VGG16 and GoogLeNet: A Case Study

1. Introduction

This paper showcases the performance and evaluation process for image classification tasks using deeper networks. It talks about VGG16 and GoogleNet CNNs and their performance in the classification task of MNIST and CIFAR-10 dataset.

2. Critical Analysis

This section analyses the original architectures, ideas and history behind the networks in discussion, i.e. VGG16 and GoogLeNet.

2.1 Model Description

2.1.1 VGG16

VGGNet was born out of the need to reduce the number of parameters in convolution layers and to make the training more efficient. It is an example of CNNs that highlight spatial exploitation. The VGG uses about 13 million parameters which can be considered as one of the main drawbacks of VGG. VGG uses a fixed filter of size 3x3 in the hidden layers to help reduce the number of variables. It is a 19 layered network that successfully demonstrated that concurrent placement of small sized filters like (3x3) could induce the same effect as a large sized filter such as (5x5) or (7x7). A max-pooling layer, placed after the convolutional layer helps tune the network. It was known mainly for homogeneous topology and the increased depth.

2.1.2 GoogLeNet

GoogLeNet was the first one to introduce the block concept for CNNs. It incorporates multi-scale convolutional transformations using split, transform and merge ideas called Inception blocks. These blocks use kernels of different sizes to effectively capture spatial information at different scales and also make the model robust to variable sized features within an image frame. GoogLeNet overcomes the problem of redundant information by using sparse connections. Original architecture is made up of 22 layers. They break the notion of simply adding more layers to get better performance by growing wider by using these different sized filters within the same layer. Such fine parameter tuning resulted in 6 million parameters as compared to 138 million of VGG.

2.2 Model Architecture

2.2.1 VGG16

VGG is a 19 layer deep network that uses fixed size kernels for convolutions. The original architecture of VGG takes a 224x224 RGB image as input which goes through the network of hidden layers. The number of hidden layers depends on the variant of VGG being used. I'm making use of VGG16 for this coursework which has 16 hidden layers. The network has a fixed 3x3 filter with stride 1, five MaxPool layers, each after a batch of convolution layers using a 2x2 kernel with stride 2. These help with the fine tuning of the network. ReLU activation has been applied to these layers, which are followed by the Dense layers having 4096 neurons and the last output with softmax activation.

2.2.2 GoogLeNet

GoogLeNet replaces the conventional convolution layers with blocks, each block being a network in itself. It uses Inception blocks and thus follows a similar architecture as inception. It has two base architectures viz, the naive version and the dimensionality reduction version. The naive one uses 3 different sized kernels 1x1, 3x3, 5x5 to perform concolutions combined with one max pooling layer. Outputs from these layers are concatenated to be passed on the module ahead. Adding an extra 1x1 convolution layer before the 3x3 and 5x5 convolution layer reduces the input channels for the next module, thus regulating the computation complexity of the model. This is the dimensionality reduction version, that saves a lot of computational cost and improves efficiency.

3. Experiments

3.1 Datasets

3.1.1 MNIST

Yann LeCun et al provide a widely used open dataset of handwritten digits, called MNIST(Modified National Institute of Standards and Technology)[3]. It is actually a subset of a much larger dataset NIST. It has a total of 70,000 images divided into 60,000 training images and the remaining 10,000 testing images. All the images are grayscale with a resolution of (28x28). MNIST is already available in keras.dataset library which can be directly imported in the code, ready to use.

3.1.2 CIFAR-10

CIFAR-10 is a smaller subset of a larger dataset which houses about 80 million tiny images. It contains 60,000 small resolution images divided into 50,000 training images and 10,000 testing images that belong to one 10 classes. The images are RGB with resolution of 32x32. Thus, there are 6000 images per class. Some examples of labels are cat, dog, frog, horse, etc. CIFAR-10 is also readily available in keras.dataset library, only needs to be imported.

3.2 Model Structure

3.2.1 VGG16

3.2.1.1 MNIST

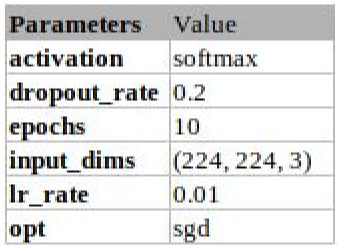
The original VGG network takes (224, 224, 3) shaped images as input and does not converge properly on other lower resolutions. Thus, the original MNIST images (28, 28, 1) needed to be reshaped, which I did with the help of padding. The network as defined in the paper has 4096 channels in the Dense layers, which I tried to change to 256 so that it could converge faster. And it did, much faster than the original 4096. I also changed the output layer with softmax activation to get the output using 10 neurons rather than the original 1000.

3.2.1.2 CIFAR-10

CIFAR images also come in small resolutions which do not go too well with VGG, hence I upscaled them to match requirements, ie. (32, 32, 3) to (224, 224, 3). As it can be seen that the images already have 3 channels so no stacking needed here. I tried to add a dropout to the last dense layer to ensure that the model does not overfit.

3.2.1.3 Evaluation

I have set up an evaluation environment which I have considered few fixed parameters and I am analysing the result on it. Below is what I have considered as my baseline model for VGG16.



Parameters	**Value**
activation	softmax
dropout_rate	0.2
epochs	10
input_dims	(224, 224, 3)
lr_rate	0.01
opt	sgd

Table 1: Different baseline parameters for VGG16

The above setup results in 99.51% test accuracy and 0.002 test loss on MNIST dataset while 72.13% test accuracy and 0.086 test loss on CIFAR10 dataset.

1. Using Different Optimisers (opt)

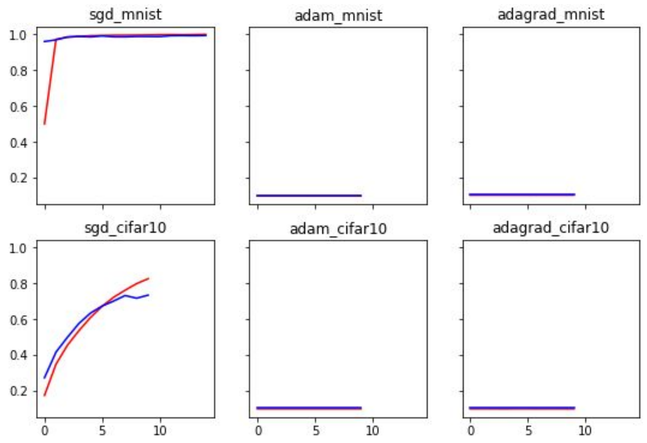


Figure 1: Top row - Train and Validation Accuracy on MNIST on different Optimisers, Bottom row - Train and Validation Accuracy on CIFAR10 different Optimisers for VGG16. Red line shows Train Accuracy, Blue line shows Validation accuracy.

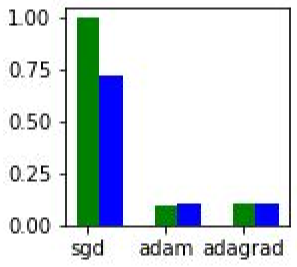


Figure 2: Test Accuracy using different Optimisers for VGG16

For the given baseline environment, when we use different optimizers we see that VGG16 trains well on stochastic gradient descent (sgd) on both the datasets. It achieves a test accuracy of 99.51% on MNIST and 72.13%. Other optimisers work well when the dimensions of the dataset is less and not 224x224x3 as we have padded with black colour.

2. Using Different Learning rates

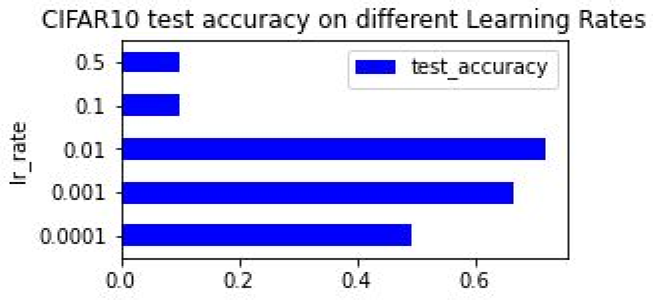


Figure 3: Test Accuracy on different learning rates for VGG16 on CIFAR10.

The above results are obvious, as we decrease the learning rate, and keep the epoch limited, we observe a decreasing trend of test accuracy and same goes the case with higher learning rate, the gradient keeps oscillating here.

3. Using Different Activation Function

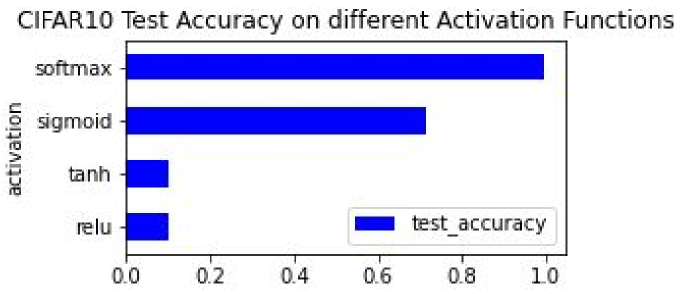


Figure 4: Test Accuracy on different activation functions for VGG16 on CIFAR10.

Here, I have changed the output layer activation function with sigmoid, tanh and relu. And it can be seen here, that tanh and relu fails dramatically on test accuracy scores while softmax and sigmoid achieves relevant results.

4. Using Different Dropout rate

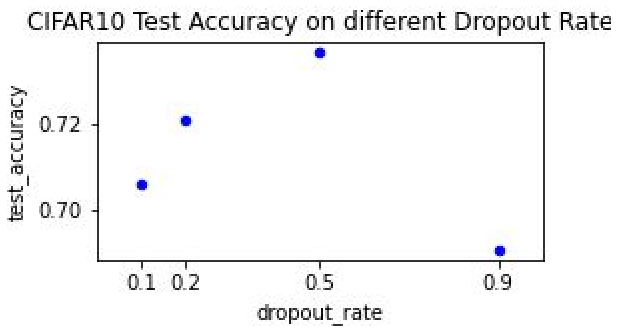


Figure 5: Test Accuracy on different Dropout rate for VGG16 on CIFAR10.

In figure 4, I have tried to change the dropout rate at the output layer, relevant results on test data are obtained when dropout is in between 0.2-0.4 as the test accuracy slowly increases then drops drastically on CIFAR10.

3.2.2 GoogLeNet

3.2.1.1 MNIST

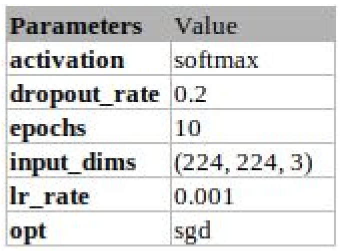
No noteworthy changes

3.2.1.2 CIFAR-10

No noteworthy changes

3.2.1.3 Evaluation

I have set up an evaluation environment which I have considered few fixed parameters and I am analysing the result on it. Below is what I have considered as my baseline model for GoogLeNet.



Parameters	Value
activation	softmax
dropout_rate	0.2
epochs	10
input_dims	(224, 224, 3)
lr_rate	0.001
opt	sgd

Table 1: Different baseline parameters for GoogLeNet

The above setup results in 96.00% test accuracy and 0.40 test loss on MNIST dataset while 63.06% test accuracy and 3.39 test loss on CIFAR10 dataset.

1. Using Different Optimisers (opt)

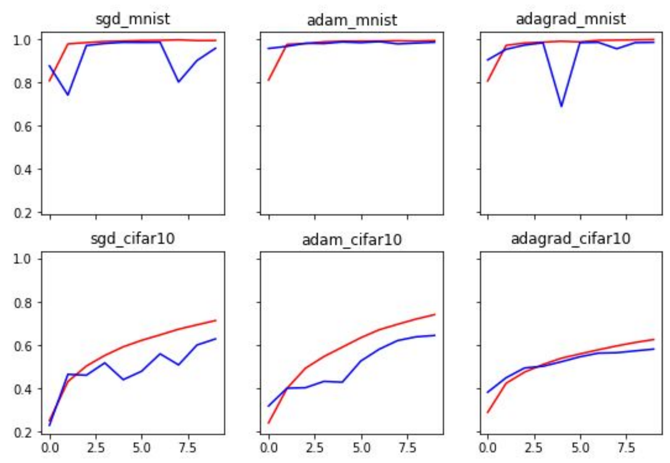


Figure 6: Top row - Train and Validation Accuracy on MNIST on different Optimisers, Bottom row - Train and Validation Accuracy on CIFAR10 different Optimisers for GoogLeNet. Red line shows Train Accuracy, Blue line shows Validation

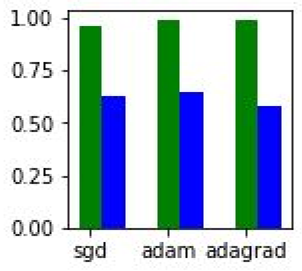


Figure 7: Test Accuracy using different Optimisers for GoogLeNet

In figure 6, we can see that GoogLeNet reaches a very good Train and Validation accuracy for MNIST dataset in 10 epochs, while the network is still learning in case of CIFAR10. In figure 7, we see good results of test accuracy for all the optimisers.

2. Using Different Learning rates

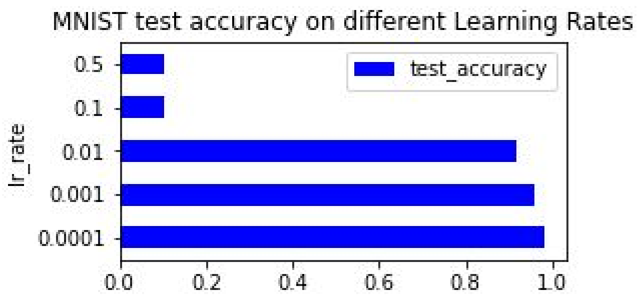


Figure 8: Test Accuracy on different learning rates for GoogLeNet.

GoogLeNet results on smaller learning rate is somewhat different from VGG16, here we see the network higher test accuracy on decreasing the learning rate.

3. Using Different Activation Function

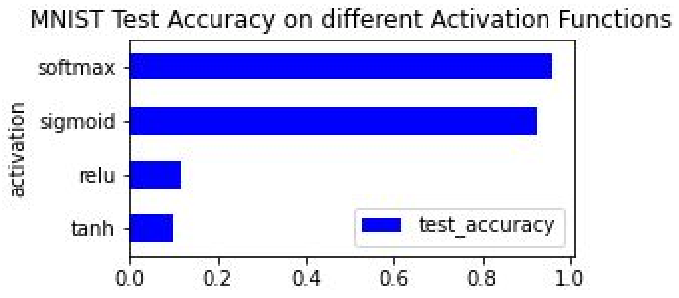


Figure 9: Test Accuracy on different activation functions for GoogLeNet.

Similar to VGG16, GoogLeNet, performs good on softmax and sigmoid, but fails on tanh and relu for the given baseline model.

4. Using Different Dropout rate

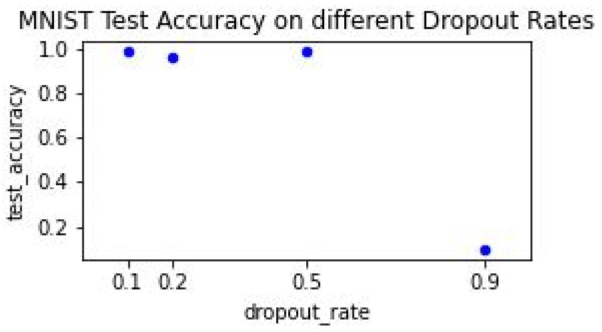


Figure 10: Test Accuracy on different Dropout rate for GoogLeNet.

The negative effect of dropout can be seen when the dropout rate is very high, otherwise GoogLeNet performs good with or without dropout.

3.3 Model Block Diagram

3.3.1 VGG

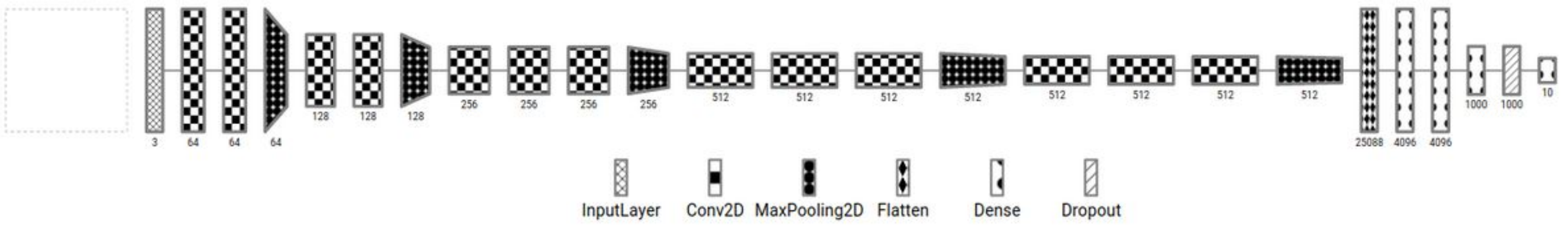


Figure 11: VGG16 network model created using https://viscom.net2vis.uni-ulm.de/

3.3.2 Inception

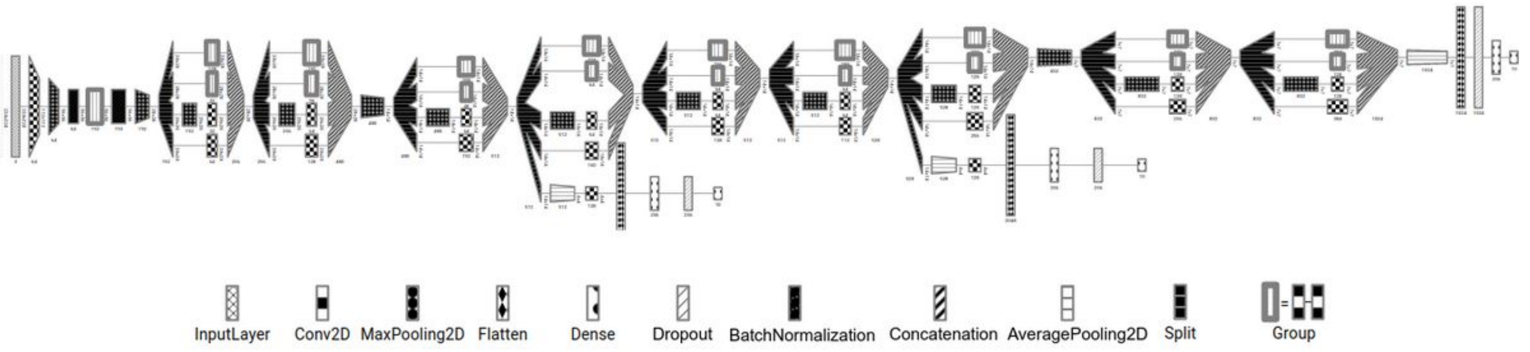


Figure 12: GoogLeNet network model created using https://viscom.net2vis.uni-ulm.de/

Conclusion

We have undertaken the study of VGG16 and GoogLeNet and understood various parameters affecting the performance of these networks on different datasets. These networks, although large utilise the full capacity of CNN architecture of DNN. Dense networks such as these affect the performance and time taken to run the model. GoogLeNet in this case performed better than VGG16, for almost all the parameters which I have checked. The auxiliary outputs produced by GoogLeNet were also relevant to the main results seen.

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]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