Project Report: Flower Classification with TPUs

CS 795/895: Practical Machine Learning (Spring, 2020)

Apurva Modi
Department of Computer Science
Old Dominion University
amodi002@odu.edu

***Abstract*— Kaggle's Flower Classification with TPUs presents a challenge to classify 104 types of flowers using the TPU resources**

Keywords— TPU, Flowers, Classification, Transfer Learning, TensorFlow, Kaggle, Image Data, Plants, macrofscore. Multiclass, Machine Learning.

I. Introduction

A. What is TPU?

A tensor processing unit (TPU) is an AI accelerator application-specific integrated circuit (ASIC) developed by Google specifically for neural network machine learning.

TPUs read data directly from Google Cloud Storage (GCS). This Kaggle utility will copy the dataset to a GCS bucket co-located with the TPU. If you have multiple datasets attached to the notebook, you can pass the name of a specific dataset to the get_gcs_path function. The name of the dataset is the name of the directory it is mounted in. Use !ls /kaggle/input/ to list attached datasets.

B. The Challenge

It's difficult to fathom just how vast and diverse our natural world is. There are over 5,000 species of mammals, 10,000 species of birds, 30,000 species of fish – and astonishingly, over 400,000 different types of flowers.

In this competition, we're challenged to build a machine learning model that identifies the type of flowers in a dataset of images (for simplicity, we're sticking to just over 100 types).

II. Dataset Description

In this competition we're classifying 104 types of flowers based on their images drawn from five different public datasets. Some classes are very narrow, containing only a particular sub-type of flower (e.g. pink primroses) while other classes contain many sub-types (e.g. wild roses).

This dataset was assembled / cleaned up / reformatted from "five different public datasets" as the description states. The list is in the "Rules" section: "collectively sourced and sampled from 5 datasets: ImageNet, Oxford 102 Category Flowers, TF Flowers, Open Images, and iNaturalist." [1]

These are all public datasets, although the creators did spend a considerable amount of time making sure the result was nice and clean. The goal was to have a new good quality dataset to play with TPUs.

A. Files

This competition is different in that images are provided in TFRecord format. The TFRecord format is a container format frequently used in Tensorflow to group and shard data data files for optimal training performace. Each file contains the id, label (the class of the sample, for training data) and img (the actual pixels in array form) information for many images. The getting started notebook has information and notes on how to load and use TPU resources! Additional information is available in the TPU documentation.

- train/*.tfrec - training samples, including labels.
- val/*.tfrec - pre-split training samples w/ labels intended to help with checking your model's performance on TPU. The split was stratified across labels.
- test/*.tfrec - samples without labels - you'll be predicting what classes of flowers these fall into.
- sample_submission.csv - a sample submission file in the correct format
    - id - a unique ID for each sample.
    - label - (in training data) the class of flower represented by the sample

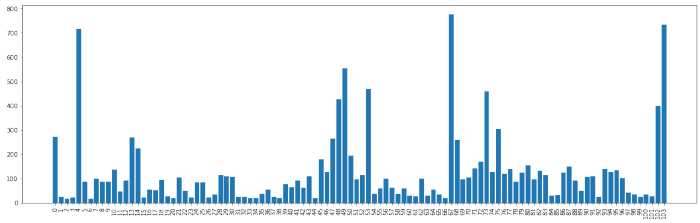

Figure 1 Sample per classes for Validation dataset.

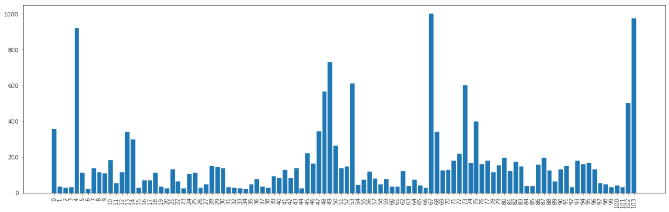

Figure 2 Sample per classes for Training dataset.

- Training Dataset: 12753 samples
- Validation Dataset: 3712 samples
- Testing Dataset: 7382 samples

Based on Figure 1 and Figure 2 we can say that the training data and the validation dataset are distributed equally in terms of classes but some of the classes have too much of sample and some of other classes have very few sample for both the testing as well as validation dataset.

III. METHODOLOGY

Running the Starter Notebook.

The starter notebook that was provided with all the sample code needed to perform the TPU operations. Loading the data, converting the data into batches for displaying the training images, the validation dataset images. Also, how to use the data augmentation, how to define a model for training and validation and how to use transfer learning. So, I went through each of the function in order to understand what is happening, how TPU scaling the performance of the training and reducing the training time.

Evaluation

Submissions are evaluated on macro F1 score. F1 score, also known as balanced F-score or F-measure. The F1 score can be interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. The relative contribution of precision and recall to the F1 score are equal. The formula for the F1 score is:

F1 = 2 * (precision * recall) / (precision + recall)

Precision = TP / (TP+FP)
Recall = TP / (TP+FN

Macro F1 score: Calculate metrics for each label and find their unweighted mean. This does not take label imbalance into account.

A. Initial Run – Vgg16 Model

I selected Vgg16 as my baseline model to perform transfer learning, just to access the potential of the model and what can be done to improve the baseline model. Another reason to use this model was that this model is less dense and was supposed to train faster with comparatively less computation, memory and recourses.

```
58892288/58889256 [==============================] - 1s 0us/step
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 6, 6, 512)         14714688
_________________________________________________________________
global_average_pooling2d (Gl (None, 512)               0
_________________________________________________________________
dense (Dense)                (None, 104)               53352
=================================================================
Total params: 14,768,040
Trainable params: 53,352
Non-trainable params: 14,714,688
_________________________________________________________________
```

Figure 3 Vgg16 Model summary

I initially trained the model without fine-tuning the Vgg16 model and till 80 epochs.

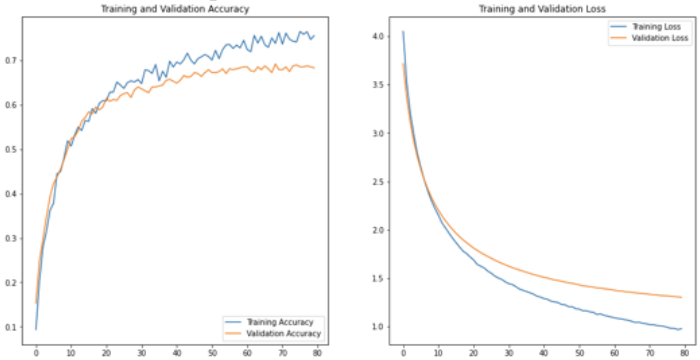

Figure 4 Vgg16 Initial run with 80 epochs.

I made use of the below mentioned method to plot the confusion matrix plot.
cmat_norm = (cmat.T / cmat.sum(axis=1)).T
I made use of the transformed matrix normalized the confusion matrix for better visualization.

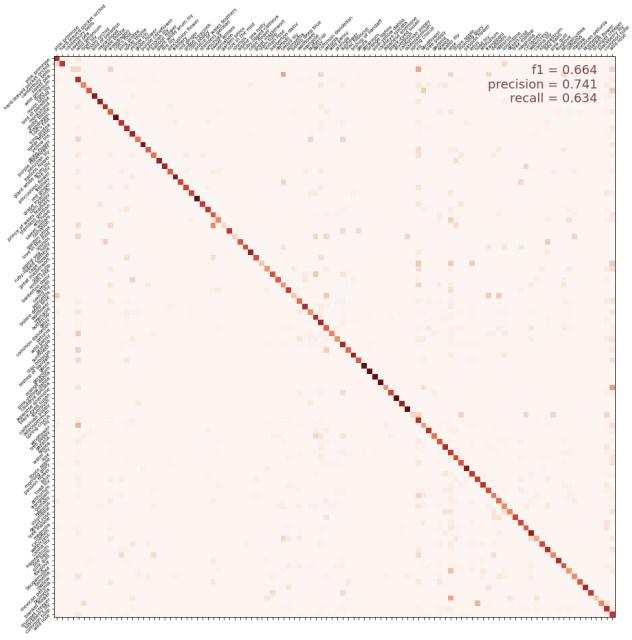

Figure 5 Vgg16 initial run confusion matrix

B. *Use of Vgg16 Model – Data Augmentation*

To Improve my baseline model, I made use of the data augmentation techniques such as rotate left to right, rotate up to down, added saturation and contrast. So that I could add noise in each of my training image through a augmentation generator. For data augmentation I thanks the **dataset.prefetch(AUTO)** statement in the calling function, this function prefetch next batch while training (autotune prefetch buffer size), which happens essentially for free on TPU. Data pipeline code is executed on the "CPU" part of the TPU while the TPU itself is computing gradients.

Figure 6 Training batch of images with data augmentation

With the applied augmentation to the training data I reduced the epochs from the baseline model to 25, perform the training and obtained the below mentioned confusion matrix with precision, recall and F1 score.

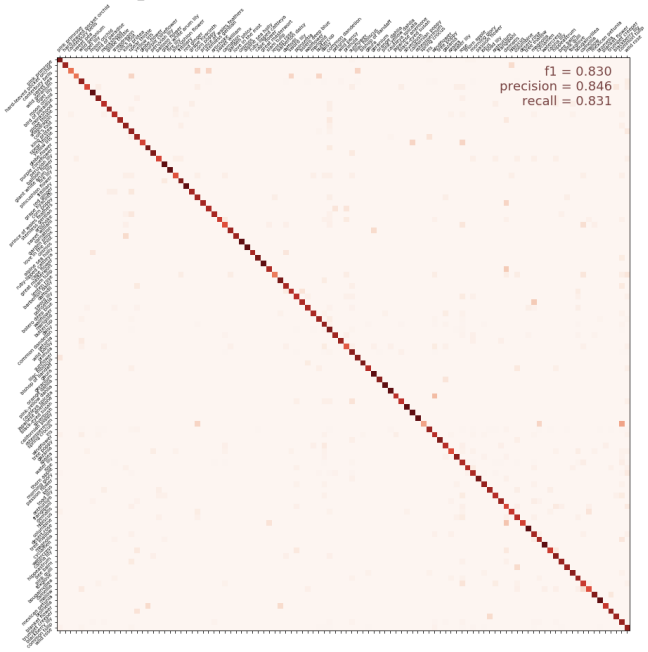

Figure 7 confusion matrix on validation set for improved vgg16 which include data augmentation as well

The model showed a significant increase in the performance and F1 score but didn't not work well on limiting the mispredictions, this may be because the model is not dense, also Vgg16 has so many weight parameters, the models is very heavy, 550 MB + of weight size which also means long inference time

C. *Why not just make the **Vgg16 model deeper**?*

- Heavier model
- More training times
- **Vanishing gradient** problem but, deeper networks can have higher test error and generalize lesser if done simply like what VGG does.

D. *EfficientNet - Sate of the Art Model*

Because of the limitation of the Vgg16 model I moved towards some of the state-of-the-art models and I selected efficientNet and denseNet201 so that I can use them as ensemble for better prediction. This idea was implemented by many of the Kagglers in their work.

EfficientNets rely on AutoML and compound scaling to achieve superior performance without compromising resource efficiency. The AutoML Mobile framework has helped develop a mobile-size baseline network, EfficientNet-B0, which is then improved by the compound scaling method to obtain EfficientNet-B1 to B7.[2]

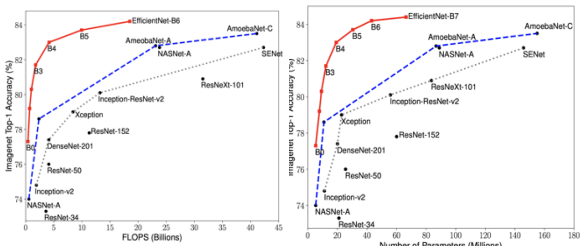

Figure 8 The performance of each model variant using the pre-trained weights [3]

EfficientNets achieve state-of-the-art accuracy on ImageNet with an order of magnitude better efficiency:

In high-accuracy regime, EfficientNet-B7 achieves the state-of-the-art 84.4% top-1 / 97.1% top-5 accuracy on ImageNet with 66M parameters and 37B FLOPS. At the same time, the model is 8.4x smaller and 6.1x faster on CPU inference than the former leader, Gpipe.

In middle-accuracy regime, EfficientNet-B1 is 7.6x smaller and 5.7x faster on CPU inference than ResNet-152, with similar ImageNet accuracy.

Compared to the widely used ResNet-50, EfficientNet-B4 improves the top-1 accuracy from 76.3% of ResNet-50 to 82.6% (+6.3%), under similar FLOPS constraints.

E. *Learning Rate and Early Stoping*

The **learning curve** mentioned below is like a ramp-up curve, using a ramp-up in the LR schedule is standard best practice when fine-tuning. The weights are pre-

trained and already have pretty good values. Hitting them with a high LR upfront would kick them too far. That's not what you want.

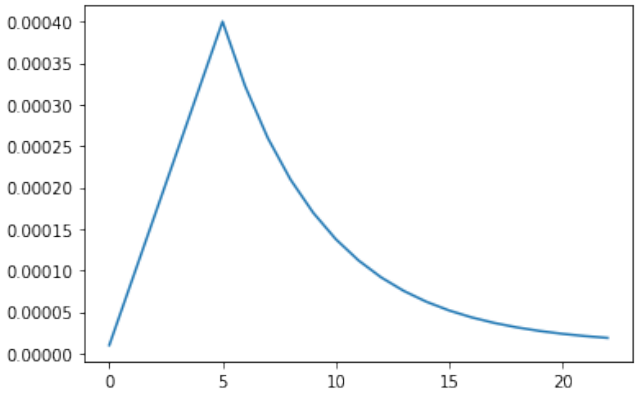

Figure 9 Learning Rate that I used for schedule model callbacks

I also made use of the **Keras Early Stopping** function while training to schedule my callback. I did this to Stop training when a monitored quantity has stopped improving

tf.keras.callbacks.EarlyStopping(min_delta=0,patience=10, verbose=1, mode='auto', restore_best_weights=True)
i.e., this callback will stop the training when there is no improvement in the validation loss for ten consecutive epochs.

F. Ensemble – DenseNet201 and EfficientNetB7

A collection of networks with the same configuration and different initial random weights is trained on the same dataset. Each model is then used to make a prediction and the actual prediction is calculated as the average of the predictions.

The number of models in the ensemble is often kept small both because of the computational expense in training models and because of the diminishing returns in performance from adding more ensemble members.[5]

```
Downloading data from https://github.com/keras-team/keras-applications/releases/download/den
senet/densenet201_weights_tf_dim_ordering_tf_kernels_notop.h5
74842112/74836368 [==============================] - 3s 0us/step
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
densenet201 (Model)          (None, 16, 16, 1920)      18321984
_________________________________________________________________
global_average_pooling2d (Gl (None, 1920)              0
_________________________________________________________________
dense (Dense)                (None, 104)               199784
=================================================================
Total params: 18,521,768
Trainable params: 18,292,712
Non-trainable params: 229,056
_________________________________________________________________
```

Figure 10 DenseNet201 model summary

In this scenario I used an ensemble of two models namely the DenseNet201 with weights of imagenet and the state-of-the-art model EfficientNet with the weights of noisy student.

```
Downloading data from https://github.com/qubvel/efficientnet/releases/download/v0.0.1/effici
entnet-b7_noisy-student_notop.h5
258072576/258068648 [==============================] - 8s 0us/step
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
efficientnet-b7 (Model)      (None, 16, 16, 2560)      64097680
_________________________________________________________________
global_average_pooling2d_1 ( (None, 2560)              0
_________________________________________________________________
dense_1 (Dense)              (None, 104)               266344
=================================================================
Total params: 64,364,024
Trainable params: 64,053,304
Non-trainable params: 310,720
_________________________________________________________________
```

Figure 11 EfficientNetB7 model summary

In my approach I trained both of the model separately on the training dataset and then saw their performance on the validation dataset.

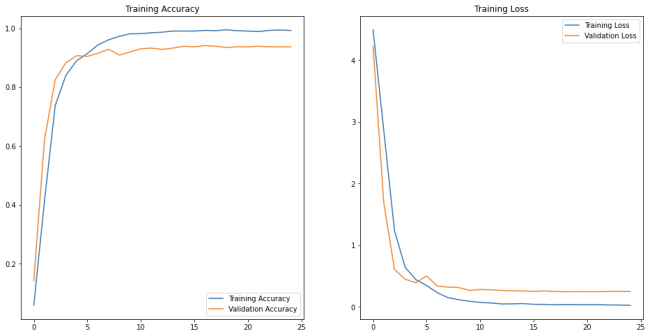

Figure 12 DenseNet201 cccuracy and Loss on train and validation with 25 epochs

Later I visualized the prediction on the validation set. The predictions and misprediction in the validation set generated by using the normalized combination of weights of the both the model in the ensemble. I also noticed the number of misprediction declined from the previous model if I used the ensemble technique.

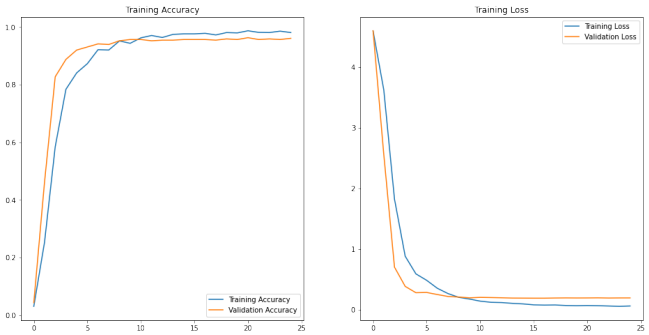

Figure 13 EfficientNetB7 accuracy and loss on train and validation with 25 epochs

When I was satisfied with the performance of the model on the validation dataset, I used my validation dataset with the training to increase the training dataset and then used the ensemble model to do the final prediction on the testing dataset for my Kaggle submission.

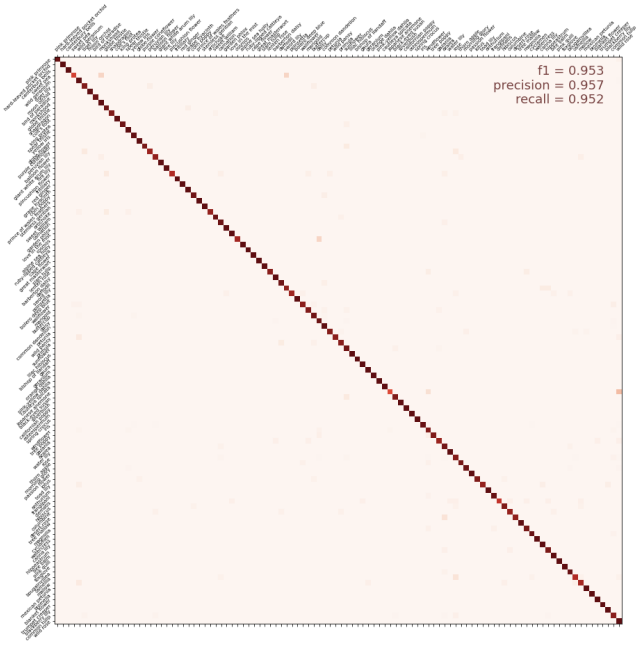

Figure 14 Ensemble - validation set confusion matrix

IV. Results and Analysis

A. *Vgg16 had lot of mispredictions.*

This is because the **Vgg16 is less Dense** than the other state of the art model like Xception, DenseNet201 and EfficientNet. Another possible reason is that there were some of the training and the validation classes with very less sample, so the vgg16 is mis predicting those classes with the classes which have far more better training and validation images.

Also, **Vgg16** encounters a **Vanishing gradient** problem but, deeper networks can have higher test error and generalize lesser if done simply like what VGG does.

B. *Training Dataset classes mislabled*

On further investigating I found out that some of the training samples are wrongly labeled

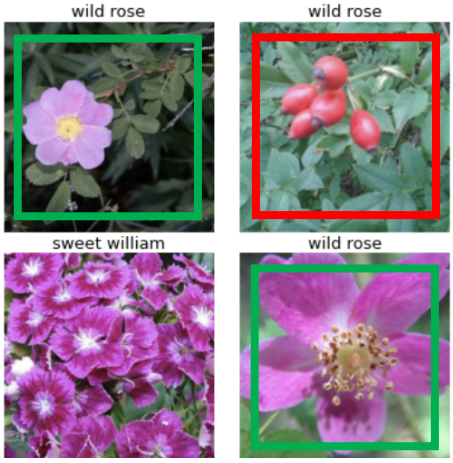

Figure 15 Wrong label for one of the training sample marked in red

C. *Benefits of Ensemble based learning.*

It can be helpful to think of varying each of the three major elements of the ensemble method; for example:

- Training Data: Vary the choice of data used to train each model in the ensemble.
- Ensemble Models: Vary the choice of the models used in the ensemble.
- Combinations: Vary the choice of the way that outcomes from ensemble members are combined.

D. *Kaggle Results*

TABLE I. LISIING OF THE KAGGLE F1 SCORE AND RANKING [6]

Model used	***Kaggle's F1 Score***	***Kaggle Ranking***
Vgg16 (with augmentation)	0.65409	667
DenseNet (with augmentation)	0.93071	570
EfficientNet (with augmentation)	0.95569	403
Ensemble - DenseNet201 and EfficientNetB7 (with augmentation)	**0.96011**	**351**

[a.] As of May 4th, 2020.

V. Future Work

I plan on to include **dataset** from **outside source** for training and to future increase validation accuracy without overfitting the model.

Perform **Image segmentation** to filter out flowers and to remove unnecessary noise in the background for some of the miss predicted and under sampled classes.

Train the ensemble is such a way that I have the best **normalized weights** for both of model used.

References

[1] https://www.kaggle.com/c/flower-classification-with-tpus/discussion/134701#769297.
[2] [3] [4] https://pypi.org/project/efficientnet/
[5] https://machinelearningmastery.com/ensemble-methods-for-deep-learning-neural-networks/
[6] https://www.kaggle.com/c/flower-classification-with-tpus/leaderboard