Flower Classification with TPUs

Herambeshwar Pendyala, Srinath Botsa

cs795/895 - Practical Machine Learning | Course project | Old Dominion University

01130541, hpend001@odu.edu

01129719, sbots001@odu.edu

***Abstract*— This document serves as a project report for course cs795/895 Practical Machine Learning. The problem is from Kaggle, the solution is given using a deep learning approach utilizing various advanced neural network algorithms like ResNet, DenseNet and EfficientNet and see which performs better in a given scenario along with the scores submitted in the Kaggle to check our positions in the leader board.**

***Keywords*— Flower Classification, TPUs, TensorFlow, Kaggle, Deep Learning, ResNet, DenseNet, EfficientNet, Image Rotation, Data Augmentation, Feature selection.**

I. Introduction

This is a Kaggle competition hosted by google cloud TPU team to make users familiar with TPUs and their usage at the same time demonstrating the power of TPU's computing power. In this competition we are classifying 104 types of flowers based on their images drawn from five different public datasets. Some classes are very narrow, containing only a sub-type of flower (e.g. pink primroses) while other classes contain many sub-types (e.g. wild roses). As part of the course Practical Machine Learning, we were given this Kaggle problem to work on as second project. This proposed solution uses Machine Learning and deep learning methods taught in the class to find the best fit to classify the flowers as well as analyse the results in detail. These methods are discussed in detail in the following sections.

II. Problem Statement

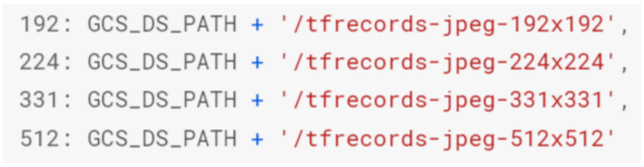
In this competition, participants are to build a machine learning model that identifies the type of flowers in a dataset of images of 104 classes. It is difficult to fathom just how vast and diverse our natural world is. There are over 5,000 species of mammals, 10,000 species of birds, 30,000 species of fish – and astonishingly, over 400,000 different types of flowers. And the prizes are given to competitors who leverage the power of TPU's provided by Kaggle.

The given problem looks more like a classification problem where we must classify them into 104 categories. Our approach to solve the problem is to use the Deep learning algorithms taught to us in the class combining with some state of art neural network architectures to predict the flower class out of given 104 classes.

a. Data Augmentation
b. ResNet
c. DenseNet201
d. EfficientNet

III. Dataset

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 shared data files for optimal training performance. 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.

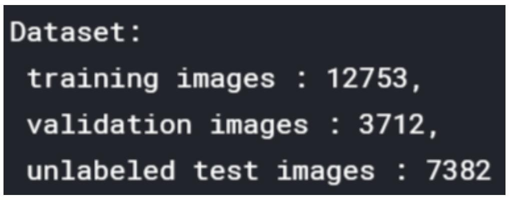


```
192: GCS_DS_PATH + '/tfrecords-jpeg-192x192',
224: GCS_DS_PATH + '/tfrecords-jpeg-224x224',
331: GCS_DS_PATH + '/tfrecords-jpeg-331x331',
512: GCS_DS_PATH + '/tfrecords-jpeg-512x512'
```

Fig. 1 Available Image Resolutions

Images are provided with 192, 224, 331, 512 resolutions and as we are using TPUs we are working on images with 512x512 resolution. We will be discussing more about TPUs in the next section.

We have divided the given data into training validation and test images.



```
Dataset:
 training images : 12753,
 validation images : 3712,
 unlabeled test images : 7382
```

Fig. 1 Data split into train, validation and test set

IV. Tensor Processing Units (TPUs)

TPUs are powerful hardware accelerators specialized in deep learning tasks. They were developed (and first used) by Google to process large image databases, such as extracting all the text from Street View. This competition is designed for you to give TPUs a try. The latest TensorFlow release (TF 2.1) was focused on TPUs and they're now supported both through the Keras high-level API and at a lower level, in models using a custom training loop.

Modern GPUs are organized around programmable "cores", a very flexible architecture that allows them to handle a variety of tasks such as 3D rendering, deep learning, physical simulations, etc. TPUs on the other hand pair a classic vector processor with a dedicated matrix multiply unit and excel at any task where large matrix multiplications dominate, such as neural networks.

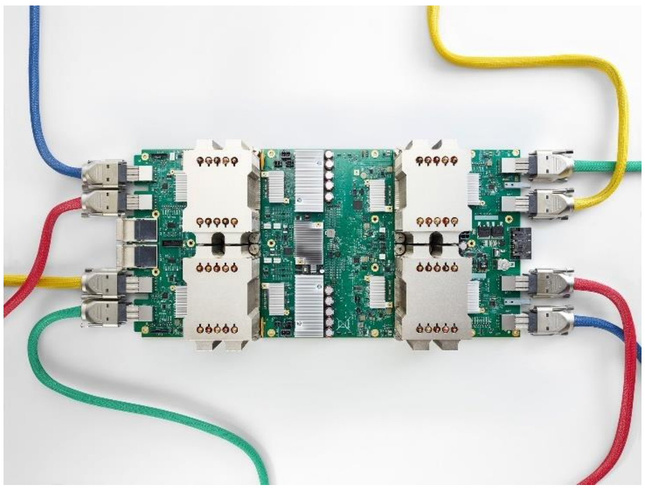


Fig. 2 Google's TPU

When Google designed the TPU, they built a domain-specific architecture. That means, instead of designing a general-purpose processor, they designed it as a matrix processor specialized for neural network workloads. TPUs cannot run word processors, control rocket engines, or execute bank transactions, but they can handle the massive multiplications and additions for neural networks, at blazingly fast speeds while consuming much less power and inside a smaller physical footprint.

The TPU loads data from memory. As each multiplication is executed, the result will be passed to next multipliers while taking summation at the same time. So the output will be the summation of all multiplication result between data and parameters. During the whole process of massive calculations and data passing, no memory access is required at all.

V. Visualizations

As we are approaching a problem, it is always good we start with a proper visualization of the data. as this gives us more insights about the data. At the very beginning we plot some of the given flowers along with the labels to see what it looks like, below are some of the images of train and validation data.

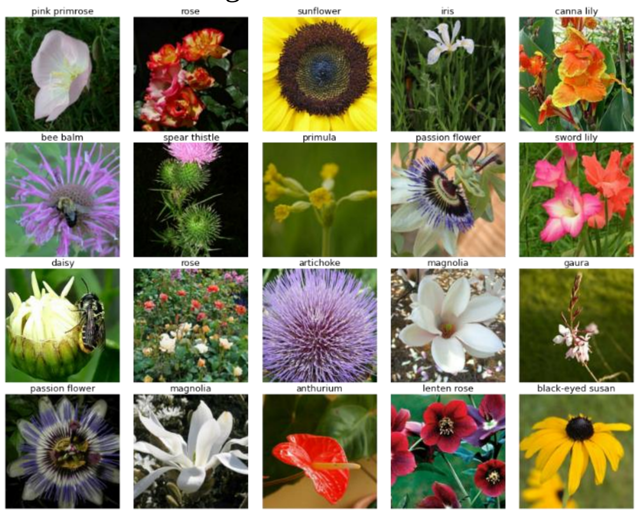


Fig. 2 Train image data

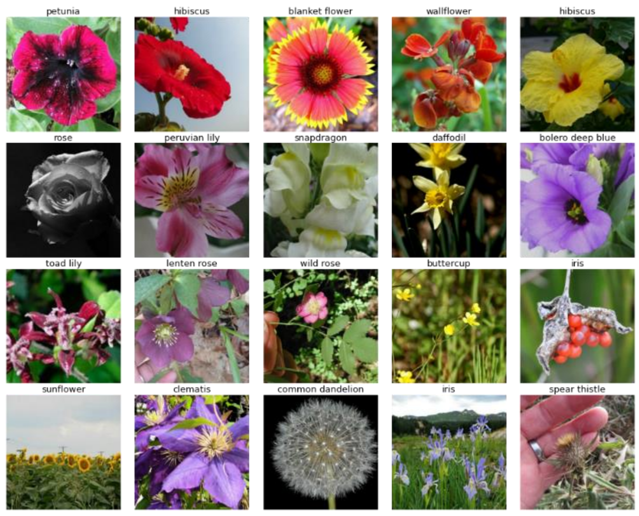


Fig. 2 Validation image data

The Test data has no labels as the submission file generated is validated by Kaggle to generate the score. Below are some of the test images.

Fig. 2 Test Image data

We have 104 classes of flowers; we plotted the labels to see the distribution of classes. After the visualization, We came to a conclusion that the datasets are largely unbalanced, but the train and validation sets have a similar distribution.

VI. Hyperparameters Used

To start with solving the problem we would like to build a deep learning model that takes images as input, and then outputs the class of the flower. As we are building neural network, It is equally important to tweak the hyper parameters to get the better performance of models. In this section we discuss about the hyperparameters used and their significance.

A. Data Augmentation

The performance of deep learning neural networks often improves with the amount of data available. Image data augmentation is a technique that can be used to artificially expand the size of a training dataset by creating modified versions of images in the dataset.

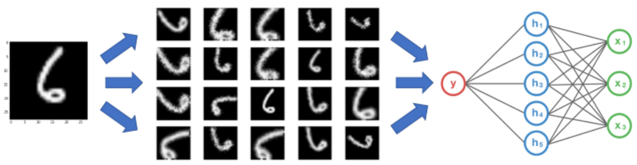


Fig. 2 data augmentation

Some of the popular Image Augmentation techniques are :

1. Flip
2. Rotation
3. Scale
4. Crop
5. Translation
6. Adding Gaussian Noise

For this project we have used flip and rotation and have done analysis in below sections.

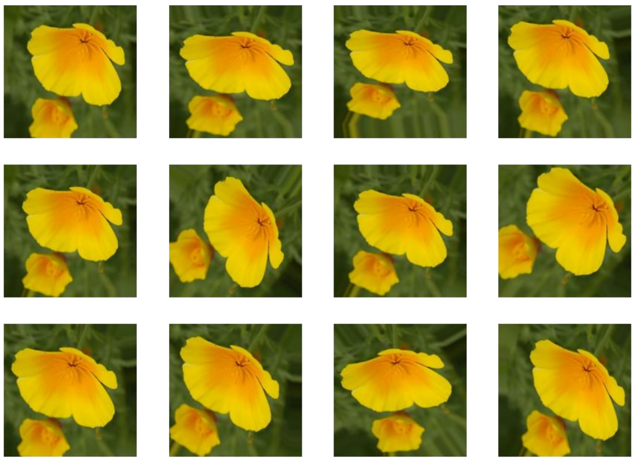


Fig. 2 Sample data augmentation

B. Transfer Learning

Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task.

It is a popular approach in deep learning where pre-trained models are used as the starting point on computer vision and natural language processing tasks given the vast compute and time resources required to develop neural network models on these problems and from the huge jumps in skill that they provide on related problems.

For this project, we used different deep learning models with pre-trained weights for a large and challenging image classification task such as the ImageNet 1000-class photograph classification competition. As without the pretrained weights, These models can take days or weeks to train on modern hardware.

C. Learning Rate Schedule

Adapting the learning rate for SGD optimization procedure can increase performance and reduce training time in any neural network. this is called as Learning rate schedule or adaptive learning rates. The simplest and perhaps most used adaptation of learning rate during training are techniques that reduce the learning rate over time. These have the benefit of making large changes at the beginning of the training procedure when larger learning rate values are used, and decreasing the learning rate such that a smaller rate and therefore smaller training updates are made to weights later in the training procedure.

This has the effect of quickly learning good weights early and fine tuning them later.

Two popular and easy to use learning rate schedules are as follows:

- Decrease the learning rate gradually based on the epoch.
- Decrease the learning rate using punctuated large drops at specific epochs.

Below is the learning rate graph depicting learning value range used. X-axis of the graph represents the epochs.

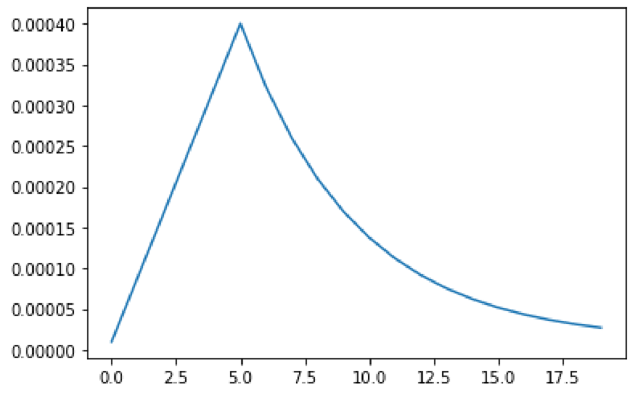


Fig. 5 Learning Rate Schedule for the whole network

Below are some of the other hyper parameters that we have used.

```
Batch Size = 128
Epochs = 20
```

VII. Methods Implemented

To start with solving the problem we would like to build a deep learning model that takes images as input, and then outputs the class of the flower. As we are building neural network, It is equally important to tweak the hyper parameters to get the better performance of models.

For this classification problem we have used various versions of Convolutional Neural Networks(CNNS) namely Resnet, DenseNet, EfficientNet. we will discuss briefly about each model in coming sections.

A. *ResNet*

We first start with resnet which is a version of CNN. In Standard CNN, input image goes through multiple convolutions and obtain high-level features as sown in below image.

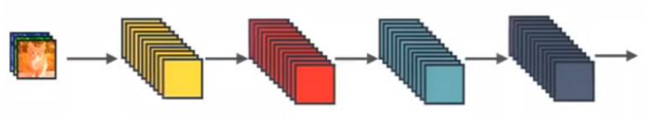


Fig. 11 Standard CNN

In ResNet, identity mapping is proposed to promote the gradient propagation. Element-wise addition is used. It can be viewed as algorithms with a state passed from one ResNet module to another one.

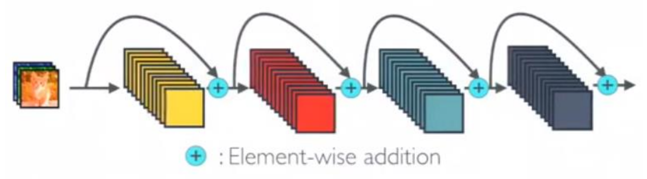


Fig. 11 ResNet

We have build this model from scratch to learn more about resnet and its architecture. And trained it without any pretrained weights to see the performance.

B. *DenseNet201*

In DenseNet, each layer obtains additional inputs from all preceding layers and passes on its own feature-maps to all subsequent layers. Concatenation is used. Each layer is receiving a “collective knowledge” from all preceding layers.

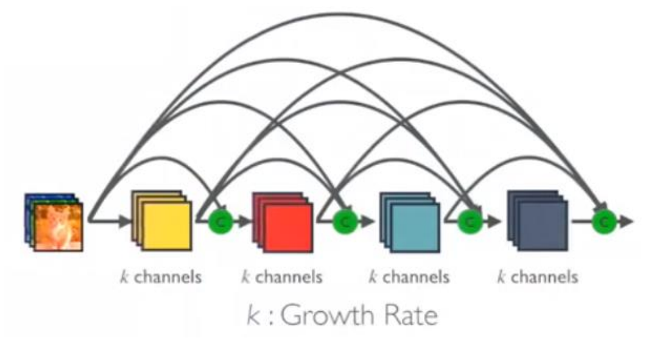


Fig. 11 DenseNet Architecture

Since each layer receives feature maps from all preceding layers, network can be thinner and compact, i.e. number of channels can be fewer. The growth rate k is the additional number of channels for each layer. The growth rate *k* is the additional number of channels for each layer.

So, it has higher computational efficiency and memory efficiency. Below are some of the advantages of DenseNet over CNN which improves the performance of the model as well.

Strong Gradient Flow: The error signal can be easily propagated to earlier layers more directly. This is a kind of implicit deep supervision as earlier layers can get direct supervision from the final classification layer.

Parameter and Computational Efficiency : For each layer, number of parameters in ResNet is directly proportional to C×C while Number of parameters in DenseNet is directly proportional to l×k×k.

Since k<<C, DenseNet has much smaller size than ResNet.

More Diversified Features : Since each layer in DenseNet receive all preceding layers as input, more diversified features and tends to have richer patterns.

In DenseNet, classifier uses features of all complexity levels. It tends to give more smooth decision boundaries. It also explains why DenseNet performs well when training data is insufficient.

C. EfficientNet

Most people in this competition has used Efficient Net for this competition. we were curious about what's Efficient and why everyone is getting good scores with this new Image architecture.

This paper discusses efficient way of scaling a Convolutional Neural Network efficiently. CNN’s must be scaled up in multiple dimensions. Scaling CNN’s only in one direction (e.g. depth only) will result in rapidly deteriorating gains relative to the computational increase needed. Most CNN’s are typically scaled up by adding more layers or deeper . e.g. ResNet18, ResNet34, ResNet152, etc. The numbers represent the total number of blocks (layers) and in general, the more layers the more ‘power’ the CNN has. Going wider is another often used scaling method and tends to capture finer details and can be easier to train. However, it’s benefits quickly saturate as well.

There are three scaling dimensions of a CNN: depth, width, and resolution. Depth simply means how deep the networks is which is equivalent to the number of layers in it. Width simply means how wide the network is. One measure of width, for example, is the number of channels in a Conv layer whereas Resolution is simply the image resolution that is being passed to a CNN. The figure below (from the paper itself) will give you a clear idea of what scaling means across different dimensions.

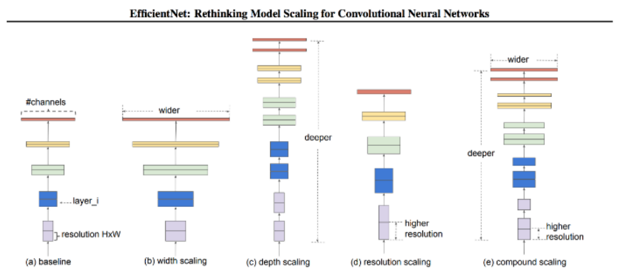


Fig. 11 Scaling CNN

To scale up efficiently, all dimensions of depth, width and resolution have to be scaled together, and there is an optimal balance for each dimension relative to the others. Intuition says that as the resolution of the images is increased, depth and width of the network should be increased as well. As the depth is increased, larger receptive fields can capture similar features that include more pixels in an image. Also, as the width is increased, more fine-grained features will be captured. To validate this intuition, the authors ran a number of experiments with different scaling values for each dimension.

These results lead to our second observation: It is critical to balance all dimensions of a network (width, depth, and resolution) during CNNs scaling for getting improved accuracy and efficiency.

The authors discovered that there is a synergy in scaling multiple dimensions together, and after an extensive grid search derived the theoretically optimal formula of "compound scaling" using the following co-efficients: Depth = 1.20, Width = 1.10, Resolution = 1.15

The authors proposed a simple yet very effective scaling technique which uses a compound coefficient ϕ to uniformly scale network width, depth, and resolution in a principled way

ϕ is a user-specified coefficient that controls how many resources are available whereas α, β, and γ specify how to assign these resources to network depth, width, and resolution, respectively.

In a CNN, Conv layers are the most compute expensive part of the network. Also, FLOPS of a regular convolution op is almost proportional to d, w^2, r^2, i.e. doubling the depth will double the FLOPS while doubling width or resolution increases FLOPS almost by four times. Hence, to make sure that the total FLOPS don't exceed 2^ϕ, the constraint applied is that $(\alpha \beta^2 \gamma^2) \approx 2$

VIII. Languages and Packages

We used Python as the programming language to solve this problem as it is easy to implement. We used Numpy, Math, re and TensorFlow for image operations. We used Keras and TensorFlow for Machine learning related utilities like importing the models and pretrained weights. Used matplotlib for visualizations.

IX. Experiments and Results

As we got an Idea about the data, we have built some helper functions to convert the images that are in TFRecord to the array format so that we can feed it as an input to neural network.

We started with a Resnet that was built from scratch without pre-training, we got a decent performance of 60 % accuracy for 100 epochs. Below is the graph that shows the training and validation loss and accuracy for a range of 100 epochs.

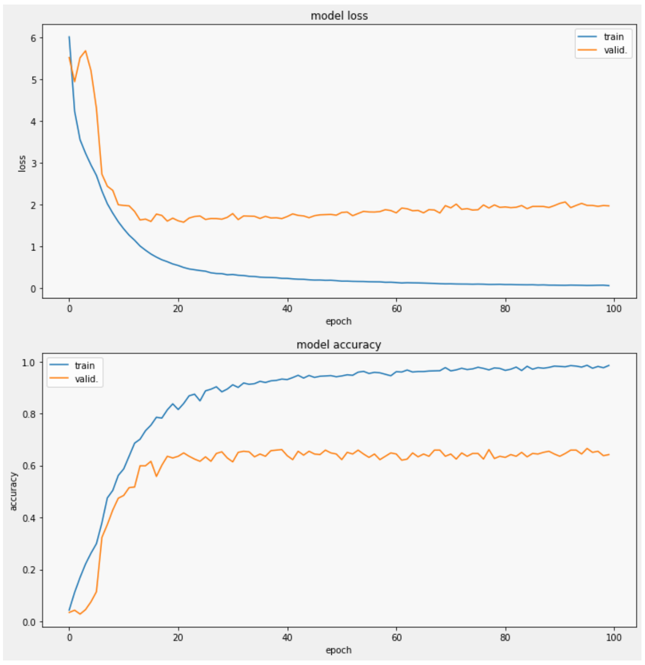
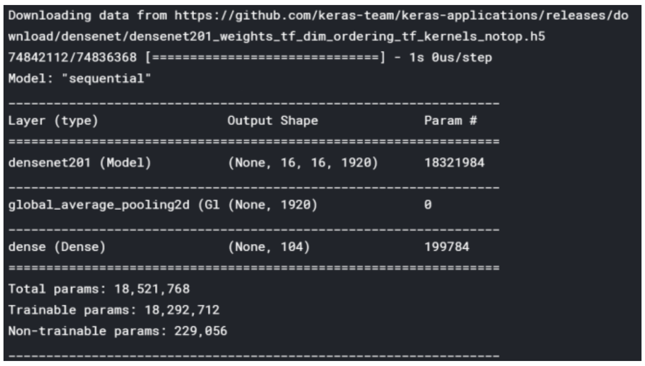


Fig. 11 Custom Resnet Performance

We can see that the model loss for training goes stable till 15 epochs, but the validation loss stops converging after 15 epochs. This tells us that the model is overfitting on training data. So, it is not converging for validation data.

Then we move into DenseNet201 which was available pre-trained in keras. We started with Densenet as an input layer, the output of which is added to an average pooling layer, and the output layer is a SoftMax layer with 104 classes. For this model we have used transfer learning with pretrained weights of imagenet. Below is the model summary.



```
Downloading data from https://github.com/keras-team/keras-applications/releases/do
wnload/densenet/densenet201_weights_tf_dim_ordering_tf_kernels_notop.h5
74842112/74836368 [==============================] - 1s 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
_________________________________________________________________
```

Fig. 11 DenseNet201 Model summary

This model gave us a descent performance both in training and validation with a training accuracy of 99% training accuracy and 95% validation accuracy. Below is the graph of the model performance for 20 epochs.

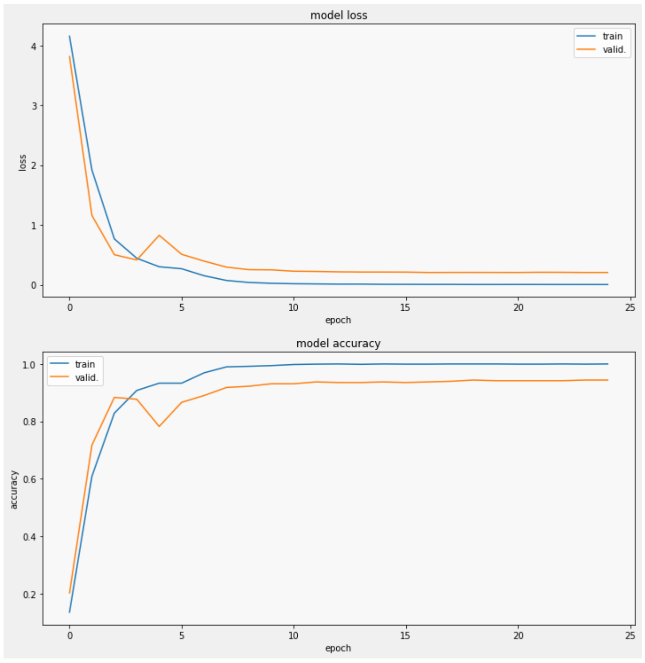
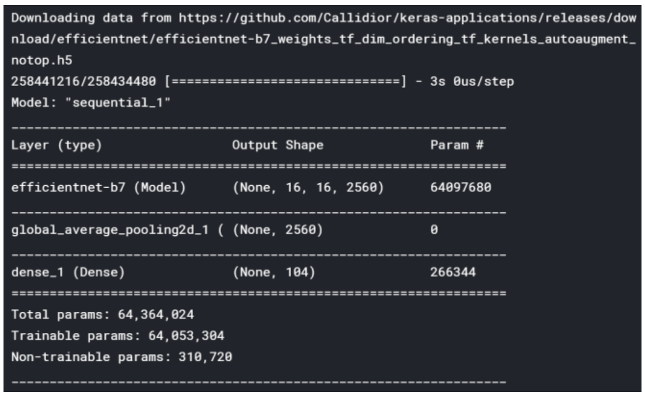


Fig. 11 DenseNet201 Performance

We can see that the model loss for training goes stable till 4 epochs, but the validation then fluctuates before stop converging after 8 epochs. We can see that gives us a descent performance but not an optimum performance.

Then we move into EfficientNet which was available with pre-trained. We started with EfficientNet as an input layer, the output of which is added to an average pooling layer, and the output layer is a SoftMax layer with 104 classes. Below is the model summary.



```
Downloading data from https://github.com/Callidior/keras-applications/releases/dow
nload/efficientnet/efficientnet-b7_weights_tf_dim_ordering_tf_kernels_autoaugment_
notop.h5
258441216/258434480 [==============================] - 3s 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
_________________________________________________________________
```

Fig. 11 EfficientNet B7 Model summary

This model gave us a better performance compared to earlier DenseNet201 and ResNet model with accuracy of 99% in training and 96% in validation. Below is the graph of the model performance for 20 epochs.

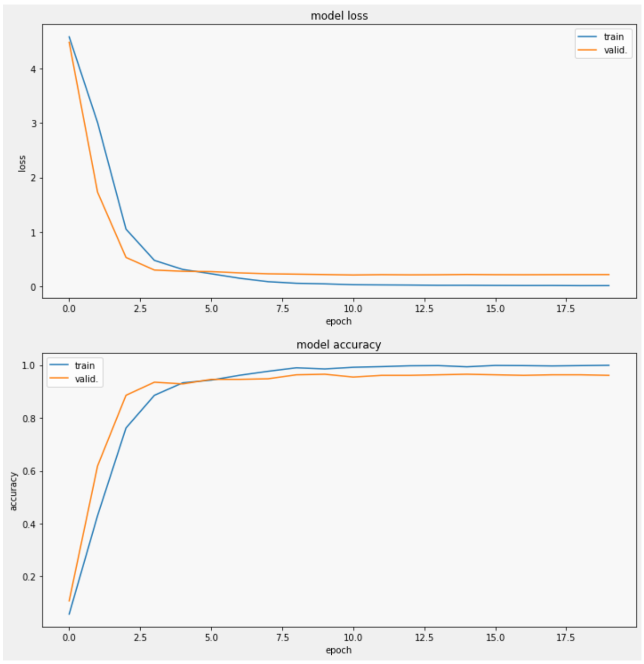


Fig. 11 EfficientNet B7 Performance

We can see that the model loss for training goes stable till 2 epochs, but the validation then fluctuates before stop converging after 4 epochs. We can see that gives us a better performance over ResNet performance and a slightly better performance over DenseNet201.

The Last method we tried was an ensemble method where use both the EfficientNet B7 and Densenet201 as Two nodes where both the layers get their input and the output is the average of outputs of both the models. This helps us to choose the best output from each model.

For evaluation we have used F1-Score, Precision and Recall. Below tables evaluate the model performances.

TABLE 1
MODEL PERFORMANCE ON TRAINING AND VALIDATION DATA

Model	**Precision**	**Recall**	**F1-Score**
Custom ResNet	0.604	0.614	0.6056
DenseNet201	0.961	0.954	0.9550
EfficientNet B7	0.954	0.960	0.9560
Ensemble (DenseNet201 and EfficientNet)	0.962	0.965	0.9620

From the tables It can be inferred that the ensemble method performs better that the models individually. So, we considered using it for out further analysis.

Below confusion Matrix for the ensemble model where it has an f1-score of 0.962.

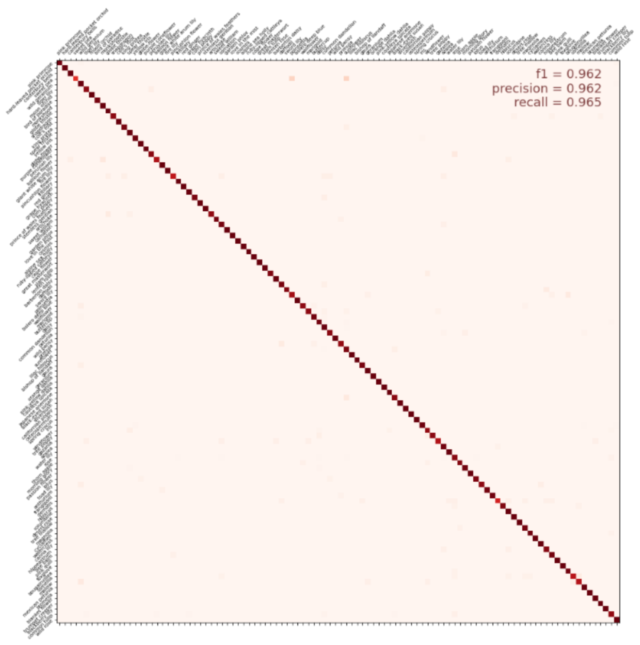


Fig. 12 Confusion Matrix of ensemble model

From our analysis above, all the pretrained models perform equally better, but ensemble method performs slightly better as we can see. We will move further into the analysis to get more insights about the mis predictions in the coming sections.

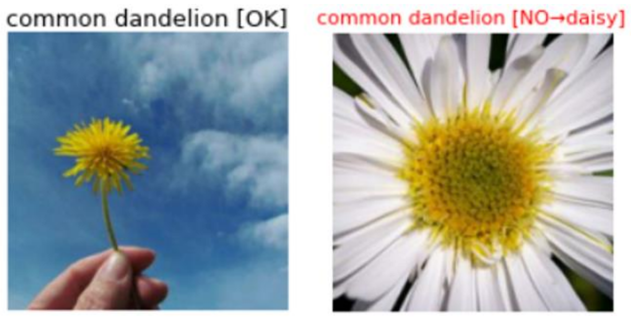
X. ANALYSIS

As we have trained and tested the models in the previous section we got a decent accuracy score of 96% on our validation. This may not be a good score, But we have used the models to generate the submission files. After generating the submission file, we submitted it and below tables show the results and our position in the competition.

TABLE 2
SCORES OF VARIOUS MODELS USED FOR CLASSIFICATION

Model	**Train Data**	**Score**
Custom ResNet	Training	0.6056
DenseNet201	Training	0.9550
DenseNet201	Training, Validation	0.9508
EfficientNet B7	Training, Validation	0.9570
EfficientNet B7	Training, Validation	0.9581
Ensemble (DenseNet201 and EfficientNet)	Training, Validation	0.9554
Ensemble (DenseNet201 and EfficientNet)	Training, Validation	0.96186

From the above tables we can see that in the two runs, where we have training and validation data combined, we gave got better results landing us in a good position in the leader board. But This is not a good sign as we have no data to validate our prediction, It also looks like we don’t have enough data for training even we are using data augmentation. So, we can explore more data augmentation techniques to improve our position also when we explore the mis predicted ones like one below.



we can see that the daisy is mis predicted as common dandelion on the right. From our analysis, we found that the image on the left is in training where we have clouds in the image along with the common dandelion, we think this might be the cause as the model learn to take hand and the colour of cloud as a feature which can cause mispredictions. There are also errors in training where we have images with people holding flowers and other things along with flowers which might lead the model in a different path.

XI. Conclusions

We have learned about neural network architectures that can used in classification purposes which include state of art architectures like Efficient Net. Learned what TPUs are and how to use them and benefit from them in processing large resolution images (512x512 in this project).

This implementation can be further improved by various methods available. The first spot on kaggle was able to achieve a descent f1-score. We can use other state of art methods like image segmentation and other data augmentation techniques and improve our position in the kaggle leaderboard.

Acknowledgment and Future Scope

As we have implemented models only using EfficientNet and Densnet201, though we got a decent performance, we can use other state of art neural network architectures like capsule nets to improve the model accuracy and prediction.

We have also seen that the data is imbalanced and there are many anomalies in the data, to mitigate these things we can use image segmentation crop the flower from the rest of the image as this might increase the accuracy of model as we are removing the redundant features. The problem of imbalanced datasets can be handled by using data augmentation techniques to increase the data size, though we tried in Kaggle, it prevents us from saving the data on it. We would like to see if we can really improve the accuracy by using data Augmentation, image segmentation on other State of the art neural network architectures.

References

[1] Data Augmentation Keras, https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/

[2] Data Augmentation in deep Learning, https://nanonets.com/blog/data-augmentation-how-to-use-deep-learning-when-you-have-limited-data-part-2/

[3] Transfer Learning Intro, https://machinelearningmastery.com/transfer-learning-for-deep-learning/

[4] Transfer Learning in practice, https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a

[5] Resnet Review, https://towardsdatascience.com/review-resnet-winner-of-ilsvrc-2015-image-classification-localization-detection-e39402bfa5d8

[6] DenseNet Review, https://towardsdatascience.com/review-densenet-image-classification-b6631a8ef803

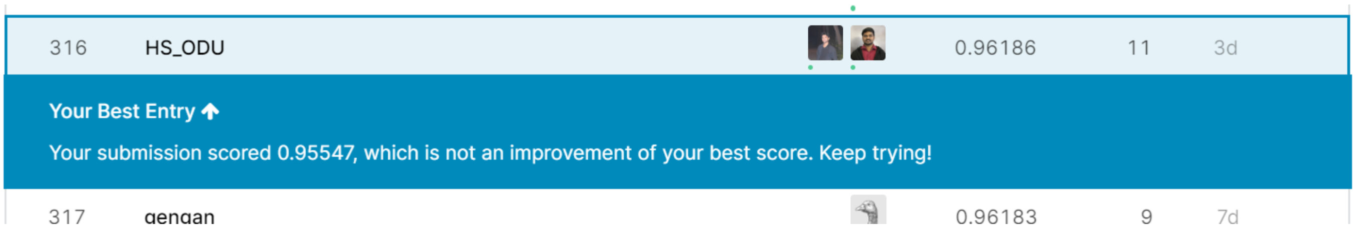
[7] EfficientNet Review, https://medium.com/@nainaakash012/efficientnet-rethinking-model-scaling-for-convolutional-neural-networks-92941c5bfb95

[8] TPUs, https://codelabs.developers.google.com/codelabs/keras-flowers-tpu/#2

[9] KernalEfficientNet,https://www.kaggle.com/kurianbenoy/introduction-kernel-what-s-efficientnet

[10] EfficientNet by Google, https://towardsdatascience.com/googles-efficientdet-an-overview-8d010fa15860

Final Position on leader Board



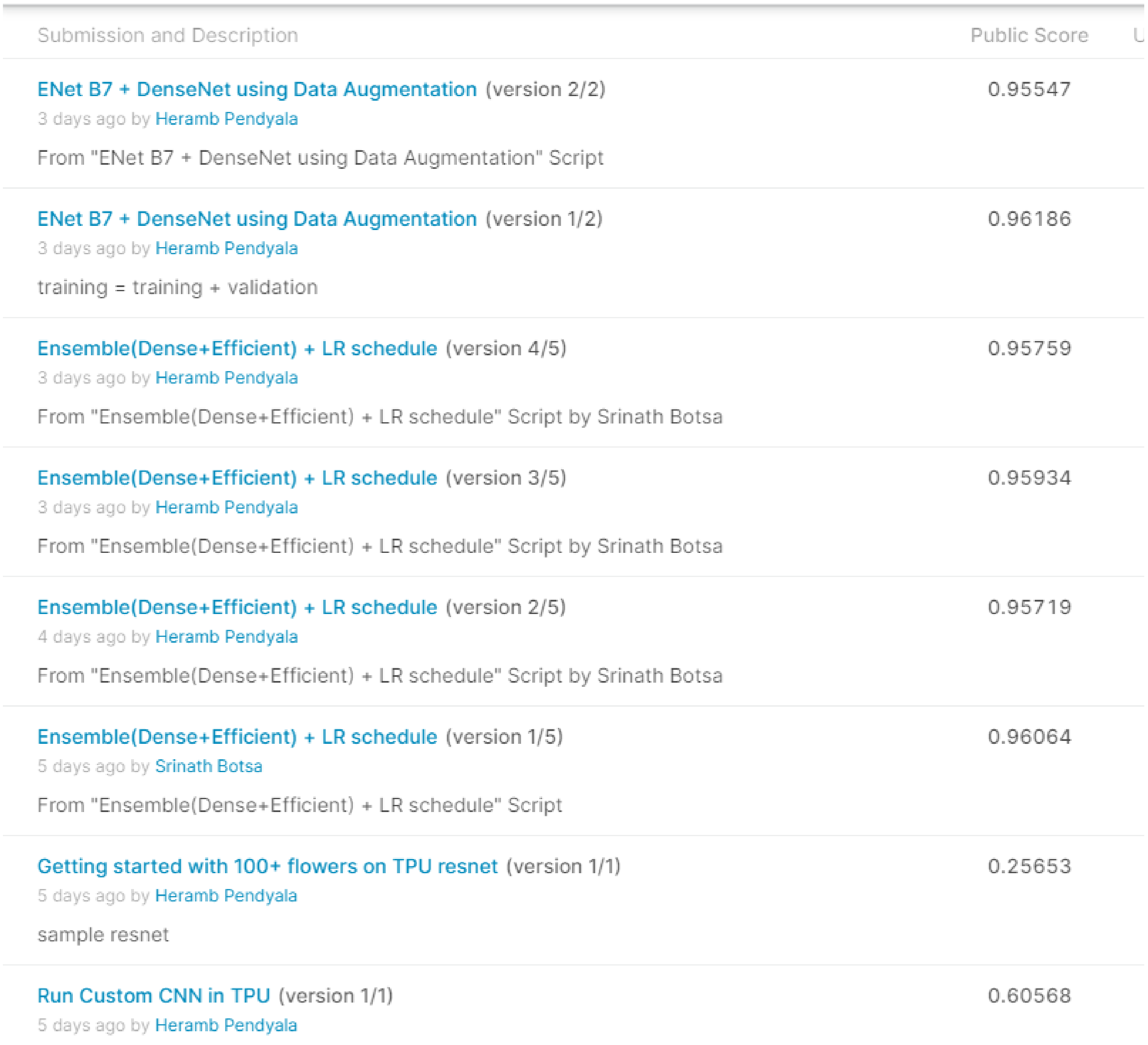
316 HS_ODU 0.96186 11 3d

Your Best Entry

Your submission scored 0.95547, which is not an improvement of your best score. Keep trying!

317 gengan 0.96183 9 7d

All scores for models submitted on kaggle



Submission and Description	Public Score
ENet B7 + DenseNet using Data Augmentation (version 2/2) 3 days ago by Heramb Pendyala From "ENet B7 + DenseNet using Data Augmentation" Script	0.95547
ENet B7 + DenseNet using Data Augmentation (version 1/2) 3 days ago by Heramb Pendyala training = training + validation	0.96186
Ensemble(Dense+Efficient) + LR schedule (version 4/5) 3 days ago by Heramb Pendyala From "Ensemble(Dense+Efficient) + LR schedule" Script by Srinath Botsa	0.95759
Ensemble(Dense+Efficient) + LR schedule (version 3/5) 3 days ago by Heramb Pendyala From "Ensemble(Dense+Efficient) + LR schedule" Script by Srinath Botsa	0.95934
Ensemble(Dense+Efficient) + LR schedule (version 2/5) 4 days ago by Heramb Pendyala From "Ensemble(Dense+Efficient) + LR schedule" Script by Srinath Botsa	0.95719
Ensemble(Dense+Efficient) + LR schedule (version 1/5) 5 days ago by Srinath Botsa From "Ensemble(Dense+Efficient) + LR schedule" Script	0.96064
Getting started with 100+ flowers on TPU resnet (version 1/1) 5 days ago by Heramb Pendyala sample resnet	0.25653
Run Custom CNN in TPU (version 1/1) 5 days ago by Heramb Pendyala	0.60568