Deeper Networks for Image Classification

Danish Mehra
School of Electronic Engineering and Computer Science
Queen Mary University of London
ec20716@qmul.ac.uk

***Abstract*— Deep learning networks have significantly aided image recognition and classification problems over the last decade. This is partially due to varying architectures, differing in many aspects, such as the types of layers, hyperparameters, etc. In this paper three of the most notable deep networks VGG [3], GoogleNet [4] and ResNet [5] have been used. Both MNIST and CIFAR10 datasets were used to perform experiments for various configurations of these networks. Finally, the paper is concluded by comparing the performance of different model families and demonstrating how variables such as data augmentation, network depth, network blocks (inception block, residual block), changing batch size, learning rate and optimizer impact the model performance in terms of loss, precision, convergence, and training time.**

***Keywords*— VGG16bn, VGG19bn, GoogleNet, ResNet**

I. INTRODUCTION

Some of the most important problems studied in the field of computer vision and machine learning are image recognition and detection. In recent years, a growing number of deep network architectures with multiple convolution layers have been used to solve this problem. In this paper, three such architectures have been discussed: VGG [3], ResNet [5], and GoogleNet [4], which have all proven to be seminal in this area.

Here, a more compact view of the dataset, the optimizer, the architecture of these models along with the different configurations have been discussed. Experiments have been performed in which it is demonstrated how changing certain factors like introducing data augmentation with image cropping, flipping, rotation and resizing (with bicubic interpolation), as well as factors like changing the learning rate, changing the batch size, changing the architecture depth (VGG16→VGG19) or introducing certain architectural characteristics like Inception Block in Google Net or Residual Block in Resnet impacts model performance and training time.

II. CRITICAL ANALYSIS/ RELATED WORK

One way to identify the successful Deep Networks would be to look at the architectures of the top competitors in ImageNet Large Scale Visual Recognition Challenge (ILSVRC). This challenge is based on the ImageNet project which is a large visual database designed for use in visual object recognition software research. Here a few of these are discussed.

Yann LeCun et al. introduced **LeNet-5** [1] in the 1998 paper "Gradient-Based Learning Applied To Document Recognition" which is a pioneering 7-level convolutional network. It's applied by several banks to recognise hand-written numbers on checks (cheques) digitized in 32x32 pixel greyscale input images. It consists of two sets of convolutional and average pooling layers, followed by a flattening convolutional layer, then two fully-connected layers and finally a softmax classifier. Unfortunately, in order to process higher resolution images, it requires larger and more convolutional layers, so this technique was constrained by the availability of computing resources. However, in 2012, when computing power was increasing, this became popular. [1]

AlexNet [2] architecture was introduced in 2012 and helped reduce the top-5 error from 26% to 15.3%. The network was based on LeNet but was deeper, with more filters per layer, and with stacked convolutional layers. It consisted of 11x11,5x5,3x3, convolutions, max pooling, dropout, data augmentation, ReLU activation function and SGD with momentum. It attached ReLU activations after every convolutional and fully connected layer.

VGG16 was proposed by K. Simonyan and A. Zisserman from the University of Oxford in the paper "Very Deep Convolutional Networks for Large-Scale Image Recognition". [3] It was the runner-up at the ILSVRC 2014 competition. The model achieved 92.7% top-5 test accuracy in ImageNet. VGGNet had various notable architectures using 3x3 kernels for convolution over multiple layers and was appealing because of its uniform architecture. The network was trained on 4 GPUs for 2–3 weeks. Unfortunately, there are two major drawbacks in using VGGNet. Firstly, it was painfully slow to train and secondly, the network architecture weights themselves were typically quite large thus concerning disk/bandwidth. These days smaller network architectures such as GoogleNet are preferred more oftenly.

GoogleNet, also called **Inception V1** was the winner of the ILSVRC 2014 competition and was proposed by Google. [4] It achieved a top-5 error rate of 6.67% and came very close to human level performance. The network used a CNN inspired by LeNet but implemented a novel element which was dubbed as an inception module. It uses batch normalization, image distortions and RMSprop. Because of this inception module, it drastically reduces the number of parameters in the whole network from 60 million (AlexNet) to 4 million. The Inception module used different filters and reduces the computational cost of training an extensive network through dimensional reduction. The GoogLeNet architecture solves both the exploding or vanishing gradient problem that large networks faced, mainly through the Inception module's utilisation.

Residual Neural Network (ResNet) by Kaiming et al. was introduced at the ILSVRC 2015. [5] It won the competition and achieved a top-5 error rate of 3.57% which beat the human-level performance on this dataset. This model consisted of a novel architecture with "skip connections" which was called a residual block and featured heavy batch normalization. As a result, this network was trained with 152 layers while still having lower complexity than VGGNet. One of the problems ResNets was able to solve was the vanishing gradient problem because the gradients could flow directly through the skip connections backwards from later layers to initial filters.

III. MODEL ARCHITECTURE

In this section, the architecture of various configurations of VGG [3], ResNet [5] and GoogleNet [4] which were used in the experimentations are discussed.

A. VGG

For VGG, two configurations VGG16 [3] and VGG19 were used. VGG16 has 13 convolutional layers in the network with 3 fully connected layers. It uses 3 x 3 filters with stride of 1 in convolution layer and uses same padding in pooling layers containing 2 x 2 filters with a stride of 2. VGG16 has a total of 138 million parameters.

Another variation VGG19 contains 19 weight layers consisting of 16 convolutional layers (same 5 pooling layers), 3 fully connected layers and a final layer for softmax function. All the hidden layers in the network have ReLU activation function.

In both the configurations of VGG two of the 3 fully connected convolutional layers have 4096 channels each whereas, the third layer has 10 channels. Here it must be mentioned that since both CIFAR10 and MNIST dataset have been used in the experimentation, the VGG network was modified before the training to have 10 output channels. Originally, VGGNet had 1000 output channels since it was trained on ImageNet Dataset.

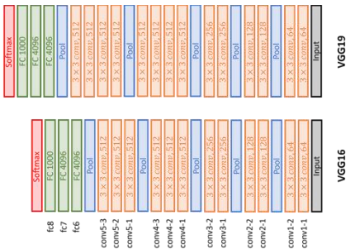


Figure 3.1: VGG16 and VGG19 Architecture

B. Inception Net V1

Inception Net V1 (GoogleNet) [4] consists of 22 layers (27 layers including pooling layers), and part of these layers are a total of 9 inception modules. An Inception Module allows us to use multiple filter sizes in a single image block, which is then concatenated and passed onto the next layer. It uses global average pooling at the end of the last inception module. A dropout layer (40%) is utilized just before the linear layer. The final layer is a linear layer with softmax activation function consisting of 1000 units, which corresponds to the 1000 classes present within the Imagenet dataset. However, since datasets like CIFAR10 and MNIST have been used, we have modified the output channels in our network to 10 similar to VGG.

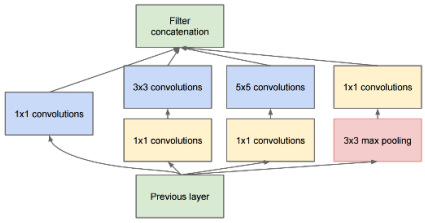


Figure 1.2: Inception Module with dimension reduction

There's one more component that was implemented by the creators of the network to regularise and prevent overfitting. This additional component is known as an Auxiliary Classifier. Auxiliary Classifiers are added to the intermediate layers of the architecture, namely the third (Inception 4[a]) and sixth (Inception4[d]). Auxiliary Classifiers are only utilised during training. Their purpose is to perform a classification based on the inputs within the network's midsection and add the loss calculated during the training back to the total loss of the network. This model has about 5 million parameters.

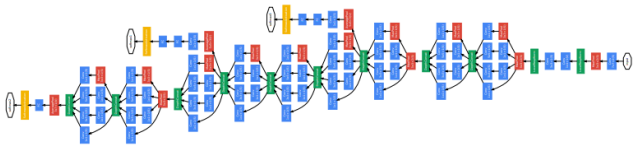


Figure 2.3: GoogleNet with Auxiliary Classifiers

C. ResNet

ResNet [5] architecture is based on residual connections which can be defined as direct connections between the input and the output of a block (multiple convolution layers). ResNet has multiple configurations of which we have used 34-layer & 18-layer architectures.

The network is essentially inspired by VGG-19 extended with added shortcut connections. ResNet18 has approx. 11 million trainable parameters with convolutional layers having filters of size 3x3. Throughout the network only 2 pooling layers are used, with one at the beginning and the second at the end of the network. Between every convolutional layer lies an identity connection. The network learns the mapping from x→F(x)+G(x) rather than x→F(x), where x is the input, the F(x) is the output of the previous layers and G(x) is the identify function also can be viewed as the shortcut or the identity connection. This function is learnt by reducing the weights to zero in the intermediate layer during training. We have also used ResNet34 configuration for our experiment. This model consists of 63.5 million parameters with 33 Conv. layers and pooling layers of size (both max pooling & average pooling) followed by fully connected layer. The network uses ReLU activation, batch normalization (BN) at the end of all convolution layers and softmax function at the end of fully connected layers.

layer name	output size	18-layer	34-layer
conv1	112×112		
conv2.x	56×56	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$
conv3.x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$
conv4.x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$
conv5.x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$
	1×1		av
FLOPs		1.8×10^9	3.6×10^9

Figure 3.5: ResNet18 and ResNet34 Architecture

IV. DATASETS USED

There are two datasets that have been used for training and testing the models for classification tasks. These are as follows:

The **MNIST dataset** [6] is a database of handwritten digits which has a training set of 60,000 examples, and a test set of

10,000 examples. In this dataset, digits have been size-normalized and centered in a fixed-size image. The resulting images are grayscale as a result of the anti-aliasing technique used by the normalization algorithm. The images were centered in a 28x28 image. The task is to classify a given image of a handwritten digit into one of 10 classes representing integer values from 0 to 9, inclusively.

Figure 4.1: Example MNIST Images

Secondly, **CIFAR-10** [7] dataset is an established computer-vision dataset used for object recognition and it consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. This dataset has been divided into 50000 training images and 10000 test images. Here are the 10 classes.

Figure 4.2: CIFAR10 Classes

V. Training Methodology

A. Loss Function

We have used Cross Entropy loss [8] for our training. This measures the performance of a classifier whose output is a probability value between 0 and 1 generalised for all the classes. Cross-entropy loss increases as the predicted probability diverges from the actual label. The mathematical formula is mentioned below.

$$L = -\frac{1}{m}\sum_{i=1}^{m} y_i \cdot \log(\hat{y}_i)$$

B. Optimization Algorithms

For optimization, both Stochastic gradient descent and Adam optimizers are used. Both these optimizers use Momentum [9] and Learning Rate as arguments. Momentum helps accelerate gradients vectors in the right directions, thus leading to faster convergence. Value of momentum used is 0.9 in the experimentations. On the other hand, learning rate is a tuning parameter which determines the step size at each iteration while moving towards a minimum of a loss function. For default, we have taken 0.05 as the Learning rate. In total, all these models are trained for 30 epochs from scratch using the following optimizers:

In **Stochastic Gradient Descent** [10]**,** which is a variation of the gradient descent algorithm, the training dataset is split into small batches to calculate model error and then to update model coefficients. Stochastic gradient descent with momentum accelerates convergence.

On the other hand, **Adam Optimizer** [11] can be looked at as a combination of RMSprop and Stochastic Gradient Descent with momentum. It uses the squared gradients to scale the learning rate like RMSprop and take advantage of momentum by using moving average of the gradient instead of gradient itself like SGD with momentum. Adam is an adaptive learning rate method, which means, it computes individual learning rates for different parameters. To adapt the learning rate, a scheduler has been used called Reducelronplateau (Pytorch) [12]. This scheduler reads a metrics quantity and if no improvement is seen for a 'patience' number of epochs, the learning rate is reduced. Default learning rate factor used is 0.1.

C. Experiments Performed

As part of the observations, first step was to keep all the hyperparameters as default and train all different configurations of the 3 different model architectures. Then accuracy of the models is compared by fine-tuning the hyperparameters. The effect of learning rates on performance is discussed [0.05,0.01,0.005, 0.0005]. Then the effect of batch sizes on performance has also been discussed. [64,128 and 256]. After that, effect of data augmentation is presented which is used to increase the diversity of the training data by applying random (but realistic) transformations such as image rotation, image flip and random crop. This essentially helps to generalise the model and avoid overfitting. For MNIST dataset which is grayscale, it was first converted to three channels then normalised and then used as input to the models.

Below we have summarised all the models we have trained:

Model Name	Dataset	Optimiser	Batch Size
VGG16	MNIST, CIFAR10*	SGD, Adam	64, 128, 256
VGG19			
GoogleNet			
ResNet18			
ResNet34			

Table 5.1: Models Summarized

*All these models have been trained with and without augmentation

VI. Observations and Outcomes

A. Observations

The first set of experimentations were performed on VGG16 model with batch normalisation. Here the outcome of the models that were trained on both the datasets (MNIST, CIFAR10) without augmentation are presented.

VGG16

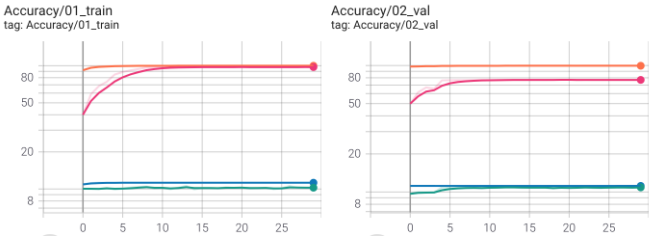


Figure 6.1 VGG16 Training and Validation Accuracy Graph

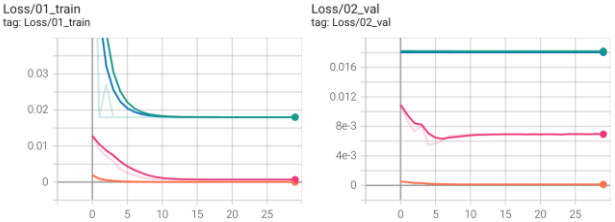


Figure 6.2 VGG16 Training and Validation Loss Graph

Model	Test Acc.	Test Loss	Val Acc.	Val Loss	Average Epoch time(s)
SGD_MNIST_0.005_128	99.5	0.00012	99.6	0.00012	69.03
SGD_CIFAR10_ 0.005_128	76.5	0.008	77.0	0.007	34.01
Adam_MNIST_ 0.005_128	11.3	0.018	11.1	0.018	73.82
Adam_CIFAR10_ 0.005_128	11.4	0.018	10.0	0.018	38.55

Table 6.1: VGG16 logs

In both the datasets, it can be observed that the models trained using Adam optimizer doesn't converge and therefore has a low validation accuracy. However, on using Stochastic gradient descent with momentum, the models performed much better. It is also clear for CIFAR10 dataset is slower to converge than the MNIST dataset which is expected because CIFAR10 is much more complex dataset than MNIST.

VGG19

The next models trained have been the VGG19 which is a deeper network compared to VGG16 with three additional convolution layers. The observations are shown below:

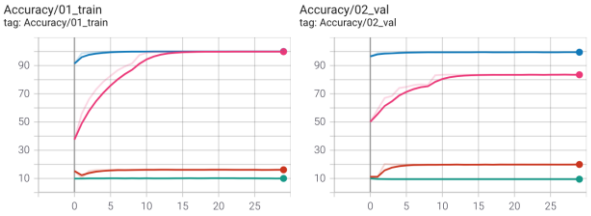


Figure 6.3 VGG19 Training and Validation Accuracy Graph

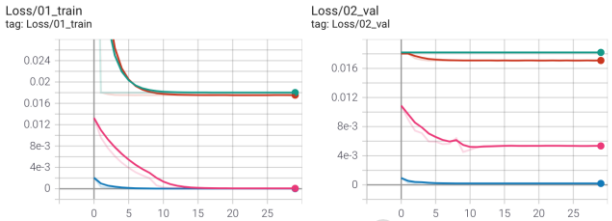


Figure 6.4 VGG19 Training and Validation Loss Graph

Model	Test Acc.	Test Loss	Val Acc.	Val Loss	Average Epoch time(s)
SGD_MNIST_0.005_128	99.43	0.0001	99.47	0.00018	50.35
SGD_CIFAR10_ 0.005_128	82.70	0.0059	83.46	0.0053	71.28
Adam_MNIST_ 0.005_128	19.79	0.017	19.96	0.017	53.96
Adam_CIFAR10_ 0.005_128	10.00	0.018	9.48	0.018	76.20

Table 6.2: VGG19 logs

From the above observations we can note that overall VGG19 has a slightly better performance compared to VGG16 due to the added layers. However, this comes at the cost of training time as can be seen. It can be seen that increasing the depth of the network helped improve performance.

ResNet18

Next experimentation was using the ResNet architecture to see the impact of introducing residual blocks on performance. We present the results below:

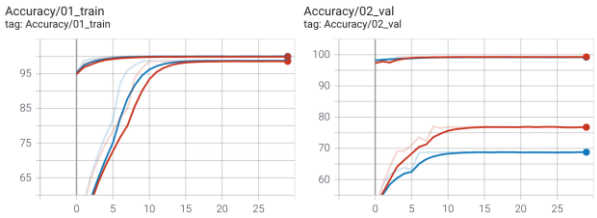


Figure 6.5 ResNet18 Training and Validation Accuracy Graph

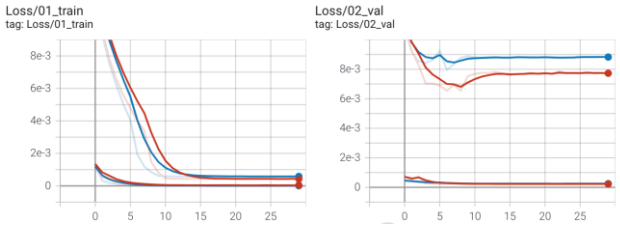


Figure 6.6 ResNet18 Training and Validation Loss Graph

Model	Test Acc.	Test Loss	Val Acc.	Val Loss	Average Epoch time(s)
SGD_MNIST_0.005_128	99.22	0.0002	99.16	0.00024	23.38
Adam_MNIST_ 0.005_128	99.34	0.0002	99.26	0.00023	24.92
SGD_CIFAR10_ 0.005_128	68.57	0.0092	68.75	0.0088	15.77
Adam_CIFAR10_ 0.005_128	76.46	0.007	76.75	0.0077	16.28

Table 6.3: ResNet18 logs

From the above it seems that the architecture provides comparable accuracies to VGG networks. However, it must be mentioned that VGGs perform better with the Adam optimizer. Moreover, in comparison Adam outperforms SGD in both the datasets in ResNet18. It can also be seen that ResNet18 trains much faster than VGGs (in fact it's about three time faster to train).

ResNet34

On experimenting with deeper Resnet network (ResNet34) the following characteristics where observed:

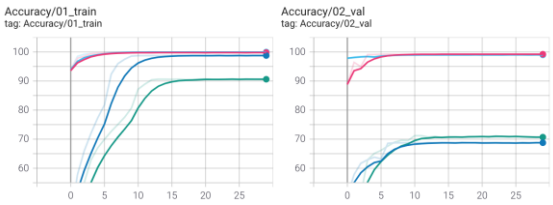


Figure 6.7 ResNet34 Training and Validation Accuracy Graph

Model	Test Acc.	Test Loss	Val Acc.	Val Loss	Average Epoch time(s)
SGD_MNIST_0.005_128	99.14	0.0002	98.97	0.00029	23.58
Adam_CIFAR10_ 0.005_128	71.20	0.01	70.70	0.009	21.63
SGD_CIFAR10_ 0.005_128	68.48	0.011	67.93	0.011	20.74
Adam_MNIST_ 0.005_128	99.42	0.0001	99.22	0.0002	24.57

Table 6.4: ResNet34 logs

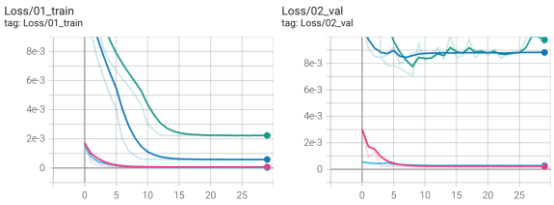


Figure 6.8 ResNet34 Training and Validation Loss Graph

With ResNet34, it can be seen that it didn't have a better performance when compared to ResNet18. In fact, with CIFAR10 we can observe that ResNet18 gives slightly better performance. With average epoch time as well ResNet18, being a less deep network, trains faster.

GoogleNet

On seeing the impact of introducing inception blocks, next experimentation was using the Inception Network v1, these are the following observations:

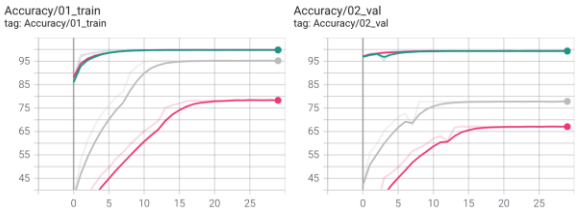


Figure 6.9 GoogleNet Training and Validation Accuracy Graph

Model	Test Acc.	Test Loss	Val Acc.	Val Loss	Average Epoch time(s)
SGD_MNIST_0.005_128	99.35	0.00016	99.33	0.00017	20.43
SGD_CIFAR10_ 0.005_128	77.51	0.006	77.89	0.0060	18.69
Adam_MNIST_ 0.005_128	99.41	0.0002	99.3	0.0002	22.94
Adam_CIFAR10_ 0.005_128	65.87	0.0082	67.03	0.008	21.96

Figure 6.10 GoogleNet Training and Validation Loss Graph

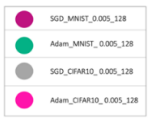


Table 6.5: GoogleNet logs

It can be observed that GoogleNet provides comparable performance to both ResNet18 and VGG16. Similar to ResNet it is faster to train. It can also be noticed that Adam worked slightly better in ResNet34 and ResNet18 than in GoogleNet.

VII. Further Evaluation

A. *Data Augmentation using VGG16*

To check the impact of data augmentation (rotation, flip, crop) we have retrained the VGG16 model with augmented data for CIFAR10. Here are the results:

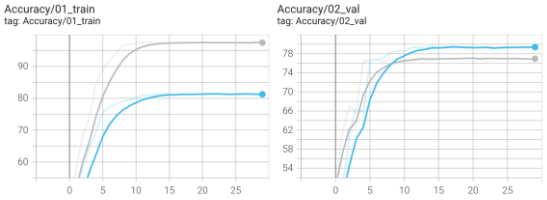


Figure 7.1 VGG16 with and without Augmentation Accuracy Graph

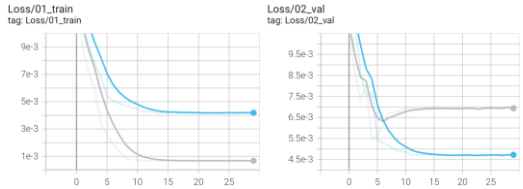


Figure 7.2 VGG16 with and without Augmentation Loss Graph

Model	Test Acc.	Test Loss	Val Acc.	Val Loss	Epoch time(s)
A_SGD_CIFAR10_0.005_128	80.17	0.004	79.4	0.004	78.13
NoA_SGD_CIFAR10_ 0.005_128	76.53	0.007	77.04	0.0069	34.01

Table 7.1: VGG16 with and without Augmentation logs

It can be seen from the above observations that while the above models trained on augmented data have slightly lower final training accuracy after 30 epochs, they have a higher test and validation accuracy indicating that data augmentation does help with model generalisation. Although as data augmentation helps diversify the training data, the average epoch time is almost double in the model trained with augmentation than without it.

B. *Impact of Learning Rate using GoogleNet*

The next experimentation has been performed to see the impact of learning rate. Here we have used GoogleNet trained on CIFAR10 for this comparison.

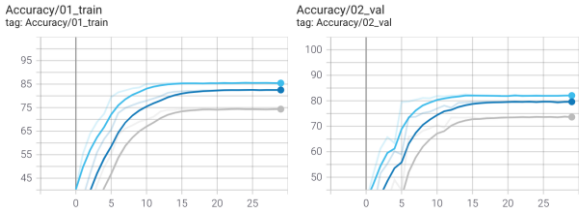
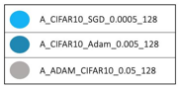


Figure 7.3 Learning Rate Accuracy Graphs

Model	Test Acc.	Test Loss	Val Acc.	Val Loss	Epoch time(s)
A_ADAM_CIFAR10_0.05_128	42.73	0.012	73.42	0.006	41.73
A_ADAM_CIFAR10_0.005_128	51.14	0.0118	79.72	0.0046	41.26
A_ADAM_CIFAR10_0.0005_128	53.39	0.011	82.12	0.0041	41.22

Table 7.2: Learning Rate logs



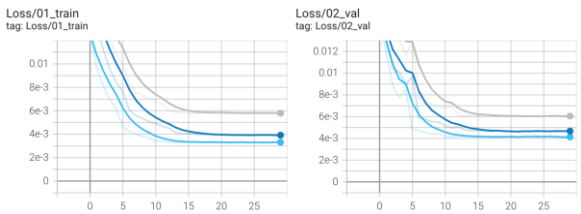


Figure 7.4 Learning Rate Loss Graphs

As we lower the learning rate through the following values [0.05,0.005,0.0005], we notice that both the validation accuracy and test accuracy increase, indicating we are closer to the minima. Interestingly the training time doesn't change considerably for these changes in learning rate.

C. Impact of Batch Size using GoogleNet and VGG16

Finally, we have presented here the impact of batch size on training the GoogleNet model. Here we have checked with batch sizes [128,256].

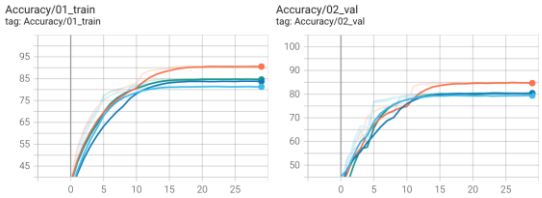


Figure 7.5 Batch Size Accuracy Graphs

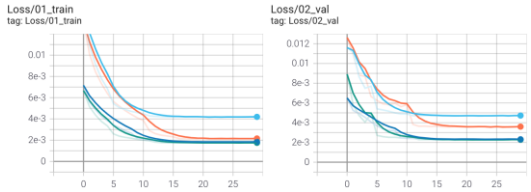
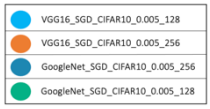


Figure 7.6 Batch Size Accuracy Graphs



Model	Test Acc.	Test Loss	Val Acc.	Val Loss	Epoch time(s)
VGG16_SGD_CIFAR10_0.005_128	80.17	0.0044	79.4	0.0047	78.12
VGG16_SGD_CIFAR10_0.005_256	81.97	0.002	80.56	0.002	75.74
GoogleNet_SGD_CIFAR10_0.005_256	45.33	0.07	80.52	0.0041	37.856
GoogleNet_SGD_CIFAR10_0.005_128	59.53	0.011	84.42	0.0022	40.06

Table 7.3: Learning Rate logs

By varying the batch sizes, the test accuracy increases as we decrease the batch size in GoogleNet. Although in VGG16, performance is almost comparable. Another thing is that although the training time doesn't seem to change considerably, it does increase when decreasing the batch size.

D. Confusion Matrix and Images

To understand how the classifier is treating each of the ten classes in either dataset, the confusion matrix (with heatmap) for VGG16_bn for both CIFAR10 and MNIST datasets have been shown in **Figure 7.7**. It appears from the data that in CIFAR10 dogs and cats tend to be misclassified more than the other classes. For illustration a few examples of both correct and incorrect classifications have been shown below in Figure 7.8 and Figure 7.9:

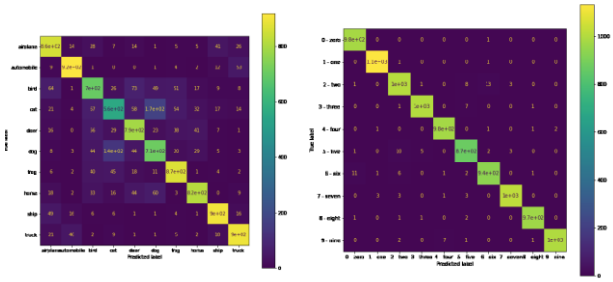


Figure 7.7 Confusion Matrices

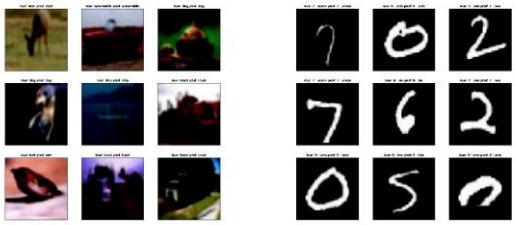


Figure 7.8 Correct Classifications

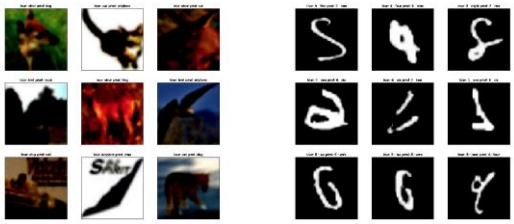


Figure 7.9 Incorrect Classifications

VIII. Future works and Conclusion

In conclusion, to summarise our observations:

- Increasing the depth of the network improves the training, validation and the test accuracy, indicating that additional convolutional layers help in boosting the classifications.
- CIFAR10 dataset is slower to converge than the MNIST dataset as CIFAR10 contains RGB images which makes it more complex that MNIST.
- Data Augmentation helps in improving test & validation accuracy thus avoiding overfitting.
- Adam optimizer doesn't perform well with VGG networks. However, stochastic gradient descent with momentum gives much better results.
- Introducing network characteristics like inception block or residual block simplifies the model (reduces parameters) and helps improving the training time. Nevertheless, overall accuracies of these networks are comparable to VGG networks.
- Decreasing the learning rate improves the performance without impacting the training time too much.
- Reducing the batch size doesn't have a major impact on the performance of the models however it makes the training process slower.

Lastly the following suggestion could be made for additional analysis. While the models have been trained for 30 epochs, it could be considered to further train them up to 100 epochs. Here, an option could be to perform transfer learning by loading a pre-trained model, trained on a different dataset like ImageNet and then modifying and training only the later layers. Also, it must be mentioned that all the models in the paper had been originally trained using 227x227x3 datasets. Since, both MNIST (28x28x1) & CIFAR10(32x32x3) have smaller images, we have resized the images to 64x64x3 using Bi-cubic interpolation (before training). Unfortunately, Due to limited availability of computational resources the datasets couldn't be resized to 227x227 images.

REFERENCES

[1] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. 1998, doi: 10.1109/5.726791.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1 (NIPS'12). Curran Associates Inc., Red Hook, NY, USA, 1097–1105.

[3] Zisserman, Andrew. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition.

[4] Szegedy, Christian, et al. "Going deeper with convolutions." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.

[5] He, Kaiming & Zhang, Xiangyu & Ren, Shaoqing & Sun, Jian. (2016). Deep Residual Learning for Image Recognition. 770-778. 10.1109/CVPR.2016.90.

[6] Deng, L. (2012). The mnist database of handwritten digit images for machine learning research. IEEE Signal Processing Magazine, 29(6), 141–142.

[7] Krizhevsky, Alex. (2012). Convolutional Deep Belief Networks on CIFAR-10.

[8] Zhang, Zhilu & Sabuncu, Mert. (2018). Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy Labels.

[9] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. 2013. On the importance of initialization and momentum in deep learning. In Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28 (ICML'13). JMLR.org, III–1139–III–1147.

[10] Towards Data Science [Internet]. Vitaly Bushaev; Dec 4, 2017. Stochastic Gradient Descent with momentum; Available from: URL: https://towardsdatascience.com/stochastic-gradient-descent-with-momentum-a84097641a5d.

[11] Kingma, Diederik & Ba, Jimmy. (2014). Adam: A Method for Stochastic Optimization. International Conference on Learning Representations.

[12] https://pytorch.org/docs/master/generated/torch.optim.lr_scheduler.ReduceLROnPlateau.html