Performance Evaluation of Different CNN Architectures on The Flower Recognition Dataset (December 2018)

Aryan Sohrabi
The Department of Computational Science
San Diego State University
California, USA
Email: asohrabi5720@sdsu.edu

Gopinath Dayalan
The Department of Computational Science
San Diego State University
California, USA
Email: gdayalan1814@sdsu.edu

***Abstract*— In this report we explore the performance of deep learning models on the flower recognition data set. Hyper parameter tuning is used to find the best set of parameters that would achieve the best performance metrics on the testing set. Because there are many hyper parameters in a neural network model, we decided to focus on certain CNN architectures. This way we wouldn't have to tune for parameters that determine the structure of the convolutional/pooling/fully connected layers. In turn, we train for a manageable set of hyper parameters such as learning rate, mini batch size, regularization and epochs. We pick the best set of hyper parameters for each CNN architecture and then compare their performance on the test set. Our evaluation metrics are accuracy, precision, recall-rate and run time.**

I. INTRODUCTION

Convolutional neural networks (CNN) have shown to be very good models for learning from image data sets. The convolutional layers are powerful because of parameter sharing and sparsity of connections. Stacking these layers with the combination of pooling and fully connected layers has shown to be superior for detecting image features and image classification. They are also easy to extend into multi-class classification models by adding a soft-max layer at the end.

Many architectures (combination of layers) have been proposed and shown to have good performance by researchers in the deep learning field. The most popular amongst these architectures are AlexNet, VGG-16, ResNets and Inception net.

The flower recognition dataset contains 4242 images of flowers. The data collection is based on the data flicr, google images, yandex images. The pictures are divided into five classes: chamomile, tulip, rose, sunflower, dandelion. For each class there are about 800 photos. Photos are about 320x240 with different sizes.

A. Challenges

One big challenge of deep learning models are the training times. Hyper parameter tuning requires us to train the model several times and this becomes inefficient when each training takes several hours or days. Solutions

1) *Transfer Learning*

This method refers to taking the weights of a model after training on a certain dataset and use them as initialization weights for training on another dataset using the same model. After this, you freeze the initial layers and only train on the final layers of your neural network. This way the number of parameters that we need to train for greatly reduces and so will the training time.

This method works because features that are detected in the early layers are common features among datasets and the later layers detect features specific to datasets.

2) *Batch Norm*

This method speeds up gradient decent by changing the shape of the loss function. At each layer during the forward pass activation values are normalized by setting the mean to zero and variance to one. Then, the mean and variance are learned during the back propagation.

3) *Optimization on gradient decent*

This method optimizes gradient decent by reducing fluctuations in non-gradient directions and thus makes the learning process faster. The fluctuations are the approximation errors caused when mini batch gradients are used to approximate the batch gradient.

4) *Learning rate decay*

This method gradually decays the learning rate so that we are able to take larger steps in the initial phase of gradient decent to learn faster and take shorter steps near the optimum to converge faster.

5) *Using GPU*

Deep learning procedure requires many large matrix operations which run faster on a GPU.

6) *Random search hyper-parameter tuning*

Unlike grid search that tries all the possible combinations of the hyper-parameters, random search chooses one value on random for each hyperparameter during each iteration. This way we can keep the running time low by using an appropriate number of iterations.

7) *Resizing images to a smaller size*

Since our images do not have a single size, we have to resize them. Resizing them to a smaller size reduces the computation and makes the running time faster.

8) *Small cross validation folds*

Bigger folds require more time because the process of training and testing on different fold combinations. We picked 3-fold so that the results are relatively accurate without taking too much time.

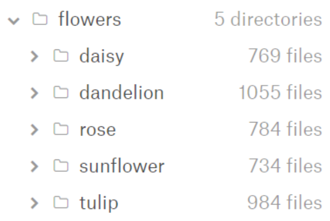
II. Running instruction

Running neutral networks on image data is computationally expensive process. Performing the training operations on CPU is time consuming, so Tesla K80 cloud GPU was hired from Floyd hub for faster processing.

Running on cloud: Use the directories paths for

III. Data preprocessing

The images are in 5 folders corresponding to the flower types (sunflower, tulip, daisy, rose, dandelion). In the first step we resize all the images to a single size and divide the RGB values by 255 to get them into normalized form. Then we load all these images into a 4-dimensional array. Image preloader function from tflearn stimulate fast loading of data. The first index is the image, the second index is the row number the third index is the column number, the fourth index is the array of RGB values. When loading each image, we load its label as a string into the same index in another array (Y values array).



flowers	5 directories
daisy	769 files
dandelion	1055 files
rose	784 files
sunflower	734 files
tulip	984 files

Fig 1. Structure of the dataset

To convert the labels into numerical data, we first assign a number 1-5 to each label and then convert that number to length 5 binary array. This method is called one hot encoding and makes the labels ready to use for computing losses in the last layer of the neural network.

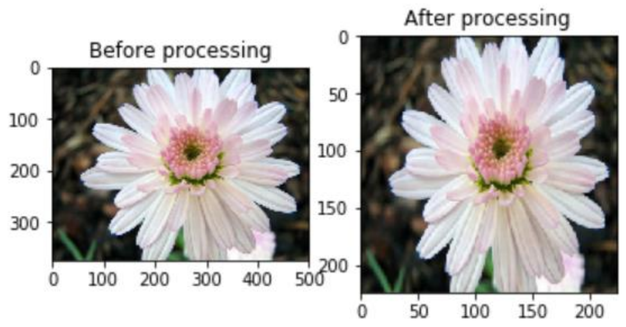


Fig 2. Representation of data before and after processing

To reduce overfitting, we perform data augmentation. This is done using a Keras library function called ImageDataGenerator which given the input data, creates a larger data set by performing operations such as shifting, zooming and mirroring on the images.

The dataset is split into training and testing set. 75% of the data are randomly selected for training and rest 25% is kept for testing the models. The hyper parameter tuning is done on a 3-fold cross validation, done automatically by sklearn's RandomizedSearchCV.

IV. Model implementation

A. VGG19 (23 layers)

Random search with 3-fold cross validation was performed for 15 iterations with the following hyper-parameter choices. Hyper-parameters number (2),(3) and (4) are for the hidden layers.

1) *Hidden layers = [1,2,3]*
   number of hidden layers after the last layer of transfer learning's base model.
2) *Activation function = ['relu', 'tanh', 'sigmoid']*
3) *Dropout rate = [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]*
4) *Neurons = [128,256,512]*
5) *Optimizer = ['RMSprop', 'Adam', 'Adamax', 'Nadam']*
6) *Optimizer = ['RMSprop', 'Adam', 'Adamax', 'Nadam']*
7) *Freeze layers = [11,15,22]*
   The layer number for which all the weights in the previous layers are frozen (set to untrainable).
8) *Epochs = [10,11,12,13,14,15,16,17,18,19,20]*
9) *Batch size = [16,32,64,128]*

Models	**Hyper Parameters**
Model 1	optimizer: Adam, neurons: 512, hidden_layers: 3, freeze_layers: 15, epochs: 16, dropout_rate: 0.5, batch_size: 128, activation: sigmoid
Model 2	optimizer: Nadam, neurons: 128, hidden_layers: 1, freeze_layers: 15, epochs: 15, dropout_rate: 0.3, batch_size: 32, activation: relu
Model 3	optimizer: RMSprop, neurons: 128, hidden_layers: 1, freeze_layers: 22, epochs: 13, dropout_rate: 0.6, batch_size: 64, activation: sigmoid
Model 4	optimizer: Nadam, neurons: 512, hidden_layers: 1, freeze_layers: 22, epochs: 19, dropout_rate: 0.4, batch_size: 32, activation: sigmoid
Model 5	optimizer: Nadam, neurons: 512, hidden_layers: 3, freeze_layers: 22, epochs: 19, dropout_rate: 0.0, batch_size: 64, activation: sigmoid
Model 6	optimizer: Adam, neurons: 128, hidden_layers: 2, freeze_layers: 11, epochs: 17, dropout_rate: 0.0, batch_size: 16, activation: relu
Model 7	optimizer: Adam, neurons: 512, hidden_layers: 1, freeze_layers: 15, epochs: 16, dropout_rate: 0.6, batch_size: 32, activation: tanh
Model 8	optimizer: Adamax, neurons: 128, hidden_layers: 3, freeze_layers: 11, epochs: 17, dropout_rate: 0.2, batch_size: 64, activation: sigmoid
Model 9	optimizer: RMSprop, neurons: 128, hidden_layers: 1, freeze_layers: 11, epochs: 19, dropout_rate: 0.2, batch_size: 32, activation: sigmoid
Model 10	optimizer: RMSprop, neurons: 256, hidden_layers: 3, freeze_layers: 15, epochs: 10, dropout_rate: 0.1, batch_size: 128, activation: sigmoid

Model 11	optimizer: Nadam, neurons: 512, hidden_layers: 3, freeze_layers: 22, epochs: 19, dropout_rate: 0.0, batch_size: 64, activation: sigmoid

Grid search methodologies is followed to select the top 10 hyper parameters based on the list of parameters provided to the training model. Each model is ranked based on the accuracy, precision and recall.

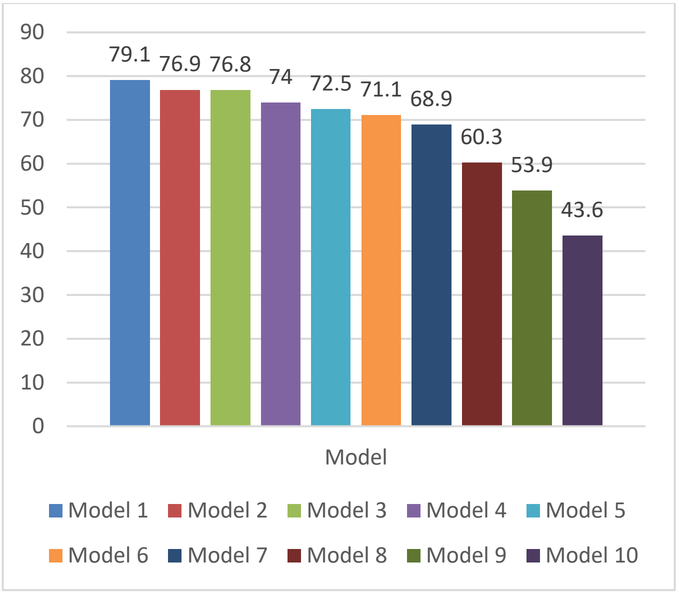


Fig 3. Accuracy of VGG19 with different hyper-parameters

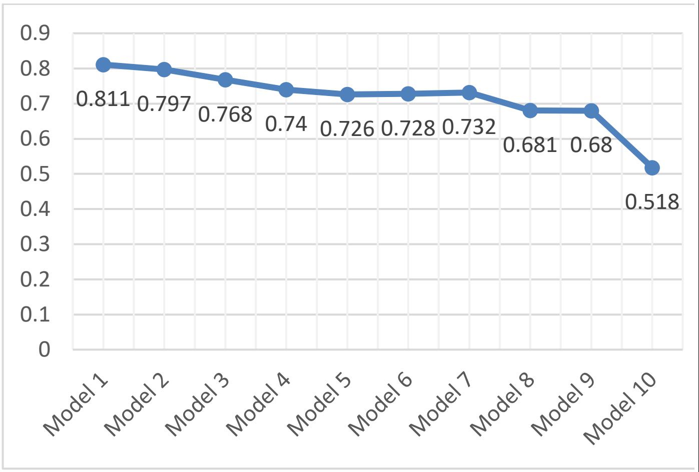


Fig 4. Top 10 Precision in 15 Models

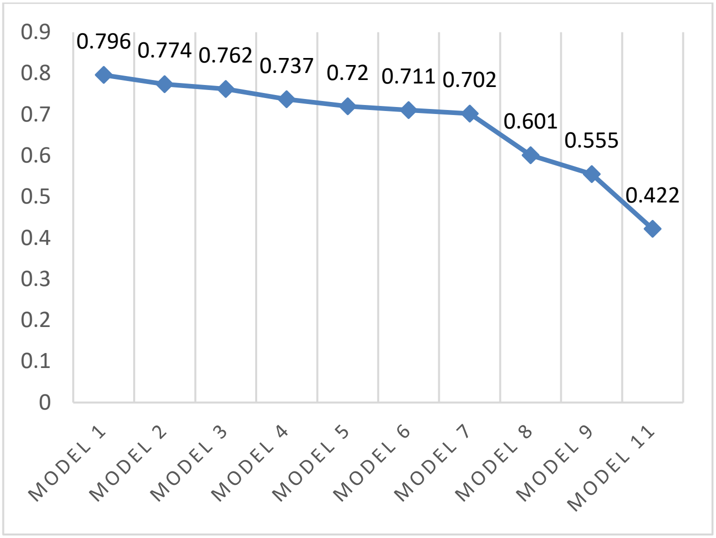


Fig 5. Top 10 recall in 15 models

B. Mobile Net (88 layers)

Random search with 3-fold cross validation was performed for 10 iterations with the same hyper-parameter choices as for (A) except for freeze layers.

1) *Freeze layers = [65,75,88]*
2) *Everything else same as (A)*

Models	Hyper Parameters
Model 1	optimizer: Adamax, neurons: 128, hidden_layers: 3, freeze_layers: 65, epochs: 17, dropout_rate: 0.2, batch_size: 64, activation: sigmoid
Model 2	optimizer: Adamax, neurons: 256, hidden_layers: 2, freeze_layers: 65, epochs: 17, dropout_rate: 0.3, batch_size: 128, activation: tanh
Model 3	optimizer: Adamax, neurons: 128, hidden_layers: 2, freeze_layers: 65, epochs: 13, dropout_rate: 0.7, batch_size: 16, activation: sigmoid
Model 4	optimizer: RMSprop, neurons: 128, hidden_layers: 1, freeze_layers: 65, epochs: 19, dropout_rate: 0.2, batch_size: 32, activation: sigmoid
Model 5	optimizer: Adamax, neurons: 512, hidden_layers: 2, freeze_layers: 65, epochs: 12, dropout_rate: 0.6, batch_size: 16, activation: tanh
Model 6	optimizer: Nadam, neurons: 512, hidden_layers: 1, freeze_layers: 88, epochs: 19, dropout_rate: 0.4, batch_size: 32, activation: sigmoid
Model 7	optimizer: RMSprop, neurons: 128, hidden_layers: 1, freeze_layers: 88, epochs: 13, dropout_rate: 0.6, batch_size: 64, activation: sigmoid
Model 8	optimizer: Nadam, neurons: 512, hidden_layers:

	3, freeze_layers: 88, epochs: 19, dropout_rate: 0.0, batch_size: 64, activation: sigmoid
Model 9	optimizer: RMSprop, neurons: 256, hidden_layers: 3, freeze_layers: 75, epochs: 10, dropout_rate: 0.1, batch_size: 128, activation: sigmoid
Model 10	optimizer: Adam, neurons: 512, hidden_layers: 3, freeze_layers: 75, epochs: 16, dropout_rate: 0.5, batch_size: 128, activation: sigmoid

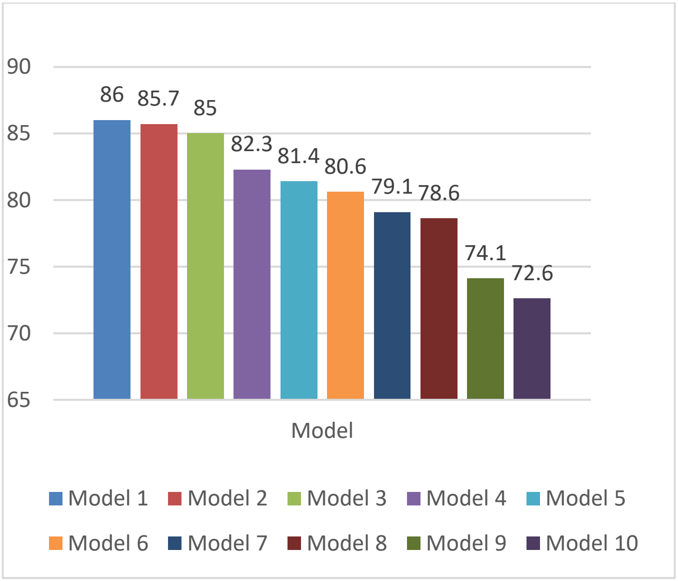


Fig 6. Accuracy of MobileNet with different hyper-parameters

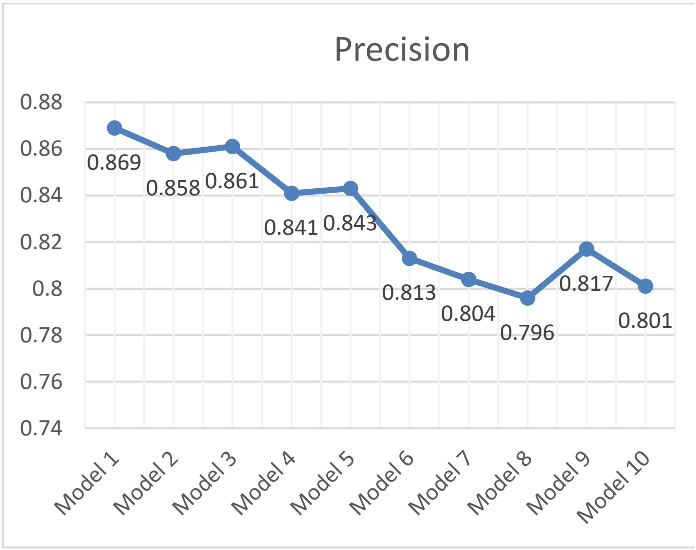


Fig 7. Precision of MobileNet with different hyper-parameters

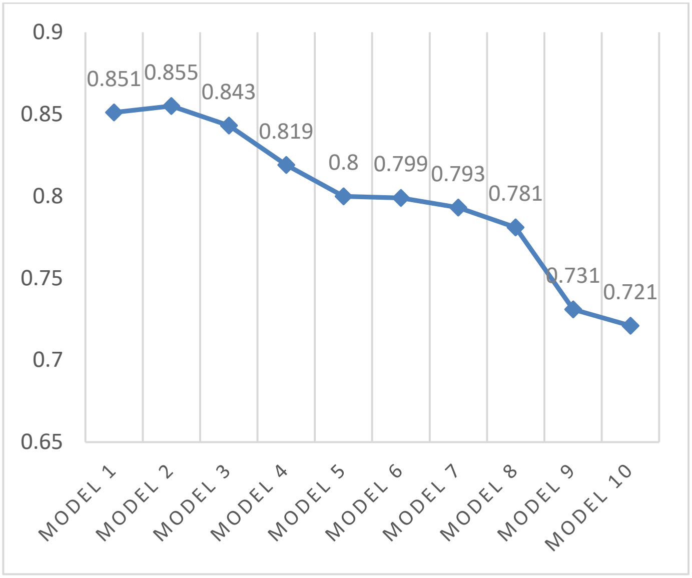


Fig 8. Recall of MobileNet with different hyper-parameters

C. VGG16 (20 layers)

Random search with 3-fold cross validation was performed for 10 iterations with the same hyper-parameter choices as for (A) except for freeze layers.

1) *Freeze layers = [65,75,88]*
2) *Everything else same as (A)*

Model 1	optimizer: Adam, neurons: 512, hidden_layers: 3, freeze_layers: 15, epochs: 16, dropout_rate: 0.5, batch_size: 128, activation: sigmoid
Model 2	optimizer: RMSprop, neurons: 128, hidden_layers: 1, freeze_layers: 20, epochs: 13, dropout_rate: 0.6, batch_size: 64, activation: sigmoid
Model 3	optimizer: Nadam, neurons: 512, hidden_layers: 1, freeze_layers: 20, epochs: 19, dropout_rate: 0.4, batch_size: 32, activation: sigmoid
Model 4	optimizer: RMSprop, neurons: 128, hidden_layers: 1, freeze_layers: 10, epochs: 19, dropout_rate: 0.2, batch_size: 32, activation: sigmoid
Model 5	optimizer: Nadam, neurons: 512, hidden_layers: 3, freeze_layers: 20, epochs: 19, dropout_rate: 0.0, batch_size: 64, activation: sigmoid
Model 6	optimizer: RMSprop, neurons: 256, hidden_layers: 3, freeze_layers: 15, epochs: 10, dropout_rate: 0.1, batch_size: 128, activation: sigmoid
Model 7	optimizer: Adamax, neurons: 128, hidden_layers: 3, freeze_layers: 10, epochs: 17, dropout_rate: 0.2, batch_size: 64, activation: sigmoid
Model 8	optimizer: Adamax, neurons: 256, hidden_layers: 2, freeze_layers: 10, epochs: 17, dropout_rate: 0.3, batch_size: 128, activation: tanh
Model 9	optimizer: Adamax, neurons: 512, hidden_layers: 2, freeze_layers: 10, epochs: 12, dropout_rate: 0.6, batch_size: 16, activation: tanh
Model 10	optimizer: Adamax, neurons: 128, hidden_layers: 2, freeze_layers: 10, epochs: 13, dropout_rate: 0.7, batch_size: 16, activation: sigmoid

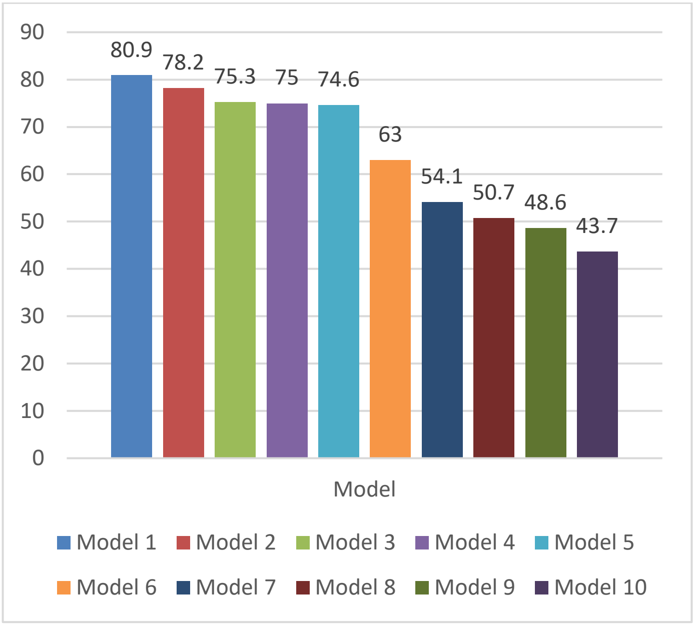


Fig 9. Accuracy of VGG 16 with different hyper-parameters

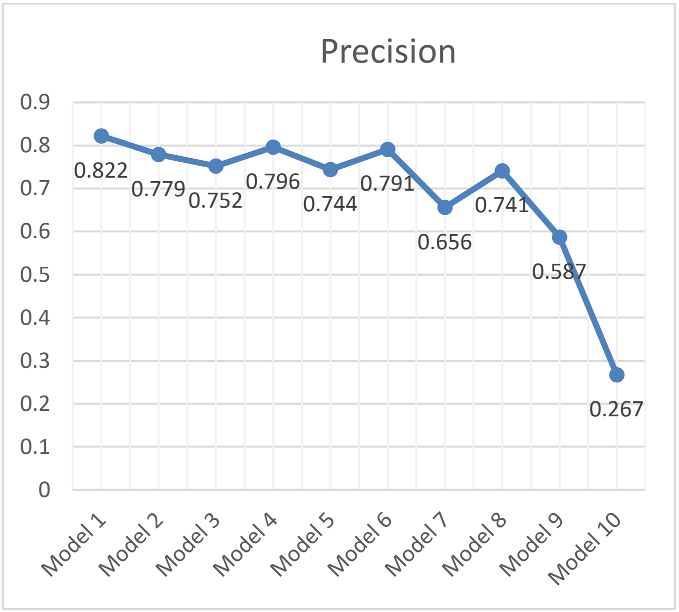


Fig 10. Precision of VGG 16 with different hyper-parameters

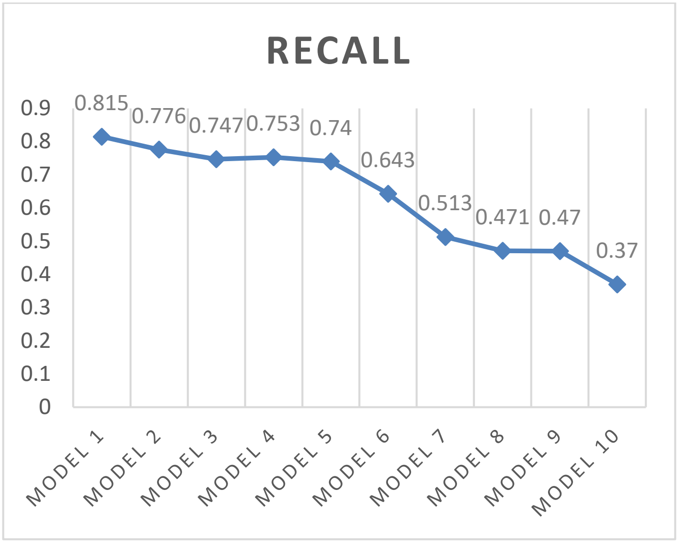


Fig 11. Recall of VGG 16 with different hyper-parameters

V. COMPARING THE BEST MODEL IN EACH ARCHITECTURE

Hyper Parameters	**Architecture**
optimizer: Adam, neurons: 512, hidden_layers: 3, freeze_layers: 15, epochs: 16, dropout_rate: 0.5, batch_size: 128, activation: sigmoid	VGG16
optimizer: Adam, neurons: 512, hidden_layers: 3, freeze_layers: 15, epochs: 16, dropout_rate: 0.5, batch_size: 128, activation: sigmoid	VGG 19
optimizer: Adamax, neurons: 128, hidden_layers: 3, freeze_layers: 65, epochs: 17, dropout_rate: 0.2, batch_size: 64, activation: sigmoid	MobileNet

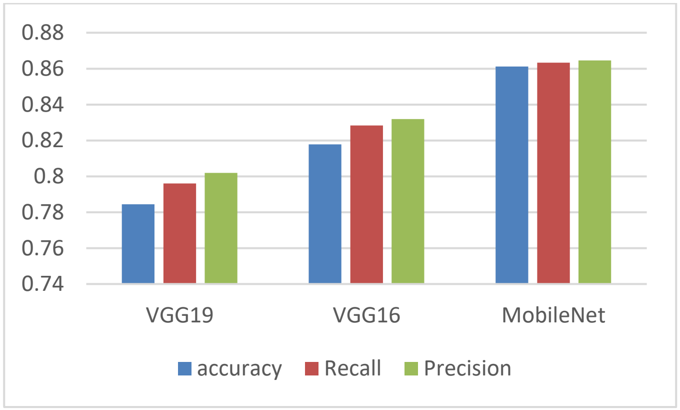


Fig 12. Accuracy, Precision and recall of VGG 16, VGG19 and Mobile Net with best hyper-parameters

A. Observation

On comparing VGG16, VGG19 and MobileNet on the testing set, it is found that accuracy, precision and recall of MobileNet is the best of all 3 architecture with the best hyper parameter. Analyzing the incorrectly labelled images, we could see they are difficult to visualize even by humans.

The activation from sigmoid and out function softmax are the hyper parameter found in all top models in all 3 architectures.

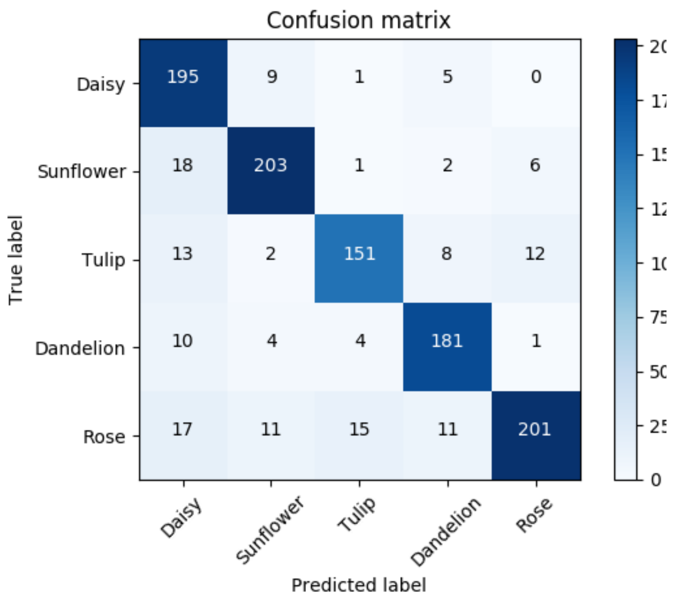


Fig 13. Confusion matrix of Mobile Net with best hyper parameters

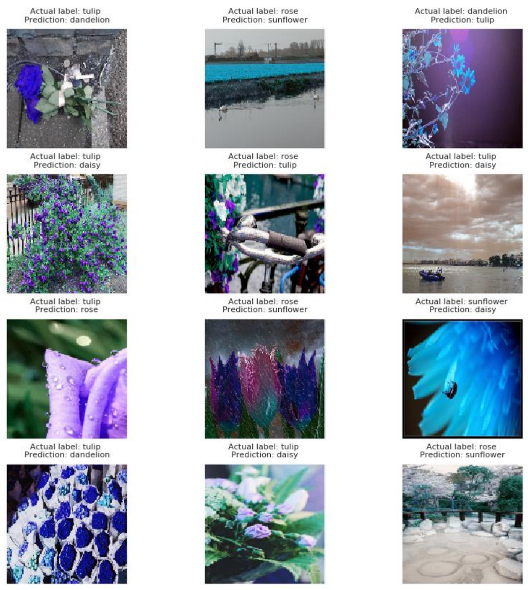


Fig 14. Incorrectly labeled images by the best predictor

B. Conclusion

From hours of training, validating and testing various CNN architecture by figuring out the best hyper parameters for each, we can conclude that MobileNet performs the best in recognizing the flower images.

References

[1] Andrew G. Howard, M. Z. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv preprint arXiv:1704.04861.

[2] Chollet. (2015). Keras. Retrieved from Keras: http://keras.io

[3] Karen Simonyan, A. Z. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. ILSVRC.

[4] Pedregosa, F. a. (2011). Scikit-learn: Machine Learning in {P}ython. Journal of Machine Learning Research.

[5] Flower Recognition – Kaggle Dataset – Alexander M https://www.kaggle.com/alxmamaev/flowers-recognition