

Deep Neural Networks For Image Classification

1. Introduction

Recent advances in deep learning has made Image Classification task possible. Deep Learning is excellent at recognizing patterns but requires a large amount of data and hence here all the models are trained using GPU. Deep learning excels in classifying an image as it is implemented with many layers of artificial neural networks where each layer is responsible for extracting one or more features of the image.

Image Datasets used:

- **MNIST** :The MNIST database of handwritten digits, has a training set of 55,000 examples and a test set of 10,000 examples and every image is 28 x 28 pixels. There are 10 classes in total (ie.0 - 9).
- **Cifar10** : The CIFAR-10 dataset consists of 60000, 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

2. Critical Analysis/Related Work

- LeNet-5[1] was used on large scale to automatically classify hand-written digits on bank cheques in the United States. LeNet-5 network is a convolutional neural network (CNN). CNNs are the foundation of modern state-of-the art method for deep learning based computer vision. LeNet is an important architecture because before it was invented, feature engineering by hand was used for character recognition, which was followed by a machine learning model to learn to classify hand engineered features. LeNet made hand engineering features redundant, because the network learns the best internal representation from raw images automatically.
- Alexnet[2] is deeper than [1], with more filters per layer, and with stacked convolutional layers. [2] uses ReLU for the nonlinearity functions (found to decrease training time as ReLUs are several times faster than the conventional tanh function). Data augmentation techniques is used which consisted of image translations, horizontal reflections, and patch extractions. Dropout layers are implemented in order to combat the problem of overfitting to the training data. The model is trained using batch stochastic gradient descent, with specific values for momentum and weight decay.
- ZFNet[3] has a similar architecture as that of [2] except for few modifications. And the model[3] was trained on lesser training samples while compared to [2]. Smaller filter size

was used and there was a decrease in stride value. A smaller filter size in the first convolution layer helps retain a lot of original pixel information in the input volume while in [2] larger filter size proved to be skipping a lot of relevant information.

- VGG[4], very deep convolutional networks uses a stack of convolutional layers with 3 x 3 filter size. Worked well on both image classification and localization tasks.
- GoogLeNet[5] used 9 Inception modules in the whole architecture, with about 100 layers in total. There was no usage of fully connected layers. They used an average pool instead. This saves a huge number of parameters. Uses 12x fewer parameters than AlexNet[2]. GoogLeNet was one of the first models that introduced the idea that CNN layers didn't always have to be stacked up sequentially.
- ResNet[6], short for Residual Networks is a classic neural network used as a backbone for many computer vision tasks which introduced the concept of skip connection. And being able to train very deep networks, it avoided the problem of vanishing gradients.

3. Method/Model Description

VGG 13:

- It is a stack of convolutional layers of different depths (here we have used 10 layers including the pooling layers) with 3 fully connected layers equipped with rectification non-linearity.

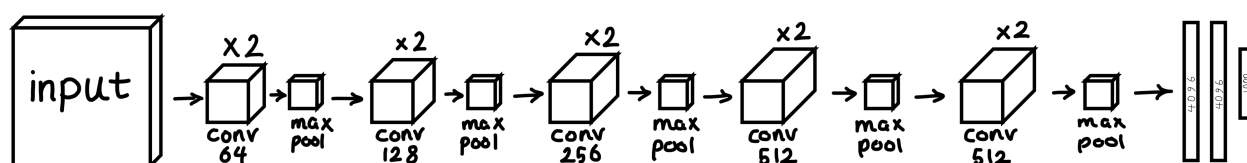


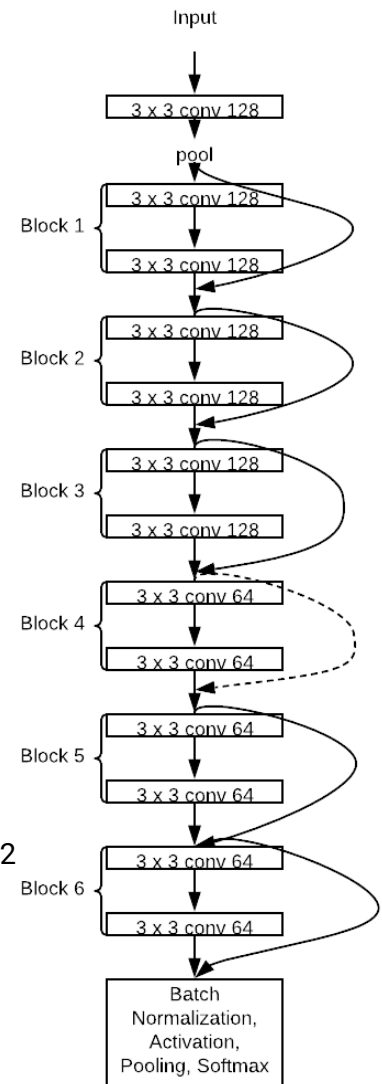
Fig 1: Architecture of VGG13

- In the model built, 5 convolutional layers and 5 pooling layers are used where 3 x 3 is the size of the filter. Max pooling is performed after every convolutional layer over a window size of 2 x 2 with stride 2.
- The input to the network is either MNIST dataset or Cifar10 dataset. For Mnist dataset the input size is 28 x 28 x 1 [Gray scale] whereas for Cifar10 it is 32 x 32 x 3 [color image]
- The first two fully connected layers have 4096 units each and the last FC layer has 1000 units. The final layer is the softmax which gives the output of class to which the image belongs to. Every hidden layer is equipped with ReLu to ensure non-linearity.

Fig 2: Architecture of Resnet-32

ResNet:

- Fig 2 shows the structure of the residual network used.
- The MNIST or Cifar10 datasets of size (28,28,1) and (32,32,3) are fed into the network.
- The network contains 6 residual blocks in total, 3 blocks has 128 filters and other three has 64 filters of 3 x 3 size.
- In residual network the input of one block is the summation of the output and input of the previous block.
- For the first residual block the input is the reshaped input image.
- The structure of the residual block makes it possible to avoid the problem of vanishing gradients.
- ReLu is used to maintain non-linearity
- Every block contains Convolutional layers, Batch normalization and ReLu operations.
- By the end of all blocks there is a pooling layer and softmax operation.
- The Resnet network build has 32 layers and hence the model is Resnet32



RNN:

In recurrent neural network, each input is evaluated on a single layer and an output is given. This can occur on a one-to-one, one-to-many, many-to-one or many-to-many input to output basis. As the RNN analyzes the sequential features of the input, an output is returned to the analysis step in a feedback loop, allowing the current feature to be analyzed in the context of the previous features.

- MNIST data is fed into BasicRNNcell with 128 input neurons.
- The network is optimized using Adam optimizer.
- In RNN, the output of one layer acts as feedback to the next layer.

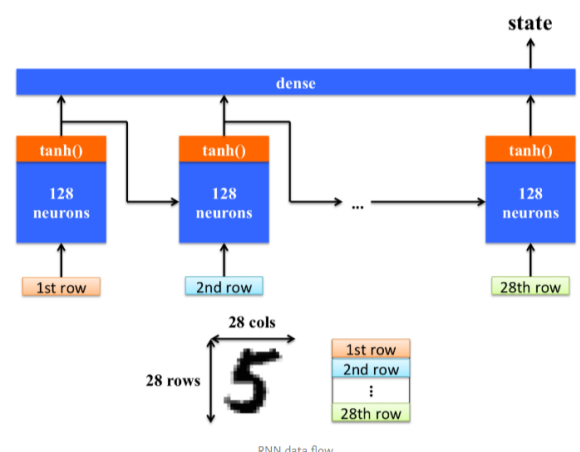
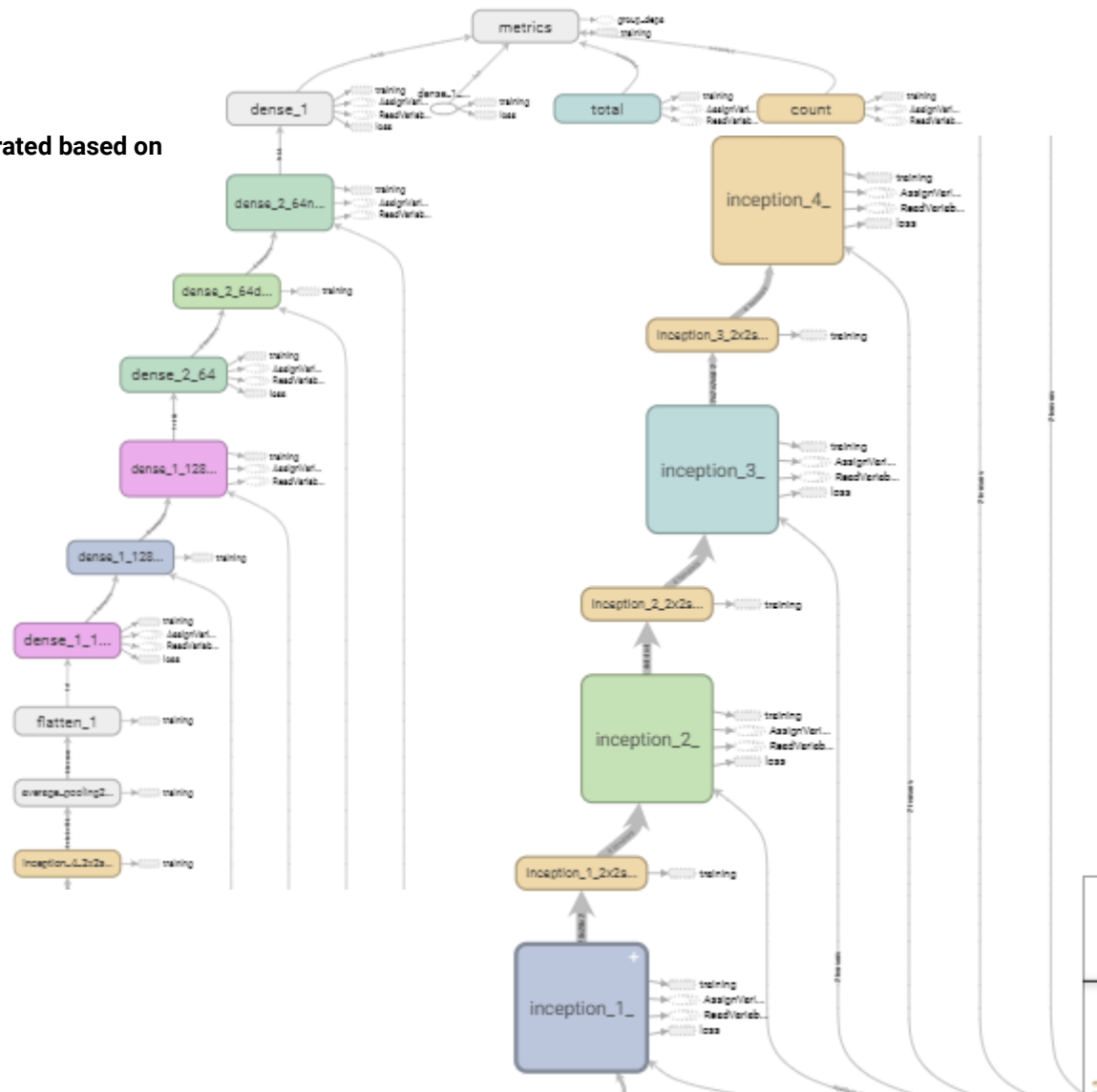


Fig 3: Architecture of RNN

GoogleNet:

Fig 4: Tensorflow graph generated based on the model trained



- MNIST or Cifar10 datasets are fed as input to GoogLeNet Neural Network with (28,28,1) and (32,32,3) shapes respectively.
- Fig 4 is the tensorflow graph generated after training the model with GoogLeNet
- The model consists of 4 Inception modules and inside every inception module, there are 1x1, 3x3 and 5x5 convolutional layers and a pooling layer as shown in Fig5.

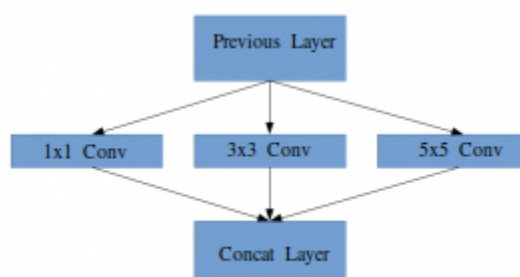


Fig 5: Inception module

- After the inception modules, there are 2 dense layers and finally softmax outputs the predicted value

4. Experiments

VGG (Very Deep convolutional network)

MNIST:

```
INFO:tensorflow:global_step/sec: 18.7978
INFO:tensorflow:loss = 0.052864876, step = 54000 (5.322 sec)
INFO:tensorflow:global_step/sec: 18.7911
INFO:tensorflow:loss = 0.02806877, step = 54100 (5.321 sec)
INFO:tensorflow:global_step/sec: 18.7873
INFO:tensorflow:loss = 0.019597799, step = 54200 (5.322 sec)
INFO:tensorflow:global_step/sec: 18.7822
INFO:tensorflow:loss = 0.017228233, step = 54300 (5.326 sec)
INFO:tensorflow:global_step/sec: 18.7741
INFO:tensorflow:loss = 0.052672286, step = 54400 (5.325 sec)
INFO:tensorflow:global_step/sec: 18.7902
INFO:tensorflow:loss = 0.04304565, step = 54500 (5.320 sec)
INFO:tensorflow:global_step/sec: 18.8065
INFO:tensorflow:loss = 0.007220093, step = 54600 (5.319 sec)
INFO:tensorflow:global_step/sec: 18.7914
INFO:tensorflow:loss = 0.04287889, step = 54700 (5.320 sec)
INFO:tensorflow:global_step/sec: 18.7695
INFO:tensorflow:loss = 0.056346122, step = 54800 (5.330 sec)
INFO:tensorflow:global_step/sec: 18.7598
INFO:tensorflow:loss = 0.10508007, step = 54900 (5.330 sec)
INFO:tensorflow:Saving checkpoints for 55000 into mnist_vgg13_model/model.ckpt.
WARNING:tensorflow:From /tensorflow-1.15.2/python3.6/tensorflow_core/python/training/saver.py:963: remove_checkpoint (f
Instructions for updating:
Use standard file APIs to delete files with this prefix.
INFO:tensorflow:Loss for final step: 0.022264814.
<tensorflow_estimator.python.estimator.estimator.Estimator at 0x7f5d9300e400>
```

Fig 6: Model Training of VGG13 on MNIST dataset

```
[10] eval_input_fn = tf.estimator.inputs.numpy_input_fn(x={"x": eval_data},
                                                         y=eval_labels,
                                                         num_epochs=1,
                                                         shuffle=False)

eval_results = mnist_classifier.evaluate(input_fn=eval_input_fn)
print(eval_results)

INFO:tensorflow:Calling model_fn.
(?, 28, 28, 1)
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Starting evaluation at 2020-05-08T06:38:13Z
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from mnist_vgg13_model/model.ckpt-55000
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Finished evaluation at 2020-05-08-06:38:16
INFO:tensorflow:Saving dict for global step 55000: accuracy = 0.9803, global_step = 55000, loss = 0.061157715
INFO:tensorflow:Saving 'checkpoint_path' summary for global step 55000: mnist_vgg13_model/model.ckpt-55000
{'accuracy': 0.9803, 'loss': 0.061157715, 'global_step': 55000}
```

Fig 7: Model Testing of VGG13 on MNIST dataset

```
INFO:tensorflow:Done calling model_fn.  
INFO:tensorflow:Graph was finalized.  
INFO:tensorflow:Restoring parameters from mnist_vgg13_model/model.ckpt-55000  
INFO:tensorflow:Running local_init_op.  
INFO:tensorflow:Done running local_init_op.  
The predicted label of index 2:  
1.0
```

```
[16] index = 2  
plt.imshow(eval_data[index].reshape(28, 28))  
print("The Actual label of index 2 :")  
print ("y = " + str(np.squeeze(eval_labels[index])))
```

```
The Actual label of index 2 :  
y = 1
```

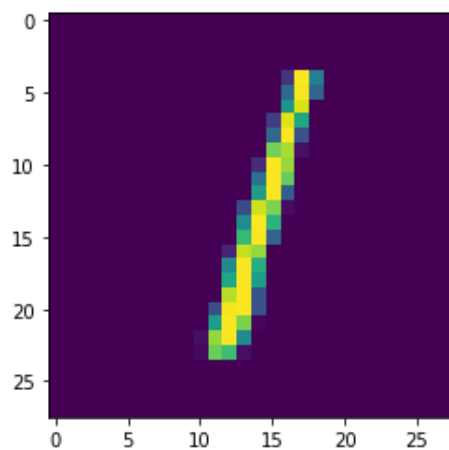


Fig 8: Predicted and Actual output of Testing sample 2

CIFAR 10:

```
[9] INFO:tensorflow:global_step/sec: 18.0333  
INFO:tensorflow:loss = 0.002338301, step = 49000 (5.539 sec)  
INFO:tensorflow:global_step/sec: 18.0101  
INFO:tensorflow:loss = 0.0009966153, step = 49100 (5.552 sec)  
INFO:tensorflow:global_step/sec: 18.0311  
INFO:tensorflow:loss = 0.00091284123, step = 49200 (5.547 sec)  
INFO:tensorflow:global_step/sec: 18.0359  
INFO:tensorflow:loss = 0.00090393523, step = 49300 (5.543 sec)  
INFO:tensorflow:global_step/sec: 18.0507  
INFO:tensorflow:loss = 0.0005410614, step = 49400 (5.539 sec)  
INFO:tensorflow:global_step/sec: 18.0373  
INFO:tensorflow:loss = 0.0011937991, step = 49500 (5.547 sec)  
INFO:tensorflow:global_step/sec: 18.0238  
INFO:tensorflow:loss = 0.00068225956, step = 49600 (5.548 sec)  
INFO:tensorflow:global_step/sec: 18.0531  
INFO:tensorflow:loss = 0.0005743781, step = 49700 (5.537 sec)  
INFO:tensorflow:global_step/sec: 18.0457  
INFO:tensorflow:loss = 0.0004845308, step = 49800 (5.544 sec)  
INFO:tensorflow:global_step/sec: 18.0102  
INFO:tensorflow:loss = 0.00094833696, step = 49900 (5.552 sec)  
INFO:tensorflow:Saving checkpoints for 50000 into /tmp/cifar10_vgg13_model/model.ckpt.  
WARNING:tensorflow:From /tensorflow-1.15.2/python3.6/tensorflow_core/python/training/saver.py:963: remove_checkpoint (f  
Instructions for updating:  
Use standard file APIs to delete files with this prefix.  
INFO:tensorflow:Loss for final step: 0.0008647936.  
<tensorflow_estimator.python.estimator.estimator.Estimator at 0x7fe9311d4e10>
```

Fig 9: Model Training of VGG13 on CIFAR 10 dataset

```
eval_input_fn = tf.estimator.inputs.numpy_input_fn(x={"x": eval_data},
                                                    y=eval_labels,
                                                    num_epochs=1,
                                                    shuffle=False)
eval_results = cifar10_classifier.evaluate(input_fn=eval_input_fn)
print(eval_results)
```

```
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Starting evaluation at 2020-05-08T07:38:25Z
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from /tmp/cifar10_vgg13_model/model.ckpt-50000
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Finished evaluation at 2020-05-08-07:38:27
INFO:tensorflow:Saving dict for global step 50000: accuracy = 0.6905, global_step = 50000, loss = 2.239098
INFO:tensorflow:Saving 'checkpoint_path' summary for global step 50000: /tmp/cifar10_vgg13_model/model.ckpt-50000
{'accuracy': 0.6905, 'loss': 2.239098, 'global_step': 50000}
```

Fig 10: Model Testing of VGG13 on CIFAR 10 dataset

```
[11] INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from /tmp/cifar10_vgg13_model/model.ckpt-50000
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
The predicted class of index 7 is:
6.0
```

```
index = 7
plt.imshow(eval_data[index].reshape(32, 32, 3))
print("The actual class of index 7 is: ")
print ("y = " + str(np.squeeze(eval_labels[index])))
```

```
The actual class of index 7 is:
y = 6
```

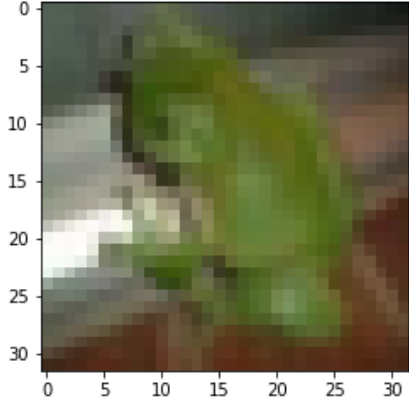


Fig 11: Predicted and Actual output of Testing sample 7

ResNet (Residual Neural Network)

MNIST:

```

429/430 [=====>.] - ETA: 0s - loss: 0.0190 - acc: 0.9937/epoch 1/80
10000/430 [=====] - 44s 102ms/step - loss: 0.0196 - acc: 0.9937 - val_loss: 0.0380 - val_acc: 0.9873
Epoch 77/80
429/430 [=====>.] - ETA: 0s - loss: 0.0205 - acc: 0.9929/epoch 1/80
10000/430 [=====] - 44s 101ms/step - loss: 0.0205 - acc: 0.9928 - val_loss: 0.0373 - val_acc: 0.9882
Epoch 78/80
429/430 [=====>.] - ETA: 0s - loss: 0.0182 - acc: 0.9942/epoch 1/80
10000/430 [=====] - 44s 101ms/step - loss: 0.0182 - acc: 0.9941 - val_loss: 0.0466 - val_acc: 0.9850
Epoch 79/80
429/430 [=====>.] - ETA: 0s - loss: 0.0194 - acc: 0.9939/epoch 1/80
10000/430 [=====] - 44s 101ms/step - loss: 0.0194 - acc: 0.9939 - val_loss: 0.0288 - val_acc: 0.9896
Epoch 80/80
429/430 [=====>.] - ETA: 0s - loss: 0.0182 - acc: 0.9942/epoch 1/80
10000/430 [=====] - 44s 101ms/step - loss: 0.0183 - acc: 0.9942 - val_loss: 0.0311 - val_acc: 0.9896
<tensorflow.python.keras.callbacks.History at 0x7f81bebb6780>

```

Fig 12: Training and Testing of Resnet on MNIST dataset

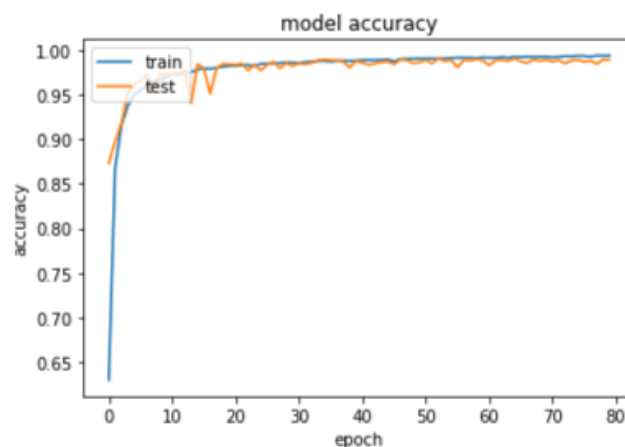


Fig 13: Graph of model accuracy vs epoch

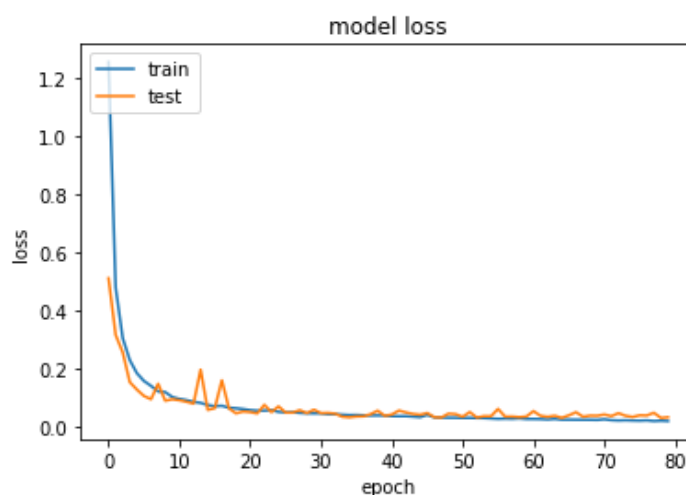


Fig 14: Graph of model loss vs epoch

CIFAR 10:

```
[24] 390/391 [=====>.] - ETA: 0s - loss: 0.1583 - acc: 0.9466Epoch 1/80
10000/391 [=====] - 119s 304ms/step - loss: 0.1582 - acc: 0.9467 - val_loss: 0.5740 - val_acc: 0.8521
Epoch 78/80
390/391 [=====>.] - ETA: 0s - loss: 0.1574 - acc: 0.9467Epoch 1/80
10000/391 [=====] - 119s 303ms/step - loss: 0.1574 - acc: 0.9466 - val_loss: 0.8099 - val_acc: 0.7999
Epoch 79/80
390/391 [=====>.] - ETA: 0s - loss: 0.1591 - acc: 0.9448Epoch 1/80
10000/391 [=====] - 119s 304ms/step - loss: 0.1593 - acc: 0.9447 - val_loss: 0.6734 - val_acc: 0.8181
Epoch 80/80
390/391 [=====>.] - ETA: 0s - loss: 0.1486 - acc: 0.9494Epoch 1/80
10000/391 [=====] - 119s 304ms/step - loss: 0.1485 - acc: 0.9494 - val_loss: 0.8021 - val_acc: 0.7960
<tensorflow.python.keras.callbacks.History at 0x7ff86e1a1198>
```

Fig 15: Training and Testing of Resnet on Cifar 10 dataset

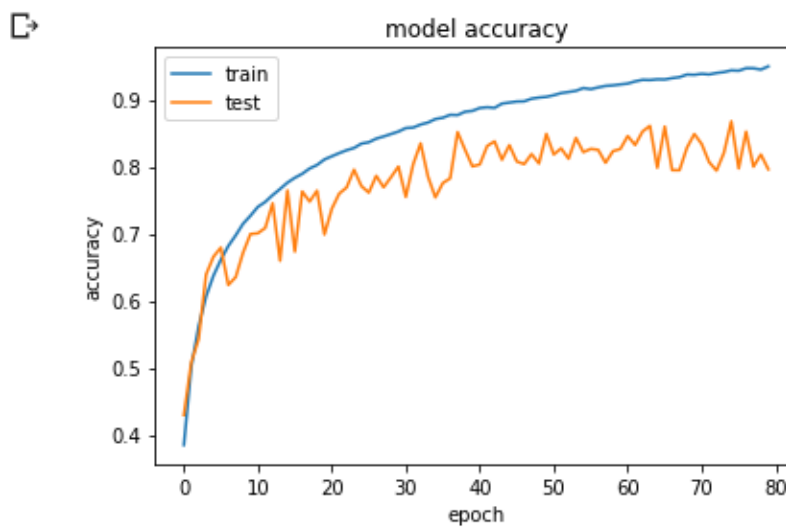


Fig 16: Graph of model accuracy vs epoch

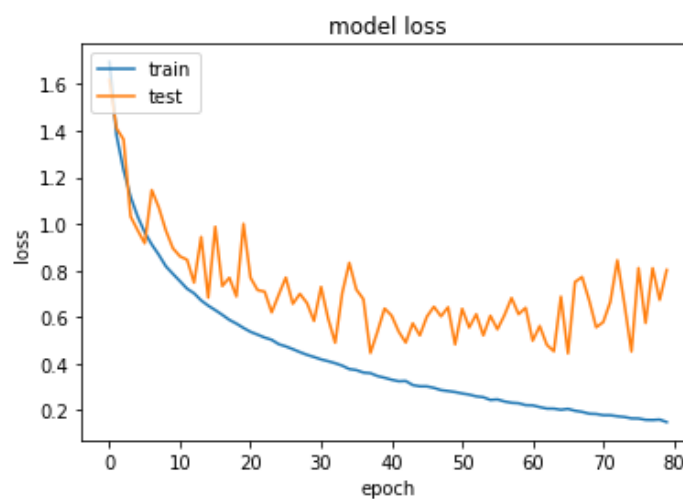


Fig 17: Graph of model loss vs epoch

RNN(Recurrent Neural Network)

MNIST:

```
Epoch: 1, Train Loss: 0.257, Train Acc: 0.922  
Epoch: 2, Train Loss: 0.172, Train Acc: 0.953  
Epoch: 3, Train Loss: 0.157, Train Acc: 0.969  
Epoch: 4, Train Loss: 0.157, Train Acc: 0.945  
Epoch: 5, Train Loss: 0.055, Train Acc: 0.977  
Epoch: 6, Train Loss: 0.074, Train Acc: 0.977  
Epoch: 7, Train Loss: 0.057, Train Acc: 0.977  
Epoch: 8, Train Loss: 0.119, Train Acc: 0.961  
Epoch: 9, Train Loss: 0.038, Train Acc: 0.992  
Epoch: 10, Train Loss: 0.129, Train Acc: 0.969  
Test Loss: 0.123, Test Acc: 0.965
```

Fig 18: Training and Testing of RNN on MNIST dataset

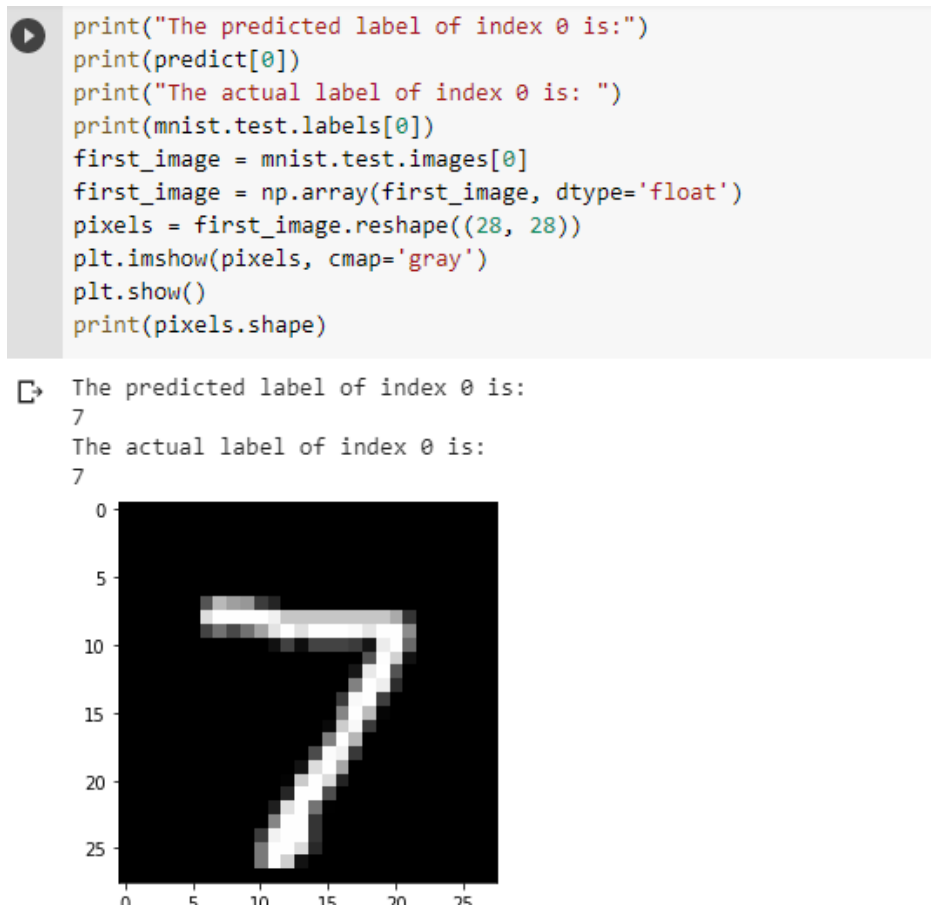


Fig 19: Predicted and actual output of a sample of RNN model on MNIST dataset

GoogleNET

MNIST:

```
Epoch 80/100
44000/44000 [=====] - 19s 440us/step - loss: 0.0047 - accuracy: 1.0000 - val_loss: 0.0097 - val_accuracy: 0.9988
Epoch 80080: val_loss did not improve from 0.00971
Epoch 81/100
44000/44000 [=====] - 19s 440us/step - loss: 0.0047 - accuracy: 1.0000 - val_loss: 0.0097 - val_accuracy: 0.9988
Epoch 80081: val_loss did not improve from 0.00971
Epoch 82/100
44000/44000 [=====] - 19s 438us/step - loss: 0.0047 - accuracy: 1.0000 - val_loss: 0.0097 - val_accuracy: 0.9988
Epoch 80082: val_loss did not improve from 0.00971
Epoch 83/100
44000/44000 [=====] - 19s 439us/step - loss: 0.0047 - accuracy: 1.0000 - val_loss: 0.0098 - val_accuracy: 0.9988
Epoch 80083: val_loss did not improve from 0.00971
Epoch 80083: early stopping
10000/10000 [=====] - 3s 263us/step
Accuracy on test set: 99.87298846244812 %
```

Fig 20: Model Training of GoogleNet on MNIST dataset

```
[ ] model.load_weights('./models/mnist-nrcrt7-06:48PM_May-04-2020.hdf5')

result = model.evaluate(x_test_gray, y_test_cat)

print(result)

10000/10000 [=====] - 2s 245us/step
[0.00949306582286954, 0.9987298846244812]
```

Fig 21: Model Testing of GoogleNet on MNIST dataset

```
[20] predict = model.predict(x_test_gray)
m = max(predict[0])
index = [i for i,j in enumerate(predict[0]) if j == m]
print("The value of the prediction of test sample with index 0 is :")
print(index)
```

```
The value of the prediction of test sample with index 0 is :
[7]
```

```
print("The actual class of test sample with index 0 is: ")
plt.imshow(x_test_gray[0,:,:], cmap='gray')
plt.show()
```

```
The actual class of test sample with index 0 is:
```

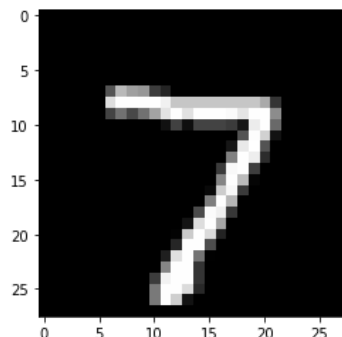


Fig 22: Predicted and Actual output of Testing sample

CIFAR 10:

```

Epoch 25/100
40000/40000 [=====] - 72s 2ms/step - loss: 0.0533 - accuracy: 0.9977 - val_loss: 0.1384 - val_accuracy: 0.9699

Epoch 00025: val_loss did not improve from 0.13199
Epoch 26/100
40000/40000 [=====] - 71s 2ms/step - loss: 0.0516 - accuracy: 0.9981 - val_loss: 0.1389 - val_accuracy: 0.9700

Epoch 00026: val_loss did not improve from 0.13199
Epoch 27/100
40000/40000 [=====] - 72s 2ms/step - loss: 0.0507 - accuracy: 0.9983 - val_loss: 0.1388 - val_accuracy: 0.9699

Epoch 00027: val_loss did not improve from 0.13199
Epoch 28/100
40000/40000 [=====] - 72s 2ms/step - loss: 0.0501 - accuracy: 0.9985 - val_loss: 0.1391 - val_accuracy: 0.9699

Epoch 00028: val_loss did not improve from 0.13199
Epoch 29/100
40000/40000 [=====] - 71s 2ms/step - loss: 0.0494 - accuracy: 0.9986 - val_loss: 0.1404 - val_accuracy: 0.9699

Epoch 00029: val_loss did not improve from 0.13199
Epoch 00029: early stopping
10000/10000 [=====] - 8s 815us/step
Accuracy on test set: 96.87296748161316 %

```

Fig 23: Model Training of GoogleNet on Cifar 10 dataset

```

model.load_weights('./models/cifar10-nrcrt7-04:40PM_May-08-2020.hdf5')

result = model.evaluate(x_test, y_test_cat)

print(result)

10000/10000 [=====] - 8s 794us/step
[0.13478883649110793, 0.9691197276115417]

```

Fig 24: Model Testing of GoogleNet on Cifar 10 dataset

```

[9] predict = model.predict(x_test)
m = max(predict[7])
index = [i for i,j in enumerate(predict[7]) if j == m]
print("The value of the prediction of test sample with index 7 is :")
print(index)

```

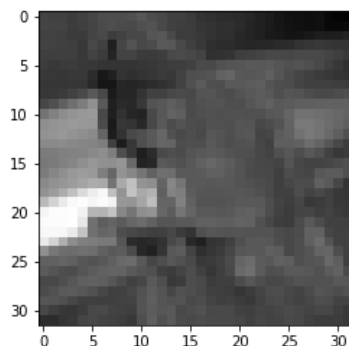
The value of the prediction of test sample with index 7 is :
[6]

```

print("The actual class of test sample with index 7 is: ")
plt.imshow(x_test[7,:,:], cmap='gray')
plt.show()

```

The actual class of test sample with index 7 is:



Label 6 represents Frog

Fig 25: Predicted and Actual output of Testing sample

5. Conclusion

Table 1: Comparison between different models

<i>Model</i>	<i>Dataset</i>	<i>Test Accuracy</i>	<i>Test Loss</i>	<i>Epoch</i>	<i>Approx Training time</i>
VGG -13	Mnist	98.03	0.0611	550	~ 50 mins
Resnet 32	Mnist	98.96	0.0311	80	~ 90 mins
RNN	Mnist	96.5	0.123	10	~ 3 mins
GoogleNet	Mnist	99.87	0.0094	83	~ 45 mins
VGG - 13	Cifar 10	69.05	2.239	500	~ 30 mins
Resnet 32	Cifar 10	79.60	0.8021	80	~ 120 mins
GoogleNet	Cifar 10	96.87	0.1347	29	~ 45 mins

- While training the model, normalization and the values of batch size, number of epochs should be taken into careful consideration. Giving a higher batch size will terminate training unsuccessfully because of limited RAM availability in most system configuration. Depending upon the model and dataset the number of epochs should be considered else the model will be underfitting, so giving a good higher value of no_of_epochs is better and when there is no improvement in training accuracy, the model stops early.
- From the above table we can observe that GoogleNet performs the best on both MNIST and Cifar 10 datasets, because it contains inception module which avoids overfitting and reduces the number of parameters.
- Training Resnet 32 takes longer duration because of it's very deep structure and presence of one fully connected layer
- Though Resnet 32 takes longer computation time, it doesn't give the best accuracy as GoogleNet, it is because in GoogleNet, the inception module allows for more efficient computation and deeper Networks through a dimensionality reduction with stacked 1×1 convolutions.
- RNN takes very less time to train because of the simplicity of the structure
- VGG 13 gives a good value of accuracy for MNIST dataset and doesn't perform so well on Cifar 10 dataset because the structure of VGG 13 is not as deep as Resnet 32 or Googlenet.

References

- [1] LeCun, Yann, et al. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE* 86.11 (1998): 2278-2324.
- [2] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems*. 2012.
- [3] Zeiler, Matthew D., and Rob Fergus. "Visualizing and understanding convolutional networks." *European conference on computer vision*. Springer, Cham, 2014.
- [4] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." *arXiv preprint arXiv:1409.1556* (2014).
- [5] Szegedy, Christian, et al. "Going deeper with convolutions." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015.
- [6] He, Kaiming, et al. "Deep residual learning for image recognition." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.

Appendix

<i>Sno</i>	<i>Figures, Tables and Graphs</i>	<i>Pg No</i>
1	Architecture of VGG-13	2
2	Architecture of ResNet-32	3
3	Architecture of RNN	3
4	Tensorflow graph generated based on model trained(GoogleNet)	4
5	Inception module	4
6	Model Training of VGG13 on MNIST dataset	5
7	Model Testing of VGG13 on MNIST dataset	5
8	Predicted and actual output of VGG on MNIST	6
9	Model Training of VGG13 on CIFAR10 dataset	6
10	Model Testing of VGG13 on CIFAR10 dataset	7
11	Predicted and actual output of VGG on CIFAR 10	7
12	Training and testing of Resnet on MNIST	8
13	Model accuracy vs epoch (Resnet - MNIST)	8
14	Model loss vs epoch (Resnet - MNIST)	8
15	Training and testing of Resnet on CIFAR 10	9
16	Model accuracy vs epoch (Resnet - CIFAR 10)	9
17	Model loss vs epoch (Resnet - CIFAR 10)	9
18	Training and testing of RNN on MNIST	10
19	Predicted and actual output of RNN on MNIST	10
20	Model Training of GoogleNet on MNIST dataset	11
21	Model Testing of GoogleNet on MNIST dataset	11
22	Predicted and actual output of GoogleNet on MNIST	11
23	Model Training of GoogleNet on CIFAR 10 dataset	12
24	Model Testing of GoogleNet on CIFAR 10 dataset	12
25	Predicted and actual output of GoogleNet on CIFAR 10	12
26	Comparison between different models	13