Deep Networks for Image Classification

Arvind R Menon 190715232

1. Introduction

The evolution of Deep Learning CNNs is well depicted in Appendix 8.1. From its origins back in 1980's, with LeCuN et al proposing the first multi-layered CNN in 1989 to the rise and rapid innovation in CNN architectures, CNN has a roller coaster ride with it's ups and downs. Since 2012, CNN architectures have been constantly improving at an exponential pace. This probably roots back to the fact that computation has become faster and more accessible. Great strides have been made in developing faster GPUs which has played a key role in the above-mentioned growth.

1. Critical Analysis[1]

1.1. Trend: LeNet, VGG,[2] Inception[3] and ResNet[4] are among the pillars on which most of the modern-day architectural innovations have been built. While the list is not exhaustive below are a few innovations made since the publication of the ResNet paper:

1.1.1. Depth Based: The best performance in Image Recognition have been centred around Very deep convolutional networks in recent years

1.1.1.1. Inception V4 and Inception ResNet[5]**:** Inception Networks' training accelerates significantly with the inclusion of residual networks. The performance is also improved. Inception-v4 which has a more uniform simplified architecture and more inception modules than Inception-v3.

1.1.2. Multi Path Based:

1.1.2.1. DenseNet:[6] DenseNet's architecture explicitly differentiates between information that is added to the network and information that is preserved. DenseNets alleviate the vanishing-gradient problem, strengthen feature propagation, encourage feature reuse, and substantially reduce the number of parameters.

1.1.3. Width Based:

1.1.3.1. Wide ResNet:[7] With a wider yet thinner network compared to ResNet, Wide ResNet achives better performance.

1.1.3.2. Xception:[8] Inception modules replace depthwise separable convolutions and slightly outperform Inception V3 on ImageNet Dataset.

1.1.3.3. ResNeXt[9]**:** ResNeXt uses a new dimension, "cardinality" (the size of the set of transformations), as an important dimension along with depth and width. As a result performance is improved over traditional ResNet.

1.1.4. Feature-Map (ChannelFMap) Exploitation based CNNs:

1.1.4.1. Squeeze and Excitation Network

1.1.5. Channel (Input) Exploitation based CNNs

1.1.6. Attention based CNNs:

1.1.6.1. Residual Attention Neural Network:[10] Residual Attention Networks proposes Residual Additional Network, by stacking Attention Modules, for training. The feedforward and feedback attention process unfolded into a single feedforward process using bottom-up top-down feedforward structure in each Attention Module.

1.2. Some of the remaining problems are:[11], [12]

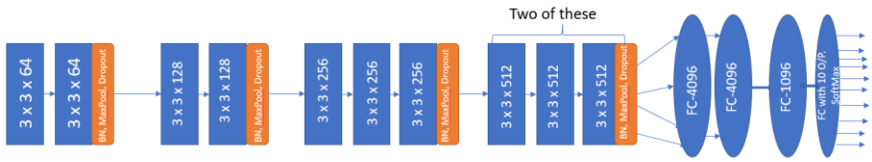
- An increase in the same could improve performance and encourage architectural innovations.
- Generalization and robustness on diverse categories of images can be improved by combining multiple and diverse architectures.
- Exploiting small and ultra-large-scale data
- Integrating common sense and Comprehensive Scene Understanding: Rather than being purely data-driven, it would be amazing if deep learning models can comprehend like human beings and incorporate common-sense.
- An additional direction to mention is meta learning, which aims to learn the learning process.
- Integrating common sense and comprehensive Scene Understanding is a problem that grabs my interest. If successful, traffic cams or CCTV can be used to automatically detect crime or emergency situations. Also, I feel it would be really cool if image recognition can be used to detect areas with harmful pathogen which otherwise are not visible to human eyes.

2. <u>Model Description:</u>

For this project I have used VGG16, ResNet30 and ResNet56 to train and test the MNIST handwritten digits dataset and CIFAR-10 dataset, respectively.

2.1. <u>**VGG16:**</u> A simple yet elegant model that leverages on stacking of multiple small receptive fields to achieve larger receptive fields like stack 2 receptive fields of 3x3 filter yields the result of a 5x5 filter. The biggest advantage is keeping the parameters comparatively less yet having a pretty deep network.

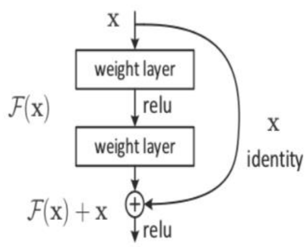
2.1.1. Architecture: The Diagram below depicts the model used for the respective dataset.



VGG16 architecture MNIST

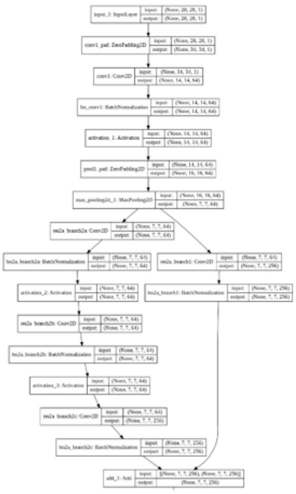
VGG16 architecture CIFAR10

2.2. ResNet: Residual Networks or ResNet was an eye opener. It opened many new avenues with the introduction of the "skip connection". It essentially helped in alleviating the "vanishing gradient problem". In my experiments, I have used a slightly modified version of ResNet30 for the MNIST dataset. For my trials with the CIFAR-10 dataset, ResNet20 and ResNet56 have been used. There are two versions to the models applied for CIFAR-10 dataset. V2[13] slightly outperforms V1[4]. The basic structure is as below. Note, the "Identity" is the import part in this architecture.

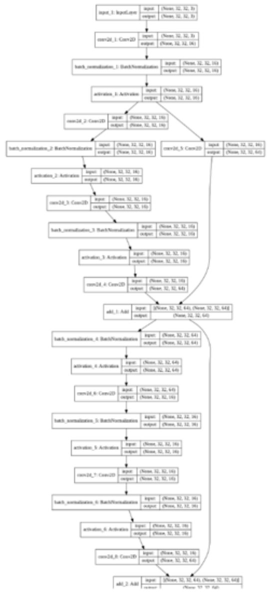


2.2.1. Architecture:[14][15]The real images have been added in the folder. Since the real image of the architecture is huge, a snippet of the different architectures used is shown below:

ResNet56V1



ResNet20-used for MNIST

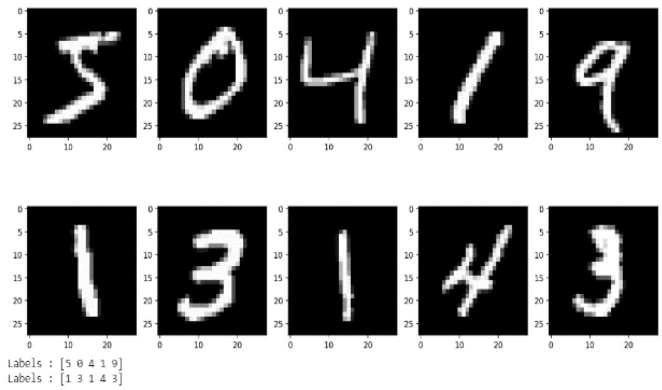


ResNet56V2

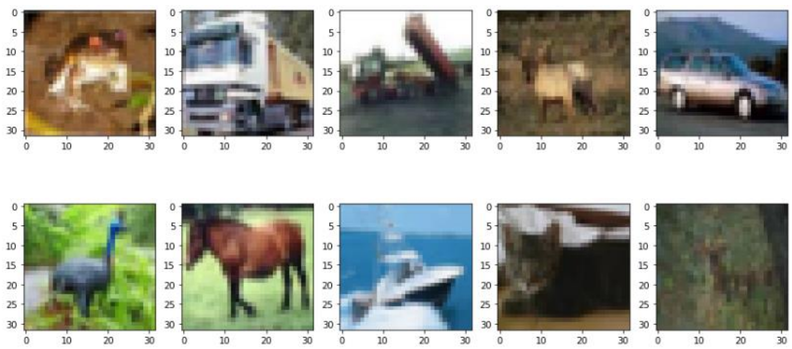
3. <u>Experiments:</u>

3.1. <u>Datasets:</u>

3.1.1. MNIST: This is a dataset of 60,000 28x28 grayscale images of the 10 digits, along with a test set of 10,000 images. The digits are handwritten. It is, by many considered, the “Hello World” dataset for Neural Networks. For more info: http://yann.lecun.com/exdb/mnist/.



3.1.2. CIFAR-10: The CIFAR-10 dataset has 60000 32x32 colour images, 3 channels, in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.
The dataset is divided into five training batches and one test batch, each with 10000 images. Truth be told even I have made mistakes in classifying a few samples.



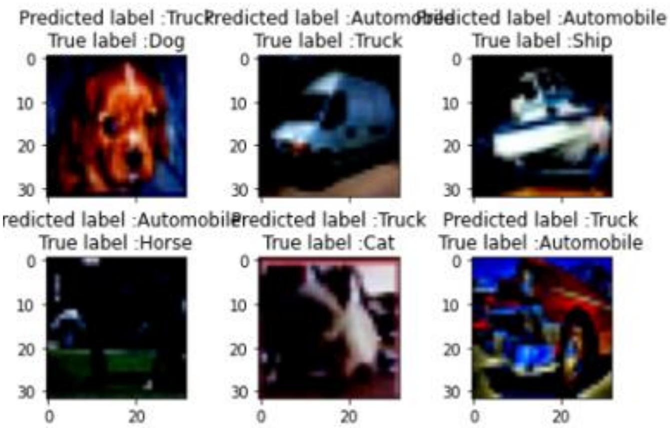
3.2. Training: The MNIST dataset was trained on 60,000 images. The CIFAR 10 was divided into Training, Validation and Test dataset. The images from both the datasets were normalized before training, validation, and testing. Some training values are not at full epoch due to truncation in Colab

Model	Epochs	Batch Size	Training Accuracy	Validation Accuracy
VGG16 on MNIST	20	64	98.38%	
VGG16 on CIFAR10	100	64	87.57%	87.72%
ResNet20 on MNIST	20	16	99.80%	99.09%
ResNet20 on CIFAR10 V1	200	64	98.48%(124th epoch)	90.94%(124th epoch)
ResNet20 on CIFAR-10 V2	200	64	95.10% (84th epoch)	91.10%(84th epoch)
ResNet56 on CIFAR10 V1	50 x 4	128	95.41%	85.90%
ResNet56 on CIFAR10 V2	50 x 4	128	95.67%	88.22%

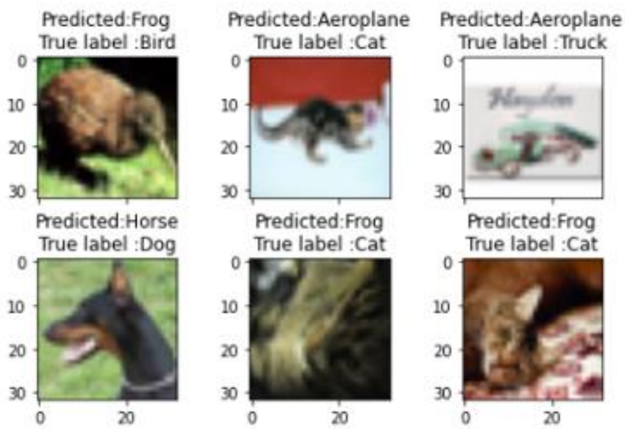
3.3. Testing: All the ResNet models were tested with 10000 samples. VGG16 for CIFAR10 and MNIST were trained with 5000 and 6000 samples respectively.

Model	Epochs	Batch Size	Test Accuracy
VGG16 on MNIST	20	64	98.98%
VGG16 on CIFAR10	100	64	86.49
ResNet20 on MNIST	20	16	99%
ResNet20 on CIFAR10 V1	200	64	90.56%
ResNet20 on CIFAR-10 V2	200	64	90.98%
ResNet56 on CIFAR10 V1	50 x 4	128	85.37%
ResNet56 on CIFAR10 V2	50 x 4	128	87.51%

Above are the numbers, to visualize a few errors, let's compare the errors on CIFAR10 by VGG16 and ResNet56



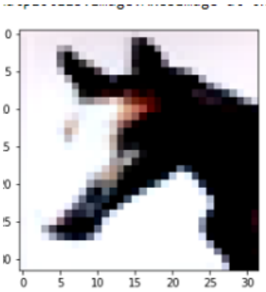
CIFAR10-ResNet56 V2



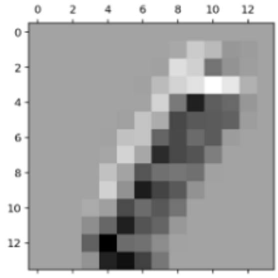
CIFAR10-VGG16

3.4. Activation Layer:

The initial activation layers of MNIST and CIFAR can be viewed as below:



Activation for a horse in ResNet56 V2



Activation for number 1 in ResNet20

4. Experiment Analysis:

4.1. Precision and Recall:

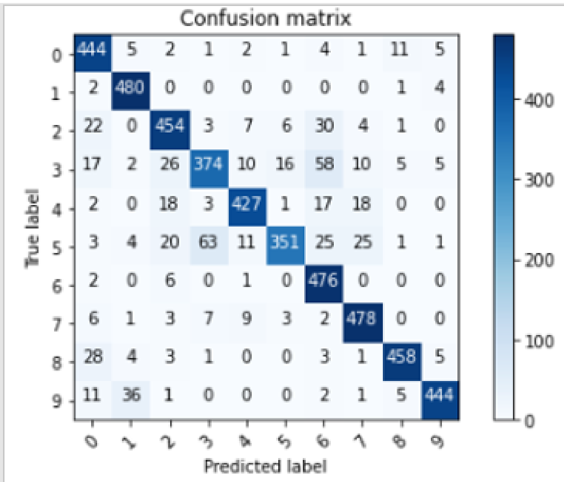
	precision	recall	f1-score	support
Class 0	1.00	0.99	1.00	624
Class 1	1.00	0.99	0.99	654
Class 2	0.98	1.00	0.99	572
Class 3	0.99	1.00	1.00	589
Class 4	1.00	0.98	0.99	580
Class 5	1.00	0.98	0.99	551
Class 6	0.98	0.99	0.99	580
Class 7	0.99	0.98	0.99	633
Class 8	0.98	0.99	0.99	585
Class 9	0.97	0.99	0.98	632
accuracy			0.99	6000
macro avg	0.99	0.99	0.99	6000
weighted avg	0.99	0.99	0.99	6000

VGG16-MNIST

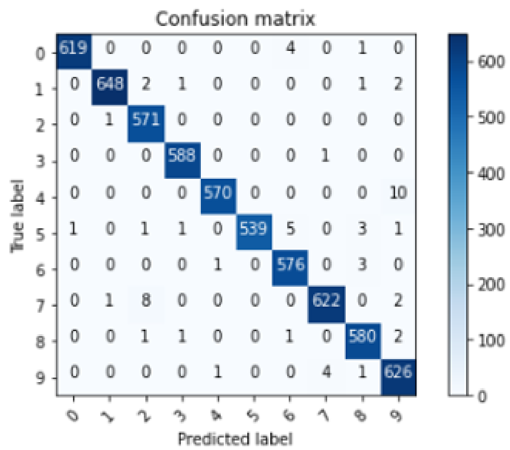
	precision	recall	f1-score	support
Class 0	0.99	1.00	1.00	980
Class 1	1.00	1.00	1.00	1135
Class 2	0.99	1.00	0.99	1032
Class 3	0.99	1.00	0.99	1010
Class 4	1.00	0.98	0.99	982
Class 5	0.99	0.99	0.99	892
Class 6	1.00	0.99	0.99	958
Class 7	0.99	0.99	0.99	1028
Class 8	0.99	0.98	0.99	974
Class 9	0.98	0.99	0.98	1009
accuracy			0.99	10000
macro avg	0.99	0.99	0.99	10000
weighted avg	0.99	0.99	0.99	10000

ResNet20-MNIST

4.2. Confusion Matrix:



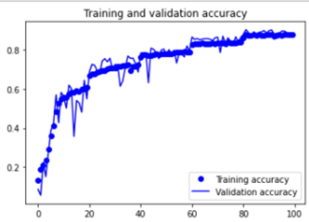
VGG16-CIFAR10



VGG16-MNIST

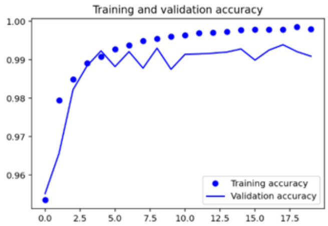
4.3. Accuracy and Loss Graph: The models converge in a few epochs

ResNet56 CIFAR10 V1

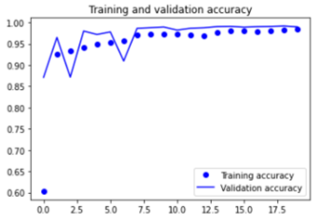


VGG16 CIFAR10

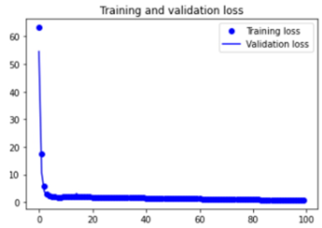
ResNet MNIST



ResNet20 MNIST

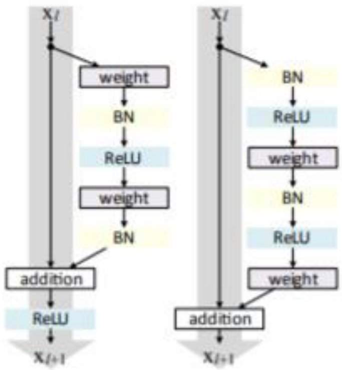


VGG16 MNIST



VGG16 CIFAR10

5. Improvements:



ResNet V1 ResNet V2

Techniques like Data Augmentation(ResNet CIFAR models and all VGG models), BatchNormalization (All models), Dropout(All VGG) has been added to the initial architecture to improve performance.

ResNet V2 based on [13] has also been implemented. An identity mapping function f(yl) = yl is construction where the activation functions (ReLU and BN) is perceived as "pre-activation" of the weight layers. This contrasts with the conventional practice of "post-activation". A new residual unit design is thus achieved

6. Conclusion:

VGG and ResNet as mentioned before are two milestones in Deep Neural Networks journey. They form the ideal launchpad for someone to learn Deep CNN architectures. VGG16 being the simpler model is faster and does amazingly well with the MNIST dataset. However, it apparently is not the ideal model for the CIFAR10 data set. ResNet on the other hand is a more intriguing architecture which is deeper and requires lot more computational power and training time as compare to VGG. However, ResNet performs better that VGG and gives better accuracy. Giving good amount of training, ResNet gives higher accuracy for CIFAR10 dataset as well. Among the models, ResNet Version 2 performs better than Version 1.

Please note that an Inception Model notebook has been included in the folder. However, since it was not properly visualised, the same has not been included in the Final Report. ResNet56 CIFAR10 V2 only was run only for two epochs after the model was loaded. Hence the Accuracy and Loss graph for the same has only two epochs. The ResNet 20 MNIST model was run after the runtime got stopped in Colab leading to the error in the notebook.

Appendix

- ## Screenshots:

1. Training Runtime:

1.1. VGG16-MNIST:

```
Epoch 8/20
843/843 [==============================] - 106s 126ms/step - loss: 0.2938 - accuracy: 0.9712 - val_loss: 0.1272 - val_accuracy:
0.9863
Epoch 9/20
843/843 [==============================] - 105s 125ms/step - loss: 0.5705 - accuracy: 0.9732 - val_loss: 0.1542 - val_accuracy:
0.9878
Epoch 10/20
843/843 [==============================] - 106s 125ms/step - loss: 0.3007 - accuracy: 0.9728 - val_loss: 0.2071 - val_accuracy:
0.9895
Epoch 11/20
843/843 [==============================] - 105s 125ms/step - loss: 0.2477 - accuracy: 0.9723 - val_loss: 0.1485 - val_accuracy:
0.9822
Epoch 12/20
843/843 [==============================] - 105s 125ms/step - loss: 0.2432 - accuracy: 0.9713 - val_loss: 68.5864 - val_accurac
y: 0.9865
Epoch 13/20
843/843 [==============================] - 106s 125ms/step - loss: 0.2763 - accuracy: 0.9695 - val_loss: 0.0746 - val_accuracy:
0.9877

Epoch 00013: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
Epoch 14/20
843/843 [==============================] - 105s 125ms/step - loss: 0.2444 - accuracy: 0.9775 - val_loss: 0.1077 - val_accuracy:
0.9905
Epoch 15/20
843/843 [==============================] - 106s 125ms/step - loss: 0.1532 - accuracy: 0.9803 - val_loss: 0.2922 - val_accuracy:
0.9910
Epoch 16/20
843/843 [==============================] - 106s 125ms/step - loss: 0.1756 - accuracy: 0.9802 - val_loss: 56.3862 - val_accurac
y: 0.9892
Epoch 17/20
843/843 [==============================] - 105s 125ms/step - loss: 0.1596 - accuracy: 0.9790 - val_loss: 0.1561 - val_accuracy:
0.9903
Epoch 18/20
843/843 [==============================] - 105s 125ms/step - loss: 0.3301 - accuracy: 0.9799 - val_loss: 0.1733 - val_accuracy:
0.9908

Epoch 00018: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.
Epoch 19/20
843/843 [==============================] - 105s 125ms/step - loss: 0.3489 - accuracy: 0.9833 - val_loss: 0.2227 - val_accuracy:
0.9918
Epoch 20/20
843/843 [==============================] - 105s 125ms/step - loss: 0.1608 - accuracy: 0.9838 - val_loss: 0.1787 - val_accuracy:
0.9898
```

1.2. VGG16-CIFAR

```
 - 55s - loss: 0.8354 - accuracy: 0.8758 - val_loss: 0.7970 - val_accuracy: 0.8854
Epoch 91/100
 - 55s - loss: 0.8316 - accuracy: 0.8765 - val_loss: 0.7503 - val_accuracy: 0.8966
Epoch 92/100
 - 55s - loss: 0.8261 - accuracy: 0.8751 - val_loss: 0.7535 - val_accuracy: 0.9006
Epoch 93/100
 - 55s - loss: 0.8506 - accuracy: 0.8692 - val_loss: 0.8039 - val_accuracy: 0.8790
Epoch 94/100
 - 55s - loss: 0.8377 - accuracy: 0.8744 - val_loss: 0.7933 - val_accuracy: 0.8842
Epoch 95/100
 - 55s - loss: 0.8261 - accuracy: 0.8772 - val_loss: 0.7851 - val_accuracy: 0.8830
Epoch 96/100
 - 55s - loss: 0.8505 - accuracy: 0.8719 - val_loss: 0.7681 - val_accuracy: 0.8950
Epoch 97/100
 - 55s - loss: 0.8327 - accuracy: 0.8764 - val_loss: 0.7762 - val_accuracy: 0.8946
Epoch 98/100
 - 55s - loss: 0.8295 - accuracy: 0.8785 - val_loss: 0.8063 - val_accuracy: 0.8872
Epoch 99/100
 - 55s - loss: 0.8307 - accuracy: 0.8761 - val_loss: 0.8459 - val_accuracy: 0.8706
Epoch 100/100
 - 55s - loss: 0.8309 - accuracy: 0.8757 - val_loss: 0.8230 - val_accuracy: 0.8772
```

1.3. ResNet20 MNIST

```
y: 0.9917
Epoch 14/20
60000/60000 [==============================] - 139s 2ms/step - loss: 0.0087 - accuracy: 0.9972 - val_loss: 0.0297 - val_accurac
y: 0.9920
Epoch 15/20
60000/60000 [==============================] - 139s 2ms/step - loss: 0.0072 - accuracy: 0.9977 - val_loss: 0.0280 - val_accurac
y: 0.9928
Epoch 16/20
60000/60000 [==============================] - 141s 2ms/step - loss: 0.0073 - accuracy: 0.9978 - val_loss: 0.0349 - val_accurac
y: 0.9899
Epoch 17/20
60000/60000 [==============================] - 140s 2ms/step - loss: 0.0064 - accuracy: 0.9978 - val_loss: 0.0319 - val_accurac
y: 0.9925
Epoch 18/20
60000/60000 [==============================] - 139s 2ms/step - loss: 0.0068 - accuracy: 0.9979 - val_loss: 0.0279 - val_accurac
y: 0.9939
Epoch 19/20
60000/60000 [==============================] - 138s 2ms/step - loss: 0.0048 - accuracy: 0.9985 - val_loss: 0.0348 - val_accurac
y: 0.9921
Epoch 20/20
60000/60000 [==============================] - 138s 2ms/step - loss: 0.0062 - accuracy: 0.9980 - val_loss: 0.0303 - val_accurac
y: 0.9909
```

1.4. ResNet20 CIFAR v1

```
y: 0.9150
Epoch 142/200
Learning rate:  1e-05
704/704 [==============================] - 33s 46ms/step - loss: 0.1733 - accuracy: 0.9878 - val_loss: 0.4703 - val_accurac
y: 0.9140
Epoch 143/200
Learning rate:  1e-05
704/704 [==============================] - 32s 46ms/step - loss: 0.1746 - accuracy: 0.9876 - val_loss: 0.4699 - val_accurac
y: 0.9152
Epoch 144/200
Learning rate:  1e-05
704/704 [==============================] - 32s 46ms/step - loss: 0.1730 - accuracy: 0.9879 - val_loss: 0.4716 - val_accurac
y: 0.9152
Epoch 145/200
Learning rate:  1e-05
704/704 [==============================] - 32s 46ms/step - loss: 0.1731 - accuracy: 0.9883 - val_loss: 0.4740 - val_accurac
y: 0.9132
Epoch 146/200
Learning rate:  1e-05
623/704 [=========================>....] - ETA: 3s - loss: 0.1725 - accuracy: 0.9888Buffered data was truncated after reachi
ng the output size limit.
```

1.5. ResNet20 CIFAR v2

```
Epoch 80/200
Learning rate:  0.001
704/704 [==============================] - 62s 89ms/step - loss: 0.4640 - accuracy: 0.9120 - val_loss: 0.8258 - val_accurac
y: 0.8172
Epoch 81/200
Learning rate:  0.001
704/704 [==============================] - 62s 89ms/step - loss: 0.4619 - accuracy: 0.9129 - val_loss: 0.6800 - val_accurac
y: 0.8500
Epoch 82/200
Learning rate:  0.0001
704/704 [==============================] - 65s 93ms/step - loss: 0.3842 - accuracy: 0.9418 - val_loss: 0.4816 - val_accurac
y: 0.9068
Epoch 83/200
Learning rate:  0.0001
704/704 [==============================] - 63s 89ms/step - loss: 0.3518 - accuracy: 0.9510 - val_loss: 0.4661 - val_accurac
y: 0.9110
Epoch 84/200
Learning rate:  0.0001
220/704 [========>.....................] - ETA: 41s - loss: 0.3380 - accuracy: 0.9555Buffered data was truncated after reach
ing the output size limit.
```

1.6. ResNet56 CIFAR v1

```
y: 0.8908
Epoch 46/50
Learning rate:  0.001
352/352 [==============================] - 66s 188ms/step - loss: 0.3295 - accuracy: 0.9561 - val_loss: 0.5925 - val_accurac
y: 0.8854
Epoch 47/50
Learning rate:  0.001
352/352 [==============================] - 66s 189ms/step - loss: 0.3407 - accuracy: 0.9532 - val_loss: 0.6608 - val_accurac
y: 0.8656
Epoch 48/50
Learning rate:  0.001
352/352 [==============================] - 66s 188ms/step - loss: 0.3391 - accuracy: 0.9526 - val_loss: 0.7672 - val_accurac
y: 0.8386
Epoch 49/50
Learning rate:  0.001
352/352 [==============================] - 66s 188ms/step - loss: 0.3354 - accuracy: 0.9551 - val_loss: 0.6014 - val_accurac
y: 0.8840
Epoch 50/50
Learning rate:  0.001
352/352 [==============================] - 67s 191ms/step - loss: 0.3355 - accuracy: 0.9541 - val_loss: 0.7231 - val_accurac
y: 0.8590
```

1.7. ResNET56 CIFAR v2

```
Epoch 41/50
Learning rate:  0.001
352/352 [==============================] - 154s 437ms/step - loss: 0.3190 - accuracy: 0.9553 - val_loss: 0.6052 - val_accuracy: 0.8870
Epoch 42/50
Learning rate:  0.001
352/352 [==============================] - 154s 437ms/step - loss: 0.3159 - accuracy: 0.9564 - val_loss: 0.6504 - val_accuracy: 0.8684
Epoch 43/50
Learning rate:  0.001
352/352 [==============================] - 154s 437ms/step - loss: 0.3195 - accuracy: 0.9562 - val_loss: 0.6193 - val_accuracy: 0.8854
Epoch 44/50
Learning rate:  0.001
352/352 [==============================] - 154s 437ms/step - loss: 0.3145 - accuracy: 0.9568 - val_loss: 0.6516 - val_accuracy: 0.8716
Epoch 45/50
Learning rate:  0.001
352/352 [==============================] - 154s 437ms/step - loss: 0.3195 - accuracy: 0.9558 - val_loss: 0.8467 - val_accuracy: 0.8456
Epoch 46/50
Learning rate:  0.001
352/352 [==============================] - 154s 437ms/step - loss: 0.3175 - accuracy: 0.9560 - val_loss: 0.5903 - val_accuracy: 0.8852
Epoch 47/50
Learning rate:  0.001
352/352 [==============================] - 154s 438ms/step - loss: 0.3172 - accuracy: 0.9559 - val_loss: 0.7924 - val_accuracy: 0.8522
Epoch 48/50
Learning rate:  0.001
352/352 [==============================] - 154s 438ms/step - loss: 0.3155 - accuracy: 0.9564 - val_loss: 0.7017 - val_accuracy: 0.8684
Epoch 49/50
Learning rate:  0.001
352/352 [==============================] - 154s 437ms/step - loss: 0.3193 - accuracy: 0.9554 - val_loss: 0.5977 - val_accuracy: 0.8812
Epoch 50/50
Learning rate:  0.001
352/352 [==============================] - 154s 438ms/step - loss: 0.3126 - accuracy: 0.9567 - val_loss: 0.6122 - val_accuracy: 0.8822
```

2. Testing Runtime:

2.1. VGG16-MNIST:

```
final_loss, final_acc = model.evaluate(X_val, Y_val, verbose=0)
print("Final loss: {0:.6f}, final accuracy: {1:.6f}".format(final_loss, final_acc))
```

```
Final loss: 0.178657, final accuracy: 0.989833
```

2.2. VGG16-CIFAR

```
final_loss, final_acc = model.evaluate(X_test, Y_test, verbose=0)
print("Final loss: {0:.6f}, final accuracy: {1:.6f}".format(final_loss, final_acc))
```

```
Final loss: 0.859706, final accuracy: 0.864900
```

2.3. ResNet20 CIFAR v1

```
predictions = model.predict(X_test, verbose=1)
```

```
10000/10000 [==============================] - 2s 196us/step
```

```
def test_accuracy():
    tp = []
    err = []
    t = 0
    #predictions = model.predict(x_test, verbose=1)
    for i in range(predictions.shape[0]):
        if (np.argmax(predictions[i]) == np.argmax(Y_test[i])):
            t = t+1
            tp.append(i)
        else:
            err.append(i)
    return t, float(t)*100/predictions.shape[0], err, tp
```

```
p = test_accuracy()
print("Test accuracy: {} %".format(p[1]))
```

```
Test accuracy: 90.56 %
```

2.4. ResNet20 CIFAR v2

```
predictions = model.predict(X_test, verbose=1)
```

```
10000/10000 [==============================] - 5s 483us/step
```

```
def test_accuracy():
    tp = []
    err = []
    t = 0
    #predictions = model.predict(x_test, verbose=1)
    for i in range(predictions.shape[0]):
        if (np.argmax(predictions[i]) == np.argmax(Y_test[i])):
            t = t+1
            tp.append(i)
        else:
            err.append(i)
    return t, float(t)*100/predictions.shape[0], err, tp
```

```
p = test_accuracy()
print("Test accuracy: {} %".format(p[1]))
```

```
Test accuracy: 90.98 %
```

2.5. *ResNet56 CIFAR v1*

```
10000/10000 [==============================] - 7s 666us/step
```

```
def test_accuracy():
    tp = []
    err = []
    t = 0
    #predictions = model.predict(x_test, verbose=1)
    for i in range(predictions.shape[0]):
        if (np.argmax(predictions[i]) == np.argmax(Y_test[i])):
            t = t+1
            tp.append(i)
        else:
            err.append(i)
    return t, float(t)*100/predictions.shape[0], err, tp
```

```
p = test_accuracy()
print("Test accuracy: {} %".format(p[1]))
```

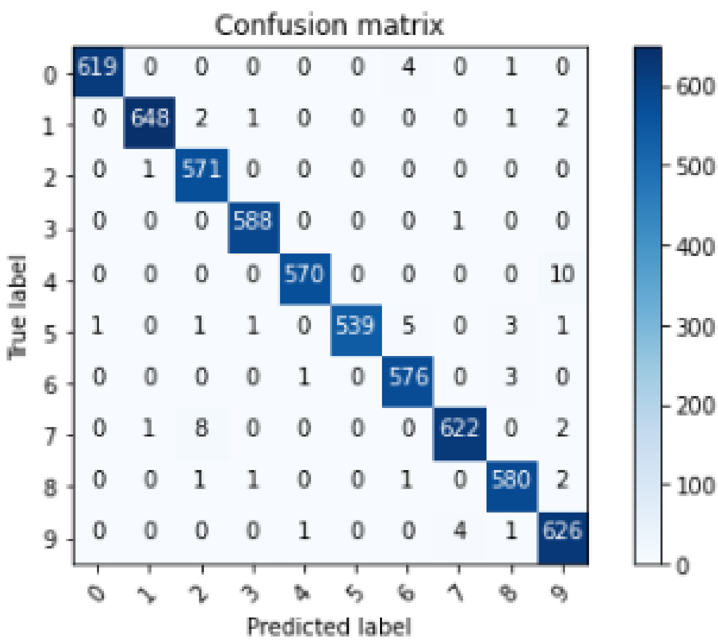
```
Test accuracy: 85.37 %
```

2.6. *ResNET56 CIFAR v2*

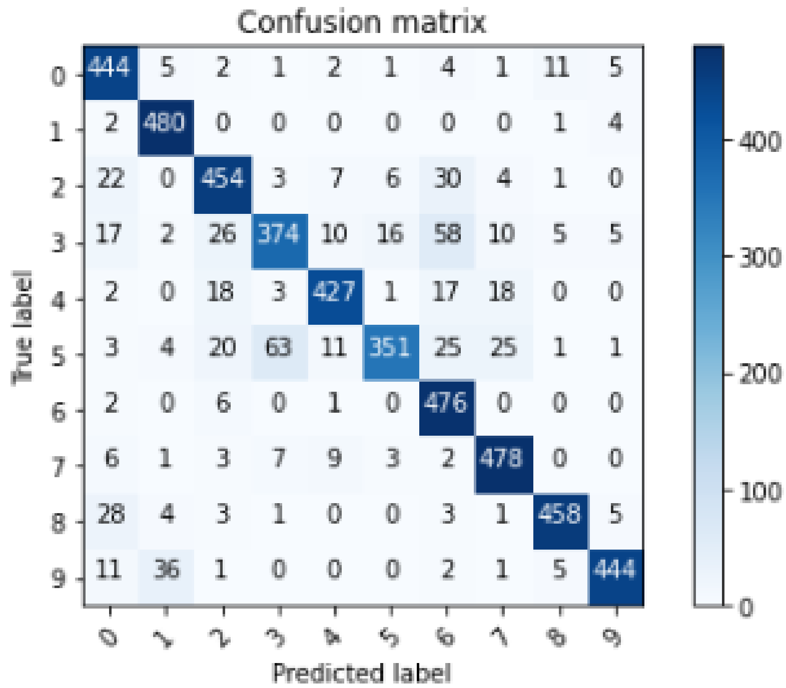
```
10000/10000 [==============================] - 12s 1ms/step
```

3. Confusion Matrix:

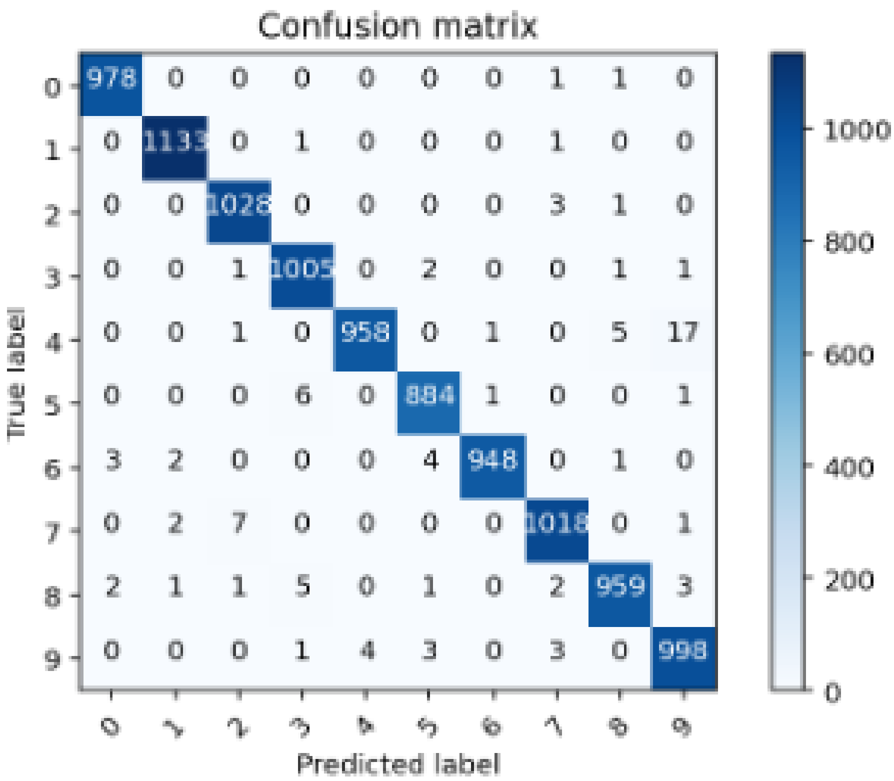
3.1. *VGG16-MNIST:*



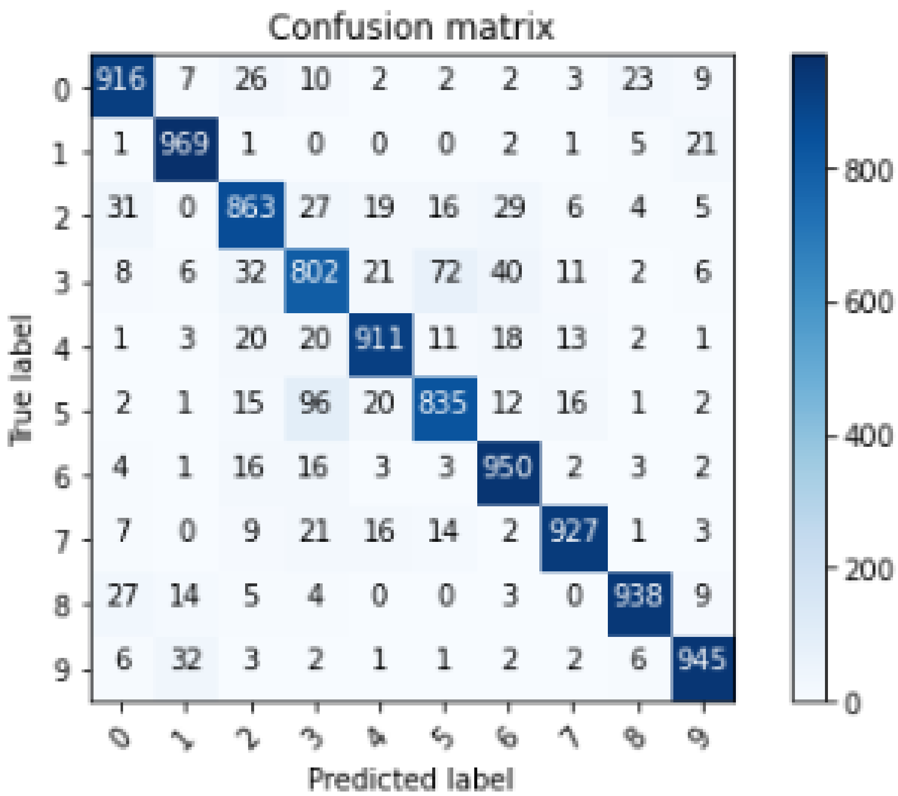
3.2. VGG16-CIFAR



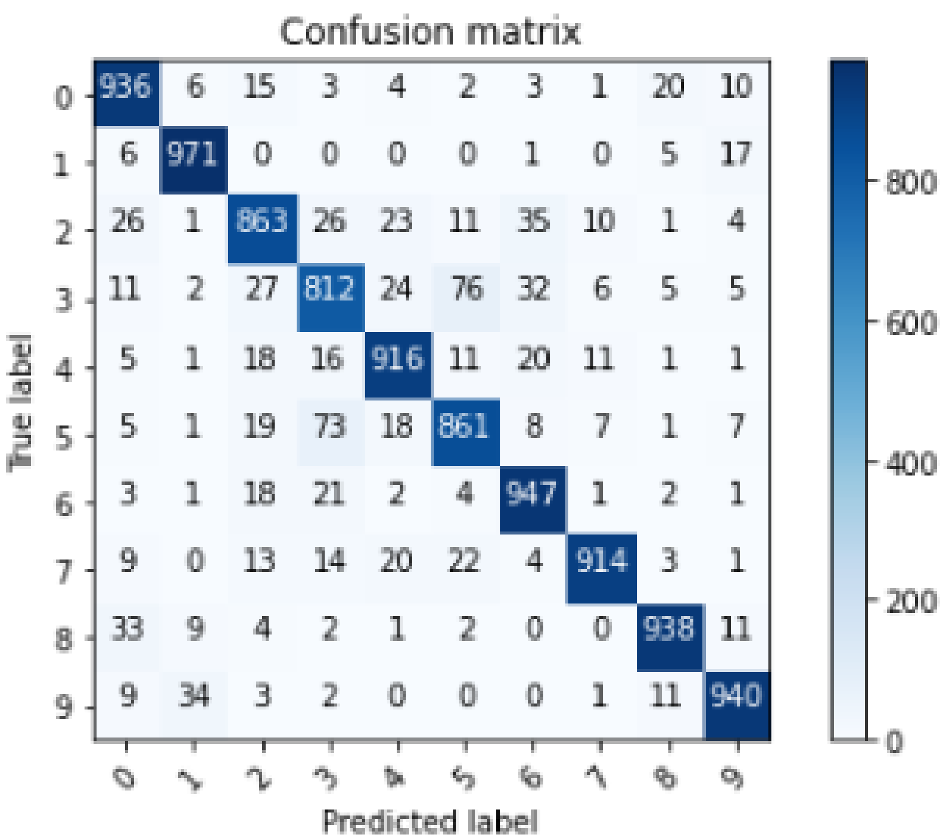
3.3. ResNet20 MNIST



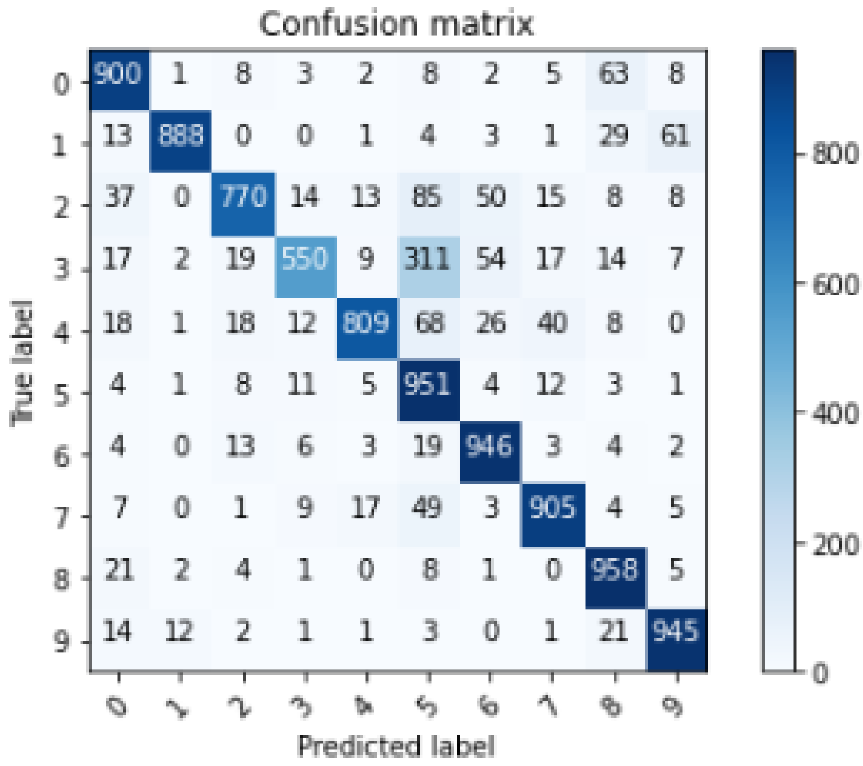
3.4. ResNet20 CIFAR v1



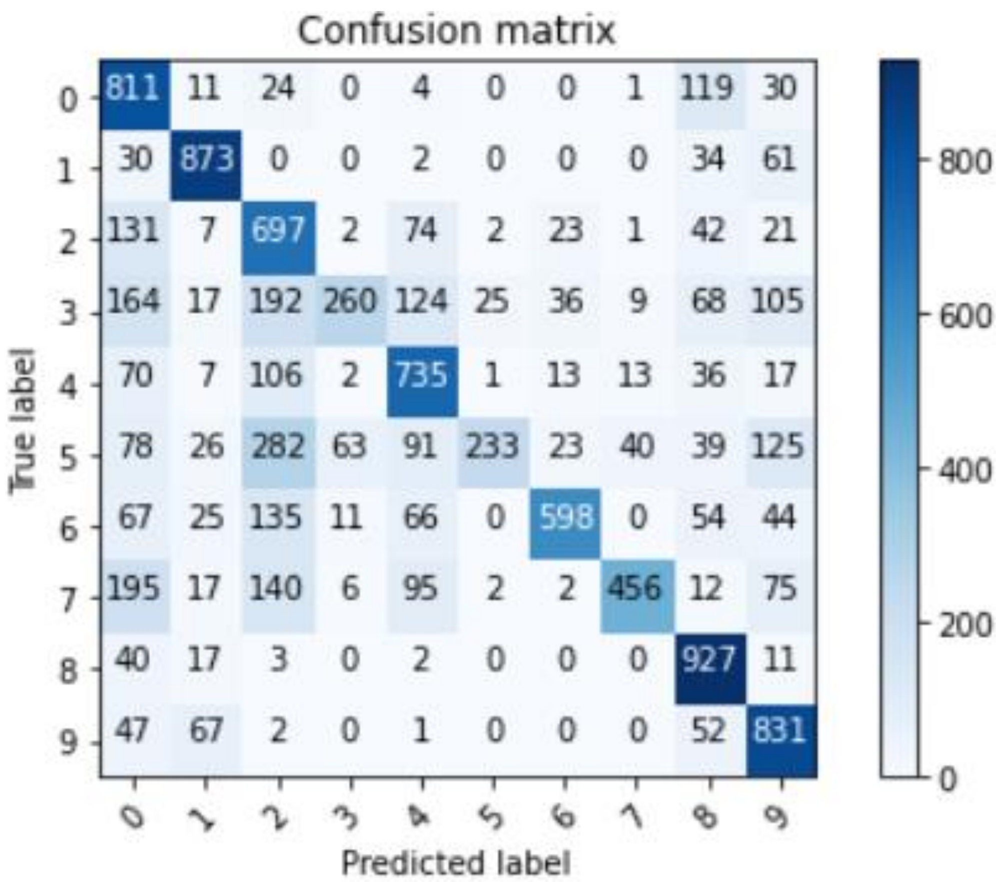
3.5. ResNet20 CIFAR v2



3.6. ResNet56 CIFAR v1

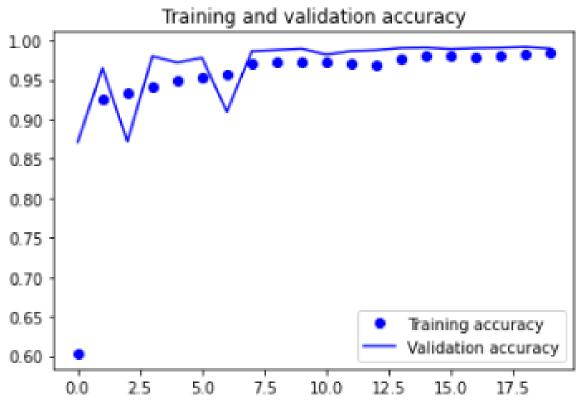


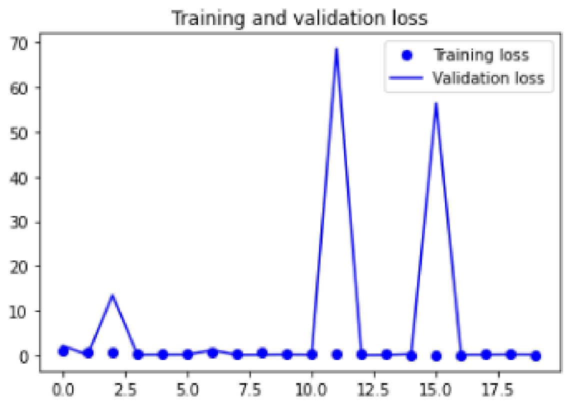
3.7. ResNET56 CIFAR v2



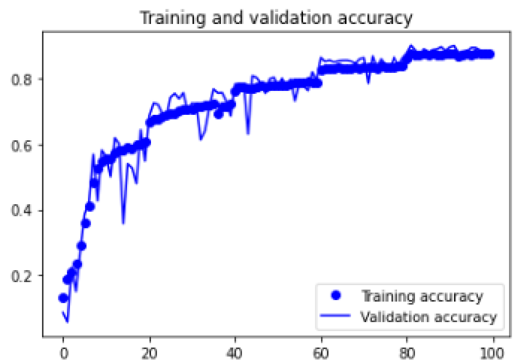
4. Accuracy and Loss Graphs

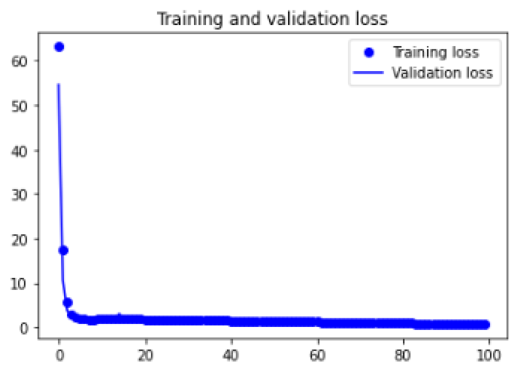
4.1. VGG16-MNIST:



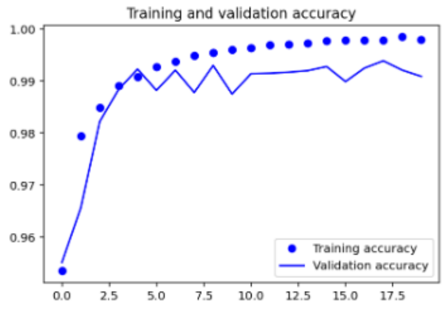


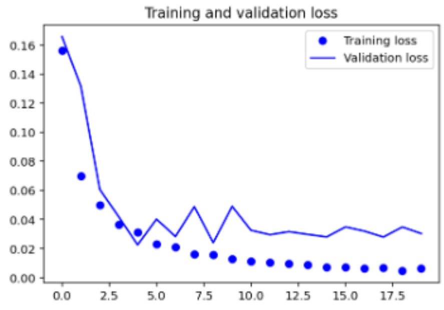
4.2. VGG16-CIFAR



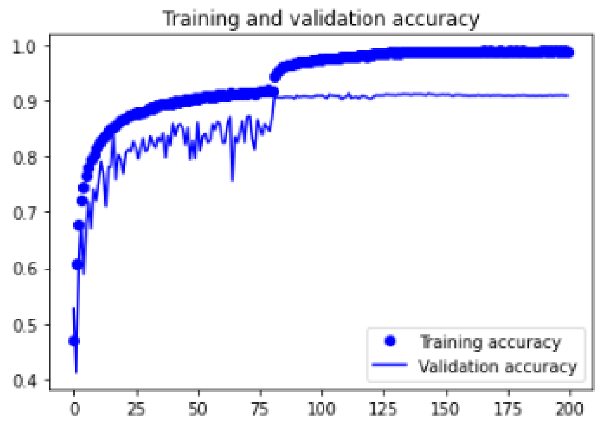


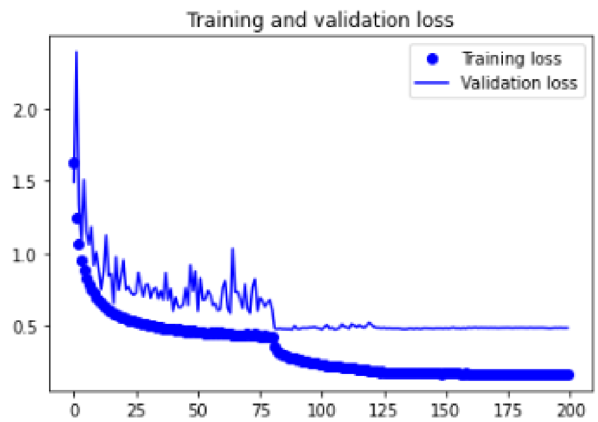
4.3. ResNet20 MNIST



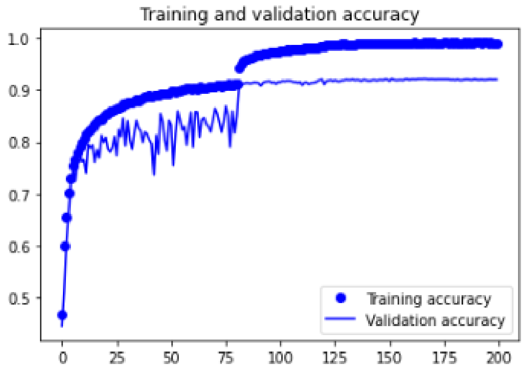


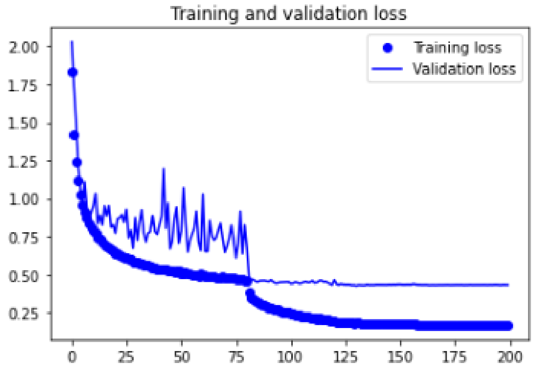
4.4. ResNet20 CIFAR v1



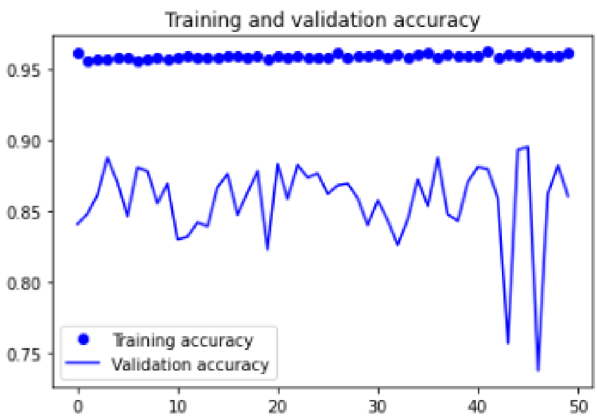


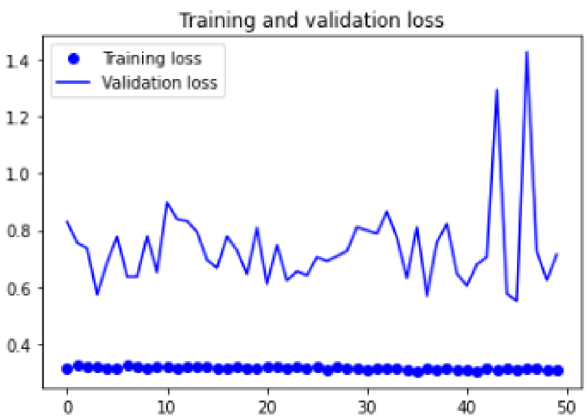
4.5. ResNet20 CIFAR v2



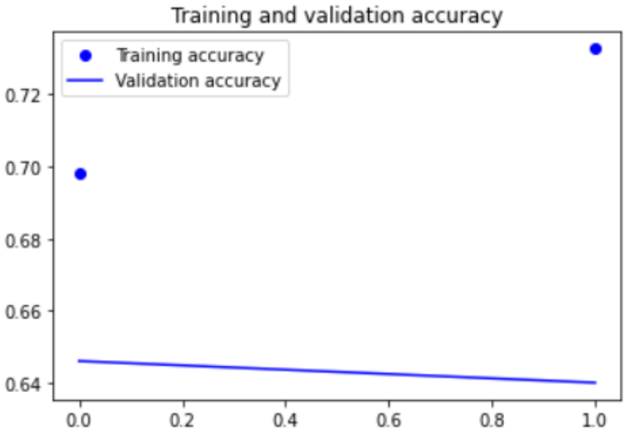


4.6. ResNet56 CIFAR v1



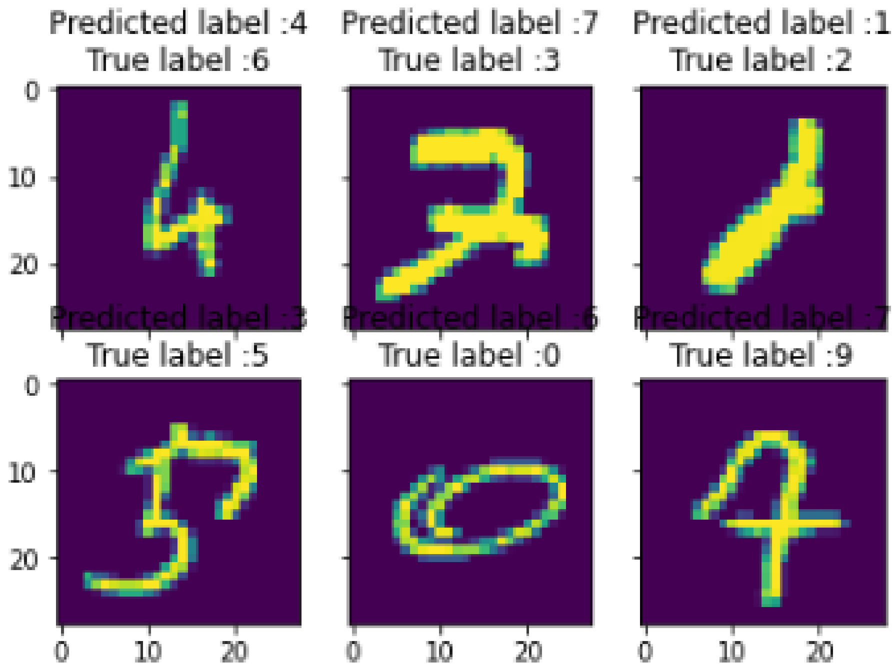


4.7. ResNET56 CIFAR v2

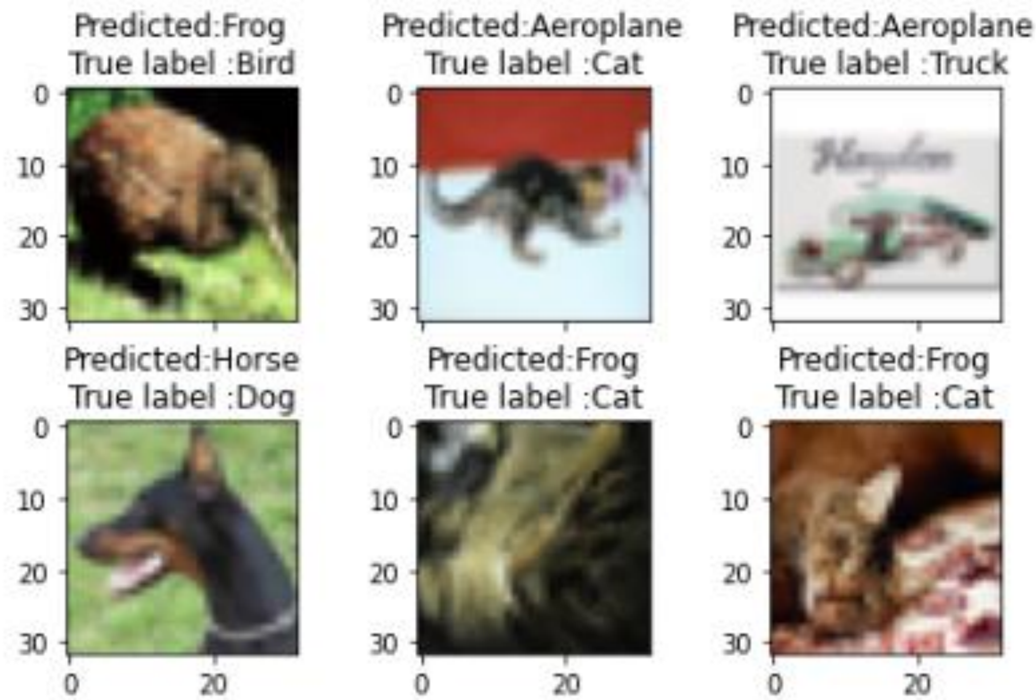


5. Testing Errors:

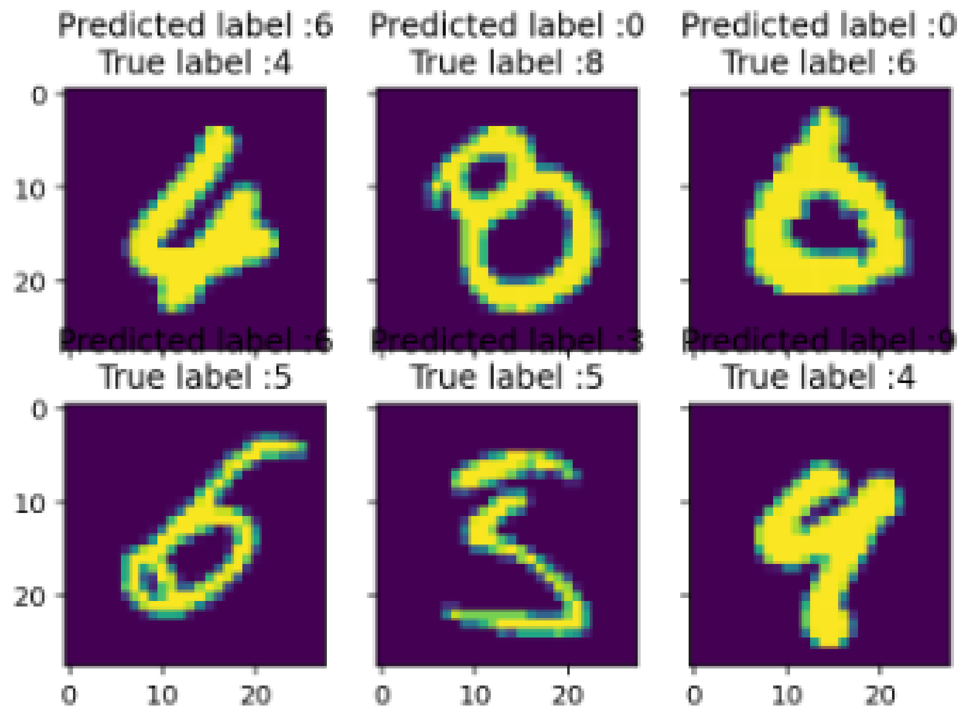
5.1. VGG16-MNIST:



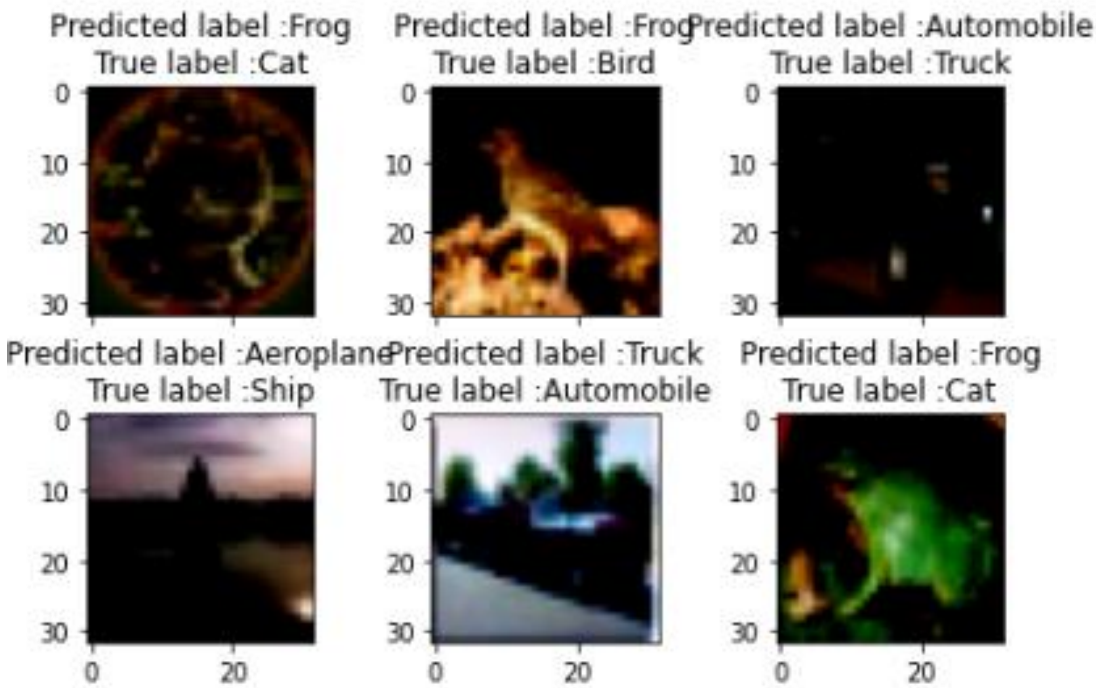
5.2. VGG16-CIFAR



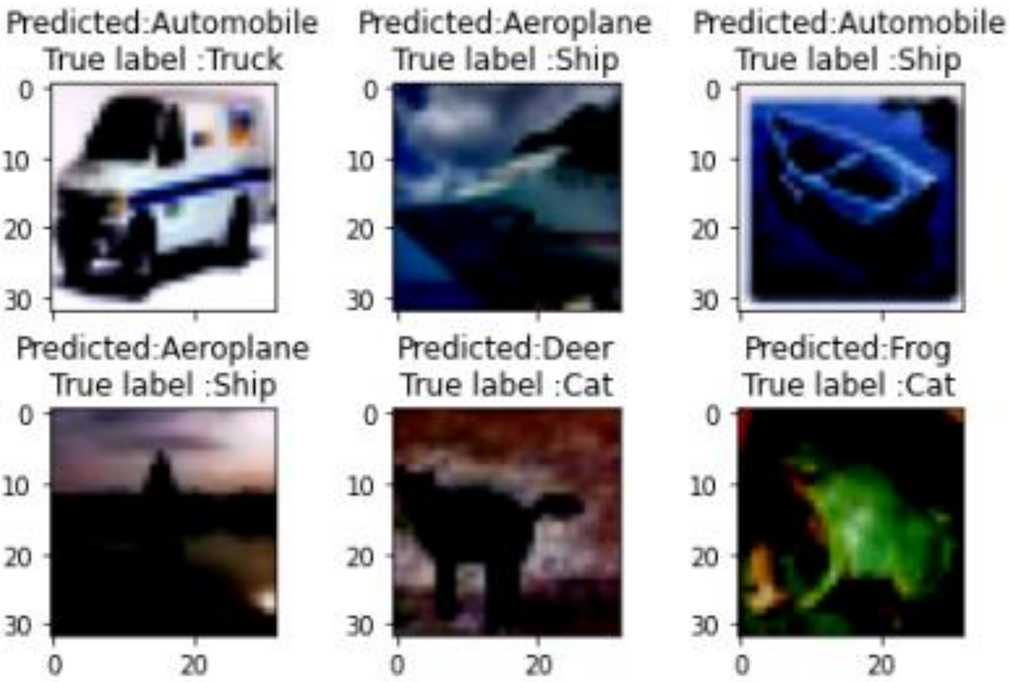
5.3. ResNet20 MNIST



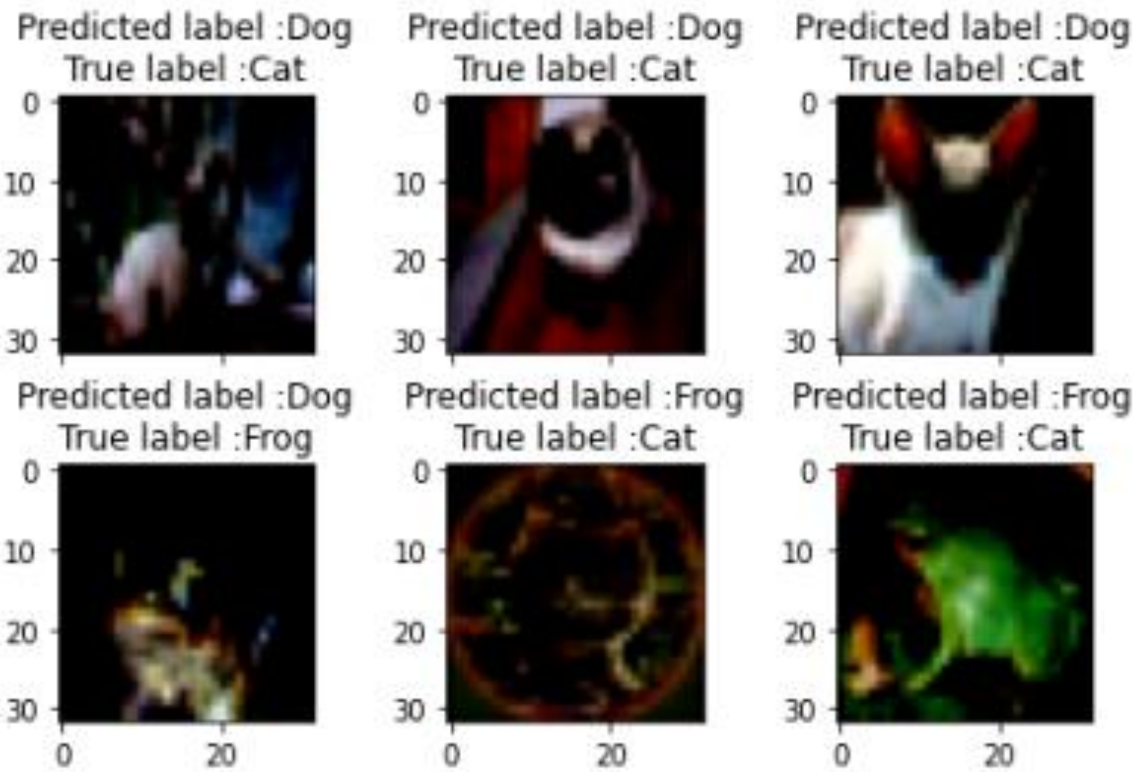
5.4. ResNet20 CIFAR v1



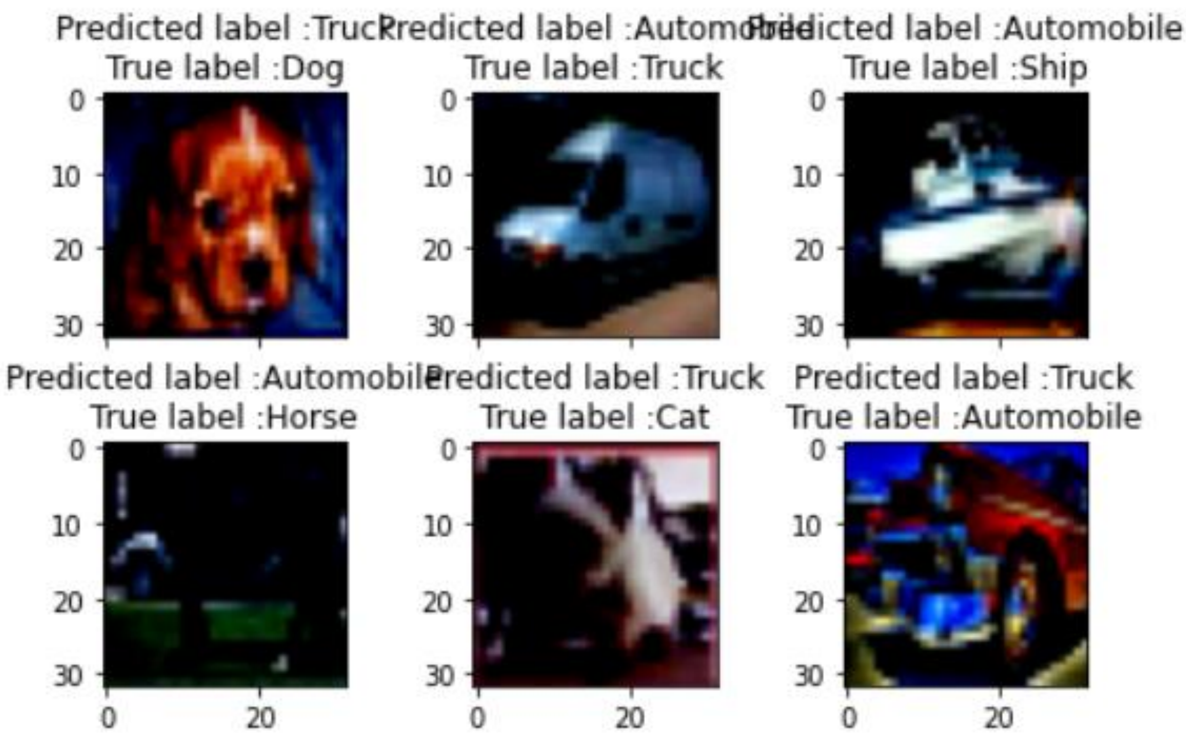
5.5. ResNet20 CIFAR v2



5.6. ResNet56 CIFAR v1



5.7. ResNET56 CIFAR v2



6. Precision and Recall:

6.1. VGG16-MNIST:

	precision	recall	f1-score	support
Class 0	1.00	0.99	1.00	624
Class 1	1.00	0.99	0.99	654
Class 2	0.98	1.00	0.99	572
Class 3	0.99	1.00	1.00	589
Class 4	1.00	0.98	0.99	580
Class 5	1.00	0.98	0.99	551
Class 6	0.98	0.99	0.99	580
Class 7	0.99	0.98	0.99	633
Class 8	0.98	0.99	0.99	585
Class 9	0.97	0.99	0.98	632
accuracy			0.99	6000
macro avg	0.99	0.99	0.99	6000
weighted avg	0.99	0.99	0.99	6000

6.2. VGG16-CIFAR

	precision	recall	f1-score	support
Class 0	0.83	0.93	0.88	476
Class 1	0.90	0.99	0.94	487
Class 2	0.85	0.86	0.86	527
Class 3	0.83	0.72	0.77	523
Class 4	0.91	0.88	0.90	486
Class 5	0.93	0.70	0.80	504
Class 6	0.77	0.98	0.86	485
Class 7	0.89	0.94	0.91	509
Class 8	0.95	0.91	0.93	503
Class 9	0.96	0.89	0.92	500
accuracy			0.88	5000
macro avg	0.88	0.88	0.88	5000
weighted avg	0.88	0.88	0.88	5000

6.3. ResNet20 MNIST

	precision	recall	f1-score	support
Class 0	0.99	1.00	1.00	980
Class 1	1.00	1.00	1.00	1135
Class 2	0.99	1.00	0.99	1032
Class 3	0.99	1.00	0.99	1010
Class 4	1.00	0.98	0.99	982
Class 5	0.99	0.99	0.99	892
Class 6	1.00	0.99	0.99	958
Class 7	0.99	0.99	0.99	1028
Class 8	0.99	0.98	0.99	974
Class 9	0.98	0.99	0.98	1009
accuracy			0.99	10000
macro avg	0.99	0.99	0.99	10000
weighted avg	0.99	0.99	0.99	10000

6.4. ResNet20 CIFAR v1

	precision	recall	f1-score	support
Class 0	0.91	0.92	0.91	1000
Class 1	0.94	0.97	0.95	1000
Class 2	0.87	0.86	0.87	1000
Class 3	0.80	0.80	0.80	1000
Class 4	0.92	0.91	0.91	1000
Class 5	0.88	0.83	0.85	1000
Class 6	0.90	0.95	0.92	1000
Class 7	0.94	0.93	0.94	1000
Class 8	0.95	0.94	0.95	1000
Class 9	0.94	0.94	0.94	1000
accuracy			0.91	10000
macro avg	0.91	0.91	0.91	10000
weighted avg	0.91	0.91	0.91	10000

6.5. ResNet20 CIFAR v2

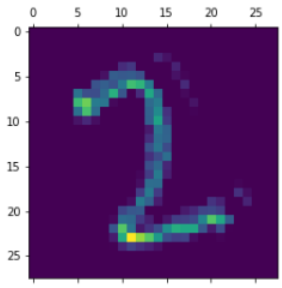
	precision	recall	f1-score	support
Class 0	0.90	0.94	0.92	1000
Class 1	0.95	0.97	0.96	1000
Class 2	0.88	0.86	0.87	1000
Class 3	0.84	0.81	0.82	1000
Class 4	0.91	0.92	0.91	1000
Class 5	0.87	0.86	0.87	1000
Class 6	0.90	0.95	0.92	1000
Class 7	0.96	0.91	0.94	1000
Class 8	0.95	0.94	0.94	1000
Class 9	0.94	0.94	0.94	1000
accuracy			0.91	10000
macro avg	0.91	0.91	0.91	10000
weighted avg	0.91	0.91	0.91	10000

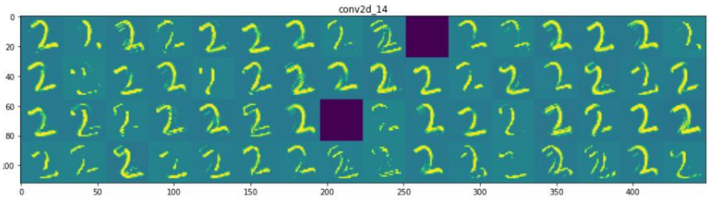
6.6. ResNet56 CIFAR v1

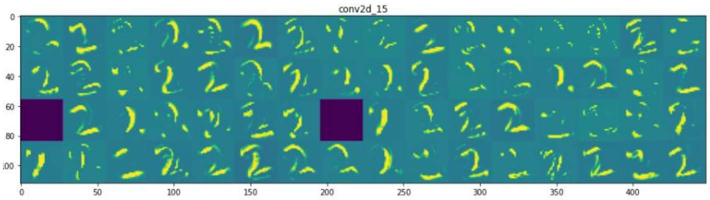
	precision	recall	f1-score	support
Class 0	0.87	0.90	0.88	1000
Class 1	0.98	0.89	0.93	1000
Class 2	0.91	0.77	0.84	1000
Class 3	0.91	0.55	0.68	1000
Class 4	0.94	0.81	0.87	1000
Class 5	0.63	0.95	0.76	1000
Class 6	0.87	0.95	0.91	1000
Class 7	0.91	0.91	0.91	1000
Class 8	0.86	0.96	0.91	1000
Class 9	0.91	0.94	0.93	1000
accuracy			0.86	10000
macro avg	0.88	0.86	0.86	10000
weighted avg	0.88	0.86	0.86	10000

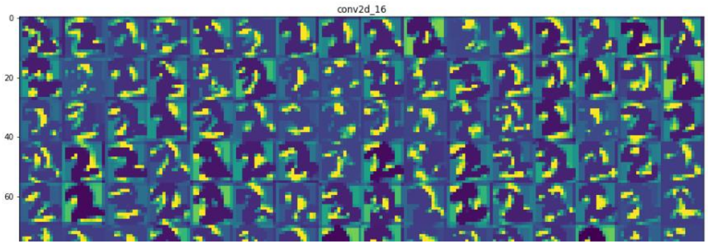
7. Layer outputs:

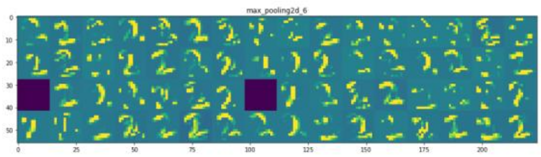
7.1. VGG16-MNIST:

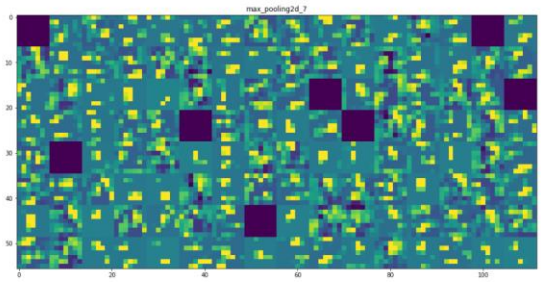


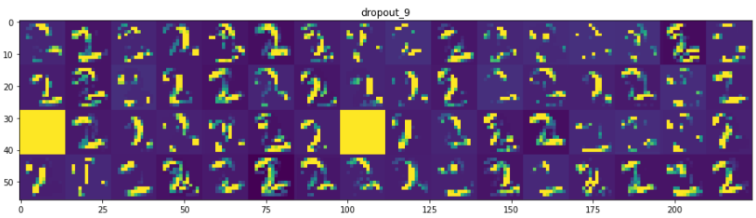




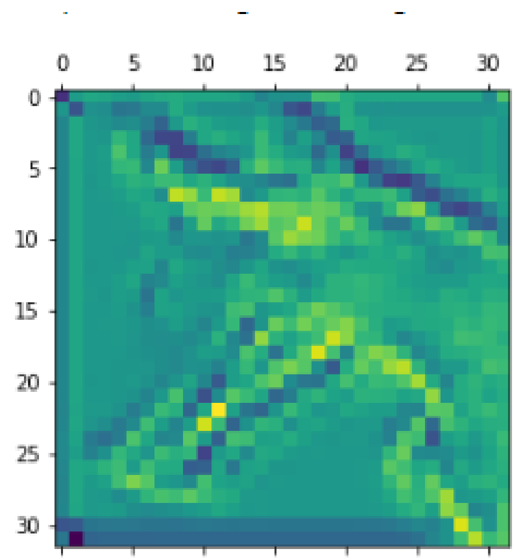


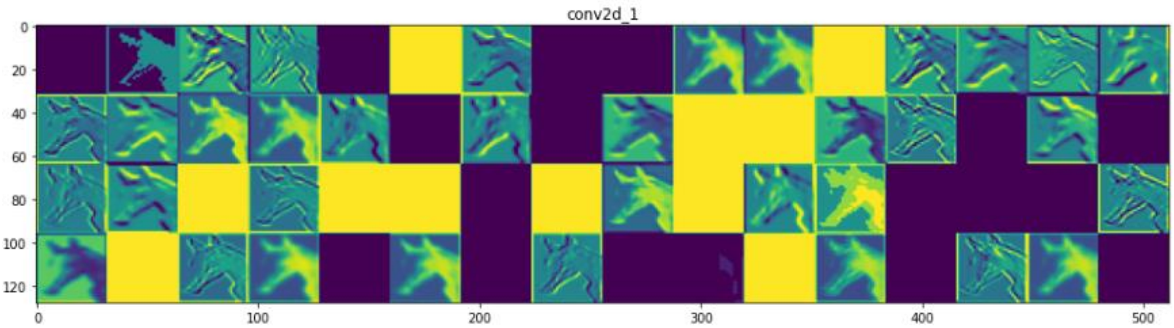


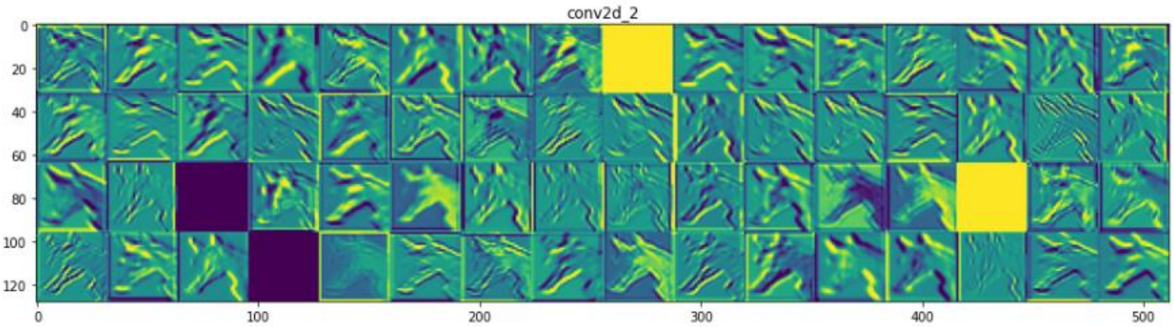


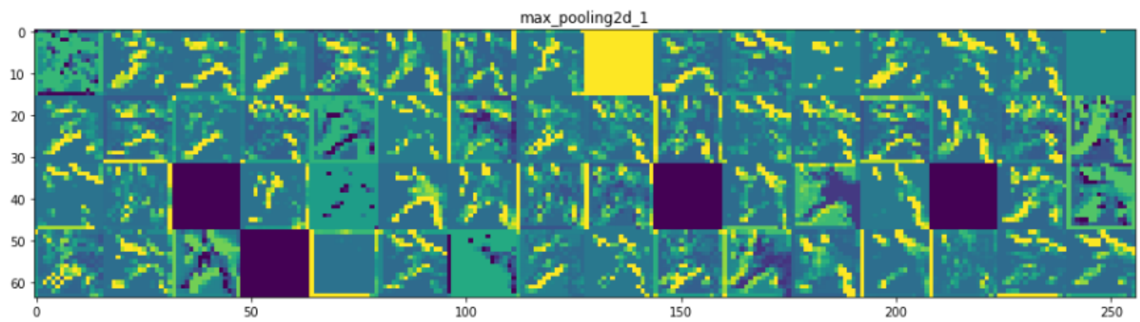


7.2. VGG16-CIFAR

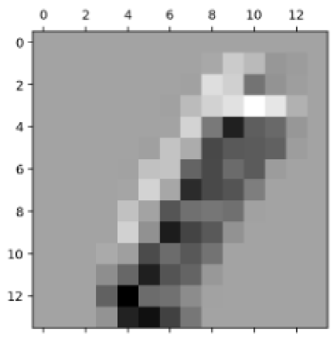


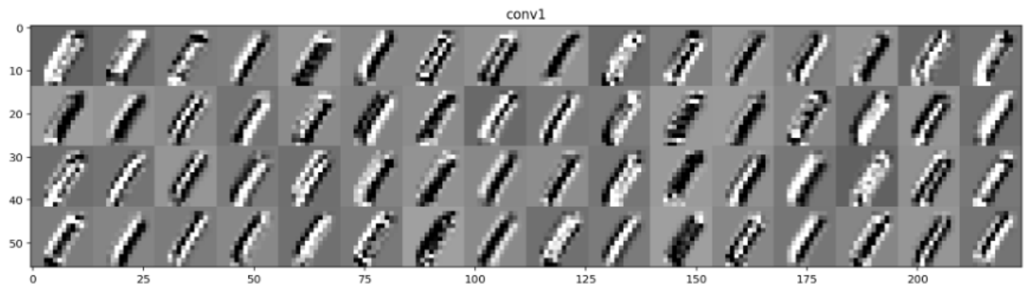


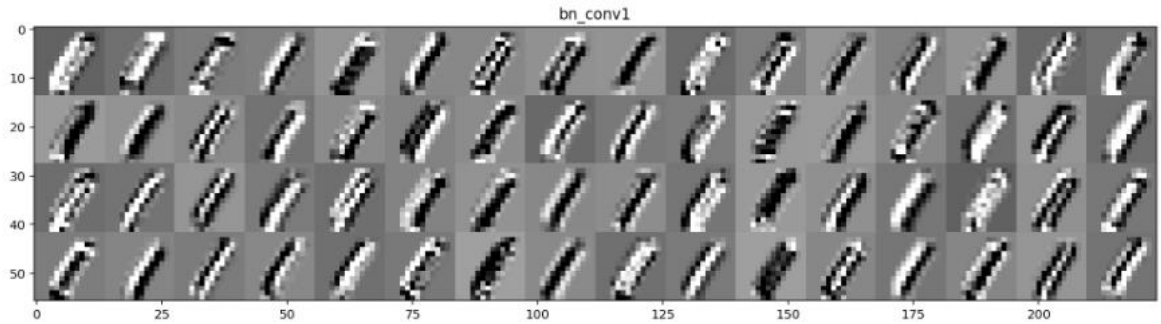


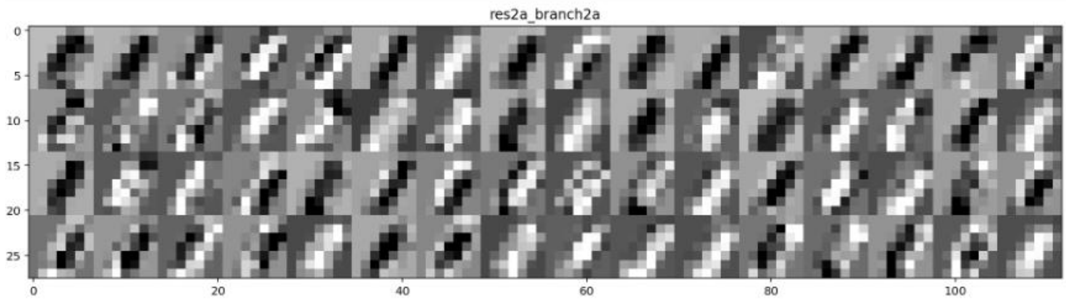


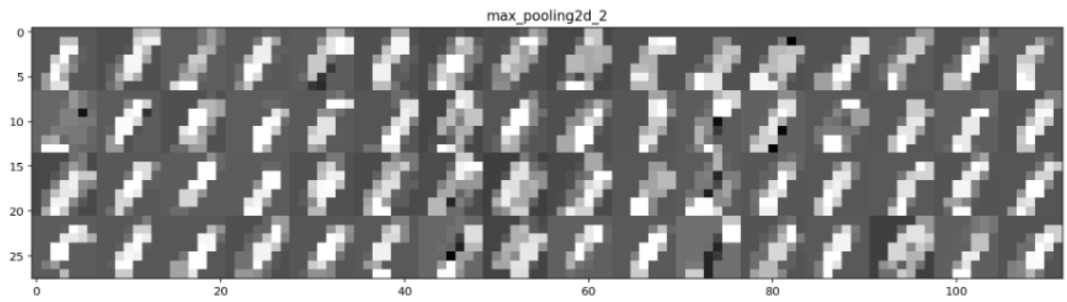
7.3. ResNet20 MNIST



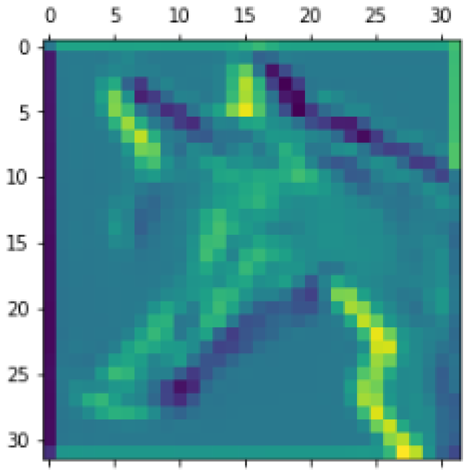


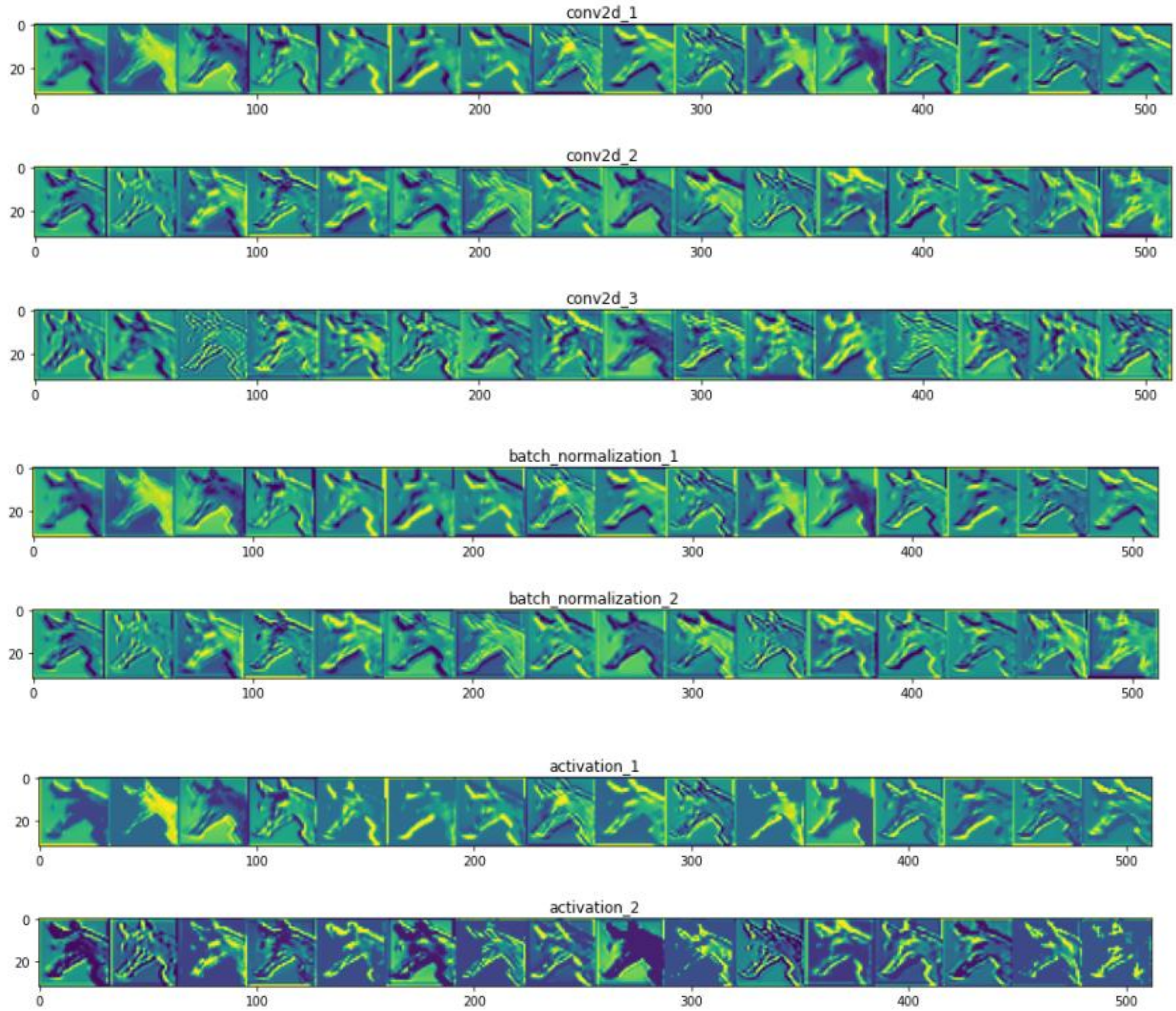




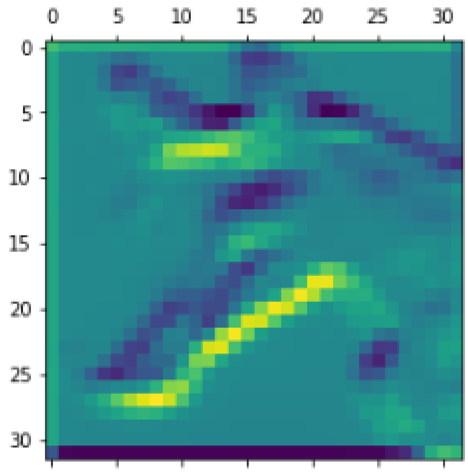


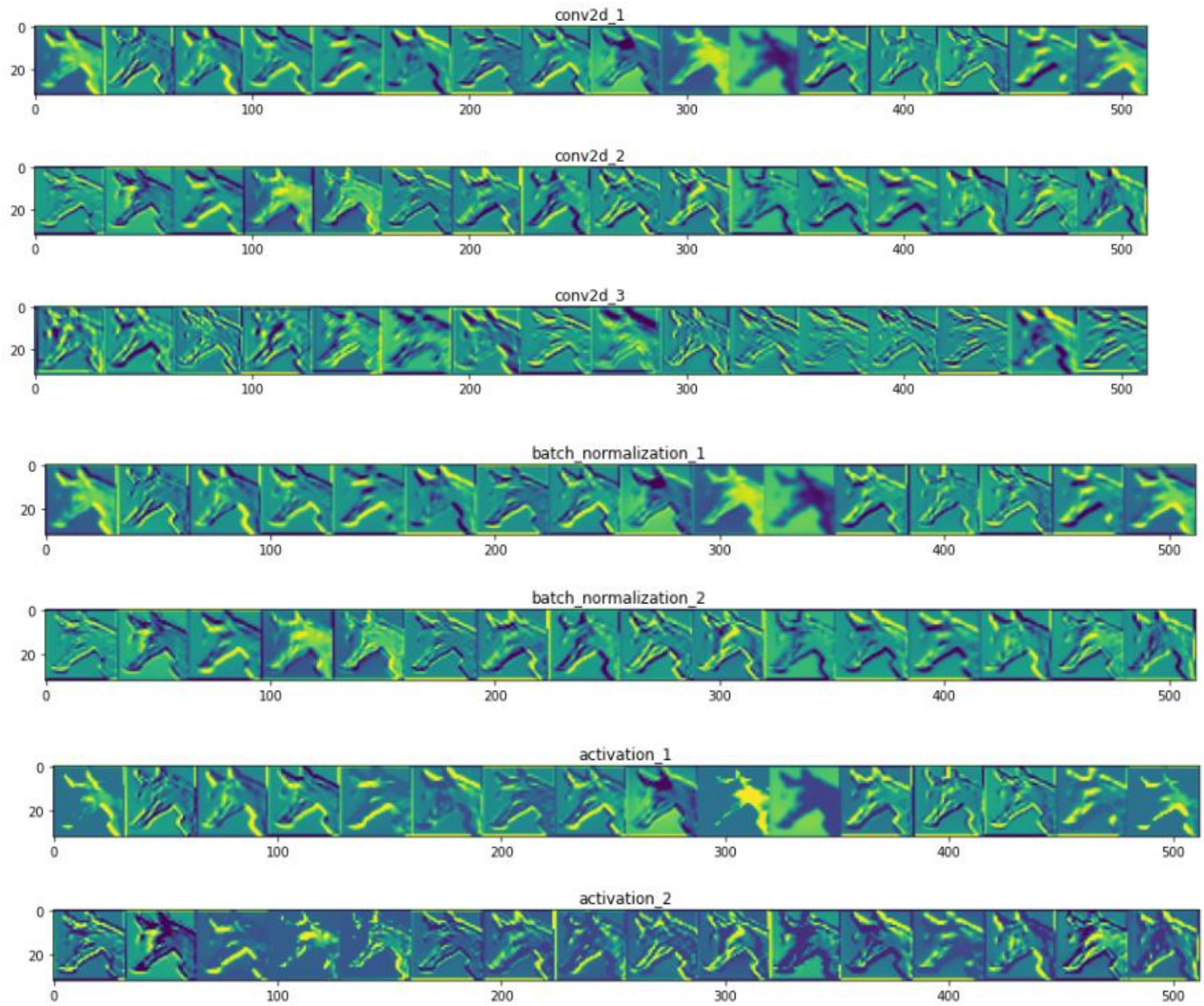
7.4. ResNet20 CIFAR v1



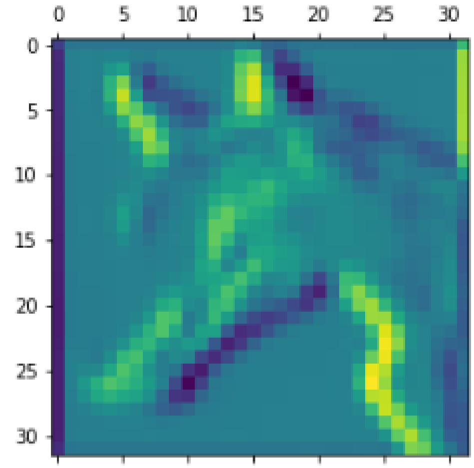


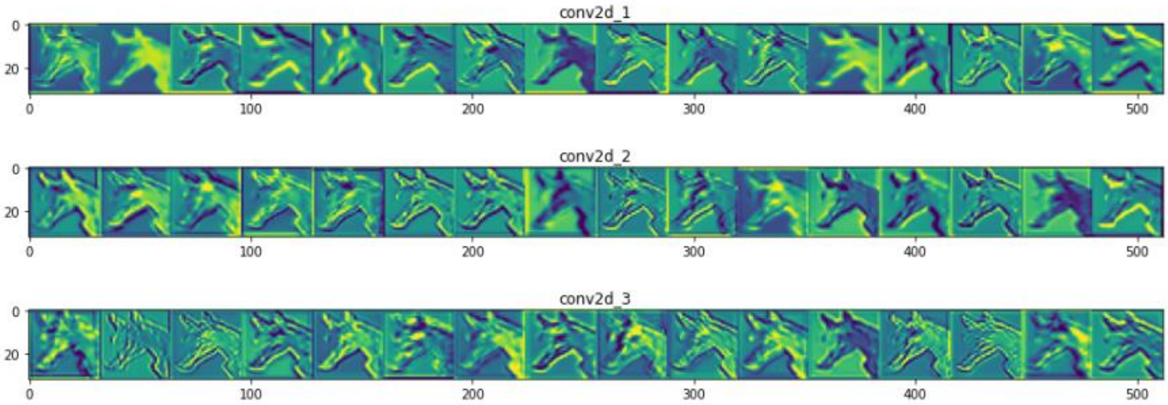
7.5. ResNet20 CIFAR v2

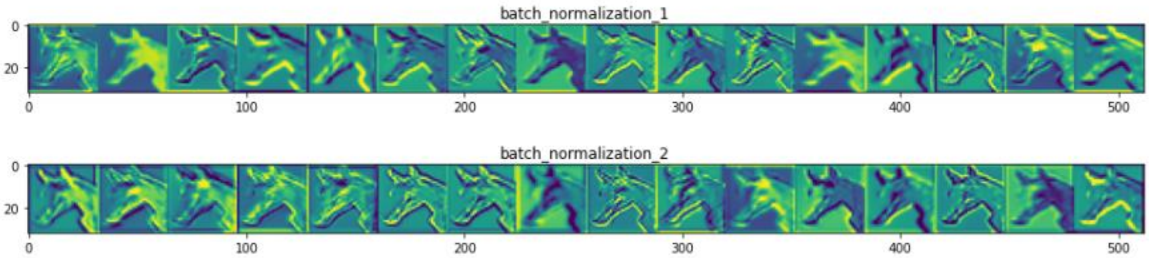


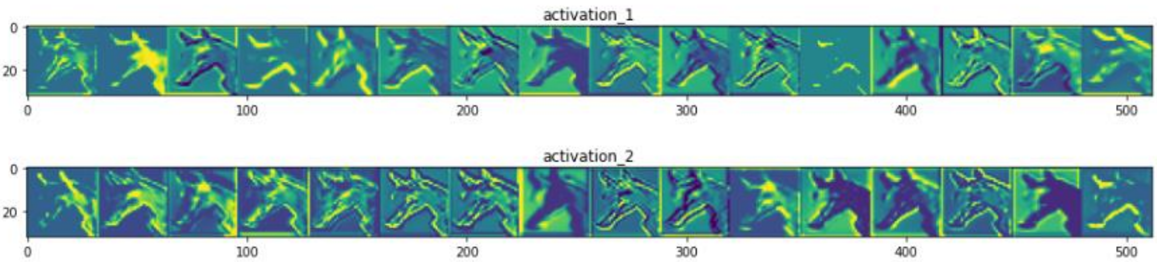


7.6. ResNet56 CIFAR v1

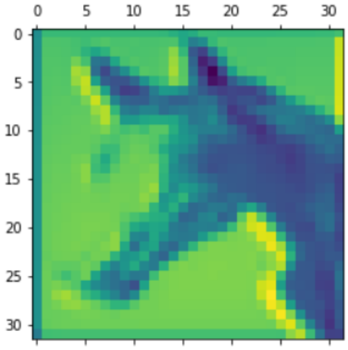








7.6.1. ResNET56 CIFAR v2



8. Tables and Images:[1]

8.1. Image 1

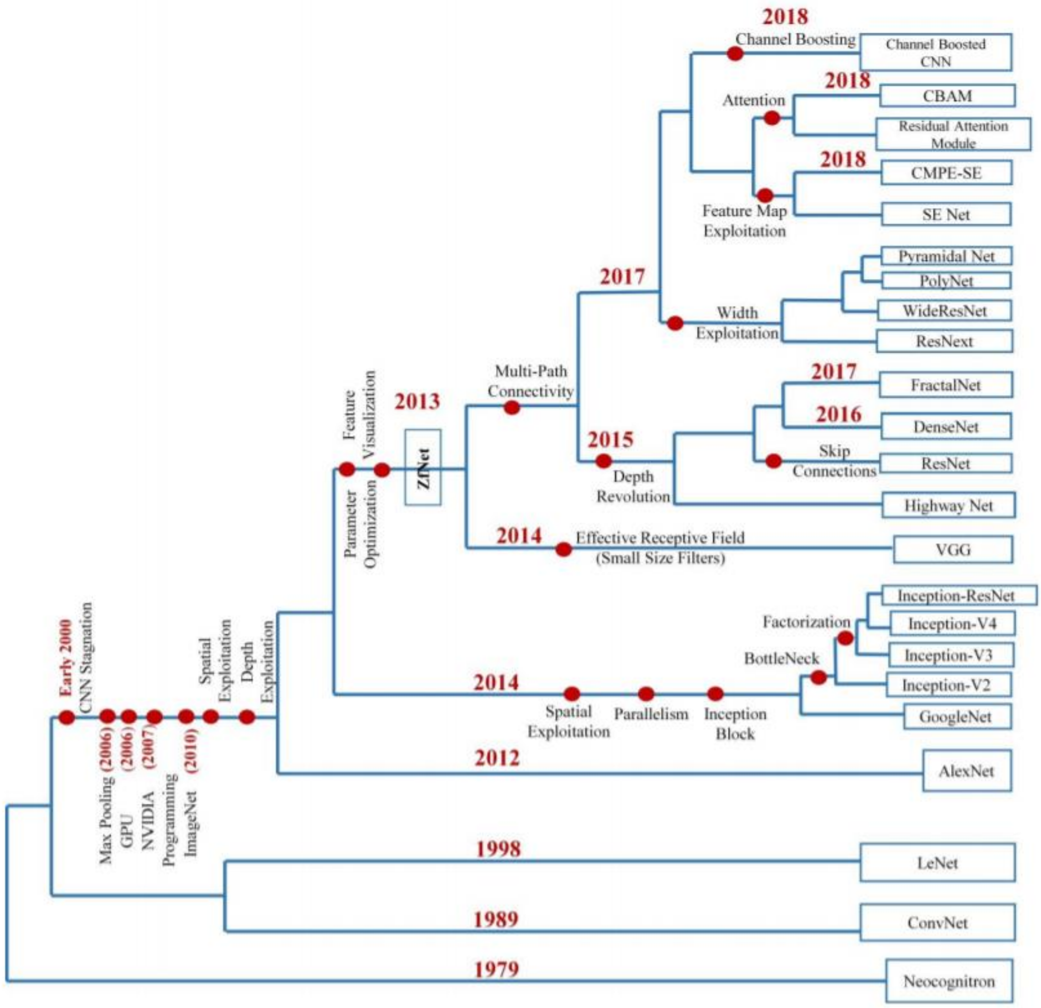


Fig. 3 Evolutionary history of deep CNNs showing architectural innovations from ConvNet till to date architectures.

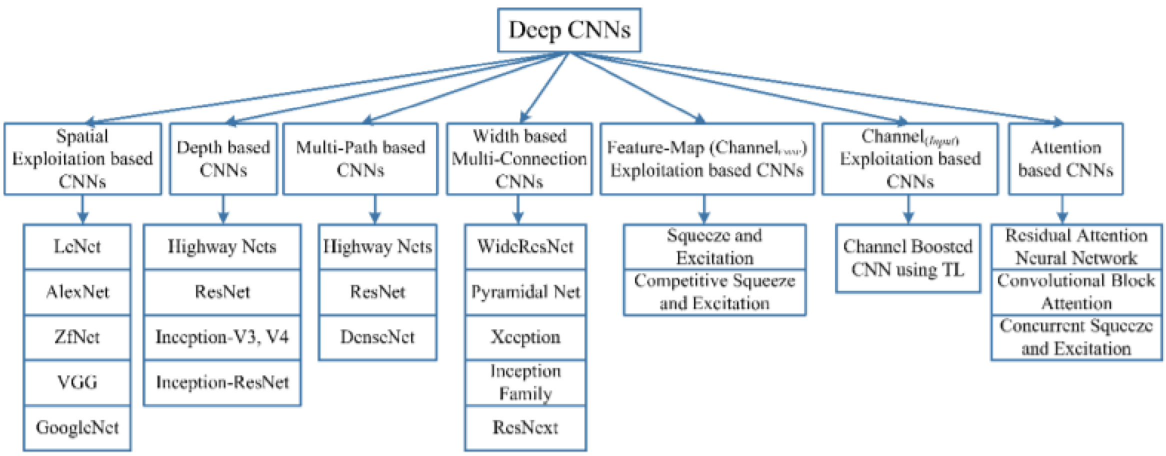


Fig. 4 Taxonomy of deep CNN architectures showing seven different categories.

Architecture Name	Year	Main contribution	Parameters	Error Rate	Depth	Category	Reference
LeNet	1998	- First popular CNN architecture	0.060 M	[dist]MNIST: 0.8 MNIST: 0.95	5	Spatial Exploitation	[69]
AlexNet	2012	- Deeper and wider than the LeNet - Uses Relu, dropout and overlap Pooling - GPUs NVIDIA GTX 580	60 M	ImageNet: 16.4	8	Spatial Exploitation	[26]
ZfNet	2014	-Visualization of intermediate layers	60 M	ImageNet: 11.7	8	Spatial Exploitation	[30]
VGG	2014	- Homogenous topology - Uses small size kernels	138 M	ImageNet: 7.3	19	Spatial Exploitation	[31]
GoogLeNet	2015	- Introduces block concept - Split transform and merge idea	4 M	ImageNet: 6.7	22	Spatial Exploitation	[32]
Inception-V3	2015	- Handles the problem of a representational bottleneck - Replace large size filters with small filters	23.6 M	ImageNet: 3.5 Multi-Crop: 3.58 Single-Crop: 5.6	159	Depth + Width	[126]
Highway Networks	2015	- Introduced an idea of Multi-path	2.3 M	CIFAR-10: 7.76	19	Depth + Multi-Path	[102]
Inception-V4	2016	- Split transform and merge idea Uses asymmetric filters	35 M	ImageNet: 4.01	70	Depth +Width	[35]
Inception-ResNet	2016	- Uses split transform merge idea and residual links	55.8M	ImageNet: 3.52	572	Depth + Width + Multi-Path	[35]
ResNet	2016	- Residual learning - Identity mapping based skip connections	25.6 M 1.7 M	ImageNet: 3.6 CIFAR-10: 6.43	152 110	Depth + Multi-Path	[33]
DelugeNet	2016	- Allows cross layer information flow in deep networks	20.2 M	CIFAR-10: 3.76 CIFAR-100: 19.02	146	Multi-path	[127]
FractalNet	2016	- Different path lengths are interacting with each other without any residual connection	38.6 M	CIFAR-10: 7.27 CIFAR-10+: 4.60 CIFAR-10++: 4.59 CIFAR-100: 28.20 CIFAR-100+: 22.49 CIFAR100++: 21.49	20 40	Multi-Path	[128]
Wide ResNet	2016	- Width is increased and depth is decreased	36.5 M	CIFAR-10: 3.89 CIFAR-100: 18.85	28 -	Width	[36]
Xception	2017	- Depth wise convolution followed by point wise convolution	22.8 M	ImageNet: 0.055	126	Width	[129]
Residual Attention Neural Network	2017	- Introduces attention mechanism	8.6 M	CIFAR-10: 3.90 CIFAR-100: 20.4 ImageNet: 4.8	452	Attention	[41]
ResNeXt	2017	- Cardinality - Homogeneous topology - Grouped convolution	68.1 M	CIFAR-10: 3.58 CIFAR-100: 17.31 ImageNet: 4.4	29 - 101	Width	[34]
Squeeze & Excitation Networks	2017	- Models interdependencies between feature-maps	27.5 M	ImageNet: 2.3	152	Feature-Map Exploitation	[116]
DenseNet	2017	- Cross-layer information flow	25.6 M 25.6 M 15.3 M 15.3 M	CIFAR-10+: 3.46 CIFAR100+:17.18 CIFAR-10: 5.19 CIFAR-100: 19.64	190 190 250 250	Multi-Path	[101]
PolyNet	2017	- Experimented structural diversity - Introduces Poly Inception module - Generalizes residual unit using polynomial compositions	92 M	ImageNet: Single:4.25 Multi:3.45	- -	Width	[38]
PyramidalNet	2017	- Increases width gradually per unit	116.4 M 27.0 M 27.0 M	ImageNet: 4.7 CIFAR-10: 3.48 CIFAR-100: 17.01	200 164 164	Width	[37]
Convolutional Block Attention Module (ResNeXt101 (32x4d) + CBAM)	2018	- Exploits both spatial and feature-map information	48.96 M	ImageNet: 5.59	101	Attention	[40]
Concurrent Spatial & Channel Excitation Mechanism	2018	- Spatial attention - Feature-map attention - Concurrent placement of spatial and channel attention	-	MALC: 0.12 Visceral: 0.09	-	Attention	[117]
Channel Boosted CNN	2018	- Boosting of original channels with additional information rich generated artificial channels	-	-	-	Channel Boosting	[39]
Competitive Squeeze & Excitation Network CMPE-SE-WRN-28	2018	- Residual and identity mappings both are used for rescaling the feature-map	36.92 M 36.90 M	CIFAR-10: 3.58 CIFAR-100: 18.47	152 152	Feature-Map Exploitation	[130]

[1] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi, "A survey of the recent architectures of deep convolutional neural networks," *Artif. Intell. Rev.*, pp. 1–68, 2020, doi: 10.1007/s10462-020-09825-6.

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *3rd Int. Conf. Learn. Represent. ICLR 2015 - Conf. Track Proc.*, pp. 1–14, 2015.

[3] C. Szegedy *et al.*, "Going deeper with convolutions," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 07-12-June-2015, pp. 1–9, 2015, doi: 10.1109/CVPR.2015.7298594.

[4] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-December, pp. 770–778, 2016, doi: 10.1109/CVPR.2016.90.

[5] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, “Inception-v4, inception-ResNet and the impact of residual connections on learning,” *31st AAAI Conf. Artif. Intell. AAAI 2017*, pp. 4278–4284, 2017.

[6] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” *Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017*, vol. 2017-January, pp. 2261–2269, 2017, doi: 10.1109/CVPR.2017.243.

[7] S. Zagoruyko and N. Komodakis, “Wide Residual Networks,” *Br. Mach. Vis. Conf. 2016, BMVC 2016*, vol. 2016-September, pp. 87.1-87.12, 2016, doi: 10.5244/C.30.87.

[8] F. Chollet, “Xception: Deep learning with depthwise separable convolutions,” *Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017*, vol. 2017-January, pp. 1800–1807, 2017, doi: 10.1109/CVPR.2017.195.

[9] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, “Aggregated residual transformations for deep neural networks,” *Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017*, vol. 2017-January, pp. 5987–5995, 2017, doi: 10.1109/CVPR.2017.634.

[10] F. Wang *et al.*, “Residual attention network for image classification,” *Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017*, vol. 2017-January, no. 1, pp. 6450–6458, 2017, doi: 10.1109/CVPR.2017.683.

[11] B. Recht, R. Roelofs, L. Schmidt, and V. Shankar, “Do CIFAR-10 Classifiers Generalize to CIFAR-10?,” pp. 1–25, 2018.

[12] C. Sun, A. Shrivastava, S. Singh, and A. Gupta, “Revisiting Unreasonable Effectiveness of Data in Deep Learning Era,” *Proc. IEEE Int. Conf. Comput. Vis.*, vol. 2017-October, pp. 843–852, 2017, doi: 10.1109/ICCV.2017.97.

[13] K. He, X. Zhang, S. Ren, and J. Sun, “Identity mappings in deep residual networks,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9908 LNCS, pp. 630–645, 2016, doi: 10.1007/978-3-319-46493-0_38.

[14] R. R. Network, “Table of Contents ResNet (Residual Network) BackGround,” pp. 1–27, 2020.

[15] D. R. Learning *et al.*, “Trains a ResNet on the CIFAR10 dataset {https://keras.io/examples/cifar10_resnet},” 2019.