Image Classification - Comparing CNN models on CIFAR for Fast Response Applications

Abstract— **The second decade of the 21st century has a plethora of high tech features embedded within the existing technology products and solutions to enhance their performance thereby making it more user centric. One such good example can be found in e-vehicles, automatic or AI (Artificial Intelligence) driven vehicles. With the use of modern neural network architectures, additional features can be embedded into an AI driven vehicle to bring them close to the way humans drive. This article involves a proposition of one such neural network architecture that can recognize (image classification) real world objects or images to make intelligent decisions while driving an automatic vehicle on the streets. It is intended to solve problems like humans responding to certain traffic signs, symbols, other adjacent vehicles or even living beings. It is just a proposition that can be extended to better utility based on a good dataset. The dataset used for this purpose is CIFAR-10 due to limitations on computation capabilities (original intention was to use traffic signs and symbols dataset). A better dataset can provide much better results in the overall application of neural network architecture (probably a more tailored dataset to recognize traffic signs, symbols, other adjacent vehicles or even living beings). This application can also be used in human driven vehicles to guide them about their surrounding environment on the street.**

There is a wide variety of image classification models out there that can classify objects to a certain label which will further be used by higher level technology (e-vehicles or automatic driving application) to infer a label and act upon it accordingly in a real world scenario (fast-response to input data). To actualize this idea we have done a comparison between different image classification model architectures and this paper represents the study of ResNet and MobileNet models that suit best for the overall application scenario in order to understand what kind of model is adaptable for their usability in vehicle applications.

Keywords*—–*CIFAR, ResNet, MobileNet, auto-driver modules

I. Introduction

Nowadays, image classification finds its application in many different technologies surrounding us. Well it obviously includes mobile phones, but it is also a good application for intelligent surveillance, optical character recognition and many more. That being said there also exists models that provide efficient results for the same. However, every application has different requirements. The performance of applications depends on these models. Like in case of autonomous driver modules that work on the input image they capture in real time. The image classification model becomes a bottleneck over here because we need the module to stop when it moves around too close to the people in its vicinity. We can't expect humans to be mindful of such scenarios when it can be automated with a fast response system.

We have compared different models during our selection for potential solutions to the problem statements and as part of the project, selected a few of them to work out their results for the sake of observing learning trends as a result of various techniques. These strategies have been discussed in more detail further in the report. It includes hyperparameter tuning, data visualization, transfer learning and so on. There was also some playing around to do with the type of optimizer and the inflections that we observed by changing them.

We were motivated to specifically do this kind of image classification project after looking at the food delivery robot that moves on footpaths alongside the street amongst people (there is one in my University campus) but it's not swift in its moves which causes it to move slower than what it should. To understand this behavior we need to revolve using inference time of any model which indirectly depends on the kind of family that architecture belongs to. In fact, we intended to use a Traffic dataset that would have completely brought down the model results to the level of application for automated driven modules (even at a smaller level) . But then we encountered the problem of computational limitations and resource crunch which made us look for something that has been standard for a dataset - CIFAR10 and CIFAR100.

Not to be over-optimistic but we believe such models will find their application in autonomous cars where the Artificial Intelligence module will be responsible to drive cabs and therefore also responsible for the safety of its surroundings. The conclusion of this report explains very well our reasons for choosing the model mentioned later. There has been a mention of proposing a lighter model by cutting down on standard models.Its because standard datasets are generic, the results were tailored to simply fast response models instead of being more specific like the food delivery robot and others.

II. Related Work

In [22], Ziyong Feng et al propose learning deep neural networks using max margin minimum classification error. Most deep neural networks use loss in cross-entropy and softmax regression to calculate the loss function for optimization. But this only raises the accuracy of true outputs while not affecting the false outputs in any way. In this paper, they proposed a new max margin classification error based on the traditional minimum classification error. This new technique not only increases the accuracy of true output classes but also reduces the accuracy of other false classes covering any shortage of cross-entropy loss. This new technique is tested on two popular deep learning datasets, MNIST and CIFAR 10. Results showed better performance.

In [21], Xin M Wang et al uses max margin minimum classification error (M3CE) to propose a new training method based on the analysis of squared error in backpropagation algorithm. Then M3CE was analyzed and combined with cross entropy to obtain better results. Results were tested on standard deep learning databases, MNIST and CIFAR 10.

In [20], Q Xiang et al, created a fruit classifier for robot fruit picking. The classifier was based on SSD mobilenet v2 architecture which is a light weight model for classification. They used a pre-trained SSD mobilenet model trained on imagenet dataset and they custom tuned it to their requirements by adding a conv2d layer and adding a softmax function. As a result they achieved an accuracy of 85.17% accuracy.

III. Datasets

A. CIFAR-10

To train the network, we made use of the CIFAR-10 dataset Fig.1. There are 60000 examples each of size 32x32 color image. Moreover, the 60000 examples are subdivided into train and test datasets having size 50000 and 10000 images respectively. There are 10 classes in the dataset. Some of the classes are airplane, horse, truck ship, TV, fridge etc.

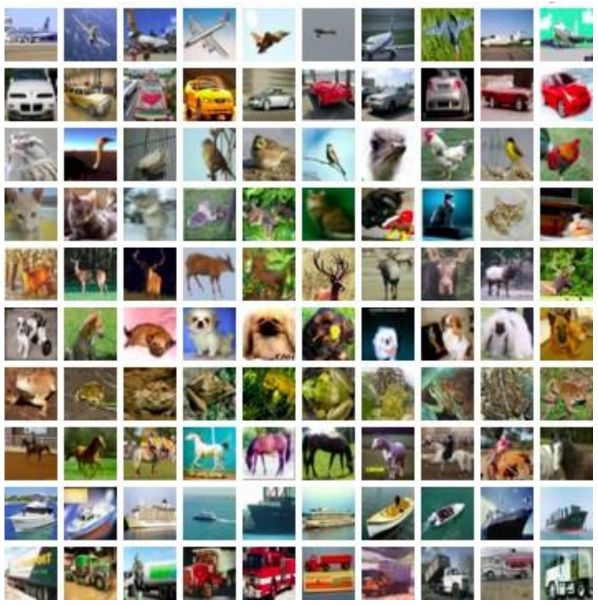


Figure1. A sample batch from the CIFAR 10 dataset [5]

B. CIFAR-100

We trained the network using the CIFAR-100 dataset Fig.2, which again contains 60000 examples with size of 32x32 pixels. There are 100 classes in the dataset. This dataset is again divided into 50000 train examples and 10000 test examples. Some of the classes are bridge, castle, table, lamp etc.

Figure2. A sample batch from the CIFAR 100 dataset

C. EDA and PCA

1. EDA (Exploratory Data Analysis)

In a data science project or data analysis project EDA(Exploratory Data Analysis) is an important step because it is a process of investigating for discovering different patterns in a dataset. It is also a process for discovering anomalies and to form hypotheses on our dataset. For numerical data, Exploratory Data Analysis gets the summary statistics of this type of data. This is mainly done in order to better understand the data.

2. PCA(Principal component Analysis)

Without any information loss, PCA Fig.3.1 represents the data in lower dimensions by calculating the dependencies between the input set of variables. For data exploration, it is one of the best tools out there and it is widely used along with EDA as explained before.

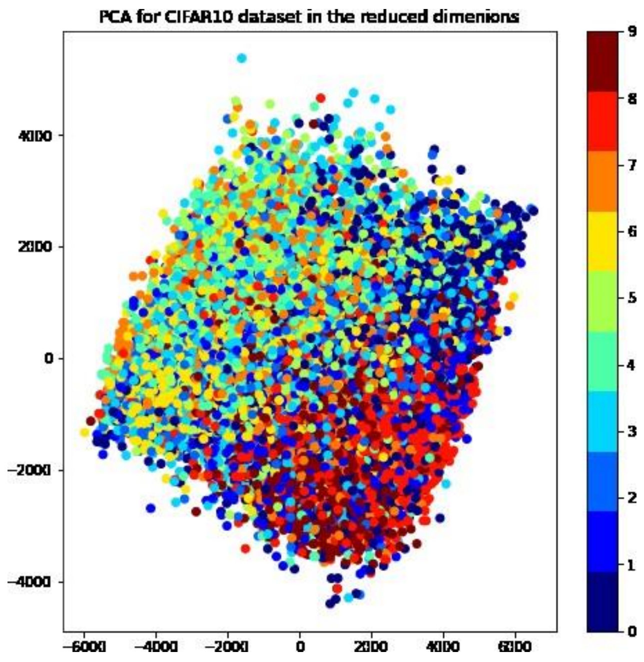


Figure 3.1 PCA for CIFAR10 dataset

3. t-SNE(Distributed Stochastic Neighbor Embedding)

t-SNE Fig.3.2 is a state of the art algorithm used for dimensionality reduction. t-SNE works by reducing higher dimensionality data into 2 or 3 dimensions. The way it works is by computing affinities between points which maintains these affinities in lower dimensional space.

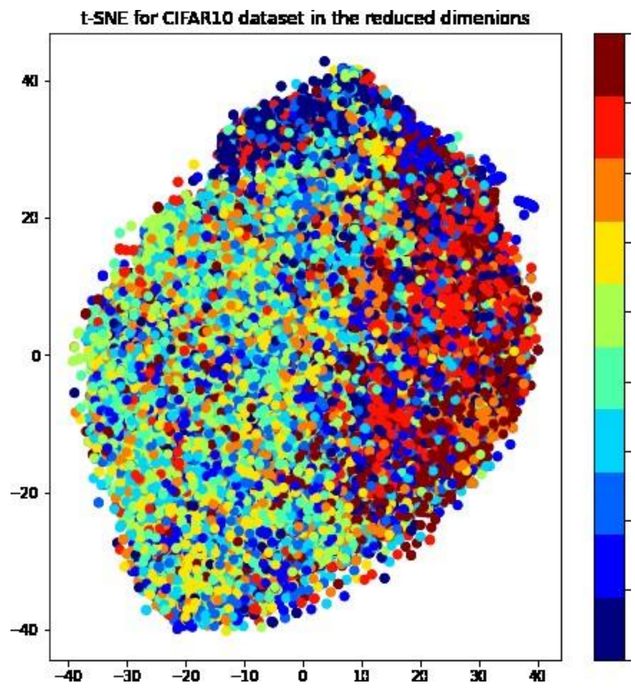


Figure 3.2 t-SNE for CIFAR10 dataset

IV. Implementation

A. *Convolutional Neural Network*

Convolutional Neural Networks or better known as CNNs takes image as input and pays attention to various features in the image which also differentiates the difference between the object in an image. Compared to any other classification models, we have less preprocessing to perform.

Similar to neurons in a human brain, the CNN gains inspiration from Visual Cortex in the human brain. A receptive field is a restricted region of the visual field which responds to stimuli from individual neurons.

Through relevant filters, Temporal and spatial dependencies are successfully captured by the ConvNet. Reduction of the parameters are involved which better fits the image dataset. Also can be said as, the more the sophistication of the image the better the CNN can understand.

B. *LeNet Architecture*

A classic Deep Neural Network Architecture Fig.4 created in 1998 for the purpose of identifying and classifying images.
LeNet uses a series of blocks each comprising two parts. First being an encoder block made from two convolutional layers. The second part consists of a dense block of three FC layers.

At the very basic level each block is made with a sigmoid activation function, a convolutional layer and an avg. pooling layer. Once our input data passes through all these layers, it needs to be flattened before passing on to the dense block, hence we have to transform the block output to a two dimensional input which can be taken in by the fully connected layer.

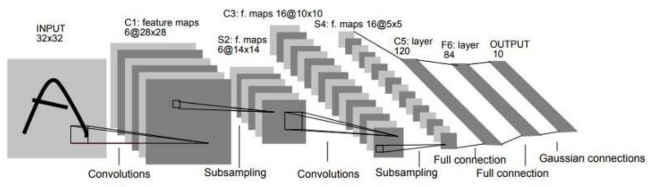


Figure 4. The LeNet Architecture.

[23] LeNet comprises around 60,000 trainable parameters and was really quick when we implemented it using Pytorch and trained it on GPU.
The only problem that were observed while working with CIFAR 10 and after optimizing the hyper parameters was that the model was not reaching really great accuracy results and that was because the architecture of LeNet is good when it comes to handling GreyScale images but doesn't perform really well when it comes to coloured images as it is not deep enough to handle colored images and hence it learns low to medium level features like edges and shapes but doesn't work really well to identify objects.[24]

C. *ResNet Architecture*

[25] [26] As Deep Neural Networks become really popular, people started thinking that by increasing the number of layers of a model, we can train just and achieve good results in just about any image classification task, but soon enough the myth behind this got over as with large number of layers came the Vanishing and Exploding Gradient problem.

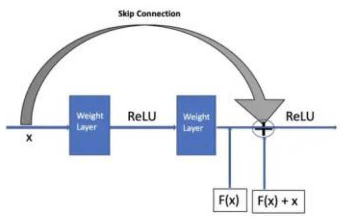


Figure 5. Residual block used to build ResNet [27]

The way that the network with residual block works is that, the output of each layer is fed to the next layer and also the output is fed to the next layers by skipping the layers in between. The main advantage of having a residual block Fig.5 is that we can train deeper networks. The skip connection is the one that is highlighted in gray because it is bypassing the layers between. It is also known as identity connection.

Vanishing gradient problem is addressed by Resnet's skip connections. The way this works is that gradient flows through the shortcut or skip connection path. Another advantage of having skip connections is that it allows learning identity functions of the model. This ensures that both the lower layers and higher layers work at the same performance.

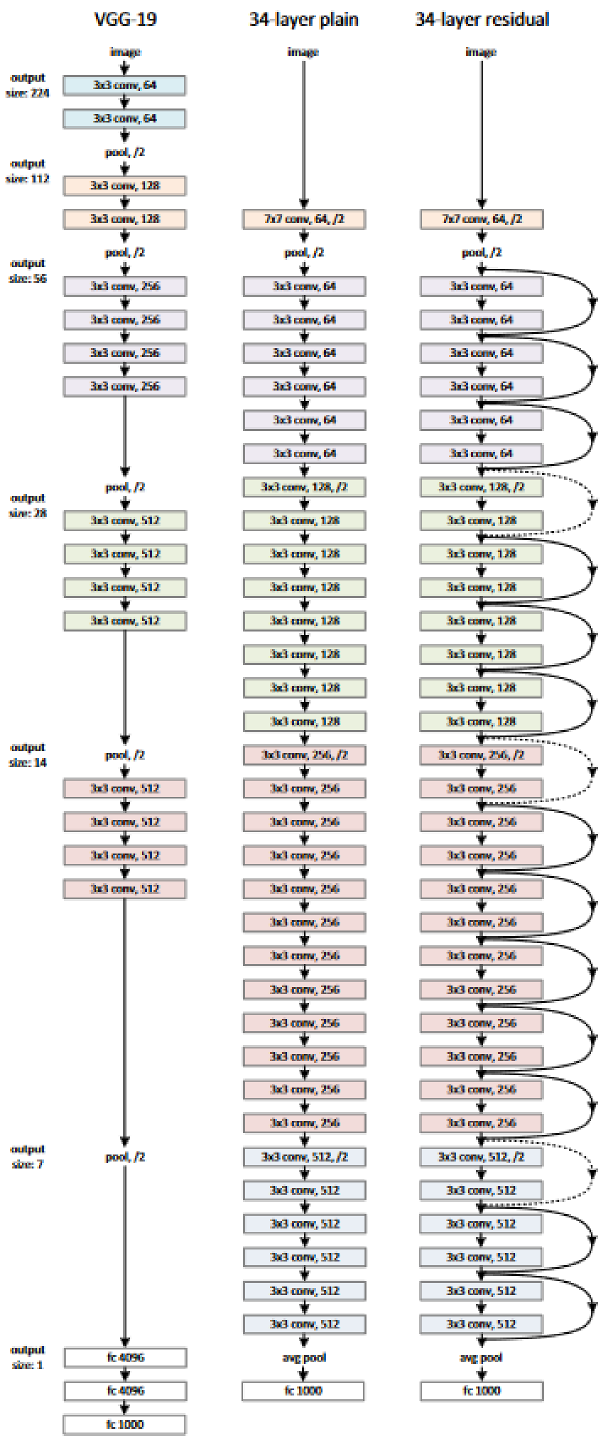


Figure 6. Comparison of ResNet34 Architecture to a simple 34 layer Convolutional Neural Network and VGG-19 [1] for comparison

While the ResNet was providing really promising results, training it from scratch was taking too long which is why we had to proceed with exploring some lighter architectures like MobileNet.

D. MobileNet

[16]This family of models has been popular amongst its peers like Efficient Net, Inception model and so on. It is so because of the concept applied at the core of its working ability.

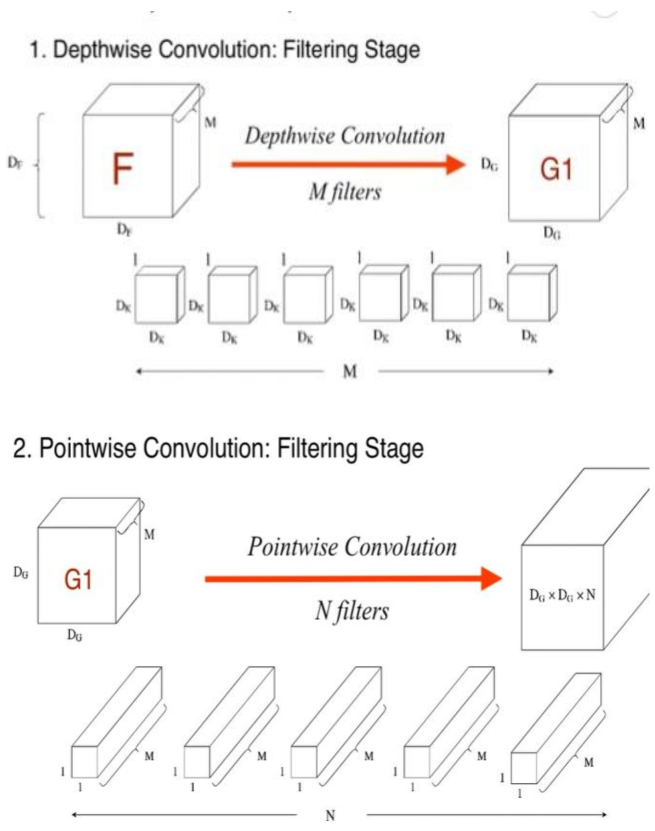
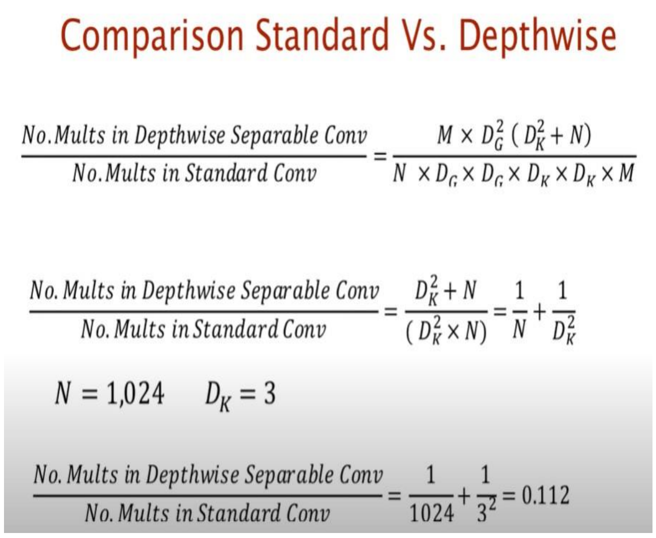

1. Depthwise Convolution: Filtering Stage
F
G1
Depthwise Convolution
M filters
D_F
D_G
D_K
M
1
2. Pointwise Convolution: Filtering Stage
Pointwise Convolution
N filters
$D_G \times D_G \times N$
N

Figure 7. Depthwise Separable Convolution

[4] The point of depthwise separable convolution is fairly easy but different from normal convolution. This is the heart of the concept in MobileNet architecture. As shown in Fig.7 when a kernel is applied for depthwise convolution, 4 dimensions play an important role in preserving the number of channels M because the convolution happens many times but it is for given dimensions of input times the number of channels. This results in a convolution operation on a single channel level and continues until it is created for every channel. There is an inversion in the idea of reducing the number of layers as we move forward with the model computation. It is due to pointwise convolution that projects more number of layers than the input to pointwise operation which is why the dimensions increase instead of decreasing.

The pointwise convolution focuses on the inverted residual concept where the end result of this step gives a new number of channels N by eliminating the input number of channels M. This is the kind of convolution that creates a difference from normal convolution because the cost of computation reduces quite well in depthwise separable convolution. Below Fig.8. shows the comparison between normal and depthwise-pointwise convolution.



Comparison Standard Vs. Depthwise

$$\frac{No.Mults\ in\ Depthwise\ Separable\ Conv}{No.Mults\ in\ Standard\ Conv} = \frac{M \times D_G^2\ (D_K^2 + N)}{N\ \times D_G \times D_G \times D_K \times D_K \times M}$$

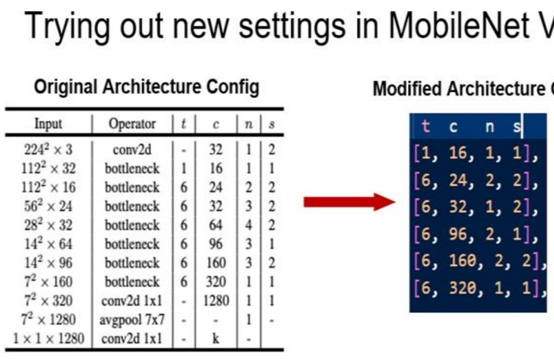
$$\frac{No.Mults\ in\ Depthwise\ Separable\ Conv}{No.Mults\ in\ Standard\ Conv} = \frac{D_K^2 + N}{(D_K^2 \times N)} = \frac{1}{N} + \frac{1}{D_K^2}$$

$$N = 1{,}024 \qquad D_K = 3$$

$$\frac{No.Mults\ in\ Depthwise\ Separable\ Conv}{No.Mults\ in\ Standard\ Conv} = \frac{1}{1024} + \frac{1}{3^2} = 0.112$$

Figure 8. Comparison of Standard Vs, Depthwise Convolutions

So the overview of complete MobileNetV2 model consists of 19 blocks each constituting batch normalization layer, convolution layer and relu6 activation function which is represented by the following Fig.9. It has standard or original configuration that is followed in MobileNetV2 model and to propose our own model, we made slight changes in the t, c, n, s configuration where t = expand-ratio, c = number of channels, n = number of blocks (with the same configuration as in that row, more like a multiplier of a block), s = stride(similar to strides used in pooling operations). The proposed model worked well despite missing configurations from the original setting.



Trying out new settings in MobileNet V2

Original Architecture Config

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Modified Architecture Config

```
 t  c   n  s
[1, 16, 1, 1],
[6, 24, 2, 2],
[6, 32, 1, 2],
[6, 96, 2, 1],
[6, 160, 2, 2],
[6, 320, 1, 1],
```

Figure 9. Original MobileNetV2 and Our proposed MobileNet (Modified) model settings [18]

The final layer consists of a fully connected layer that converts the results into a sequential classifier with pre-configured classes set to 1000. They are replaceable with our required number of classes like 10 for CIFAR10 and 100 for CIFAR100. We thought of further modifying the network by further shrinking the dimensions of 7x7 image to 3x3 image but it was not worth the risk of working with such a diminished output.

We also thought of using other configurations of mobilenet like its version v3 (small/large) but its specifications are more significant to study real time hardware capacity which is out of scope for our project.

Following is the final overview diagram of mobilenetv2 architecture at figure 10

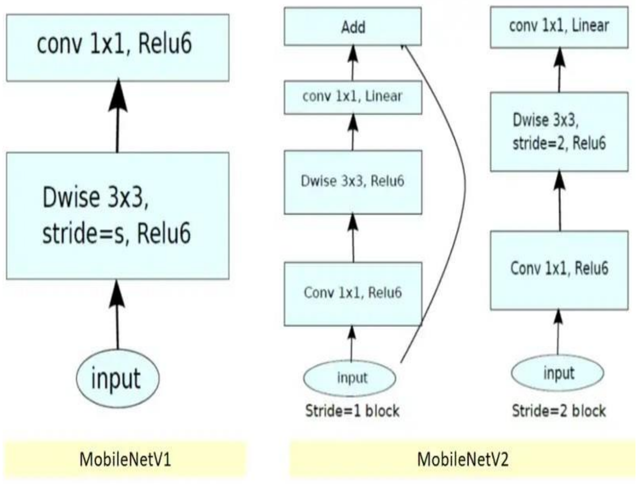


Figure 10. MobileNet internal Architecture

E. Transfer Learning

Whenever we have to train larger models, we go for the best possible hardware on research and industrial scale which brings us to transfer learning because here we can directly reuse the learning that was computed earlier for a model configuration so that we can start training by trusting the parent model's learnings. This idea is helpful in many ways when using multiple datasets to determine the performance of various model architectures which happens to be done in our project. In our scenario we use standard library configurations with no pre-trained weights (by default in practice imagenet dataset weights are downloaded is set to be true but we set it false so that we can train our own parent model and use it to transfer the learning to its child model). The steps followed in with our experience is as follows in figure 11

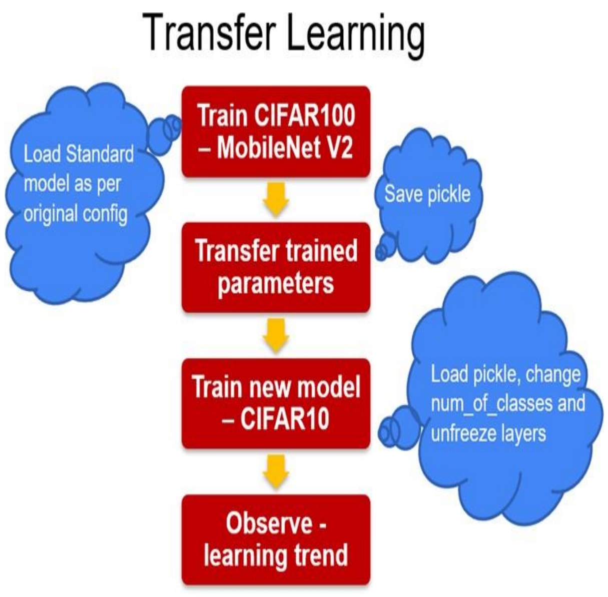

Transfer Learning
Load Standard model as per original config
Train CIFAR100 – MobileNet V2
Save pickle
Transfer trained parameters
Train new model – CIFAR10
Load pickle, change num_of_classes and unfreeze layers
Observe - learning trend

Figure 11. Transfer Learning

We did not get a very accurate result. But the main purpose of doing so was to understand how the learning from 100 class dataset scores in lower 10 class configuration. From what we understand is that transfer is good enough between similar kinds of datasets but maybe the model does not stand well due to the fact that there is a total of 90 classes of difference between two datasets rather than similarity on an overall basis between the two datasets. We have also made sure to reduce learning time by using a pre-trained model from the parent model trained on CIFAR100. To make up for the difference in dataset while training the child model on CIFAR-10, we have unfreezed the layers at the end of standard (original) configuration of mobilenetV2 as shown in Fig.9 earlier. It is our observation that for best practice while training on GPU for long intervals, it's good to set checkpoints in the implementation and make sure that every model created is saved on the user's hard-drive because free GPU is very limited and difficult to retain when you need it the most. Number of classes will also change based on the dataset used. Transfer learning also saves a lot of computation power in terms of applicability.

V. EMPIRICAL ANALYSIS / RESULT

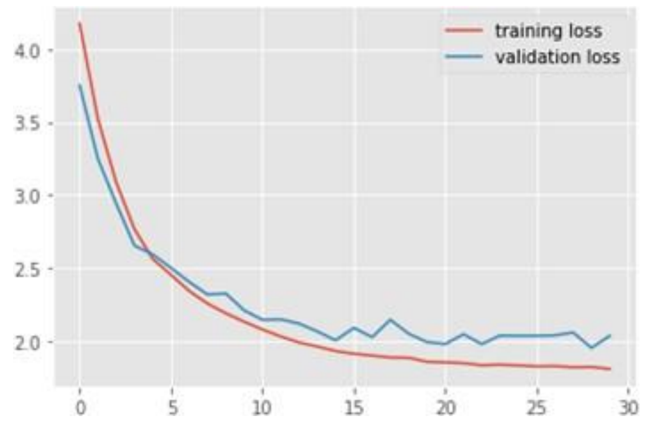


Figure 12.1 ResNet Loss Curve Lr = 0.1, SGD optimizer

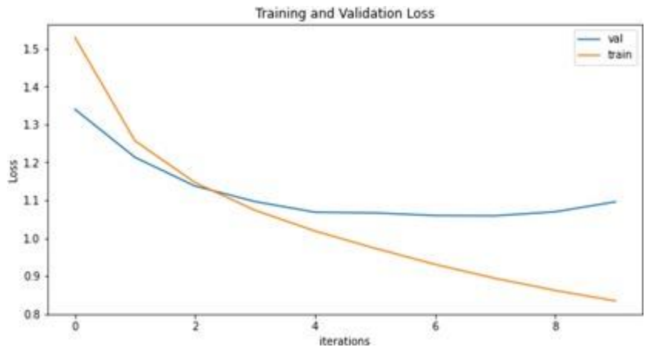


Figure 12.2 LeNet Loss curve Lr = 0.001, Adam optimizer

Failure with ResNet-34 was our motivation to give up on heavy models that take too much time for training. So the quality or results is not our priority with this model, which is why the loss curves even though smooth enough, and accuracy seems decent, we cannot use it for fast response models due to its heavy structural complexity and high inference time. LeNet also falls in the same category of failure models for fast response applications because it has noisy output since it is suitable for gray-scaled images and not for diverse real time (or close to them) images.

So now we explore the mobilenetV2 model and specifically the model that we obtained after changing mobilenetV2 internal configurations (let us call it modified mobilenet) to reduce training time and observe better results. The following shows impact of learning rate changes that can be observed best in diversity of results of Adam Optimizer for learning rates 0.1 and 0.01 as follows in Fig 12.3, 12.4, 12.5, 12.6

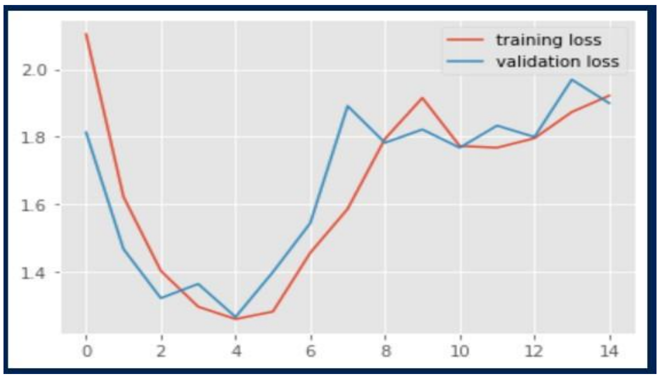


Figure 12.3 MobileNetV2 Loss Curve Lr = 0.1, Adam optimizer

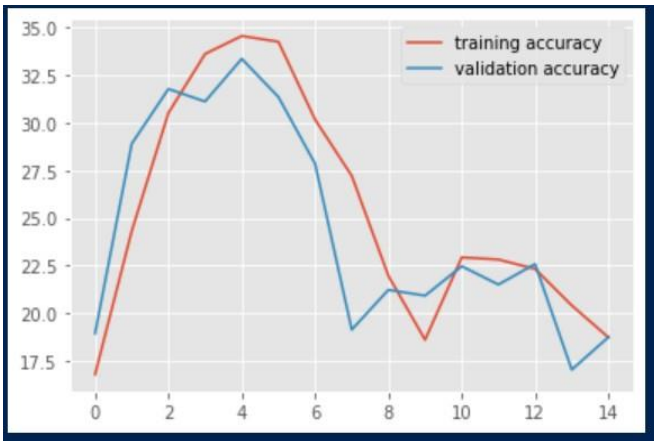


Figure 12.4 MobileNetV2 Accuracy Curve Lr = 0.1, Adam optimizer

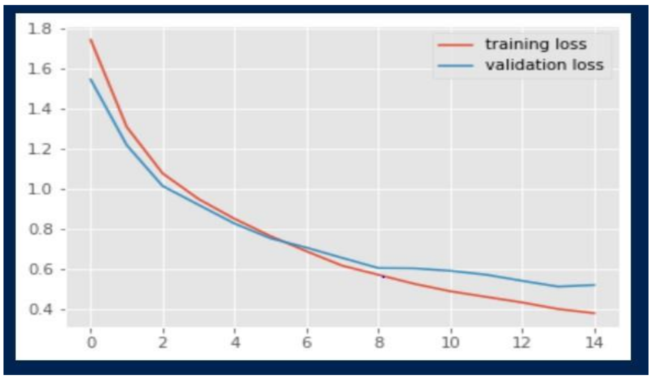


Figure 12.5 MobileNetV2 Loss Curve Lr = 0.01, Adam optimizer

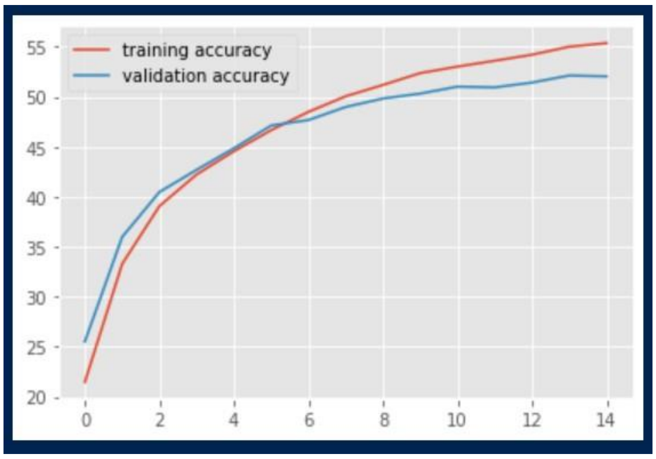


Figure 12.6 MobileNetV2 Accuracy Curve Lr = 0.01, Adam optimizer

The same trend was observed across various other combinations of hyperparameters but it is worth highlighting in Adam optimizer as it is known to be good for getting decent curves which does not seem to be the case with learning rate going higher than what is used in practice (0.01).

Also about epochs, and transfer learning even though the outputs as shown below are not great at all, the uneven graph confirms our point that datasets with higher num of classes need higher epoch count which is not possible with the limited computational capabilities of google colab. If we had high computation power then we could have been able to observe the result difference in epochs diversity and how it actually affects the model trained. Below are the graphs that show the effect of lower epoch run for higher class/label count (CIFAR100 - parent model in transfer learning) due to limited computational capacity (limited GPU availability) in Fig 12.7, 12.8, 12.9, 12.10

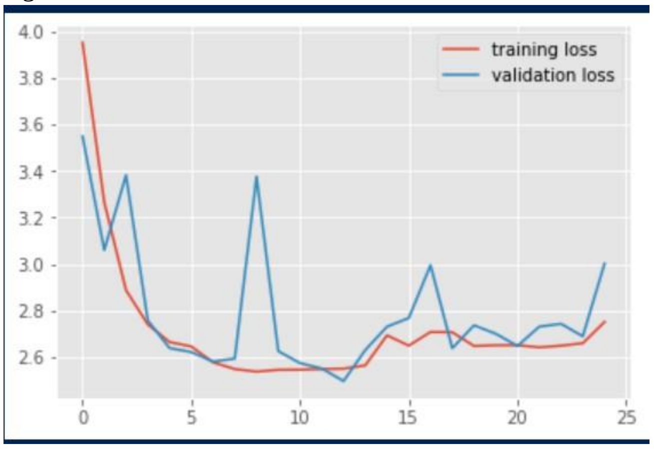


Figure 12.7 Transfer learning, MobileNetV2 (original config) on CIFAR100 - parent model in transfer learning Loss Curve Lr = 0.1, epoch = 25

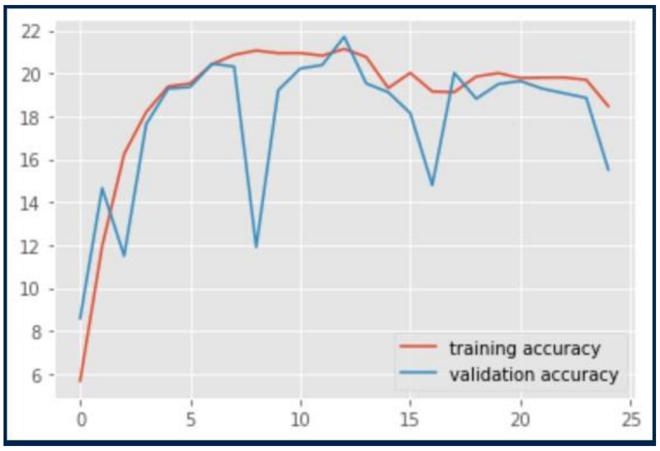


Figure 12.8 Transfer learning, MobileNetV2 (original config) on CIFAR100 - parent model in transfer learning Accuracy Curve Lr = 0.1, epochs = 25

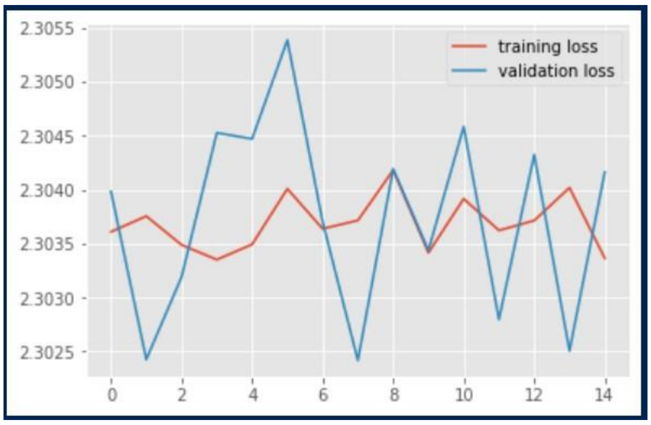


Figure 12..9 Transfer learning, MobileNetV2 (original config) on CIFAR10 - child model in transfer learning Loss Curve Lr = 0.1, epoch = 15

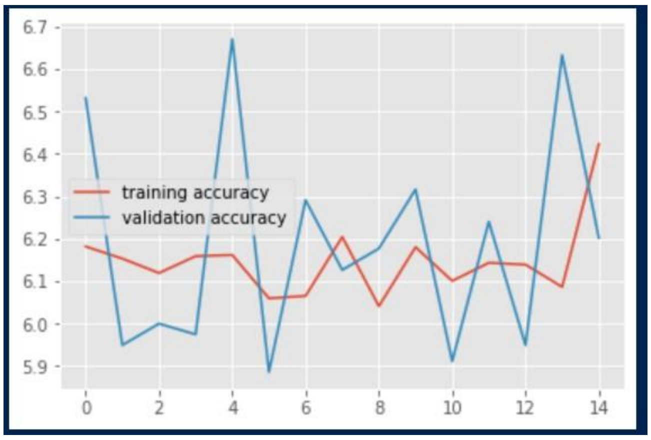


Figure 12.10 Transfer learning, MobileNetV2 (original config) on CIFAR10 - child model in transfer learning Accuracy Curve Lr = 0.1, epochs = 15

Despite all this, transfer learning has one observation that might be useful but may or may not be true, when we train child model by using parent model's learnings (primarily weights), it is not by coincidence that their accuracy values are in ratio ⅓ to ¼ (child model's accuracy/ parent model's accuracy). There must be some relation between some similarity index between two datasets and their model accuracy values when transfer learning is done between them. We can confirm these facts if there is enough GPU provided to run various combinations of parent and child models for transfer learning on a significant number of instances.

Coming to the importance of different optimizers used in model instances, their results show something significant about each of their usage based on the way they work as optimizers in models.

SGD - Since it calculates unit step (there is constant size descent in its gradient as it is a simple technique), the graph is not smooth because it does not have any direction to guide its convergence which makes it very generic and the graph turns out to be noisy. Observe below diagrams to witness it in Fig 13.1, 13.2

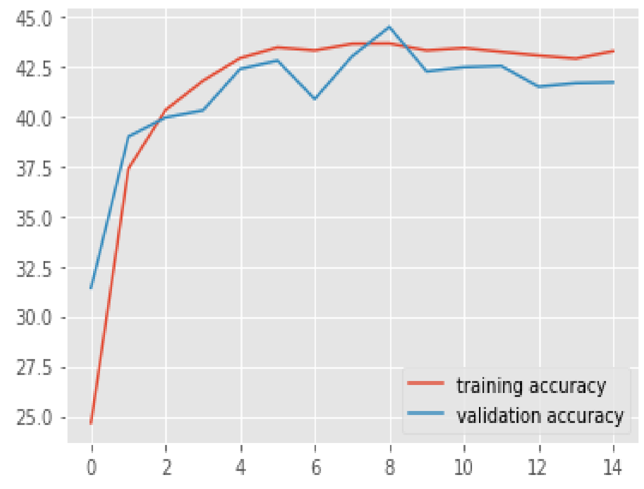


Figure 13.1 SGD Accuracy curve with lr = 0.1

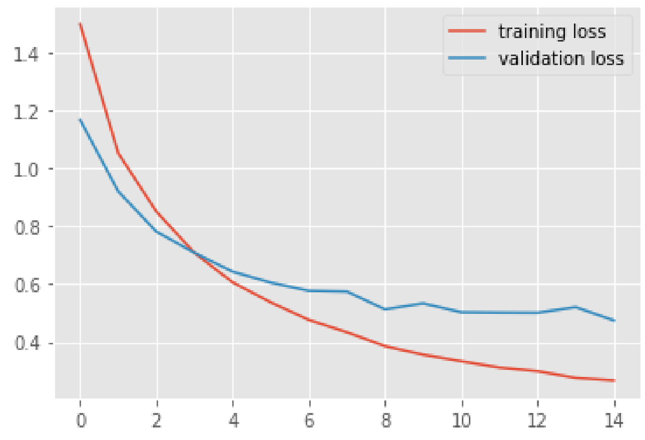


Figure 13.2 SGD loss curve with lr = 0.01

Adadelta - It is a superior version of Adagrad (not affected by learning rate decay problem) as it moves ahead with a component of limiting past gradient to a limit, so here averaging is used and not gradient sum and with every time step the gradiention is affected. This gives good loss but unreliable accuracy at times and is expensive (took some more time when compared to other optimizers). Observe below diagrams to witness it in Fig 14.1, 14.2, 14.3, 14.4

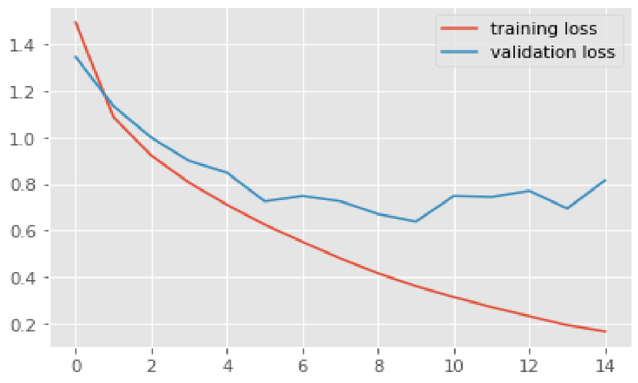


Figure 14.1 Adadelta loss curve with lr = 0.1

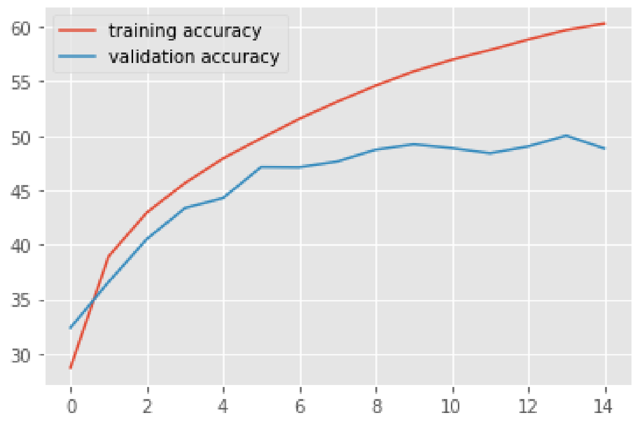


Figure 14.2 Adadelta accuracy curve with lr = 0.1

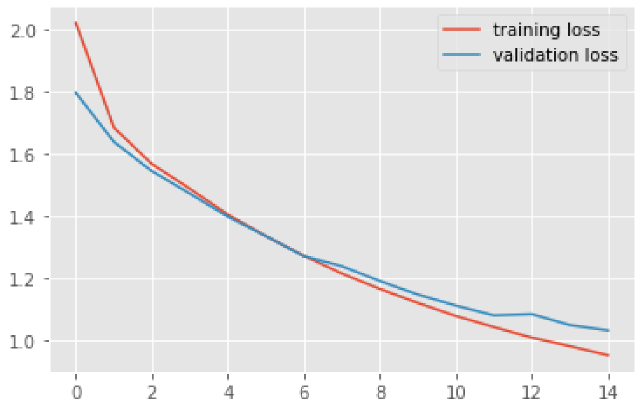


Figure 14.3 Adadelta loss curve with lr = 0.01

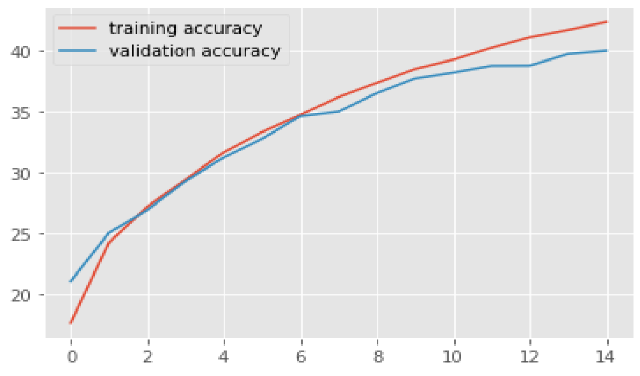


Figure 14.4 Adadelta accuracy curve with lr = 0.01

Adam - Makes use of velocity and momentum which has significant changes in results based on past gradient decreases (exponential - averaged out). This is the behavior that might drive it down fast enough to cross and move on the other side of minima which is the reason for its poor performance on learning rate = 0.1 but it changes quite a lot for learning rate = 0.01. Observe below diagrams to witness it in Fig 15.1, 15.2

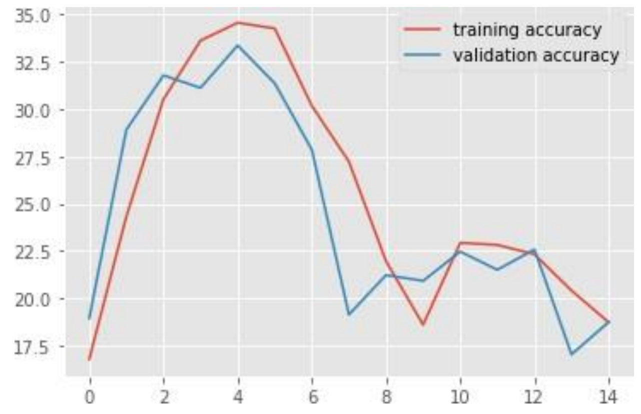


Figure 15.1 Adam accuracy curve with lr = 0.1

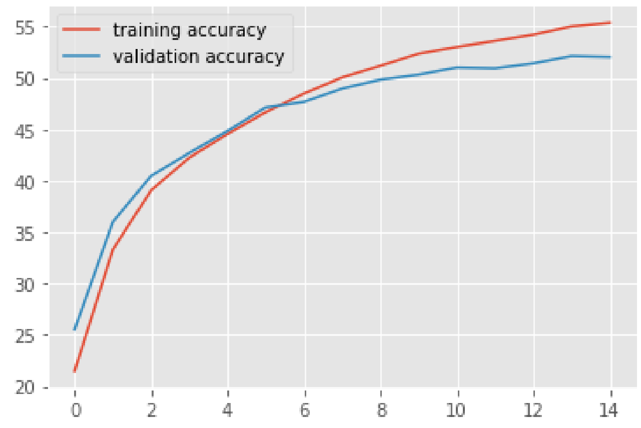


Figure 15.2 Adam accuracy curve with lr = 0.01

VI. CONCLUSION

This project concludes with an inference that the modified mobilenet (10-11 blocks total) based on the original mobilenetV2 architecture (18-19 blocks total) is better in terms of reducing the computations, simplifying the architecture for the required purpose of its usage (fast response models) without hurting the performance significantly. Compared to other models, its hardware utilization (RAM, etc in smart-phone like devices) is also less. Moreover it works using depthwise-pointwise separable convolutions which is a popular concept for easing up on computation costs. Following table 1 shows the resultant of the important model instances from this project (for CIFAR 10 model). Best result and applicability as per our project is by Proposed MobileNet Architecture (MobileNet model with reduced number of blocks as mentioned earlier).

Architecture	Learning rate	Optimizer	Accuracy %
ResNet34	0.1	SGD	75.5
LeNet	0.001	Adam	63
Proposed MobileNet Architecture	0.1	AdaDelta	76.25
Proposed MobileNet Architecture	0.1	Adam	28.95
Proposed MobileNet Architecture	0.1	SGD	66.56
Proposed MobileNet Architecture	0.01	AdaDelta	62.67
Proposed MobileNet Architecture	0.01	Adam	82.07
Proposed MobileNet Architecture	0.01	SGD	83.27

Table.1 - Shows results for important python notebooks for this project

REFERENCES

[1] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770-778, doi: 10.1109/CVPR.2016.90.

[2] Y. Bengio, P. Simard and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," in *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157-166, March 1994, doi: 10.1109/72.279181.

[3] https://www.datasciencecentral.com/lenet-5-a-classic-cnn-architecture/

[4] https://www.youtube.com/watch?v=T7o3xvJLuHk

[5] https://www.kaggle.com/c/cifar-10/

[6] https://en.wikipedia.org/wiki/CIFAR-10

[7] Fedesoriano,https://www.kaggle.com/datasets/fedesoriano/cifar100"

[8] DamianOrellana"https://medium.com/@damian.c036/training-model-to-classify-cifar100-with-resnet-4512d7a596a1"

[9] Prabhu,"https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148"

[10] Dharmaraj,https://medium.com/@draj0718/convolutional-neural-networks-cnn-architectures-explained-716fb197b243"

[11] https://paperswithcode.com/method/max-pooling

[12] PriyanshiSharma,"https://iq.opengenus.org/pooling-layers/"

[13] Yann LeCun Leon Bottou Yoshua Bengio and Patrick Haffner, "Gradient Based Learning Applied to Document Recognition", Proc of the IEEE, November 1998

[14] ShipraSaxena,"https://www.analyticsvidhya.com/blog/2021/03/the-architecture-of-lenet-5/"

[15] ShipraSaxena,"https://www.analyticsvidhya.com/blog/2021/03/introduction-to-the-architecture-of-alexnet/"

[16] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A. and Chen, L.C., 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 4510-4520).

[17] https://www.semanticscholar.org/paper/HIGHLY-EFFICIENT-8-BIT-LOW-PRECISION-INFERENCE/f809f9e5a03817d238718723a7b4ac04abcd3f12/figure/3

[18] https://pytorch.org/hub/pytorch_vision_mobilenet_v2/

[19] https://towardsdatascience.com/review-mobilenetv2-light-weight-model-image-classification-8febb490e61c

[20] Xiang, Q., Wang, X., Li, R., Zhang, G., Lai, J. and Hu, Q., 2019, October. Fruit image classification based on Mobilenetv2 with transfer learning technique. In *Proceedings of the 3rd International Conference on Computer Science and Application Engineering* (pp. 1-7).

[21] Xin, M., Wang, Y. Research on image classification models based on deep convolutional neural networks. J Image Video Proc. 2019, 40 (2019). https://doi.org/10.1186/s13640-019-0417-8

[22] Z. Feng, Z. Sun and L. Jin, "Learning deep neural network using max-margin minimum classification error," 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2016, pp. 2677-2681, doi: 10.1109/ICASSP.2016.7472163.

[23] https://www.analyticsvidhya.com/blog/2021/03/the-architecture-of-lenet-5/

[24] https://www.linkedin.com/pulse/lenet-architecture-asif-tandel/

[25] https://pytorch.org/vision/main/models/generated/torchvision.models.resnet34.html

[26] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770-778, doi: 10.1109/CVPR.2016.90.

[27] https://medium.com/analytics-vidhya/understanding-resnet-architecture-869915cc2a98