Deeper Networks for Image Classification

ECS795P Deep Learning and
Computer Vision
Jahnvi Sikligar
210538601

***Abstract*— Image Classification is a difficult task that involves categorizing images. To solve the image classification problem, deep learning models are frequently used. The goal of this paper is to investigate various Deep CNN architectures by implementing RedNet50 and VGG16 and perform image classification on MNIST and CIFAR10 datasets. For evaluation purpose both the architectures have been trained with similar epochs and batch sizes.**

Keywords—Image classification, ResNet50, VGG16, MNIST, CIFAR10, CNN

I. Introduction

Deep convolutional neural networks have made numerous advances in image classification . Deep convolution networks typically incorporate low/mid/high level highlights and classifiers in an end-to-end multilayer style, and the "levels" of features can be advanced by the number of stacked layers (depth). Fortunately, current GPUs, combined with exceptionally optimised 2D convolution implementations, are powerful enough to enable the training of large CNNs, and ongoing datasets, such as ImageNet, contain enough labelled examples to train models without significant overfitting. The following are the specific contributions to this report: implementation of two deep networks, VGG and ResNet. Deep neural networks were trained using the datasets MNIST and CIFAR10. In this paper, we review the experiments performed with the two models VGG and ResNet on the MNIST and CIFAR10 datasets, where the datasets and testing results are discussed.

II. Critical Analysis

AlexNet was the first to use Convolutional Networks in the field of Computer Vision. Its architecture was very similar to LeNet's, but it was deeper, larger, and featured Convolutional Layers stacked on top of each other instead of pooling layers after each convolution layer. ZFNet was created by altering the architecture hyperparameters, such as lowering the stride and filter size of the first layer and increasing the size of the convolutional layers in the middle of the network.

VGGNet, the runner-up in the 2014 ILSVRC, was introduced by Karen Simonyan and Andrew Zisserman. This network emphasised the importance of network depth in achieving good performance. VGG16, their final best network, has convolution/fully-connected layers and an extremely homogeneous architecture that only performs 3x3 convolutions and 2x2 pooling throughout the network. One of the issues with the VGGNet is the large number of parameters (140M), the majority of which are in the FC layer, which requires a lot more memory and is very expensive.

GoogLeNet, which was launched by Google in 2014, was the ILSVRC 2014 winning architecture. GoogLeNet introduced the concept of Inception Module, which dramatically reduced the number of parameters in the network (4M, compared to AlexNet's 60M) despite increasing the network's depth and width. GoogLeNet overcame VGGNet's limitations by using the Average Pooling layer instead of the Fully Connected layers at the top of the ConvNet, reducing the number of parameters without affecting network performance. So far, many improved variants of GoogLeNet have been released, the most recent being Inception-v4. Deep model training takes time, and due to a lack of data, such models are prone to overfitting.

ResNet (Residual Network) architecture was introduced in 2015 to improve training while making networks deeper. The degradation problem occurs as networks become deeper. The architecture uses special skip connections to connect output from the previous layer to the layer ahead, making the network deeper while ensuring that the error rate is not higher than the architecture's shallower versions.

III. Method / Model Description

A. VGG16

i. Description

VGGNet arose from the need to reduce the number of parameters in convolution layers and improve training efficiency. It is an example of a CNN that focuses on spatial exploitation. One of the main disadvantages of VGG is that it uses approximately 13 million parameters. To help reduce the number of variables, VGG employs a fixed filter of size 3x3 in the hidden layers. It is a 19-layered network that successfully demonstrated that the simultaneous placement of small sized filters such as (3x3) could produce the same effect as a large sized filter such as (5x5) or (7x7). A max-pooling layer placed after the convolutional layer aids in network tuning.

ii. Architecture

VGG is a 19-layer deep network that performs convolutions with fixed-size kernels. VGG's original architecture accepts a 224x224 RGB image as input and routes it through a network of hidden layers. The number of hidden layers is determined by the VGG variant used. For this course, VGG16, which has 16 hidden layers is being used. The network includes a fixed 3x3 filter with stride 1, five MaxPool layers, each following a batch of convolution layers with a 2x2 kernel and stride 2. These contribute to network fine tuning. These layers have been activated with ReLU, followed by Dense layers with 4096 neurons and the final output with softmax activation.

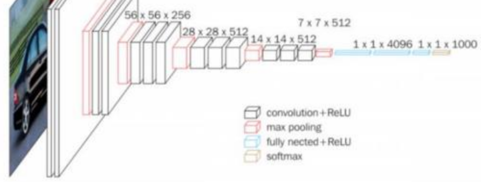


Fig A – VGG16 Architecture

B. *ResNet50*

i. Description

ResNet, or Residual Neural Network, is a novel architecture that features "skip connections" and heavy batch normalisation. Such skip connections are known as gated recurrent units or gated units, and they are similar to recent successful RNN elements. This method can train neural networks with 152 layers while remaining less complex than VGGNet. All ResNet configurations follow a similar configuration, with the only difference being the depth of building blocks (shown in brackets). From 18 layers (ResNet18) to 152 layers (ResNet152) (ResNet152).

ii. Architecture

The ResNet50 architecture is based on the above model, but there is one significant difference. Due to concerns about the time required to train the layers, the building block was modified into a bottleneck design in this case. Instead of the previous two layers, this time a three-layer stack was used. As a result, each of the ResNet34's 2-layer blocks was replaced with a 3-layer bottleneck block, resulting in the ResNet 50 architecture. This model is much more accurate than the 34-layer ResNet model. ResNet's 50-layer performance is 3.8 billion FLOPS.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Fig B – ResNet50 Architecture

C. *Datasets*

i. MNIST

The MNIST (Modified National Institute of Standards and Technology) database of handwritten digits 0 to 9 contains 60,000 training examples and 10,000 test examples. Each image is 28 x 28 pixels in size. There are a total of ten classes that correspond to the digits [0-9]. All of the images are one-channel grayscale.

ii. CIFAR-10

CIFAR10 is made up of 60000 coloured images of size 32x32 (50,000 in the training set and 10,000 in the test set) (3-channels). There are ten classes, each with 60000 images. Airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck are the class names.

D. *Data Augmentation*

Image rotation is a popular augmentation technique that allows the model to become insensitive to object orientation. By passing an integer value in the rotation range argument to the ImageDataGenerator class, you can randomly rotate images through any degree between 0 and 360.

E. *Parameter Tuning*

i. Optimizers

VGGNet and ResNet are trained using various optimizers such as Stochastic Gradient Descent, momentum addition, and the ADAM optimizer. ADAM is commonly regarded as an improvement over RMSprop for improving model performance.

ii. Epochs and Batch Size

For all models, the batch size and epochs are currently fixed at 128 and 50, respectively. To train at the maximum batch size capacity for each model, batch size is reduced. Increasing the batch size improves model accuracy at the expense of computation cost.

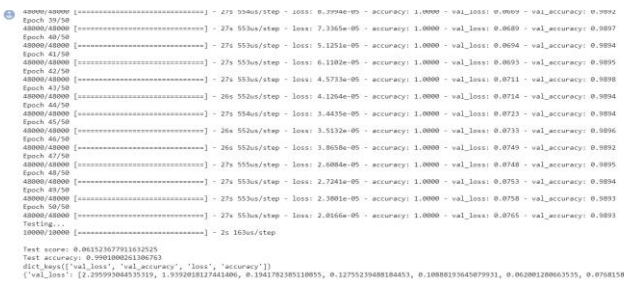
IV. Experiments

The models in this paper have been trained on both MNIST and CIFAR10 datasets. The results of them have been discussed below with same configurations for all the models. Configurations:- Epochs: 50, Batch size: 128.

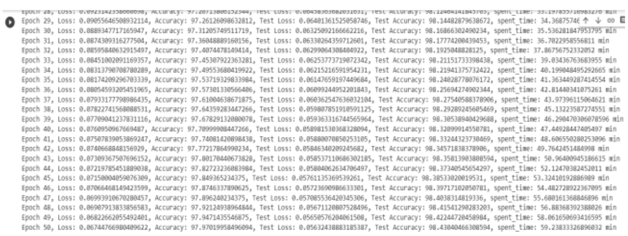
A. *MNIST*

i. Training

a. VGG 16



b. ResNet10



ii. Testing

a. VGG 16

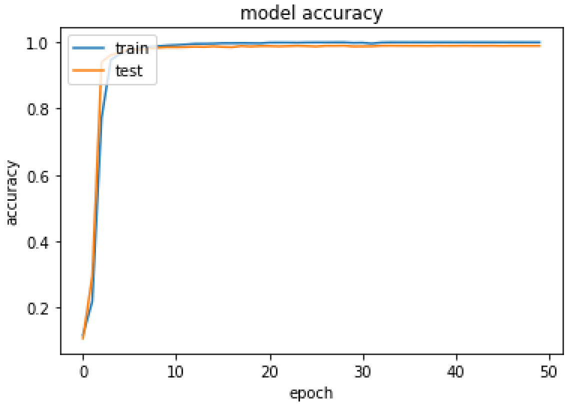


Fig. C – Graph of Testing accuracy vs Epoch

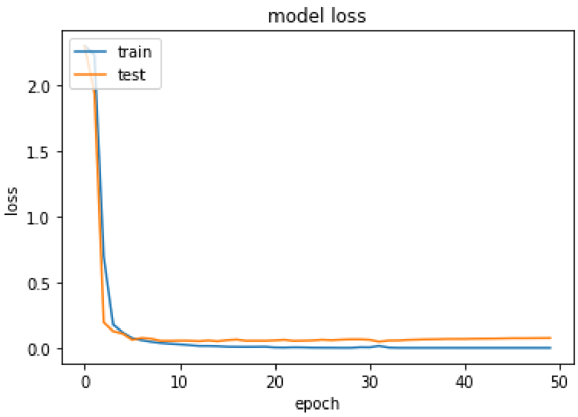


Fig. D – Graph of Testing loss vs Epoch

b. ResNet10

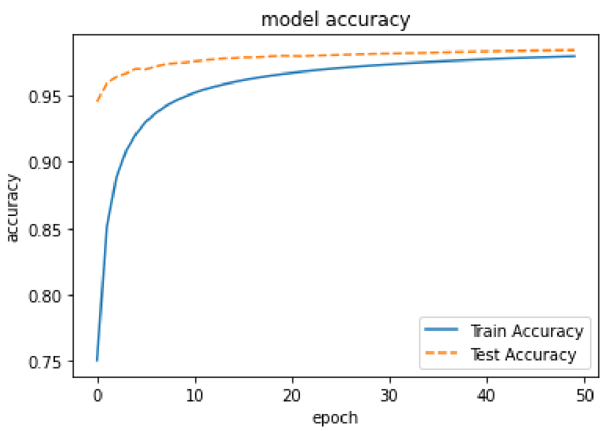


Fig. D – Graph of Testing accuracy vs Epoch

According to the environment settings and execution output, the ResNet50 model achieved training accuracy of 97.97% and test accuracy of 98.43% on Epoch 50. The **Table I** summarizes the '*Training accuracy*' and '*Testing accuracy*' of both VGG16 and ResNet50 models respectively.

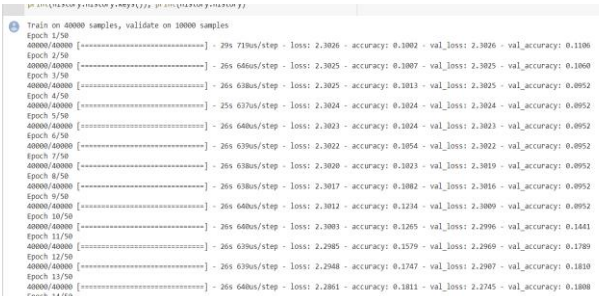
TABLE I.

MNIST Dataset		
Models	***Training***	***Testing***
VGG16	98.9300%	99.0100 %
ResNet50	97.9701 %	98.4304 %

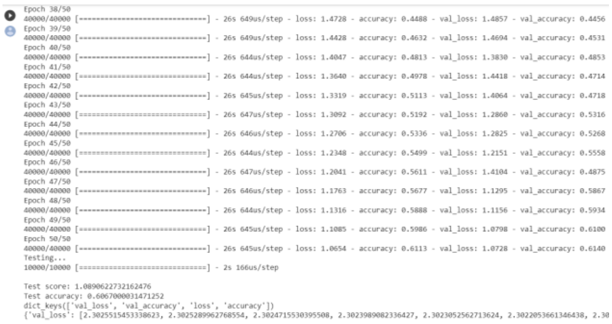
B. *CIFAR-10*

i. Training

a. VGG 16



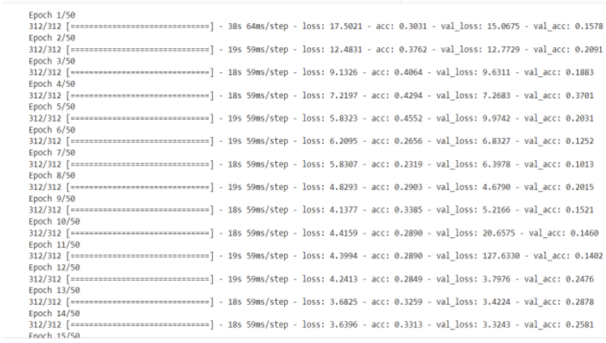
```
Train on 40000 samples, validate on 10000 samples
Epoch 1/50
40000/40000 [==============================] - 29s 719us/step - loss: 2.3026 - accuracy: 0.1002 - val_loss: 2.3026 - val_accuracy: 0.1106
Epoch 2/50
40000/40000 [==============================] - 26s 646us/step - loss: 2.3025 - accuracy: 0.1007 - val_loss: 2.3025 - val_accuracy: 0.1060
Epoch 3/50
40000/40000 [==============================] - 26s 638us/step - loss: 2.3025 - accuracy: 0.1013 - val_loss: 2.3025 - val_accuracy: 0.0952
Epoch 4/50
40000/40000 [==============================] - 25s 637us/step - loss: 2.3024 - accuracy: 0.1024 - val_loss: 2.3024 - val_accuracy: 0.0952
Epoch 5/50
40000/40000 [==============================] - 26s 640us/step - loss: 2.3023 - accuracy: 0.1024 - val_loss: 2.3023 - val_accuracy: 0.0952
Epoch 6/50
40000/40000 [==============================] - 26s 639us/step - loss: 2.3022 - accuracy: 0.1054 - val_loss: 2.3022 - val_accuracy: 0.0952
Epoch 7/50
40000/40000 [==============================] - 26s 638us/step - loss: 2.3020 - accuracy: 0.1023 - val_loss: 2.3019 - val_accuracy: 0.0952
Epoch 8/50
40000/40000 [==============================] - 26s 638us/step - loss: 2.3017 - accuracy: 0.1082 - val_loss: 2.3016 - val_accuracy: 0.0952
Epoch 9/50
40000/40000 [==============================] - 26s 640us/step - loss: 2.3012 - accuracy: 0.1234 - val_loss: 2.3009 - val_accuracy: 0.0952
Epoch 10/50
40000/40000 [==============================] - 26s 640us/step - loss: 2.3003 - accuracy: 0.1265 - val_loss: 2.2996 - val_accuracy: 0.1441
Epoch 11/50
40000/40000 [==============================] - 26s 639us/step - loss: 2.2985 - accuracy: 0.1579 - val_loss: 2.2969 - val_accuracy: 0.1789
Epoch 12/50
40000/40000 [==============================] - 26s 639us/step - loss: 2.2948 - accuracy: 0.1747 - val_loss: 2.2907 - val_accuracy: 0.1810
Epoch 13/50
40000/40000 [==============================] - 26s 640us/step - loss: 2.2861 - accuracy: 0.1811 - val_loss: 2.2745 - val_accuracy: 0.1808
```



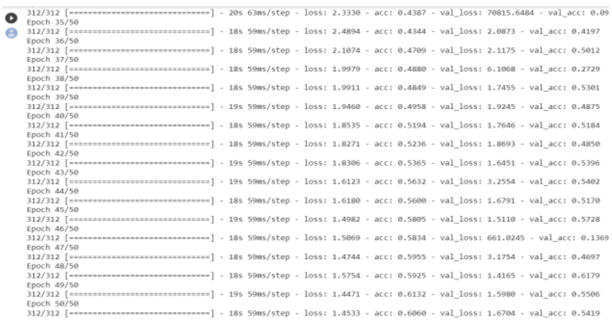
```
Epoch 38/50
40000/40000 [==============================] - 26s 649us/step - loss: 1.4728 - accuracy: 0.4488 - val_loss: 1.4857 - val_accuracy: 0.4456
Epoch 39/50
40000/40000 [==============================] - 26s 649us/step - loss: 1.4428 - accuracy: 0.4632 - val_loss: 1.4694 - val_accuracy: 0.4531
Epoch 40/50
40000/40000 [==============================] - 26s 644us/step - loss: 1.4047 - accuracy: 0.4813 - val_loss: 1.3830 - val_accuracy: 0.4853
Epoch 41/50
40000/40000 [==============================] - 26s 644us/step - loss: 1.3640 - accuracy: 0.4978 - val_loss: 1.4418 - val_accuracy: 0.4714
Epoch 42/50
40000/40000 [==============================] - 26s 645us/step - loss: 1.3319 - accuracy: 0.5113 - val_loss: 1.4064 - val_accuracy: 0.4718
Epoch 43/50
40000/40000 [==============================] - 26s 647us/step - loss: 1.3092 - accuracy: 0.5192 - val_loss: 1.2860 - val_accuracy: 0.5316
Epoch 44/50
40000/40000 [==============================] - 26s 646us/step - loss: 1.2706 - accuracy: 0.5336 - val_loss: 1.2825 - val_accuracy: 0.5268
Epoch 45/50
40000/40000 [==============================] - 26s 644us/step - loss: 1.2348 - accuracy: 0.5499 - val_loss: 1.2151 - val_accuracy: 0.5558
Epoch 46/50
40000/40000 [==============================] - 26s 647us/step - loss: 1.2041 - accuracy: 0.5611 - val_loss: 1.4104 - val_accuracy: 0.4875
Epoch 47/50
40000/40000 [==============================] - 26s 646us/step - loss: 1.1763 - accuracy: 0.5677 - val_loss: 1.1295 - val_accuracy: 0.5867
Epoch 48/50
40000/40000 [==============================] - 26s 644us/step - loss: 1.1316 - accuracy: 0.5888 - val_loss: 1.1156 - val_accuracy: 0.5934
Epoch 49/50
40000/40000 [==============================] - 26s 645us/step - loss: 1.1085 - accuracy: 0.5986 - val_loss: 1.0798 - val_accuracy: 0.6100
Epoch 50/50
40000/40000 [==============================] - 26s 645us/step - loss: 1.0654 - accuracy: 0.6113 - val_loss: 1.0728 - val_accuracy: 0.6140
Testing...
10000/10000 [==============================] - 2s 166us/step

Test score: 1.0890622732162476
Test accuracy: 0.6067000031471252
dict_keys(['val_loss', 'val_accuracy', 'loss', 'accuracy'])
{'val_loss': [2.3025515453338623, 2.3025289962768554, 2.3024715530395508, 2.3023989082336427, 2.3023052562713624, 2.3022053661346438, 2.30
```

b. ResNet10



```
Epoch 1/50
312/312 [==============================] - 38s 64ms/step - loss: 17.5021 - acc: 0.3031 - val_loss: 15.0675 - val_acc: 0.1578
Epoch 2/50
312/312 [==============================] - 19s 59ms/step - loss: 12.4831 - acc: 0.3762 - val_loss: 12.7729 - val_acc: 0.2091
Epoch 3/50
312/312 [==============================] - 18s 59ms/step - loss: 9.1326 - acc: 0.4064 - val_loss: 9.6311 - val_acc: 0.1883
Epoch 4/50
312/312 [==============================] - 18s 59ms/step - loss: 7.2197 - acc: 0.4294 - val_loss: 7.2683 - val_acc: 0.3701
Epoch 5/50
312/312 [==============================] - 19s 59ms/step - loss: 5.8323 - acc: 0.4552 - val_loss: 9.9742 - val_acc: 0.2031
Epoch 6/50
312/312 [==============================] - 19s 59ms/step - loss: 6.2095 - acc: 0.2656 - val_loss: 6.8327 - val_acc: 0.1252
Epoch 7/50
312/312 [==============================] - 18s 59ms/step - loss: 5.8307 - acc: 0.2319 - val_loss: 6.3978 - val_acc: 0.1013
Epoch 8/50
312/312 [==============================] - 19s 59ms/step - loss: 4.8293 - acc: 0.2903 - val_loss: 4.6790 - val_acc: 0.2015
Epoch 9/50
312/312 [==============================] - 18s 59ms/step - loss: 4.1377 - acc: 0.3385 - val_loss: 5.2166 - val_acc: 0.1521
Epoch 10/50
312/312 [==============================] - 18s 59ms/step - loss: 4.4159 - acc: 0.2890 - val_loss: 20.6575 - val_acc: 0.1460
Epoch 11/50
312/312 [==============================] - 19s 59ms/step - loss: 4.3994 - acc: 0.2890 - val_loss: 127.6330 - val_acc: 0.1402
Epoch 12/50
312/312 [==============================] - 19s 59ms/step - loss: 4.2413 - acc: 0.2849 - val_loss: 3.7976 - val_acc: 0.2476
Epoch 13/50
312/312 [==============================] - 18s 59ms/step - loss: 3.6825 - acc: 0.3259 - val_loss: 3.4224 - val_acc: 0.2878
Epoch 14/50
312/312 [==============================] - 18s 59ms/step - loss: 3.6396 - acc: 0.3313 - val_loss: 3.3243 - val_acc: 0.2581
```



```
312/312 [==============================] - 20s 63ms/step - loss: 2.3330 - acc: 0.4387 - val_loss: 70815.6484 - val_acc: 0.09
Epoch 35/50
312/312 [==============================] - 18s 59ms/step - loss: 2.4894 - acc: 0.4344 - val_loss: 2.0873 - val_acc: 0.4197
Epoch 36/50
312/312 [==============================] - 18s 59ms/step - loss: 2.1074 - acc: 0.4709 - val_loss: 2.1175 - val_acc: 0.5012
Epoch 37/50
312/312 [==============================] - 18s 59ms/step - loss: 1.9979 - acc: 0.4880 - val_loss: 6.1068 - val_acc: 0.2729
Epoch 38/50
312/312 [==============================] - 18s 59ms/step - loss: 1.9911 - acc: 0.4849 - val_loss: 1.7455 - val_acc: 0.5301
Epoch 39/50
312/312 [==============================] - 19s 59ms/step - loss: 1.9460 - acc: 0.4958 - val_loss: 1.9245 - val_acc: 0.4875
Epoch 40/50
312/312 [==============================] - 18s 59ms/step - loss: 1.8535 - acc: 0.5194 - val_loss: 1.7646 - val_acc: 0.5184
Epoch 41/50
312/312 [==============================] - 18s 59ms/step - loss: 1.8271 - acc: 0.5236 - val_loss: 1.8693 - val_acc: 0.4850
Epoch 42/50
312/312 [==============================] - 19s 59ms/step - loss: 1.8306 - acc: 0.5365 - val_loss: 1.6451 - val_acc: 0.5396
Epoch 43/50
312/312 [==============================] - 19s 59ms/step - loss: 1.6123 - acc: 0.5632 - val_loss: 3.2554 - val_acc: 0.5402
Epoch 44/50
312/312 [==============================] - 18s 59ms/step - loss: 1.6180 - acc: 0.5600 - val_loss: 1.6791 - val_acc: 0.5170
Epoch 45/50
312/312 [==============================] - 19s 59ms/step - loss: 1.4982 - acc: 0.5805 - val_loss: 1.5110 - val_acc: 0.5728
Epoch 46/50
312/312 [==============================] - 18s 59ms/step - loss: 1.5069 - acc: 0.5834 - val_loss: 661.0245 - val_acc: 0.1369
Epoch 47/50
312/312 [==============================] - 18s 59ms/step - loss: 1.4744 - acc: 0.5955 - val_loss: 3.1754 - val_acc: 0.4697
Epoch 48/50
312/312 [==============================] - 18s 59ms/step - loss: 1.5754 - acc: 0.5925 - val_loss: 1.4165 - val_acc: 0.6179
Epoch 49/50
312/312 [==============================] - 19s 59ms/step - loss: 1.4471 - acc: 0.6132 - val_loss: 1.5980 - val_acc: 0.5506
Epoch 50/50
312/312 [==============================] - 18s 59ms/step - loss: 1.4533 - acc: 0.6060 - val_loss: 1.6704 - val_acc: 0.5419
```

ii. Testing

a. VGG 16

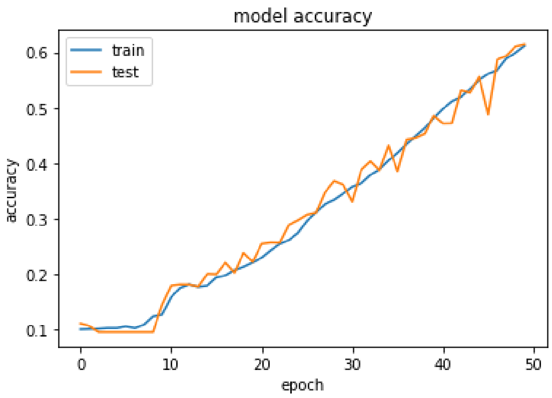


Fig. E – Graph of Testing accuracy vs Epoch

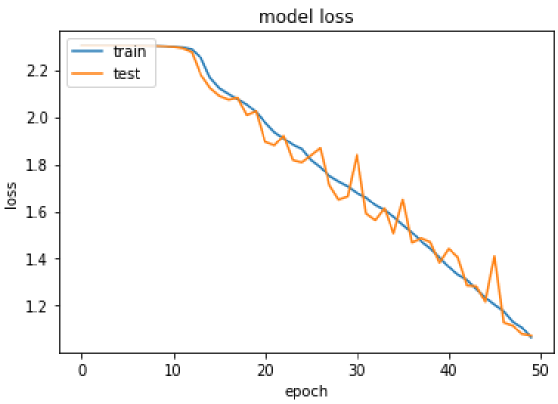
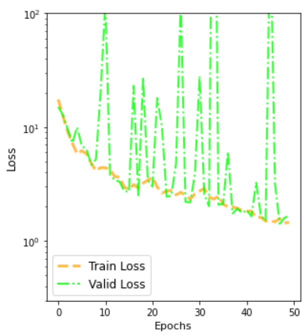


Fig. F – Graph of Testing accuracy vs Epoch

b. ResNet10



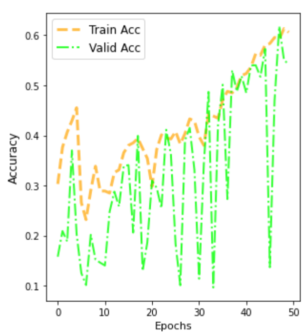


Fig. G – Graph of Testing accuracy vs Epoch (right)
Fig. H – Graph of Testing loss vs Epoch (left)
Note: 'Validation' here refers to 'Testing'

Classification Report:

	precision	recall	f1-score	support
airplane	0.22	0.84	0.35	1000
automobile	0.69	0.60	0.64	1000
bird	0.73	0.31	0.44	1000
cat	0.67	0.02	0.04	1000
deer	0.81	0.14	0.23	1000
dog	0.00	0.00	0.00	1000
frog	0.49	0.87	0.63	1000
horse	0.55	0.76	0.64	1000
ship	0.85	0.43	0.57	1000
truck	0.74	0.74	0.74	1000
accuracy			0.47	10000
macro avg	0.57	0.47	0.43	10000
weighted avg	0.57	0.47	0.43	10000

Fig. I – Further evaluation

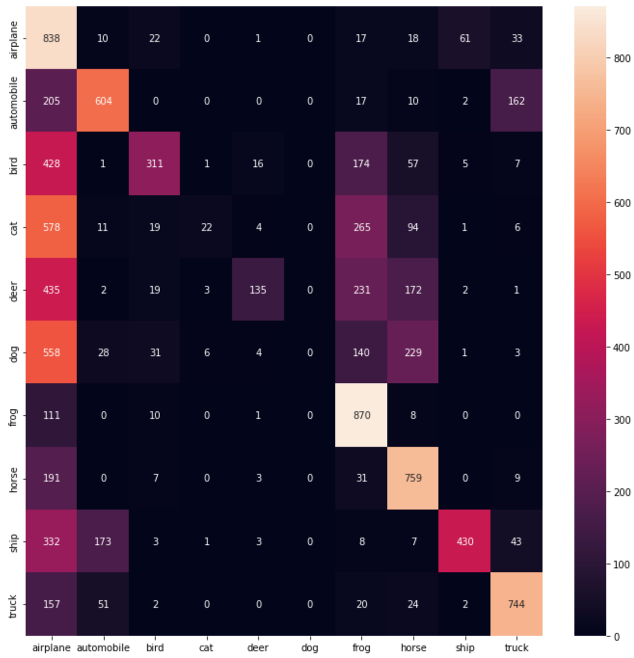


Fig. J – Heatmap for Confusion Matrix

According to the environment settings and execution output, the VGG16 model achieved training accuracy of 61.40% and test accuracy of 60.67% on Epoch 50. The **Table II** summarizes the '*Training accuracy*' and '*Testing accuracy*' of both VGG16 and ResNet50 models respectively.

TABLE II.

CIFAR-10 Dataset		
Models	***Training***	***Testing***
VGG16	61.4000 %	60.6700 %
ResNet50	54.1900 %	53.9200 %

V. Conclusion

We investigated the performance of VGG16 and ResNet50 networks on different datasets while using the same parameter tuning. These networks, despite their size, make full use of the CNN architecture of DNN. Dense networks like these have an impact on the model's performance and run time. The results show that both the VGG-16 and ResNet50 deep convolutional neural networks can achieve high accuracy results on the MNIST dataset using purely supervised learning. To keep the experiment simple, both models were tested with MNIST datasets to see how well they performed. In comparison to the VGG-16 model, the ResNet50 model has a faster training rate and better early epoch results.

Further Evaluation

It will be interesting to compare the performance of the vgg-16 and resnet50 models on a difficult dataset such as ImageNet in the future. Increasing the layer count to upgrade to vgg19 and ResNet152 models and observing performance with various types of datasets. Alternately, you can evaluate model performance by adding or removing a layer.

References

[1] Krizhevsky, A., Sutskever, I. and Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).

[2] Rawat, W. and Wang, Z., 2017. Deep convolutional neural networks for image classification: A comprehensive review. Neural computation, 29(9), pp.2352-2449.

[3] Zeiler, M.D. and Fergus, R., 2014, September. Visualizing and understanding convolutional networks. In European conference on computer vision (pp. 818-833). Springer, Cham.

[4] Simonyan, K. and Zisserman, A., 2014. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

[5] He, K., Zhang, X., Ren, S. and Sun, J., 2016. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).