_response = {
```

```
            'statusCode': 400,
            'body': {
                "Error": "ValueError",
                "ErrorMessage": format(val_error),
                "Image": event['filename']
            }
        }
        logger.error("Error function %s: %s",
                     context.invoked_function_arn, format(val_error))

    return lambda_response

def get_image_type(image):
    """
    Gets the format of the image. Raises an error
    if the type is not PNG or JPEG.
    :param image: The image that you want to check.
    :return The type of the image.

    """
    image_type = imghdr.what(None, image)

    if image_type == "jpeg":
        content_type = "image/jpeg"
    elif image_type == "png":
        content_type = "image/png"
    else:
        logger.info("Invalid image type")
        raise ValueError(
            "Invalid file format. Supply a jpeg or png format file.")
    return content_type


def reject_on_classification(prediction, confidence_limit):
    """
    Returns True if the anomaly confidence is greater than or equal to
    the supplied confidence limit.
    :param image: The name of the image file that was analyzed.
    :param prediction: The DetectAnomalyResult object returned from DetectAnomalies
    :param confidence_limit: The minimum acceptable confidence. Float value between
 0 and 1.
    :return: True if the error condition indicates an anomaly, otherwise False.
    """
```

```
    reject = False

    if prediction['IsAnomalous'] and prediction['Confidence'] >= confidence_limit:
        reject = True
        reject_info = (f"Rejected: Anomaly confidence
({prediction['Confidence']:.2%}) is greater"
                       f" than limit ({confidence_limit:.2%})")
        logger.info("%s", reject_info)

    if not reject:
        logger.info("No anomalies found.")
    return reject
```

4. Lambda 함수를 배포하려면 배포를 선택합니다.

4단계: Lambda 함수 테스트

이 단계에서는 컴퓨터의 Python 코드를 사용하여 로컬 이미지 또는 Amazon S3 버킷의 이미지를 Lambda 함수에 전달합니다. 로컬 컴퓨터에서 전달된 이미지는 6291456바이트보다 작아야 합니다. 이미지가 더 크면 Amazon S3 버킷에 이미지를 업로드하고 이미지에 대한 Amazon S3 경로를 사용하여 스크립트를 호출합니다. Amazon S3 버킷에 이미지 파일을 업로드하는 방법에 대한 자세한 내용은 객체 업로드를 참조하세요.

Lambda 함수를 생성한 리전과 동일한 AWS 리전에서 코드를 실행해야 합니다. Lambda 콘솔의 함수 세부 정보 페이지의 탐색 모음에서 Lambda 함수의 AWS 리전을 볼 수 있습니다.

AWS Lambda 함수가 제한 시간 오류를 반환하는 경우 Lambda 함수의 제한 시간을 연장합니다. 자세한 내용은 함수 제한 시간 구성(콘솔)을 참조하세요.

코드에서 Lambda 함수를 호출하는 방법에 대한 자세한 내용은 AWS Lambda 함수 호출을 참조하세요.

Lambda 함수를 테스트하는 방법

1. 아직 수행하지 않은 경우 다음 작업을 수행하십시오.

   a. 클라이언트 코드를 사용하는 사용자에게 `lambda:InvokeFunction` 권한이 있는지 확인하십시오. 다음 권한을 사용할 수 있습니다.

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [
        {
            "Sid": "LambdaPermission",
            "Effect": "Allow",
            "Action": "lambda:InvokeFunction",
            "Resource": "ARN for lambda function"
        }
    ]
}
```

Lambda 콘솔의 함수 개요에서 Lambda 함수 함수에 대한 ARN을 가져올 수 있습니다.

액세스 권한을 제공하려면 사용자, 그룹 또는 역할에 권한을 추가하세요:

- 의 사용자 및 그룹 AWS IAM Identity Center:

  권한 세트를 생성합니다. AWS IAM Identity Center 사용 설명서의 권한 세트 생성의 지침을 따릅니다.

- 보안 인증 공급자를 통해 IAM에서 관리되는 사용자:

  ID 페더레이션을 위한 역할을 생성합니다. IAM 사용 설명서의 Create a role for a third-party identity provider (federation)의 지침을 따릅니다.

- IAM 사용자:
  - 사용자가 맡을 수 있는 역할을 생성합니다. IAM 사용 설명서에서 Create a role for an IAM user의 지침을 따릅니다.
  - (권장되지 않음)정책을 사용자에게 직접 연결하거나 사용자를 사용자 그룹에 추가합니다. IAM 사용 설명서에서 사용자(콘솔)에 권한 추가의 지침을 따르세요.

b. AWS SDK for Python을 설치하고 구성합니다. 자세한 내용은 4단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.

c. 1단계: AWS Lambda 함수 생성(콘솔)의 7단계에서 지정한 모델을 시작합니다.

2. 다음 코드를 `client.py` 이름의 파일에 저장합니다.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

"""
Purpose: Shows how to call the anomaly detection
AWS Lambda function.
"""
```

```
from botocore.exceptions import ClientError

import argparse
import logging
import base64
import json
import boto3
from os import environ

logger = logging.getLogger(__name__)

def analyze_image(function_name, image):
    """
    Analyzes an image with an AWS Lambda function.
    :param image: The image that you want to analyze.
    :return The status and classification result for
    the image analysis.
    """

    lambda_client = boto3.client('lambda')

    lambda_payload = {}

    if image.startswith('s3://'):
        logger.info("Analyzing image from S3 bucket: %s", image)
        bucket, key = image.replace("s3://", "").split("/", 1)
        s3_object = {
            'Bucket': bucket,
            'Name': key
        }
        lambda_payload = {"S3Object": s3_object}

    # Call the lambda function with the image.
    else:
        with open(image, 'rb') as image_file:
            logger.info("Analyzing local image image: %s ", image)
            image_bytes = image_file.read()
            data = base64.b64encode(image_bytes).decode("utf8")
            lambda_payload = {"image": data, "filename": image}

    response = lambda_client.invoke(FunctionName=function_name,
                                    Payload=json.dumps(lambda_payload))
    return json.loads(response['Payload'].read().decode())
```

```
def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """

    parser.add_argument(
        "function", help="The name of the AWS Lambda function "
        "that you want to use to analyze the image.")
    parser.add_argument(
        "image", help="The local image that you want to analyze.")


def main():
    """
    Entrypoint for script.
    """
    try:
        logging.basicConfig(level=logging.INFO,
                            format="%(levelname)s: %(message)s")

        # Get command line arguments.
        parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
        add_arguments(parser)
        args = parser.parse_args()

        # Analyze image and display results.

        result = analyze_image(args.function, args.image)

        status = result['statusCode']

        if status == 200:
            classification = result['body']
            print(f"classification: {classification['Reject']}")
            print(f"Message: {classification['RejectMessage']}")
        else:
            print(f"Error: {result['statusCode']}")
            print(f"Message: {result['body']}")

    except ClientError as error:
        logging.error(error)
        print(error)
```

```
if __name__ == "__main__":
    main()
```

3. 코드를 실행합니다. 명령줄 인수의 경우 Lambda 함수 이름과 분석하려는 로컬 이미지의 경로를 제공하십시오. 예시:

```
python client.py function_name /bucket/path/image.jpg
```

성공하면 이미지에서 발견된 예외 항목에 대한 분류가 출력됩니다. 분류가 반환되지 않는 경우 1단계: AWS Lambda 함수 생성(콘솔)의 7단계에서 설정한 신뢰도를 낮추는 것을 고려해 보십시오.

4. Lambda 함수 사용을 마쳤지만 다른 애플리케이션에서 해당 모델을 사용하지 않는 경우 모델을 중지하세요. 다음에 Lambda 함수를 사용하려는 경우 모델을 시작해야 한다는 점을 기억하십시오.

엣지 디바이스에서 Amazon Lookout for Vision 모델 사용하기

에서 관리하는 엣지 디바이스에서 Amazon Lookout for Vision 모델을 사용할 수 있습니다 AWS IoT Greengrass Version 2. AWS IoT Greengrass는 오픈 소스 사물 인터넷 (IoT) 엣지 런타임 및 클라우드 서비스입니다. 이를 사용하여 디바이스에서 IoT 애플리케이션을 구축, 배포 및 관리할 수 있습니다. 자세한 내용은 AWS IoT Greengrass 단원을 참조하십시오.

클라우드에서 학습시킨 것과 동일한 Amazon Lookout for Vision 모델을 AWS IoT Greengrass V2 호환 가능한 엣지 디바이스에 배포합니다. 그런 다음 데이터를 클라우드로 계속 스트리밍하지 않고도 배포된 모델을 사용하여 공장 현장과 같은 온프레미스에서 이상 탐지를 수행할 수 있습니다. 이렇게 하면 실시간 이미지 분석을 통해 대역폭 비용을 최소화하고 로컬에서 이상 현상을 탐지할 수 있습니다.

> **Tip**
>
> 를 사용하여 Lookout for Vision 모델을 배포하기 전에 AWS IoT Greengrass Version 2 개발자 안내서를 읽는 AWS IoT Greengrass것이 좋습니다. 자세한 내용은 AWS IoT Greengrass란?을 참조하세요.

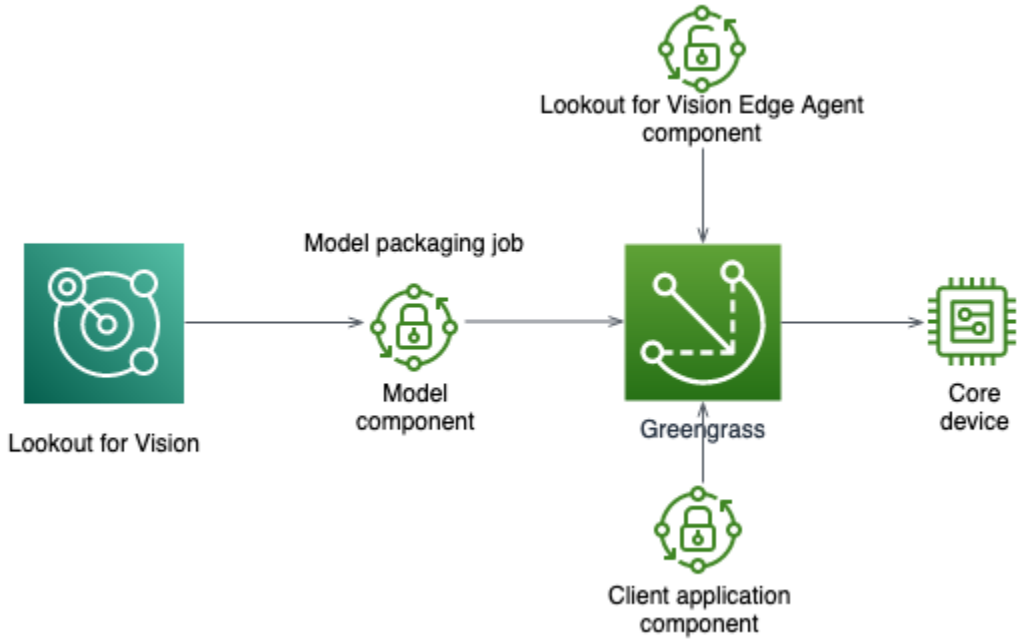
AWS IoT Greengrass V2 코어 디바이스에서 Lookout for Vision 모델을 사용하려면 모델 및 지원 소프트웨어를 코어 디바이스에 구성 요소로 배포합니다. 구성 요소는 Greengrass 코어 디바이스에서 실행되는 Lookout for Vision 모델과 같은 소프트웨어 모듈입니다. 컴포넌트에는 두 가지 형태가 있습니다. 사용자 지정 구성 요소는 사용자가 만든 구성 요소이며 사용자만 액세스할 수 있습니다. 이를 개인 구성 요소라고도 합니다. AWS 제공된 구성 요소는가 AWS 제공하는 사전 빌드된 구성 요소입니다. 공용 구성 요소라고도 합니다. 자세한 내용은 https://docs.aws.amazon.com/greengrass/v2/developerguide/public-components.html 단원을 참조하십시오.

Lookout for Vision 모델 및 지원 소프트웨어의 코어 디바이스에 배포하는 구성 요소는 다음과 같습니다.

- 모델 구성 요소. Lookout for Vision 모델을 포함하는 사용자 지정 구성 요소입니다. 모델 구성 요소를 만들려면 Lookout for Vision을 사용하여 모델 패키징 작업을 생성합니다. 모델 패키징 작업은 모델에 대한 구성 요소를 생성하고 이를 내에서 사용자 지정 구성 요소로 사용할 수 있도록 합니다 AWS IoT Greengrass V2. 자세한 내용은 Amazon Lookout for Vision 모델 패키징 단원을 참조하십시오.

- 클라이언트 애플리케이션 구성 요소. 비즈니스 요구 사항에 맞는 코드를 구현하는 사용자 지정 구성 요소입니다. 조립 후 촬영한 이미지에서 비정상적인 회로 기판을 찾는 경우를 예로 들 수 있습니다. 자세한 내용은 클라이언트 애플리케이션 구성 요소 작성 단원을 참조하십시오.
- Amazon Lookout for Vision 엣지 에이전트 구성 요소. 모델 사용 및 관리를 위한 API를 제공하는 AWS 제공된 구성 요소입니다. 예를 들어 클라이언트 애플리케이션 구성 요소의 코드는 `DetectAnomalies` API를 사용하여 이미지의 이상을 탐지할 수 있습니다. Lookout for Vision 엣지 에이전트 구성 요소는 모델 구성 요소의 종속성입니다. 모델 구성 요소를 배포하면 코어 디바이스에 자동으로 설치됩니다. 자세한 내용은 Amazon Lookout for Vision Edge Agent API 참조 단원을 참조하십시오.

모델 구성 요소 및 클라이언트 애플리케이션 구성 요소를 생성한 후 AWS IoT Greengrass V2 를 사용하여 구성 요소 및 종속성을 코어 디바이스에 배포할 수 있습니다. 자세한 내용은 디바이스에 구성 요소 배포 단원을 참조하십시오.



Important

모델이 코어 디바이스에서 수행하는 예측은 `DetectAnomalies` 클라우드에 호스팅된 동일한 모델을 사용하여 수행한 예측과 다를 수 있습니다. 프로덕션 환경에서 모델을 사용하기 전에 코어 디바이스에서 테스트하는 것이 좋습니다.

기기 호스팅 모델과 클라우드 호스팅 모델 간의 예측 불일치를 줄이려면 학습 데이터 세트의 정상 및 비정상 이미지 수를 늘리는 것이 좋습니다. 학습 데이터세트의 크기를 늘리기 위해 기존 이미지를 재사용하지 않는 것이 좋습니다.

AWS IoT Greengrass Version 2 코어 디바이스에 모델 및 클라이언트 애플리케이션 구성 요소 배포

AWS IoT Greengrass Version 2 코어 디바이스에 Amazon Lookout for Vision 모델 및 클라이언트 애플리케이션 구성 요소를 배포하는 절차는 다음과 같습니다.

1. 를 사용하여 코어 디바이스를 설정합니다 AWS IoT Greengrass Version 2.
2. Lookout for Vision을 사용하여 모델 패키징 작업을 생성하십시오. 이 작업을 통해 모델 구성 요소가 생성됩니다.
3. 클라이언트 애플리케이션 구성 요소를 작성하세요. 구성 요소는 비즈니스 로직을 구현합니다.
4. 를 사용하여 모델 구성 요소와 클라이언트 애플리케이션 구성 요소를 코어 디바이스에 배포합니다 AWS IoT Greengrass V2.

구성 요소와 종속성을 코어 디바이스에 배포한 후에는 코어 디바이스에서 모델을 사용할 수 있습니다.

> **Note**
>
> Lookout for Vision 모델 및 클라이언트 애플리케이션 구성 요소를 생성하고 배포하려면 동일한 AWS 리전 및 AWS 계정을 사용해야 합니다.

AWS IoT Greengrass Version 2 코어 디바이스 요구 사항

AWS IoT Greengrass Version 2 코어 디바이스에서 Amazon Lookout for Vision 모델을 사용하려면 모델에 코어 디바이스의 다양한 요구 사항이 있습니다.

주제

- 테스트를 거친 디바이스, 칩 아키텍처 및 운영 체제
- 코어 디바이스 메모리 및 스토리지
- 필수 소프트웨어

테스트를 거친 디바이스, 칩 아키텍처 및 운영 체제

Amazon Lookout for Vision은 다음 하드웨어에서 작동할 것으로 예상됩니다.

- CPU 아키텍처입니다.
  - X86_64 (x86 명령어 세트의 64비트 버전)
  - Arch64 (ARMv8 64비트 CPU)
- (GPU 가속 추론 전용) 충분한 메모리 용량을 갖춘 NVIDIA GPU 가속기 (실행 중인 모델의 경우 최소 6.0GB)

Amazon Lookout for Vision 팀은 다음 디바이스, 칩 아키텍처 및 운영 체제에서 Lookout for Vision 모델을 테스트했습니다.

디바이스

장치	운영 체제	아키텍처	액셀러레이터	컴파일러 옵션

jetson_xavier (NVIDIA® Jetson AGX Xavier)	Linux	Aarch64	NVIDIA	`{"gpu-code": "sm_72", "trt-ver": "7.1.3", "cuda-ver": "10.2"}`
				
				`{"gpu-code": "sm_72", "trt-ver": "8.2.1", "cuda-ver": "10.2"}`

g4dn.xlarge (NVIDIA T4 Tensor Core GPU가 탑재된 EC2 인스턴스 (G4))	Linux	X86_64/X86-64	NVIDIA	`{"gpu-code": "sm_75", "trt-ver": "7.1.3", "cuda-ver": "10.2"}`

장치	운영 체제	아키텍처	액셀러레이터	컴파일러 옵션
g5.xlarge (NVIDIA T4 Tensor Core GPU가 탑재된 EC2 인스턴스 (G5))	Linux	X86_64/X86-64	NVIDIA	`{"gpu-code": "sm_80", "trt-ver": "8.2.0", "cuda-ver": "11.2"}`
c5.2xlarge (Amazon EC2 C5 인스턴스)	Linux	X86_64/X86-64	CPU	`{"mcpu": "core-avx2"}`

코어 디바이스 메모리 및 스토리지

단일 모델과 Amazon Lookout for Vision 엣지 에이전트를 실행하려면 코어 디바이스에 다음과 같은 메모리 및 스토리지 요구 사항이 있어야 합니다. 클라이언트 애플리케이션 구성 요소에 더 많은 메모리와 스토리지가 필요할 수 있습니다.

- 스토리지 — 최소 1.5GB
- 메모리 - 실행 중인 모델의 경우 최소 6.0GB

필수 소프트웨어

코어 디바이스에는 다음 소프트웨어가 필요합니다.

Jetson 디바이스

코어 디바이스가 Jetson 디바이스인 경우 코어 디바이스에 다음 소프트웨어를 설치해야 합니다.

소프트웨어	지원되는 버전
Jetpack SDK	4.4에서 4.6.1
Lookout for Vision Edge Agent 버전 1.x를 위한 Python 및 Python 가상 환경	3.8 또는 3.9

X86 하드웨어

코어 디바이스에서 x86 하드웨어를 사용하는 경우 코어 디바이스에 다음 소프트웨어를 설치해야 합니다.

CPU 추론

소프트웨어	지원되는 버전
Lookout for Vision Edge Agent 버전 1.x를 위한 Python 및 Python 가상 환경	3.8 또는 3.9

GPU 가속 추론

소프트웨어 버전은 사용하는 NVIDIA GPU의 마이크로아키텍처에 따라 달라집니다.

Ampere 이전의 마이크로아키텍처를 지원하는 NVIDIA GPU (컴퓨팅 성능은 8.0 미만)

Ampere 이전의 마이크로아키텍처 (컴퓨팅 성능이 8.0 미만) 를 갖춘 NVIDIA GPU에 필요한 소프트웨어입니다. `gpu-code`값은 `sm_80`보다 작아야 합니다.

소프트웨어	지원되는 버전
NVIDIA CUDA	10.2
NVIDIA TensorRT	7.1.3 이상 8.0.0 미만
Lookout for Vision Edge Agent 버전 1.x를 위한 Python 및 Python 가상 환경	3.8 또는 3.9

Ampere 마이크로아키텍처를 지원하는 NVIDIA GPU (컴퓨팅 성능 8.0)

Ampere 마이크로아키텍처 (컴퓨팅 성능은 8.0)를 지원하는 NVIDIA GPU에 필요한 소프트웨어입니다. `gpu-code`이 `sm_80`이어야 합니다.

소프트웨어	지원되는 버전
NVIDIA CUDA	11.2

소프트웨어	지원되는 버전
NVIDIA TensorRT	8.2.0
Lookout for Vision Edge Agent 버전 1.x를 위한 Python 및 Python 가상 환경	3.8 또는 3.9

AWS IoT Greengrass Version 2 코어 디바이스 설정

Amazon Lookout for Vision은 AWS IoT Greengrass Version 2 를 사용하여 모델 구성 요소, Amazon Lookout for Vision Edge Agent 구성 요소 및 클라이언트 애플리케이션 구성 요소를 AWS IoT Greengrass V2 코어 디바이스에 간단하게 배포할 수 있습니다. 사용할 수 있는 디바이스 및 하드웨어에 대한 자세한 내용은 AWS IoT Greengrass Version 2 코어 디바이스 요구 사항 단원을 참조하세요.

코어 디바이스 설정

다음 정보를 사용하여 코어 디바이스를 설정합니다.

코어 디바이스를 설정하려면

1. GPU 라이브러리를 설정합니다. GPU 가속 추론을 사용하지 않는 경우에는 이 단계를 수행하지 마세요.

   a. CUDA를 지원하는 GPU가 있는지 확인하세요. 자세한 내용은 CUDA-Capable GPU가 있는지 확인을 참조하십시오.

   b. 다음 중 하나를 수행하여 기기에 CUDA, CuDNN 및 TensorRT를 설정합니다.

      - Jetson 디바이스를 사용하는 경우, JetPack 버전 4.4 - 4.6.1을 설치하세요. 자세한 내용은 JetPack 아카이브를 참조하십시오.
      - x86 기반 하드웨어를 사용하고 NVIDIA GPU 마이크로아키텍처가 Ampere 이전 버전인 경우 (컴퓨팅 성능이 8.0 미만), 다음을 수행하십시오.
        1. Linux용 NVIDIA CUDA 설치 가이드의 지침에 따라 CUDA 버전 10.2를 설정합니다.
        2. NVIDIA CuDNN 설명서의 지침에 따라 cuDNN을 설치합니다.
        3. NVIDIA TENSORRT 설명서의 지침에 따라 TensorRT (버전 7.1.3 이상, 8.0.0 이전)를 설정합니다.
      - x86 기반 하드웨어를 사용하고 있고 NVIDIA GPU 마이크로아키텍처가 Ampere (컴퓨팅 성능은 8.0)인 경우 다음을 수행하십시오.

    1. Linux용 NVIDIA CUDA 설치 가이드의 지침에 따라 CUDA 버전 11.2를 설정합니다.
    2. NVIDIA CuDNN 설명서의 지침에 따라 cuDNN을 설치합니다.
    3. NVIDIA TENSORRT 설명서의 지침에 따라 TensorRT (버전 8.2.0)를 설정합니다.
2. AWS IoT Greengrass Version 2 코어 디바이스에 코어 소프트웨어를 설치합니다. 자세한 내용은 AWS IoT Greengrass Version 2 개발자 안내서에서 AWS IoT Greengrass Core 소프트웨어 설치를 참조하십시오.
3. 모델을 저장하는 Amazon S3 버킷에서 읽으려면 AWS IoT Greengrass Version 2 설정 중에 생성하는 IAM 역할(토큰 교환 역할)에 권한을 연결합니다. 자세한 내용은 구성 요소 아티팩트에 대한 S3 버킷 액세스 허용을 참조하십시오.
4. 명령 프롬프트에서 다음 명령을 입력하여 Python과 Python 가상 환경을 코어 장치에 설치합니다.

```
sudo apt install python3.8 python3-venv python3.8-venv
```

5. 다음 명령을 사용하여 Greengrass 사용자를 비디오 그룹에 추가합니다. 이렇게 하면 Greengrass가 배포한 구성 요소가 GPU에 액세스할 수 있습니다.

```
sudo usermod -a -G video ggc_user
```

6. (선택 사항) 다른 사용자로부터 Lookout for Vision Edge Agent API를 호출하려면 필요한 사용자를 `ggc_group`에 추가합니다. 이를 통해 사용자는 Unix 도메인 소켓을 통해 Lookout for Vision 엣지 에이전트와 통신할 수 있습니다.

```
sudo usermod -a -G ggc_group $(whoami)
```

Amazon Lookout for Vision 모델 패키징

모델 패키징 작업은 Amazon Lookout for Vision 모델을 모델 구성 요소로 패키징합니다.

모델 패키징 작업을 생성하려면 패키징할 모델을 선택하고 작업에서 생성하는 모델 구성 요소에 대한 설정을 제공합니다. 성공적으로 학습된 모델만 패키징할 수 있습니다.

Lookout for Vision 콘솔 또는 AWS SDK를 사용하여 모델 패키징 작업을 생성할 수 있습니다. 생성한 모델 패키징 작업에 대한 정보도 얻을 수 있습니다. 자세한 내용은 모델 패키징 작업에 대한 정보 가져오기 단원을 참조하십시오. AWS IoT Greengrass V2 콘솔 또는 AWS SDK를 사용하여 AWS IoT Greengrass Version 2 코어 디바이스에 구성 요소를 배포할 수 있습니다.

주제

패키지 설정

다음 정보를 사용하여 모델 패키징 작업의 패키지 설정을 결정하십시오.

모델 패키징 작업을 생성하려면 모델 패키징 (콘솔) 또는 모델 패키징 (SDK)을 참조하십시오.

주제

대상 하드웨어

모델에 맞는 대상 디바이스 또는 대상 플랫폼을 선택할 수 있지만 둘 다 선택할 수는 없습니다. 자세한 내용은 테스트를 거친 디바이스, 칩 아키텍처 및 운영 체제 단원을 참조하십시오.

대상 디바이스

모델의 대상 디바이스 (예: NVIDIA® Jetson AGX Xavier). 컴파일러 옵션을 지정할 필요가 없습니다.

대상 플랫폼

Amazon Lookout for Vision은 다음과 같은 플랫폼 구성을 지원합니다.

- X86_64 (x86 명령어 세트의 64비트 버전) 및 Aarch64 (ARMv8 64비트 CPU) 아키텍처.
- Linux 운영 체제
- NVIDIA 또는 CPU 액셀러레이터를 사용한 추론.

대상 플랫폼에 맞는 컴파일러 옵션을 지정해야 합니다.

컴파일러 옵션

컴파일러 옵션을 사용하면 AWS IoT Greengrass Version 2 코어 디바이스의 대상 플랫폼을 지정할 수 있습니다. 다음 명령 옵션을 지정할 수 있습니다.

NVIDIA 액셀러레이터

- `gpu-code`— 모델 구성 요소를 실행하는 코어 디바이스의 GPU 코드를 지정합니다.
- `trt-ver`— TensorRT 버전을 x.y.z. 형식으로 지정합니다.
- `cuda-ver`— CUDA 버전을 x.y 형식으로 지정합니다.

CPU 액셀러레이터

- (선택 사항) `mcpu` — 명령 세트를 지정합니다. 예: `core-avx2`. 값을 제공하지 않으면 Lookout for Vision은 값`core-avx2`을 사용합니다.

JSON 형식으로 옵션을 지정합니다. 예시:

```
{"gpu-code": "sm_75", "trt-ver": "7.1.3", "cuda-ver": "10.2"}
```

더 많은 예시는 테스트를 거친 디바이스, 칩 아키텍처 및 운영 체제를 참조합니다.

구성 요소 설정

모델 패키징 작업은 모델을 포함하는 모델 컴포넌트를 생성합니다. 이 작업은가 모델 구성 요소를 코어 디바이스에 배포하는 데 AWS IoT Greengrass V2 사용하는 아티팩트를 생성합니다.

기존 구성 요소와 동일한 구성 요소 이름 및 구성 요소 버전을 사용하여 모델 구성 요소를 만들 수 없습니다.

구성 요소 이름

Lookout for Vision이 모델 패키징 중에 생성하는 모델 구성 요소의 이름입니다. 지정한 구성 요소 이름이 AWS IoT Greengrass V2 콘솔에 표시됩니다. 클라이언트 애플리케이션 구성 요소용으로 만든 레시피의 구성 요소 이름을 사용합니다. 자세한 내용은 클라이언트 애플리케이션 구성 요소 생성 단원을 참조하십시오.

구성 요소 설명

(선택 사항) 모델 구성 요소에 대한 설명입니다.

구성 요소 버전입니다.

모델 구성 요소의 버전 번호입니다. 기본 버전 번호를 그대로 사용하거나 직접 선택할 수 있습니다. 버전 번호는 의미 체계 버전 번호 체계인 major.minor.patch를 따라야 합니다. 예를 들어 1.0.0 버전은 구

성 요소의 첫 번째 주요 릴리스입니다. 자세한 내용을 알아보려면 의미 체계 버전 관리 2.0.0을 참조하세요. 값을 제공하지 않으면 Lookout for Vision은 모델의 버전 번호를 사용하여 버전을 생성합니다.

구성 요소 위치

모델 패키징 작업에서 모델 구성 요소 아티팩트를 저장할 Amazon S3 위치입니다. Amazon S3 버킷은 AWS IoT Greengrass Version 2을 사용하는 AWS 리전 및 AWS 계정과 같은 AWS 리전에 있어야 합니다. Amazon S3 버킷 생성에 대한 자세한 내용은 버킷 생성을 참조하세요.

Tags

태그를 사용하여 구성 요소를 식별, 구성, 검색 및 필터링할 수 있습니다. 각 태그는 사용자 정의 키와 값으로 구성된 레이블입니다. 모델 패키징 작업이 Greengrass에서 모델 컴포넌트를 생성할 때 태그가 모델 컴포넌트에 첨부됩니다. 구성 요소는 AWS IoT Greengrass V2 리소스입니다. 태그는 모델과 같은 Lookout for Vision 리소스에 첨부되지 않습니다. 자세한 내용은 AWS 리소스에 태그 지정 단원을 참조하세요.

모델 패키징 (콘솔)

Amazon Lookout for Vision 콘솔을 사용하여 모델 패키징 작업을 생성할 수 있습니다.

패키징 설정에 대한 내용은 패키지 설정 단원을 참조하세요.

모델 패키징을 하려면(콘솔)

1. Lookout for Vision에서 패키징 작업 아티팩트 (모델 구성 요소) 를 저장하는 데 사용하는 Amazon S3 버킷을 생성하거나 기존 버킷을 재사용합니다.
2. https://console.aws.amazon.com/lookoutvision/에서 Amazon Lookout for Vision 콘솔을 엽니다.
3. Get started를 선택합니다.
4. 왼쪽 탐색 창에서 프로젝트를 선택합니다.
5. 프로젝트 섹션에서 패키징하려는 모델이 포함된 프로젝트를 선택합니다.
6. 왼쪽 탐색 창의 프로젝트 이름에서 엣지 모델 패키지를 선택합니다.
7. 모델 패키징 작업 섹션에서 모델 패키징 작업 생성을 선택합니다.
8. 패키지 설정을 입력합니다. 자세한 내용은 패키지 설정 단원을 참조하십시오.
9. 모델 패키지 생성을 선택합니다.
10. 패키징 작업이 완료될 때까지 기다리십시오. 작업 상태가 성공이면 작업이 완료됩니다.
11. 모델 패키징 작업 섹션에서 패키징 작업을 선택합니다.

12. AWS IoT Greengrass Version 2에서 모델 구성 요소를 계속 배포하려면 Greengrass에서 배포 계속을 선택합니다. 자세한 내용은 디바이스에 구성 요소 배포 단원을 참조하십시오.

모델 패키징 (SDK)

모델 패키징 작업을 생성하여 모델을 모델 구성 요소로 패키징합니다. 모델 패키징 작업을 생성하려면 StartModelPackagingJob API를 호출합니다. 완료하는 데 시간이 걸릴 수 있습니다. 현재 상태를 확인하려면 DescribeModelPackagingJob을 호출하고 응답의 `Status` 필드를 확인하십시오.

패키징 설정에 대한 내용은 패키지 설정 단원을 참조하세요.

다음 절차에서는 AWS CLI를 사용하여 패키징 작업을 시작하는 방법을 보여줍니다. 대상 플랫폼 또는 대상 디바이스용으로 모델을 패키징할 수 있습니다. Java 코드의 예는 StartModelPackagingJob 참조하십시오.

모델 (SDK)을 패키징하려면

1. 아직 설치하지 않은 경우 AWS CLI 및 AWS SDKs를 설치하고 구성합니다. 자세한 내용은 4단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.
2. 모델 패키징 작업을 시작할 수 있는 올바른 권한이 있는지 확인해야 합니다. 자세한 내용은 StartModelPackagingJob를 참조하십시오.
3. 다음 CLI 명령을 사용하여 대상 디바이스 또는 대상 플랫폼에 맞게 모델을 패키징할 수 있습니다.

   Target platform

   다음 CLI 명령은 NVIDIA 액셀러레이터를 사용하여 대상 플랫폼용 모델을 패키징하는 방법을 보여줍니다.

   다음 값을 변경합니다.

   - `project_name`은 패키징하려는 모델이 포함된 프로젝트 이름으로 바꿉니다.
   - `model_version`은 패키징하려는 모델의 버전으로 바꿉니다.
   - (선택 사항) `description`은 모델 패키징 작업에 대한 설명으로 바꿉니다.
   - `architecture` 모델 구성 요소를 실행하는 AWS IoT Greengrass Version 2 코어 디바이스의 아키텍처(`ARM64` 또는 `X86_64`)로.
   - `gpu_code`은 모델 컴포넌트를 실행하는 코어 디바이스의 GPU 코드로 변경합니다.
   - `trt_ver`은 코어 디바이스에 설치한 TensorRT 버전으로 변경합니다.
   - `cuda_ver`은 코어 디바이스에 설치한 CUDA 버전으로 변경합니다.

- component_name를 생성하려는 모델 구성 요소의 이름으로 바꿉니다 AWS IoT Greengrass V2.
- (선택 사항) component_version은 패키징 작업에서 생성되는 모델 구성 요소의 버전으로 변경합니다. major.minor.patch 형식을 사용합니다. 예를 들어 1.0.0은 구성 요소의 첫 번째 주요 릴리스입니다.
- bucket은 패키징 작업이 모델 구성 요소 아티팩트를 저장하는 Amazon S3 버킷으로 변경합니다.
- prefix은 패키징 작업이 모델 구성 요소 아티팩트를 저장하는 Amazon S3 버킷 내 위치로 변경합니다.
- (선택 사항) component_description은 모델 구성 요소 설명으로 변경합니다.
- (선택 사항) tag_key1 및 tag_key2는 모델 컴포넌트에 첨부된 태그의 키로 변경합니다.
- (선택 사항) tag_value1 및 tag_value2는 모델 컴포넌트에 첨부된 태그의 키 값으로 변경합니다.

```
aws lookoutvision start-model-packaging-job \
  --project-name project_name \
  --model-version model_version \
  --description="description" \
  --configuration
 "Greengrass={TargetPlatform={Os='LINUX',Arch='architecture',Accelerator='NVIDIA'},Compi
code\": \"gpu_code\", \"trt-ver\": \"trt_ver\", \"cuda-ver\":
 \"cuda_ver\"}',S3OutputLocation={Bucket='bucket',Prefix='prefix'},ComponentName='Compon
 {Key='tag_key2',Value='tag_value2'}]}" \
  --profile lookoutvision-access
```

예시:

```
aws lookoutvision start-model-packaging-job \
  --project-name test-project-01 \
  --model-version 1 \
  --description="Model Packaging Job for G4 Instance using TargetPlatform
 Option" \
  --configuration
 "Greengrass={TargetPlatform={Os='LINUX',Arch='X86_64',Accelerator='NVIDIA'},CompilerOpt
code\": \"sm_75\", \"trt-ver\": \"7.1.3\", \"cuda-ver\":
 \"10.2\"}',S3OutputLocation={Bucket='bucket',Prefix='test-project-01/
folder'},ComponentName='SampleComponentNameX86TargetPlatform',ComponentVersion='0.1.0',C
```

```
is my component',Tags=[{Key='modelKey0',Value='modelValue'},
{Key='modelKey1',Value='modelValue'}]}" \
 --profile lookoutvision-access
```

Target Device

다음 CLI 명령을 사용하여 대상 디바이스용 모델을 패키징할 수 있습니다.

다음 값을 변경합니다.

- `project_name`은 패키징하려는 모델이 포함된 프로젝트 이름으로 바꿉니다.
- `model_version`은 패키징하려는 모델의 버전으로 바꿉니다.
- (선택 사항) `description`은 모델 패키징 작업에 대한 설명으로 바꿉니다.
- `component_name`를 생성하려는 모델 구성 요소의 이름으로 바꿉니다 AWS IoT Greengrass V2.
- (선택 사항) `component_version`은 패키징 작업에서 생성되는 모델 구성 요소의 버전으로 변경합니다. `major.minor.patch` 형식을 사용합니다. 예를 들어 1.0.0은 구성 요소의 첫 번째 주요 릴리스입니다.
- `bucket`은 패키징 작업이 모델 구성 요소 아티팩트를 저장하는 Amazon S3 버킷으로 변경합니다.
- `prefix`은 패키징 작업이 모델 구성 요소 아티팩트를 저장하는 Amazon S3 버킷 내 위치로 변경합니다.
- (선택 사항) `component_description`은 모델 구성 요소 설명으로 변경합니다.
- (선택 사항) `tag_key1` 및 `tag_key2`는 모델 컴포넌트에 첨부된 태그의 키로 변경합니다.
- (선택 사항) `tag_value1` 및 `tag_value2`는 모델 컴포넌트에 첨부된 태그의 키 값으로 변경합니다.

```
aws lookoutvision start-model-packaging-job \
  --project-name project_name \
  --model-version model_version \
  --description="description" \
  --configuration
 "Greengrass={TargetDevice='jetson_xavier',S3OutputLocation={Bucket='bucket',Prefix='pre
 {Key='tag_key2',Value='tag_value2'}]}" \
  --profile lookoutvision-access
```

예시:

```
aws lookoutvision start-model-packaging-job \
  --project-name project_01 \
  --model-version 1 \
  --description="description" \
  --configuration
 "Greengrass={TargetDevice='jetson_xavier',S3OutputLocation={Bucket='bucket',Prefix='com
 model component',Tags=[{Key='tag_key1',Value='tag_value1'},
 {Key='tag_key2',Value='tag_value2'}]}" \
  --profile lookoutvision-access
```

4. 응답의 JobName 값을 기록합니다. 이 정보는 다음 단계에서 필요합니다. 예시:

```
{
    "JobName": "6bcfd0ff-90c3-4463-9a89-6b4be3daf972"
}
```

5. 작업의 현재 상태를 가져오는 데 DescribeModelPackagingJob을 사용합니다. 다음을 변경합니다.

   - project_name은 사용 중인 프로젝트 이름으로 바꿉니다.
   - job_name은 이전 단계에서 기록해 둔 작업의 이름으로 바꿉니다.

```
aws lookoutvision describe-model-packaging-job \
    --project-name project_name \
    --job-name job_name \
    --profile lookoutvision-access
```

   Status값이 SUCCEEDED이면 모델 패키징 작업이 완료된 것입니다. 값이 다른 경우 잠시 기다린 후 다시 시도하십시오.

6. 를 사용하여 배포를 계속합니다 AWS IoT Greengrass V2. 자세한 내용은 디바이스에 구성 요소 배포 단원을 참조하십시오.

모델 패키징 작업에 대한 정보 가져오기

Amazon Lookout for Vision 콘솔 및 AWS SDK를 사용하여 생성한 모델 패키징 작업에 대한 정보를 가져올 수 있습니다.

주제

모델 패키징 작업 정보 가져오기 (콘솔)

모델 패키징 작업 정보를 가져오려면 (콘솔)

1. https://console.aws.amazon.com/lookoutvision/에서 Amazon Lookout for Vision 콘솔을 엽니다.
2. Get started를 선택합니다.
3. 왼쪽 탐색 창에서 프로젝트를 선택합니다.
4. 프로젝트 섹션에서 보려는 모델 패키징 작업이 포함된 프로젝트를 선택합니다.
5. 왼쪽 탐색 창의 프로젝트 이름에서 엣지 모델 패키지를 선택합니다.
6. 모델 패키징 작업 섹션에서 보려는 모델 패키징 작업을 선택합니다. 모델 패키징 작업의 세부 정보 페이지가 표시됩니다.

모델 패키징 작업 정보 (SDK) 가져오기

AWS SDK를 사용하여 프로젝트의 모델 패키징 작업을 나열하고 특정 모델 패키징 작업에 대한 정보를 가져올 수 있습니다.

모델 패키징 작업 목록

ListModelPackagingJobs API를 호출하여 프로젝트의 모델 패키징 작업을 나열할 수 있습니다. 응답에는 각 모델 패키징 작업에 대한 정보를 제공하는 ModelPackagingJobMetadata 객체 목록이 포함됩니다. 또한 목록이 불완전할 경우 다음 결과 세트를 가져오는 데 사용할 수 있는 페이지 매김 토큰도 포함되어 있습니다.

모델 패키징 작업을 나열하려면

1. 아직 설치하지 않은 경우 AWS CLI 및 AWS SDKs를 설치하고 구성합니다. 자세한 내용은 4단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.
2. 다음 CLI 명령을 사용하십시오. `project_name`을 사용하려는 프로젝트의 이름으로 변경합니다.

```
aws lookoutvision list-model-packaging-jobs \
  --project-name project_name \
```

```
    --profile lookoutvision-access
```

모델 패키징 작업 설명

DescribeModelPackagingJob API를 사용하여 모델 패키징 작업에 대한 정보를 가져옵니다. 응답은 작업의 현재 상태와 기타 정보가 포함된 ModelPackagingDescription 객체입니다.

패키지를 설명하려면

1. 아직 설치하지 않은 경우 AWS CLI 및 AWS SDKs를 설치하고 구성합니다. 자세한 내용은 4단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.
2. 다음 CLI 명령을 사용하십시오. 다음을 변경합니다.
   - `project_name`은 사용 중인 프로젝트 이름으로 바꿉니다.
   - `job_name`은 작업의 이름입니다. StartModelPackagingJob을 호출하면 작업 이름 (`JobName`)을 얻습니다.

```
aws lookoutvision describe-model-packaging-job \
    --project-name project_name \
    --job-name job_name \
    --profile lookoutvision-access
```

클라이언트 애플리케이션 구성 요소 작성

클라이언트 애플리케이션 구성 요소는 사용자가 작성하는 사용자 지정 AWS IoT Greengrass Version 2 구성 요소입니다. AWS IoT Greengrass Version 2 코어 디바이스에서 Amazon Lookout for Vision 모델을 사용하는 데 필요한 비즈니스 로직을 구현합니다.

모델에 액세스하기 위해 클라이언트 애플리케이션 구성 요소는 Lookout for Vision Edge Agent 구성 요소를 사용합니다. Lookout for Vision Edge Agent 구성 요소는 모델을 사용하여 이미지를 분석하고 코어 디바이스에서 모델을 관리하는 데 사용하는 API를 제공합니다.

Lookout for Vision Edge Agent API는 원격 프로시저 호출을 위한 프로토콜인 gRPC를 사용하여 구현됩니다. 자세한 내용은 gRPC을 참조하십시오. gRPC가 지원하는 모든 언어를 사용하여 코드를 작성할 수 있습니다. 예제 Python 코드를 제공합니다. 자세한 내용은 클라이언트 애플리케이션 구성 요소에서 모델 사용 단원을 참조하십시오.

> **Note**
>
> Lookout for Vision Edge Agent 구성 요소는 배포하는 모델 구성 요소의 종속 항목입니다. 모델 구성 요소를 코어 디바이스에 배포하면 코어 디바이스에 자동으로 배포됩니다.

클라이언트 애플리케이션 구성 요소를 작성하려면 다음 작업을 수행합니다.

1. gRPC를 사용하도록 환경을 설정하고 타사 라이브러리를 설치합니다.
2. 모델을 사용할 코드를 작성하세요.
3. 코드를 사용자 지정 구성 요소로 코어 디바이스에 배포합니다.

사용자 지정 GStreamer 파이프라인에서 이상 탐지를 수행하는 방법을 보여주는 클라이언트 애플리케이션 구성 요소의 예는 https://github.com/awslabs/aws-greengrass-labs-lookoutvision-gstreamer을 참조하십시오.

환경 설정

클라이언트 코드를 작성하기 위해 개발 환경은 Amazon Lookout for Vision 모델 구성 요소 및 종속 항목을 배포한 AWS IoT Greengrass Version 2 코어 디바이스에 원격으로 연결합니다. 또는 코어 디바이스에서 코드를 작성할 수 있습니다. 자세한 내용은 AWS IoT Greengrass 개발 도구 및 AWS IoT Greengrass 구성 요소 개발을 참조하십시오.

클라이언트 코드는 gRPC 클라이언트를 사용하여 Amazon Lookout for Vision 엣지 에이전트에 액세스해야 합니다. 이 섹션에서는 gRPC를 사용하여 개발 환경을 설정하고 `DetectAnomalies` 예제 코드에 필요한 타사 종속성을 설치하는 방법을 보여줍니다.

클라이언트 코드 작성을 완료한 후에는 사용자 지정 구성 요소를 만들고 사용자 지정 구성 요소를 에지 디바이스에 배포합니다. 자세한 내용은 클라이언트 애플리케이션 구성 요소 생성 단원을 참조하십시오.

주제

- gRPC 설정
- 타사 종속성 추가하기

gRPC 설정

개발 환경에서는 Lookout for Vision Edge Agent API를 호출하기 위해 코드에서 사용하는 gRPC 클라이언트가 필요합니다. 이렇게 하려면 Lookout for Vision Edge Agent의 `.proto`서비스 정의 파일을 사용하여 gRPC 스텁을 생성합니다.

> **Note**
>
> Lookout for Vision Edge Agent 애플리케이션 번들에서도 서비스 정의 파일을 가져올 수 있습니다. 애플리케이션 번들은 Lookout for Vision Edge Agent 구성 요소가 모델 구성 요소의 종속성으로 설치될 때 설치됩니다. 애플리케이션 번들은 `/greengrass/v2/packages/artifacts-unarchived/aws.iot.lookoutvision.EdgeAgent/`*`edge_agent_version`*`/lookoutvision_edge_agent`에 있습니다. 사용 중인 Lookout for Vision 엣지 에이전트 버전으로 `edge_agent_version`을 교체하십시오. 애플리케이션 번들을 받으려면 Lookout for Vision 엣지 에이전트를 코어 디바이스에 배포해야 합니다.

gRPC를 설정하려면 다음을 수행하세요.

1. zip 파일인 proto.zip를 다운로드하세요. zip 파일에는.proto 서비스 정의 파일 (`edge-agent.proto`)이 들어 있습니다.
2. 콘텐츠의 압축을 풉니다.
3. 명령 프롬프트를 열고 `edge-agent.proto`이 포함된 폴더로 이동합니다.
4. 다음 명령을 사용하여 Python 클라이언트 인터페이스를 생성합니다.

   ```
   %%bash
   python3 -m pip install grpcio
   python3 -m pip install grpcio-tools
   python3 -m grpc_tools.protoc --proto_path=. --python_out=. --grpc_python_out=.
    edge-agent.proto
   ```

   명령이 성공하면 작업 디렉토리에 스텁 `edge_agent_pb2_grpc.py` 및 `edge_agent_pb2.py`이 생성됩니다.
5. 모델을 사용하는 클라이언트 코드를 작성하세요. 자세한 내용은 클라이언트 애플리케이션 구성 요소에서 모델 사용 단원을 참조하십시오.

타사 종속성 추가하기

`DetectAnomalies` 예제 코드는 Pillow 라이브러리를 사용하여 이미지 작업을 수행합니다. 자세한 내용은 이미지 바이트를 사용하여 이상 징후를 탐지합니다. 단원을 참조하십시오.

다음 명령을 사용하여 Pillow 라이브러리를 설치합니다.

```
python3 -m pip install Pillow
```

클라이언트 애플리케이션 구성 요소에서 모델 사용

클라이언트 애플리케이션 구성 요소의 모델을 사용하는 단계는 클라우드에 호스팅된 모델을 사용하는 단계와 비슷합니다.

1. 모델 실행을 시작합니다.
2. 이미지에서 이상을 감지합니다.
3. 더 이상 필요하지 않은 경우 모델을 중지합니다.

Amazon Lookout for Vision Edge 에이전트는 모델을 시작하고, 이미지에서 이상을 감지하고, 모델을 중지할 수 있는 API를 제공합니다. API를 사용하여 디바이스의 모델을 나열하고 배포된 모델에 대한 정보를 얻을 수도 있습니다. 자세한 내용은 Amazon Lookout for Vision Edge Agent API 참조 단원을 참조하십시오.

gRPC 상태 코드를 확인하여 오류 정보를 얻을 수 있습니다. 자세한 내용은 오류 정보 가져오기 단원을 참조하십시오.

gRPC가 지원하는 모든 언어를 사용하여 코드를 작성할 수 있습니다. 예제 Python 코드를 제공합니다.

주제

- 클라이언트 애플리케이션 구성 요소의 스텁 사용
- 모델 시작
- 이상 감지
- 모델 중지
- 디바이스의 모델 목록
- 모델 설명
- 오류 정보 가져오기

클라이언트 애플리케이션 구성 요소의 스텁 사용

다음 코드를 사용하여 Lookout for Vision 엣지 에이전트를 통해 모델에 대한 액세스를 설정합니다.

```
import grpc
from edge_agent_pb2_grpc import EdgeAgentStub
import edge_agent_pb2 as pb2

# Creating stub.
 with grpc.insecure_channel("unix:///tmp/aws.iot.lookoutvision.EdgeAgent.sock") as
 channel:
    stub = EdgeAgentStub(channel)
    # Add additional code that works with Edge Agent in this block to prevent resources
 leakage
```

모델 시작

StartModel API를 호출하여 모델을 시작합니다. 모델을 시작하는 데 시간이 걸릴 수 있습니다. DescribeModel을 호출하면 현재 상태를 확인할 수 있습니다. `status` 필드 값이 Running이면 모델이 실행 중인 것입니다.

예제 코드

component_name# 모델 구성 요소의 이름으로 바꾸십시오.

```
import time

import grpc
from edge_agent_pb2_grpc import EdgeAgentStub
import edge_agent_pb2 as pb2


model_component_name = "component_name"


def start_model_if_needed(stub, model_name):
    # Starting model if needed.
    while True:
        model_description_response =
 stub.DescribeModel(pb2.DescribeModelRequest(model_component=model_name))
        print(f"DescribeModel() returned {model_description_response}")
        if model_description_response.model_description.status == pb2.RUNNING:
            print("Model is already running.")
```

```
            break
        elif model_description_response.model_description.status == pb2.STOPPED:
            print("Starting the model.")
            stub.StartModel(pb2.StartModelRequest(model_component=model_name))
            continue
        elif model_description_response.model_description.status == pb2.FAILED:
            raise Exception(f"model {model_name} failed to start")
        print(f"Waiting for model to start.")
        if model_description_response.model_description.status != pb2.STARTING:
            break
        time.sleep(1.0)


# Creating stub.
with grpc.insecure_channel("unix:///tmp/aws.iot.lookoutvision.EdgeAgent.sock") as
 channel:
    stub = EdgeAgentStub(channel)
    start_model_if_needed(stub, model_component_name)
```

이상 감지

DetectAnomaliesAPI를 사용하여 이미지의 이상을 감지합니다.

이 DetectAnomalies 작업에서는 이미지 비트맵이 RGB888 압축 형식으로 전달될 것으로 예상합니다. 첫 번째 바이트는 빨간색 채널을, 두 번째 바이트는 녹색 채널을, 세 번째 바이트는 파란색 채널을 나타냅니다. 이미지를 다른 형식 (예: BGR)으로 제공하는 경우 DetectAnomalies의 예측은 올바르지 않습니다.

기본적으로 OpenCV는 이미지 비트맵에 BGR 형식을 사용합니다. OpenCV를 사용하여 DetectAnomalies로 분석할 이미지를 캡처하는 경우 이미지를 DetectAnomalies에 전달하기 전에 이미지를 RGB888 형식으로 변환해야 합니다.

DetectAnomalies에 제공하는 이미지의 너비와 높이 치수는 모델을 학습시키는 데 사용한 이미지와 같아야 합니다.

이미지 바이트를 사용하여 이상 징후를 탐지합니다.

이미지를 이미지 바이트로 제공하여 이미지의 이상을 감지할 수 있습니다. 다음 예제에서는 로컬 파일 시스템에 저장된 이미지에서 이미지 바이트를 검색합니다.

*sample.jpg*를 분석하려는 이미지 파일의 이름으로 바꿉니다. *component_name*을 모델 구성 요소의 이름으로 바꾸십시오.

```
import time

from PIL import Image
import grpc
from edge_agent_pb2_grpc import EdgeAgentStub
import edge_agent_pb2 as pb2


model_component_name = "component_name"

....
# Detecting anomalies.
def detect_anomalies(stub, model_name, image_path):
    image = Image.open(image_path)
    image = image.convert("RGB")
    detect_anomalies_response = stub.DetectAnomalies(
        pb2.DetectAnomaliesRequest(
            model_component=model_name,
            bitmap=pb2.Bitmap(
                width=image.size[0],
                height=image.size[1],
                byte_data=bytes(image.tobytes())
            )
        )
    )
    print(f"Image is anomalous -
 {detect_anomalies_response.detect_anomaly_result.is_anomalous}")
    return detect_anomalies_response.detect_anomaly_result


# Creating stub.
with grpc.insecure_channel("unix:///tmp/aws.iot.lookoutvision.EdgeAgent.sock") as
 channel:
    stub = EdgeAgentStub(channel)
    start_model_if_needed(stub, model_component_name)
    detect_anomalies(stub, model_component_name, "sample.jpg")
```

공유 메모리 세그먼트를 사용하여 이상 징후를 탐지합니다.

이미지를 POSIX 공유 메모리 세그먼트에 이미지 바이트로 제공하여 이미지의 이상을 감지할 수 있습니다. 최상의 성능을 내려면 DetectAnomalies 요청에 공유 메모리를 사용하는 것이 좋습니다. 자세한 내용은 DetectAnomalies 단원을 참조하십시오.

모델 중지

모델을 더 이상 사용하지 않는 경우 모델 실행을 중지하는 StopModel API입니다.

```
stop_model_response = stub.StopModel(
    pb2.StopModelRequest(
         model_component=model_component_name
    )
)
print(f"New status of the model is {stop_model_response.status}")
```

디바이스의 모델 목록

the section called “ListModels” API를 사용하여 디바이스에 배포된 모델을 나열할 수 있습니다.

```
models_list_response = stub.ListModels(
    pb2.ListModelsRequest()
)
for model in models_list_response.models:
    print(f"Model Details {model}")
```

모델 설명

DescribeModel API를 호출하여 기기에 배포된 모델에 대한 정보를 얻을 수 있습니다. `DescribeModel`를 사용하면 모델의 현재 상태를 가져오는 데 유용합니다. 예를 들어, `DetectAnomalies`을 호출하기 전에 먼저 모델이 실행 중인지 알아야 합니다. 예제 코드는 모델 시작 항목을 참조하세요.

오류 정보 가져오기

gRPC 상태 코드는 API 결과를 보고하는 데 사용됩니다.

다음 예제와 같이 `RpcError` 예외를 포착하여 오류 정보를 가져올 수 있습니다. 오류 상태 코드에 대한 자세한 내용은 API의 참조 항목을 참조하십시오.

```
# Error handling.
try:
    stub.DetectAnomalies(detect_anomalies_request)
except grpc.RpcError as e:
```

```
    print(f"Error code: {e.code()}, Status: {e.details()}")
```

클라이언트 애플리케이션 구성 요소 생성

gRPC 스텁을 생성하고 클라이언트 애플리케이션 코드를 준비했으면 클라이언트 애플리케이션 구성 요소를 생성할 수 있습니다. 생성하는 구성 요소는 AWS IoT Greengrass Version 2 코어 디바이스에 배포하는 사용자 지정 구성 요소입니다 AWS IoT Greengrass V2. 생성한 레시피는 사용자 지정 구성 요소를 설명합니다. 레시피에는 배포해야 하는 모든 종속성이 포함되어 있습니다. 이 경우 Amazon Lookout for Vision 모델 패키징에서 생성할 모델 구성요소를 지정합니다. 이러한 구성 요소 레시피에 대한 자세한 내용은 AWS IoT Greengrass Version 2 구성 요소 레시피 참조를 참조하십시오.

이 주제의 절차에서는 레시피 파일에서 클라이언트 애플리케이션 구성 요소를 생성하고 사용자 AWS IoT Greengrass V2 지정 구성 요소로 게시하는 방법을 보여줍니다. AWS IoT Greengrass V2 콘솔 또는 AWS SDK를 사용하여 구성 요소를 게시할 수 있습니다.

사용자 지정 구성 요소를 만드는 방법에 대한 자세한 내용은 AWS IoT Greengrass V2설명서의 다음을 참조하십시오.

- 디바이스에서 구성 요소를 개발하고 테스트하십시오.
- Create AWS IoT Greengrass 구성 요소
- 코어 디바이스에 배포할 구성 요소를 게시하세요.

주제

- 클라이언트 애플리케이션 구성 요소를 게시하기 위한 IAM 권한
- 레시피 생성
- 클라이언트 애플리케이션 구성 요소 게시 (콘솔)
- 클라이언트 애플리케이션 구성 요소 (SDK) 게시

클라이언트 애플리케이션 구성 요소를 게시하기 위한 IAM 권한

클라이언트 애플리케이션 구성 요소를 만들고 게시하려면 다음과 같은 IAM 권한이 필요합니다.

- `greengrass:CreateComponentVersion`
- `greengrass:DescribeComponent`
- `s3:PutObject`

레시피 생성

이 절차에서는 간단한 클라이언트 애플리케이션 구성 요소의 레시피를 생성합니다. `lookoutvision_edge_agent_example.py`의 코드는 디바이스에 배포된 모델을 나열하며, 구성 요소를 코어 디바이스에 배포한 후 자동으로 실행됩니다. 출력을 보려면 구성 요소를 배포한 후 구성 요소 로그를 확인하십시오. 자세한 내용은 디바이스에 구성 요소 배포 단원을 참조하십시오. 준비가 되면 이 절차를 사용하여 비즈니스 로직을 구현하는 코드의 레시피를 만드십시오.

레시피는 JSON 또는 YAML 형식 파일로 생성합니다. 또한 Amazon S3 버킷에 클라이언트 애플리케이션 코드를 업로드합니다.

클라이언트 애플리케이션 구성 요소 레시피를 만들려면

1. 아직 하지 않았다면 gRPC 스텁 파일을 만듭니다. 자세한 내용은 gRPC 설정 단원을 참조하십시오.
2. 다음 코드를 `lookoutvision_edge_agent_example.py`이라는 파일에 저장합니다.

```
import grpc
from edge_agent_pb2_grpc import EdgeAgentStub
import edge_agent_pb2 as pb2

# Creating stub.
with grpc.insecure_channel("unix:///tmp/aws.iot.lookoutvision.EdgeAgent.sock") as
 channel:
    stub = EdgeAgentStub(channel)
    # Add additional code that works with Edge Agent in this block to prevent
 resources leakage

    models_list_response = stub.ListModels(
        pb2.ListModelsRequest()
    )
    for model in models_list_response.models:
        print(f"Model Details {model}")
```

3. Amazon S3 버킷을 생성 (또는 기존 버킷 사용)하여 클라이언트 애플리케이션 구성 요소의 소스 파일을 저장합니다. 버킷은 AWS 계정과 AWS IoT Greengrass Version 2 및 Amazon Lookout for Vision을 사용하는 AWS 리전에 있어야 합니다.
4. 이전 단계에서 생성한 Amazon S3 버킷에 `lookoutvision_edge_agent_example.py`, `edge_agent_pb2_grpc.py and edge_agent_pb2.py`을 업로드합니다. 각 파일

의 Amazon S3 경로를 기록합니다. gRPC 설정에서 edge_agent_pb2_grpc.py 및 edge_agent_pb2.py를 생성합니다.

5. 편집기에서 다음과 같은 JSON 또는 YAML 레시피 파일을 생성합니다.

   - model_component은 모델 구성 요소의 이름으로 생성합니다. 자세한 내용은 구성 요소 설정 단원을 참조하십시오.
   - URI 항목을 lookoutvision_edge_agent_example.pyedge_agent_pb2_grpc.py, 및 edge_agent_pb2.py의 S3 경로로 변경합니다.

   JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.lookoutvision.EdgeAgentPythonExample",
  "ComponentVersion": "1.0.0",
  "ComponentType": "aws.greengrass.generic",
  "ComponentDescription": "Lookout for Vision Edge Agent Sample Application",
  "ComponentPublisher": "Sample App Publisher",
  "ComponentDependencies": {
    "model_component": {
      "VersionRequirement": ">=1.0.0",
      "DependencyType": "HARD"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "pip3 install grpcio grpcio-tools protobuf Pillow",
        "run": {
          "script": "python3 {artifacts:path}/
lookoutvision_edge_agent_example.py"
        }
      },
      "Artifacts": [
        {
          "Uri": "S3 path to lookoutvision_edge_agent_example.py"
        },
        {
```

```
          "Uri": "S3 path to edge_agent_pb2_grpc.py"
        },
        {
          "Uri": "S3 path to edge_agent_pb2.py"
        }
      ]
    }
  ],
  "Lifecycle": {}
}
```

YAML

```
---
RecipeFormatVersion: 2020-01-25
ComponentName: com.lookoutvison.EdgeAgentPythonExample
ComponentVersion: 1.0.0
ComponentDescription: Lookout for Vision Edge Agent Sample Application
ComponentPublisher: Sample App Publisher
ComponentDependencies:
  model_component:
    VersionRequirement: '>=1.0.0'
    DependencyType: HARD
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: |-
        pip3 install grpcio
        pip3 install grpcio-tools
        pip3 install protobuf
        pip3 install Pillow
      run:
        script: |-
          python3 {artifacts:path}/lookout_vision_agent_example.py
    Artifacts:
      - URI: S3 path to lookoutvision_edge_agent_example.py
      - URI: S3 path to edge_agent_pb2_grpc.py
      - URI: S3 path to edge_agent_pb2.py
```

6. JSON 또는 YAML 파일을 컴퓨터에 저장합니다.
7. 다음 중 하나를 수행하여 클라이언트 애플리케이션 구성 요소를 생성합니다.

- AWS IoT Greengrass 콘솔을 사용하려면를 수행합니다클라이언트 애플리케이션 구성 요소 게시 (콘솔).
- AWS SDK를 사용하려면를 수행합니다클라이언트 애플리케이션 구성 요소 (SDK) 게시.

클라이언트 애플리케이션 구성 요소 게시 (콘솔)

AWS IoT Greengrass V2 콘솔을 사용하여 클라이언트 애플리케이션 구성 요소를 게시할 수 있습니다.

클라이언트 애플리케이션 구성 요소를 게시하려면

1. 아직 만들지 않았다면 레시피 생성을 수행하여 클라이언트 애플리케이션 구성 요소 레시피를 만듭니다.
2. https://console.aws.amazon.com/iot/ AWS IoT Greengrass 콘솔을 엽니다.
3. 왼쪽 탐색 창의 Greengrass 에서 구성 요소를 선택합니다.
4. 내 구성 요소에서 구성 요소 생성을 선택합니다.
5. JSON 형식 레시피를 사용하려면 구성 요소 생성 페이지에서 레시피를 JSON으로 입력을 선택합니다. YAML 형식 레시피를 사용하려면 레시피를 YAML로 입력을 선택합니다.
6. 레시피에서 기존 레시피를 레시피 생성에서 만든 JSON 또는 YAML 레시피로 대체합니다.
7. 구성 요소 생성을 선택합니다.
8. 다음으로 클라이언트 애플리케이션 구성 요소를 배포합니다.

클라이언트 애플리케이션 구성 요소 (SDK) 게시

CreateComponentVersion API를 사용하여 클라이언트 애플리케이션 구성 요소를 게시할 수 있습니다.

클라이언트 애플리케이션 구성 요소 (SDK) 를 게시하려면

1. 아직 만들지 않았다면 레시피 생성을 수행하여 클라이언트 애플리케이션 구성 요소 레시피를 만듭니다.
2. 명령 프롬프트에서 다음 명령을 입력하여 클라이언트 애플리케이션 구성 요소를 생성합니다. 레시피 생성에서 생성한 레시피 파일의 이름으로 `recipe-file`을 바꿉니다.

   ```
   aws greengrassv2 create-component-version --inline-recipe fileb://recipe-file
   ```

   응답의 구성 요소의 ARN을 기록합니다. 이 정보는 다음 단계에서 필요합니다.

3. 다음 명령을 사용하여 클라이언트 애플리케이션 구성 요소 상태를 가져옵니다. `component-arn`을 이전 단계에서 기록한 정책 ARN으로 바꾸세요. `componentState`의 값이 `DEPLOYABLE`이면 클라이언트 응용 프로그램 구성 요소가 준비된 것입니다.

```
aws greengrassv2 describe-component --arn component-arn
```

4. 다음으로 클라이언트 애플리케이션 구성 요소를 배포합니다.

디바이스에 구성 요소 배포

모델 구성 요소와 클라이언트 애플리케이션 구성 요소를 AWS IoT Greengrass Version 2 코어 디바이스에 배포하려면 AWS IoT Greengrass V2 콘솔을 사용하거나 CreateDeployment API를 사용합니다. 자세한 내용은AWS IoT Greengrass Version 2 개발자 안내서의 배포 생성을 참조하세요. 코어 디바이스에 배포된 구성 요소를 업데이트하는 방법에 대한 자세한 내용은 배포 수정을 참조하십시오.

주제

- 구성 요소 배포를 위한 IAM 권한
- 구성 요소 배포 (콘솔)
- 구성 요소 (SDK) 배포

구성 요소 배포를 위한 IAM 권한

를 사용하여 구성 요소를 배포하려면 다음 권한이 AWS IoT Greengrass V2 필요합니다.

- `greengrass:ListComponents`
- `greengrass:ListComponentVersions`
- `greengrass:ListCoreDevices`
- `greengrass:CreateDeployment`
- `greengrass:GetDeployment`
- `greengrass:ListDeployments`

`CreateDeployment` 및 `GetDeployment`는 종속 작업이 있어야 합니다. 자세한 내용은 AWS IoT Greengrass V2에서 정의한 작업을 참조하십시오.

IAM 권한 변경에 대한 자세한 내용은 사용자 권한 변경을 참조하십시오.

구성 요소 배포 (콘솔)

다음 절차를 사용하여 클라이언트 애플리케이션 구성 요소를 코어 디바이스에 배포하십시오. 클라이언트 애플리케이션은 모델 구성 요소 (Lookout for Vision Edge Agent에 따라 달라짐)에 따라 달라집니다. 클라이언트 애플리케이션 구성 요소를 배포하면 모델 구성 요소 및 Lookout for Vision Edge Agent의 배포도 시작됩니다.

> **Note**
>
> 기존 배포에 구성 요소를 추가할 수 있습니다. 사물 그룹에 구성 요소를 배포할 수도 있습니다.

이 절차를 실행하려면 AWS IoT Greengrass V2 코어 디바이스가 구성되어 있어야 합니다. 자세한 내용은 AWS IoT Greengrass Version 2 코어 디바이스 설정 단원을 참조하십시오.

구성 요소를 디바이스에 배포하려면

1. https://console.aws.amazon.com/iot/ AWS IoT Greengrass 콘솔을 엽니다.
2. 왼쪽 탐색 창의 Greengrass에서 배포를 선택합니다.
3. 배포에서 생성을 선택합니다.
4. 대상 지정 페이지에서 다음 작업을 수행합니다.
    1. 배포 정보 아래에서 친숙한 배포 이름을 입력하거나 수정합니다.
    2. 배포 대상에서 코어 디바이스를 선택하고 대상 이름을 입력합니다.
    3. 다음을 선택합니다.
5. 구성 요소 선택 페이지에서 다음을 수행합니다.
    1. 내 구성 요소에서 클라이언트 응용 프로그램 구성 요소의 이름을 선택합니다 (`com.lookoutvison.EdgeAgentPythonExample`).
    2. 다음을 선택합니다.
6. 구성 요소 구성 페이지에서 현재 구성을 유지하고 다음을 선택합니다.
7. 고급 설정 구성 페이지에서 현재 설정을 유지하고 다음을 선택합니다.
8. 검토 페이지에서 배포를 선택하여 구성 요소 배포를 시작합니다.

배포 상태 확인 (콘솔)

AWS IoT Greengrass V2 콘솔에서 배포 상태를 확인할 수 있습니다. 클라이언트 응용 프로그램 구성 요소가 the section called “클라이언트 애플리케이션 구성 요소 생성”의 예제 레시피와 코드를 사용하는 경우 배포가 완료된 후 클라이언트 응용 프로그램 구성 요소 로그를 확인하십시오. 성공하면 구성 요소에 배포된 Lookout for Vision 모델 목록이 로그에 포함됩니다.

AWS SDK를 사용하여 배포 상태를 확인하는 방법에 대한 자세한 내용은 배포 상태 확인을 참조하세요.

배포 상태를 확인하려면

1. https://console.aws.amazon.com/iot/ AWS IoT Greengrass 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 코어 디바이스 를 선택합니다.
3. Greengrass 코어 디바이스에서 코어 디바이스를 선택합니다.
4. 현재 배포 상태를 보려면 배포 탭을 선택합니다.
5. 배포가 성공하면 (상태가 완료됨) 코어 디바이스에서 터미널 창을 열고 `/greengrass/v2/logs/com.lookoutvison.EdgeAgentPythonExample.log`에서 클라이언트 애플리케이션 구성 요소 로그를 확인하십시오. 배포에서 예제 레시피와 코드를 사용하는 경우 로그에는 `lookoutvision_edge_agent_example.py`의 출력이 포함됩니다. 예시:

   ```
   Model Details model_component:"ModelComponent"
   ```

구성 요소 (SDK) 배포

다음 절차를 사용하여 클라이언트 애플리케이션 구성 요소, 모델 구성 요소 및 Amazon Lookout for Vision Edge Agent를 코어 디바이스에 배포하십시오.

1. `deployment.json` 파일을 생성하여 구성 요소의 배포 구성을 정의합니다. 이 파일은 다음 예제와 비슷합니다.

   ```
   {
     "targetArn":"targetArn",
     "components": {
       "com.lookoutvison.EdgeAgentPythonExample": {
         "componentVersion": "1.0.0",
         "configurationUpdate": {
         }
   ```

```
    }
  }
}
```

- targetArn 필드에서 *targetArn*을(를) 다음 형식으로 배포 대상으로 지정할 사물 또는 사물 그룹의 Amazon 리소스 이름(ARN)으로 바꿉니다.
  - 사물: arn:aws:iot:*region*:*account-id*:thing/*thingName*
  - 사물 그룹: arn:aws:iot:*region*:*account-id:*thinggroup/*thingGroupName*

2. 배포 대상에 수정하려는 기존 배포가 있는지 확인하십시오. 다음을 수행합니다.

   a. 다음 명령을 실행하여 배포 대상의 배포를 나열합니다. 를 target AWS IoT 사물 또는 사물 그룹의 Amazon 리소스 이름(ARN)targetArn으로 바꿉니다. 현재 AWS 리전에 있는 사물의 ARN을 가져오려면 명령 aws iot list-things을 사용하십시오.

   ```
   aws greengrassv2 list-deployments --target-arn targetArn
   ```

   응답에는 대상에 대한 최신 배포 목록이 포함되어 있습니다. 응답이 비어 있는 경우 대상에 기존 배포가 없는 것이므로, 3단계로 건너뛸 수 있습니다. 그렇지 않으면 다음 단계에서 사용할 수 있도록 응답에서 deploymentId을 복사합니다.

   b. 다음 명령을 실행하여 배포의 세부 정보를 가져옵니다. 이러한 세부 정보에는 메타데이터, 구성 요소 및 작업 구성이 포함됩니다. deploymentId을 전 단계의 ID로 바꿉니다.

   ```
   aws greengrassv2 get-deployment --deployment-id deploymentId
   ```

   c. 이전 명령의 응답에서 다음 키-값 쌍을 deployment.json으로 복사합니다. 새 배포에서 이러한 값을 변경할 수 있습니다.

   - deploymentName— 배포의 이름.
   - components—배포의 구성 요소. 구성 요소를 제거하려면 이 개체에서 구성 요소를 제거하십시오.
   - deploymentPolicies— 배포 정책.
   - tags— 배포의 태그.

3. 다음 명령을 실행하여 디바이스에 구성 요소를 배포합니다. 응답의 deploymentId 값을 기록합니다.

```
aws greengrassv2 create-deployment \
```

```
    --cli-input-json file://path/to/deployment.json
```

4. 다음 명령을 실행하여 배포 상태를 확인합니다. `deployment-id`을 이전 단계에서 기록한 값으로 변경합니다. `deploymentStatus`의 값이 `COMPLETED`인 경우 배포가 성공적으로 완료된 것입니다.

```
aws greengrassv2 get-deployment --deployment-id deployment-id
```

5. 배포가 성공하면 코어 디바이스에서 터미널 창을 열고 `/greengrass/v2/logs/com.lookoutvison.EdgeAgentPythonExample.log`에서 클라이언트 애플리케이션 구성 요소 로그를 확인하십시오. 배포에서 예제 레시피와 코드를 사용하는 경우 로그에는 `lookoutvision_edge_agent_example.py`의 출력이 포함됩니다. 예시:

```
Model Details model_component:"ModelComponent"
```

Amazon Lookout for Vision Edge Agent API 참조

이 섹션은 Amazon Lookout for Vision Edge Agent에 대한 API 참조입니다.

모델을 사용하여 이상 감지

DetectAnomalies API를 사용하여 AWS IoT Greengrass Version 2 코어 디바이스에서 실행 중인 모델을 사용하여 이미지의 이상을 감지합니다.

모델 정보 가져오기

코어 디바이스에 배포된 모델에 대한 정보를 가져오는 API.

- ListModels
- DescribeModel

모델 실행

코어 디바이스에 배포되는 Amazon Lookout for Vision 모델을 시작하고 중지하기 위한 API입니다.

- StartModel
- StopModel

DetectAnomalies

제공된 이미지에서 이상을 감지합니다.

DetectAnomalies의 응답 양식에는 이미지에 하나 이상의 예외가 포함되어 있다는 부울 예측과 예측에 대한 신뢰값이 포함됩니다. 모델이 분할 모델인 경우 응답에는 다음이 포함됩니다.

- 각 예외 유형을 고유한 색상으로 포함하는 마스크 이미지입니다. DetectAnomalies이 마스크 이미지를 공유 메모리에 저장하거나 마스크를 이미지 바이트로 반환하도록 할 수 있습니다.
- 예외 유형이 포함하는 이미지의 백분율 영역입니다.
- 마스크 이미지에 있는 예외 유형의 16진수 색상입니다.

> **Note**
>
> DetectAnomalies와 함께 사용하는 모델이 실행 중이어야 합니다. DescribeModel를 호출하여 현재 상태를 확인할 수 있습니다. 모델을 시작하려면 StartModel 단원을 참조하십시오.

DetectAnomalies은 인터리브 RGB888 형식의 패킹된 비트맵 (이미지)을 지원합니다. 첫 번째 바이트는 빨간색 채널을, 두 번째 바이트는 녹색 채널을, 세 번째 바이트는 파란색 채널을 나타냅니다. 이미지를 다른 형식 (예: BGR)으로 제공하는 경우 DetectAnomalies의 예측은 올바르지 않습니다.

기본적으로 OpenCV는 이미지 비트맵에 BGR 형식을 사용합니다. OpenCV를 사용하여 DetectAnomalies로 분석할 이미지를 캡처하는 경우 이미지를 DetectAnomalies에 전달하기 전에 이미지를 RGB888 형식으로 변환해야 합니다.

지원되는 최소 이미지 크기는 64x64픽셀입니다. 지원되는 최대 이미지 크기는 4096x4096픽셀입니다.

Protobuf 메시지 또는 공유 메모리 세그먼트를 통해 이미지를 전송할 수 있습니다. 큰 이미지를 protobuf 메시지로 직렬화하면 DetectAnomalies에 대한 호출 지연 시간이 크게 늘어날 수 있습니다. 지연 시간을 최소화하려면 공유 메모리를 사용하는 것이 좋습니다.

```
rpc DetectAnomalies(DetectAnomaliesRequest) returns (DetectAnomaliesResponse);
```

DetectAnomaliesRequest

DetectAnomalies의 입력 파라미터.

```
message Bitmap {
  int32 width = 1;
  int32 height = 2;
  oneof data {
    bytes byte_data = 3;
    SharedMemoryHandle shared_memory_handle = 4;
  }
}
```

```
message SharedMemoryHandle {
  string name = 1;
  uint64 size = 2;
  uint64 offset = 3;
}
```

```
message AnomalyMaskParams {
SharedMemoryHandle shared_memory_handle = 2;
}
```

```
message DetectAnomaliesRequest {
string model_component = 1;
Bitmap bitmap = 2;
AnomalyMaskParams anomaly_mask_params = 3;
}
```

Bitmap

DetectAnomalies와 함께 분석에 사용할 이미지.

width

이미지의 너비(픽셀)

height

이미지의 높이(픽셀)

byte_data

protobuf 메시지에 전달된 이미지 바이트.

shared_memory_handle

공유 메모리 세그먼트에 전달된 이미지 바이트

SharedMemoryHandle

POSIX 공유 메모리 세그먼트를 나타냅니다.

name

POSIX 메모리 세그먼트의 이름입니다. 공유 메모리 생성에 대한 자세한 내용은 shm_open을 참조하십시오.

size

오프셋에서 시작하는 이미지 버퍼 크기 (바이트).

offset

공유 메모리 세그먼트의 시작 부분부터 이미지 버퍼의 시작 부분까지의 오프셋 (바이트)

AnomalyMaskParams

예외 마스크를 출력하기 위한 파라미터. (세그멘테이션 모델).

shared_memory_handle

마스크의 이미지 바이트를 포함합니다 (`shared_memory_handle`이 제공되지 않은 경우).

DetectAnomaliesRequest

model_component

사용하려는 모델이 포함된 AWS IoT Greengrass V2 구성 요소의 이름입니다.

bitmap

`DetectAnomalies`와 함께 분석에 사용할 이미지.

anomaly_mask_params

마스크 출력을 위한 선택적 파라미터. (세그멘테이션 모델).

DetectAnomaliesResponse

DetectAnomalies의 응답.

```
message DetectAnomalyResult {
bool is_anomalous = 1;
float confidence = 2;
Bitmap anomaly_mask = 3;
repeated Anomaly anomalies = 4;
float anomaly_score = 5;
float anomaly_threshold = 6;
}
```

```
message Anomaly {
string name = 1;
PixelAnomaly pixel_anomaly = 2;
```

```
message PixelAnomaly {
float total_percentage_area = 1;
string hex_color = 2;
}
```

```
message DetectAnomaliesResponse {
  DetectAnomalyResult detect_anomaly_result = 1;
}
```

이상 항목

이미지에서 발견된 이상을 나타냅니다. (세그멘테이션 모델).

name

이미지에서 발견된 예외 유형의 이름. name은 학습 데이터 세트의 이상 유형에 매핑됩니다. 이 서비스는 DetectAnomalies의 응답에 배경 이상 유형을 자동으로 삽입합니다.

pixel_anomaly

예외 유형을 포함하는 픽셀 마스크에 대한 정보입니다.

PixelAnomaly

예외 유형을 포함하는 픽셀 마스크에 대한 정보입니다. (세그멘테이션 모델).

total_percentage_area

예외 유형이 포함하는 이미지의 백분율 영역입니다.

hex_color

이미지의 예외 유형을 나타내는 16진수 색상 값입니다. 색상은 학습 데이터 세트에 사용된 예외 유형의 색상에 매핑됩니다.

DetectAnomalyResult

is_anomalous

이미지에 예외가 포함되어 있는지 여부를 나타냅니다. `true`은 이미지에 예외가 포함된 경우. `false`는 이미지가 정상인 경우

confidence

`DetectAnomalies`의 예측의 정확성에 대한 신뢰도. `confidence`는 0과 1 사이의 부동 소수점 값입니다.

anomaly_mask

shared_memory_handle이 제공되지 않은 경우 마스크의 이미지 바이트를 포함합니다. (세그멘테이션 모델).

예외

입력 영상에서 발견된 0개 이상의 예외 항목 목록. (세그멘테이션 모델).

anomaly_score

이미지에 대해 예측된 예외 항목이 예외 항목이 없는 이미지에서 얼마나 벗어나는지를 정량화하는 숫자입니다. `anomaly_score`는 ~ (일반 이미지와의 최소 편차) `0.0`에서 1.0 (일반 이미지와의 최대 편차) 범위의 부동 소수점 값입니다. Amazon Lookout for Vision은 이미지에 대한 예측이 정상인 경우에도 `anomaly_score`에 대한 값을 반환합니다.

anomaly_threshold

영상의 예측된 분류가 정상인지 비정상인지를 결정하는 숫자 (부동 소수점). anomaly_score이 anomaly_threshold의 값보다 크거나 같은 이미지는 비정상 이미지로 간주됩니다. anomaly_threshold 아래의 anomaly_score 값은 정상 이미지를 나타냅니다. 모델이 사용하는 anomaly_threshold값은 모델을 학습시킬 때 Amazon Lookout for Vision에서 계산합니다. anomaly_threshold 값은 설정하거나 변경할 수 없습니다.

상태 코드

코드	숫자	설명
정상	0	DetectAnomalies 은 예측을 성공적으로 수행했습니다.
UNKNOWN	2	알 수 없는 오류가 발생했습니다.
INVALID_ARGUMENT	3	하나 이상의 입력 파라미터가 유효하지 않습니다. 자세한 내용은 오류 메시지를 확인하세요.
NOT_FOUND	5	지정된 이름의 모델을 찾을 수 없습니다.
RESOURCE_EXHAUSTED	8	리소스가 충분하지 않아 이 작업을 수행할 수 없습니다. 예를 들어 Lookout for Vision Edge Agent는 DetectAnomalies 을 호출하는 속도를 따라갈 수 없습니다. 자세한 내용은 오류 메시지를 확인하세요.
FAILED_PRECONDITION	9	DetectAnomalies 은 RUNNING 상태가 아닌 모델에 대해 호출되었습니다.
INTERNAL	13	내부 서비스 오류가 발생했습니다.

DescribeModel

AWS IoT Greengrass Version 2 코어 디바이스에 배포된 Amazon Lookout for Vision 모델을 설명합니다.

```
rpc DescribeModel(DescribeModelRequest) returns (DescribeModelResponse);
```

DescribeModelRequest

```
message DescribeModelRequest {
  string model_component = 1;
}
```

model_component

설명하려는 모델이 포함된 AWS IoT Greengrass V2 구성 요소의 이름입니다.

DescribeModelResponse

```
message ModelDescription {
  string model_component = 1;
  string lookout_vision_model_arn = 2;
  ModelStatus status = 3;
  string status_message = 4;
}
```

```
message DescribeModelResponse {
  ModelDescription model_description = 1;
}
```

ModelDescription

model_component

Amazon Lookout for Vision 모델을 포함하는 AWS IoT Greengrass Version 2 구성 요소의 이름입니다.

lookout_vision_model_arn

AWS IoT Greengrass V2 구성 요소를 생성하는 데 사용된 Amazon Lookout for Vision 모델의 Amazon 리소스 이름 ARN입니다.

status

모델의 현재 상태입니다. 자세한 내용은 ModelStatus 단원을 참조하십시오.

status_message

모델의 상태 메시지입니다.

상태 코드

코드	숫자	설명
정상	0	호출이 성공했습니다.
UNKNOWN	2	알 수 없는 오류가 발생했습니다.
INVALID_ARGUMENT	3	하나 이상의 입력 파라미터가 유효하지 않습니다. 자세한 내용은 오류 메시지를 확인하세요.
NOT_FOUND	5	제공된 이름을 가진 모델을 찾을 수 없습니다.
INTERNAL	13	내부 서비스 오류가 발생했습니다.

ListModels

AWS IoT Greengrass Version 2 코어 디바이스에 배포된 모델을 나열합니다.

```
rpc ListModels(ListModelsRequest) returns (ListModelsResponse);
```

ListModelsRequest

```
message ListModelsRequest {}
```

ListModelsResponse

```
message ModelMetadata {
  string model_component = 1;
  string lookout_vision_model_arn = 2;
  ModelStatus status = 3;
  string status_message = 4;
}
```

```
message ListModelsResponse {
  repeated ModelMetadata models = 1;
}
```

ModelMetadata

model_component

Amazon Lookout for Vision 모델을 포함하는 AWS IoT Greengrass Version 2 구성 요소의 이름입니다.

lookout_vision_model_arn

AWS IoT Greengrass V2 구성 요소를 생성하는 데 사용된 Amazon Lookout for Vision 모델의 Amazon 리소스 이름(ARN)입니다.

status

모델의 현재 상태입니다. 자세한 내용은 ModelStatus 단원을 참조하십시오.

status_message

모델의 상태 메시지입니다.

상태 코드

코드	숫자	설명
정상	0	호출이 성공했습니다.

코드	숫자	설명
UNKNOWN	2	알 수 없는 오류가 발생했습니다.
INTERNAL	13	내부 서비스 오류가 발생했습니다.

StartModel

AWS IoT Greengrass Version 2 코어 디바이스에서 실행되는 모델을 시작합니다. 모델 실행을 시작하는 데 시간이 걸릴 수 있습니다. 현재 상태를 확인하려면 DescribeModel 항목을 호출하세요. `Status` 필드가 `RUNNING`면 모델이 실행 중인 것입니다.

동시에 실행할 수 있는 모델 수는 코어 디바이스의 하드웨어 사양에 따라 달라집니다.

```
rpc StartModel(StartModelRequest) returns (StartModelResponse);
```

StartModelRequest

```
message StartModelRequest {
  string model_component = 1;
}
```

model_component

시작하려는 모델이 포함된 AWS IoT Greengrass Version 2 구성 요소의 이름입니다.

StartModelResponse

```
message StartModelResponse {
  ModelStatus status = 1;
}
```

status

모델의 현재 상태입니다. 응답은 호출이 성공했을 경우 `STARTING`입니다. 자세한 내용은 ModelStatus 단원을 참조하십시오.

상태 코드

코드	숫자	설명
정상	0	모델이 시작합니다.
UNKNOWN	2	알 수 없는 오류가 발생했습니다.
INVALID_ARGUMENT	3	하나 이상의 입력 파라미터가 유효하지 않습니다. 자세한 내용은 오류 메시지를 확인하세요.
NOT_FOUND	5	제공된 이름을 가진 모델을 찾을 수 없습니다.
RESOURCE_EXHAUSTED	8	리소스가 충분하지 않아 이 작업을 수행할 수 없습니다. 예를 들어, 메모리가 부족하여 모델을 로드할 수 없습니다. 자세한 내용은 오류 메시지를 확인하세요.
FAILED_PRECONDITION	9	중지 또는 실패 상태가 아닌 모델에 대해 메서드가 호출되었습니다.
INTERNAL	13	내부 서비스 오류가 발생했습니다.

StopModel

AWS IoT Greengrass Version 2 코어 디바이스에서 실행 중인 모델을 중지합니다. 모델이 중지되면가를 `StopModel` 반환합니다. 응답의 `Status` 필드가 `STOPPED` 면 모델이 성공적으로 중지된 것입니다.

```
rpc StopModel(StopModelRequest) returns (StopModelResponse);
```

StopModelRequest

```
message StopModelRequest {
  string model_component = 1;
}
```

model_component

중지하려는 모델이 포함된 AWS IoT Greengrass Version 2 구성 요소의 이름입니다.

StopModelResponse

```
message StopModelResponse {
  ModelStatus status = 1;
}
```

status

모델의 현재 상태입니다. 응답은 호출이 성공했을 경우 STOPPED입니다. 자세한 내용은 ModelStatus 단원을 참조하십시오.

상태 코드

코드	숫자	설명
정상	0	모델이 중지되고 있습니다.
UNKNOWN	2	알 수 없는 오류가 발생했습니다.
INVALID_ARGUMENT	3	하나 이상의 입력 파라미터가 유효하지 않습니다. 자세한 내용은 오류 메시지를 확인하세요.
NOT_FOUND	5	제공된 이름을 가진 모델을 찾을 수 없습니다.

코드	숫자	설명
FAILED_PRECONDITION	9	RUNNING 상태가 아닌 모델에 대해 메서드가 호출되었습니다.
INTERNAL	13	내부 서비스 오류가 발생했습니다.

ModelStatus

AWS IoT Greengrass Version 2 코어 디바이스에 배포된 모델의 상태입니다. 현재 상태를 확인하려면 DescribeModel 항목을 호출하세요.

```
enum ModelStatus {
  STOPPED = 0;
  STARTING = 1;
  RUNNING = 2;
  FAILED = 3;
  STOPPING = 4;
}
```

Amazon Lookout for Vision 대시보드 사용

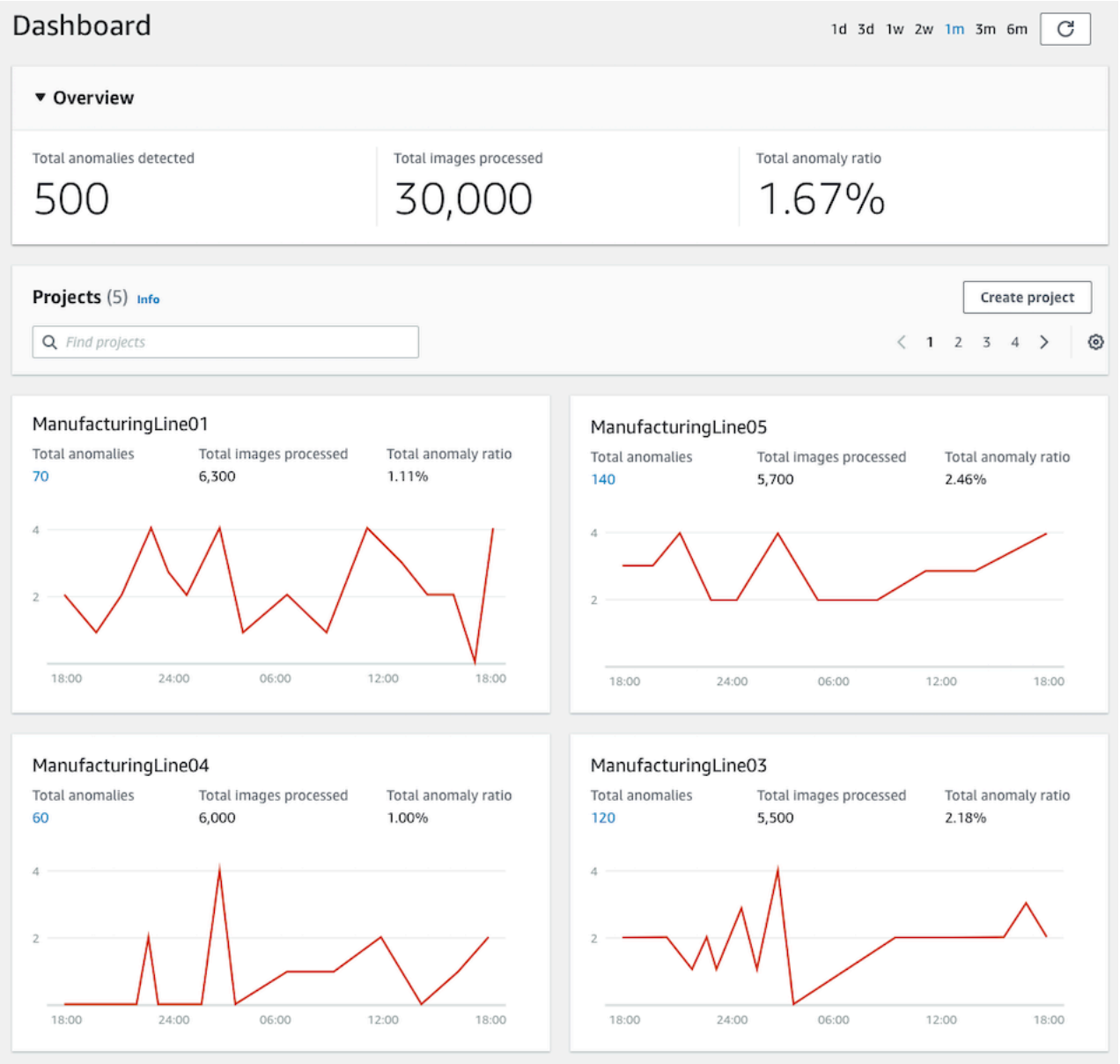
대시보드는 지난 주에 탐지된 총 이상 항목 수와 같은 Amazon Lookout for Vision 프로젝트의 지표 개요를 제공합니다. 대시보드를 통해 모든 프로젝트의 개요와 각 개별 프로젝트에 대한 개요를 볼 수 있습니다. 지표를 표시할 타임라인을 선택할 수 있습니다. 대시보드를 사용해 새 프로젝트를 생성할 수도 있습니다.

개요 섹션에는 총 프로젝트 수, 총 이미지 수, 모든 프로젝트에서 감지한 총 이미지 수가 표시됩니다.

프로젝트 섹션에는 개별 프로젝트에 대한 다음과 같은 개요 정보가 표시됩니다.

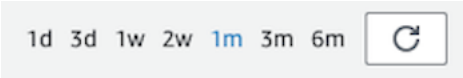
- 탐지된 총 이상 항목 수
- 처리된 총 이미지 수입니다.
- 전체 이상 항목 비율 (즉, 이상 항목이 감지된 이미지의 백분율)
- 그래프는 선택한 기간 동안의 이상 탐지 결과를 보여줍니다.

프로젝트에 대한 추가 정보도 얻을 수 있습니다.

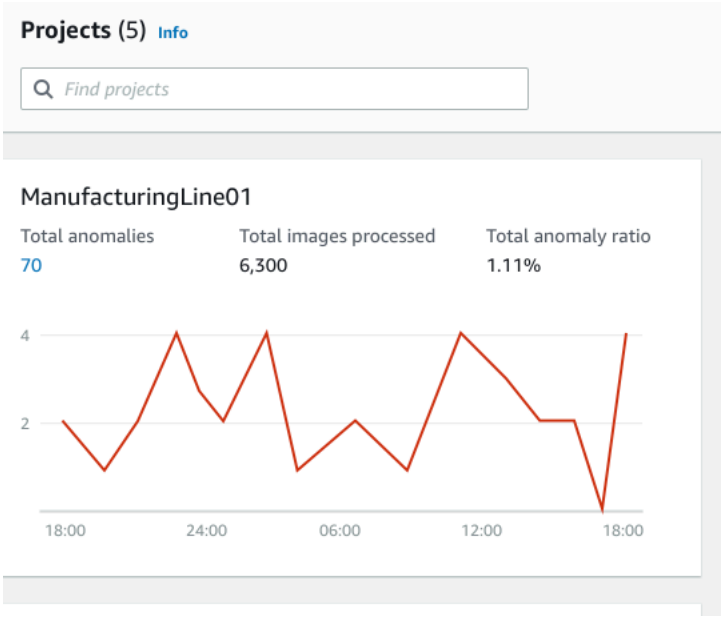


대시보드를 사용하려면

1. https://console.aws.amazon.com/lookoutvision/에서 Amazon Lookout for Vision 콘솔을 엽니다.
2. Get started를 선택합니다.
3. 왼쪽 탐색 창에서 대시보드를 선택합니다.
4. 특정 기간 동안의 지표를 보려면 다음 작업을 수행합니다.
   a. 대시보드 오른쪽 상단에서 기간을 선택합니다.
   b. 새로 고침 버튼을 선택하여 새 타임라인이 포함된 대시보드를 표시합니다.



5. 프로젝트에 대한 추가 세부 정보를 보려면 프로젝트 섹션에서 프로젝트 이름(예:`ManufacturingLine01`)을 선택합니다.



6. 프로젝트를 만들려면 프로젝트 섹션에서 프로젝트 만들기를 선택합니다.

Amazon Lookout for Vision 리소스에 대한 관리

콘솔 또는 AWS SDK를 사용하여 Amazon Lookout for Vision 리소스를 관리할 수 있습니다. Amazon Lookout for Vision에는 다음과 같은 리소스가 있습니다.

- Projects
- 데이터세트
- 모델
- 평가판 확인

> **Note**
>
> 평가판 확인 작업은 삭제할 수 없습니다. 또한 AWS SDK를 사용하여 평가판 감지를 관리할 수 없습니다.

주제

프로젝트 보기

Amazon Lookout for Vision 프로젝트 목록과 개별 프로젝트에 대한 정보는 콘솔 또는 AWS SDK를 사용하여 얻을 수 있습니다.

> **Note**
>
> 프로젝트 목록은 결국 일관성이 있습니다. 프로젝트를 만들거나 삭제하는 경우 프로젝트 목록이 최신 상태가 되기까지 잠시 기다려야 할 수 있습니다.

프로젝트 보기 (콘솔)

콘솔에서 프로젝트를 보려면 다음 절차의 단계를 수행하세요.

프로젝트를 보려면

1. https://console.aws.amazon.com/lookoutvision/에서 Amazon Lookout for Vision 콘솔을 엽니다.
2. Get started를 선택합니다.
3. 왼쪽 탐색 창에서 프로젝트를 선택합니다. 프로젝트 보기가 표시됩니다.
4. 프로젝트 이름을 선택하면 프로젝트 세부 정보를 볼 수 있습니다.

프로젝트 보기 (SDK)

프로젝트는 단일 사용 사례에 맞게 데이터 세트와 모델을 관리합니다. 기계 부품의 이상을 감지하는 경우를 예로 들 수 있습니다. 다음 예시에서는 프로젝트 목록을 가져오기 위해 `ListProjects`을 호출합니다.

프로젝트 (SDK) 를 보려면

1. 아직 설치하지 않은 경우 AWS CLI 및 AWS SDKs를 설치하고 구성합니다. 자세한 내용은 4단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.
2. 다음 예제 코드를 사용하여 프로젝트를 확인하십시오.

   CLI

   `list-projects` 명령어를 사용하여 계정에 있는 프로젝트를 나열합니다.

   ```
   aws lookoutvision list-projects \
     --profile lookoutvision-access
   ```

   `describe-project` 속성을 사용하여 집합에 대한 정보를 가져옵니다.

project-name의 값을 설명하고자 하는 프로젝트 이름으로 변경합니다.

```
aws lookoutvision describe-project --project-name project_name \
  --profile lookoutvision-access
```

Python

이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져옵니다. 전체 예제는 여기에서 확인하세요.

```
    @staticmethod
    def list_projects(lookoutvision_client):
        """
        Lists information about the projects that are in in your AWS account
        and in the current AWS Region.

        :param lookoutvision_client: A Boto3 Lookout for Vision client.
        """
        try:
            response = lookoutvision_client.list_projects()
            for project in response["Projects"]:
                print("Project: " + project["ProjectName"])
                print("\tARN: " + project["ProjectArn"])
                print("\tCreated: " + str(["CreationTimestamp"]))
                print("Datasets")
                project_description = lookoutvision_client.describe_project(
                    ProjectName=project["ProjectName"]
                )
                if not project_description["ProjectDescription"]["Datasets"]:
                    print("\tNo datasets")
                else:
                    for dataset in project_description["ProjectDescription"][
                        "Datasets"
                    ]:
                        print(f"\ttype: {dataset['DatasetType']}")
                        print(f"\tStatus: {dataset['StatusMessage']}")

                print("Models")
                response_models = lookoutvision_client.list_models(
                    ProjectName=project["ProjectName"]
                )
                if not response_models["Models"]:
```

```
                    print("\tNo models")
                else:
                    for model in response_models["Models"]:
                        Models.describe_model(
                            lookoutvision_client,
                            project["ProjectName"],
                            model["ModelVersion"],
                        )


    print("-----------------------------------------------------------\n")
            print("Done!")
        except ClientError:
            logger.exception("Problem listing projects.")
            raise
```

Java V2

이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져옵니다. 전체 예제는 여기에서 확인하세요.

```
/**
 * Lists the Amazon Lookout for Vision projects in the current AWS account and
 AWS
 * Region.
 *
 * @param lfvClient   An Amazon Lookout for Vision client.
 * @param projectName The name of the project that you want to create.
 * @return List<ProjectMetadata> Metadata for each project.
 */
public static List<ProjectMetadata> listProjects(LookoutVisionClient lfvClient)
                throws LookoutVisionException {

        logger.log(Level.INFO, "Getting projects:");
        ListProjectsRequest listProjectsRequest = ListProjectsRequest.builder()
                        .maxResults(100)
                        .build();

        List<ProjectMetadata> projectMetadata = new ArrayList<>();
```

```
        ListProjectsIterable projects =
 lfvClient.listProjectsPaginator(listProjectsRequest);

        projects.stream().flatMap(r -> r.projects().stream())
                        .forEach(project -> {
                                projectMetadata.add(project);
                                logger.log(Level.INFO, project.projectName());
                        });

        logger.log(Level.INFO, "Finished getting projects.");

        return projectMetadata;

}
```

프로젝트 삭제

콘솔의 프로젝트 보기 페이지에서 DeleteProject 작업을 사용하여 프로젝트를 삭제할 수 있습니다.

프로젝트 데이터 세트에서 참조하는 이미지는 삭제되지 않습니다.

프로젝트 삭제(콘솔)

프로젝트를 삭제하려면 다음 절차에 따르세요. 콘솔 절차를 사용하면 관련 모델 버전과 데이터 세트가 자동으로 삭제됩니다.

프로젝트 삭제

1. https://console.aws.amazon.com/lookoutvision/에서 Amazon Lookout for Vision 콘솔을 엽니다.
2. Get started를 선택합니다.
3. 왼쪽 탐색 창에서 프로젝트를 선택합니다.
4. 프로젝트 페이지에서 삭제할 프로젝트를 선택합니다.
5. 페이지 상단에서 삭제를 선택합니다.
6. 삭제 대화 상자에 삭제를 입력하여 프로젝트 삭제를 확인합니다.
7. 필요한 경우 관련 데이터 세트 및 모델을 모두 삭제하도록 선택합니다.
8. 프로젝트 삭제를 선택합니다.

프로젝트 (SDK) 삭제

Amazon Lookout for Vision 프로젝트를 삭제하려면 DeleteProject를 호출하고 삭제하려는 프로젝트의 이름을 입력합니다.

프로젝트를 삭제하려면 먼저 프로젝트에 있는 모든 모델을 삭제해야 합니다. 자세한 내용은 모델 삭제 (SDK) 단원을 참조하십시오. 모델과 관련된 데이터 세트도 삭제해야 합니다. 자세한 내용은 데이터 세트 삭제 단원을 참조하십시오.

프로젝트를 삭제하는 데 다소 시간이 걸릴 수 있습니다. 이 기간 동안의 프로젝트 상태는 `DELETING`입니다. 후속 `DeleteProject`를 호출할 때 삭제한 프로젝트가 포함되지 않으면 프로젝트가 삭제됩니다.

프로젝트 (SDK)를 삭제하려면

1. 아직 설치하지 않은 경우 AWS CLI 및 AWS SDKs를 설치하고 구성합니다. 자세한 내용은 4단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.
2. 프로젝트를 삭제하려면 다음 절차에 따르세요.

   AWS CLI

   `project-name`의 값을 삭제하려는 프로젝트의 이름으로 변경합니다.

   ```
   aws lookoutvision delete-project --project-name project_name \
     --profile lookoutvision-access
   ```

   Python

   이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져옵니다. 전체 예제는 여기에서 확인하세요.

   ```
       @staticmethod
       def delete_project(lookoutvision_client, project_name):
           """
           Deletes a Lookout for Vision Model

           :param lookoutvision_client: A Boto3 Lookout for Vision client.
           :param project_name: The name of the project that you want to delete.
           """
           try:
   ```

```
            logger.info("Deleting project: %s", project_name)
            response =
 lookoutvision_client.delete_project(ProjectName=project_name)
            logger.info("Deleted project ARN: %s ", response["ProjectArn"])
        except ClientError as err:
            logger.exception("Couldn't delete project %s.", project_name)
            raise
```

Java V2

이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져옵니다. 전체 예제는 여기에서 확인하세요.

```
/**
 * Deletes an Amazon Lookout for Vision project.
 *
 * @param lfvClient   An Amazon Lookout for Vision client.
 * @param projectName The name of the project that you want to create.
 * @return String The ARN of the deleted project.
 */
public static String deleteProject(LookoutVisionClient lfvClient, String
 projectName)
                throws LookoutVisionException {

        logger.log(Level.INFO, "Deleting project: {0}", projectName);

        DeleteProjectRequest deleteProjectRequest =
 DeleteProjectRequest.builder()
                        .projectName(projectName)
                        .build();

        DeleteProjectResponse response =
 lfvClient.deleteProject(deleteProjectRequest);

        logger.log(Level.INFO, "Deleted project: {0} ARN: {1}",
                        new Object[] { projectName, response.projectArn() });

        return response.projectArn();

}
```

데이터 세트 보기

프로젝트에는 모델 학습 및 테스트에 사용되는 단일 데이터 세트가 있을 수 있습니다. 또는 학습 데이터 세트와 테스트 데이터 세트를 분리할 수도 있습니다. 콘솔을 사용하여 데이터 세트를 볼 수 있습니다. `DescribeDataset` 작업을 사용하여 데이터 세트에 대한 정보 (훈련 또는 테스트)를 가져올 수도 있습니다.

프로젝트의 데이터 세트 보기 (콘솔)

다음 절차의 단계를 수행하여 콘솔에서 프로젝트 데이터 세트를 봅니다.

데이터 세트를 보려면 (콘솔)

1. https://console.aws.amazon.com/lookoutvision/에서 Amazon Lookout for Vision 콘솔을 엽니다.
2. Get started를 선택합니다.
3. 왼쪽 탐색 창에서 프로젝트를 선택합니다.
4. 프로젝트 페이지에서 보려는 데이터 세트가 포함된 프로젝트를 선택합니다.
5. 왼쪽 탐색 창에서 데이터 세트를 선택하여 데이터 세트 세부 정보를 봅니다. 학습 데이터 세트와 테스트 데이터 세트가 있는 경우 각 데이터 세트의 탭이 표시됩니다.

프로젝트 (SDK)의 데이터 세트 보기

`DescribeDataset` 작업을 사용하여 프로젝트와 관련된 학습 또는 테스트 데이터 세트에 대한 정보를 얻을 수 있습니다.

데이터 세트 (SDK)를 보려면

1. 아직 설치하지 않은 경우 AWS CLI 및 AWS SDKs를 설치하고 구성합니다. 자세한 내용은 4단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.
2. 다음 예제 코드를 사용하여 데이터 세트를 확인하세요.

   CLI

   다음 값을 변경합니다.

   - `project-name`은 확인할 모델이 포함된 프로젝트 이름으로 변경합니다.
   - `dataset-type`은 보려는 데이터 세트 유형(`train`또는`test`)으로 변경합니다.

```
aws lookoutvision describe-dataset --project-name project name\
  --dataset-type train or test \
  --profile lookoutvision-access
```

Python

이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져옵니다. 전체 예제는 여기에서 확인하세요.

```
    @staticmethod
    def describe_dataset(lookoutvision_client, project_name, dataset_type):
        """
        Gets information about a Lookout for Vision dataset.

        :param lookoutvision_client: A Boto3 Lookout for Vision client.
        :param project_name: The name of the project that contains the dataset
 that
                             you want to describe.
        :param dataset_type: The type (train or test) of the dataset that you
 want
                             to describe.
        """
        try:
            response = lookoutvision_client.describe_dataset(
                ProjectName=project_name, DatasetType=dataset_type
            )
            print(f"Name: {response['DatasetDescription']['ProjectName']}")
            print(f"Type: {response['DatasetDescription']['DatasetType']}")
            print(f"Status: {response['DatasetDescription']['Status']}")
            print(f"Message: {response['DatasetDescription']['StatusMessage']}")
            print(f"Images: {response['DatasetDescription']['ImageStats']
['Total']}")
            print(f"Labeled: {response['DatasetDescription']['ImageStats']
['Labeled']}")
            print(f"Normal: {response['DatasetDescription']['ImageStats']
['Normal']}")
            print(f"Anomaly: {response['DatasetDescription']['ImageStats']
['Anomaly']}")
        except ClientError:
            logger.exception("Service error: problem listing datasets.")
            raise
```

```
        print("Done.")
```

Java V2

이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져옵니다. 전체 예제는 여기에서 확인하세요.

```
/**
 * Gets the description for a Amazon Lookout for Vision dataset.
 *
 * @param lfvClient   An Amazon Lookout for Vision client.
 * @param projectName The name of the project in which you want to describe a
 *                    dataset.
 * @param datasetType The type of the dataset that you want to describe (train
 *                    or test).
 * @return DatasetDescription A description of the dataset.
 */
public static DatasetDescription describeDataset(LookoutVisionClient lfvClient,
                String projectName,
                String datasetType) throws LookoutVisionException {

        logger.log(Level.INFO, "Describing {0} dataset for project {1}",
                        new Object[] { datasetType, projectName });

        DescribeDatasetRequest describeDatasetRequest =
 DescribeDatasetRequest.builder()
                        .projectName(projectName)
                        .datasetType(datasetType)
                        .build();

        DescribeDatasetResponse describeDatasetResponse =
 lfvClient.describeDataset(describeDatasetRequest);
        DatasetDescription datasetDescription =
 describeDatasetResponse.datasetDescription();

        logger.log(Level.INFO, "Project: {0}\n"
                        + "Created: {1}\n"
                        + "Type: {2}\n"
                        + "Total: {3}\n"
                        + "Labeled: {4}\n"
                        + "Normal: {5}\n"
                        + "Anomalous: {6}\n",
```

```
                        new Object[] {
                                        datasetDescription.projectName(),
                                        datasetDescription.creationTimestamp(),
                                        datasetDescription.datasetType(),

 datasetDescription.imageStats().total().toString(),

 datasetDescription.imageStats().labeled().toString(),

 datasetDescription.imageStats().normal().toString(),

 datasetDescription.imageStats().anomaly().toString(),
                        });

        return datasetDescription;

}
```

데이터 세트에 이미지 추가

데이터 세트를 생성한 후 데이터 세트에 이미지를 더 추가할 수 있습니다. 예를 들어 모델 평가 결과가 좋지 않은 것으로 확인되면 이미지를 더 추가하여 모델의 품질을 개선할 수 있습니다. 테스트 데이터 세트를 만든 경우 이미지를 더 추가하면 모델 성능 지표의 정확도를 높일 수 있습니다.

데이터 세트를 업데이트한 후 모델을 재학습하세요.

주제

- 더 많은 이미지 추가
- 더 많은 이미지 추가 (SDK)

더 많은 이미지 추가

로컬 컴퓨터에서 이미지를 업로드하여 데이터 세트에 이미지를 더 추가할 수 있습니다. SDK로 라벨이 있는 이미지를 더 추가하려면 UpdateDataSetEntries 작업을 사용하세요.

데이터 세트에 이미지를 더 추가하려면 (콘솔)

1. 작업을 선택하고 이미지를 추가할 데이터 세트를 선택합니다.

2. 데이터 세트에 업로드할 이미지를 선택합니다. 로컬 컴퓨터에서 업로드할 이미지를 선택하거나 드래그할 수 있습니다. 한 번에 최대 30개의 이미지를 업로드할 수 있습니다.
3. 이미지 업로드를 선택합니다.
4. 변경 사항 저장을 선택합니다.

이미지를 더 추가했으면 모델을 학습시키는 데 사용할 수 있도록 이미지에 레이블을 지정해야 합니다. 자세한 내용은 이미지 분류 (콘솔) 단원을 참조하십시오.

더 많은 이미지 추가 (SDK)

SDK로 라벨이 있는 이미지를 더 추가하려면 UpdateDataSetEntries 작업을 사용하세요. 추가하려는 이미지가 포함된 매니페스트 파일을 제공합니다. 매니페스트 파일의 JSON 라인 `source-ref` 필드에 이미지를 지정하여 기존 이미지를 업데이트할 수도 있습니다. 자세한 내용은 매니페스트 파일 만들기 단원을 참조하십시오.

데이터세트 (SDK)에 더 많은 이미지를 추가하려면

1. 아직 설치하지 않은 경우 AWS CLI 및 AWS SDKs를 설치하고 구성합니다. 자세한 내용은 4단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.
2. 다음 예제 코드를 사용하여 데이터 세트에 이미지를 더 추가하세요.

   CLI

   다음 값을 변경합니다.

   - `project-name`은 업데이트하려는 데이터 세트가 포함된 프로젝트 이름으로 변경합니다.
   - `dataset-type`은 업데이트하려는 데이터 세트 유형 (`train`또는`test`)으로 변경합니다.
   - `changes`은 데이터 세트가 포함된 매니페스트 파일이 업데이트되는 위치로 변경합니다.

```
aws lookoutvision update-dataset-entries\
    --project-name project\
    --dataset-type train or test\
    --changes fileb://manifest file \
    --profile lookoutvision-access
```

Python

이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져옵니다. 전체 예제는 여기에서 확인하세요.

```
    @staticmethod
    def update_dataset_entries(lookoutvision_client, project_name, dataset_type,
 updates_file):
        """
        Adds dataset entries to an Amazon Lookout for Vision dataset.
        :param lookoutvision_client: The Amazon Rekognition Custom Labels Boto3
 client.
        :param project_name: The project that contains the dataset that you want
 to update.
        :param dataset_type: The type of the dataset that you want to update
 (train or test).
        :param updates_file: The manifest file of JSON Lines that contains the
 updates.
        """

        try:
            status = ""
            status_message = ""
            manifest_file = ""

            # Update dataset entries
            logger.info(f"""Updating {dataset_type} dataset for project
 {project_name}
with entries from {updates_file}.""")

            with open(updates_file) as f:
                manifest_file = f.read()

            lookoutvision_client.update_dataset_entries(
                ProjectName=project_name,
                DatasetType=dataset_type,
                Changes=manifest_file,
            )

            finished = False
            while finished == False:
```

```
                dataset =
lookoutvision_client.describe_dataset(ProjectName=project_name,

DatasetType=dataset_type)

                status = dataset['DatasetDescription']['Status']
                status_message = dataset['DatasetDescription']['StatusMessage']

                if status == "UPDATE_IN_PROGRESS":
                    logger.info(
                        (f"Updating {dataset_type} dataset for project
{project_name}."))
                    time.sleep(5)
                    continue

                if status == "UPDATE_FAILED_ROLLBACK_IN_PROGRESS":
                    logger.info(
                        (f"Update failed, rolling back {dataset_type} dataset
for project {project_name}."))
                    time.sleep(5)
                    continue

                if status == "UPDATE_COMPLETE":
                    logger.info(
                        f"Dataset updated: {status} : {status_message} :
{dataset_type} dataset for project {project_name}.")
                    finished = True
                    continue

                if status == "UPDATE_FAILED_ROLLBACK_COMPLETE":
                    logger.info(
                        f"Rollback complated after update failure: {status} :
{status_message} : {dataset_type} dataset for project {project_name}.")
                    finished = True
                    continue

                logger.exception(
                    f"Failed. Unexpected state for dataset update: {status} :
{status_message} : {dataset_type} dataset for project {project_name}.")
                raise Exception(
                    f"Failed. Unexpected state for dataset update: {status} :
{status_message} :{dataset_type} dataset for project {project_name}.")

            logger.info(f"Added entries to dataset.")
```

```
            return status, status_message

        except ClientError as err:
            logger.exception(
                f"Couldn't update dataset: {err.response['Error']['Message']}")
            raise
```

Java V2

이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져옵니다. 전체 예제는 여기에서 확인하세요.

```
/**
 * Updates an Amazon Lookout for Vision dataset from a manifest file.
 * Returns after Lookout for Vision updates the dataset.
 *
 * @param lfvClient    An Amazon Lookout for Vision client.
 * @param projectName  The name of the project in which you want to update a
 *                     dataset.
 * @param datasetType  The type of the dataset that you want to update (train or
 *                     test).
 * @param manifestFile The name and location of a local manifest file that you
 want to
 * use to update the dataset.
 * @return DatasetStatus The status of the updated dataset.
 */

public static DatasetStatus updateDatasetEntries(LookoutVisionClient lfvClient,
 String projectName,
                String datasetType, String updateFile) throws
 FileNotFoundException, LookoutVisionException,
                InterruptedException {

        logger.log(Level.INFO, "Updating {0} dataset for project {1}",
                        new Object[] { datasetType, projectName });

        InputStream sourceStream = new FileInputStream(updateFile);
        SdkBytes sourceBytes = SdkBytes.fromInputStream(sourceStream);

        UpdateDatasetEntriesRequest updateDatasetEntriesRequest =
 UpdateDatasetEntriesRequest.builder()
                        .projectName(projectName)
```

```
                        .datasetType(datasetType)
                        .changes(sourceBytes)
                        .build();

        lfvClient.updateDatasetEntries(updateDatasetEntriesRequest);

        boolean finished = false;
        DatasetStatus status = null;

        // Wait until update completes.

        do {

                DescribeDatasetRequest describeDatasetRequest =
DescribeDatasetRequest.builder()
                                .projectName(projectName)
                                .datasetType(datasetType)
                                .build();
                DescribeDatasetResponse describeDatasetResponse = lfvClient
                                .describeDataset(describeDatasetRequest);

                DatasetDescription datasetDescription =
describeDatasetResponse.datasetDescription();

                status = datasetDescription.status();

                switch (status) {

                        case UPDATE_COMPLETE:
                                logger.log(Level.INFO, "{0} Dataset updated for
project {1}.",
                                                new Object[] { datasetType,
projectName });
                                finished = true;
                                break;

                        case UPDATE_IN_PROGRESS:

                                logger.log(Level.INFO, "{0} Dataset update for
project {1} in progress.",
                                                new Object[] { datasetType,
projectName });
                                TimeUnit.SECONDS.sleep(5);
```

```
                                break;

                        case UPDATE_FAILED_ROLLBACK_IN_PROGRESS:
                                logger.log(Level.SEVERE,
                                                "{0} Dataset update failed for
 project {1}. Rolling back",
                                                new Object[] { datasetType,
 projectName });

                                TimeUnit.SECONDS.sleep(5);

                                break;

                        case UPDATE_FAILED_ROLLBACK_COMPLETE:
                                logger.log(Level.SEVERE,
                                                "{0} Dataset update failed for
 project {1}. Rollback completed.",
                                                new Object[] { datasetType,
 projectName });
                                finished = true;
                                break;

                        default:
                                logger.log(Level.SEVERE,
                                                "{0} Dataset update failed for
 project {1}. Unexpected error returned.",
                                                new Object[] { datasetType,
 projectName });
                                finished = true;

                }

        } while (!finished);

        return status;

}
```

3. 이전 단계를 반복하고 다른 데이터 세트 유형의 값을 제공합니다.

데이터 세트에서 이미지 삭제하기

데이터 세트에서 직접 이미지를 삭제할 수는 없습니다. 대신 기존 데이터 세트를 삭제하고 제거하려는 이미지가 없는 새 데이터 세트를 만들어야 합니다. 이미지를 제거하는 방법은 이미지를 기존 데이터 세트 (매니페스트 파일, Amazon S3 버킷 또는 로컬 컴퓨터)로 가져온 방법에 따라 다릅니다.

AWS SDK를 사용하여 이미지를 제거할 수도 있습니다. 이는 이미지 분할 매니페스트 파일 없이 이미지 분할 모델을 생성할 때 유용하므로 Amazon Lookout for Vision 콘솔을 사용하여 이미지 마스크를 다시 그릴 필요가 없습니다.

주제

- 데이터 세트에서 이미지 제거 (콘솔)
- 데이터세트 (SDK) 에서 이미지 제거

데이터 세트에서 이미지 제거 (콘솔)

Amazon Lookout for Vision 콘솔을 사용하여 데이터 세트 에서 이미지를 제거하려면 다음 절차를 따르십시오.

데이터 세트에서 이미지를 제거하려면 (콘솔)

1. 프로젝트의 데이터 세트 갤러리를 엽니다.
2. 제거하려는 각 이미지의 이름을 기록해 둡니다.
3. 기존 데이터 세트를 삭제합니다.
4. 다음 중 하나를 수행합니다.
   - 매니페스트 파일로 데이터세트를 만든 경우, 다음과 같이 하세요.
     a. 텍스트 편집기에서 데이터세트를 만드는 데 사용한 매니페스트 파일을 엽니다.
     b. 2단계에서 기록해 둔 각 이미지의 JSON 라인을 제거합니다. `source-ref` 필드를 확인하여 이미지의 JSON 라인을 식별할 수 있습니다.
     c. 매니페스트 파일을 저장합니다.
     d. 업데이트된 매니페스트 파일로 새 데이터세트를 생성하세요.
   - Amazon S3 버킷에서 가져온 이미지로 데이터 세트를 생성한 경우 다음을 수행합니다.
     a. 2단계에서 메모한 이미지를 Amazon S3 버킷에서 삭제합니다.

b. Amazon S3 버킷에 남아 있는 이미지를 사용하여 새 데이터 세트를 생성합니다. 폴더 이름으로 이미지를 분류하는 경우 다음 단계에서 이미지를 분류할 필요가 없습니다.

c. 다음 중 하나를 수행합니다.

- 이미지 분류 모델을 만드는 경우 레이블이 지정되지 않은 각 이미지를 분류하십시오.
- 이미지 분할 모델을 만드는 경우 레이블이 지정되지 않은 각 이미지를 분류하고 분할하십시오.

- Amazon S3 버킷에서 가져온 이미지로 데이터 세트를 생성한 경우 다음을 수행합니다.

a. 컴퓨터에 사용하려는 이미지가 들어 있는 폴더를 만드세요. 데이터 세트에서 제거하려는 이미지를 포함하지 마세요. 자세한 내용은 로컬 컴퓨터에 저장된 이미지를 사용하여 데이터 세트 만들기 단원을 참조하십시오.

b. 4.a단계에서 만든 폴더의 이미지가 포함된 데이터 세트를 생성합니다.

c. 다음 중 하나를 수행합니다.

- 이미지 분류 모델을 만드는 경우 레이블이 지정되지 않은 각 이미지를 분류하십시오.
- 이미지 분할 모델을 만드는 경우 레이블이 지정되지 않은 각 이미지를 분류하고 분할하십시오.

5. 모델을 훈련시키십시오.

데이터세트 (SDK) 에서 이미지 제거

AWS SDK를 사용하여 데이터 세트에서 이미지를 제거할 수 있습니다.

데이터 세트에서 이미지를 제거하려면

1. 프로젝트의 데이터 세트 갤러리를 엽니다.
2. 제거하려는 각 이미지의 이름을 기록해 둡니다.
3. ListDataSetEntries 작업을 사용하여 데이터 세트의 JSON 라인을 내보냅니다.
4. 내보낸 JSON 라인을 사용하여 매니페스트 파일을 생성합니다.
5. 텍스트 편집기에서 매니니페스트 파일을 엽니다.
6. 2단계에서 기록해 둔 각 이미지의 JSON 라인을 제거합니다. `source-ref` 필드를 확인하여 이미지의 JSON 라인을 식별할 수 있습니다.
7. 매니페스트 파일을 저장합니다.

8. 기존 데이터 세트를 삭제합니다.
9. 업데이트된 매니페스트 파일로 새 데이터세트를 생성세요.
10. 모델을 훈련시키십시오.

데이터 세트 삭제

콘솔 또는 `DeleteDataset` 작업을 사용하여 프로젝트에서 데이터 세트를 삭제할 수 있습니다. 데이터세트에서 참조하는 이미지는 삭제되지 않습니다. 학습 데이터 세트와 테스트 데이터 세트가 있는 프로젝트에서 테스트 데이터 세트를 삭제하면 프로젝트는 단일 데이터 세트 프로젝트로 되돌아갑니다. 나머지 데이터 세트는 훈련 중에 분할되어 훈련 데이터 세트와 테스트 데이터 세트가 만들어집니다. 훈련 데이터 세트를 삭제하면 새 훈련 데이터 세트를 만들 때까지 프로젝트에서 모델을 학습시킬 수 없습니다.

데이터 세트 삭제 (콘솔)

데이터 세트를 삭제하려면 다음 절차에 따라 단계를 수행합니다. 프로젝트의 모든 데이터세트를 삭제하면 데이터세트 만들기 페이지가 표시됩니다.

데이터 세트 (콘솔)를 삭제하려면

1. https://console.aws.amazon.com/lookoutvision/에서 Amazon Lookout for Vision 콘솔을 엽니다.
2. Get started를 선택합니다.
3. 왼쪽 탐색 창에서 프로젝트를 선택합니다.
4. 프로젝트 페이지에서 삭제하려는 데이터 세트가 포함된 프로젝트를 선택합니다.
5. 왼쪽 탐색 창에서 데이터 세트를 선택합니다.
6. 작업을 선택한 다음 삭제할 데이터 세트를 선택합니다.
7. 삭제 대화 상자에 삭제를 입력하여 데이터 세트 삭제를 확인합니다.
8. 학습 데이터 세트 삭제 또는 테스트 데이터 세트 삭제를 선택하여 데이터 세트를 삭제합니다.

데이터 세트 (SDK) 삭제

`DeleteDataset` 작업을 사용하여 데이터 세트를 삭제합니다.

데이터 세트 (SDK)를 삭제하려면

1. 아직 설치하지 않은 경우 AWS CLI 및 AWS SDKs를 설치하고 구성합니다. 자세한 내용은 4단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.
2. 모델을 삭제하려면 다음 예제 코드를 사용합니다.

   CLI

   다음 값을 변경합니다.

   - `project-name`은 삭제할 모델이 포함된 프로젝트 이름으로 변경합니다.
   - `dataset-type`은 삭제하려는 데이터 세트에 따라 `train` 또는 `test` 둘 중 하나로 변경합니다. 데이터 세트 프로젝트가 하나뿐인 경우 데이터 세트를 삭제하도록 `train`로 지정하세요.

```
aws lookoutvision delete-dataset --project-name project name\
  --dataset-type dataset type \
  --profile lookoutvision-access
```

   Python

   이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져옵니다. 전체 예제는 여기에서 확인하세요.

```
    @staticmethod
    def delete_dataset(lookoutvision_client, project_name, dataset_type):
        """
        Deletes a Lookout for Vision dataset

        :param lookoutvision_client: A Boto3 Lookout for Vision client.
        :param project_name: The name of the project that contains the dataset
that
                             you want to delete.
        :param dataset_type: The type (train or test) of the dataset that you
                             want to delete.
        """
        try:
            logger.info(
                "Deleting the %s dataset for project %s.", dataset_type,
project_name
```

```
            )
            lookoutvision_client.delete_dataset(
                ProjectName=project_name, DatasetType=dataset_type
            )
            logger.info("Dataset deleted.")
        except ClientError:
            logger.exception("Service error: Couldn't delete dataset.")
            raise
```

Java V2

이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져옵니다. 전체 예제는 여기에서 확인하세요.

```
/**
 * Deletes the train or test dataset in an Amazon Lookout for Vision project.
 *
 * @param lfvClient   An Amazon Lookout for Vision client.
 * @param projectName The name of the project in which you want to delete a
 *                    dataset.
 * @param datasetType The type of the dataset that you want to delete (train or
 *                    test).
 * @return Nothing.
 */
public static void deleteDataset(LookoutVisionClient lfvClient, String
 projectName, String datasetType)
                throws LookoutVisionException {

        logger.log(Level.INFO, "Deleting {0} dataset for project {1}",
                        new Object[] { datasetType, projectName });

        DeleteDatasetRequest deleteDatasetRequest =
 DeleteDatasetRequest.builder()
                        .projectName(projectName)
                        .datasetType(datasetType)
                        .build();

        lfvClient.deleteDataset(deleteDatasetRequest);

        logger.log(Level.INFO, "Deleted {0} dataset for project {1}",
                        new Object[] { datasetType, projectName });
```

```
}
```

프로젝트 (SDK) 에서 데이터 세트 내보내기

AWS SDK를 사용하여 Amazon Lookout for Vision 프로젝트에서 Amazon S3 버킷 위치로 데이터 세트를 내보낼 수 있습니다.

데이터 세트를 내보내면 소스 프로젝트의 데이터 세트 복사본을 사용하여 Lookout for Vision 프로젝트를 만드는 등의 작업을 수행할 수 있습니다. 특정 버전의 모델에 사용된 데이터세트의 스냅샷을 만들 수도 있습니다.

이 절차의 Python 코드는 프로젝트의 학습 데이터 세트 (매니페스트 및 데이터 세트 이미지)를 지정한 대상 Amazon S3 위치로 내보냅니다. 프로젝트에 있는 경우 코드는 테스트 데이터 세트 매니페스트와 데이터 세트 이미지도 내보냅니다. 대상은 원본 프로젝트와 동일한 Amazon S3 버킷 또는 다른 Amazon S3 버킷에 있을 수 있습니다. 코드는 ListDataSetEntries 작업을 사용하여 데이터세트 매니페스트 파일을 가져옵니다. Amazon S3 작업은 데이터세트 이미지와 업데이트된 매니페스트 파일을 대상 Amazon S3 위치에 복사합니다.

이 절차는 프로젝트의 데이터세트를 내보내는 방법을 보여줍니다. 내보낸 데이터 세트를 사용하여 새 프로젝트를 생성하는 방법도 보여줍니다.

프로젝트 (SDK)에서 데이터세트를 내보내려면

1. 아직 설치하지 않은 경우 AWS CLI 및 AWS SDKs를 설치하고 구성합니다. 자세한 내용은 4단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.
2. 데이터세트 내보내기에 사용할 대상 Amazon S3 경로를 결정합니다. 목적지가 Amazon Lookout for Vision이 지원하는 AWS 지역에 있는지 확인하십시오. 새 Amazon S3 버킷을 만드는 방법은 버킷 생성을 참조하십시오.
3. 사용자에게 데이터세트 내보내기를 위한 대상 Amazon S3 경로와 소스 프로젝트 데이터세트의 이미지 파일에 대한 S3 위치에 대한 액세스 권한이 있는지 확인하십시오. 이미지 파일이 어느 위치에나 있을 수 있다고 가정하는 다음 정책을 사용할 수 있습니다. *##/##*를 데이터세트를 내보낼 대상 버킷 및 경로로 바꾸십시오.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PutExports",
```

```
            "Effect": "Allow",
            "Action": [
                "S3:PutObjectTagging",
                "S3:PutObject"
            ],
            "Resource": "arn:aws:s3:::bucket/path/*"
        },
        {
            "Sid": "GetSourceRefs",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectTagging",
                "s3:GetObjectVersion"
            ],
            "Resource": "*"
        }
    ]
}
```

액세스 권한을 제공하려면 사용자, 그룹 또는 역할에 권한을 추가하세요:

- 의 사용자 및 그룹 AWS IAM Identity Center:

  권한 세트를 생성합니다. AWS IAM Identity Center 사용 설명서의 권한 세트 생성의 지침을 따릅니다.

- 보안 인증 공급자를 통해 IAM에서 관리되는 사용자:

  ID 페더레이션을 위한 역할을 생성합니다. IAM 사용 설명서의 Create a role for a third-party identity provider (federation)의 지침을 따릅니다.

- IAM 사용자:
  - 사용자가 맡을 수 있는 역할을 생성합니다. IAM 사용 설명서에서 Create a role for an IAM user의 지침을 따릅니다.
  - (권장되지 않음)정책을 사용자에게 직접 연결하거나 사용자를 사용자 그룹에 추가합니다. IAM 사용 설명서에서 사용자(콘솔)에 권한 추가의 지침을 따르세요.

4. 다음 코드를 `dataset_export.py` 이름의 파일에 저장합니다.

```
"""
Purpose
```

```
Shows how to export the datasets (manifest files and images)
from an Amazon Lookout for Vision project to a new Amazon
S3 location.
"""

import argparse
import json
import logging

import boto3
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)


def copy_file(s3_resource, source_file, destination_file):
    """
    Copies a file from a source Amazon S3 folder to a destination
    Amazon S3 folder.
    The destination can be in a different S3 bucket.
    :param s3: An Amazon S3 Boto3 resource.
    :param source_file: The Amazon S3 path to the source file.
    :param destination_file: The destination Amazon S3 path for
    the copy operation.
    """

    source_bucket, source_key = source_file.replace("s3://", "").split("/", 1)
    destination_bucket, destination_key = destination_file.replace("s3://",
 "").split(
        "/", 1
    )

    try:
        bucket = s3_resource.Bucket(destination_bucket)
        dest_object = bucket.Object(destination_key)
        dest_object.copy_from(CopySource={"Bucket": source_bucket, "Key":
 source_key})
        dest_object.wait_until_exists()
        logger.info("Copied %s to %s", source_file, destination_file)
    except ClientError as error:
        if error.response["Error"]["Code"] == "404":
            error_message = (
                f"Failed to copy {source_file} to "
```

```
                f"{destination_file}. : {error.response['Error']['Message']}"
            )
            logger.warning(error_message)
            error.response["Error"]["Message"] = error_message
        raise


def upload_manifest_file(s3_resource, manifest_file, destination):
    """
    Uploads a manifest file to a destination Amazon S3 folder.
    :param s3: An Amazon S3 Boto3 resource.
    :param manifest_file: The manifest file that you want to upload.
    :destination: The Amazon S3 folder location to upload the manifest
    file to.
    """

    destination_bucket, destination_key = destination.replace("s3://",
 "").split("/", 1)

    bucket = s3_resource.Bucket(destination_bucket)

    put_data = open(manifest_file, "rb")
    obj = bucket.Object(destination_key + manifest_file)

    try:
        obj.put(Body=put_data)
        obj.wait_until_exists()
        logger.info("Put manifest file '%s' to bucket '%s'.", obj.key,
 obj.bucket_name)
    except ClientError:
        logger.exception(
            "Couldn't put manifest file '%s' to bucket '%s'.", obj.key,
 obj.bucket_name
        )
        raise
    finally:
        if getattr(put_data, "close", None):
            put_data.close()


def get_dataset_types(lookoutvision_client, project):
    """
    Determines the types of the datasets (train or test) in an
    Amazon Lookout for Vision project.
```

```
    :param lookoutvision_client: A Lookout for Vision Boto3 client.
    :param project: The Lookout for Vision project that you want to check.
    :return: The dataset types in the project.
    """

    try:
        response = lookoutvision_client.describe_project(ProjectName=project)

        datasets = []

        for dataset in response["ProjectDescription"]["Datasets"]:
            if dataset["Status"] in ("CREATE_COMPLETE", "UPDATE_COMPLETE"):
                datasets.append(dataset["DatasetType"])
        return datasets

    except lookoutvision_client.exceptions.ResourceNotFoundException:
        logger.exception("Project %s not found.", project)
        raise


def process_json_line(s3_resource, entry, dataset_type, destination):
    """
    Creates a JSON line for a new manifest file, copies image and mask to
    destination.
    :param s3_resource: An Amazon S3 Boto3 resource.
    :param entry: A JSON line from the manifest file.
    :param dataset_type: The type (train or test) of the dataset that
    you want to create the manifest file for.
    :param destination: The destination Amazon S3 folder for the manifest
    file and dataset images.
    :return: A JSON line with details for the destination location.
    """
    entry_json = json.loads(entry)

    print(f"source: {entry_json['source-ref']}")

    # Use existing folder paths to ensure console added image names don't clash.
    bucket, key = entry_json["source-ref"].replace("s3://", "").split("/", 1)
    logger.info("Source location: %s/%s", bucket, key)

    destination_image_location = destination + dataset_type + "/images/" + key

    copy_file(s3_resource, entry_json["source-ref"], destination_image_location)
```

```
    # Update JSON for writing.
    entry_json["source-ref"] = destination_image_location

    if "anomaly-mask-ref" in entry_json:
        source_anomaly_ref = entry_json["anomaly-mask-ref"]
        mask_bucket, mask_key = source_anomaly_ref.replace("s3://", "").split("/",
 1)

        destination_mask_location = destination + dataset_type + "/masks/" +
 mask_key
        entry_json["anomaly-mask-ref"] = destination_mask_location

        copy_file(s3_resource, source_anomaly_ref, entry_json["anomaly-mask-ref"])

    return entry_json


def write_manifest_file(
    lookoutvision_client, s3_resource, project, dataset_type, destination
):
    """
    Creates a manifest file for a dataset. Copies the manifest file and
    dataset images (and masks, if present) to the specified Amazon S3 destination.
    :param lookoutvision_client: A Lookout for Vision Boto3 client.
    :param project: The Lookout for Vision project that you want to use.
    :param dataset_type: The type (train or test) of the dataset that
    you want to create the manifest file for.
    :param destination: The destination Amazon S3 folder for the manifest file
    and dataset images.
    """

    try:
        # Create a reusable Paginator
        paginator = lookoutvision_client.get_paginator("list_dataset_entries")

        # Create a PageIterator from the Paginator
        page_iterator = paginator.paginate(
            ProjectName=project,
            DatasetType=dataset_type,
            PaginationConfig={"PageSize": 100},
        )

        output_manifest_file = dataset_type + ".manifest"
```

```
        # Create manifest file then upload to Amazon S3 with images.
        with open(output_manifest_file, "w", encoding="utf-8") as manifest_file:
            for page in page_iterator:
                for entry in page["DatasetEntries"]:
                    try:
                        entry_json = process_json_line(
                            s3_resource, entry, dataset_type, destination
                        )

                        manifest_file.write(json.dumps(entry_json) + "\n")

                    except ClientError as error:
                        if error.response["Error"]["Code"] == "404":
                            print(error.response["Error"]["Message"])
                            print(f"Excluded JSON line: {entry}")
                        else:
                            raise
        upload_manifest_file(
            s3_resource, output_manifest_file, destination + "datasets/"
        )

    except ClientError:
        logger.exception("Problem getting dataset_entries")
        raise


def export_datasets(lookoutvision_client, s3_resource, project, destination):
    """
    Exports the datasets from an Amazon Lookout for Vision project to a specified
    Amazon S3 destination.
    :param project: The Lookout for Vision project that you want to use.
    :param destination: The destination Amazon S3 folder for the exported datasets.
    """
    # Add trailing backslash, if missing.
    destination = destination if destination[-1] == "/" else destination + "/"

    print(f"Exporting project {project} datasets to {destination}.")

    # Get each dataset and export to destination.

    dataset_types = get_dataset_types(lookoutvision_client, project)
    for dataset in dataset_types:
        logger.info("Copying %s dataset to %s.", dataset, destination)
```

```
        write_manifest_file(
            lookoutvision_client, s3_resource, project, dataset, destination
        )

    print("Exported dataset locations")
    for dataset in dataset_types:
        print(f"   {dataset}: {destination}datasets/{dataset}.manifest")

    print("Done.")


def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """

    parser.add_argument("project", help="The project that contains the dataset.")
    parser.add_argument("destination", help="The destination Amazon S3 folder.")


def main():
    """
    Exports the datasets from an Amazon Lookout for Vision project to a
    destination Amazon S3 location.
    """
    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")
    parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
    add_arguments(parser)

    args = parser.parse_args()

    try:
        session = boto3.Session(profile_name="lookoutvision-access")
        lookoutvision_client = session.client("lookoutvision")
        s3_resource = session.resource("s3")

        export_datasets(
            lookoutvision_client, s3_resource, args.project, args.destination
        )
    except ClientError as err:
        logger.exception(err)
        print(f"Failed: {format(err)}")
```

```
if __name__ == "__main__":
    main()
```

5. 코드를 실행합니다. 다음 명령줄 인수를 제공하세요.

   - 프로젝트 — 내보내려는 데이터 세트가 포함된 소스 프로젝트의 이름입니다.
   - 대상 — 데이터 세트의 대상 Amazon S3 경로입니다.

   예제: `python dataset_export.py myproject s3://bucket/path/`
6. 코드에 표시되는 매니페스트 파일 위치를 기록해 둡니다. 8단계에서 이 이름이 필요합니다.
7. 프로젝트를 생성합니다.의 지침에 따라 내보낸 데이터 세트로 새 Lookout for Vision 프로젝트를 만드십시오.
8. 다음 중 하나를 수행합니다.
   - 매니페스트 파일로 데이터세트 만들기 (콘솔)의 지침에 따라 Lookout for Vision 콘솔을 사용하여 새 프로젝트에 사용할 데이터 세트를 만들 수 있습니다. 1~6단계를 수행하지 않아도 됩니다.

     12단계의 경우 다음을 수행합니다:

     a. 소스 프로젝트에 테스트 데이터세트가 있는 경우 훈련 데이터세트와 테스트 데이터세트 분리를 선택하고, 그렇지 않으면 단일 데이터세트를 선택하세요.

     b. .manifest 파일 위치에는 6단계에서 기록해 둔 적절한 매니페스트 파일 (훈련 또는 테스트)의 위치를 입력합니다.
   - CreateDataset 작업을 사용하면 매니페스트 파일 (SDK)로 데이터세트 만들기의 코드를 사용하여 새 프로젝트의 데이터세트를 만들 수 있습니다. `manifest_file` 파라미터에는 6단계에서 기록해 둔 매니페스트 파일 위치를 사용하세요. 소스 프로젝트에 테스트 데이터세트가 있는 경우 코드를 다시 사용하여 테스트 데이터세트를 만드세요.
9. 준비가 되었으면 모델 학습의 지침에 따라 모델을 학습시키세요.

모델 보기

한 프로젝트에는 모델의 여러 버전이 존재할 수 있습니다. 콘솔을 사용하여 프로젝트의 모델을 볼 수 있습니다. `ListModels` 작업을 사용할 수도 있습니다.

> **Note**
>
> 모델 목록은 결과의 일관성이 있습니다. 모델을 생성할 경우 모델 목록이 최신 상태가 될 때까지 잠시 기다려야 할 수 있습니다.

모델 보기 (콘솔)

다음 절차의 단계를 수행하여 콘솔에서 프로젝트 모델을 봅니다.

모델을 보려면 (콘솔)

1. https://console.aws.amazon.com/lookoutvision/에서 Amazon Lookout for Vision 콘솔을 엽니다.
2. Get started를 선택합니다.
3. 왼쪽 탐색 창에서 프로젝트를 선택합니다.
4. 프로젝트 페이지에서 보려는 모델이 포함된 프로젝트를 선택합니다.
5. 왼쪽 탐색 창에서 모델 을 선택한 다음 모델 세부 정보를 봅니다.

모델 보기 (SDK)

모델의 버전을 보려면 `ListModels` 작업을 사용합니다. 특정 모델 버전에 대한 정보를 가져오려면 `DescribeModel` 작업을 사용합니다. 다음 예제는 프로젝트의 모든 모델 버전을 나열한 다음 개별 모델 버전의 성능 및 출력 구성 정보를 표시합니다.

모델을 보려면 (SDK)

1. 아직 설치하지 않은 경우 AWS CLI 및 AWS SDKs를 설치하고 구성합니다. 자세한 내용은 4단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.
2. 다음 예제 코드를 사용하여 모델을 나열하고 모델에 대한 정보를 얻을 수 있습니다.

   CLI

   `list-models` 명령을 사용하여 프로젝트의 모델을 나열합니다.

   다음 값을 변경합니다.

   - `project-name`은 확인할 모델이 포함된 프로젝트 이름으로 변경합니다.

```
aws lookoutvision list-models --project-name project name \
  --profile lookoutvision-access
```

describe-model 명령을 사용하여 모델에 대한 정보를 가져옵니다. 다음 값을 변경합니다.

- project-name은 확인할 모델이 포함된 프로젝트 이름으로 변경합니다.
- model-version은 설명하고자 하는 모델의 버전으로 변경합니다.

```
aws lookoutvision describe-model --project-name project name\
    --model-version model version \
   --profile lookoutvision-access
```

Python

이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져옵니다. 전체 예제는 여기에서 확인하세요.

```
    @staticmethod
    def describe_models(lookoutvision_client, project_name):
        """
        Gets information about all models in a Lookout for Vision project.

        :param lookoutvision_client: A Boto3 Lookout for Vision client.
        :param project_name: The name of the project that you want to use.
        """
        try:
            response =
 lookoutvision_client.list_models(ProjectName=project_name)
            print("Project: " + project_name)
            for model in response["Models"]:
                Models.describe_model(
                    lookoutvision_client, project_name, model["ModelVersion"]
                )
                print()
            print("Done...")
        except ClientError:
            logger.exception("Couldn't list models.")
            raise
```

Java V2

이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져옵니다. 전체 예제는 여기에서 확인하세요.

```
/**
 * Lists the models in an Amazon Lookout for Vision project.
 *
 * @param lfvClient   An Amazon Lookout for Vision client.
 * @param projectName The name of the project that contains the models that
 *                    you want to list.
 * @return List <Metadata> A list of models in the project.
 */
public static List<ModelMetadata> listModels(LookoutVisionClient lfvClient,
 String projectName)
                throws LookoutVisionException {

        ListModelsRequest listModelsRequest = ListModelsRequest.builder()
                        .projectName(projectName)
                        .build();

        // Get a list of models in the supplied project.
        ListModelsResponse response = lfvClient.listModels(listModelsRequest);

        for (ModelMetadata model : response.models()) {
                logger.log(Level.INFO, "Model ARN: {0}\nVersion: {1}\nStatus:
 {2}\nMessage: {3}", new Object[] {
                                        model.modelArn(),
                                        model.modelVersion(),
                                        model.statusMessage(),
                                        model.statusAsString() });
        }

        return response.models();

}
```

모델 삭제

콘솔이나 DeleteModel 작업을 사용하여 모델 버전을 삭제할 수 있습니다. 실행 중이거나 학습 중인 모델 버전은 삭제할 수 없습니다.

모델이 실행 중인 버전인 경우 먼저 StopModel 작업을 사용하여 모델 버전을 중지하십시오. 자세한 내용은 Amazon Lookout for Vision 모델 중지하기 단원을 참조하십시오. 모델이 학습 중인 경우 모델이 완료될 때까지 기다린 후 모델을 삭제하십시오.

모델을 삭제하는 데 몇 초가 걸릴 수도 있습니다. 모델이 삭제되었는지 확인하려면 ListProjects를 호출하고 모델 버전 (ModelVersion)이 Models 배열에 있는지 확인하십시오.

모델 삭제 (콘솔)

콘솔에서 모델을 삭제하려면 다음 단계를 수행하세요.

모델을 삭제하려면(콘솔)

1. https://console.aws.amazon.com/lookoutvision/에서 Amazon Lookout for Vision 콘솔을 엽니다.
2. Get started를 선택합니다.
3. 왼쪽 탐색 창에서 프로젝트를 선택합니다.
4. 프로젝트 페이지에서 삭제하려는 모델이 포함된 프로젝트를 선택합니다.
5. 왼쪽 탐색 창에서 모델을 선택합니다.
6. 모델 보기에서 삭제할 모델의 라디오 버튼을 선택합니다.
7. 페이지 상단에서 삭제를 선택합니다.
8. 삭제 대화 상자에 삭제를 입력하여 모델 삭제를 확인합니다.
9. 모델 삭제를 선택하여 모델을 삭제합니다.

모델 삭제 (SDK)

DeleteModel 작업을 수행한 모델을 삭제하려면 다음 절차를 따르세요.

모델 (SDK) 을 삭제하려면

1. 아직 설치하지 않은 경우 AWS CLI 및 AWS SDKs를 설치하고 구성합니다. 자세한 내용은 4단계: AWS CLI 및 AWS SDKs 설정 단원을 참조하십시오.
2. 모델을 삭제하려면 다음 예제 코드를 사용합니다.

CLI

다음 값을 변경합니다.

- `project-name`은 삭제할 모델이 포함된 프로젝트 이름으로 변경합니다.
- `model-version`을 삭제할 모델의 버전으로 변경합니다.

```
aws lookoutvision delete-model --project-name project name\
  --model-version model version \
  --profile lookoutvision-access
```

Python

이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져옵니다. 전체 예제는 여기에서 확인하세요.

```
    @staticmethod
    def delete_model(lookoutvision_client, project_name, model_version):
        """
        Deletes a Lookout for Vision model. The model must first be stopped and
can't
        be in training.

        :param lookoutvision_client: A Boto3 Lookout for Vision client.
        :param project_name: The name of the project that contains the desired
model.
        :param model_version: The version of the model that you want to delete.
        """
        try:
            logger.info("Deleting model: %s", model_version)
            lookoutvision_client.delete_model(
                ProjectName=project_name, ModelVersion=model_version
            )

            model_exists = True
            while model_exists:
                response =
lookoutvision_client.list_models(ProjectName=project_name)

                model_exists = False
```

```
                for model in response["Models"]:
                    if model["ModelVersion"] == model_version:
                        model_exists = True

                if model_exists is False:
                    logger.info("Model deleted")
                else:
                    logger.info("Model is being deleted...")
                    time.sleep(2)

            logger.info("Deleted Model: %s", model_version)
        except ClientError:
            logger.exception("Couldn't delete model.")
            raise
```

Java V2

이 코드는 AWS 설명서 SDK 예제 GitHub 리포지토리에서 가져옵니다. 전체 예제는 여기에서 확인하세요.

```
/**
* Deletes an Amazon Lookout for Vision model.
*
* @param lfvClient    An Amazon Lookout for Vision client. Returns after the
 model is deleted.
* @param projectName  The name of the project that contains the model that you
 want to delete.
* @param modelVersion The version of the model that you want to delete.
* @return void
*/
public static void deleteModel(LookoutVisionClient lfvClient,
                String projectName,
                String modelVersion) throws LookoutVisionException,
 InterruptedException {

        DeleteModelRequest deleteModelRequest = DeleteModelRequest.builder()
                        .projectName(projectName)
                        .modelVersion(modelVersion)
                        .build();

        lfvClient.deleteModel(deleteModelRequest);
```

```
        boolean deleted = false;

        do {

                ListModelsRequest listModelsRequest =
 ListModelsRequest.builder()
                                .projectName(projectName)
                                .build();

                // Get a list of models in the supplied project.
                ListModelsResponse response =
 lfvClient.listModels(listModelsRequest);

                ModelMetadata modelMetadata = response.models().stream()
                                .filter(model ->
 model.modelVersion().equals(modelVersion)).findFirst()
                                .orElse(null);

                if (modelMetadata == null) {
                        deleted = true;
                        logger.log(Level.INFO, "Deleted: Model version {0} of
 project {1}.",
                                        new Object[] { modelVersion,
 projectName });

                } else {
                        logger.log(Level.INFO, "Not yet deleted: Model version
 {0} of project {1}.",
                                        new Object[] { modelVersion,
 projectName });
                        TimeUnit.SECONDS.sleep(60);
                }

        } while (!deleted);

}
```

모델 태깅

태그를 사용하여 Amazon Lookout for Vision 모델을 식별, 구성, 검색 및 필터링할 수 있습니다. 각 태그는 사용자 정의 키와 값으로 구성된 레이블입니다. 예를 들어 모델에 대한 청구 여부를 결정하는 데

도움이 되도록 모델에 `Cost center` 키를 지정하고 적절한 비용 센터 번호를 값으로 추가할 수 있습니다. 자세한 내용은 AWS 리소스에 태그 지정 단원을 참조하세요.

태그를 사용하여 다음을 수행합니다.

- 비용 할당 태그를 사용하여 모델의 청구를 추적하세요. 자세한 내용은 비용 할당 태그 사용을 참조하세요.
- Identity and Access Management (IAM)를 사용하여 모델에 대한 액세스를 제어합니다. 리소스 태그에 대한 자세한 내용은 리소스 태그를 사용한 AWS 리소스 액세스 제어를 참조하세요.
- 모델 관리를 자동화합니다. 예를 들어, 비용을 절감하기 위해 업무 외 시간에 개발 모델의 가동을 중단하는 자동화된 시작 또는 중지 스크립트를 실행할 수 있습니다. 자세한 내용은 훈련된 Amazon Lookout for Vision 모델 실행 단원을 참조하십시오.

Amazon Lookout for Vision 콘솔을 사용하거나 AWS SDKs.

주제

모델 태그 지정 (콘솔)

Amazon Lookout for Vision 콘솔을 사용하여 모델에 태그를 추가하고, 모델에 첨부된 태그를 확인하고, 태그를 제거할 수 있습니다.

태그 추가 또는 제거 (콘솔)

이 절차는 기존 모델에 태그를 추가하거나 기존 모델에서 태그를 제거하는 방법을 설명합니다. 새 모델을 학습할 때 새 모델에 태그를 추가할 수도 있습니다. 자세한 내용은 모델 학습 단원을 참조하십시오.

기존 모델에 태그를 추가하거나 기존 모델에서 태그를 제거하려면 (콘솔)

1. https://console.aws.amazon.com/lookoutvision/에서 Amazon Lookout for Vision 콘솔을 엽니다.
2. Get started를 선택합니다.
3. 탐색 창에서 프로젝트를 선택합니다.
4. 프로젝트 리소스 페이지에서 태그를 지정할 모델이 들어 있는 프로젝트를 선택합니다.
5. 탐색 창의 이전에 선택한 프로젝트에서 모델을 선택합니다.

6. 모델 섹션에서 태그를 추가할 모델을 선택합니다.
7. 모델의 세부 정보 페이지에서 태그 탭을 선택합니다.
8. 태그 섹션에서 태그 관리를 선택합니다.
9. 태그 관리 페이지에서 새 태그 추가를 선택하세요.
10. 키와 값을 입력합니다.
    a. 키에 키 이름을 입력합니다.
    b. 값에 값을 입력합니다.
11. 태그를 더 추가하려면 9 및 10단계를 반복합니다.
12. (선택 사항) 태그를 제거하려면 제거하려는 태그 옆에 있는 제거를 선택합니다. 이전에 저장한 태그를 제거하는 경우 변경 내용을 저장하면 해당 태그가 제거됩니다.
13. 변경 사항을 저장하려면 변경 사항 저장을 선택합니다.

모델 태그 보기 (콘솔)

Amazon Lookout for Vision 콘솔을 사용하여 모델에 첨부된 태그를 볼 수 있습니다.

프로젝트 내 모든 모델에 첨부된 태그를 보려면 AWS SDK를 사용해야 합니다. 자세한 내용은 모델 태그 나열 (SDK) 단원을 참조하십시오.

모델에 연결된 태그를 보려면

1. https://console.aws.amazon.com/lookoutvision/에서 Amazon Lookout for Vision 콘솔을 엽니다.
2. Get started를 선택합니다.
3. 탐색 창에서 프로젝트를 선택합니다.
4. 프로젝트 리소스 페이지에서 보려는 태그의 모델이 포함된 프로젝트를 선택합니다.
5. 탐색 창의 이전에 선택한 프로젝트에서 모델을 선택합니다.
6. 모델 섹션에서 태그를 보려는 모델을 선택합니다.
7. 모델의 세부 정보 페이지에서 태그 탭을 선택합니다. 태그는 태그 섹션에 표시됩니다.

모델 태그 지정(SDK)

AWS SDK를 사용하여 다음을 수행할 수 있습니다.

- 새 모델에 태그 추가

- 기존 모델에 태그 추가
- 모델에 지정된 태그 나열
- 모델에서 태그 제거

이 섹션에는 AWS CLI 예제가 포함되어 있습니다. 를 설치하지 않은 경우 섹션을 AWS CLI참조하세요4단계: AWS CLI 및 AWS SDKs 설정.

새 모델에 태그 추가 (SDK)

CreateModel 작업을 사용하여 모델을 생성할 때 태그를 추가할 수 있습니다. Tags 배열 입력 파라미터에 하나 이상의 태그를 지정합니다.

```
aws lookoutvision create-model --project-name "project name"\
  --output-config '{ "S3Location": { "Bucket": "output bucket", "Prefix":  "output
 folder" } }'\
  --tags '[{"Key":"Key","Value":"Value"}]' \
  --profile lookoutvision-access
```

모델 생성 및 훈련에 대한 자세한 정보는 모델 학습 (SDK) 항목을 참조하세요.

기존 모델에 태그 추가 (SDK)

기존 모델에 하나 이상의 태그를 추가하려면 TagResource 작업을 사용합니다. 모델의 Amazon 리소스 이름(ARN)(ResourceArn)과 추가할 태그(Tags)를 지정합니다.

```
aws lookoutvision tag-resource --resource-arn "resource-arn"\
  --tags '[{"Key":"Key","Value":"Value"}]' \
  --profile lookoutvision-access
```

자바 코드의 예는 TagModel을 참조하십시오.

모델 태그 나열 (SDK)

모델에 연결된 태그를 나열하려면 ListTagsForResource 작업을 사용하고 모델의 Amazon 리소스 이름(ARN)인 (ResourceArn)을 지정합니다. 응답은 지정된 모델에 연결된 태그 키 및 값의 맵입니다.

```
aws lookoutvision list-tags-for-resource --resource-arn resource-arn \
  --profile lookoutvision-access
```

프로젝트에서 특정 태그가 있는 모델을 확인하려면 ListModels 항목을 호출하여 모델 목록을 가져오세요. 그런 다음 ListModels 응답에서 각 모델에 ListTagsForResource를 직접 호출하세요. ListTagsForResource의 응답을 검사하여 필요 태그가 있는지 확인하세요.

자바 코드의 예는 ListModelTags를 참조하십시오. 모든 프로젝트에서 태그 값을 검색하는 Python 코드의 예는 find_tag.py를 참조하십시오.

모델에서 태그 제거 (SDK)

모델에서 하나 이상의 태그를 제거하려면 UntagResource 작업을 사용합니다. 제거할 모델의 Amazon 리소스 이름 (ARN) (ResourceArn)과 태그 키 (Tag-Keys)를 지정합니다.

```
aws lookoutvision untag-resource --resource-arn resource-arn\
  --tag-keys '["Key"]' \
  --profile lookoutvision-access
```

자바 코드의 예는 UntagModel을 참조하십시오.

평가판 확인 작업 보기

콘솔을 사용하여 평가판 확인을 볼 수 있습니다. AWS SDK를 사용하여 평가판 감지 작업을 볼 수 없습니다.

> **Note**
>
> 평가판 확인 목록은 결과의 일관성이 있습니다. 평가판 확인을 생성하는 경우 평가판 확인 목록이 최신 상태가 되기까지 잠시 기다려야 할 수 있습니다.

평가판 확인 작업 보기 (콘솔)

평가판 확인을 보려면 다음 절차를 따르세요.

평가판 확인 작업을 보려면

1. https://console.aws.amazon.com/lookoutvision/에서 Amazon Lookout for Vision 콘솔을 엽니다.
2. Get started를 선택합니다.
3. 왼쪽 탐색 창에서 평가판 확인을 선택합니다.

4. 평가판 확인 페이지에서 평가판 확인 작업을 선택하여 세부 정보를 확인합니다.

예제 코드 및 데이터세트

다음은 Amazon Lookout for Vision과 함께 사용할 수 있는 코드 예제 및 데이터 세트입니다.

주제

- 예제 코드
- 예제 데이터 세트

예제 코드

Amazon Lookout for Vision에 대한 다음 코드 예제를 사용할 수 있습니다.

예제	설명
GitHub	Amazon Lookout for Vision 모델을 학습시키고 호스팅하는 Python 코드 예시.
Amazon Lookout for Vision 랩	회로판 예제 이미지로 모델을 만드는 데 사용할 수 있는 Python Notebook.
Python 코드 예제	Amazon Lookout for Vision 문서에서 사용되는 Python 예제
Java 코드 예제	Amazon Lookout for Vision 문서에서 사용되는 Java 예제

예제 데이터 세트

다음은 Amazon Lookout for Vision과 함께 사용할 수 있는 예제 데이터 세트입니다.

주제

- 이미지 분할 데이터 세트
- 이미지 분류 데이터 세트

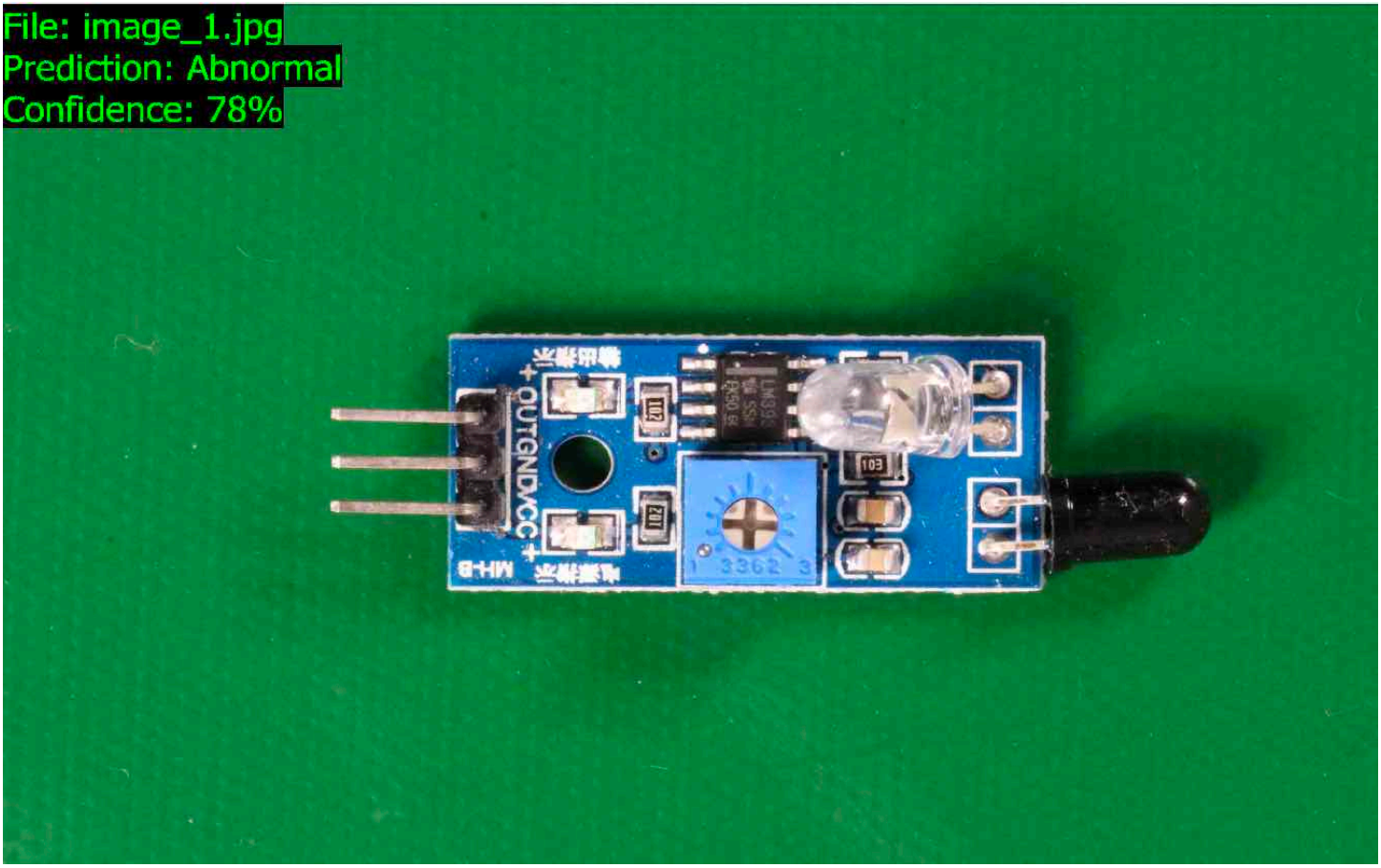
이미지 분할 데이터 세트

Amazon Lookout for Vision을 시작하기이미지 분할 모델을 만드는 데 사용할 수 있는 깨진 쿠키 데이터 세트를 제공합니다.

이미지 분할 모델을 생성하는 다른 데이터 세트에 대해서는 GPU를 사용하지 않고 엣지에서 Amazon Lookout for Vision을 사용하여 이상 현상의 위치를 식별하기를 참조하십시오.

이미지 분류 데이터 세트

Amazon Lookout for Vision은 이미지 분류 모델을 생성하는 데 사용할 수 있는 회로판의 예제 이미지를 제공합니다.



https://github.com/aws-samples/amazon-lookout-for-vision GitHub 리포지토리에서 이미지를 복사할 수 있습니다. 이미지는 `circuitboard` 폴더에 있습니다.

`circuitboard` 폴더에는 다음과 같은 폴더가 있습니다.

- `train`— 교육 데이터 세트에서 사용할 수 있는 이미지.

- `test`— 테스트 데이터 세트에 사용할 수 있는 이미지.
- `extra_images`— 시험 탐지를 실행하거나 DetectAnomalies 연산을 통해 학습된 모델을 시험해 보는 데 사용할 수 있는 이미지.

`train` 및 `test` 폴더에는 각각 `normal`라는 하위 폴더 (정상 이미지 포함)와 `anomaly`라는 하위 폴더 (이상이 있는 이미지 포함)가 있습니다.

> **Note**
>
> 나중에 콘솔로 데이터 세트를 만들면 Amazon Lookout for Vision에서 폴더 이름 (`normal` 및 `anomaly`)을 사용하여 이미지에 자동으로 레이블을 지정할 수 있습니다. 자세한 내용은 the section called “Amazon S3 버킷” 단원을 참조하십시오.

데이터 세트 이미지를 준비하려면

1. https://github.com/aws-samples/amazon-lookout-for-vision 리포지토리를 컴퓨터에 복제합니다. 자세한 내용은 리포지토리 복제를 참조하십시오.
2. Amazon S3 버킷을 생성합니다. 자세한 내용은 S3 버킷 정책을 생성하려면 어떻게 해야 합니까? 단원을 참조하십시오.
3. 명령 프롬프트에 다음 명령을 입력하여 컴퓨터의 데이터 세트 이미지를 Amazon S3 버킷으로 복사합니다.

```
aws s3 cp --recursive your-repository-folder/circuitboard s3://your-bucket/
circuitboard
```

이미지를 업로드한 후 모델을 생성할 수 있습니다. 이전에 회로 기판 이미지를 업로드한 Amazon S3 위치의 이미지를 추가하여 이미지를 자동으로 분류할 수 있습니다. 모델을 성공적으로 학습할 때마다 그리고 모델이 실행 (호스팅)된 시간에 대해 요금이 부과된다는 점을 기억하십시오.

분류 모델을 생성하는 방법

1. Do 프로젝트 생성(콘솔).
2. Do Amazon S3 버킷에 저장된 이미지를 사용하여 데이터 세트를 생성합니다..
   - 6단계에서는 훈련 및 테스트 데이터 세트 분리 탭을 선택합니다.

- 8a단계의 경우 데이터세트 이미지 준비하기에서 업로드한 교육 이미지의 S3 URI를 입력합니다. 예: `s3://your-bucket/circuitboard/train`. 8b단계에서 테스트 데이터 세트에 대한 S3 URI를 입력합니다. 예: `s3://your-bucket/circuitboard/test`.
- 9단계를 반드시 수행하십시오.

3. Do 모델 교육 (콘솔).
4. Do 모델 시작 (콘솔).
5. Do 이미지에서 이상 탐지. `test_images` 폴더의 이미지를 사용할 수 있습니다.
6. 모델을 완성하면 모델 중지하기 (콘솔) 작업을 완료하십시오.

Amazon Lookout for Vision의 보안

의 클라우드 보안 AWS 이 최우선 순위입니다. AWS 고객은 보안에 가장 민감한 조직의 요구 사항을 충족하도록 구축된 데이터 센터 및 네트워크 아키텍처의 이점을 누릴 수 있습니다.

보안은 AWS 와 사용자 간의 공동 책임입니다. 공동 책임 모델은 이 사항을 클라우드의 보안 및 클라우드 내 보안으로 설명합니다.

- 클라우드 보안 - AWS 는 AWS 클라우드에서 AWS 서비스를 실행하는 인프라를 보호할 책임이 있습니다. AWS 또한는 안전하게 사용할 수 있는 서비스를 제공합니다. 타사 감사자는 AWS 규정 준수 프로그램 일환으로 보안의 효과를 정기적으로 테스트하고 확인합니다. Amazon Lookout for Vision에 적용되는 규정 준수 프로그램에 대한 자세한 내용은 규정 준수 프로그램의 범위에 속하는 AWS 서비스 를 참조하세요.
- 클라우드의 보안 - 사용자의 책임은 사용하는 AWS 서비스에 따라 결정됩니다. 또한 귀하는 귀사의 데이터 민감도, 귀사의 요구 사항, 관련 법률 및 규정을 비롯한 기타 요소에 대해서도 책임이 있습니다.

이 설명서는 Lookout for Vision 사용 시 책임 분담 모델을 적용하는 방법을 이해하는 데 도움이 됩니다. 다음 주제에서는 보안 및 규정 준수 목표를 충족하도록 Lookout for Vision를 구성하는 방법을 보여줍니다. 또한 Lookout for Vision 리소스를 모니터링하고 보호하는 데 도움이 되는 다른 AWS 서비스를 사용하는 방법도 알아봅니다.

주제

Amazon Lookout for Vision의 데이터 보호

AWS 공동 책임 모델 Amazon Lookout for Vision의 데이터 보호에 적용됩니다. 이 모델에 설명된 대로 AWS 는 모든를 실행하는 글로벌 인프라를 보호할 책임이 있습니다 AWS 클라우드. 사용자는 인프라에서 호스팅되는 콘텐츠를 관리해야 합니다. 사용하는 AWS 서비스 의 보안 구성과 관리 태스크에 대한 책임도 사용자에게 있습니다. 데이터 프라이버시에 대한 자세한 내용은 데이터 프라이버시 FAQ를

참조하세요. 유럽의 데이터 보호에 대한 자세한 내용은 AWS 보안 블로그의 AWS 공동 책임 모델 및 GDPR 블로그 게시물을 참조하세요.

데이터 보호를 위해 자격 증명을 보호하고 AWS 계정 AWS IAM Identity Center 또는 AWS Identity and Access Management (IAM)를 사용하여 개별 사용자를 설정하는 것이 좋습니다. 이렇게 하면 개별 사용자에게 자신의 직무를 충실히 이행하는 데 필요한 권한만 부여됩니다. 또한 다음과 같은 방법으로 데이터를 보호하는 것이 좋습니다.

- 각 계정에 다중 인증(MFA)을 사용하세요.
- SSL/TLS를 사용하여 AWS 리소스와 통신합니다. TLS 1.2는 필수이며 TLS 1.3을 권장합니다.
- 를 사용하여 API 및 사용자 활동 로깅을 설정합니다 AWS CloudTrail. CloudTrail 추적을 사용하여 AWS 활동을 캡처하는 방법에 대한 자세한 내용은 AWS CloudTrail 사용 설명서의 CloudTrail 추적 작업을 참조하세요.
- AWS 암호화 솔루션과 내부의 모든 기본 보안 제어를 사용합니다 AWS 서비스.
- Amazon S3에 저장된 민감한 데이터를 검색하고 보호하는 데 도움이 되는 Amazon Macie와 같은 고급 관리형 보안 서비스를 사용하세요.
- 명령줄 인터페이스 또는 API를 AWS 통해에 액세스할 때 FIPS 140-3 검증 암호화 모듈이 필요한 경우 FIPS 엔드포인트를 사용합니다. 사용 가능한 FIPS 엔드포인트에 대한 자세한 내용은 Federal Information Processing Standard(FIPS) 140-3을 참조하세요.

고객의 이메일 주소와 같은 기밀 정보나 중요한 정보는 태그나 이름 필드와 같은 자유 형식 텍스트 필드에 입력하지 않는 것이 좋습니다. 여기에는 Lookout for Vision 또는 기타 AWS 서비스 에서 콘솔, API AWS CLI또는 AWS SDKs를 사용하여 작업하는 경우가 포함됩니다. 이름에 사용되는 태그 또는 자유 형식 텍스트 필드에 입력하는 모든 데이터는 청구 또는 진단 로그에 사용될 수 있습니다. 외부 서버에 URL을 제공할 때 해당 서버에 대한 요청을 검증하기 위해 자격 증명을 URL에 포함해서는 안 됩니다.

데이터 암호화

다음 정보는 Lookout for Vision에서 데이터 암호화를 사용하여 데이터를 보호하는 방법을 설명합니다.

저장 시 암호화

이미지

모델을 학습시키기 위해 Amazon Lookout for Vision은 소스 교육 및 테스트 이미지의 사본을 만듭니다. 복사된 이미지는 사용자가 제공하는 AWS 소유 키 또는 키를 사용한 서버 측 암호화를 사용하

여 Amazon Simple Storage Service(S3)에서 유휴 시 암호화됩니다. 키는 AWS 키 관리 서비스(SSE-KMS)를 통해 저장됩니다. 소스 이미지는 영향을 받지 않습니다. 자세한 내용은 모델 학습 단원을 참조하십시오.

Amazon Lookout for Vision 모델

기본적으로 학습된 모델 및 매니페스트 파일은 AWS AWS Key Management Service(SSE-KMS)에 저장된 KMS 키를 사용한 서버 측 암호화를 통해 Amazon S3에서 암호화됩니다. Lookout for Vision은 AWS 소유 키를 사용합니다. 자세한 내용은 서버 측 암호화를 사용하여 데이터 보호를 참조하십시오. 훈련 결과는 `CreateModel`에 대한 `output_bucket` 입력 파라미터에 지정된 버킷에 기록됩니다. 훈련 결과는 버킷에 대해 구성된 암호화 설정을 사용하여 암호화됩니다 (`output_bucket`).

Amazon Lookout for Vision 콘솔 버킷

Amazon Lookout for Vision 콘솔은 프로젝트를 관리하는 데 사용할 수 있는 Amazon S3 버킷 (콘솔 버킷)을 생성합니다. 콘솔 버킷은 기본 Amazon S3 암호화를 사용하여 암호화됩니다. 자세한 내용을 알아보려면 S3 버킷에 대한 Amazon Simple Storage Service 기본 암호화를 참조하세요. 자체 KMS 키를 사용하는 경우 콘솔 버킷을 생성한 후에 구성하십시오. 자세한 내용은 서버 측 암호화를 사용하여 데이터 보호를 참조하십시오. Amazon Lookout for Vision은 콘솔 버킷에 대한 퍼블릭 액세스를 차단합니다.

전송 중 암호화

Amazon Lookout for Vision API 엔드포인트는 HTTPS를 통한 보안 연결만을 지원합니다. 모든 통신은 TLS(전송 계층 보안)를 통해 암호화됩니다.

키 관리

AWS KMS(키 관리 서비스)를 사용하여 Amazon S3 버킷에 저장하는 입력 이미지의 암호화를 관리할 수 있습니다. 자세한 내용은 5단계: (선택 사항) 고유한 AWS Key Management Service 키 사용 단원을 참조하십시오.

기본적으로 이미지는 AWS가 소유하고 관리하는 키로 암호화됩니다. 자체 AWS Key Management Service (KMS) 키를 사용하도록 선택할 수도 있습니다. 자세한 내용은 AWS Key Management Service 개념을 참조하십시오.

인터네트워크 트래픽 개인 정보

Amazon Lookout for Vision용 Amazon Virtual Private Cloud(Amazon VPC) 엔드포인트는 VPC 내의 논리적 엔터티로서, Amazon Lookout for Vision에만 연결을 허용합니다. Amazon VPC는 Amazon Lookout for Vision로 요청을 라우팅하고, 응답을 다시 VPC로 라우팅합니다. 자세한 내용은 Amazon

VPC 사용 설명서의 VPC 엔드포인트를 참조하세요. Amazon Lookout for Vision과 함께 Amazon VPC 엔드포인트를 사용하는 방법에 대한 자세한 내용은 인터페이스 엔드포인트 (AWS PrivateLink)를 사용하여 Amazon Lookout for Vision에 액세스을 참조하십시오.

Amazon Lookout for Vision용 ID 및 액세스 관리

AWS Identity and Access Management (IAM)는 관리자가 AWS 리소스에 대한 액세스를 안전하게 제어할 수 AWS 서비스 있도록 도와주는 입니다. IAM 관리자는 CloudWatch 리소스를 사용하도록 인증(로그인) 및 권한(권한 있음)을 받을 수 있는 사용자를 제어합니다. IAM은 추가 비용 없이 사용할 수 AWS 서비스 있는 입니다.

주제

- 대상
- ID를 통한 인증
- 정책을 사용하여 액세스 관리
- Amazon Lookout for Vision이 IAM과 작동하는 방식
- Amazon Lookout for Vision에 대한 자격 증명 기반 정책 예시
- AWS Amazon Lookout for Vision에 대한 관리형 정책
- Amazon Lookout for Vision ID 및 액세스 문제 해결

대상

사용 방법 AWS Identity and Access Management (IAM)은 Lookout for Vision에서 수행하는 작업에 따라 다릅니다.

서비스 사용자 - Lookout for Vision 서비스를 사용하여 작업을 수행하는 경우 필요한 자격 증명과 권한을 관리자가 제공합니다. 더 많은 Lookout for Vision 기능을 사용하여 작업을 수행하게 되면 추가 권한이 필요할 수 있습니다. 액세스 권한 관리 방법을 이해하면 관리자에게 올바른 권한을 요청하는 데 도움이 됩니다. Lookout for Vision의 기능에 액세스할 수 없는 경우 Amazon Lookout for Vision ID 및 액세스 문제 해결 단원을 참조하세요.

서비스 관리자 - 회사에서 Lookout for Vision 리소스를 책임지고 있는 경우 Lookout for Vision에 대한 전체 액세스 권한을 가지고 있을 것입니다. 서비스 관리자는 서비스 사용자가 액세스해야 하는 Lookout for Vision 기능과 리소스를 결정합니다. 그런 다음 IAM 관리자에게 요청을 제출하여 서비스 사용자의 권한을 변경해야 합니다. 이 페이지의 정보를 검토하여 IAM의 기본 개념을 이해하세요. 회사

가 Lookout for Vision에서 IAM을 사용하는 방법에 대해 자세히 알아보려면 Amazon Lookout for Vision이 IAM과 작동하는 방식 단원을 참조하세요.

IAM 관리자 - IAM 관리자라면 Lookout for Vision에 대한 액세스 권한 관리 정책 작성 방법을 자세히 알고 싶을 것입니다. IAM에서 사용할 수 있는 Lookout for Vision 자격 증명 기반 정책 예제를 보려면 Amazon Lookout for Vision에 대한 자격 증명 기반 정책 예시 단원을 참조하세요.

ID를 통한 인증

인증은 AWS 자격 증명으로에 로그인하는 방법입니다. IAM 사용자 또는 AWS 계정 루트 사용자 IAM 역할을 수임하여 로 인증(로그인 AWS)되어야 합니다.

자격 증명 소스를 통해 제공된 자격 증명을 사용하여 페더레이션 자격 증명 AWS 으로에 로그인할 수 있습니다. AWS IAM Identity Center (IAM Identity Center) 사용자, 회사의 Single Sign-On 인증 및 Google 또는 Facebook 자격 증명은 페더레이션 자격 증명의 예입니다. 페더레이션형 ID로 로그인할 때 관리자가 이전에 IAM 역할을 사용하여 ID 페더레이션을 설정했습니다. 페더레이션을 사용하여 AWS 에 액세스하면 간접적으로 역할을 수임하게 됩니다.

사용자 유형에 따라 AWS Management Console 또는 AWS 액세스 포털에 로그인할 수 있습니다. 에 로그인하는 방법에 대한 자세한 내용은 AWS 로그인 사용 설명서의에 로그인하는 방법을 AWS참조하세요. AWS 계정

AWS 프로그래밍 방식으로에 액세스하는 경우는 자격 증명을 사용하여 요청에 암호화 방식으로 서명할 수 있는 소프트웨어 개발 키트(SDK)와 명령줄 인터페이스(CLI)를 AWS 제공합니다. AWS 도구를 사용하지 않는 경우 직접 요청에 서명해야 합니다. 권장 방법을 사용하여 요청에 직접 서명하는 자세한 방법은 IAM 사용 설명서에서 API 요청용AWS Signature Version 4를 참조하세요.

사용하는 인증 방법에 상관없이 추가 보안 정보를 제공해야 할 수도 있습니다. 예를 들어는 다중 인증(MFA)을 사용하여 계정의 보안을 강화할 것을 AWS 권장합니다. 자세한 내용은 AWS IAM Identity Center 사용 설명서에서 다중 인증 및 IAM 사용 설명서에서 IAM의AWS 다중 인증을 참조하세요.

AWS 계정 루트 사용자

를 생성할 때 계정의 모든 AWS 서비스 및 리소스에 대한 완전한 액세스 권한이 있는 하나의 로그인 자격 증명으로 AWS 계정시작합니다. 이 자격 증명을 AWS 계정 테루트 사용자라고 하며 계정을 생성하는 데 사용한 이메일 주소와 암호로 로그인하여 액세스합니다. 일상적인 작업에 루트 사용자를 사용하지 않을 것을 강력히 권장합니다. 루트 사용자 자격 증명을 보호하고 루트 사용자만 수행할 수 있는 작업을 수행하는 데 사용합니다. 루트 사용자로 로그인해야 하는 전체 작업 목록은 IAM 사용 설명서의 루트 사용자 보안 인증이 필요한 작업을 참조하세요.

페더레이션 자격 증명

가장 좋은 방법은 관리자 액세스가 필요한 사용자를 포함한 인간 사용자에게 자격 증명 공급자와의 페더레이션을 사용하여 임시 자격 증명을 사용하여 AWS 서비스 에 액세스하도록 요구하는 것입니다.

페더레이션 자격 증명은 엔터프라이즈 사용자 디렉터리, 웹 자격 증명 공급자, AWS Directory Service, Identity Center 디렉터리 또는 자격 증명 소스를 통해 제공된 자격 증명을 사용하여 AWS 서비스 에 액세스하는 모든 사용자의 사용자입니다. 페더레이션 자격 증명에 액세스할 때 역할을 AWS 계정수임하고 역할은 임시 자격 증명을 제공합니다.

중앙 집중식 액세스 관리를 위해 AWS IAM Identity Center을(를) 사용하는 것이 좋습니다. IAM Identity Center에서 사용자 및 그룹을 생성하거나 모든 및 애플리케이션에서 사용할 수 있도록 자체 자격 증명 소스의 사용자 AWS 계정 및 그룹 집합에 연결하고 동기화할 수 있습니다. IAM Identity Center에 대한 자세한 내용은 AWS IAM Identity Center 사용 설명서에서 IAM Identity Center란 무엇인가요?를 참조하세요.

IAM 사용자 및 그룹

IAM 사용자는 단일 사용자 또는 애플리케이션에 대한 특정 권한이 AWS 계정 있는 내 자격 증명입니다. 가능하면 암호 및 액세스 키와 같은 장기 자격 증명이 있는 IAM 사용자를 생성하는 대신 임시 자격 증명을 사용하는 것이 좋습니다. 하지만 IAM 사용자의 장기 자격 증명이 필요한 특정 사용 사례가 있는 경우, 액세스 키를 교체하는 것이 좋습니다. 자세한 내용은 IAM 사용 설명서의 장기 보안 인증이 필요한 사용 사례의 경우, 정기적으로 액세스 키 교체를 참조하세요.

IAM 그룹은 IAM 사용자 컬렉션을 지정하는 자격 증명입니다. 사용자는 그룹으로 로그인할 수 없습니다. 그룹을 사용하여 여러 사용자의 권한을 한 번에 지정할 수 있습니다. 그룹을 사용하면 대규모 사용자 집합의 권한을 더 쉽게 관리할 수 있습니다. 예를 들어, IAMAdmins라는 그룹이 있고 이 그룹에 IAM 리소스를 관리할 권한을 부여할 수 있습니다.

사용자는 역할과 다릅니다. 사용자는 한 사람 또는 애플리케이션과 고유하게 연결되지만, 역할은 해당 역할이 필요한 사람이라면 누구나 수임할 수 있습니다. 사용자는 영구적인 장기 자격 증명을 가지고 있지만, 역할은 임시 보안 인증만 제공합니다. 자세한 내용은 IAM 사용 설명서에서 IAM 사용자 사용 사례를 참조하세요.

IAM 역할

IAM 역할은 특정 권한이 AWS 계정 있는 내 자격 증명입니다. IAM 사용자와 유사하지만, 특정 개인과 연결되지 않습니다. 에서 IAM 역할을 일시적으로 수임하려면 사용자에서 IAM 역할(콘솔)로 전환할 AWS Management Console수 있습니다. https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_use_switch-role-console.html 또는 AWS API 작업을 호출하거나 사용자 지정 URL을 AWS

CLI 사용하여 역할을 수임할 수 있습니다. 역할 사용 방법에 대한 자세한 내용은 IAM 사용 설명서의 역할 수임 방법을 참조하세요.

임시 보안 인증이 있는 IAM 역할은 다음과 같은 상황에서 유용합니다.

- 페더레이션 사용자 액세스 - 페더레이션 ID에 권한을 부여하려면 역할을 생성하고 해당 역할의 권한을 정의합니다. 페더레이션 ID가 인증되면 역할이 연결되고 역할에 정의된 권한이 부여됩니다. 페더레이션 관련 역할에 대한 자세한 내용은 IAM 사용 설명서의 Create a role for a third-party identity provider (federation)를 참조하세요. IAM Identity Center를 사용하는 경우, 권한 집합을 구성합니다. 인증 후 ID가 액세스할 수 있는 항목을 제어하기 위해 IAM Identity Center는 권한 집합을 IAM의 역할과 연관짓습니다. 권한 집합에 대한 자세한 내용은 AWS IAM Identity Center 사용 설명서의 권한 집합을 참조하세요.
- 임시 IAM 사용자 권한 - IAM 사용자 또는 역할은 IAM 역할을 수임하여 특정 작업에 대한 다양한 권한을 임시로 받을 수 있습니다.
- 교차 계정 액세스 - IAM 역할을 사용하여 다른 계정의 사용자(신뢰할 수 있는 보안 주체)가 내 계정의 리소스에 액세스하도록 허용할 수 있습니다. 역할은 계정 간 액세스를 부여하는 기본적인 방법입니다. 그러나 일부 에서는 (역할을 프록시로 사용하는 대신) 정책을 리소스에 직접 연결할 AWS 서비스 수 있습니다. 교차 계정 액세스에 대한 역할과 리소스 기반 정책의 차이점을 알아보려면 IAM 사용 설명서의 IAM의 교차 계정 리소스 액세스를 참조하세요.
- 교차 서비스 액세스 - 일부는 다른에서 기능을 AWS 서비스 사용합니다 AWS 서비스. 예를 들어, 서비스에서 호출하면 일반적으로 해당 서비스는 Amazon EC2에서 애플리케이션을 실행하거나 Amazon S3에 객체를 저장합니다. 서비스는 직접적으로 호출하는 위탁자의 권한을 사용하거나, 서비스 역할을 사용하거나, 또는 서비스 연결 역할을 사용하여 이 작업을 수행할 수 있습니다.
  - 전달 액세스 세션(FAS) - IAM 사용자 또는 역할을 사용하여에서 작업을 수행하는 경우 AWS보안 주체로 간주됩니다. 일부 서비스를 사용하는 경우, 다른 서비스에서 다른 작업을 시작하는 작업을 수행할 수 있습니다. FAS는를 호출하는 보안 주체의 권한을 다운스트림 서비스에 AWS 서비스 대한 요청과 AWS 서비스함께 사용합니다. FAS 요청은 서비스가 다른 AWS 서비스 또는 리소스와의 상호 작용을 완료해야 하는 요청을 수신할 때만 수행됩니다. 이 경우, 두 작업을 모두 수행할 수 있는 권한이 있어야 합니다. FAS 요청 시 정책 세부 정보는 전달 액세스 세션을 참조하세요.
  - 서비스 역할 - 서비스 역할은 서비스가 사용자를 대신하여 작업을 수행하기 위해 맡는 IAM 역할입니다. IAM 관리자는 IAM 내에서 서비스 역할을 생성, 수정 및 삭제할 수 있습니다. 자세한 정보는 IAM 사용 설명서의 Create a role to delegate permissions to an AWS 서비스를 참조하세요.
  - 서비스 연결 역할 - 서비스 연결 역할은에 연결된 서비스 역할의 한 유형입니다 AWS 서비스. 서비스는 사용자를 대신하여 작업을 수행하기 위해 역할을 수임할 수 있습니다. 서비스 연결 역할은에 표시 AWS 계정 되며 서비스가 소유합니다. IAM 관리자는 서비스 링크 역할의 권한을 볼 수 있지만 편집은 할 수 없습니다.

- Amazon EC2에서 실행되는 애플리케이션 - IAM 역할을 사용하여 EC2 인스턴스에서 실행되고 AWS CLI 또는 AWS API 요청을 수행하는 애플리케이션의 임시 자격 증명을 관리할 수 있습니다. 이는 EC2 인스턴스 내에 액세스 키를 저장할 때 권장되는 방법입니다. EC2 인스턴스에 AWS 역할을 할당하고 모든 애플리케이션에서 사용할 수 있도록 하려면 인스턴스에 연결된 인스턴스 프로파일을 생성합니다. 인스턴스 프로필에는 역할이 포함되어 있으며 EC2 인스턴스에서 실행되는 프로그램이 임시 보안 인증을 얻을 수 있습니다. 자세한 정보는 IAM 사용 설명서의 IAM 역할을 사용하여 Amazon EC2 인스턴스에서 실행되는 애플리케이션에 권한 부여를 참조하세요.

정책을 사용하여 액세스 관리

정책을 AWS 생성하고 자격 증명 또는 리소스에 연결하여 AWS 에서 액세스를 제어합니다. 정책은 자격 증명 또는 리소스와 연결된 AWS 경우 권한을 정의하는의 객체입니다.는 보안 주체(사용자, 루트 사용자 또는 역할 세션)가 요청할 때 이러한 정책을 AWS 평가합니다. 정책에서 권한은 요청이 허용되거나 거부되는 지를 결정합니다. 대부분의 정책은에 JSON 문서 AWS 로 저장됩니다. JSON 정책 문서의 구조와 콘텐츠에 대한 자세한 정보는 IAM 사용 설명서의 JSON 정책 개요를 참조하세요.

관리자는 AWS JSON 정책을 사용하여 대상에 액세스할 수 있는 사용자를 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

기본적으로, 사용자 및 역할에는 어떠한 권한도 없습니다. 사용자에게 사용자가 필요한 리소스에서 작업을 수행할 권한을 부여하려면 IAM 관리자가 IAM 정책을 생성하면 됩니다. 그런 다음 관리자가 IAM 정책을 역할에 추가하고, 사용자가 역할을 수임할 수 있습니다.

IAM 정책은 작업을 수행하기 위해 사용하는 방법과 상관없이 작업에 대한 권한을 정의합니다. 예를 들어, `iam:GetRole` 작업을 허용하는 정책이 있다고 가정합니다. 해당 정책이 있는 사용자는 AWS Management Console AWS CLI, 또는 API에서 역할 정보를 가져올 수 있습니다 AWS .

ID 기반 정책

ID 기반 정책은 IAM 사용자, 사용자 그룹 또는 역할과 같은 ID에 연결할 수 있는 JSON 권한 정책 문서입니다. 이러한 정책은 사용자 및 역할이 어떤 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 제어합니다. 자격 증명 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서에서 고객 관리형 정책으로 사용자 지정 IAM 권한 정의를 참조하세요.

ID 기반 정책은 인라인 정책 또는 관리형 정책으로 한층 더 분류할 수 있습니다. 인라인 정책은 단일 사용자, 그룹 또는 역할에 직접 포함됩니다. 관리형 정책은의 여러 사용자, 그룹 및 역할에 연결할 수 있는 독립 실행형 정책입니다 AWS 계정. 관리형 정책에는 AWS 관리형 정책 및 고객 관리형 정책이 포함됩니다. 관리형 정책 또는 인라인 정책을 선택하는 방법을 알아보려면 IAM 사용 설명서의 관리형 정책 및 인라인 정책 중에서 선택을 참조하세요.

리소스 기반 정책

리소스 기반 정책은 리소스에 연결하는 JSON 정책 설명서입니다. 리소스 기반 정책의 예제는 IAM 역할 신뢰 정책과 Amazon S3 버킷 정책입니다. 리소스 기반 정책을 지원하는 서비스에서 서비스 관리자는 이러한 정책을 사용하여 특정 리소스에 대한 액세스를 통제할 수 있습니다. 정책이 연결된 리소스의 경우 정책은 지정된 위탁자가 해당 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 정의합니다. 리소스 기반 정책에서 위탁자를 지정해야 합니다. 보안 주체에는 계정, 사용자, 역할, 페더레이션 사용자 또는가 포함될 수 있습니다 AWS 서비스.

리소스 기반 정책은 해당 서비스에 있는 인라인 정책입니다. 리소스 기반 정책에서는 IAM의 AWS 관리형 정책을 사용할 수 없습니다.

액세스 제어 목록(ACL)

액세스 제어 목록(ACL)은 어떤 보안 주체(계정 멤버, 사용자 또는 역할)가 리소스에 액세스할 수 있는 권한을 가지고 있는지를 제어합니다. ACL은 JSON 정책 문서 형식을 사용하지 않지만 리소스 기반 정책과 유사합니다.

Amazon S3 AWS WAF및 Amazon VPC는 ACLs. ACL에 관한 자세한 내용은 Amazon Simple Storage Service 개발자 가이드의 액세스 제어 목록(ACL) 개요를 참조하세요.

기타 정책 타입

AWS 는 덜 일반적인 추가 정책 유형을 지원합니다. 이러한 정책 타입은 더 일반적인 정책 유형에 따라 사용자에게 부여되는 최대 권한을 설정할 수 있습니다.

- 권한 경계 – 권한 경계는 ID 기반 정책에 따라 IAM 엔터티(IAM 사용자 또는 역할)에 부여할 수 있는 최대 권한을 설정하는 고급 기능입니다. 개체에 대한 권한 경계를 설정할 수 있습니다. 그 결과로 얻는 권한은 객체의 ID 기반 정책과 그 권한 경계의 교집합입니다. `Principal` 필드에서 사용자나 역할을 지정하는 리소스 기반 정책은 권한 경계를 통해 제한되지 않습니다. 이러한 정책 중 하나에 포함된 명시적 거부는 허용을 재정의합니다. 권한 경계에 대한 자세한 정보는 IAM 사용 설명서의 IAM 엔티티에 대한 권한 경계를 참조하세요.
- 서비스 제어 정책(SCPs) - SCPs는의 조직 또는 조직 단위(OU)에 대한 최대 권한을 지정하는 JSON 정책입니다 AWS Organizations. AWS Organizations 는 비즈니스가 소유 AWS 계정 한 여러를 그룹화하고 중앙에서 관리하기 위한 서비스입니다. 조직에서 모든 기능을 활성화할 경우, 서비스 제어 정책(SCP)을 임의의 또는 모든 계정에 적용할 수 있습니다. SCP는 각각을 포함하여 멤버 계정의 엔터티에 대한 권한을 제한합니다 AWS 계정 루트 사용자. 조직 및 SCP에 대한 자세한 내용은 AWS Organizations 사용 설명서에서 Service control policies을 참조하세요.

- 리소스 제어 정책(RCP) - RCP는 소유한 각 리소스에 연결된 IAM 정책을 업데이트하지 않고 계정의 리소스에 대해 사용 가능한 최대 권한을 설정하는 데 사용할 수 있는 JSON 정책입니다. RCP는 멤버 계정의 리소스에 대한 권한을 제한하며 조직에 속하는지 여부에 AWS 계정 루트 사용자관계없이를 포함한 자격 증명에 대한 유효 권한에 영향을 미칠 수 있습니다. RCP를 AWS 서비스 지원하는 목록을 포함하여 조직 및 RCPs에 대한 자세한 내용은 AWS Organizations 사용 설명서의 리소스 제어 정책(RCPs)을 참조하세요.
- 세션 정책 – 세션 정책은 역할 또는 페더레이션 사용자에 대해 임시 세션을 프로그래밍 방식으로 생성할 때 파라미터로 전달하는 고급 정책입니다. 결과적으로 얻는 세션의 권한은 사용자 또는 역할의 ID 기반 정책의 교차와 세션 정책입니다. 또한 권한을 리소스 기반 정책에서 가져올 수도 있습니다. 이러한 정책 중 하나에 포함된 명시적 거부는 허용을 재정의합니다. 자세한 정보는 IAM 사용 설명서의 세션 정책을 참조하세요.

여러 정책 유형

여러 정책 유형이 요청에 적용되는 경우, 결과 권한은 이해하기가 더 복잡합니다. 가 여러 정책 유형이 관련될 때 요청을 허용할지 여부를 AWS 결정하는 방법을 알아보려면 IAM 사용 설명서의 정책 평가 로직을 참조하세요.

Amazon Lookout for Vision이 IAM과 작동하는 방식

IAM을 사용하여 Lookout for Vision에 대한 액세스를 관리하기 전에 Lookout for Vision와 함께 사용할 수 있는 IAM 기능을 알아보세요.

Amazon Lookout for Vision에서 사용할 수 있는 IAM 기능

IAM 기능	Lookout for Vision 지원
ID 기반 정책	예
리소스 기반 정책	아니요
정책 작업	예
정책 리소스	예
정책 조건 키(서비스별)	예
ACLs	아니요

IAM 기능	Lookout for Vision 지원
ABAC(정책 내 태그)	부분
임시 자격 증명	예
전달 액세스 세션(FAS)	예
서비스 역할	아니요
서비스 연결 역할	아니요

Lookout for Vision 및 기타 AWS 서비스가 대부분의 IAM 기능과 작동하는 방식을 전체적으로 알아보려면 IAM 사용 설명서의 AWS IAM으로 작업하는 서비스를 참조하세요.

Lookout for Vision에 대한 자격 증명 기반 정책

ID 기반 정책 지원: 예

ID 기반 정책은 IAM 사용자, 사용자 그룹 또는 역할과 같은 ID에 연결할 수 있는 JSON 권한 정책 문서입니다. 이러한 정책은 사용자 및 역할이 어떤 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 제어합니다. 자격 증명 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서에서 고객 관리형 정책으로 사용자 지정 IAM 권한 정의를 참조하세요.

IAM ID 기반 정책을 사용하면 허용되거나 거부되는 작업과 리소스뿐 아니라 작업이 허용되거나 거부되는 조건을 지정할 수 있습니다. ID 기반 정책에서는 위탁자가 연결된 사용자 또는 역할에 적용되므로 위탁자를 지정할 수 없습니다. JSON 정책에서 사용하는 모든 요소에 대해 알아보려면 IAM 사용 설명서의 IAM JSON 정책 요소 참조를 참조하세요.

Lookout for Vision에 대한 자격 증명 기반 정책 예제

Lookout for Vision 자격 증명 기반 정책 예시를 보려면 Amazon Lookout for Vision에 대한 자격 증명 기반 정책 예시 섹션을 참조하세요.

Lookout for Vision 내 리소스 기반 정책

리소스 기반 정책 지원: 아니요

리소스 기반 정책은 리소스에 연결하는 JSON 정책 설명서입니다. 리소스 기반 정책의 예제는 IAM 역할 신뢰 정책과 Amazon S3 버킷 정책입니다. 리소스 기반 정책을 지원하는 서비스에서 서비스 관리자

는 이러한 정책을 사용하여 특정 리소스에 대한 액세스를 통제할 수 있습니다. 정책이 연결된 리소스의 경우 정책은 지정된 위탁자가 해당 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 정의합니다. 리소스 기반 정책에서 위탁자를 지정해야 합니다. 보안 주체에는 계정, 사용자, 역할, 페더레이션 사용자 또는가 포함될 수 있습니다 AWS 서비스.

교차 계정 액세스를 활성화하려는 경우, 전체 계정이나 다른 계정의 IAM 개체를 리소스 기반 정책의 위탁자로 지정할 수 있습니다. 리소스 기반 정책에 크로스 계정 보안 주체를 추가하는 것은 트러스트 관계 설정의 절반밖에 되지 않는다는 것을 유념하세요. 보안 주체와 리소스가 다른 경우 신뢰할 수 AWS 계정있는 계정의 IAM 관리자는 보안 주체 엔터티(사용자 또는 역할)에게 리소스에 액세스할 수 있는 권한도 부여해야 합니다. 엔터티에 ID 기반 정책을 연결하여 권한을 부여합니다. 하지만 리소스 기반 정책이 동일 계정의 위탁자에 액세스를 부여하는 경우, 추가 자격 증명 기반 정책이 필요하지 않습니다. 자세한 내용은 IAM 사용 설명서의 교차 계정 리소스 액세스를 참조하세요.

Lookout for Vision에 대한 정책 작업

정책 작업 지원: 예

관리자는 AWS JSON 정책을 사용하여 대상에 액세스할 수 있는 사용자를 지정할 수 있습니다. 즉, 어떤 위탁자가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

JSON 정책의 `Action` 요소는 정책에서 액세스를 허용하거나 거부하는 데 사용할 수 있는 작업을 설명합니다. 정책 작업은 일반적으로 연결된 AWS API 작업과 이름이 동일합니다. 일치하는 API 작업이 없는 권한 전용 작업 같은 몇 가지 예외도 있습니다. 정책에서 여러 작업이 필요한 몇 가지 작업도 있습니다. 이러한 추가 작업을 일컬어 종속 작업이라고 합니다.

연결된 작업을 수행할 수 있는 권한을 부여하기 위한 정책에 작업을 포함하세요.

Lookout for Vision 작업 목록을 보려면 서비스 승인 참조에서 Lookout for Vision에서 정의한 작업을 참조하세요.

Lookout for Vision의 정책 작업은 작업 앞에 다음 접두사를 사용합니다.

```
lookoutvision
```

단일 문에서 여러 작업을 지정하려면 다음과 같이 쉼표로 구분합니다.

```
"Action": [
      "lookoutvision:action1",
```

```
      "lookoutvision:action2"
         ]
```

Lookout for Vision 자격 증명 기반 정책 예시를 보려면 Amazon Lookout for Vision에 대한 자격 증명 기반 정책 예시 섹션을 참조하세요.

Lookout for Vision에 대한 정책 리소스

정책 리소스 지원: 예

관리자는 AWS JSON 정책을 사용하여 대상에 액세스할 수 있는 사용자를 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

`Resource` JSON 정책 요소는 작업이 적용되는 하나 이상의 객체를 지정합니다. 문에는 `Resource`또는 `NotResource`요소가 반드시 추가되어야 합니다. 모범 사례에 따라 Amazon 리소스 이름(ARN)을 사용하여 리소스를 지정합니다. 리소스 수준 권한이라고 하는 특정 리소스 유형을 지원하는 작업에 대해 이를 수행할 수 있습니다.

작업 나열과 같이 리소스 수준 권한을 지원하지 않는 작업의 경우, 와일드카드(*)를 사용하여 해당 문이 모든 리소스에 적용됨을 나타냅니다.

```
"Resource": "*"
```

리소스 유형 및 해당 ARN 목록을 보려면 서비스 승인 참조에서 Lookout for Vision에서 정의한 리소스를 참조하세요. 각 리소스의 ARN을 지정할 수 있는 작업을 알아보려면 Lookout for Vision에서 정의한 작업을 참조하세요.

Lookout for Vision 자격 증명 기반 정책 예시를 보려면 Amazon Lookout for Vision에 대한 자격 증명 기반 정책 예시 섹션을 참조하세요.

Amazon Lookout for Vision에 대한 정책 조건 키

서비스별 정책 조건 키 지원: 예

관리자는 AWS JSON 정책을 사용하여 대상에 액세스할 수 있는 사용자를 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

Condition 요소(또는 Condition 블록)를 사용하면 정책이 발효되는 조건을 지정할 수 있습니다. Condition 요소는 옵션입니다. 같거나 작음과 같은 조건 연산자를 사용하여 정책의 조건을 요청의 값과 일치시키는 조건식을 생성할 수 있습니다.

한 문에서 여러 Condition 요소를 지정하거나 단일 Condition 요소에서 여러 키를 지정하는 경우, AWS 는 논리적 AND 작업을 사용하여 평가합니다. 단일 조건 키에 여러 값을 지정하는 경우는 논리적 OR 작업을 사용하여 조건을 AWS 평가합니다. 문의 권한을 부여하기 전에 모든 조건을 충족해야 합니다.

조건을 지정할 때 자리 표시자 변수를 사용할 수도 있습니다. 예를 들어, IAM 사용자에게 IAM 사용자 이름으로 태그가 지정된 경우에만 리소스에 액세스할 수 있는 권한을 부여할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 IAM 정책 요소: 변수 및 태그를 참조하세요.

AWS 는 전역 조건 키와 서비스별 조건 키를 지원합니다. 모든 AWS 전역 조건 키를 보려면 IAM 사용 설명서의 AWS 전역 조건 컨텍스트 키를 참조하세요.

Lookout for Vision 조건 키 목록을 보려면 서비스 승인 참조의 Lookout for Vision에 사용되는 조건 키를 참조하세요. 조건 키를 사용할 수 있는 작업과 리소스를 알아보려면 Lookout for Vision에서 정의한 작업을 참조하세요.

Lookout for Vision 자격 증명 기반 정책 예시를 보려면 Amazon Lookout for Vision에 대한 자격 증명 기반 정책 예시 섹션을 참조하세요.

Lookout for Vision에 대한 ACL

ACL 지원: 아니요

액세스 제어 목록(ACL)은 어떤 위탁자(계정 멤버, 사용자 또는 역할)가 리소스에 액세스할 수 있는 권한을 가지고 있는지를 제어합니다. ACL은 JSON 정책 문서 형식을 사용하지 않지만 리소스 기반 정책과 유사합니다.

ABAC와 함께 사용하는 Lookout for Vision

ABAC 지원(정책의 태그): 부분적

속성 기반 액세스 제어(ABAC)는 속성에 근거하여 권한을 정의하는 권한 부여 전략입니다. 에서는 AWS이러한 속성을 태그라고 합니다. IAM 엔터티(사용자 또는 역할)와 많은 AWS 리소스에 태그를 연결할 수 있습니다. ABAC의 첫 번째 단계로 개체 및 리소스에 태그를 지정합니다. 그런 다음 위탁자의 태그가 액세스하려는 리소스의 태그와 일치할 때 작업을 허용하도록 ABAC 정책을 설계합니다.

ABAC는 빠르게 성장하는 환경에서 유용하며 정책 관리가 번거로운 상황에 도움이 됩니다.

태그에 근거하여 액세스를 제어하려면 `aws:ResourceTag/key-name`, `aws:RequestTag/key-name` 또는 `aws:TagKeys` 조건 키를 사용하여 정책의 조건 요소에 태그 정보를 제공합니다.

서비스가 모든 리소스 유형에 대해 세 가지 조건 키를 모두 지원하는 경우, 값은 서비스에 대해 예입니다. 서비스가 일부 리소스 유형에 대해서만 세 가지 조건 키를 모두 지원하는 경우, 값은 부분적입니다.

ABAC에 대한 자세한 내용은 IAM 사용 설명서의 ABAC 권한 부여를 통한 권한 정의를 참조하세요. ABAC 설정 단계가 포함된 자습서를 보려면 IAM 사용 설명서의 속성 기반 액세스 제어(ABAC) 사용을 참조하세요.

Lookout for Vision에서 임시 자격 증명 사용

임시 자격 증명 지원: 예

임시 자격 증명을 사용하여 로그인할 때 일부 AWS 서비스 가 작동하지 않습니다. 임시 자격 증명으로 AWS 서비스 작업하는을 비롯한 자세한 내용은 AWS 서비스 IAM 사용 설명서의 IAM으로 작업하는를 참조하세요.

사용자 이름과 암호를 제외한 방법을 AWS Management Console 사용하여에 로그인하는 경우 임시 자격 증명을 사용합니다. 예를 들어 회사의 SSO(Single Sign-On) 링크를 AWS 사용하여에 액세스하면 해당 프로세스가 임시 자격 증명을 자동으로 생성합니다. 또한 콘솔에 사용자로 로그인한 다음 역할을 전환할 때 임시 자격 증명을 자동으로 생성합니다. 역할 전환에 대한 자세한 내용은 IAM 사용 설명서의 사용자에서 IAM 역할로 전환(콘솔)을 참조하세요.

AWS CLI 또는 AWS API를 사용하여 임시 자격 증명을 수동으로 생성할 수 있습니다. 그런 다음 이러한 임시 자격 증명을 사용하여 장기 액세스 키를 사용하는 대신 동적으로 임시 자격 증명을 생성하는 access AWS. AWS recommends에 액세스할 수 있습니다. 자세한 정보는 IAM의 임시 보안 자격 증명 섹션을 참조하세요.

Lookout for Vision을 위한 전달 액세스 세션

전달 액세스 세션(FAS) 지원: 예

IAM 사용자 또는 역할을 사용하여에서 작업을 수행하는 경우 AWS보안 주체로 간주됩니다. 일부 서비스를 사용하는 경우, 다른 서비스에서 다른 작업을 시작하는 작업을 수행할 수 있습니다. FAS는를 호출하는 보안 주체의 권한을 다운스트림 서비스에 AWS 서비스 대한 요청과 AWS 서비스함께 사용합니다. FAS 요청은 서비스가 다른 AWS 서비스 또는 리소스와의 상호 작용을 완료해야 하는 요청을 수신할 때만 수행됩니다. 이 경우, 두 작업을 모두 수행할 수 있는 권한이 있어야 합니다. FAS 요청 시 정책 세부 정보는 전달 액세스 세션을 참조하세요.

Lookout for Vision의 서비스 역할

서비스 역할 지원: 아니요

서비스 역할은 서비스가 사용자를 대신하여 작업을 수행하는 것으로 가정하는 IAM 역할입니다. IAM 관리자는 IAM 내에서 서비스 역할을 생성, 수정 및 삭제할 수 있습니다. 자세한 정보는 IAM 사용 설명서의 Create a role to delegate permissions to an AWS 서비스를 참조하세요.

> **Warning**
> 서비스 역할에 대한 권한을 변경하면 Lookout for Vision 기능이 중단될 수 있습니다. Lookout for Vision에서 관련 지침을 제공하는 경우에만 서비스 역할을 편집하세요.

Lookout for Vision의 서비스 연결 역할

서비스 링크 역할 지원: 아니요

서비스 연결 역할은에 연결된 서비스 역할의 한 유형입니다 AWS 서비스. 서비스는 사용자를 대신하여 작업을 수행하기 위해 역할을 수임할 수 있습니다. 서비스 연결 역할은에 표시 AWS 계정 되며 서비스가 소유합니다. IAM 관리자는 서비스 링크 역할의 권한을 볼 수 있지만 편집은 할 수 없습니다.

서비스 연결 역할 생성 또는 관리에 대한 자세한 내용은 IAM으로 작업하는AWS 서비스를 참조하세요. 서비스 연결 역할 열에서 `Yes`이(가) 포함된 서비스를 테이블에서 찾습니다. 해당 서비스에 대한 서비스 연결 역할 설명서를 보려면 예 링크를 선택합니다.

Amazon Lookout for Vision에 대한 자격 증명 기반 정책 예시

기본적으로 사용자 및 역할에는 Lookout for Vision 리소스를 생성하거나 수정할 수 있는 권한이 없습니다. 또한 AWS Management Console, AWS Command Line Interface (AWS CLI) 또는 AWS API를 사용하여 작업을 수행할 수 없습니다. 사용자에게 사용자가 필요한 리소스에서 작업을 수행할 권한을 부여하려면 IAM 관리자가 IAM 정책을 생성하면 됩니다. 그런 다음 관리자가 IAM 정책을 역할에 추가하고, 사용자가 역할을 맡을 수 있습니다.

이러한 예제 JSON 정책 문서를 사용하여 IAM 자격 증명 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서의 IAM 정책 생성(콘솔)을 참조하세요.

각 리소스 유형에 대한 ARN 형식을 포함하여 Lookout for Vision에서 정의한 작업 및 리소스 유형에 대한 자세한 내용은 서비스 인증 참조에서 Lookout for Vision에 대한 작업, 리소스 및 조건 키를 참조하세요.

주제

정책 모범 사례

ID 기반 정책에 따라 계정에서 사용자가 Lookout for Vision 리소스를 생성, 액세스 또는 삭제할 수 있는지 여부가 결정됩니다. 이 작업으로 인해 AWS 계정에 비용이 발생할 수 있습니다. ID 기반 정책을 생성하거나 편집할 때는 다음 지침과 권장 사항을 따릅니다.

- AWS 관리형 정책을 시작하고 최소 권한으로 이동 - 사용자 및 워크로드에 권한 부여를 시작하려면 많은 일반적인 사용 사례에 대한 권한을 부여하는 AWS 관리형 정책을 사용합니다. 에서 사용할 수 있습니다 AWS 계정. 사용 사례에 맞는 AWS 고객 관리형 정책을 정의하여 권한을 추가로 줄이는 것이 좋습니다. 자세한 정보는 IAM 사용 설명서의 AWS 관리형 정책 또는 AWS 직무에 대한 관리형 정책을 참조하세요.
- 최소 권한 적용 – IAM 정책을 사용하여 권한을 설정하는 경우, 작업을 수행하는 데 필요한 권한만 부여합니다. 이렇게 하려면 최소 권한으로 알려진 특정 조건에서 특정 리소스에 대해 수행할 수 있는 작업을 정의합니다. IAM을 사용하여 권한을 적용하는 방법에 대한 자세한 정보는 IAM 사용 설명서에 있는 IAM의 정책 및 권한을 참조하세요.
- IAM 정책의 조건을 사용하여 액세스 추가 제한 – 정책에 조건을 추가하여 작업 및 리소스에 대한 액세스를 제한할 수 있습니다. 예를 들어, SSL을 사용하여 모든 요청을 전송해야 한다고 지정하는 정책 조건을 작성할 수 있습니다. 조건을 사용하여 AWS 서비스와 같은 특정를 통해 사용되는 경우 서비스 작업에 대한 액세스 권한을 부여할 수도 있습니다 AWS CloudFormation. 자세한 정보는 IAM 사용 설명서의 IAM JSON 정책 요소: 조건을 참조하세요.
- IAM Access Analyzer를 통해 IAM 정책을 확인하여 안전하고 기능적인 권한 보장 - IAM Access Analyzer에서는 IAM 정책 언어(JSON)와 모범 사례가 정책에서 준수되도록 새로운 및 기존 정책을 확인합니다. IAM Access Analyzer는 100개 이상의 정책 확인 항목과 실행 가능한 추천을 제공하여 안전하고 기능적인 정책을 작성하도록 돕습니다. 자세한 내용은 IAM 사용 설명서의 IAM Access Analyzer에서 정책 검증을 참조하세요.
- 다중 인증(MFA) 필요 -에서 IAM 사용자 또는 루트 사용자가 필요한 시나리오가 있는 경우 추가 보안을 위해 MFA를 AWS 계정켭니다. API 작업을 직접 호출할 때 MFA가 필요하면 정책에 MFA 조건을 추가합니다. 자세한 내용은 IAM 사용 설명서의 MFA를 통한 보안 API 액세스를 참조하세요.

IAM의 모범 사례에 대한 자세한 내용은 IAM 사용 설명서의 IAM의 보안 모범 사례를 참조하세요.

Amazon Lookout for Vision 프로젝트 하나에 액세스

이 예제에서는 AWS 계정의 사용자에게 Amazon Lookout for Vision 프로젝트 중 하나에 대한 액세스 권한을 부여하려고 합니다.

```
{
    "Sid": "SpecificProjectOnly",
    "Effect": "Allow",
    "Action": [
        "lookoutvision:DetectAnomalies"
    ],
    "Resource": "arn:aws:lookoutvision:us-east-1:123456789012:model/myproject/*"
}
```

태그 기반 정책 예제

태그 기반 정책은 보안 주체가 태그가 지정된 리소스에 대해 수행할 수 있는 작업을 지정하는 JSON 정책 문서입니다.

예제: 태그를 사용하여 리소스 액세스

이 예제 정책은 `stage` 키와 `production` 값으로 태그가 지정된 모든 리소스에 대해 `DetectAnomalies` 작업을 사용할 수 있는 권한을 AWS 계정의 사용자 또는 역할에 부여합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "LookoutVision:DetectAnomalies"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/stage": "production"
                }
            }
        }
   ]
}
```

태그를 사용하여 특정 Amazon Lookout for Vision 작업에 대한 액세스를 거부할 수 있습니다.

이 예제 정책은 `stage` 키와 `production` 값으로 태그가 지정된 모든 모델에서 `DeleteModel`또는 `StopModel` 작업을 호출할 수 있는 AWS 계정의 사용자 또는 역할에 대한 권한을 거부합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "LookoutVision:DeleteModel",
                "LookoutVision:StopModel"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/stage": "production"
                }
            }
        }
   ]
}
```

AWS Amazon Lookout for Vision에 대한 관리형 정책

AWS 관리형 정책은에서 생성하고 관리하는 독립 실행형 정책입니다 AWS. AWS 관리형 정책은 사용자, 그룹 및 역할에 권한 할당을 시작할 수 있도록 많은 일반적인 사용 사례에 대한 권한을 제공하도록 설계되었습니다.

AWS 관리형 정책은 모든 AWS 고객이 사용할 수 있으므로 특정 사용 사례에 대해 최소 권한 권한을 부여하지 않을 수 있습니다. 사용 사례에 고유한 고객 관리형 정책을 정의하여 권한을 줄이는 것이 좋습니다.

AWS 관리형 정책에 정의된 권한은 변경할 수 없습니다. 가 AWS 관리형 정책에 정의된 권한을 AWS 업데이트하면 정책이 연결된 모든 보안 주체 자격 증명(사용자, 그룹 및 역할)에 영향을 미칩니다. AWS 는 새 AWS 서비스 가 시작되거나 기존 서비스에 새 API 작업을 사용할 수 있게 되면 AWS 관리형 정책을 업데이트할 가능성이 높습니다.

자세한 내용은 IAM 사용 설명서의 AWS 관리형 정책을 참조하세요.

AWS 관리형 정책: Amazon LookOutVisionReadOnlyAccess

AmazonLookoutVisionReadOnlyAccess정책을 사용하여 다음과 같은 Amazon Lookout for Vision 작업 (SDK 작업) 을 통해 사용자가 Amazon Lookout for Vision (및 종속성) 에 읽기 전용으로 액세스할 수 있도록 허용할 수 있습니다. 예를 들면, 기존 모델에 대한 정보를 가져오는 데 DescribeModel을 사용할 수 있습니다.

- DescribeDataset
- DescribeModel
- DescribeModelPackagingJob
- DescribeProject
- ListDatasetEntries
- ListModelPackagingJobs
- ListModels
- ListProjects
- ListTagsForResource

읽기 전용 작업을 호출하려면 사용자에게 Amazon S3 버킷 권한이 필요하지 않습니다. 하지만 작업 응답에는 Amazon S3 버킷에 대한 참조가 포함될 수 있습니다. 예를 들어, ListDatasetEntries의 응답 양식의 source-ref 항목은 Amazon S3 버킷의 이미지에 대한 참조입니다. 사용자가 참조된 버킷에 액세스해야 하는 경우 Amazon S3 버킷 권한을 추가합니다. 예를 들어, 사용자가 source-ref 필드에서 참조하는 이미지를 다운로드하기를 원할 수 있습니다. 자세한 내용은 Amazon S3 버킷 권한 부여 단원을 참조하십시오.

AmazonLookoutVisionReadOnlyAccess 정책을 IAM 보안 인증에 연결할 수 있습니다.

권한 세부 정보

이 정책에는 다음 권한이 포함되어 있습니다.

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [
        {
            "Sid": "LookoutVisionReadOnlyAccess",
            "Effect": "Allow",
            "Action": [
                "lookoutvision:DescribeDataset",
                "lookoutvision:DescribeModel",
                "lookoutvision:DescribeProject",
                "lookoutvision:DescribeModelPackagingJob",
                "lookoutvision:ListDatasetEntries",
                "lookoutvision:ListModels",
                "lookoutvision:ListProjects",
                "lookoutvision:ListTagsForResource",
                "lookoutvision:ListModelPackagingJobs"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS관리형 정책: AmazonLookOutVisionFullAccess

AmazonLookoutVisionFullAccess정책을 사용하여 Amazon Lookout for Vision 작업 (SDK 작업)을 통해 Amazon Lookout for Vision (및 종속성) 에 대한 전체 액세스 권한을 사용자에게 허용할 수 있습니다. 예를 들어 Amazon Lookout for Vision 콘솔을 사용하지 않고도 모델을 학습시킬 수 있습니다. 자세한 내용은 작업을 참조하십시오.

데이터 세트를 생성하거나(CreateDataset) 모델()을 생성하려면 데이터 세트 이미지, Amazon SageMaker AI Ground Truth 매니페스트 파일 및 훈련 출력을 저장하는 Amazon S3 버킷에 대한 전체 액세스 권한이 CreateModel사용자에게 있어야 합니다. Amazon SageMaker 자세한 내용은 2단계: 권한 설정 단원을 참조하십시오.

AmazonLookoutVisionConsoleFullAccess 정책을 사용하여 Amazon Lookout for Vision SDK 작업에 권한을 부여할 수도 있습니다.

AmazonLookoutVisionFullAccess 정책을 IAM 보안 인증에 연결할 수 있습니다.

권한 세부 정보

이 정책에는 다음 권한이 포함되어 있습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "LookoutVisionFullAccess",
            "Effect": "Allow",
            "Action": [
                "lookoutvision:*"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS 관리형 정책: Amazon LookOutVision ConsoleAccess

`AmazonLookoutVisionFullAccess` 정책을 사용하여 사용자가 Amazon Lookout for Vision 콘솔, 작업 (SDK 작업) 및 서비스에 있는 모든 종속성에 대한 전체 액세스 권한을 허용할 수 있습니다. 자세한 내용은 Amazon Lookout for Vision을 시작하기 단원을 참조하십시오.

`LookoutVisionConsoleFullAccess` 정책에는 Amazon Lookout for Vision 콘솔 버킷에 대한 권한이 포함되어 있습니다. 콘솔에 대한 자세한 내용은 3단계: 콘솔 버킷 생성 단원을 참조하십시오. 데이터세트, 이미지 및 Amazon SageMaker AI Ground Truth 매니페스트 파일을 다른 Amazon S3 버킷에 저장하려면 사용자에게 추가 권한이 필요합니다. 자세한 내용은 the section called “Amazon S3 버킷 권한 설정” 단원을 참조하십시오.

`AmazonLookoutVisionConsoleFullAccess` 정책을 IAM 보안 인증에 연결할 수 있습니다.

권한 그룹화

이 정책은 제공된 권한에 따라 명령문으로 그룹화됩니다.

- `LookoutVisionFullAccess`— 액세스를 통해 모든 Lookout for Vision 작업을 수행할 수 있습니다.
- `LookoutVisionConsoleS3BucketSearchAccess`— 호출자가 소유한 모든 Amazon S3 버킷을 나열할 수 있습니다. Lookout for Vision은 이 작업을 사용하여 AWS 지역별 Lookout for Vision 콘솔 버킷 (호출자 계정에 버킷이 있는 경우) 을 식별합니다.
- `LookoutVisionConsoleS3BucketFirstUseSetupAccessPermissions`— Lookout for Vision 콘솔 버킷 이름 패턴과 일치하는 Amazon S3 버킷을 생성하고 구성할 수 있습니다. Lookout

for Vision은 버킷을 찾을 수 없는 경우 이러한 작업을 사용하여 지역별 Lookout for Vision 콘솔 버킷을 만들고 구성합니다.

- `LookoutVisionConsoleS3BucketAccess`— Lookout for Vision 콘솔 버킷 이름 패턴과 일치하는 버킷에서 종속 Amazon S3 작업을 수행할 수 있습니다. Lookout for `s3:ListBucket` Vision은 Amazon S3 버킷에서 데이터 세트를 생성할 때와 시험 탐지 작업을 시작할 때 이미지 객체를 검색하는 데 사용합니다. Lookout for Vision은 `s3:GetBucketLocation` 및 `s3:GetBucketVersioning`를 사용하여 버킷의 AWS 리전, 소유자 및 구성을 다음과 같은 일부로 검증합니다.
  - 데이터 세트 생성
  - 모델 교육
  - 평가판 탐지 작업 시작
  - 평가판 탐지 피드백 수행

  `LookoutVisionConsoleS3ObjectAccess`— Lookout for Vision 콘솔 버킷 이름 패턴과 일치하는 버킷 내에서 Amazon S3 객체를 읽고 쓸 수 있습니다. Lookout for Vision은 이러한 작업을 사용하여 콘솔 갤러리 보기에 이미지를 표시하고 데이터 세트에 사용할 새 이미지를 업로드합니다. 또한 이러한 권한을 통해 Lookout for Vision은 데이터 세트를 생성하고, 모델을 학습시키고, 평가판 탐지 작업을 시작하고, 임상시험 탐지 피드백을 수행하는 동안 메타데이터를 작성할 수 있습니다.
- `LookoutVisionConsoleDatasetLabelingToolsAccess` - 종속 Amazon SageMaker AI GroundTruth 레이블 지정 작업을 허용합니다. Lookout for Vision은 이러한 작업을 사용하여 S3 버킷에서 이미지를 스캔하고, GroundTruth 매니페스트 파일을 생성하고, 시험 탐지 작업 결과에 검증 레이블로 주석을 답니다.
- `LookoutVisionConsoleDashboardAccess`- 아마존 CloudWatch 지표를 읽을 수 있습니다. Lookout for Vision은 이러한 작업을 사용하여 대시보드 그래프와 예외 항목 탐지 통계를 채웁니다.
- `LookoutVisionConsoleTagSelectorAccess`— 계정별 태그 키 및 태그 값 제안을 읽을 수 있습니다. Lookout for Vision은 이러한 권한을 사용하여 태그 관리 콘솔 페이지 내에서 태그 키 및 태그 값에 대한 권장 사항을 제공합니다.
- `LookoutVisionConsoleKmsKeySelectorAccess` - AWS Key Management Service (KMS) 키 및 별칭을 나열할 수 있습니다. Amazon Lookout for Vision은 이 권한을 사용하여 암호화를 위한 고객 관리형 KMS 키를 지원하는 특정 Lookout for Vision 작업의 권장 태그 선택 항목에 KMS 키를 채웁니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Sid": "LookoutVisionFullAccess",
            "Effect": "Allow",
            "Action": [
                "lookoutvision:*"
            ],
            "Resource": "*"
        },
        {
            "Sid": "LookoutVisionConsoleS3BucketSearchAccess",
            "Effect": "Allow",
            "Action": [
                "s3:ListAllMyBuckets"
            ],
            "Resource": "*"
        },
        {
            "Sid": "LookoutVisionConsoleS3BucketFirstUseSetupAccess",
            "Effect": "Allow",
            "Action": [
                "s3:CreateBucket",
                "s3:PutBucketVersioning",
                "s3:PutLifecycleConfiguration",
                "s3:PutEncryptionConfiguration",
                "s3:PutBucketPublicAccessBlock"
            ],
            "Resource": "arn:aws:s3:::lookoutvision-*"
        },
        {
            "Sid": "LookoutVisionConsoleS3BucketAccess",
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket",
                "s3:GetBucketLocation",
                "s3:GetBucketAcl",
                "s3:GetBucketVersioning"
            ],
            "Resource": "arn:aws:s3:::lookoutvision-*"
        },
        {
            "Sid": "LookoutVisionConsoleS3ObjectAccess",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
```

```
                "s3:GetObjectVersion",
                "s3:PutObject",
                "s3:AbortMultipartUpload",
                "s3:ListMultipartUploadParts"
            ],
            "Resource": "arn:aws:s3:::lookoutvision-*/*"
        },
        {
            "Sid": "LookoutVisionConsoleDatasetLabelingToolsAccess",
            "Effect": "Allow",
            "Action": [
                "groundtruthlabeling:RunGenerateManifestByCrawlingJob",
                "groundtruthlabeling:AssociatePatchToManifestJob",
                "groundtruthlabeling:DescribeConsoleJob"
            ],
            "Resource": "*"
        },
        {
            "Sid": "LookoutVisionConsoleDashboardAccess",
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetMetricData",
                "cloudwatch:GetMetricStatistics"
            ],
            "Resource": "*"
        },
        {
            "Sid": "LookoutVisionConsoleTagSelectorAccess",
            "Effect": "Allow",
            "Action": [
                "tag:GetTagKeys",
                "tag:GetTagValues"
            ],
            "Resource": "*"
        },
        {
            "Sid": "LookoutVisionConsoleKmsKeySelectorAccess",
            "Effect": "Allow",
            "Action": [
                "kms:ListAliases"
            ],
            "Resource": "*"
        }
    ]
```

```
}
```

AWS 관리형 정책: Amazon LookOutVision ConsoleReadOnlyAccess

AmazonLookoutVisionConsoleReadOnlyAccess정책을 사용하여 사용자가 Amazon Lookout for Vision 콘솔, 작업 (SDK 작업) 및 서비스에 있는 모든 종속성에 대한 읽기 전용 액세스를 허용할 수 있습니다.

이 AmazonLookoutVisionConsoleReadOnlyAccess 정책에는 Amazon Lookout for Vision 콘솔 버킷에 대한 Amazon S3 권한이 포함됩니다. 데이터 세트 이미지 또는 Amazon SageMaker AI Ground Truth 매니페스트 파일이 다른 Amazon S3 버킷에 있는 경우 사용자에게 추가 권한이 필요합니다. 자세한 내용은 the section called “Amazon S3 버킷 권한 설정” 단원을 참조하십시오.

AmazonLookoutVisionConsoleReadOnlyAccess 정책을 IAM 보안 인증에 연결할 수 있습니다.

권한 그룹화

이 정책은 제공된 권한에 따라 명령문으로 그룹화됩니다.

- LookoutVisionReadOnlyAccess— 액세스 권한을 통해 읽기 전용 Lookout for Vision 작업을 수행할 수 있습니다.
- LookoutVisionConsoleS3BucketSearchAccess— 호출자가 소유한 모든 S3 버킷을 나열할 수 있습니다. Lookout for Vision은 이 작업을 사용하여 AWS 지역별 Lookout for Vision 콘솔 버킷 (호출자 계정에 버킷이 있는 경우) 을 식별합니다.
- LookoutVisionConsoleS3ObjectReadAccess— Lookout for Vision 콘솔 버킷에서 Amazon S3 객체 및 Amazon S3 객체 버전을 읽을 수 있습니다. Lookout for Vision은 이러한 작업을 사용하여 데이터세트, 모델 및 시험 탐지에 이미지를 표시합니다.
- LookoutVisionConsoleDashboardAccess— Amazon CloudWatch 지표를 읽을 수 있습니다. Lookout for Vision은 이러한 작업을 사용하여 감지된 대시보드 그래프 및 이상 현상에 대한 통계를 채웁니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "LookoutVisionReadOnlyAccess",
            "Effect": "Allow",
```

```
            "Action": [
                "lookoutvision:DescribeDataset",
                "lookoutvision:DescribeModel",
                "lookoutvision:DescribeProject",
                "lookoutvision:DescribeTrialDetection",
                "lookoutvision:DescribeModelPackagingJob",
                "lookoutvision:ListDatasetEntries",
                "lookoutvision:ListModels",
                "lookoutvision:ListProjects",
                "lookoutvision:ListTagsForResource",
                "lookoutvision:ListTrialDetections",
                "lookoutvision:ListModelPackagingJobs"
            ],
            "Resource": "*"
        },
        {
            "Sid": "LookoutVisionConsoleS3BucketSearchAccess",
            "Effect": "Allow",
            "Action": [
                "s3:ListAllMyBuckets"
            ],
            "Resource": "*"
        },
        {
            "Sid": "LookoutVisionConsoleS3ObjectReadAccess",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": "arn:aws:s3:::lookoutvision-*/*"
        },
        {
            "Sid": "LookoutVisionConsoleDashboardAccess",
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetMetricData",
                "cloudwatch:GetMetricStatistics"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS 관리형 정책에 대한 Vision 업데이트 확인

이 서비스가 이러한 변경 사항을 추적하기 시작한 이후 Lookout for Vision의 AWS 관리형 정책 업데이트에 대한 세부 정보를 봅니다. 이 페이지의 변경 사항에 대한 자동 알림을 받으려면 Lookout for Vision 문서 기록 페이지에서 RSS 피드를 구독하세요.

모델 패키징 작업 추가	Amazon Lookout for Vision은 AmazonLookoutVisionFullAccess 및 AmazonLookoutVisionConsoleFullAccess에 다음과 같은 모델 패키징 작업을 추가했습니다. • DescribeModelPackagingJob • ListModelPackagingJobs • StartModelPackagingJob Amazon Lookout for Vision은 AmazonLookoutVisionReadOnlyAccess 및 AmazonLookoutVisionConsoleReadOnlyAccess 정책에 다음과 같은 모델 패키징 작업을 추가했습니다. • DescribeModelPackagingJob • ListModelPackagingJobs	2021년 12월 7일
새 정책이 추가됨	Amazon Lookout for Vision에 다음과 같은 정책이 추가됨 • AmazonLookoutVisionReadOnlyAccess • AmazonLookoutVisionFullAccess • 아마존 룩아웃 비전 콘솔 풀 액세스 • AmazonLookoutVisionConsoleReadOnlyAccess	2021년 5월 11일
Lookout for Vision에서 변경 내용 추적 시작	Amazon Lookout for Vision은 AWS 관리형 정책에 대한 변경 사항 추적을 시작했습니다.	2021년 3월 1일

Amazon Lookout for Vision ID 및 액세스 문제 해결

다음 정보를 사용하여 Lookout for Vision 및 IAM에서 발생할 수 있는 공통적인 문제를 진단하고 수정할 수 있습니다.

주제

- Lookout for Vision에서 작업을 수행할 권한이 없음
- iam:PassRole을 수행하도록 인증되지 않음
- 내 외부의 사람들이 내 Lookout for Vision 리소스에 액세스 AWS 계정 하도록 허용하고 싶습니다.

Lookout for Vision에서 작업을 수행할 권한이 없음

작업을 수행할 권한이 없다는 오류가 수신되면, 작업을 수행할 수 있도록 정책을 업데이트해야 합니다.

다음의 예제 오류는 `mateojackson` IAM 사용자가 콘솔을 사용하여 가상 *`my-example-widget`* 리소스에 대한 세부 정보를 보려고 하지만 가상 `lookoutvision:`*`GetWidget`* 권한이 없을 때 발생합니다.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 lookoutvision:GetWidget on resource: my-example-widget
```

이 경우, `lookoutvision:`*`GetWidget`* 작업을 사용하여 *`my-example-widget`* 리소스에 액세스할 수 있도록 `mateojackson` 사용자 정책을 업데이트해야 합니다.

도움이 필요한 경우 AWS 관리자에게 문의하세요. 관리자는 로그인 자격 증명을 제공한 사람입니다.

iam:PassRole을 수행하도록 인증되지 않음

`iam:PassRole` 작업을 수행할 수 있는 권한이 없다는 오류가 수신되면 Lookout for Vision에 역할을 전달할 수 있도록 정책을 업데이트해야 합니다.

일부 AWS 서비스 에서는 새 서비스 역할 또는 서비스 연결 역할을 생성하는 대신 기존 역할을 해당 서비스에 전달할 수 있습니다. 이렇게 하려면 사용자가 서비스에 역할을 전달할 수 있는 권한을 가지고 있어야 합니다.

다음 예시 오류는 `marymajor`라는 IAM 사용자가 콘솔을 사용하여 Lookout for Vision에서 작업을 수행하려고 하는 경우에 발생합니다. 하지만 작업을 수행하려면 서비스 역할이 부여한 권한이 서비스에 있어야 합니다. Mary는 서비스에 역할을 전달할 수 있는 권한을 가지고 있지 않습니다.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

이 경우, Mary가 `iam:PassRole` 작업을 수행할 수 있도록 Mary의 정책을 업데이트해야 합니다.

도움이 필요한 경우 AWS 관리자에게 문의하세요. 관리자는 로그인 자격 증명을 제공한 사람입니다.

내 외부의 사람들이 내 Lookout for Vision 리소스에 액세스 AWS 계정 하도록 허용하고 싶습니다.

다른 계정의 사용자 또는 조직 외부의 사람이 리소스에 액세스할 때 사용할 수 있는 역할을 생성할 수 있습니다. 역할을 수임할 신뢰할 수 있는 사람을 지정할 수 있습니다. 리소스 기반 정책 또는 액세스 제어 목록(ACL)을 지원하는 서비스의 경우, 이러한 정책을 사용하여 다른 사람에게 리소스에 대한 액세스 권한을 부여할 수 있습니다.

자세히 알아보려면 다음을 참조하세요.

- 에서 이러한 기능을 지원하는지 여부를 알아보려면 Amazon Lookout for Vision이 IAM과 작동하는 방식 단원을 참조하세요.
- 소유 AWS 계정 한에서 리소스에 대한 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 소유 AWS 계정 한 다른의 IAM 사용자에게 액세스 권한 제공을 참조하세요.
- 타사에 리소스에 대한 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 타사 AWS 계정 소유에 대한 액세스 권한 제공을 AWS 계정참조하세요.
- ID 페더레이션을 통해 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 외부에서 인증된 사용자에게 액세스 권한 제공(ID 페더레이션)을 참조하세요.
- 크로스 계정 액세스에 대한 역할과 리소스 기반 정책 사용의 차이점을 알아보려면 IAM 사용 설명서의 IAM의 크로스 계정 리소스 액세스를 참조하세요.

Amazon Lookout for Vision에 대한 규정 준수 검증

타사 감사자는 여러 규정 준수 프로그램의 일환으로 Amazon Lookout for Vision의 보안 및 AWS 규정 준수를 평가합니다. Amazon Lookout for Vision은 일반 데이터 보호 규정 (GDPR)을 준수합니다.

AWS 서비스 가 특정 규정 준수 프로그램의 범위 내에 있는지 알아보려면 AWS 서비스 규정 준수 프로그램 범위 섹션을 참조하고 관심 있는 규정 준수 프로그램을 선택합니다. 일반 정보는 AWS 규정 준수 프로그램.

를 사용하여 타사 감사 보고서를 다운로드할 수 있습니다 AWS Artifact. 자세한 내용은 Downloading Reports inDownloading AWS Artifact 참조하세요.

사용 시 규정 준수 책임은 데이터의 민감도, 회사의 규정 준수 목표 및 관련 법률과 규정에 따라 AWS 서비스 결정됩니다.는 규정 준수를 지원하기 위해 다음 리소스를 AWS 제공합니다.

- 보안 규정 준수 및 거버넌스 - 이러한 솔루션 구현 가이드에서는 아키텍처 고려 사항을 설명하고 보안 및 규정 준수 기능을 배포하는 단계를 제공합니다.
- HIPAA 적격 서비스 참조 - HIPAA 적격 서비스가 나열되어 있습니다. 모든가 HIPAA에 적합한 AWS 서비스 것은 아닙니다.
- AWS 규정 준수 리소스 -이 워크북 및 가이드 모음은 업계 및 위치에 적용될 수 있습니다.
- AWS 고객 규정 준수 가이드 - 규정 준수의 관점에서 공동 책임 모델을 이해합니다. 이 가이드에는 여러 프레임워크(미국 국립표준기술연구소(NIST), 결제카드 산업 보안 표준 위원회(PCI), 국제표준화기구(ISO))의 보안 제어에 대한 지침을 보호하고 AWS 서비스 매핑하는 모범 사례가 요약되어 있습니다.
- AWS Config 개발자 안내서의 규칙을 사용하여 리소스 평가 -이 AWS Config 서비스는 리소스 구성이 내부 관행, 업계 지침 및 규정을 얼마나 잘 준수하는지 평가합니다.
- AWS Security Hub - 이를 AWS 서비스 통해 내 보안 상태를 포괄적으로 볼 수 있습니다 AWS. Security Hub는 보안 컨트롤을 사용하여 AWS 리소스를 평가하고 보안 업계 표준 및 모범 사례에 대한 규정 준수를 확인합니다. 지원되는 서비스 및 제어 목록은 Security Hub 제어 참조를 참조하세요.
- Amazon GuardDuty - 의심스러운 악의적인 활동이 있는지 환경을 모니터링하여 사용자, AWS 계정 워크로드, 컨테이너 및 데이터에 대한 잠재적 위협을 AWS 서비스 탐지합니다. GuardDuty는 특정 규정 준수 프레임워크에서 요구하는 침입 탐지 요구 사항을 충족하여 PCI DSS와 같은 다양한 규정 준수 요구 사항을 따르는 데 도움을 줄 수 있습니다.
- AWS Audit Manager - 이를 AWS 서비스 통해 AWS 사용량을 지속적으로 감사하여 위험 및 규정 및 업계 표준 준수를 관리하는 방법을 간소화할 수 있습니다.

Amazon Lookout for Vision의 복원성

AWS 글로벌 인프라는 AWS 리전 및 가용 영역을 중심으로 구축됩니다. AWS 리전은 지연 시간이 짧고 처리량이 높으며 중복성이 높은 네트워킹과 연결된 물리적으로 분리되고 격리된 여러 가용 영역을 제공합니다. 가용 영역을 사용하면 중단 없이 영역 간에 자동으로 장애 극복 조치가 이루어지는 애플리케이션 및 데이터베이스를 설계하고 운영할 수 있습니다. 가용 영역은 기존의 단일 또는 다중 데이터 센터 인프라보다 가용성, 내결함성, 확장성이 뛰어납니다.

AWS 리전 및 가용 영역에 대한 자세한 내용은 AWS 글로벌 인프라를 참조하세요.

Amazon Lookout for Vision 인프라 보안

관리형 서비스인 Amazon Lookout for Vision은 AWS 글로벌 네트워크 보안으로 보호됩니다. AWS 보안 서비스 및가 인프라를 AWS 보호하는 방법에 대한 자세한 내용은 AWS 클라우드 보안을 참조하세요. 인프라 보안 모범 사례를 사용하여 AWS 환경을 설계하려면 Security Pillar AWS Well-Architected Framework의 인프라 보호를 참조하세요.

AWS 게시된 API 호출을 사용하여 네트워크를 통해 Lookout for Vision에 액세스합니다. 고객은 다음을 지원해야 합니다.

- Transport Layer Security(TLS) TLS 1.2는 필수이며 TLS 1.3을 권장합니다.
- DHE(Ephemeral Diffie-Hellman) 또는 ECDHE(Elliptic Curve Ephemeral Diffie-Hellman)와 같은 완전 전송 보안(PFS)이 포함된 암호 제품군 Java 7 이상의 최신 시스템은 대부분 이러한 모드를 지원합니다.

또한 요청은 액세스 키 ID 및 IAM 위탁자와 관련된 보안 암호 액세스 키를 사용하여 서명해야 합니다. 또는 AWS Security Token Service(AWS STS)를 사용하여 임시 자격 증명을 생성하여 요청에 서명할 수 있습니다.

Amazon Lookout for Vision 모니터링

모니터링은 Amazon Lookout for Vision와 다른 AWS 솔루션의 안정성, 가용성 및 성능을 유지하는 데 중요한 부분입니다. AWS는 Lookout for Vision를 관찰하고, 문제 발생 시 보고하고, 적절한 경우 자동 조치를 취하는 다음과 같은 모니터링 도구를 제공합니다.

- Amazon CloudWatch는 AWS 리소스와 실행 중인 애플리케이션을 AWS 실시간으로 모니터링합니다. 지표를 수집 및 추적하고, 사용자 지정 대시보드를 생성할 수 있으며, 지정된 지표가 지정한 임곗값에 도달하면 사용자에게 알리거나 조치를 취하도록 경보를 설정할 수 있습니다. 예를 들어 CloudWatch에서 Amazon EC2 인스턴스의 CPU 사용량 또는 기타 지표를 추적하고 필요할 때 자동으로 새 인스턴스를 시작할 수 있습니다. 자세한 내용은 CloudWatch 사용 설명서를 참조하세요.
- Amazon CloudWatch Logs로 Amazon EC2 인스턴스, CloudTrail, 기타 소스의 로그 파일을 모니터링, 저장 및 액세스할 수 있습니다. CloudWatch Logs는 로그 파일의 정보를 모니터링하고 특정 임곗값에 도달하면 사용자에게 알릴 수 있습니다. 또한 매우 내구력 있는 스토리지에 로그 데이터를 저장할 수 있습니다. 자세한 내용은 Amazon CloudWatch Logs 사용 설명서를 참조하세요.
- Amazon EventBridge를 사용하여 AWS 서비스를 자동화하고 애플리케이션 가용성 문제 또는 리소스 변경과 같은 시스템 이벤트에 자동으로 대응할 수 있습니다. AWS 서비스의 이벤트는 거의 실시간으로 EventBridge로 전달됩니다. 원하는 이벤트만 표시하도록 간단한 규칙을 작성한 후 규칙과 일치하는 이벤트 발생 시 실행할 자동화 작업을 지정할 수 있습니다. 자세한 내용은 Amazon EventBridge 사용 설명서를 참조하세요.
- AWS CloudTrail는 AWS 계정에서 또는 계정을 대신하여 수행된 API 호출 및 관련 이벤트를 캡처하고 사용자가 지정한 Amazon S3 버킷에 로그 파일을 전송합니다. 호출된 사용자 및 계정 AWS, 호출이 수행된 원본 IP 주소, 호출이 발생한 시기를 식별할 수 있습니다. 자세한 내용은 AWS CloudTrail 사용 설명서를 참조하십시오.

Amazon CloudWatch를 사용한 Lookout for Vision 모니터링

원시 데이터를 수집하여 읽기 가능하며 실시간에 가까운 지표로 처리하는 CloudWatch를 통해 Lookout for Vision를 모니터링할 수 있습니다. 이러한 통계는 15개월간 보관되므로 기록 정보에 액세스하고 웹 애플리케이션 또는 서비스가 어떻게 실행되고 있는지 전체적으로 더 잘 파악할 수 있습니다. 특정 임곗값을 주시하다가 해당 임곗값이 충족될 때 알림을 전송하거나 조치를 취하도록 경보를 설정할 수도 있습니다. 자세한 내용은 Amazon CloudWatch 사용 설명서를 참조하세요.

Lookout for Vision 서비스는 `AWS/LookoutVision`네임스페이스에서 다음 지표를 보고합니다.

지표	설명
DetectedAnomalyCount	프로젝트에서 탐지된 이상 현상의 수 유효한 통계: Sum,Average 단위: 수
ProcessedImageCount	이상 탐지를 통해 실행된 총 이미지 수 유효한 통계: Sum,Average 단위: 수
InvalidImageCount	유효하지 않아 결과를 반환하지 못한 이미지 수 유효한 통계: Sum,Average 단위: 수
SuccessfulRequestCount	성공한 API 호출 수입니다. 유효한 통계: Sum,Average 단위: 수
ErrorCount	API 오류 수입니다. 유효한 통계: Sum,Average 단위: 수
ThrottledCount	제한으로 인한 API 오류 수 유효한 통계: Sum,Average 단위: 수
Time	Lookout for Vision에서 이상 탐지 결과를 계산하는 데 걸리는 시간 (밀리초) 유효한 통계: Data Samples,Average

지표	설명
	단위: Average 통계의 경우 밀리초, Data Samples 통계의 경우 개수
MinInferenceUnits	StartModel 요청 중에 지정된 최소 추론 단위 수입니다. 유효한 통계: Average 단위: 수
MaxInferenceUnits	StartModel 요청 중에 지정된 추론 단위의 최대 개수. 유효한 통계: Average 단위: 수
DesiredInferenceUnits	Lookout for Vision이 확대 또는 축소되는 추론 단위의 수입니다. 유효한 통계: Average 단위: 수
InServiceInferenceUnits	모델이 사용하는 추론 단위의 수. 유효한 통계: Average 평균 통계를 사용하여 사용된 인스턴스 수에 대한 1분 평균을 구하는 것이 좋습니다. 단위: 수

Lookout for Vision 지표에 다음 차원을 지원합니다.

차원	설명
ProjectName	프로젝트별로 지표를 분할하여 문제가 있거나 업데이트가 필요한 프로젝트를 확인할 수 있습니다.
ModelVersion	메트릭을 모델 버전별로 분할하여 어떤 모델에 문제가 있거나 업데이트가 필요한지 확인할 수 있습니다.

를 사용하여 Lookout for Vision API 호출 로깅 AWS CloudTrail

Amazon Lookout for Vision은 Lookout for Vision에서 사용자, 역할 또는 AWS CloudTrail서비스가 수행한 작업에 대한 레코드를 제공하는 AWS 서비스와 통합됩니다. CloudTrail은 Lookout for Vision에 대한 모든 API 호출을 이벤트로 캡처합니다. 캡처되는 호출에는 Lookout for Vision 콘솔로부터의 호출과 Lookout for Vision API 작업에 대한 코드 호출이 포함됩니다. 추적을 생성하면 Lookout for Vision 이벤트를 포함한 CloudTrail 이벤트를 지속적으로 Amazon S3 버킷에 배포할 수 있습니다. 트레일을 구성하지 않은 경우에도 CloudTrail 콘솔의 이벤트 기록에서 최신 이벤트를 볼 수 있습니다. CloudTrail에서 수집한 정보를 사용하여 Lookout for Vision에 수행된 요청, 요청이 수행된 IP 주소, 요청을 수행한 사람, 요청이 수행된 시간 및 추가 세부 정보를 확인할 수 있습니다.

CloudTrail에 대한 자세한 내용은 AWS CloudTrail 사용 설명서를 참조하세요.

CloudTrail에서 Lookout for Vision 정보

AWS 계정을 생성할 때 계정에서 CloudTrail이 활성화됩니다. Lookout for Vision에서 활동이 발생하면 해당 활동이 이벤트 기록의 다른 AWS 서비스 이벤트와 함께 CloudTrail 이벤트에 기록됩니다. AWS 계정에서 최근 이벤트를 보고 검색하고 다운로드할 수 있습니다. 자세한 정보는 CloudTrail 이벤트 기록을 사용하여 이벤트 보기를 참조하세요.

Lookout for Vision에 대한 이벤트를 포함하여 AWS 계정의 이벤트를 지속적으로 기록하려면 추적을 생성합니다. CloudTrail은 추적을 사용하여 Amazon S3 버킷으로 로그 파일을 전송할 수 있습니다. 콘솔에서 추적을 생성하면 기본적으로 모든 리전에 추적이 적용됩니다. 추적은 AWS 파티션의 모든 리전에서 이벤트를 로깅하고 지정한 Amazon S3 버킷으로 로그 파일을 전송합니다. 또한 CloudTrail 로그에서 수집된 이벤트 데이터를 추가로 분석하고 조치를 취하도록 다른 AWS 서비스를 구성할 수 있습니다. 자세한 내용은 다음 자료를 참조하세요.

- 트레일 생성 개요
- CloudTrail 지원 서비스 및 통합

- CloudTrail에서 Amazon SNS 알림 구성
- 여러 리전으로부터 CloudTrail 로그 파일 받기 및 여러 계정으로부터 CloudTrail 로그 파일 받기

모든 Lookout for Vision 작업은 CloudTrail에서 로깅되며 Lookout for Vision API 참조 문서에 문서화됩니다. 예를 들어 `CreateProject`, `DetectAnomalies`, `StartModel` 작업을 직접 호출하면 CloudTrail 로그 파일에 항목이 생성됩니다.

Amazon Lookout for Vision API 호출을 모니터하는 경우 다음 API에 대한 호출을 볼 수 있습니다.

- lookoutvision:StartTrialDetection
- lookoutvision:ListTrialDetection
- lookoutvision:DescribeTrialDetection

이러한 특수 호출은 Amazon Lookout for Vision에서 추적 탐지와 관련된 다양한 작업을 지원하는 데 사용됩니다. 자세한 내용은 평가판 확인 작업을 통한 모델 검증 단원을 참조하십시오.

모든 이벤트 또는 로그 항목에는 요청을 생성했던 사용자에 대한 정보가 포함됩니다. 자격 증명을 이용하면 다음을 쉽게 판단할 수 있습니다.

- 요청이 루트 또는 AWS Identity and Access Management 사용자 자격 증명으로 이루어졌는지 여부.
- 역할 또는 페더레이션 사용자에 대한 임시 보안 인증을 사용하여 요청이 생성되었는지 여부.
- 요청이 다른 AWS 서비스에서 이루어졌는지 여부입니다.

자세한 설명은 CloudTrail userIdentity 요소를 참조하세요.

Lookout for Vision 로그 파일 항목에 대한 이해

추적이란 지정한 Amazon S3 버킷에 이벤트를 로그 파일로 입력할 수 있게 하는 구성입니다. CloudTrail 로그 파일에는 하나 이상의 로그 항목이 포함될 수 있습니다. 이벤트는 모든 소스로부터의 단일 요청을 나타내며 요청 작업, 작업 날짜와 시간, 요청 파라미터 등에 대한 정보가 들어 있습니다. CloudTrail 로그 파일은 퍼블릭 API 직접 호출의 주문 스택 트레이스가 아니므로 특정 순서로 표시되지 않습니다.

다음은 `CreateDataset` 작업을 보여주는 CloudTrail 로그 항목이 나타낸 예시입니다.

```
{
  "eventVersion": "1.08",
```

```
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AROAYN4CJAYDEXAMPLE:user",
    "arn": "arn:aws:sts::123456789012:assumed-role/Admin/MyUser",
    "accountId": "123456789012",
    "accessKeyId": "ASIAYN4CJAYEXAMPLE",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AROAYN4CJAYDCGEXAMPLE",
        "arn": "arn:aws:iam::123456789012:role/Admin",
        "accountId": "123456789012",
        "userName": "Admin"
      },
      "webIdFederationData": {},
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2020-11-20T13:15:09Z"
      }
    }
  },
  "eventTime": "2020-11-20T13:15:43Z",
  "eventSource": "lookoutvision.amazonaws.com",
  "eventName": "CreateDataset",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "128.0.0.1",
  "userAgent": "aws-cli/3",
  "requestParameters": {
    "projectName": "P1",
    "datasetType": "train",
    "datasetSource": {
      "groundTruthManifest": {
        "s3Object": {
          "bucket": "myuser-bucketname",
          "key": "training.manifest"
        }
      }
    },
    "clientToken": "EXAMPLE-0526-47dd-a5d3-2ca975820a34"
  },
  "responseElements": {
    "status": "CREATE_IN_PROGRESS"
  },
  "requestID": "EXAMPLE-15e1-4bc9-be38-cda2537c75bf",
```

```
  "eventID": "EXAMPLE-c5e7-43e0-8449-8d9b87e15acb",
  "readOnly": false,
  "eventType": "AwsApiCall",
  "managementEvent": true,
  "eventCategory": "Management",
  "recipientAccountId": "123456789012"
}
```

다음을 사용하여 Amazon Lookout for Vision 리소스 만들기 AWS CloudFormation

Amazon Lookout for Vision은 AWS 리소스를 모델링하고 설정하는 데 도움이 되는 AWS CloudFormation서비스와 통합되어 리소스와 인프라를 생성하고 관리하는 데 소요되는 시간을 줄일 수 있습니다. 원하는 모든 AWS 리소스를 설명하는 템플릿을 생성하고 해당 리소스를 AWS CloudFormation 프로비저닝하고 구성합니다.

AWS CloudFormation 를 사용하여 Amazon Lookout for Vision 프로젝트를 프로비저닝하고 구성할 수 있습니다.

를 사용하면 템플릿을 재사용하여 Lookout for Vision 프로젝트를 일관되고 반복적으로 설정할 AWS CloudFormation수 있습니다. 프로젝트를 한 번만 설명하고 여러 AWS 계정 및 리전에서 동일한 프로젝트를 반복적으로 프로비저닝할 수 있습니다.

Lookout for Vision 및 AWS CloudFormation 템플릿

Lookout for Vision 및 관련 서비스에 대한 프로젝트를 프로비저닝하고 구성하려면 AWS CloudFormation 템플릿을 이해해야 합니다. 템플릿은 JSON 또는 YAML로 서식 지정된 텍스트 파일입니다. 이러한 템플릿은 AWS CloudFormation 스택에서 프로비저닝하려는 리소스를 설명합니다. JSON 또는 YAML에 익숙하지 않은 경우 Designer를 사용하여 AWS CloudFormation AWS CloudFormation 템플릿을 시작할 수 있습니다. 자세한 내용은 AWS CloudFormation 사용 설명서에서 AWS CloudFormation Designer이란 무엇입니까?를 참조하세요.

JSON 및 YAML 템플릿의 예를 비롯한 Lookout for Vision 프로젝트에 대한 자세한 내용은 LookoutVision 리소스 유형 참조를 참조하세요.

에 대해 자세히 알아보기 AWS CloudFormation

에 대해 자세히 알아보려면 다음 리소스를 AWS CloudFormation참조하세요.

- AWS CloudFormation
- AWS CloudFormation 사용 설명서
- AWS CloudFormation API Reference
- AWS CloudFormation 명령줄 인터페이스 사용 설명서

인터페이스 엔드포인트 (AWS PrivateLink)를 사용하여 Amazon Lookout for Vision에 액세스

AWS PrivateLink 를 사용하여 VPC와 Amazon Lookout for Vision 간에 프라이빗 연결을 생성할 수 있습니다. 인터넷 게이트웨이, NAT 디바이스, VPN 연결 또는 AWS Direct Connect 연결을 사용하지 않고 VPC에 있는 것처럼 Lookout for Vision에 액세스할 수 있습니다. VPC의 인스턴스에서 Lookout for Vision에 액세스하는 데 퍼블릭 IP 주소가 필요하지 않습니다.

AWS PrivateLink에서 제공되는 인터페이스 엔드포인트를 생성하여 이 프라이빗 연결을 설정합니다. 인터페이스 엔드포인트에 대해 사용 설정하는 각 서브넷에서 엔드포인트 네트워크 인터페이스를 생성합니다. 이는 Lookout for Vision로 향하는 트래픽의 진입점 역할을 하는 요청자 관리형 네트워크 인터페이스입니다.

자세한 내용은 AWS PrivateLink 가이드의를 AWS 서비스 통한 액세스를 AWS PrivateLink 참조하세요.

Lookout for Vision VPC 엔드포인트에 대한 고려 사항

Lookout for VisionS에 대한 인터페이스 엔드포인트를 설정하려면 먼저 AWS PrivateLink 가이드의 Considerations을 검토합니다.

Lookout for Vision은 인터페이스 엔드포인트를 통해 모든 API 작업에 대한 호출을 지원합니다.

Lookout for Vision에 대한 VPC 엔드포인트 정책이 지원됩니다. 기본적으로 인터페이스 엔드포인트를 통해 Lookout for Vision에 대한 전체 액세스가 허용됩니다. 또는 보안 그룹을 엔드포인트 네트워크 인터페이스와 연결하여 인터페이스 엔드포인트를 통해 Lookout for Vision으로 향하는 트래픽을 제어할 수 있습니다.

Lookout for Vision에 대한 인터페이스 VPC 엔드포인트 생성

Amazon VPC 콘솔 또는 AWS Command Line Interface ()를 사용하여 Lookout for Vision에 대한 인터페이스 엔드포인트를 생성할 수 있습니다AWS CLI. 자세한 내용은 AWS PrivateLink 설명서의 인터페이스 엔드포인트 생성을 참조하세요.

다음 서비스 이름을 사용하여 Lookout for Vision에 대한 인터페이스 엔드포인트를 생성합니다.

```
com.amazonaws.region.lookoutvision
```

인터페이스 엔드포인트에 프라이빗 DNS를 사용하도록 설정하는 경우, 리전에 대한 기본 DNS 이름을 사용하여 Lookout for Vision에 API 요청을 할 수 있습니다. 예: `lookoutvision.us-east-1.amazonaws.com`.

Lookout for Vision에 대한 VPC 엔드포인트 정책 생성

엔드포인트 정책은 인터페이스 엔드포인트에 연결할 수 있는 IAM 리소스입니다. 기본 엔드포인트 정책을 사용하면 인터페이스 엔드포인트를 통해 Lookout for Vision에 대한 전체 액세스를 허용합니다. VPC에서 Lookout for Vision에 허용되는 액세스를 제어하려면 사용자 지정 엔드포인트 정책을 인터페이스 엔드포인트에 연결합니다.

엔드포인트 정책은 다음 정보를 지정합니다.

- 작업을 수행할 수 있는 보안 주체 (AWS 계정, IAM 사용자, IAM 역할)
- 수행할 수 있는 작업.
- 작업을 수행할 수 있는 리소스.

자세한 내용은AWS PrivateLink 가이드의 엔드포인트 정책을 사용하여 서비스에 대한 액세스 제어를 참조하세요.

예제:Lookout for Vision 작업에 대한 VPC 엔드포인트 정책

다음은 Lookout for Vision에 대한 사용자 지정 엔드포인트 정책의 예제입니다. 이 정책을 인터페이스 엔드포인트에 연결하면 VPC 인터페이스 엔드포인트에 액세스할 수 있는 모든 사용자가 `myModel`프로젝트와 연결된`myProject` Lookout for Vision 모델에 대한 `DetectAnomalies` API 작업을 호출할 수 있도록 지정합니다.

```
{
   "Statement":[
      {
         "Principal":"*",
         "Effect":"Allow",
         "Action":[
            "lookoutvision:DetectAnomalies"
         ],
         "Resource": "arn:aws:lookoutvision:us-west-2:123456789012:model/myProject/
myModel"
      }
   ]
```

```
}
```

Amazon Lookout for Vision 할당량

다음 표에서는 Amazon Lookout for Vision의 현재 할당량을 설명합니다. 변경할 수 있는 할당량에 대한 자세한 내용은 AWS 서비스 할당량을 참조하십시오.

모델 할당량

다음 할당량은 모델의 테스트, 교육 및 기능에 적용됩니다.

리소스	할당량
지원되는 파일 형식	PNG 및 JPEG 이미지 형식
Amazon S3 버킷에 있는 이미지 파일의 최소 이미지 크기	64픽셀 x 64픽셀
Amazon S3 버킷에 있는 이미지 파일의 최대 이미지 크기	최대 크기는 4096픽셀 X 4096픽셀입니다. 크기가 작을수록 더 빠르게 업로드할 수 있습니다.
프로젝트에 사용되는 이미지 파일의 다양한 이미지 크기	데이터 세트의 모든 이미지는 크기가 같아야 합니다.
Amazon S3 버킷에 있는 이미지의 최대 파일 크기	8MB
레이블 부족	훈련 전에 이미지에 정상 또는 이상 레이블을 지정해야 합니다. 레이블이 없는 이미지는 훈련 중에 무시됩니다.
훈련 데이터 세트에서 정상으로 레이블이 지정된 최소 이미지 수	학습 데이터 세트와 테스트 데이터 세트가 분리된 프로젝트의 경우 10개, 데이터 세트가 하나뿐인 프로젝트의 경우 20개.
학습 데이터세트에서 예외 항목으로 레이블이 지정된 최소 이미지 수	훈련 및 테스트 데이터 세트가 분리된 프로젝트의 경우 0, 단일 데이터 세트로 구성된 프로젝트의 경우 10.
분류 훈련 데이터 세트의 최대 이미지 수	16,000

리소스	할당량
분류 테스트 데이터 세트의 최대 이미지 수	4,000
테스트 데이터 세트에서 정상으로 표시된 이미지의 최소 개수	10
테스트 데이터 세트에서 이상 징후로 분류된 최소 이미지 수	10
예외 위치 파악 훈련 데이터 세트의 최대 이미지 수	8000
예외 위치 파악 테스트 데이터 세트의 최대 이미지 수	800
시험 탐지 데이터 세트의 최대 이미지 수	2,000
최대 데이터 세트 매니페스트 파일 크기	1GB
모델 내 학습 데이터 세트의 최대 개수	1
최대 교육 시간	24시간
최대 테스트 시간	24시간
프로젝트의 최대 예외 항목 레이블 수	100
마스크 이미지의 최대 예외 항목 레이블 수	20
예외 항목 레이블의 최소 이미지 수. 계산하려면 이미지에 한 가지 유형의 예외 항목 레이블만 포함되어야 합니다.	단일 데이터 세트 프로젝트의 경우 20개. 훈련 데이터 세트와 테스트 데이터 세트가 분리된 프로젝트의 각 데이터 세트당 10개입니다.

Amazon Lookout for Vision에 대한 문서 기록

다음 표는 Amazon Lookout for Vision 개발자 안내서의 각 릴리스에서 변경된 중요 사항에 대해 설명합니다. 이 설명서에 대한 업데이트 알림을 받으려면 RSS 피드를 구독하면 됩니다.

- 최종 설명서 업데이트: 2023년 2월 20일

변경 사항	설명	날짜
지원 종료 알림	지원 종료 공지: 2025년 10월 31일에는 Amazon Lookout for Vision에 대한 지원을 중단할 AWS 예정입니다. 2025년 10월 31일 이후에는 Lookout for Vision 콘솔 또는 Lookout for Vision 리소스에 더 이상 액세스할 수 없습니다. 자세한 내용은이 블로그 게시물을 참조하세요.	2024년 10월 10일
Lambda 함수 예제 추가	AWS Lambda 함수에서 이상을 찾는 방법을 보여주는 예제입니다. 자세한 내용은 AWS Lambda 함수를 사용한 이상 찾기를 참조하십시오.	2023년 2월 20일
에 대한 IAM 지침 업데이트 AWS WAF	IAM 모범 사례에 따라 가이드가 업데이트되었습니다. 자세한 내용은 IAM의 보안 모범 사례를 참조하십시오.	2023년 2월 8일
데이터세트 내보내기 예제 추가	`ListDatasetEntries` 작업을 사용하여 Amazon Lookout for Vision 프로젝트에서 데이터 세트를 내보내는 방법을 보여주는 Python 예제가 추가되었습니다. 자세한 내용	2022년 12월 2일

	은 프로젝트에서 데이터 세트 내보내기 (SDK)를 참조하십시오.	
시작하기 주제 업데이트	예제 데이터 세트를 사용하여 이미지 세분화 모델을 만드는 방법을 보여주기 위해 시작하기를 업데이트했습니다. 자세한 내용은 Amazon Lookout for Vision 시작하기를 참조하세요.	2022년 10월 20일
문제 해결 주제 추가됨	모델 훈련 문제 해결 항목이 추가되었습니다. 자세한 내용은 모델 훈련 문제 해결을 참조하세요.	2022년 10월 17일
Amazon SageMaker AI Ground Truth 작업 사용에 대한 주제 추가	이미지에 직접 레이블을 지정하는 대신 Amazon SageMaker AI Ground Truth 작업을 사용하여 분류 및 이미지 세분화 모델을 위한 이미지에 레이블을 지정할 수 있습니다. 자세한 내용은 Amazon SageMaker AI Ground Truth 작업 사용을 참조하세요.	2022년 8월 19일
Amazon Lookout for Vision은 이제 이상 항목 현지화를 제공합니다.	이미지에서 다양한 유형의 이상 (예: 긁힘, 찌그러짐, 찢어짐)이 있는 위치, 이상 항목의 레이블 및 크기를 찾는 분할 모델을 생성할 수 있습니다. 자세한 내용은 학습된 Amazon Lookout for Vision 모델 실행을 참조하십시오.	2022년 8월 16일

Amazon Lookout for Vision은 이제 엣지 디바이스에서 CPU 추론을 제공합니다.	이제 Amazon Lookout for Vision 모델을 배포하여 GPU 가속기 없이도 CPU 하나만으로 Linux를 실행하는 x86 컴퓨팅 플랫폼에서 로컬에서 추론을 실행할 수 있습니다. 자세한 내용은 CPU 가속기를 참조하십시오.	2022년 8월 16일
Amazon Lookout for Vision은 이제 추론 단위를 자동으로 조정할 수 있습니다.	수요 급증에 대응하기 위해 Amazon Lookout for Vision은 이제 모델에서 사용하는 추론 단위의 수를 확장할 수 있습니다. 자세한 내용은 학습된 Amazon Lookout for Vision 모델 실행을 참조하십시오.	2022년 8월 16일
Java 예제 추가	Amazon Lookout for Vision 개발자 가이드에 이제 자바 예제가 포함되어 있습니다. 자세한 내용은 AWS SDK 시작하기를 참조하세요.	2022년 5월 2일
에지 디바이스에 모델 배포 정식 출시	이제에서 관리하는 엣지 디바이스에 대한 모델 배포 AWS IoT Greengrass Version 2 를 정식으로 사용할 수 있습니다. 자세한 내용은 엣지 디바이스에서 Amazon Lookout for Vision 모델 사용을 참조하십시오.	2022년 3월 14일
콘솔 버킷 정보가 업데이트됨	콘솔 버킷 콘텐츠 및 콘솔 버킷 생성에 대한 대체 접근 방식에 대한 정보가 업데이트되었습니다. 자세한 내용은 4단계: 콘솔 버킷 생성을 참조하십시오.	2022년 3월 7일

CSV 파일에서 매니페스트 파일 생성	이제 CSV 파일에서 분류 정보를 읽는 스크립트를 사용하여 매니페스트 파일을 간단하게 만들 수 있습니다. 자세한 내용은 CSV 파일로 매니페스트 파일 생성을 참조하세요.	2022년 2월 10일
에지 디바이스로의 모델 배포 릴리스 미리 보기	이제에서 관리하는 엣지 디바이스에 대한 모델 배포의 미리 보기 릴리스 AWS IoT Greengrass Version 2 를 사용할 수 있습니다. 자세한 내용은 엣지 디바이스에서 Amazon Lookout for Vision 모델 사용을 참조하십시오.	2021년 12월 7일
새로운 Python 및 Java 2 예제 추가	`DetectAnomalies` 를 사용하여 이미지를 분석하기 위한 Python 및 Java 2 예제를 추가했습니다. 자세한 내용은 이미지에서 이상 감지를 참조하세요.	2021년 9월 7일
새로운 AWS 관리형 정책이 추가되었습니다.	Amazon Lookout for Vision은 AWS 관리형 정책에 대한 지원을 추가합니다. 자세한 내용은 Amazon Lookout for Vision의 AWS 관리형 정책을 참조하십시오.	2021년 5월 11일
추론 단위 정보가 업데이트되었습니다.	추론 단위 및 요금 청구 방법을 설명하는 정보가 추가되었습니다. 자세한 내용은 학습된 Amazon Lookout for Vision 모델 실행을 참조하십시오.	2021년 3월 15일

Amazon Lookout for Vision이 정식 출시되었습니다.	이제 Amazon Lookout for Vision이 정식 출시되었습니다. Python 코드 예제가 모델 학습과 같은 비동기 작업을 처리하도록 업데이트되었습니다.	2021년 2월 17일
태그 지정 및 AWS CloudFormation 지원이 추가되었습니다.	이제 Amazon Lookout for Vision 모델에 태그를 지정하고를 사용하여 프로젝트를 생성할 수 있습니다 AWS CloudFormation. 자세한 내용은 모델 태깅 및 AWS CloudFormation을 사용하여 Amazon Lookout for Vision 프로젝트 생성을 참조하십시오.	2021년 1월 31일
새로운 특성 및 가이드	Amazon Lookout for Vision 서비스 Amazon Lookout for Vision 개발자 안내서가 최초 릴리스되었습니다.	2020년 12월 1일

AWS 용어집

최신 AWS 용어는 AWS 용어집 참조의 AWS 용어집을 참조하세요.