1. Introduction

This paper aims at performing and evaluating image classification tasks with deeper networks such as VGG, ResNet and GoogleNet. For this purpose, I have used VGG16 and GoogleNet CNNs and performed the classification task on MNIST and CIFAR10 dataset. The whole process has been discussed below in detail.

2. Critical Analysis

In this paper, I perform an Image Classification task using two different deeper networks to evaluate their performance and effectiveness, i.e. VGG16[1] and GoogleNet[2].

2.1 Model Overview

2.1.1 VGG16

VGGNet was developed with the aim of reducing the number of parameters in the corresponding convolution layers and also to improve the training time of the model. VGG took forward the idea that small sized filters can help improve the performance of a CNN, proposed by ZfNet[5]. Small sized filters also provide the benefit of low computational complexity by reducing the number of parameters. There are many variants of VGG out there such as VGG13, VGG16, VGG19, etc and the major difference between them is only the number of layers working inside the network. I have used VGG16 and it has about 13M parameters which makes it a little computationally expensive. An advantage of VGG is the fixed kernel size(3x3) in the conv and maxpool layers which helps to reduce the number of variables.

2.1.2 GoogLeNet

GoogLeNet is another deep CNN that claims to offer state of the art results for Image Classification. It was one of the first to introduce the Inception block concept in CNNs. The main selling point of Inception is high accuracy and efficient resource consumption. GoogLeNet is a 22 layer network which uses about 6M parameters. It also makes use of different sized kernels for effective recognition of variable sized features. Prior to the Inception architecture, most of the popular CNNs just kept adding more layers to get better performance. But Inception was heavily engineered to grow wider by having filters with multiple sizes operate on the same level. GoogleNet follows heterogeneous topology which is also a drawback due to the complexity of the network.

2.2 Model Architecture

2.2.1 VGG16

The VGG takes a fixed-size 224x224 RGB image as input which is passed through a stack of conv layers. The number of these layers varies depending upon the variation of VGG being used. These layers have a 3x3 filter with a stride of 1 pixel. Five max-pooling layers, after a regular batch of conv layers in the configuration, are used to perform spatial pooling with a 2x2 kernel and stride 2. All of the hidden layers use ReLU rectification. These conv layers are followed by three FC(fully-connected) layers, two of which 4096 channels each and the last output layer which is the soft-max layer has 1000 channels, one for each class.

2.2.2 GoogLeNet

The Inception model has 2 base architectures, one is the naive version that performs convolutions on an input using three differently sized filters i.e 1x1, 3x3 and 5x5. It also has one max pooling layer. The outputs are then concatenated and then passed onto the next module. Another version has an added extra 1x1 convolution before the 3x3 and 5x5 convolutions. The idea here is to ultimately reduce the computational cost by reducing the number of input channels. This architecture of Inception with added Dimensionality Reduction feature came to be known as GoogLeNet.

3. Experiments

3.1 Datasets

3.1.1 MNIST

The MNIST Database of handwritten digits[3] is a subset of a larger NIST dataset which is openly available here. It consists of a training set which has 60,000 examples and a training set with 10,000 examples. For the purpose of this coursework I have further divided the train dataset into train and validation datasets in a ratio of 1:3. The dataset contains size-normalized images of centered digits. All the images are grayscale with 28x28 resolution i.e. (28, 28, 1). For the code purpose, the dataset was collected from *keras.datasets* library. It returns two tuples which contain the training and test examples.

3.1.2 CIFAR-10

The CIFAR-10 dataset[4] is a labelled subset of a larger dataset which has about 80M tiny images. CIFAR-10 contains 50,000 training and 10,000 test samples that have images that belong to one of the 10 classes. Thus, 6000 images per class in total, with each image being 32x32 with 3 RGB channels. Airplane, frog, cat,dog are some of the classes in the dataset. Like MNIST, this was also imported directly from *keras.datasets* library.

3.2 Model Design

3.2.1 VGG16

3.2.1.1 MNIST

By default, VGG architecture takes 224x224 RGB images, i.e. (224, 224, 3) as input and MNIST on the other hand consists of 28x28 grayscale images (28, 28, 1). To make up for this difference, I rescaled them using padding and stacked the layers to make it 3 channeled. VGG needs at least 32x32 images to process, otherwise at the last block of the network. The original VGG architecture has 4096 neurons in the Dense layers, I changed it to 256 neurons to make it converge faster than usual. And also the number of neurons in the last output layers were changed to match the number of classes in MNIST, i.e. 10. A dropout has also been added to the second last dense layer to improve the performance and reduce the risk of overfitting. The rest of the network has been constructed as original.

3.2.1.2 CIFAR-10

Similar to VGG, CIFAR images also needed resizing as they were (32, 32, 3) which I changed to (224, 224, 3) using (0,0) black padding. As the images are already RGB, they did not need to be stacked. CIFAR-10 also has 10 classes, hence the number of neurons in the last output layer were changed accordingly. I added a dropout to the second last dense layer as it was starting to overfit after a few epochs.

3.2.1.3 Model Evaluation

After understanding the model well, I have setup a baseline model as below

	model_name	activation	dataset	dropout_rate	epochs	input_dims	lr_rate	opt
0	VGG16	softmax	mnist	0.2	15.0	(224, 224, 3)	0.0100	sgd

Table 1: Baseline model Description

The above setup results in 99.8607% train accuracy, 99.0606% validation accuracy, 4.483153e-03 train loss, with 99.22% test accuracy and 3.033516e-02 test loss. I have evaluated VGG16 by changing different parameters such as listed below.

1. Using different Optimiser and datasets (MNIST/CIFAR10)

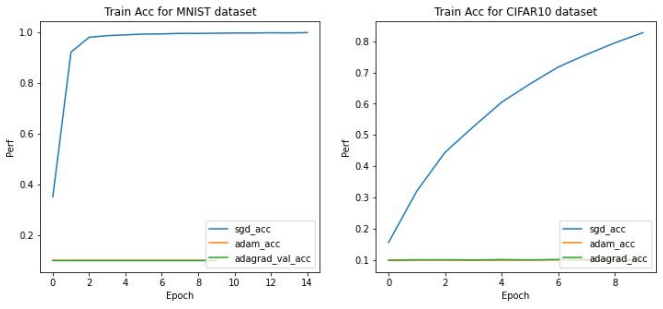


Figure 1: Training Accuracy on different datasets using different optimisers

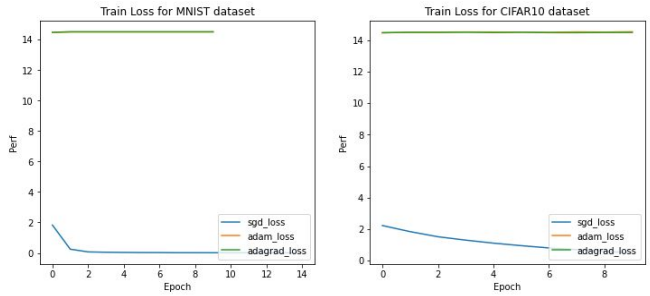


Figure 2: Training loss on different datasets using different optimisers

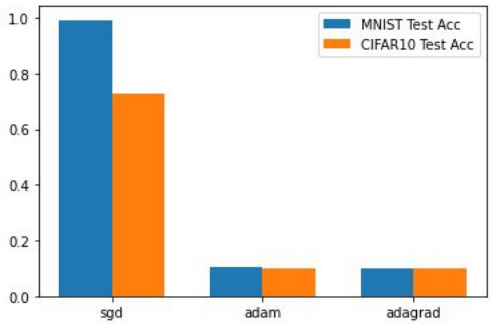
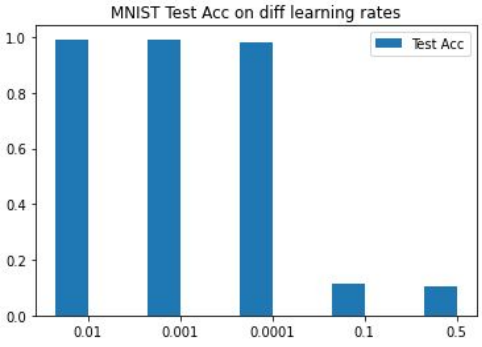


Figure 3: Test Accuracy on different datasets using different optimisers

In Figure 1 and 2, we observe that VGG16 performs well on stochastic gradient descent(sgd) for the given baseline environment, while other optimisers have oscillating gradients and fail to learn.

Figure 3, shows test accuracy results on different optimisers, and clearly sgd is performing better than the other two, adam and adagrad.

2. Using Different Learning rate

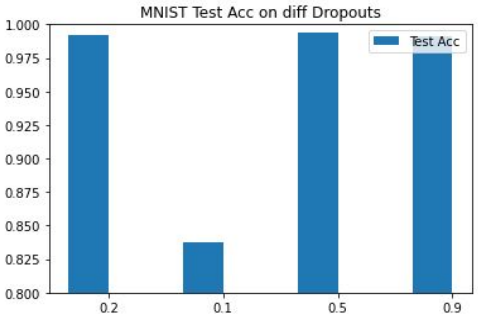


	opt	dataset	lr_rate	test_accuracy
0	sgd	mnist	0.0100	0.9922
6	sgd	mnist	0.0010	0.9898
7	sgd	mnist	0.0001	0.9803
8	sgd	mnist	0.1000	0.1135
9	sgd	mnist	0.5000	0.1032

Figure 4: Top-Test Accuracy on MNIST with different learning rate, Bottom-Table of experiment condition with different learning rate

From the figure and table above, it is clearly evident that VGG16 attain good results with slower learning rate.

3. Using Different Dropout rate at the output layer

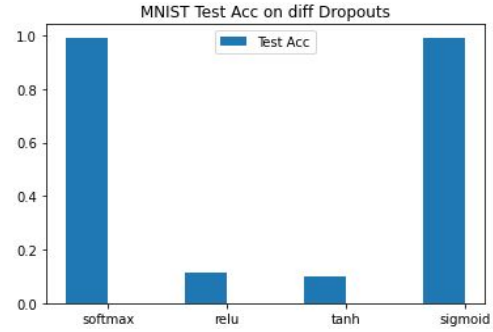


	opt	dataset	dropout_rate	test_accuracy
0	sgd	mnist	0.2	0.9922
13	sgd	mnist	0.1	0.8378
14	sgd	mnist	0.5	0.9940
15	sgd	mnist	0.9	0.9915

Figure 5: Top-Test Accuracy on MNIST with different learning rate, Bottom-Table of experiment condition with different learning rate

VGG16 performs exceptionally well with or without dropout at the output layer and the table above explains it clearly.

4. Using Different Activation function at the output layer



	opt	dataset	activation	test_accuracy
0	sgd	mnist	softmax	0.9922
10	sgd	mnist	relu	0.1135
11	sgd	mnist	tanh	0.0980
12	sgd	mnist	sigmoid	0.9914

Figure 6: Top-Test Accuracy on MNIST with activation functions, Bottom-Table of experiment condition with different activation functions

For the given baseline setup, VGG16 network performs very well with softmax and sigmoid activation function on the output layer but fails to give any good result with tanh or relu.

3.2.2 GoogLeNet

3.2.1.1 MNIST

Just like VGG, only the images were to be rescaled from (28, 28, 1) to (224, 224, 3) using padding and stacking. The rest of the architecture is the same.

3.2.1.2 CIFAR-10

Just like VGG, only the images were to be rescaled from (32, 32, 3) to (224, 224, 3) using padding. The rest of the network is the same.

3.2.1.3 Model Evaluation

After understanding the model well, I have setup a baseline model as below

	model_name	activation	dataset	dropout_rate	epochs	input_dims	lr_rate	opt
0	inception	softmax	mnist	0.2	10.0	(224, 224, 3)	0.0010	sgd

Tabel 2: Baseline model for GoogLeNet

The above setup results in 98.56% and 0.171548 test loss. I have evaluated VGG16 by changing different parameters such as listed below.

1. Using different Optimiser and datasets (MNIST/CIFAR10)

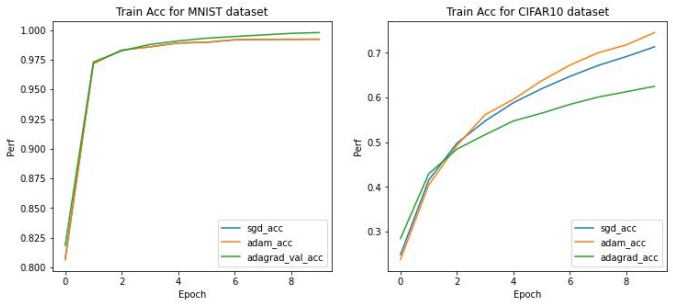


Figure 7: Training Accuracy on different datasets using different optimisers

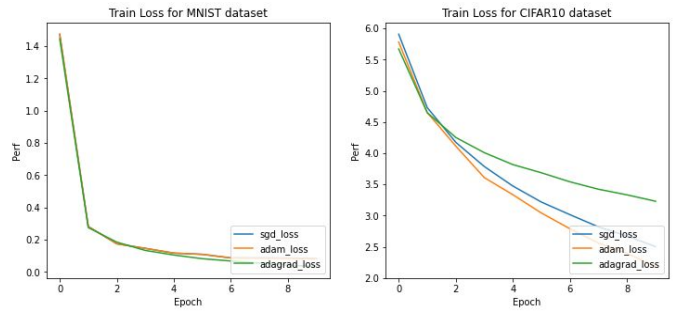


Figure 8: Training loss on different datasets using different optimisers

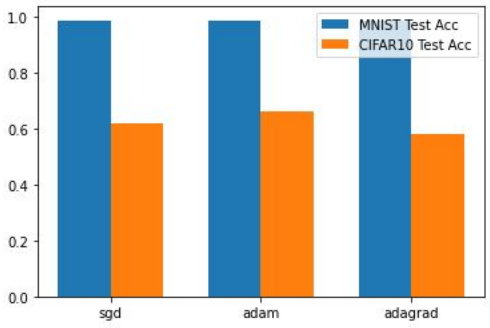


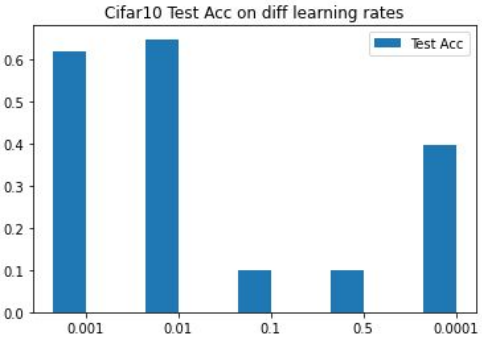
Figure 9: Test Accuracy on different datasets using different optimisers

In Figure 9 and 10, we observe that for each of the optimisers GoogLeNet produces good results which was not the case with VGG16. In the set epoch size of 10, MNIST is trained within 1-2 epochs while the

network can be seen learning on CIFAR10 smoothly.

Figure 10, shows that we achieve good test accuracy results on all the optimisers for both the datasets.

2. Using Different Learning rate

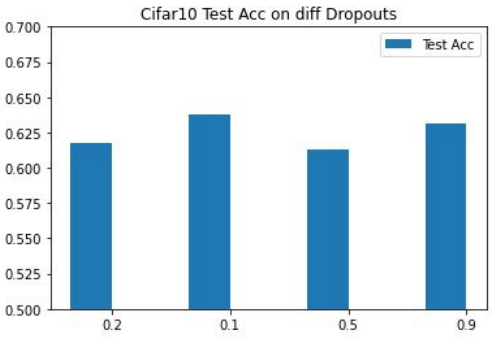


	opt	dataset	lr_rate	test_accuracy
3	sgd	cifar10	0.0010	0.6180
6	sgd	cifar10	0.0100	0.6478
7	sgd	cifar10	0.1000	0.1000
8	sgd	cifar10	0.5000	0.1000
9	sgd	cifar10	0.0001	0.3967

Figure 10: Top-Test Accuracy on CIFAR10 with different learning rate, Bottom-Table of experiment condition with different learning rate

On Increasing the learning rate, we observe a normal phenomena the test accuracy decrease to 10%. At 0.01, the network achieves very good results, lowering it further, we see reduction in performance.

3. Using Different Dropout rate at the output layer

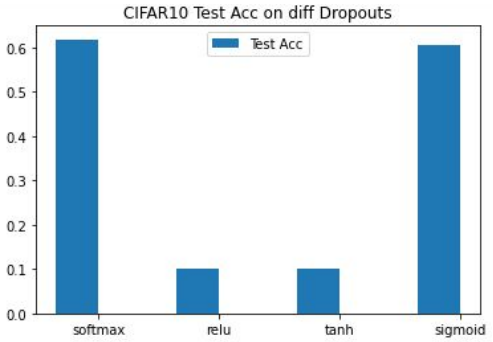


	opt	dataset	dropout_rate	test_accuracy
3	sgd	cifar10	0.2	0.6180
13	sgd	cifar10	0.1	0.6374
14	sgd	cifar10	0.5	0.6129
15	sgd	cifar10	0.9	0.6316

Figure 11: Top-Test Accuracy on CIFAR10 with different learning rate, Bottom-Table of experiment condition with different learning rate

With the addition of the dropout layer at the output layer, we do not see any improvement in the network performance. The model results with overall test accuracy of about 62%.

4. Using Different Activation function at the output layer



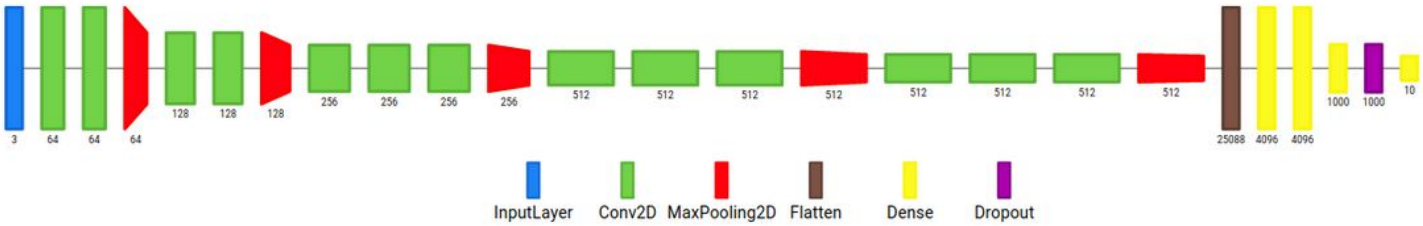
	opt	dataset	activation	test_accuracy
3	sgd	cifar10	softmax	0.6180
10	sgd	cifar10	relu	0.1000
11	sgd	cifar10	tanh	0.1000
12	sgd	cifar10	sigmoid	0.6052

Figure 6: Top-Test Accuracy on CIFAR10 with activation functions, Bottom-Table of experiment condition with different activation functions

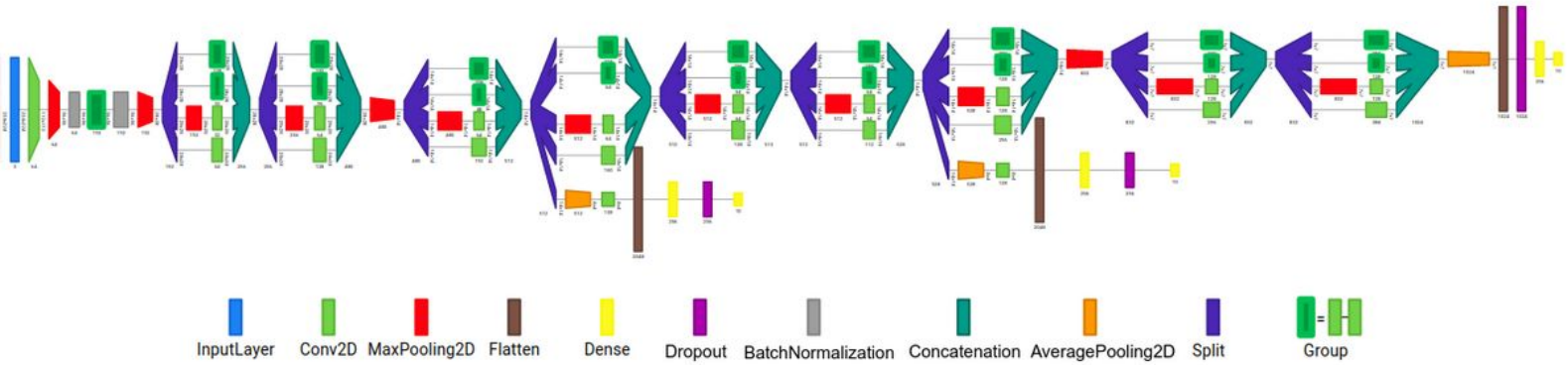
When we are using different activation layers at the output layers, with relu and tanh, we see the network fails to learn where as, if we place sigmoid and softmax, we get good results.

3.3 Network Design

3.3.1 VGG16



3.3.2 GoogLeNet



Figures generated via [Net2Vis]

3. Conclusion

With the availability of large datasets and computational capacity, it is possible to explore the possibilities of Deep CNNs. Here, I have implemented and evaluated two such networks, VGG16 and the GoogLeNet. Both the networks have their own pros and cons under different conditions. The performance of both the networks highly depends on the dataset, input dimensions and the hyperparameters used to train the networks. From the graphs attached above it is evident that for most of the cases GoogLeNet performed better and yielded better results. Along with that, it was fast and efficient as compared to its counterpart. Tuning and selecting the best parameters for the network is a crucial step for any CNN model and it is a time consuming process to experiment with various combinations of all the optimizers, activation functions and learning rates.

References

[1]*Very Deep Convolutional Networks For Large-scale Image Recognition* - Karen Simonyan, Andrew Zisserman, Visual Geometry Group, Department of Engineering Science, University of Oxford, ICLR (2015)

[2]*Going Deeper with Convolutions* - Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich, Google Inc., University of North Carolina, Chapel Hill, University of Michigan, Ann Arbor, Magic Leap Inc., ILSVRC(2014), CVPR (2015)

[3]*The Mnist Database Of Handwritten Digits* - Yann LeCun, Courant Institute NYU, Corinna Cortes, Google Labs New York, Christopher J.C. Burges, Microsoft Research Redmond

[4]*Learning Multiple Layers of Features from Tiny Images*, Alex Krizhevsky, 2009

[5]*Visualizing and Understanding Convolutional Networks* - Matthew D Zeiler, Rob Fergus, CVPR(2013)

[6]*A Survey of the Recent Architectures of Deep Convolutional Neural Networks* - Asifullah Khan, AnabiaSohail, Umme Zahoora, and Aqsa Saeed Qureshi, Pattern Recognition Lab, DCIS, PIEAS, Nilore, Islamabad 45650, Pakistan2 Deep Learning Lab, Center for Mathematical Sciences, PIEAS, Nilore, Islamabad 45650, Pakistan