Image Classification using Prevalent Models

Dhasarath Dhayanidhi
University of Adelaide
a1906632@adealide.edu.au

Abstract

Image Classification is one of the basic yet very popular supervised machine learning tasks. Imagenet is one such contest example, since 2010 this contest has a significant motivation for development of new architectures which perform better. The intent of this paper is to train a few of such prevalent classification models, compare them and determine any possible conclusions or patterns. The most efficient models are chosen for comparison from already tested results, and they are trained from scratch.

1. Introduction

Image classification is subfield of computer vision and deep learning where a set of labeled images are used for training and tested using images with labels retained. Imagenet is one such Image classification contest, which has been started in the year 2010, which lead to development of many such models like Resnet, Mobilenet, Shufflenet, Densenet, VGG etc. We will be focusing first on the different types of layers used on building this model and then move with the comparison of the models.

1.1. Dataset

"CIFAR-10, CIFAR-100 which are subsets of labeled 80 million tiny datasets collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton." The tiny images are collected by anonymous authors using automated scripts using words/nouns from Wordnet. "The dataset was created in 2006 and contains 53,464 different nouns, directly copied from Wordnet. Those terms were then used to automatically download images of the corresponding noun from Internet search engines at the time (using the available filters at the time) to collect the 80 million images (at tiny 32x32 resolution; the original high-res versions were never stored)." We are using the CIFAR-10 subset downloaded from University of Toronto site and then unpickled and then loaded in respective data loader.

1.2. Model Selection

Due to the given computing constraint, and to train efficient models, the models are selected based on a metric which was calculated by dividing the already known performance (accuracy) with the gflops which is a computing measure which tells how much computing the model needs do to be run). FLOPS is the floating-point operations per second. Where, GFLOPS is gigaflops calculated by,

$$GFLOPS = 10^9 * FLOPS \qquad (1)$$

An excel sheet (attached in resource folder) was created from a table of models, their accuracy and their GFLOPS loaded from Pytorch page, then the metric was calculated for the models, then model was selected. Please refer to the attached excel sheet in github for more info. Different versions of Mobilenet and Shufflenet dominated the first few entries, so these were selected and then Resnet was selected for the one of bigger models. So according to the information as of now, the three models should perform similarly as they have around 70 percent accuracy in the data.

1.3. Choice

The choice of using pretrained models vs using the model architectures and training from scratch was made by considering two things. One being, even though the models are trained on Imagenet, they have different structures leading to different parameter weight and bias values, so some might take lesser fine tuning to converge but some might take longer or more fine tuning runs, since we wanted to compare the models, we will be making sure that the training of them are same except the structure of the models and their hyperparameters like (optimizer and learning rates). Second being, there is a high chance of Imagenet's training samples ending in CIFAR-10's test set, so to make the comparisons unbiased the models will be trained from scratch.

1.4. History of CNN’s

The way image classification was done before the introduction of CNN is to take all the pixels of the image and use a deep neural network and pass them as input to the model. There were two disadvantages with the approach, one being that essential spatial information is not considered that is, the model doesn’t have the info or can’t learn the importance of nearby pixels of a pixel (pixel-pixel importance is not considered). The other being unimportant or unnecessary operations are done, since we are operating on the entire image.

In 1989, Yann LeCun and team from bell laboratories, introduced CNNs in their paper “Handwritten Digit Recognition with a Back-Propagation Network” which dealt with recognition of digits from images, which was later used in multiple places to automate digit recognition (one such use is in post offices to read postcodes to assort them accordingly). The CNNs allowed the model’s to retain the image’s 2d shape till they reach the fully connected layers. Models with CNNs performed better than ones without, since they have the added advantage of spatial information and areas of importance.

The introduction of CNNs lead to the rise of many better performing models over the years like Vit, VGG for image classification, YOLO, mask RCNN for image segmentation, etc. This leads to significant use of deep learning for computer vision tasks. Many significant products and services have arisen because of the above, like self-driving cars, facial recognition using cctv, automated billing systems in retail stores etc.

2. Architecture

There are different types of layers and operations in each of these models some being:

- Convolution layers
- Pooling layers (Max Pooling and Avg Pooling)
- Residual Connections
- Batch Normalization
- Fully Connected Neurons
- Activation Functions

Usually, the architecture is composed of the above components, starting with a few batches where each batch is made of convolution layers followed by pooling or batch normalization layers with activation functions and a flattening operation before being passed to the fully connected layer which will give us the probabilities of each class.

2.1. Convolution layer

Convolution is a mathematical operation which starts with 2 matrices, input and a filter matrix. Usually, the filter is smaller in shape, then the filter is placed on the top left corner of the input matrix, a dot product is taken and stored in a new matrix, then the filter matrix is slided horizontally and the dot product is taken again and stored in the new matrix, the same is repeated until it has reached a horizontal end, then it is repeated by making the starting point by sliding the filter vertically down from the first position (the top left), then the same horizontal sliding process is repeated again, similarly the vertical sliding process is repeated until the whole image is covered. Usually when we want to maintain the same shape between the input and output matrix, we do a 0 padding which surrounds the input matrix with a 0 element on all sides. and a stride parameter controls how many elements we slide over when sliding horizontally and vertically.

A pictorial representation can be seen in Figure 1 below. The blue part is the input image, and the green part is the output image, the dotted part is the zero padding, and the black shaded part is the filter. The same filter is maintained throughout a single convolution process. Given input image is of size (I, I) and stride S, Padding P and filter of size (F,F), the shape of output image (O, O) is given by,

$$O = \frac{I - F + (2 * P)}{S} + 1 \qquad (1)$$

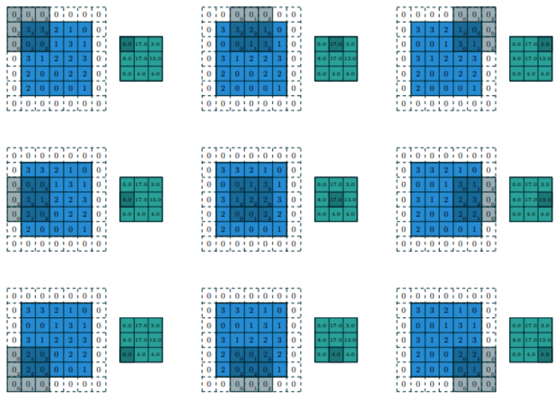


Figure 1: Convolution example with stride = 2

2.2. Pooling layer

There are two major types of pooling, max pooling and average pooling. Pooling is similar to convolution where the filter is taken and put on the input matrix but instead of taking the dot product, max pooling takes the maximum of values the filter is currently placed on, and average pooling takes the average of the values. Pictorial representation of the pooling is given below.

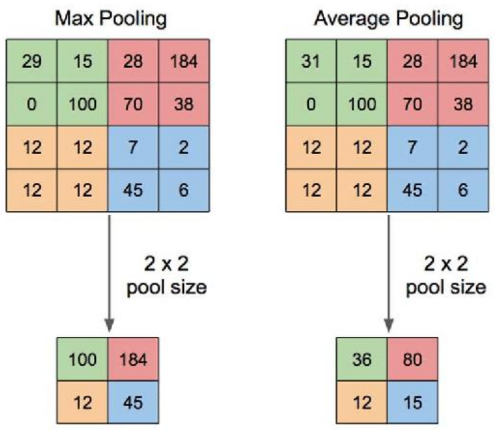


Figure 2: Pooling Layers

2.3. Residual Connections

Residual connection establishes a path for the data to reach the deeper parts of the model without being filtered out by skipping through the layers of the network. There are two types of residual connection blocks, one being bottleneck block which consists of three convolution layers and a residual connection, other being pre-activation block where the residual connection is applied after the activation function in the layers. Bottleneck blocks are used in computer vision models like resnet, while the pre activation block is used in transformers.

2.4. Batch Normalization

Batch Normalization is a technique to make the models faster by normalizing the output of a layer before passing it to the next layer. "Batch normalization applies a transformation that maintains the mean output close to 0 and the output standard deviation close to 1." During training the normalization is done using the current output it is handling, but during validation or testing the outputs will be normalized using the running average of the mean and standard deviation of the batches it has seen during training.

2.5. Fully connected layer

Fully connected layers are made up of multiple layers of where in each layer is composed of a few neurons. These layers are interconnected as we have seen in the deep neural networks. The images are flattened before its being passed to the fully connected layers as they only accept linear array as input. Usually there is only one layer in the fully connected layers in the case of CNN models, since most of the heavy lifting is done by the Convolution layers, the fully connected layer just has an output layer with the as many neurons as the number of classes.

2.6. Resnet18

Resnet18 is the 18 layered variant of the Resnet model family. Resnet was released in the year 2015, which introduced Residual Connections, and claimed that these connections will make it easier to train deeper models which was difficult before. The model was trained on Imagenet and has won 2015 ILSVRC (Imagenet Contest) & COCO.

"We present a residual learning framework to ease the training of networks that are substantially deeper than those used previously. We explicitly reformulate the layers as learning residual functions with reference to the layer inputs, instead of learning unreferenced functions"

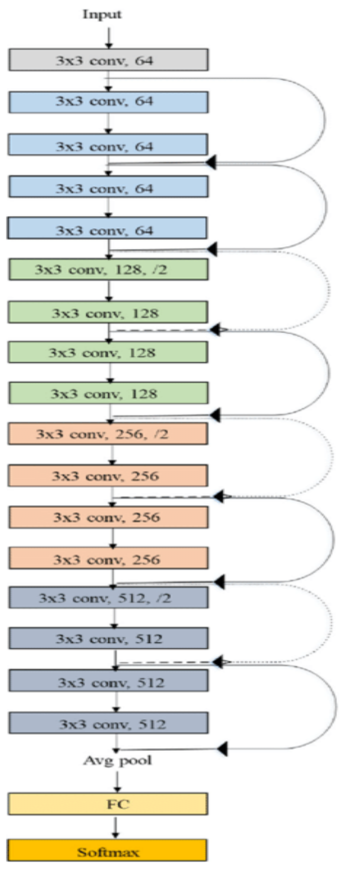


Figure 3: Resnet 18 Architecture

2.7. MobileNetV2

Mobilenet was released in 2017 with the intention of it being used in mobile (ARM devices) or low compute platforms and systems, which has lesser compute and needs the model to be efficient like embedded systems, etc. The model initialization function has two parameters which control the depth and height of the model, and it uses depth wise convolution. MobilenetV2 was released in 2018, with added inverted residuals and linear bottlenecks, which gives an improvement over the version 1. Although it should be noted that depth wise convolutions are not accelerated or designed for gpus, so Mobilenet in gpu might be slower than Resnet. We have used a Resnet with width multiplier as 0.5.

“The MobileNetV2 architecture is based on an inverted residual structure where the input and output of the residual block are thin bottleneck layers opposite to traditional residual models which use expanded representations in the input an MobileNetV2 uses lightweight depth wise convolutions to filter features in the intermediate expansion layer.”

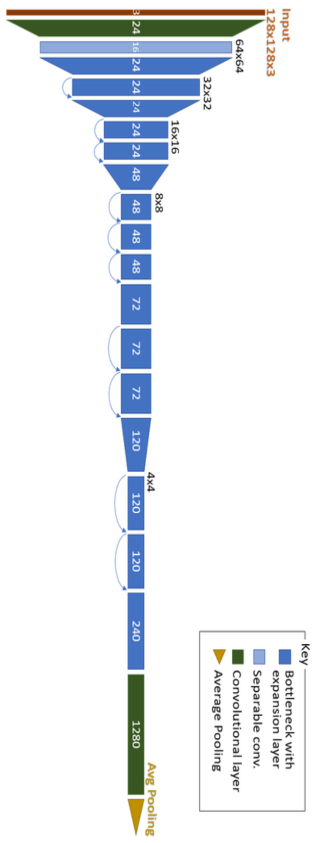


Figure 4: Architecture of a MobilnetV2

2.8. ShuffleNetV2

Similar to Mobilenet Shufflenet also aimed for being used in mobiles (ARM devices), platforms with low compute. The first version of Shufflenet was released in 2017, which uses accelerated floating-point operations, pointwise group convolution and channel shuffle to reduce the needed compute 10-150 MFLOPS (10^6*FLOPS). The main advantage with shufflenet apart from the speed is it uses channel shuffle, which allows the convolution layers to allow flow of information across the feature channels. In ShufflenetV2 they have designed models by optimizing the speed of the model rather than the FLOPS of the model since as they have said,

“Speed also depends on the other factors such as memory access cost and platform characteristics. Thus, this work proposes to evaluate the direct metric on the target platform, beyond only considering FLOPs”

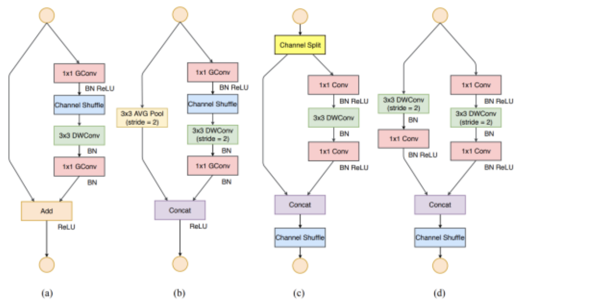


Fig. 3: Building blocks of ShuffleNet v1 [15] and this work. (a): the basic ShuffleNet unit; (b) the ShuffleNet unit for spatial down sampling (2×); (c) our basic unit; (d) our unit for spatial down sampling (2×). **DWConv**: depthwise convolution. **GConv**: group convolution.

Figure 4: Architecture of a ShufflenetV2

ShuffleNet_V2_X0_5 model was used, which has 0.5x output channels.

3. Experimentation

The experimentation was performed with the aim of achieving the best performance (accuracy) with the least amount of training (50 fixed epochs), it should be noted that some models which is deeper and wider than others would need lesser learning rates which would lead to slower convergence, so the results might be more accurate if trained till convergence, since there was a constraint on the compute, a fixed 50 epochs was used to train the final versions of each model.

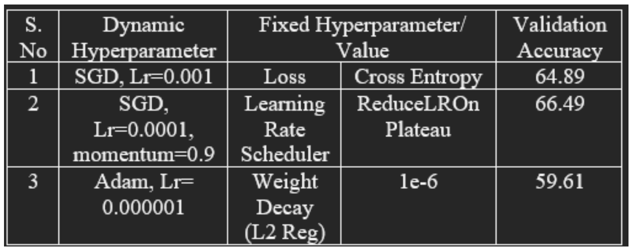
For each of the models (Resnet, Mobilenet, Shufflenet), 3 different sets of hyperparameters were tested with were experimented with,

1. SGD without Momentum
2. SGD with Momentum
3. Adam

S. No	Fixed Hyperparameter	Parameter Used
1	Loss	Cross Entropy
2	Learning Rate Scheduler	ReduceLROnPlateau
3	Weight Decay (L2 Reg)	1e-6

Figure 5: Fixed Hyperparameters for each model

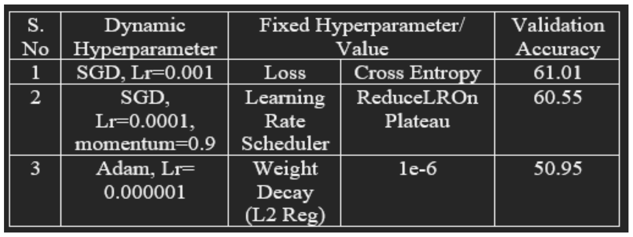
The training process is to take ach of such models is trained for 30 epoch then the training and validation accuracies were monitored manually, if any overfitting were found the model was halted and made changes to learning rate and momentum and scheduler parameters were changed till it reached a better point or to an agreeable loss curve. Only Adam and SGD are used since they are generally the best performing optimizers and since the models were huge it's better to go with the trusted optimizers.



S. No	Dynamic Hyperparameter	Fixed Hyperparameter/ Value		Validation Accuracy
1	SGD, Lr=0.001	Loss	Cross Entropy	64.89
2	SGD, Lr=0.0001, momentum=0.9	Learning Rate Scheduler	ReduceLROn Plateau	66.49
3	Adam, Lr= 0.000001	Weight Decay (L2 Reg)	1e-6	59.61

Figure 6: Resnet Model Results

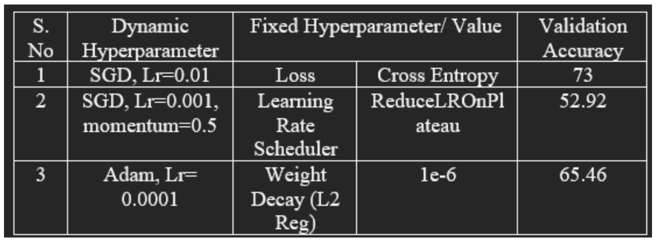
Of the above models, the 2nd model was selected as it had better validation accuracy and lesser overfitting from the loss curves, the same process of training the model will be repeated for the final model but with 50 epochs.



S. No	Dynamic Hyperparameter	Fixed Hyperparameter/ Value		Validation Accuracy
1	SGD, Lr=0.001	Loss	Cross Entropy	61.01
2	SGD, Lr=0.0001, momentum=0.9	Learning Rate Scheduler	ReduceLROn Plateau	60.55
3	Adam, Lr= 0.000001	Weight Decay (L2 Reg)	1e-6	50.95

Figure 7: Mobilenet Model Results

Of the above models, model 2 was selected based on the validation accuracy, although all of them had similar loss curves, model 3 was rejected since it was converging slower. Since model 1 and 2 had similar validation accuracy, model 2 was chosen since there were lesser fluctuations in the loss curves.



S. No	Dynamic Hyperparameter	Fixed Hyperparameter/ Value		Validation Accuracy
1	SGD, Lr=0.01	Loss	Cross Entropy	73
2	SGD, Lr=0.001, momentum=0.5	Learning Rate Scheduler	ReduceLROnPl ateau	52.92
3	Adam, Lr= 0.0001	Weight Decay (L2 Reg)	1e-6	65.46

Figure 8: Shufflenet Model Results

Of the above models, model 1 was chosen considering the validation accuracy, it was starting to overfit around the 15th epoch, the Lr scheduler parameters were changed accordingly.

Model	Validation	Test
Resnet (2)	72.16	71.27
Mobilenet (2)	67.28	66.91
Shufflenet (1)	68.48	68.21

Figure 9: Final Model Results

4. Conclusion

From the above experimentations, we have observed that the test and the validation accuracies are similar for each model. There was a bit of overfitting for Resnet starting around epoch 20, but for the other models there weren't any significant overfitting.

From the above-mentioned architecture of the models, the resnet is a deeper and more compute hungry model, while Mobilenet and shufflenet are built for lesser compute needing platforms. So, if the compute is not an issue Resnet will be preferred but if limited by compute Shufflenet is preferred since it has better performance when compared with Mobilenet and has been optimized to be run better on ARM and mobile devices. Although this study was concluded above, it can be further improved by better hyper-parameter tuning, and training for further epochs which might reveal a different set of results. Since we were constricted compute wise and time wise, we finish the study at this point.

Code: It is available in the same repository as assignment 1.

5. References

[1] CIFAR, "University of Toronto," [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html.

[2] TinyImages. [Online]. Available: https://groups.csail.mit.edu/vision/TinyImages/.

[3] GFLOPS. [Online]. Available: https://en.wikipedia.org/wiki/Floating_point_operations_per_second.

[4] Pytorch. [Online]. Available: https://pytorch.org/vision/main/models.html.

[5] h. o. cnns. [Online]. Available: https://towardsdatascience.com/the-history-of-convolutional-neural-networks-for-image-classification-1989-today-5ea8a5c5fe20.

[6] "Yann leCun," [Online]. Available: https://proceedings.neurips.cc/paper/1989/file/53c3bce66e43be4f209556518c2fcb54-Paper.pdf.

[7] C. Image. [Online]. Available: https://arxiv.org/pdf/1603.07285.

[8] M. p. Image. [Online]. Available: https://www.researchgate.net/figure/llustration-of-Max-Pooling-and-Average-Pooling-Figure-2-above-shows-an-example-of-max_fig2_333593451.

[9] B. Normalization. [Online]. Available: https://keras.io/api/layers/normalization_layers/batch_normalization/.

[10] Resnet. [Online]. Available: https://www.researchgate.net/figure/Original-ResNet-18-Architecture_fig1_336642248.

[11] Resnet. [Online]. Available: https://arxiv.org/abs/1512.03385.

[12] Mobilenet. [Online]. Available: https://arxiv.org/pdf/1704.04861.

[13] Mobilenet. [Online]. Available: https://arxiv.org/abs/1801.04381.

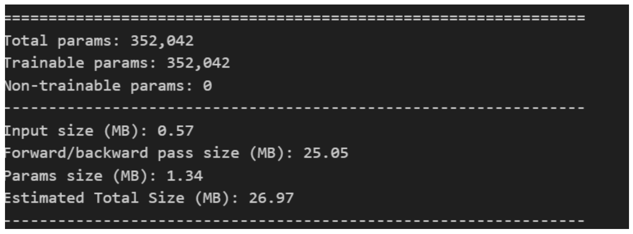
[14] Shufflenet. [Online]. Available: https://arxiv.org/abs/1707.01083.

[15] Mobilenet. [Online]. Available: https://arxiv.org/abs/1807.11164.

[16] Mobilenet. [Online]. Available: https://paperswithcode.com/method/shufflenet-v2.

[17] C. Shuffle. [Online]. Available: https://paperswithcode.com/method/channel-shuffle.

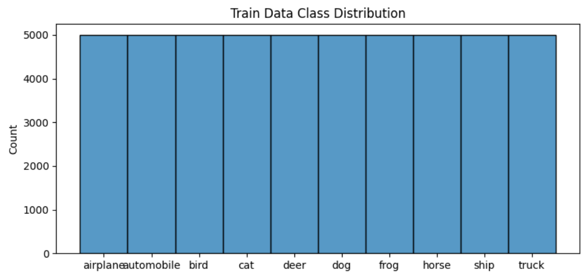
6. Appendix

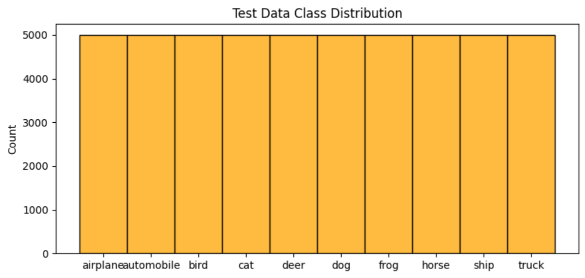


```
================================================================
Total params: 352,042
Trainable params: 352,042
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 25.05
Params size (MB): 1.34
Estimated Total Size (MB): 26.97
----------------------------------------------------------------
```

ShufflenetV2_X_0_5 -- model specifications

Please refer to the resources folder for more information.

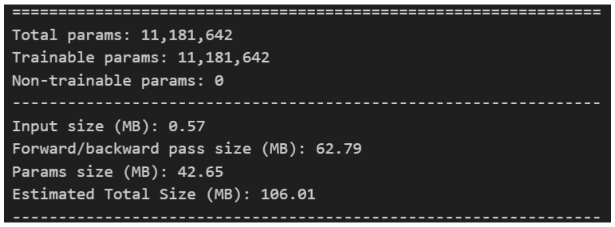




Class Distribution

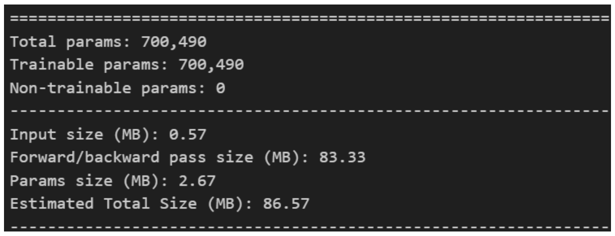
Train Samples:['automobile', 'truck', 'automobile', 'ship', 'frog']

Example Images



```
================================================================
Total params: 11,181,642
Trainable params: 11,181,642
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 62.79
Params size (MB): 42.65
Estimated Total Size (MB): 106.01
----------------------------------------------------------------
```

Resnet 18 -- model specifications



```
================================================================
Total params: 700,490
Trainable params: 700,490
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 83.33
Params size (MB): 2.67
Estimated Total Size (MB): 86.57
----------------------------------------------------------------
```

MobilenetV2 -- model specifications

S. No	Dynamic Hyperparameter	Fixed Hyperparameter/ Value		Validation Accuracy
1	SGD, Lr=0.001	Loss	Cross Entropy	64.89
2	SGD, Lr=0.0001, momentum=0.9	Learning Rate Scheduler	ReduceLROn Plateau	66.49
3	Adam, Lr= 0.000001	Weight Decay (L2 Reg)	1e-6	59.61

Resnet Metrics

S. No	Dynamic Hyperparameter	Fixed Hyperparameter/ Value		Validation Accuracy
1	SGD, Lr=0.001	Loss	Cross Entropy	61.01
2	SGD, Lr=0.0001, momentum=0.9	Learning Rate Scheduler	ReduceLROn Plateau	60.55
3	Adam, Lr= 0.000001	Weight Decay (L2 Reg)	1e-6	50.95

Mobilenet Metrics

S. No	Dynamic Hyperparameter	Fixed Hyperparameter/ Value		Validation Accuracy
1	SGD, Lr=0.01	Loss	Cross Entropy	73
2	SGD, Lr=0.001, momentum=0.5	Learning Rate Scheduler	ReduceLROnPl ateau	52.92
3	Adam, Lr= 0.0001	Weight Decay (L2 Reg)	1e-6	65.46

Shufflenet Metrics