Google Cloud

How to classify images with TensorFlow using Google Cloud Machine Learning and Cloud Dataflow

December 16, 2016

Slaven Bilac
Software Engineer

For specialized image-classification use cases, using Google Cloud Dataflow and Google Cloud Machine Learning makes it easy to train and implement machine-learning models.

Google Cloud Vision API is a popular service that allows users to classify images into categories, appropriate for multiple common use cases across several industries. For those users whose category requirements map to the pre-built, pre-trained machine-learning model reflected in the API, this approach is ideal. However, other users have more specialized requirements — for example, to identify specific products and soft goods in mobile-phone photos, or to detect nuanced differences between particular animal species in wildlife photography. For them, it can be more efficient to train and serve a new image model using Google Cloud Machine Learning (Cloud ML), the managed service for building and running machine-learning models at scale using the open source TensorFlow deep-learning framework.

In this post, we'll build a simple model in Cloud ML using a small set of labeled flower images. This dataset has been selected for ease of explanation only; we've successfully used the same implementation for several proprietary datasets covering cases like interior-design classification (e.g., carpet vs. hardwood floor) and animated-character classification. The code can be found here and can easily be adapted to run on different datasets.

In addition to image files, we've provided a CSV file (all_data.csv) containing the image URIs and labels. We randomly split this data into two files, train_set.csv and eval_set.csv, with 90% data for training and 10% for eval, respectively.

```
gs://cloud-ml-data/img/flower_photos/dandelion/17388674711_6dca8a2e8b_n.jpg,dandelion
gs://cloud-ml-data/img/flower_photos/sunflowers/9555824387_32b151e9b0_m.jpg,sunflowers
gs://cloud-ml-data/img/flower_photos/daisy/14523675369_97c31d0b5b.jpg,daisy
gs://cloud-ml-data/img/flower_photos/roses/512578026_f6e6f2ad26.jpg,roses
gs://cloud-ml-data/img/flower_photos/tulips/497305666_b5d4348826_n.jpg,tulips...
```

We also need a text file containing all the labels (dict.txt), which is used to sequentially map labels to internally used IDs. In this case, `daisy` would become ID `0` and `tulips` would become `4`. If the label isn't in the file, it will be ignored from preprocessing and training.

```
daisy
dandelion
roses
sunflowers
tulips
```

Because we only have a small set of images in this sample dataset, it would be hard to build an accurate classification model from scratch. Instead, we'll use an approach called transfer learning, taking a pre-trained model called Inception, and using it to extract image features that we'll then use to train a new classifier.

Before following along below, be sure to set up your project and environment first.

Preprocessing

We start with a set of labeled images in a Google Cloud Storage bucket and preprocess them to extract the image features from the bottleneck layer (typically the penultimate layer) of the Inception network. Although processing images in this manner can be reasonably expensive, each image can be processed independently and in parallel, making this task a great candidate for Cloud Dataflow.

We process each image to produce its feature representation (also known as an embedding) in the form of a k-dimensional vector of floats (in our case 2,048 dimensions). The preprocessing includes converting the image format, resizing images, and running the converted image through a pre-trained model to get the embeddings. Final output will be written in directory specified by `--output_path`.

```
# Assign appropriate values.
PROJECT=$(gcloud config list project --format "value(core.project)")
JOB_ID="flowers_${USER}_$(date +%Y%m%d_%H%M%S)"
BUCKET="gs://${PROJECT}-ml"
GCS_PATH="${BUCKET}/${USER}/${JOB_ID}"
DICT_FILE=gs://cloud-ml-data/img/flower_photos/dict.txt
# Preprocess the eval set.
python trainer/preprocess.py \
  --input_dict "$DICT_FILE" \
  --input_path "gs://cloud-ml-data/img/flower_photos/eval_set.csv" \
  --output_path "${GCS_PATH}/preproc/eval" \
  --cloud
# Preprocess the train set.
python trainer/preprocess.py \
  --input_dict "$DICT_FILE" \
  --input_path "gs://cloud-ml-data/img/flower_photos/train_set.csv"
\
  --output_path "${GCS_PATH}/preproc/train" \
  --cloud
```

The flowers dataset is about 3,600 images in size and can be processed on a single machine. For larger sets, though, the preprocessing becomes the bottleneck and parallelization can lead to a huge increase in throughput. To measure the benefit of parallelizing preprocessing on Google Cloud, we ran the above preprocessing on 1 million sample images from the Open Image Dataset. We found that while it takes several days to preprocess 1 million images locally, it takes less than 2 hours on the cloud when we use 100 workers with four cores each!

Modeling

Once we've preprocessed data, we can then train a simple classifier. The network will comprise a single fully-connected layer with RELU activations and with one output for each label in the dictionary to replace the original output layer. Final output is computed using the softmax function. Note that in training stages we're using the dropout technique, which randomly ignores a subset of input weights to prevent over-fitting to the training dataset.

Figure 1: We retrain only a small section of the original inception graph.

Training

The training can be run using the following command:

```
# Submit training job.
gcloud ml-engine jobs submit training "$JOB_ID" \
  --module-name trainer.task \
  --package-path trainer \
  --staging-bucket "$BUCKET" \
  --region us-central1 \
  -- \
  --output_path "${GCS_PATH}/training" \
  --eval_data_paths "${GCS_PATH}/preproc/eval*" \
  --train_data_paths "${GCS_PATH}/preproc/train*"
# Monitor training logs.
gcloud ml-engine jobs stream-logs "$JOB_ID"
```

We can monitor progress of the training using tensorboard. As you can see from the image on the right, for the flowers dataset we reach accuracy of ~90% on our eval set

Figure 2: Loss and accuracy graphs in Tensorboard.

Prediction

For prediction, we don't want to separate the image preprocessing and inference into two separate steps because we need to perform both in sequence for every image. Instead, we create a single TensorFlow graph that produces the image embedding and does the classification using the trained model in one step.

Deploying and using the model

We deploy the model based on these instructions. After a few minutes, the model will become available for use via Cloud ML Prediction API.

```
MODEL_NAME=flowers  # for example
VERSION_NAME=v1  # for example
gcloud ml-engine models create ${MODEL_NAME} \
  --regions us-central1
gcloud ml-engine versions create \
  --origin ${GCS_PATH}/training/model/ \
  --model ${MODEL_NAME} \
  ${VERSION_NAME}
gcloud ml-engine versions set-default --model ${MODEL_NAME}
${VERSION_NAME}
```

The format of the request is JSON with image content encoded using base64 encoding.

```
# Copy the image to local disk.
gsutil cp gs://cloud-ml-
data/img/flower_photos/tulips/4520577328_a94c11e806_n.jpg flower.jpg
# Create request message in json format.
python -c 'import base64, sys, json; img =
base64.b64encode(open(sys.argv[1], "rb").read()); print
json.dumps({"key":"0", "image_bytes": {"b64": img}})' flower.jpg &>
request.json
# Call prediction service API to get classifications
gcloud ml-engine predict --model ${MODEL_NAME} --json-instances
request.json
```

Alternatively, you can use a command-line tool to encode multiple images into a single JSON request.

---

tulips!By Ginny is licenced under CC BY 2.0

Using an example image from the eval set, we get the following response when running the above predict command:

```
predictions:
 - key: '0'
   prediction: 4
   scores:
   - 8.11998e-09
   - 2.64907e-08
   - 1.10307e-06
   - 3.69488e-11
   - 0.999999
   - 3.35913e-09
```

It correctly identifies the most likely category as `4`, which is the ID of `tulip`, with score of `0.99`. Very nice!

Next steps

In short, if you have relatively specialized image-classification category requirements not reflected in Cloud Vision API, using Cloud Dataflow and Cloud ML can make it easy to train classification models using labeled images and deploy them for online classification.

If you have any feedback on the design or documentation of this example or have other issues, please report it on GitHub; pull request and contributions are also welcome. We're always working to improve and simplify our products, so stay tuned for new improvements in the near future!

See also:

- Cloud Dataflow documentation
- Cloud ML documentation
- Cloud Vision API documentation

Posted in Google Cloud—AI & Machine Learning

Related articles

Inside Google Cloud

What’s new with Google Cloud

By Google Cloud Content & Editorial • 2-minute read

Data Analytics

Run faster and more cost-effective Dataproc jobs

By Christian Yarros • 11-minute read

Developers & Practitioners

Opinary generates recommendations faster on Cloud Run

By Doreen Sacker • 6-minute read

Data Analytics

Accelerate integrated Salesforce insights with Google Cloud Cortex Framework

By Danielle Brannon • 2-minute read