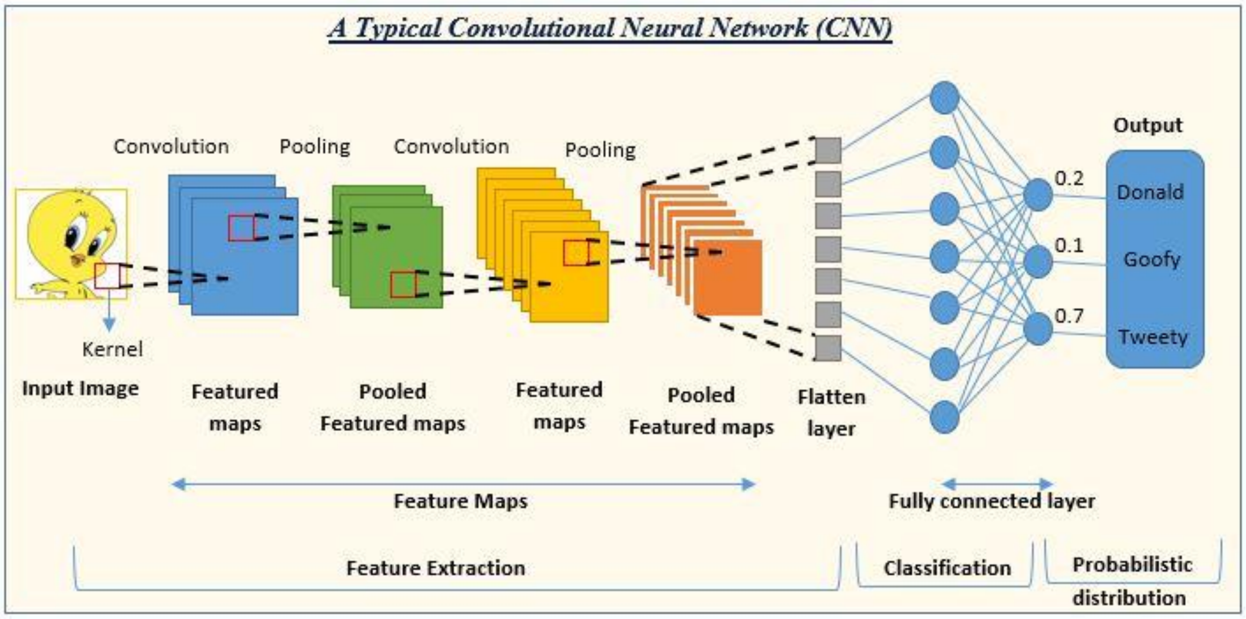
Project Final Report: MobileNets – Efficient Convolutional Neural Networks for Mobile Vision Applications

Section 1: Task Definition, Evaluation Protocol and Data

Ever since AlexNet [1] Convolutional Neural Networks (CNNs), computer vision applications have evolved immensely so that the algorithm is ubiquitous in the machine learning community. This neural network architecture trains parameters ranging in millions to bring the state-of-the-art applications [2 ,3 ,4 ,5] into existence. In this report we address the problem of achieving this task using light-weight model derived from CNNs empowering many real-world computers vision applications that have lesser computational resources, using MobileNet architecture [6].

Convolutional Neural networks, also known as shift invariant neural networks, make use of weighted convolutional kernels or filters by sliding the kernel across the input feature and producing translated feature maps. Since the kernels are usually much shorter in size it breaks down the input into smaller and simpler structures, helping in avoiding over-fitting compared to multi-layer perceptron. The translation is done multiple in a hierarchical manner accommodating down sampling the input and finally using a fully connected layer to produce outputs.



We have referred to the work done in [6], to build the model and evaluate it across a range of datasets addressing tasks like Image recognition. The datasets we have planned to use in the process of training and later testing the model are Stanford Dogs [8], CIFAR-10 [9]. Since the model is targeted to be Mobile, we will be evaluating the model based on the accuracy achieved with respect to the number of parameters learned in the training process. This evaluation will also help us in defining a trade-off between computational cost and performance.

Section 2: Neural Network Machine Learning Model

CNNs spend most of the computation in the convolution operations using kernels or filters of producing features ranging in varied sizes and vast depths. These operations ensure two things, filtering and combining. Filtering occurs in the sliding operation whereas the combining happens during the weighted multiplication done across the total depth of the input.

If the standard input to the convolutional layer is of dimensions D_F x D_F x M, where M is the depth of the input and rest is the spatial size of the input. On using a K size convolutional kernel, with N number of instances, a convolution operation can be shown in the below image 2.1. The respective computational cost can be calculated as shown in image 2.2.

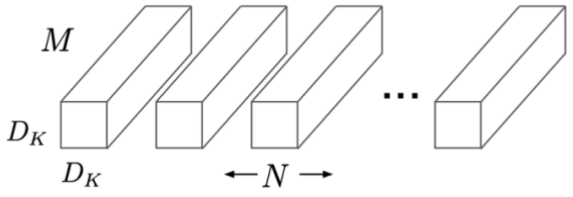


Image 2.1: Standard Convolution Kernel

$$D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F$$

Image 2.2: Computational cost

MobileNet saves computation by tackling the tasks, discussed in first paragraph, separately by introducing depthwise convolutions and pointwise convolutions. Depthwise convolutions (Image 2.3), introduced in [7] also used in Inception model [8], are general convolution but use only one filter per input depth, i.e., one-to-one mapping between an input channel and filter with no aggregation of features collected at each depth. Whereas pointwise convolutions (Image 2.4) are convolution operations involving 1x1 kernel helps in combining the output formed from depthwise convolutions. Accordingly, Image 2.5 and 2.7 show the computational cost associated with each operation.

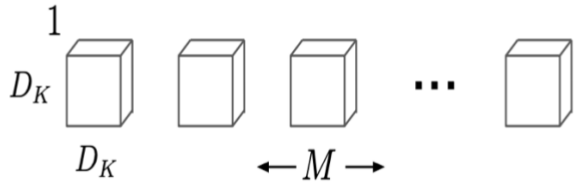


Image 2.3: Depthwise Convolution kernel

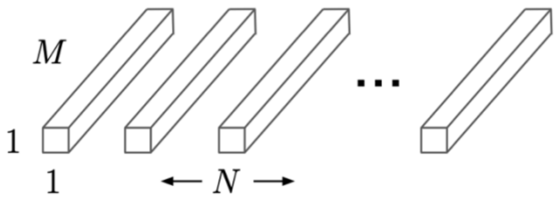


Image 2.4: Pointwise Convolution kernel

$$D_K \cdot D_K \cdot M \cdot D_F \cdot D_F$$

Image 2.4: Depthwise computational cost

$$M \cdot N \cdot D_F \cdot D_F$$

Image 2.6: Pointwise computational cost

Summing up together, the computational saved on using mobilenets can be shown through the below equation. The most expensive component here is the D^2_K, i.e., on using a kernel of 3 x 3 size the mobilenets would utilize nearly 8 to 9 times less cost than standard convolutions.

$$
\begin{aligned}
& \frac{D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F}{D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F} \\
= \quad & \frac{1}{N} + \frac{1}{D_K^2}
\end{aligned}
$$

Image 2.7: Comparing computational cost between standard convolution and combination of depthwise, pointwise convolution operations.

The architecture also introduces two new hyperparameters – width multiplier (α), resolution multiplier (β). Width multiplier gives the flexibility of controlling the depth of the kernel used in the point-wise convolution, also the depth of the output feature produces i.e., M becomes αM. Resolution multiplier helps control the size of the input and the subsequent reorientations formed by layers i.e., using the parameter the spatial size of the input will be represented as βD_F x βD_F. Altogether, by tuning these two we reduce the computation pertaining to number of parameters and number of multiplication and addition operations. Width multiplier will help us in proving the amount of computation that can be reduced in terms of number of weights whereas the resolution multiplier in terms of number of multiplication and addition operations.

Each convolutional layer will accommodate all the elements discussed above, making multiple changes to a standard convolutional layer. The Mobilenet architecture two regularization techniques i.e., Dropout and Batch Normalization. Dropout helps avoiding overfitting by “dropping off” a fraction of neurons from the forwards passes and the weight updates, this operation is only utilized during the training phase. Whereas Batch Normalization ensures that each layer receives an input which that a zero mean and unit variance by normalizing the activations produced at previous layers. As the name reflects, this operation is performed at every mini batch. Lastly, each layer except the final layer will be using ReLU activation functions. Altogether, the design each layer can be found in the Image 2.9.

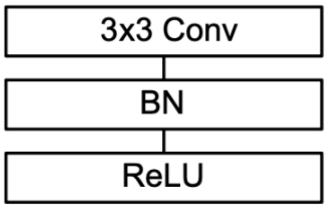


Image 2.8: Standard Convolutional layer

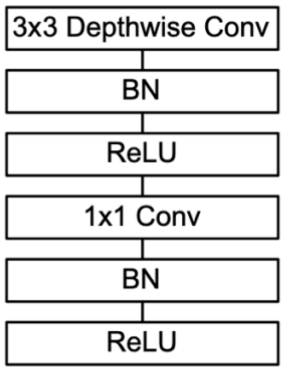


Image 2.9: Mobilenet Convolutional layer

Section 3.1: Experiment Design

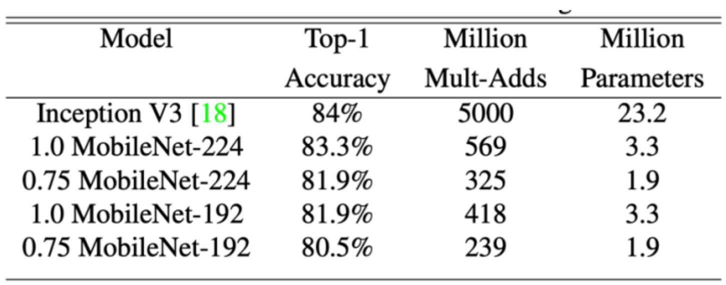
- **Research Question:**
    - To what degree we can expect to lose performance on using MobileNets?
    - Also, know what degree of impact is made by the multiplier hyperparameters in terms of computation cost and performance achieved.
- **Hypothesis:**
    - Using MobileNets, a CNN architecture produces results comparable to state-of-the-art models, using much lesser parameters and computations.
- **Variables:**
    - **Independent Variables:** For the independent variables we have 6 hyperparameters, namely:
        1. Input Shape: Optional shape tuple, describing the size of the input image. Two formats are allowed – (width, height, depth) e.g. (224,224,3) or (depth, width, height) e.g. (3, 224, 224). Not required if the input tensor is provided.
        2. Width multiplier (alpha): In a layer with M input channels and N output channels, applying the width parameter alpha results in alphaM input channels and alphaN output channels. Alpha belongs in the range (0, 1] with alpha = 1 being the baseline MobileNet.
        3. Depth multiplier (rho): Defaulted to 1, used to change the spatial size of the input feature at every layer by modifying the resolution.
        4. Initial learning Rate: Determines the step size taken by the optimization algorithm during the initial stages of training a machine learning model.
        5. Beta_1: The exponential decay rate for the first moment estimates, often denoted as "beta_1," controls how much weight is given to past gradients when computing the running average.
        6. Beta_2: The exponential decay rate for the second moment estimates, often denoted as "beta_2," controls how much weight is given to past squared gradients when computing the running average.

    - **Control Variables:** In our experimental setup, 1 control variable was defined, which is the freeze_conv parameter in the final code associated with this report. When set to True, it freezes the bottom convolutional layers, i.e., renders them untrainable. Only the top 5 dense layers are thus used in model training.

    - **Dependent Variables:** We identify three dependent variables, namely, testing accuracy, total parameters, and the learning curve.
        1. Testing Accuracy: The percentage of test samples correctly classified vs the total number of test samples. Tells the authors how optimally the weights were updated during the training process.
        2. Total Parameters: A variable that is dependent on the width multiplier parameter. Closely tied to our research question vis-a-vis computational cost. It is also used to test our hypothesis.
        3. Learning Curve: With respect to this experiment, it tells the authors whether there is overfitting. It is also useful to determine the optimal number of epochs for which the model should be trained, to train the models faster.

Section 3.2: Experiment Methodology

- **Baseline:** For the baseline results, the authors used the classification accuracy results of the model tested on the Stanford Dogs dataset, which were published originally in the MobileNets paper [6]. Below are the baseline results



Model	Top-1 Accuracy	Million Mult-Adds	Million Parameters
Inception V3 [18]	84%	5000	23.2
1.0 MobileNet-224	83.3%	569	3.3
0.75 MobileNet-224	81.9%	325	1.9
1.0 MobileNet-192	81.9%	418	3.3
0.75 MobileNet-192	80.5%	239	1.9

  This baseline was chosen because, given the limited compute resources available, this result was the most reproducible by our experiment. It also helps us answer our research question which had to do with the effect of changing multiplier hyperparameters on model accuracy.

- **Implementation Details:**
  - The Tensorflow implementation of MobileNets version 1 was used to build our model. The implementation itself was not changed but imported as a module in our code.
  - The preprocessing on the Stanford Dogs dataset was done using built-in Tensorflow methods. Below is an exhaustive list of the preprocessing methods used.
    - Tf.image.resize; Tf.keras.applications.mobilenet.preprocess_input; Map; Batch; Prefetch
  - Like section 4.3 of the original MobileNets paper [6], the authors use a model which is pretrained on a dataset much larger than Stanford Dogs and then is fine-tuned to perform particularly well on our dataset. This is achieved by loading the model with weights=imagenet, which loads a model pretrained on ImageNet dataset.
  - The authors keep the convolutional layers frozen, i.e., untrainable during our experiment to benefit from transfer learning. The feature extraction part is handled by the top 5 layers of our model, which are the dense layers.

Section 4: Experimental Results and Discussion

Overall, the experiment ran for 219.45 minutes, with each model being trained for 30 epochs. The experiment was implemented on Tesla T4 GPU's. The exact number of processing units is not known at present. Accuracy, model size and learning curves were the metrics collected.

Section 4.1: Accuracy vs Model Size

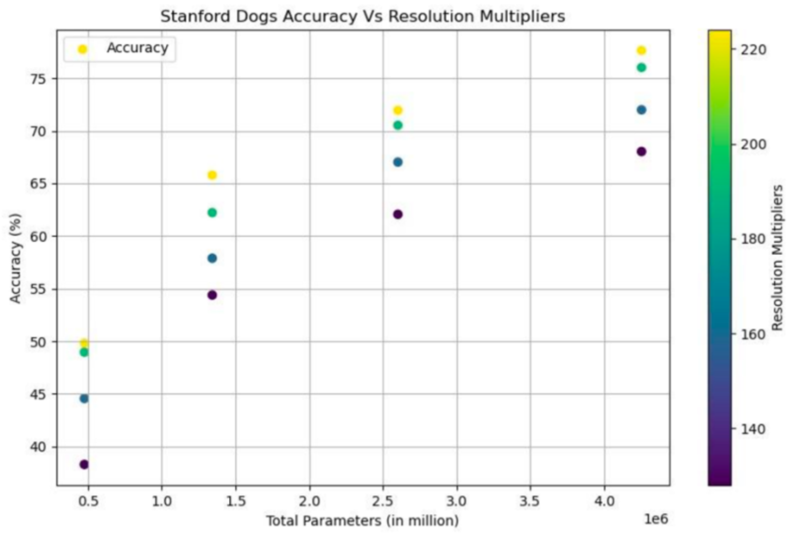


Figure 4.1

Research Question: What degree of impact is made by the multiplier hyperparameters in terms of computation cost and the performance achieved?

Model accuracy drops as the model size, expressed in million parameters, drops. For a given constant model size, the accuracy is directly proportional to the resolution of the image, so as the resolution drops so does the accuracy. This is because as the resolution is decreased, the internal representation of each layer reduces [6]. From the figure it can also be inferred that a reduction in width multiplier reduces the computational cost of the model by decreasing the number of parameters in the model.

Comparing our results against the specified baseline in Section 3.2 produces the following table:

Model name	Width multiplier	Resolution multiplier	Parameters (Million)	Top-1 Accuracy (%)
Inception V3	-	-	23.2	80.3
mobilenet_1.00_224	1.00	224	4.25	77.6
mobilenet_1.00_192	1.00	192	4.25	76.0
mobilenet_0.75_224	0.75	224	2.60	71.9
mobilenet_0.75_192	0.75	192	2.60	70.6

Research Question: To what degree can we expect to lose performance on using MobileNets?

The authors trained an instance of the Inception V3 model on Stanford Dogs dataset to gauge the comparable performance with the baseline (see section 3.2). With respect to the baseline, there is a 6 - 10% divergence in the test accuracies. This can primarily be attributed to different datasets used to pretrain the model in the original paper vs. our model. Our model is pre-trained on the ImageNet dataset, whereas the original paper uses a modified version of the dataset used in [9]. This variation in the initial weights used makes an impact on how optimally the convergence takes place.

Section 4.2: Learning Curve

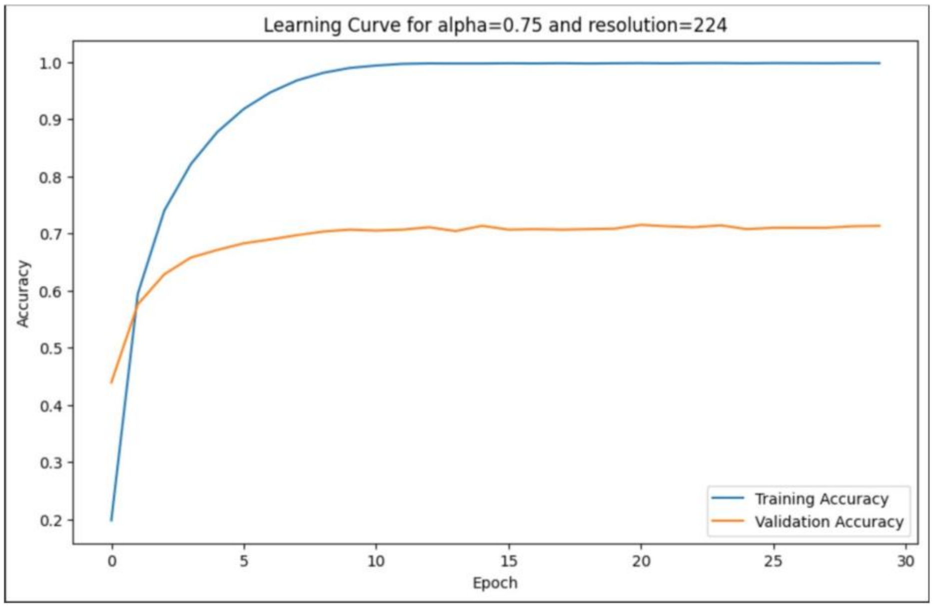


Figure 4.2

Figure 4.2 shows the learning curve achieved by the MobileNet model with width multiplier=0.75 and resolution=224. As per the observation, the model accelerates the learning in the first 7 epochs and later takes small steps in the process of finding the optimal minimal cost. This is where our dynamic learning rate helps the model in keeping the learning progressive. In case the learning rate was not updated, the gradients would have overshoot causing the model to overfit, resulting in steep drops in validation accuracies.

The learning on the training has been consistent after the 10th epoch, i.e., nearly 100% accuracy, on the other hand, the validation accuracy has been struggling to find a steep learning. Again, we believe the learning rates being further less could have provided an extra boost to the validation accuracies. But this would have required us to run the model for more epochs, meaning, creating a need for more compute resources. Even further degrading the learning rate, would have also, caused the vanishing gradient problem.

Section 5: References

1. *Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, pages 1097–1105, 2012*
2. *K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014.*
3. *C.Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. arXiv preprint arXiv:1512.00567, 2015.*
4. *Szegedy, S. Ioffe, and V. Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. arXiv preprint arXiv:1602.07261, 2016.*
5. *K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. arXiv preprint arXiv:1512.03385, 2015.*
6. *Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv preprint arXiv:1704.04861,2017.*
7. *L. Sifre. Rigid motion scattering for image classification. PhD thesis, Ph. D. thesis, 2014.*
8. *S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167, 2015.*
9. *J. Krause, B. Sapp, A. Howard, H. Zhou, A. Toshev,T. Duerig, J. Philbin, and L. Fei-Fei. The unreasonable effectiveness of noisy data for fine-grained recognition. ArXiv preprint arXiv:1511.06789, 2015.*