Report of a Machine Learning project to build an image classifier, using Apache Beam, and Google Cloud Machine Learning

Veronica Tuffrey
City, University of London

1 INTRODUCTION

1.1 Overview: Machine Learning (ML) is "*about making computers modify or adapt their actions … so that these actions get more accurate, where accuracy is measured by how well the chosen actions reflect the correct ones*".[1] As computers and algorithms have become more powerful, an increasingly common application of ML is image classification, for which the ML technique used is Convolutional Neural Networks. This document reports an example of the application of neural networks, using Apache Beam (to prepare the data) and Google Cloud ML (using TensorFlow, an open-source software development kit created by Google), to build and run models for two image classification tasks.

1.2 Datasets: The following two datasets were used, both freely available on the web:

- *"Flowers":* In this dataset there are around 3600 images of five categories of flowers (daisy; dandelion; roses; sunflowers and tulips). Images were downloaded from `gs://cloud-samples-data/ml-engine/flowers/` and code files from `https://github.com/GoogleCloudPlatform/cloudml-samples`[2]
- *Coastline:* 10,000 images of coastline are available from `gs://tamucc_coastline/esi_images`. The images were acquired by drones and classified into categories based on their potential to suffer flood damage, using characteristics including coarseness of sand; fresh-, salt-, or brackish-water; sheltered or exposed; herbaceous or woody vegetation, existence of man-made structures etc.[3] Information about the dataset was obtained from `https://codelabs.developers.google.com/codelabs/scd-coastline/index.html?index=..%2F..cloud-quest-scientific-data#0`

1.3 Pipeline, including pre-training of network:[4] This is a summary of the pipeline applied -

- Preprocessing: Labelled images in a Google Cloud Storage bucket are pre-processed to extract the image features from the bottleneck layer of the Inception network.[5] This converts the image format, resizes the images, and runs the converted image through a pre-trained model to get the embeddings.[6]
- Modelling and training: The network comprises a single fully-connected layer with RELU (rectified linear unit) activation function, and one output for each label to replace the original output layer. Final output is computed using the softmax function (this assigns decimal probabilities to each class that sum to one). During training, "dropout" is used, by which a randomly selected subset of input weights is ignored to prevent over-fitting to the training dataset. Only a small set of the original inception graph is retrained.
- Prediction: A single TensorFlow graph is created, that produces the image embedding and does the classification using the trained model in one step.

2 METHODS

- A GCP account was created, and a Google Cloud Platform (GCP) project was automatically set up at that time. The AI Platform ("Cloud Machine Learning Engine") and Compute Engine APIs were enabled. A "bucket" was created via the shell, using the name of the project "Plenary-hybrid-234522".
- The Google Cloud Shell was activated, and source code for the Flowers dataset was obtained by cloning from https://github.com/GoogleCloudPlatform/cloudml-samples
- Apache Beam and the necessary package for Image preprocessing (pillow) were installed, while TensorFlow was already installed in the Cloud Shell. Code in the file sample.sh from the Github site included commands to set up environment variables, pre-process the data, train the model and use the model for prediction. This code was written into a Python script, and run using the Google Cloud Shell from within the cloudml-samples-master > flowers directory.
- A new bucket was created, and the Coastline dataset copied to it. The .csv files with the labels were downloaded from the bucket to the local PC, split into train/test files (70%/30%[7]) using python code, and the two new .csv files uploaded back to the bucket. The code was run several times to test and modify.

[1] Marsland S. (2011) *Machine learning: an algorithmic perspective*. Florida, US: Chapman and Hall/CRC.

[2] Files downloaded contained the image URIs and labels: all_data.csv (complete dataset); train_set.csv (90% data); eval_set.csv (10% data). Also, a text file contained all the labels (dict.txt), which is used to sequentially map labels to internally used IDs.

[3] The Harte Research Institute for Gulf of Mexico Studies, part of the Texas A & M University, Corpus Christi (TAMUCC) commissioned the image acquisition, and granted permission for the images to be used for educational purposes.

[4] Information in this section is derived from https://cloud.google.com/blog/products/gcp/how-to-classify-images-with-tensorflow-using-google-cloud-machine-learning-and-cloud-dataflow

[5] The inception network is a pre-trained model from which image features are extracted. It was used to train a new classifier.

[6] Embeddings are the feature representation in the form of a 2048 dimensional vector of floats.

[7] 70%/ 30% was chosen following advice that a 90%/10% split, equivalent to the flowers dataset, would take too long to process.

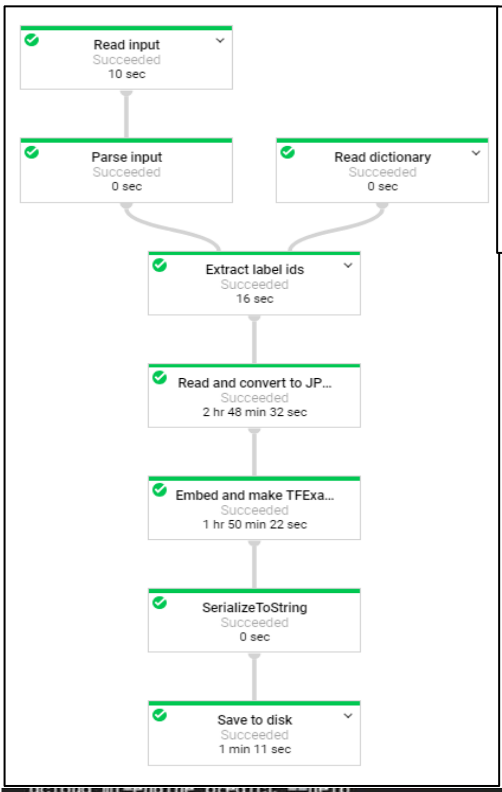
- The Flowers code was modified to use with the Coastline dataset, by increasing number of labels (under trainingInput) and changing source address of the image for prediction (Section 1.2). The code was tested and run, and metrics for Coastline data were compared with those for the Flowers dataset*.
- The Coastline code was modified to explore the effect of varied server/cluster configurations*.
- Effect of dropout was explored*. For this a command was added to the trainingInput section of the code, and a selection of values tested from 0.2 – 0.99 for Flowers, and 0.1 – 0.99 for Coastline.

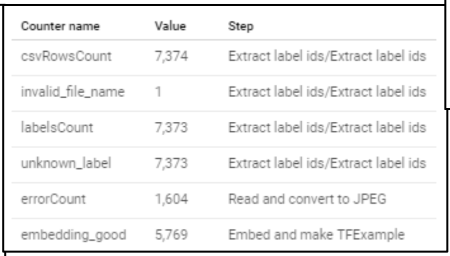
* Visualisations of metrics as they evolved during training were obtained using TensorBoard. Accuracy, loss and time taken for pre-processing were recorded from logs accessed via “Dataflow” on the Google console. For training, success /failure and time taken were accessed from “AI Platform”. The number of TFrecords output from training were obtained for each job from the folders “preproc” under “Storage” on the GCP.[8]

3 FINDINGS AND DISCUSSION

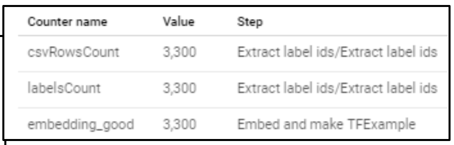
<u>3.1 Application of source code for Flowers and Coastline Datasets</u>: Figure 1 shows an example job summary obtained via “Dataflow” for processing the Coastline training data. The total elapsed time for this process was 40.8 minutes, while the times shown in the boxes are the sum of times at that particular step across all threads in all workers, and these enable comparison of speed across the steps. Here the slowest steps are reading and converting the files to JPEG, and embedding and making the tensorflow example.

Figure 2 shows that for the Coastline data there were a large proportion of errors (around one fifth) for reading and converting to JPEG, which meant embeddings were not obtained for many of the images. This may be due some of the .jpg suffixes for image names in the CSV file being written in a different case. For the Flowers dataset there was no such issue (Figure 3).





Counter name	Value	Step
csvRowsCount	7,374	Extract label ids/Extract label ids
invalid_file_name	1	Extract label ids/Extract label ids
labelsCount	7,373	Extract label ids/Extract label ids
unknown_label	7,373	Extract label ids/Extract label ids
errorCount	1,604	Read and convert to JPEG
embedding_good	5,769	Embed and make TFExample



Counter name	Value	Step
csvRowsCount	3,300	Extract label ids/Extract label ids
labelsCount	3,300	Extract label ids/Extract label ids
embedding_good	3,300	Embed and make TFExample

Figure 1 (far left): Example counters from job summary, Coastline (#222627)

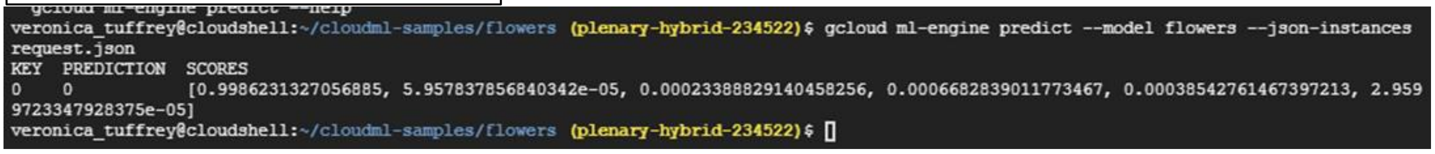
Figure 2 (near left): Example graph from job summary, Coastline (#222627)

Figure 3 (above right): Example counter from job summary, Flowers (#211907)

Training time was similar for the two datasets (mean c. 12 minutes), however times taken for pre-processing differed. For Flowers (N = 20) mean was 7.2 minutes for eval and 12.8 for train. For Coastline (n = 7) mean was 22.5 minutes for eval and 41.4 for train[9].

The command for predictions did not run successfully from within the code file, but predictions and probabilities were obtained by running the command separately in the shell afterwards. An example of output is shown in Figure 4 below, where the daisy image was correctly classified into class 1 with a probability of 0.999, with probabilities for the other four classes being very low.

Figure 4: Example of output from prediction command for an image of a daisy



```
 gcloud ml-engine predict --help
veronica_tuffrey@cloudshell:~/cloudml-samples/flowers (plenary-hybrid-234522)$ gcloud ml-engine predict --model flowers --json-instances
request.json
KEY  PREDICTION  SCORES
0    0           [0.9986231327056885, 5.957837856840342e-05, 0.00023388829140458256, 0.0006682839011773467, 0.00038542761467397213, 2.959
9723347928375e-05]
veronica_tuffrey@cloudshell:~/cloudml-samples/flowers (plenary-hybrid-234522)$ []
```

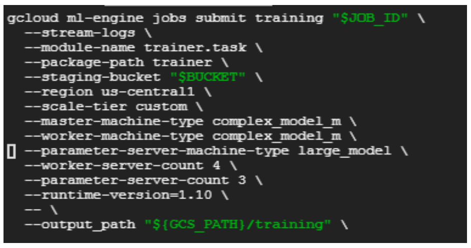
<u>3.2 Effect of different server/cluster configurations for Coastline Dataset:</u> Starting with the set of predefined cluster specifications known as “scale tiers” available within the AI platform, varied configurations were trialled. The configurations are summarised in Table 1, which shows that several of the trials encountered fatal errors, including the attempt to include a GPU (graphics processing unit). An attempt at choosing a custom tier and increasing the specification of machine types from those used in STANDARD_1 also failed (see Figure 5 for

[8] For Flowers the number of. tfrecord.gz files varied from 9 to 12 for preproc eval, and 31 to 45 for preproc train. For Coastline the ranges were 51 to 57 for preproc eval, and 56 to 64 for preproc train. Since within each of these files, there can be any number of records (corresponding to individual images), it is not clear how to interpret these numbers and/or whether they are useful. Presumably the reason the values are more similar between eval and train for Coastline is that the proportions in the datasets were more similar (30 v.70%) than for Flowers (10 v. 90%).

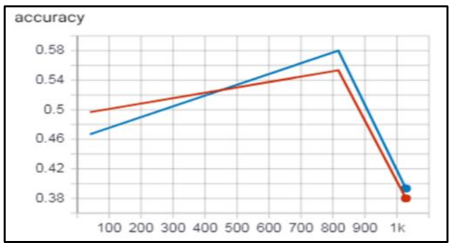
[9] This is using the default configuration of one master machine type n1-standard-4, 8 Cloud TPU workers, and no parameter servers.

code), as did the attempt to trial the BASIC configuration of one standard Master. For the trial of PREMIUM_1, the accuracies reached peak values at step 790 and then dropped (Figure 6) while for STANDARD_1 the eval accuracy stopped improving at around step 300. Advice on explanations for these observations will be sought.

Figure 5 (left): Code extract for training using custom configuration . For Figs 6 and 7, Red is eval set and Blue is train set.
Figure 6 (centre) Output from PREMIUM_1 (job #222911). Figure 7 (right) Output from STANDARD_1 (job #191326)



```
gcloud ml-engine jobs submit training "$JOB_ID" \
  --stream-logs \
  --module-name trainer.task \
  --package-path trainer \
  --staging-bucket "$BUCKET" \
  --region us-central1 \
  --scale-tier custom \
  --master-machine-type complex_model_m \
  --worker-machine-type complex_model_m \
▯ --parameter-server-machine-type large_model \
  --worker-server-count 4 \
  --parameter-server-count 3 \
  --runtime-version=1.10 \
  -- \
  --output_path "${GCS_PATH}/training" \
```



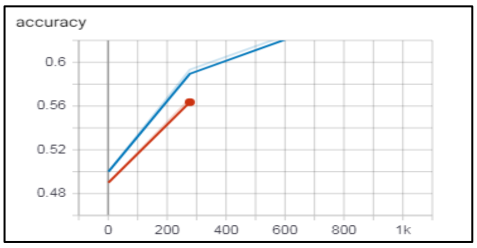
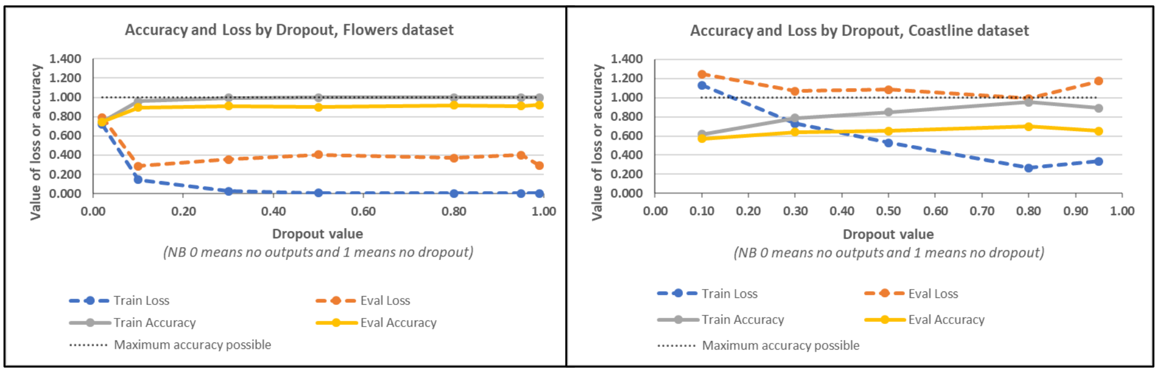


Table 1: Values of Accuracy and Loss from trial of configurations on Coastline dataset

AI Platform	Master		Workers		Parameter servers			*Duration*	All values on final step (1000)				
Scale-tier name	Number	Type	Number	Type	Number	Type		*Mins*	LossTrain	AccTrain	LossEval	AccEval	
BASIC	1	n1-standard-4	0		0								FAILED Weds 17th
STANDARD_1	1	n1-highcpu-8	4	n1-highcpu-8	3	n1-standard-4		*8.00*	1.079	0.663	1.237	0.613	Tues 16th OK
PREMIUM_1	1	n1-highcpu-16	19	n1-highcpu-16	11	n1-highmem-8	1000 steps	*6.73*	1.720	0.393	1.786	0.380	Tues 16th slow
	1	n1-highcpu-16	19	n1-highcpu-16	11	n1-highmem-8	3000 steps	*7.80*	1.249	0.587	1.355	0.573	Weds 17th OK
BASIC_GPU	0		1	n1-standard-8	0								FAILED Weds 17th
				with one k80 GPU									
BASIC_TPU	1	n1-standard-4	8	Cloud TPU	0			12.48	0.269	0.953	0.992	0.700	Many examples
CUSTOM_1	1	n1-highcpu-16	4	n1-highcpu-16	3	n1-highmem-8							FAILED

3.3 Effect of dropout value for Flowers and Coastline Datasets:

Figures 8 and 9: Metrics for Flowers dataset (left) and Coastline dataset (right) by dropout value



As described above in Section 1.3, dropout is a regularisation technique applied to prevent overfitting. If the value is too low, the technique has minimal effect, while if too high, the network under-learns. Figures 2 and 3 show that effect of dropout differed between datasets.[10] For Flowers, small values of D (very few units turning on during training) leads to underfitting since both the training and evaluation errors are high. As D increases, the loss quickly goes down and stays constant for all other values, even when values are almost equal to 1. For Coastline, trends in accuracy and loss values indicate optimal dropout value is around 0.8. Training duration showed no trend to change as D increased.[11]

One of the drawbacks of dropout is that it increases training time. A network with dropout typically takes 2-3 times longer to train than a standard neural network of the same architecture.[12] The value of this multiple was not explored in the current study because dropout was used for all training runs (D in the original code was 0.5).

4 CONCLUSIONS: The project revealed that powerful analytical tools are available in the cloud to private users, and much valuable material exists on the web to support novice users (provided both by Google and individual bloggers). The main lesson learnt relates to the amounts of time needed for running the models, even though the datasets were not huge. Also Google cloud is not 100% reliable, it can stop working mid-run (related to issues affecting all Google functions, not just the Google console). Useful future work would be to explore the effect of using GPUs for training. Since the images and relevant files have been stored, and code developed, it would be relatively straightforward to do this once the author has identified how to access GPUs.

[10] Dropout value" in Figures 8 and 9 is the probability of retaining a unit.
[11] Times taken (minutes) were 11.4; 12.7; 10.9; 11.5; 11.3; 12.7 and 12.1 for values of dropout between 0.02 and 0.99 for the Flowers dataset, and 11.6; 12.6; 12.2; 12.5 and 11.2 for values of dropout between 0.1 and 0.95 for the Coastline dataset.
[12] Srivastava. et al. (2014) Dropout: A Simple Way to Prevent Neural Networks from Overfitting *Journal of Machine Learning Research* 15 pp.1929-58