Cloud AutoML Vision

AutoML Vision API Tutorial

This tutorial demonstrates how to create a new model with your own set of training images, evaluate the results and predict the classification of test image using AutoML Vision.

The tutorial uses a dataset with images of five different kinds of flowers: sunflowers, tulips, daisy, roses and dandelions. It covers training a custom model, evaluating model performance, and classifying new images using the custom model.

Viewing Code Samples: Most of the code samples in this tutorial are taken from larger code files located in GitHub (https://github.com/). You can view and download the complete file from which a code sample is taken by clicking the "View on GitHub" button provided above a sample.

Prerequisites

Configure your project environment

1. Sign in (https://accounts.google.com/Login) to your Google Account.

   If you don't already have one, sign up for a new account (https://accounts.google.com/SignUp).

2. In the Cloud Console, on the project selector page, select or create a Google Cloud project.

★ **Note**: If you don't plan to keep the resources that you create in this procedure, create a project instead of selecting an existing project. After you finish these steps, you can delete the project, removing all resources associated with the project.

GO TO THE PROJECT SELECTOR PAGE (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/PROJECTSELECT

3. Make sure that billing is enabled for your Google Cloud project. Learn how to confirm billing is enabled for your project (https://cloud.google.com/billing/docs/how-to/modify-project).

4. Enable the AutoML Vision APIs.

   ENABLE THE APIS (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/FLOWS/ENABLEAPI?APIID=STORAGE-

5. Install the `gcloud` command line tool (https://cloud.google.com/sdk/downloads#interactive).
6. Follow the instructions to create a service account and download a key file (https://cloud.google.com/iam/docs/creating-managing-service-accounts#creating_a_service_account)
   .
7. Set the `GOOGLE_APPLICATION_CREDENTIALS` environment variable to the path to the service account key file that you downloaded when you created the service account. For example:

```
export GOOGLE_APPLICATION_CREDENTIALS=key-file
```

8. Add your new service account to the **AutoML Editor** IAM role with the following commands. Replace ***project-id*** with the name of your GCP project and replace ***service-account-name*** with the name of your new service account, for example `service-account1@myproject.iam.gserviceaccount.com`:

```
gcloud auth login
gcloud projects add-iam-policy-binding project-id \
 --member="user:your-userid@your-domain" \
 --role="roles/automl.admin"
gcloud projects add-iam-policy-binding project-id \
 --member=serviceAccount:service-account-name \
 --role="roles/automl.editor"
```

9. Allow the AutoML Vision service accounts to access your Google Cloud project resources:

```
gcloud projects add-iam-policy-binding project-id \
 --member="serviceAccount:custom-vision@appspot.gserviceaccount.com" \
 --role="roles/storage.admin"
```

10. Install the client library (https://cloud.google.com/vision/automl/docs/client-libraries).
11. Set the `PROJECT_ID` and `REGION_NAME` environment variables.

    Replace ***project-id*** with the Project ID (https://cloud.google.com/resource-manager/docs/creating-managing-projects#identifying_projects)
    of your Google Cloud Platform project. AutoML Vision currently requires the location `us-central1`.

```
export PROJECT_ID="project-id"
export REGION_NAME="us-central1"
```

12. Create a Google Cloud Storage bucket to store the documents that you will use to train your custom model.

    The bucket name must be in the format: `$PROJECT_ID-vcm`. The following command creates a storage bucket in the `us-central1` region named `$PROJECT_ID-vcm`.

    ```
    gsutil mb -p $PROJECT_ID -c regional -l $REGION_NAME gs://$PROJECT_ID-vcm/
    ```

13. Copy the publicly available dataset of flower images from `gs://cloud-ml-data/img/flower_photos/` into your Google Cloud Storage bucket.

    In your Cloud Shell session, enter:

    ```
    gsutil -m cp -R gs://cloud-ml-data/img/flower_photos/  gs://$PROJECT_ID-vcm/img
    ```

    The file copying takes about 20 minutes to complete.

14. The sample dataset contains a .csv file with the location and labels for each image. (See Preparing your training data (https://cloud.google.com/vision/automl/docs/prepare) for details about the required format). Update the .csv file to point to the files in your own bucket:

    ```
    gsutil cat gs://$PROJECT_ID-vcm/img/flower_photos/all_data.csv | sed "s:cloud-m
    ```

    Then copy the updated .csv file into your bucket:

    ```
    gsutil cp all_data.csv gs://$PROJECT_ID-vcm/csv/
    ```

Source code file locations

You can download the source code from the location provided below. After downloading, you can copy the source code into your AutoML Vision project folder.

PYTHON JAVA NODE.JS

The tutorial consists of these Python files (https://github.com/GoogleCloudPlatform/python-docs-samples/tree/automl-refactor/automl/cloud-client) :

- `vision_classification_create_dataset.py` (https://github.com/GoogleCloudPlatform/python-docs-samples/tree/automl-refactor/automl/cloud-client/vision_classification_create_dataset.py) – Includes functionality to create a dataset
- `import_dataset.py` (https://github.com/GoogleCloudPlatform/python-docs-samples/tree/automl-refactor/automl/cloud-client/import_dataset.py) – Includes functionality to import a dataset
- `vision_classification_create_model.py` (https://github.com/GoogleCloudPlatform/python-docs-samples/tree/automl-refactor/automl/cloud-client/vision_classification_create_model.py) – Includes functionality to create a model
- `list_model_evaluations.py` (https://github.com/GoogleCloudPlatform/python-docs-samples/tree/automl-refactor/automl/cloud-client/list_model_evaluations.py) – Includes functionality to list model evaluations
- `vision_classification_predict.py` (https://github.com/GoogleCloudPlatform/python-docs-samples/tree/automl-refactor/automl/cloud-client/vision_classification_predict.py) – Includes functionality related to prediction
- `delete_model.py` (https://github.com/GoogleCloudPlatform/python-docs-samples/tree/automl-refactor/automl/cloud-client/delete_model.py) - Include functionality to delete a model

Running the application

Step 1: Create the Flowers dataset

The first step in creating a custom model is to create an empty dataset that will eventually hold the training data for the model. When you create a dataset, you specify the type of classification you want your custom model to perform:

- MULTICLASS assigns a single label to each classified image

- MULTILABEL allows an image to be assigned multiple labels

This tutorial creates a dataset named `flowers` and uses MULTICLASS.

Copy the Code

PYTHON JAVA NODE.JS

PAGE=EDITOR&OPEN_IN_EDITOR=AUTOML/CLOUD-CLIENT/VISION_CLASSIFICATION_CREATE_DATASET.
-REFACTOR/AUTOML/CLOUD-CLIENT/VISION_CLASSIFICATION_CREATE_DATASET.PY) FEEDBACK (#)

```
from google.cloud import automl

# TODO(developer): Uncomment and set the following variables
# project_id = 'YOUR_PROJECT_ID'
# display_name = 'YOUR_DATASET_NAME'

client = automl.AutoMlClient()

# A resource that represents Google Cloud Platform location.
project_location = client.location_path(project_id, 'us-central1')
# Specify the classification type
# Types:
# MultiLabel: Multiple labels are allowed for one example.
# MultiClass: At most one label is allowed per example.
metadata = automl.types.ImageClassificationDatasetMetadata(
    classification_type=automl.enums.ClassificationType.MULTILABEL)
dataset = automl.types.Dataset(
    display_name=display_name,
    image_classification_dataset_metadata=metadata)

# Create a dataset with the dataset metadata in the region.
response = client.create_dataset(project_location, dataset)

created_dataset = response.result()

# Display the dataset information
print(u'Dataset name: {}'.format(created_dataset.name))
print(u'Dataset id: {}'.format(created_dataset.name.split("/")[-1]))
```

Request

Run the `create_dataset` function to create an empty dataset. You must modify the following lines of code:

- Set the `project_id` to your PROJECT_ID
- Set the `display_name` for the dataset (`flowers`)
- Change `MULTILABEL` to `MULTICLASS

PYTHON JAVA NODE.JS

```
python vision_classification_create_dataset.py
```

Response

The response includes the details of the newly created dataset, including the Dataset ID that you'll use to reference the dataset in future requests. We recommend that you set an environment variable `DATASET_ID` to the returned Dataset ID value.

```
Dataset name: projects/216065747626/locations/us-central1/datasets/ICN7372141011136
Dataset id: ICN7372141011130533778
Dataset display name: flowers
Image classification dataset specification:
       classification_type: MULTICLASS
Dataset example count: 0
Dataset create time:
       seconds: 1530251987
       nanos: 216586000
```

Step 2: Import images into the dataset

The next step is to populate the dataset with training images labeled using the target labels.

The `import_data` function interface takes as input a .csv file that lists the locations of all training images and the proper label for each one. (See Prepare your data (https://cloud.google.com/vision/automl/docs/prepare) for details about the required format.) For this tutorial, we will use the labeled images that you copied into your Google Cloud Storage bucket, which are listed in `gs://$PROJECT_ID-vcm/csv/all_data.csv`.

Copy the Code

PYTHON JAVA NODE.JS

YTHON-DOCS-SAMPLES&PAGE=EDITOR&OPEN_IN_EDITOR=AUTOML/CLOUD-CLIENT/IMPORT_DATASET.
SAMPLES/BLOB/AUTOML-REFACTOR/AUTOML/CLOUD-CLIENT/IMPORT_DATASET.PY) FEEDBACK (#)

```
from google.cloud import automl

# TODO(developer): Uncomment and set the following variables
# project_id = 'YOUR_PROJECT_ID'
# dataset_id = 'YOUR_DATASET_ID'
# path = 'gs://BUCKET_ID/path_to_training_data.csv'

client = automl.AutoMlClient()
# Get the full path of the dataset.
dataset_full_id = client.dataset_path(
    project_id, 'us-central1', dataset_id)
# Get the multiple Google Cloud Storage URIs
input_uris = path.split(',')
gcs_source = automl.types.GcsSource(
    input_uris=input_uris)
input_config = automl.types.InputConfig(
    gcs_source=gcs_source)
# Import data from the input URI
response = client.import_data(dataset_full_id, input_config)

print('Processing import...')
print(u'Data imported. {}'.format(response.result()))
```

Request

Run the `import_data` function to import the training content. The first piece of code to change is the Dataset ID from the previous step and the second is the URI of `all_data.csv`. You must modify the following lines of code:

- Set the `project_id` to your ***PROJECT_ID***
- Set the `dataset_id` for the dataset (from the output of the previous step)
- Set the `path` which is the URI of the (`gs://YOUR_PROJECT_ID-vcm/csv/all_data.csv`)

- python import_dataset.py {Python}
- mvn compile exec:java -Dexec.mainClass="com.example.automl.ImportDataset" {Java}
- node import_dataset.js {Node.js}

Response

```
Processing import...
Dataset imported.
```

Step 3: Create (train) the model

Now that you have a dataset of labeled training images, you can train a new model.

Copy the Code

PYTHON JAVA NODE.JS

&PAGE=EDITOR&OPEN_IN_EDITOR=AUTOML/CLOUD-CLIENT/VISION_CLASSIFICATION_CREATE_MODEL.

ML-REFACTOR/AUTOML/CLOUD-CLIENT/VISION_CLASSIFICATION_CREATE_MODEL.PY) FEEDBACK (#)

```
from google.cloud import automl

# TODO(developer): Uncomment and set the following variables
# project_id = 'YOUR_PROJECT_ID'
# dataset_id = 'YOUR_DATASET_ID'
# display_name = 'YOUR_MODEL_NAME'

client = automl.AutoMlClient()

# A resource that represents Google Cloud Platform location.
project_location = client.location_path(project_id, 'us-central1')
# Leave model unset to use the default base model provided by Google
metadata = automl.types.ImageClassificationModelMetadata()
model = automl.types.Model(
    display_name=display_name,
    dataset_id=dataset_id,
    image_classification_model_metadata=metadata)
```

```
# Create a model with the model metadata in the region.
response = client.create_model(project_location, model)

print(u'Training operation name: {}'.format(response.operation.name))
print('Training started...')
```

Request

Call the `create_model` function to create a model. The Dataset ID is from the previous steps. You must modify the following lines of code:

- Set the `project_id` to your ***PROJECT_ID***
- Set the `dataset_id` for the dataset (from the output of the previous step)
- Set the `display_name` for your model (`flowers_model`)
- python vision_classification_create_model.py {Python}
- mvn compile exec:java -Dexec.mainClass="com.example.automl.VisionClassificationCreateModel" {Java}
- node vision_classification_create_model.js {Node.js}

Response

The `create_model` function starts a training operation and prints the operation name. Training happens asynchronously and can take a while to complete, so you can use the operation ID to check training status (https://cloud.google.com/vision/automl/docs/models#get-operation). When training is complete, `create_model` returns the Model ID. As with the Dataset ID, you might want to set an environment variable `MODEL_ID` to the returned Model ID value.

```
Training operation name: projects/216065747626/locations/us-central1/operations/ICN3
Training started...
Model name: projects/216065747626/locations/us-central1/models/ICN768334683937180326
Model id: ICN7683346839371803263
Model display name: flowers_model
Image classification model metadata:
Training budget: 1
Training cost: 1
Stop reason:
Base model id:
Model create time:
```

```
        seconds: 1529649600
        nanos: 966000000
Model deployment state: deployed
```

Step 4: Evaluate the model

After training, you can evaluate your model's readiness by reviewing its precision (https://developers.google.com/machine-learning/crash-course/glossary#precision), recall (https://developers.google.com/machine-learning/crash-course/glossary#recall), and F1 score (https://en.wikipedia.org/wiki/F1_score).

The `display_evaluation` function takes the Model ID as a parameter.

Copy the Code

PYTHON JAVA NODE.JS

)CS-SAMPLES&PAGE=EDITOR&OPEN_IN_EDITOR=AUTOML/CLOUD-CLIENT/LIST_MODEL_EVALUATIONS.

/BLOB/AUTOML-REFACTOR/AUTOML/CLOUD-CLIENT/LIST_MODEL_EVALUATIONS.PY) **FEEDBACK** (#)

```
from google.cloud import automl

# TODO(developer): Uncomment and set the following variables
# project_id = 'YOUR_PROJECT_ID'
# model_id = 'YOUR_MODEL_ID'

client = automl.AutoMlClient()
# Get the full path of the model.
model_full_id = client.model_path(project_id, 'us-central1', model_id)

print('List of model evaluations:')
for evaluation in client.list_model_evaluations(model_full_id, ''):
    print(u'Model evaluation name: {}'.format(evaluation.name))
    print(
        u'Model annotation spec id: {}'.format(
            evaluation.annotation_spec_id))
    print(u'Create Time:')
    print(u'\tseconds: {}'.format(evaluation.create_time.seconds))
    print(u'\tnanos: {}'.format(evaluation.create_time.nanos / 1e9))
    print(u'Evaluation example count: {}'.format(
```

```
        evaluation.evaluated_example_count))
    print('Translation model evaluation metrics: {}'.format(
          evaluation.translation_evaluation_metrics))
```

Request

Make a request to display the overall evaluation performance of the model by executing the following request. You must modify the following lines of code:

- Set the `project_id` to your ***PROJECT_ID***
- Set the `model_id` to your model's id
- python list_model_evaluations.py {Python}
- mvn compile exec:java -Dexec.mainClass="com.example.automl.ListModelEvaluations" {Java}
- node list_model_evaluations.js {Node.js}

Response

If the precision and recall scores are too low, you can strengthen the training dataset and re-train your model. For more information, see Evaluating models (https://cloud.google.com/vision/automl/docs/evaluate).

```
Precision and recall are based on a score threshold of 0.5
Model Precision: 96.3%
Model Recall: 95.7%
Model F1 score: 96.0%
Model Precision@1: 96.33%
Model Recall@1: 95.74%
Model F1 score@1: 96.04%
```

Step 5: Use a model to make a prediction

When your custom model meets your quality standards, you can use it to classify new flower images.

Copy the Code

PYTHON JAVA NODE.JS

MPLES&PAGE=EDITOR&OPEN_IN_EDITOR=AUTOML/CLOUD-CLIENT/VISION_CLASSIFICATION_PREDICT.

/AUTOML-REFACTOR/AUTOML/CLOUD-CLIENT/VISION_CLASSIFICATION_PREDICT.PY) FEEDBACK (#)

```
from google.cloud import automl

# TODO(developer): Uncomment and set the following variables
# project_id = 'YOUR_PROJECT_ID'
# model_id = 'YOUR_MODEL_ID'
# file_path = 'path_to_local_file.jpg'

prediction_client = automl.PredictionServiceClient()

# Get the full path of the model.
model_full_id = prediction_client.model_path(
    project_id, 'us-central1', model_id
)

# Read the file.
with open(file_path, 'rb') as content_file:
    content = content_file.read()

image = automl.types.Image(image_bytes=content)
payload = automl.types.ExamplePayload(image=image)

# params is additional domain-specific parameters.
# score_threshold is used to filter the result
params = {'score_threshold': '0.8'}

response = prediction_client.predict(model_full_id, payload, params)
print('Prediction results:')
for result in response.payload:
    print(u'Predicted class name: {}'.format(result.display_name))
    print(u'Predicted class score: {}'.format(result.classification.score))
```

Request

For the `predict` function you must modify the following lines of code:

- Set the `project_id` to your ***PROJECT_ID***

- Set the `model_id` to your model's id
- Set the `file_path` to the downloaded file ("resources/test.png")
- python vision_classification_predict.py {Python}
- mvn compile exec:java -Dexec.mainClass="com.example.automl.VisionClassificationPredict" {Java}
- node vision_classification_predict.js {Node.js}

Response

The function returns the classification score for how well the image matches each category, exceeding the stated confidence threshold of 0.7.

```
Prediction results:
Predicted class name: dandelion
Predicted class score: 0.9702693223953247
```

Step 6: Delete the model

When you are done using this sample model, you can delete it permanently. You will no longer be able to use the model for prediction.

Copy the Code

PYTHON JAVA NODE.JS

PYTHON-DOCS-SAMPLES&PAGE=EDITOR&OPEN_IN_EDITOR=AUTOML/CLOUD-CLIENT/DELETE_MODEL.

S-SAMPLES/BLOB/AUTOML-REFACTOR/AUTOML/CLOUD-CLIENT/DELETE_MODEL.PY) **FEEDBACK** (#)

```
from google.cloud import automl

# TODO(developer): Uncomment and set the following variables
# project_id = 'YOUR_PROJECT_ID'
# model_id = 'YOUR_MODEL_ID'

client = automl.AutoMlClient()
# Get the full path of the model.
```

```
model_full_id = client.model_path(project_id, 'us-central1', model_id)
response = client.delete_model(model_full_id)

print(u'Model deleted. {}'.format(response.result()))
```

Request

Make a request with operation type `delete_model` to delete a model you created you must modify the following lines of code:

- Set the `project_id` to your ***PROJECT_ID***
- Set the `model_id` to your model's id
- python delete_model.py {Python}
- mvn compile exec:java -Dexec.mainClass="com.example.automl.DeleteModel" {Java}
- node delete_model.js {Node.js}

Response

```
Model deleted.
```