# INM432 Big Data Coursework – GCP ML Image Classification Experiments

**Axil Sudra**

## Introduction

The rapid development in Big Data technologies has made the field of predictive analytics more accessible to data experiments with high volume, variety and velocity [1]. Large scale image classification has been a significant area in which big data technologies have been used in the past. In this report, we discuss the process of using Google Cloud ML Platform (GCP) to build customized image classification (deep learning) models from two diverse datasets; a flowers dataset and a coastline dataset. Furthermore, we explore the effects of using different cluster configurations with (and without) GPUs on the training process for our deep learning models and investigate the result of varying the dropout rate. Throughout the experiments mentioned previously, we compare accuracy, loss and computational performance metrics for both datasets.

## GCP ML Process

### Preliminaries

In order to use GCP's services a few preliminary steps were executed. We began by creating a new project with a unique 'project id'. Note that this can be manually set by the user or auto-generated by GCP. Once the project had been initiated, we were required to enable various APIs and services that were to be used when completing data pre-processing, training and model deployment. These included the 'Dataproc Clusters API', the 'Dataflow API' and the 'AI Platform Cloud Machine Learning Engine API'. The results of our data experiments needed to be stored for analysis later so we created a bucket using GCP's 'Storage Service'. A bucket can be created directly from the 'Storage Service' interface or through GCP's shell environment with commands.

### Flowers Dataset

Following the preliminaries above, we were now setup to use GCP to complete an image classification task using the flowers dataset in the cloud shell environment. We begin by defining a few environment variables which will be re-used in the experiment. To execute the task at hand, we required the relevant python files for pre-processing and training; these are copied from GitHub and stored in the cloud shell's home directory under the cloned folder 'cloudml-samples'. As we were conducting our experiment on the flowers dataset, we moved into the 'flowers' folder to access the relevant python files. To successfully complete data pre-processing and training we installed two fundamental packages; Apache Beam [2] was used to pre-process the flowers dataset through a dataflow pipeline and Pillow [3] was used for imaging.

Given that we had successfully set up our cloud environment, we ran pre-processing on the flowers dataset. Note that the dataset was split into an evaluation and training set; thus, each set was pre-processed separately. The progress of each pre-processing task was displayed in the GCP's dataflow page. The dataflow jobs for the evaluation and training set lasted 7 minutes and 24 seconds and 12 minutes and 20 seconds respectively. After pre-processing, we trained a model on our data; for this we used GCP's AI platform which used a method called transfer learning [4]. Training the model for the flowers dataset lasted 20 minutes and 53 seconds. Lastly, we saved the training results by creating and deploying a model in GCP's cloud ML engine (AI platform).

By successfully executing the steps above, we could use our saved model to make predictions on flower images. The loss, accuracy and prediction results are discussed in the (primary) analysis section below.

### Coastline Dataset

The process for completing an image classification task using the coastline dataset was significantly similar to that of the flowers dataset. As the data was different to the flowers dataset, we retrieved all relevant files for coastline and saved to our bucket. Note that the coastline dataset was split into evaluation and training set manually using a local python file and re-uploaded to our bucket. Furthermore, we were required create a coastline folder in the cloud shell directory and copy the 'trainer' folder (which was situated in the flowers folder and contained relevant python files for pre-processing and training) to this new directory location. Once these steps had been completed, pre-processing and training were identical to the flowers dataset (although the 'eval_set_size' and 'label_count' parameters of the task.py file were set to 2343 and 18 respectively when training).

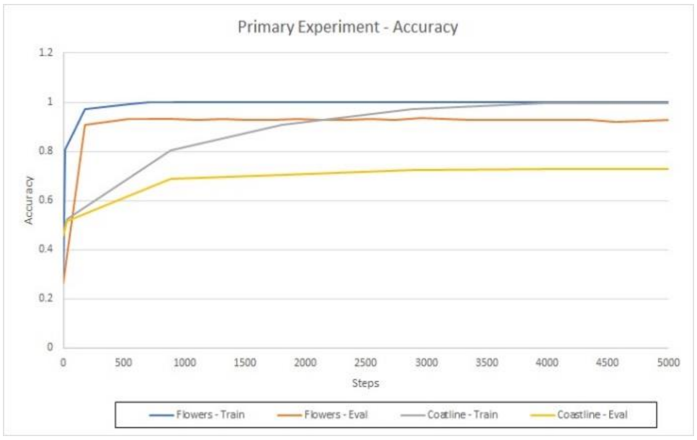
The pre-processing dataflow jobs for the evaluation and training set lasted 22 minutes and 28 seconds and 45 minutes and 59 seconds respectively. The significant increase in pre-processing time in comparison to the flowers dataset was expected as this dataset is much larger. Training a model using the coastline dataset lasted 9 minutes and 25 seconds; this was quite surprising as we anticipated the model to take significantly longer to train.

Once we had completed the steps above we were ready to use our saved model to make predictions on coastline images. As with the flowers dataset, the loss, accuracy and prediction results for coastline are discussed in the (primary) analysis section below.

## **Analysis of Experimental Results**

### Primary Experiment Results

The primary experiment was a simple implementation and comparison of the flowers and coastline dataset. We used TensorBoard [5] to analyse the accuracy and loss of the evaluation and training set of both flowers and coastline dataset; these results are presented in Figure 1.



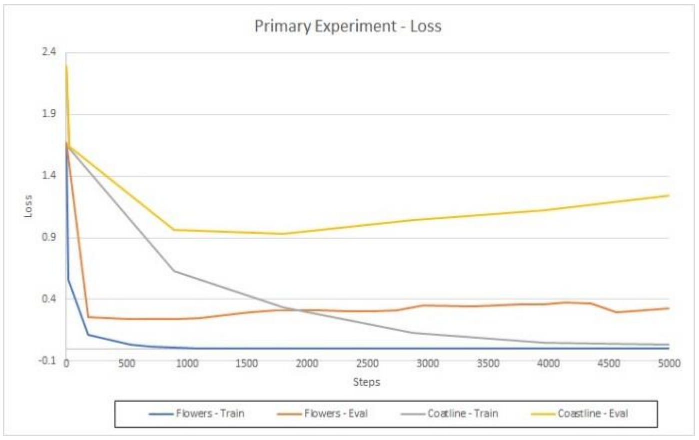


Figure 1: Left – Evaluation and Training Set Accuracy | Right – Evaluation and Training Set Loss

The flowers training set converged to 100.00% accuracy in a rapid fashion (≈700 steps) compared to the coastline training set which achieved a maximum accuracy of 99.80% in the 5000 steps. In terms of training loss, the flowers data attained a minimum rate of 1.20% whilst the coastline data attained a minimum rate of 3.30%. The evaluation accuracy and loss rate for flowers were recorded at 92.70% and 33.40% respectively whilst for coastline they were recorded at 72.70% and 124.30% (which was quite unexpected) respectively.

### Server/Cluster Configuration Experiment Results

The second experiment involved investigating the effects of training on different server/cluster configurations (with and without GPUs) for the flowers and coastline dataset. In the primary experiment we used computing resources situated in central US; for this experiment we changed the region of the computing resources to the east of the US. Table 1 shows our training results that use different server/cluster configurations (with and without GPUs).

| Dataset | Server/Cluster Configuration Description | Training Time | Accuracy % Eval/Train | Loss % Eval/Train |
|---|---|---|---|---|
| Flowers | Master Machine: Standard GPU (1 NVIDIA Tesla K80 GPU) | 24:00 m/s | 92.00% / 100.00% | 40.20% / 0.00% |
| Flowers | Master Machine: Complex Model M GPU (4 NVIDIA Tesla K80 GPUs) | 21:20 m/s | 93.30% / 100.00% | 38.40% / 0.00% |
| Flowers | Master Machine: N1 High CPU 16 | 18:53 m/s | 93.30% / 100.00% | 36.50% / 0.00% |
| Flowers | Master Machine: N1 High CPU 32 | 19:27 m/s | 92.30% / 100.00% | 38.00% / 0.00% |
| Flowers | Master Machine: N1 High CPU 16<br>Master Accelerator: 4 NVIDIA Tesla K80 GPUs<br>Worker Machine: 6 N1 High CPU 16s<br>Parameter Server Machine: 3 N1 Highmem 8s | 10:57 m/s | 91.00% / 99.70% | 28.20% / 6.60% |
| Flowers | Master Machine: N1 High CPU 32<br>Master Accelerator: 4 NVIDIA Tesla K80 GPUs<br>Worker Machine: 6 N1 High CPU 32s<br>Parameter Server Machine: 3 N1 Highmem 16s | 11:37 m/s | 92.00% / 99.70% | 34.70% / 0.90% |
| Coastline | Master Machine: Standard GPU (1 NVIDIA Tesla K80 GPU) | 9:11 m/s | 72.20% / 99.90% | 127.20% / 1.80% |
| Coastline | Master Machine: Complex Model M GPU (4 NVIDIA Tesla K80 GPUs) | 12:48 m/s | 71.80% / 99.90% | 126.20% / 2.10% |
| Coastline | Master Machine: N1 High CPU 16 | 8:43 m/s | 72.60% / 99.60^ | 128.40% / 2.80% |
| Coastline | Master Machine: N1 High CPU 32 | 8:17 m/s | 71.90% / 100.00% | 121.10% / 2.50% |
| Coastline | Master Machine: N1 High CPU 16<br>Master Accelerator: 4 NVIDIA Tesla K80 GPUs<br>Worker Machine: 6 N1 High CPU 16s<br>Parameter Server Machine: 3 N1 Highmem 8s | 11:42 m/s | 63.40% / 98.60% | 115.20% / 6.60% |
| Coastline | Master Machine: N1 High CPU 32<br>Master Accelerator: 4 NVIDIA Tesla K80 GPUs<br>Worker Machine: 6 N1 High CPU 32s<br>Parameter Server Machine: 3 N1 Highmem 16s | 9:57 m/s | 60.70% / 62.60% | 127.50% / 109.80% |

Table 1: Server/Cluster Configuration Description and Performance Metrics

Adding GPUs to existing CPUs should result in significant improvements in training factors (or general-purpose computing tasks [6]). The effect of using a GPU-enabled AI platform machine (in this case the Complex Model M GPU (4 NVIDIA Tesla K80 GPUs) improved the evaluation accuracy the most for the flowers dataset, although training time increased from the primary experiment. When we used compute engine machines with attached GPUs for the flowers dataset, training time was decreased drastically although at the (slight) expense of the accuracy and loss rates. In comparison, the coastline dataset did not really benefit from using GPU-enabled AI platform machines and different compute engine machines (with and without attached GPUs); the training times only decreased to a certain extent and the accuracy and loss rates did not improve (they substantially deteriorated in some cases). We observed through the training log files that GPU usage was 0% when using machines with GPUs which could explain the indifference in performance metrics.

Dropout Experiment Results

In the final experiment we investigated the result of altering the dropout; we used dropout of 0.1, 0.5 and 0.9. Figure 2 displays the effects on accuracy and loss rate on the flowers and coastline dataset. Note that dropout is the process of dropping out neurons from a neural network during the training phase to prevent overfitting [7].

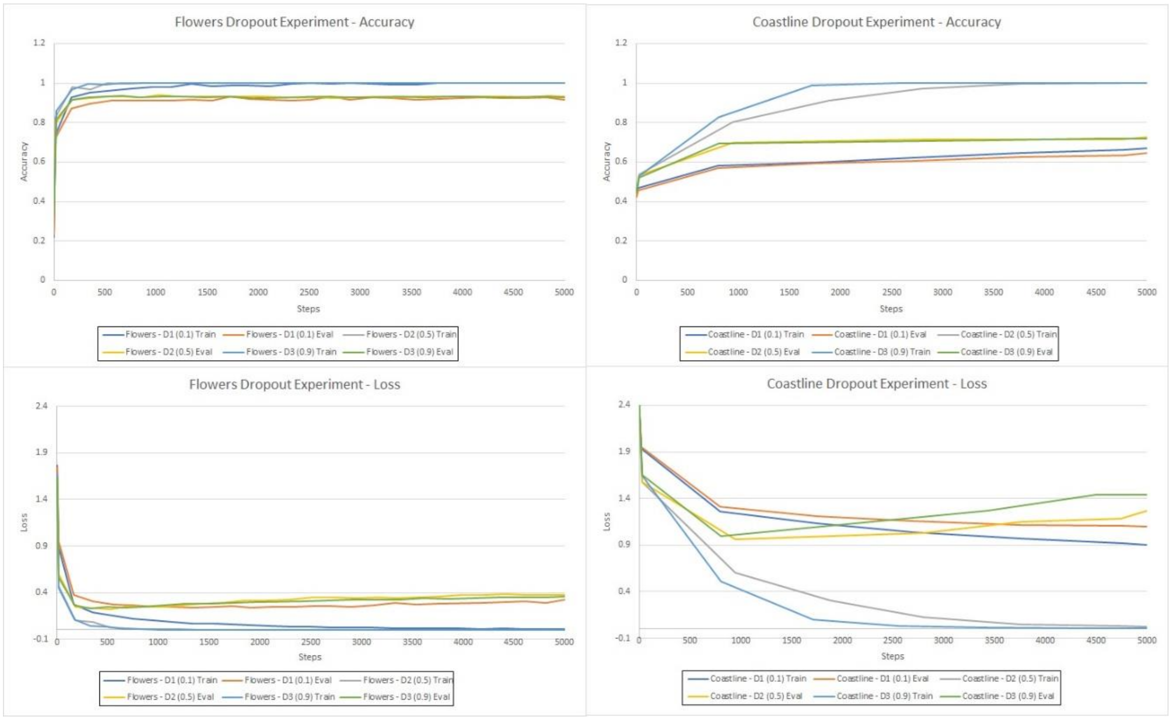


Figure 2: (Training and Evaluation Set) Accuracy and Loss | Left – Flowers Dataset | Right – Coastline Dataset

We found that a dropout of 0.9 produced an accuracy equally as good as a dropout of 0.5 on both datasets. A dropout of 0.1 seemed to overfit the flowers evaluation set more than the coastline evaluation set (i.e. the flowers dropout of 0.9 performed well during training and poorly during evaluation, whilst both sets performed disappointingly for coastline data).

## Conclusion

Throughout our experiments, the flowers dataset produced better results than the coastline dataset; this could be due to the complexity of the images contained in the coastline dataset. We found that the use of GPUs considerably reduced training times for the flowers dataset but increased for the coastline dataset which was very strange (the coastline dataset was significantly larger than that of flowers and so should have had a greater effect in using GPUs). The final experiment demonstrated that higher dropouts tend to produce higher accuracy and loss rates. However, using a larger dropout proved not to speed-up the training process (at least for the coastline dataset) which again was very peculiar. Possible future work could include experimenting with more powerful server/cluster configurations, different network architectures and a much larger dataset on GCP.

## References

[1] Sagiroglu, S. and Sinanc, D. (2013) ‘Big data: A review’ *2013 International Conference on Collaboration Technologies and Systems (CTS)*, pp. 42-27. Available at: https://ieeexplore.ieee.org/abstract/document/6567202/authors#authors (Accessed: April 2019).
[2] *Apache Beam Documetation* (2019). Available at: https://beam.apache.org/documentation/ (Accessed: April 2019).
[3] *Pillow* (2019). Available at: https://pillow.readthedocs.io/en/stable/ (Accessed: April 2019).
[4] Hussain, M. *et al.* (2018) ‘A study on CNN Transfer Learning for Image Classification’ *Advances in Computational Intelligence Systems*, pp. 192-202. Available at: https://link.springer.com/chapter/10.1007/978-3-319-97982-3_16 (Accessed: April 2019).
[5] *TensorBoard: Visualizing Learning* (2019). Available at: https://www.tensorflow.org/guide/summaries_and_tensorboard (Accessed: April 2019).
[6] Owens, D.J. *et al.* (2008) ‘GPU Computing’ *Proceedings of the IEEE*, 96(5), pp. 879-899. Available at: https://ieeexplore.ieee.org/document/4490127 (Accessed: April 2019).
[7] *Dropout in (Deep) Machine learning* (2016). Available at: https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5 (Accessed: April 2019).