Deep Learning Fractal - 3

Assignment 1 Report

Debonil Ghosh
Roll No: M21AIE225
Executive MTech
Artificial Intelligence
Indian Institute of Technology, Jodhpur

Problem Statement:

Q1: Implement AE with three layers of encoders and decoders each (from scratch on your own) on MNIST database and compare with inbuilt function.

Q2: implement denoising AE (using AE from first question, convert into D-AE) and compare with inbuilt D-AE. (using MNIST database)

- Use accuracy and t-sne for comparison.

- compare with respect to Softmax loss and Center loss

Bonus question: using inbuilt function, compare AE and D-AE on CIFAR-10 database.

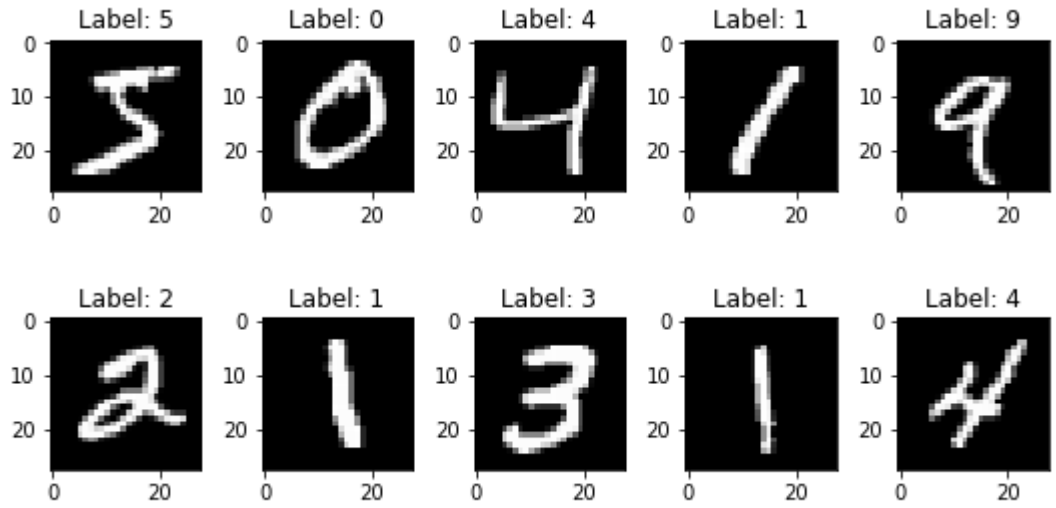
Submit your code files and a single report for the assignment. Report all the results along with some output generated by AE and D-AE.

Given Dataset: **MNIST Handwritten Digit**

Training Set: 60000 28×28 Gray (1 channels) Images

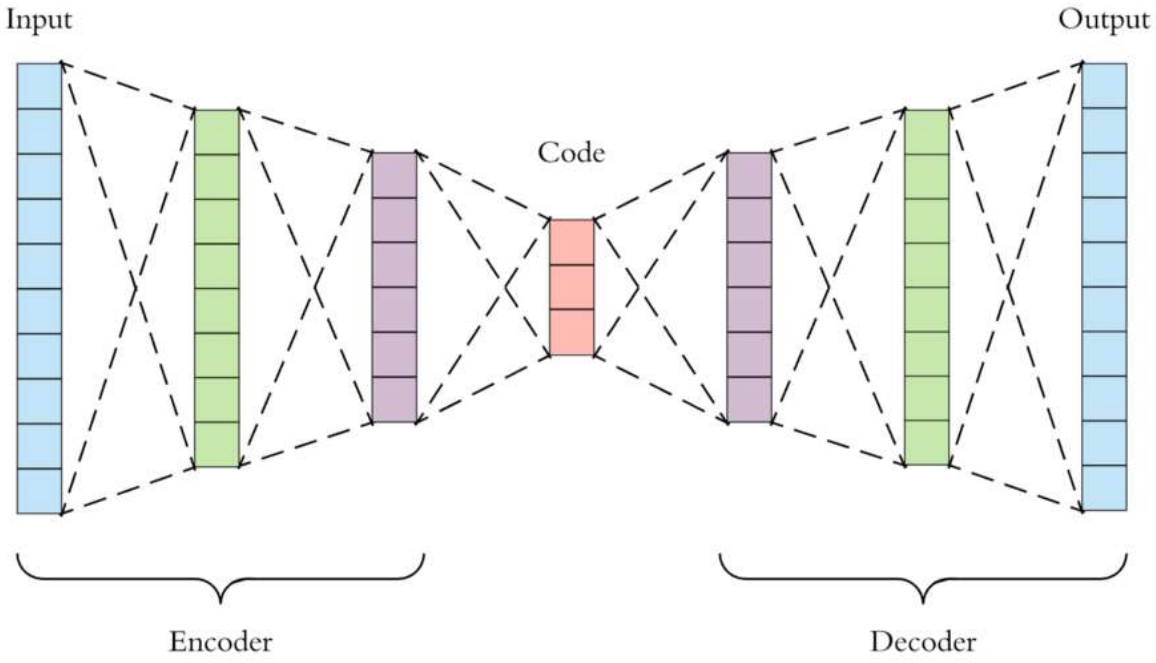
Testing Set: 10000 28×28 Gray (1 channels) Images

Data Samples:

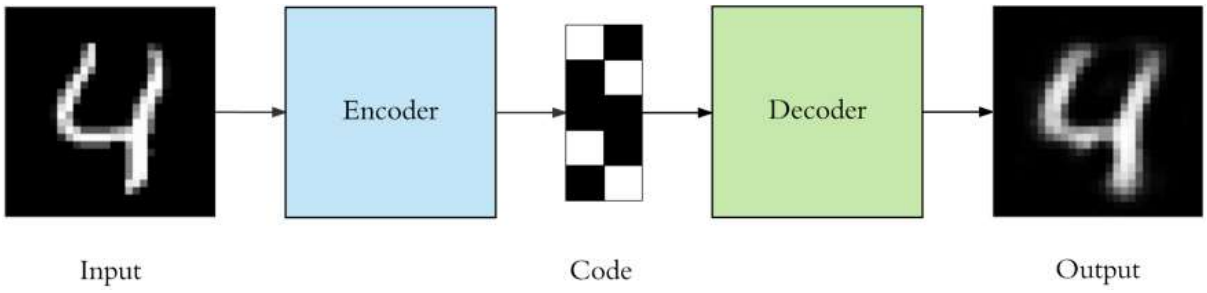


Q1: Implement AE with three layers of encoders and decoders each (from scratch on your own) on MNIST database and compare with inbuilt function.

Autoencoders are a specific type of feedforward neural networks where the input is the same as the output. They compress the input into a lower-dimensional code and then reconstruct the output from this representation. The code is a compact "summary" or "compression" of the input, also called the latent-space representation.



As visualized above, we can take an unlabelled dataset and frame it as a supervised learning problem tasked with outputting x^, a reconstruction of the original input x. This network can be trained by minimizing the reconstruction error, L(x,x^), which measures the differences between our original input and the consequent reconstruction. The bottleneck is a key attribute of our network design; without the presence of an information bottleneck, our network could easily learn to simply memorize the input values by passing these values along through the network.



Hyper Parameters Used:

Hyper Parameter	Value
Learning Rate	**0.003**
Number of Epochs	**15**
Batch Size	**64**
Loss Function	**Binary Cross Entropy Loss**
Optimizer	**Adam**

Model Used for AE from scratch:

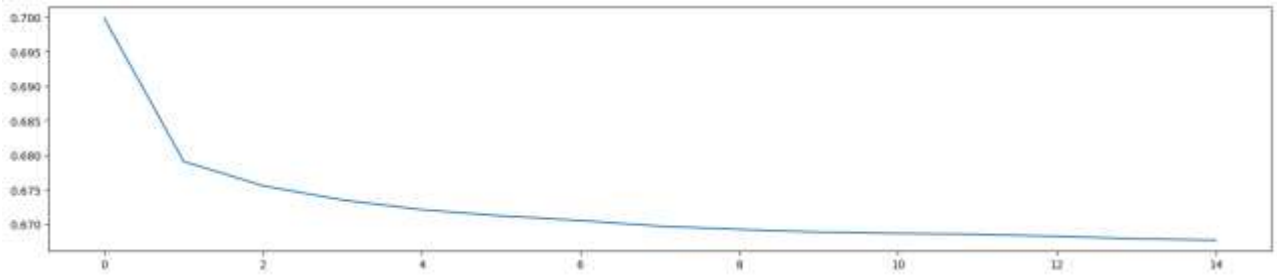
```
Autoencoder(
  (encoder): Sequential(
    (0): Linear(in_features=784, out_features=256, bias=True)
    (1): ReLU()
    (2): Linear(in_features=256, out_features=32, bias=True)
    (3): ReLU()
    (4): Linear(in_features=32, out_features=16, bias=True)
  )
  (decoder): Sequential(
    (0): Linear(in_features=16, out_features=32, bias=True)
    (1): ReLU()
    (2): Linear(in_features=32, out_features=256, bias=True)
    (3): ReLU()
    (4): Linear(in_features=256, out_features=784, bias=True)
    (5): Softmax(dim=None)
  )
)
```

Training of Model:

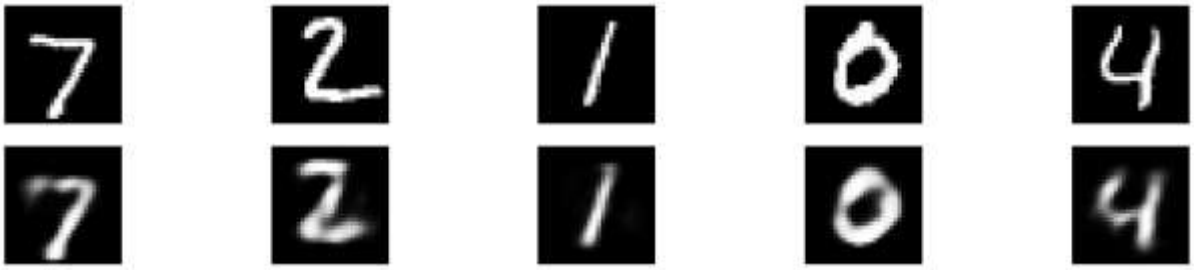
```
Epoch 0 - Training loss: 0.6998346765031184
Epoch 1 - Training loss: 0.679092197847773
Epoch 2 - Training loss: 0.675565375003225
Epoch 3 - Training loss: 0.6734950225998853
Epoch 4 - Training loss: 0.6720934793639031
Epoch 5 - Training loss: 0.6712066595361177
Epoch 6 - Training loss: 0.6705225137378107
Epoch 7 - Training loss: 0.6697057108762168
Epoch 8 - Training loss: 0.6692271794972897
Epoch 9 - Training loss: 0.6688430201905623
Epoch 10 - Training loss: 0.6686527947627151
Epoch 11 - Training loss: 0.6685410642039293
Epoch 12 - Training loss: 0.6682214680383963
Epoch 13 - Training loss: 0.6678789160780306
Epoch 14 - Training loss: 0.6676722010696875

Training Time (in minutes) = 6.382478229204813
```

Training Loss with epoch:



Sample Reconstructions using trained AE model from scratch:

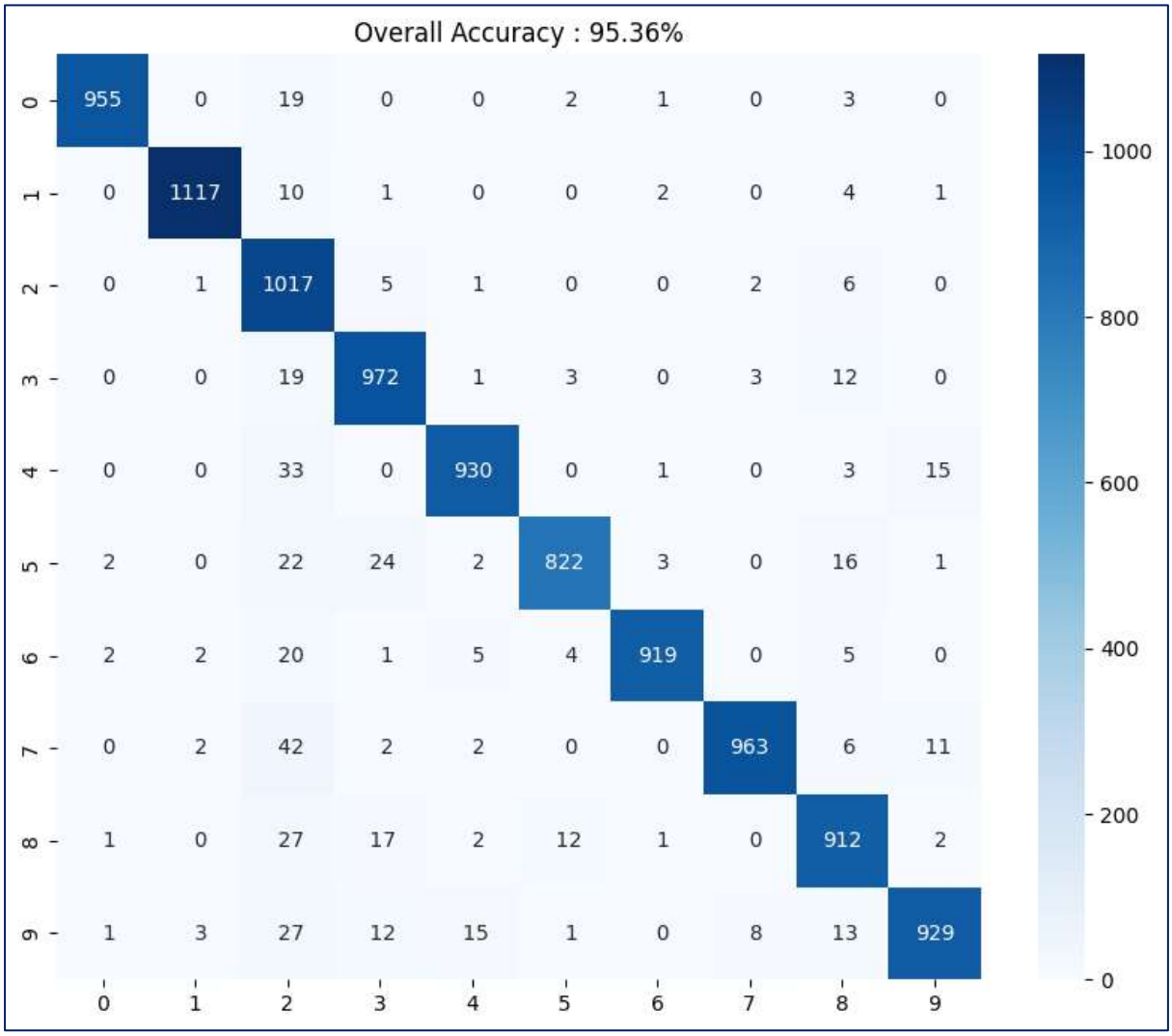


Classification Report :

Class	precision	recall	f1-score	support
0.0	0.99	0.97	0.98	980
1.0	0.99	0.98	0.99	1135
2.0	0.82	0.99	0.90	1032
3.0	0.94	0.96	0.95	1010
4.0	0.97	0.95	0.96	982
5.0	0.97	0.92	0.95	892
6.0	0.99	0.96	0.98	958
7.0	0.99	0.94	0.96	1028
8.0	0.93	0.94	0.93	974
9.0	0.97	0.92	0.94	1009

Accuracy			0.95	10000
Macro Avg	0.96	0.95	0.95	10000
Weighted Avg	0.96	0.95	0.95	10000

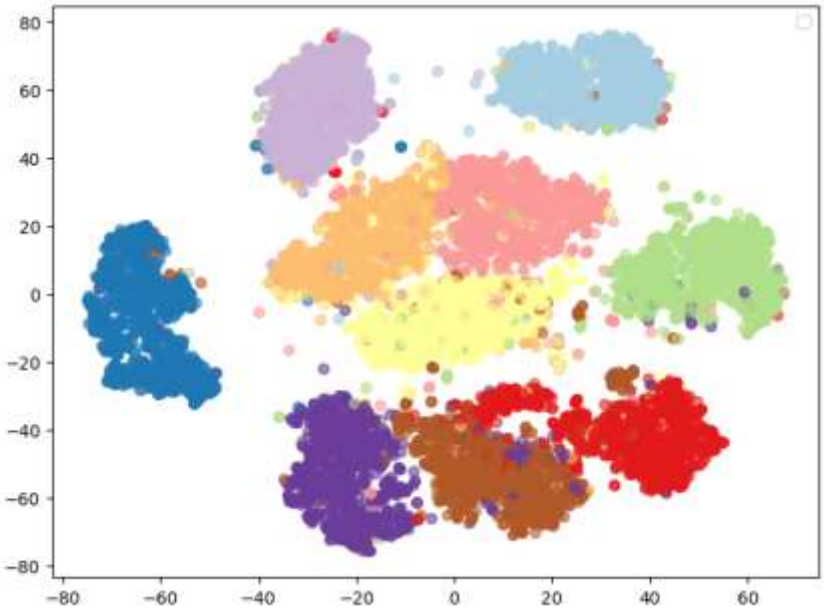
Confusion Matrix:

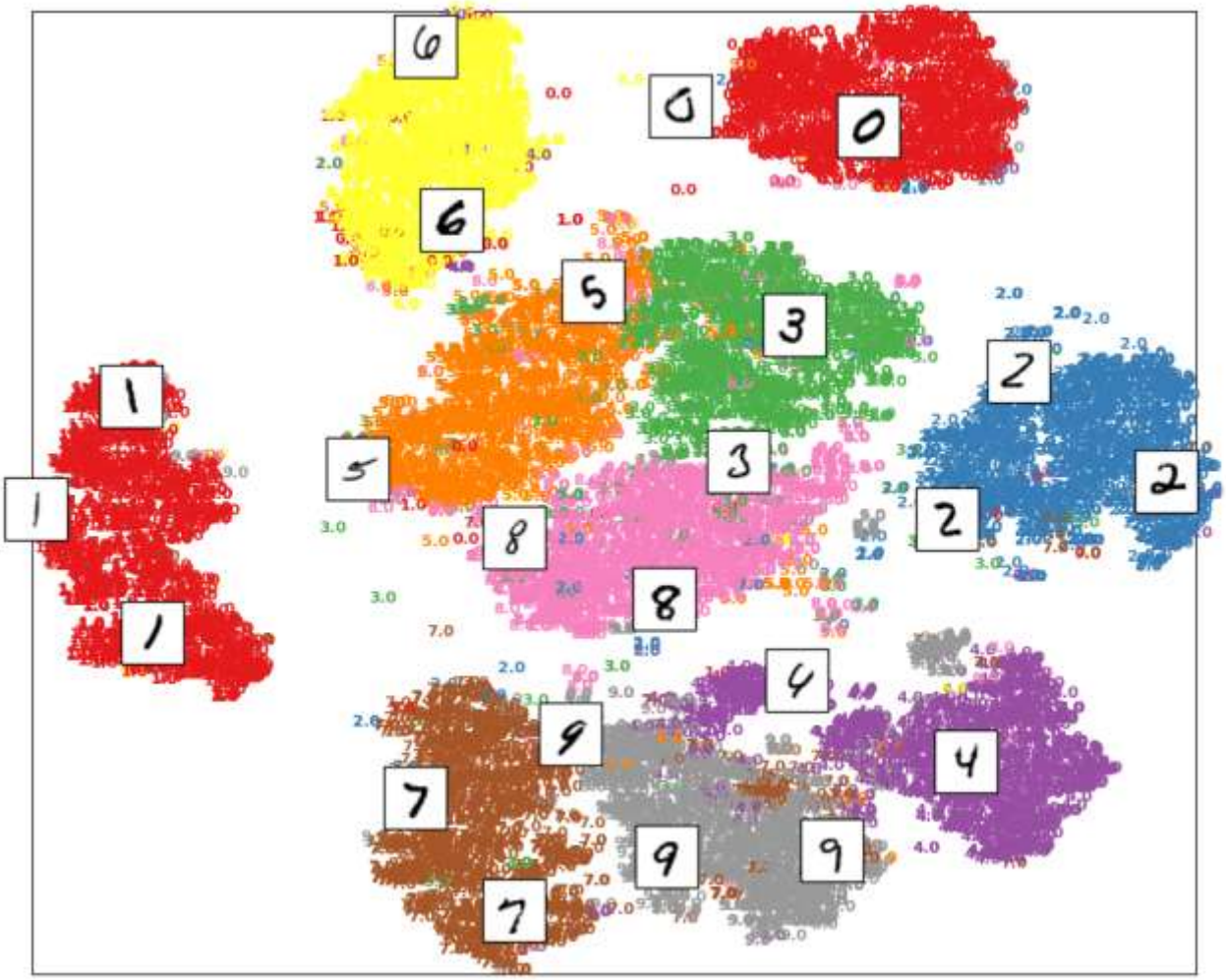


Class wise Accuracy Score:

Class	0	1	2	3	4	5	6	7	8	9
Accy.	97.45	98.41	98.55	96.24	94.70	92.15	95.93	93.68	93.63	92.07

T-SNE plot on embedding of AE from scratch:





Q2: Implement denoising AE (using AE from first question, convert into D-AE) and compare with inbuilt D-AE. (using MNIST database)

Denoising autoencoders:

Keeping the code layer small forced our autoencoder to learn an intelligent representation of the data. There is another way to force the autoencoder to learn useful features, which is adding random noise to its inputs and making it recover the original noise-free data. This way the autoencoder can't simply copy the input to its output because the input also contains random noise. We are asking it to subtract the noise and produce the underlying meaningful data. This is called a denoising autoencoder.

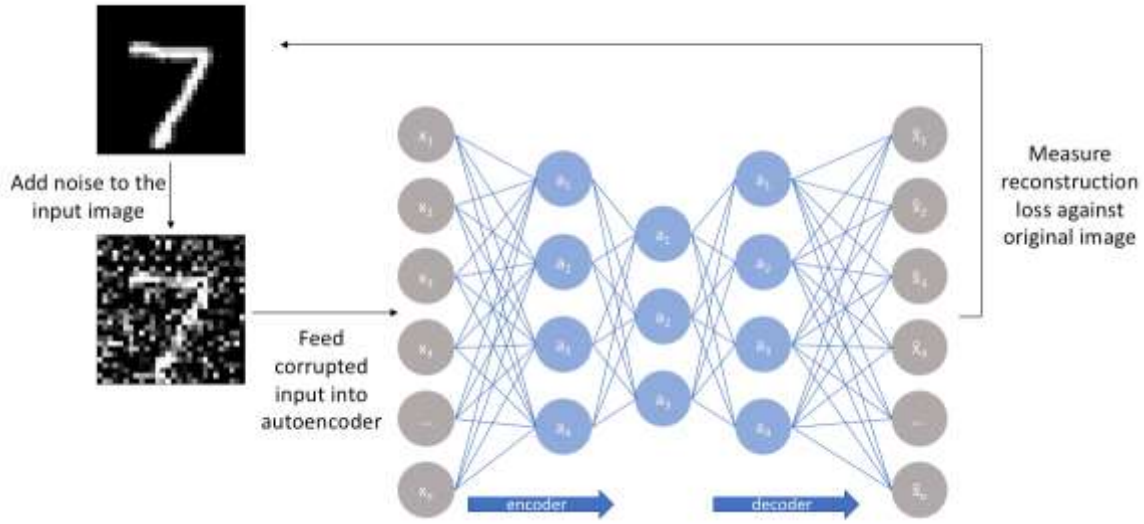


Image source: https://www.jeremyjordan.me/autoencoders/

Hyper Parameters Used:

Hyper Parameter	Value
Learning Rate	**0.0002**
Number of Epochs	**20**
Batch Size	**64**
Loss Function	**Cross Entropy Loss**
Optimizer	**Adam**

Model Used:

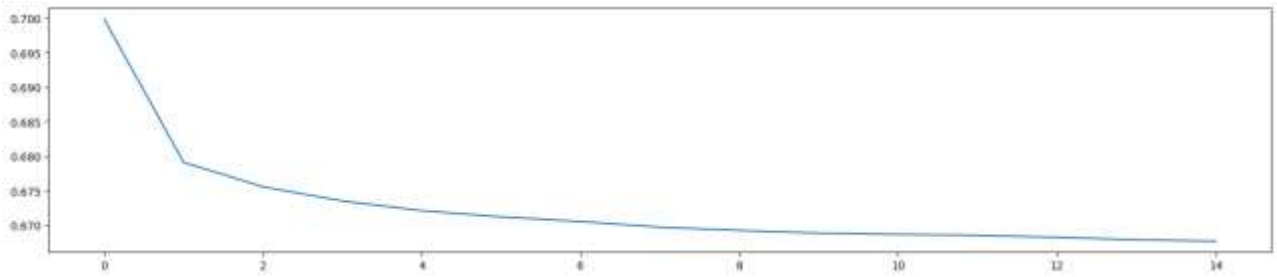
```
CNN(
  (conv_model): Sequential(
    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), padding_mode=replicate)
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
padding_mode=replicate)
    (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
padding_mode=replicate)
    (7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (classification_model): Sequential(
    (0): Dropout(p=0.2, inplace=False)
    (1): Linear(in_features=2048, out_features=512, bias=True)
    (2): ReLU()
    (3): Dropout(p=0.2, inplace=False)
    (4): Linear(in_features=512, out_features=64, bias=True)
    (5): ReLU()
    (6): Linear(in_features=64, out_features=6, bias=True)
    (7): LogSoftmax(dim=1)
  )
)
```

Training of Model:

```
Epoch 0 - Training loss: 0.6998346765031184
Epoch 1 - Training loss: 0.679092197847773
Epoch 2 - Training loss: 0.675565375003225
Epoch 3 - Training loss: 0.6734950225998853
Epoch 4 - Training loss: 0.6720934793639031
Epoch 5 - Training loss: 0.6712066595361177
Epoch 6 - Training loss: 0.6705225137378107
Epoch 7 - Training loss: 0.6697057108762168
Epoch 8 - Training loss: 0.6692271794972897
Epoch 9 - Training loss: 0.6688430201905623
Epoch 10 - Training loss: 0.6686527947627151
Epoch 11 - Training loss: 0.6685410642039293
Epoch 12 - Training loss: 0.6682214680383963
Epoch 13 - Training loss: 0.6678789160780306
Epoch 14 - Training loss: 0.6676722010696875

Training Time (in minutes) = 6.382478229204813
```

Training Loss with epoch:



Confusion Matrix:

Accuracy Achieved: 79.7%

Class wise Accuracy:

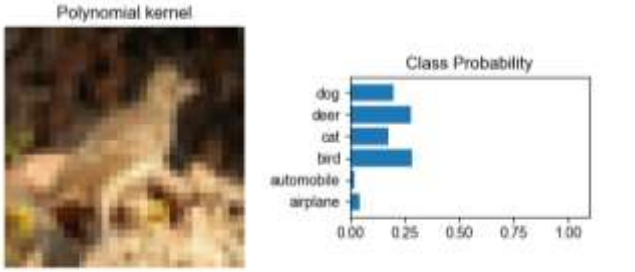
Airplane	Automobile	Bird	Cat	Deer	Dog
0.87948207	0.95325203	0.76604555	0.65714286	0.78409091	0.75027624

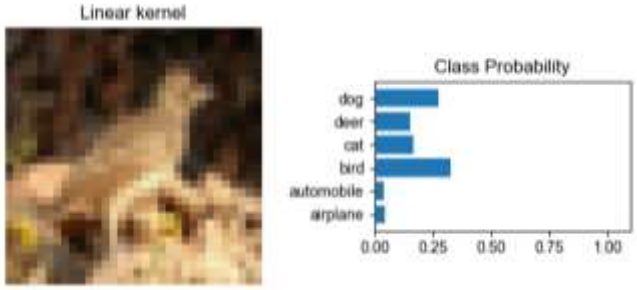
Q3. Use any non-deep learning techniques such as SVM, Naive Bayes, or Logistic Regression to perform classification and report the accuracy.

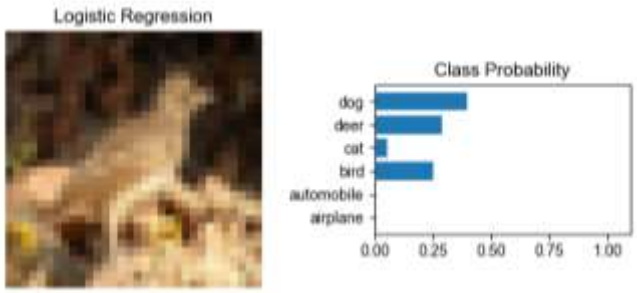
Non-Deep Learning Technics Used:

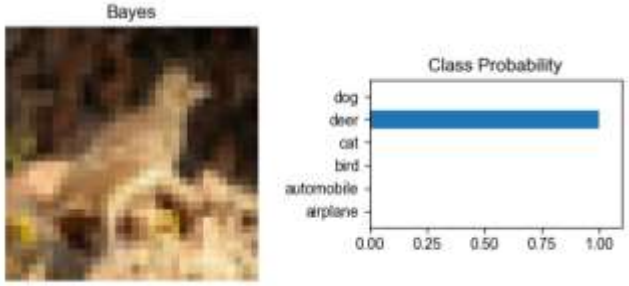
1. **SVM Linear Kernel**
2. **SVM Polynomial Kernel**
3. **Logistic Regression**
4. **Gaussian Naïve Bayes**

Sample Class probabilities from each method:



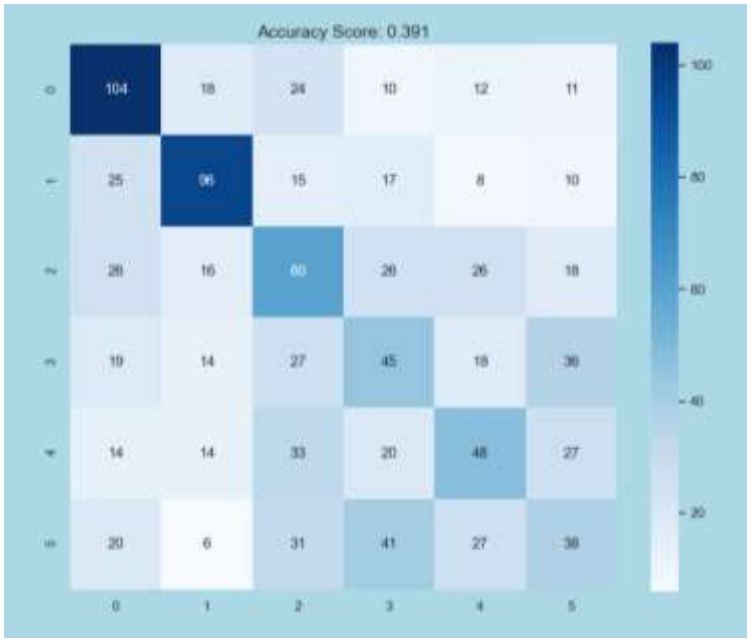






Confusion Matrix of each method:

SVM Linear Kernel:



Overall Accuracy Score: **0.391**

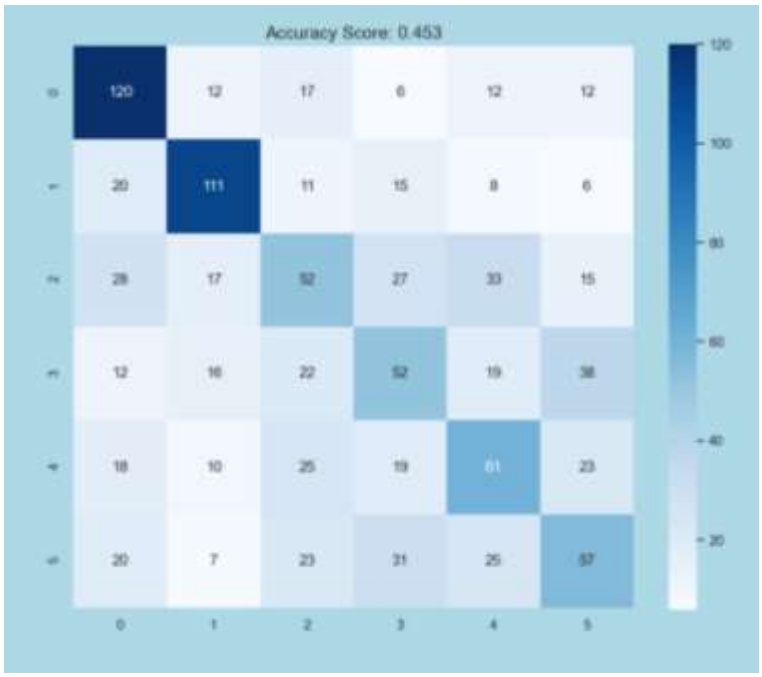
Classwise Accuracy Score: [0.58100559 0.56140351 0.34883721 0.28301887 0.30769231 0.23312883]

SVM Polynomial Kernel:

Overall Accuracy Score: **0.497**

Classwise Accuracy Score: [0.6424581 0.7251462 0.43604651 0.34591195 0.47435897 0.33128834]

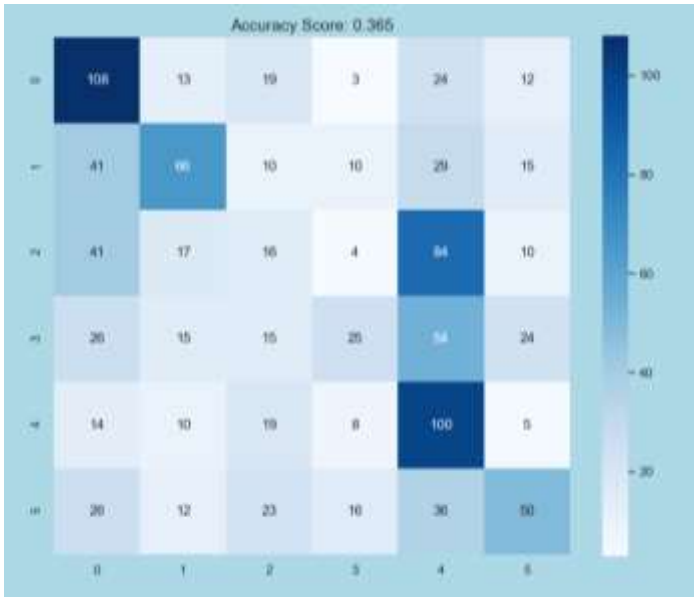
Logistic Regression:



Overall Accuracy Score: **0.453**

Classwise Accuracy Score: [0.67039106 0.64912281 0.30232558 0.32704403 0.39102564 0.34969325]

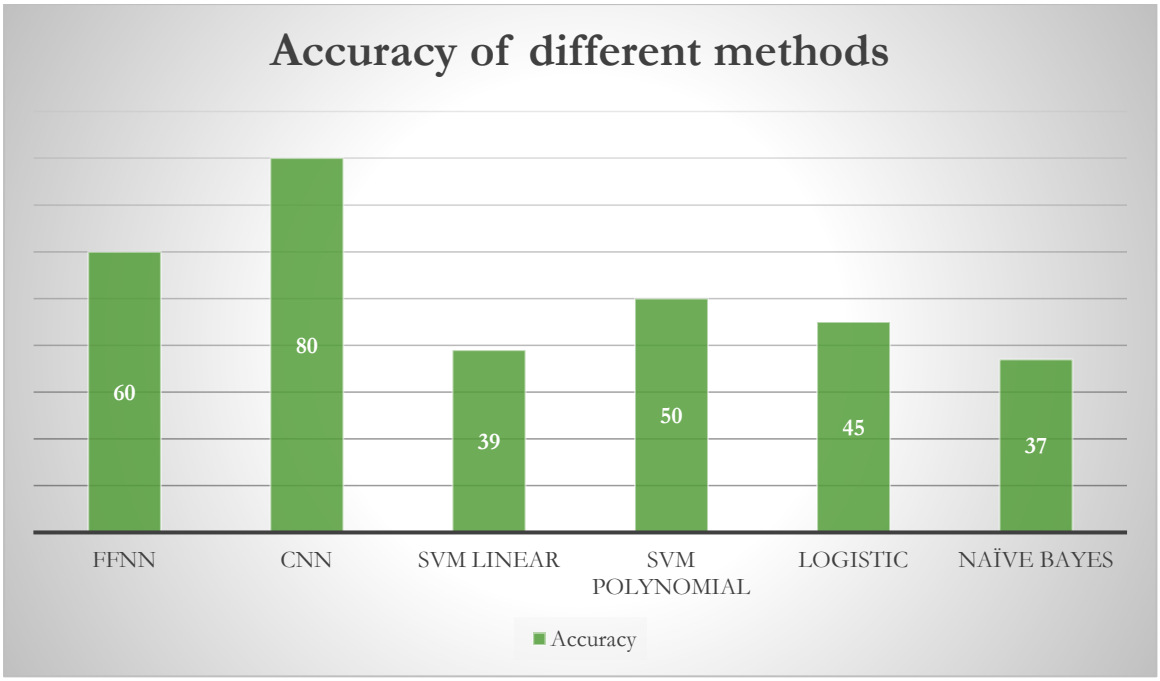
Gaussian Naïve Bayes:



Overall Accuracy Score: **0.365**

Classwise Accuracy Score: [0.60335196 0.38596491 0.09302326 0.1572327 0.64102564 0.30674847]

Q4. Compare and discuss 1, 2, and 3 with respect to accuracy

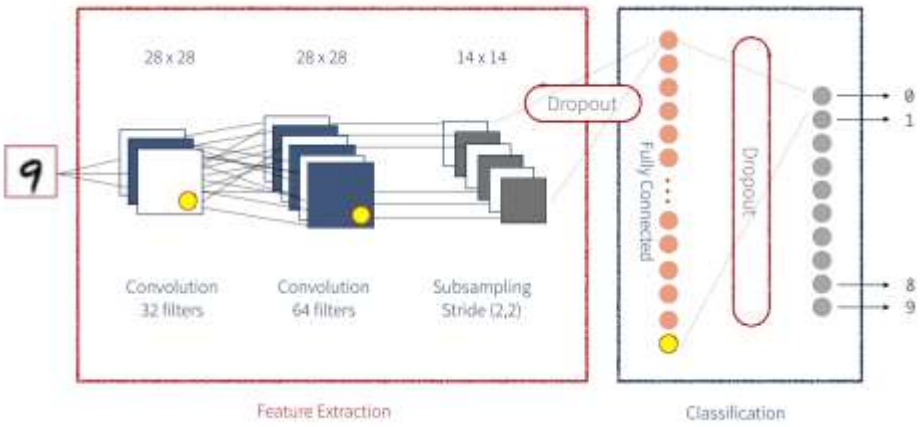


Comparing above results, primary observation is that Accuracy (CNN)> Accuracy (FFNN) > Accuracy (Other Non-Deep Learning Algorithms).

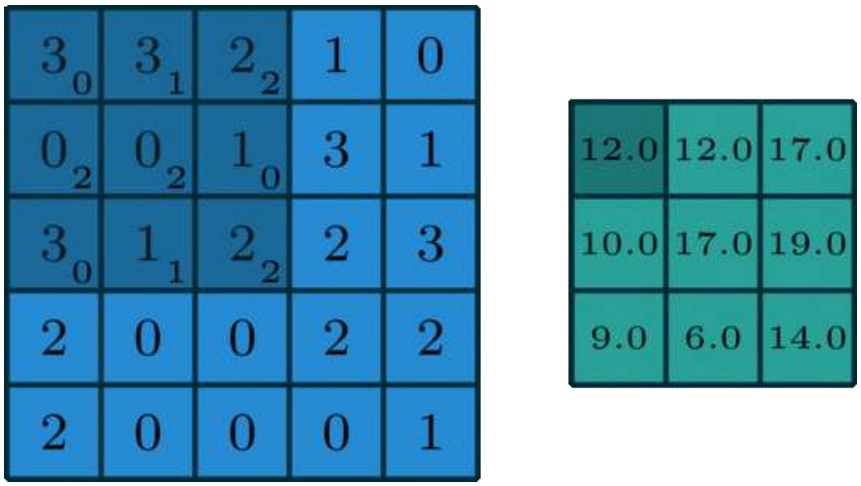
Artificial Neural Network is capable of learning any nonlinear function. ANNs have the capacity to learn weights that map any input to the output. One of the main reasons behind universal approximation is the activation function. Activation functions introduce nonlinear properties to the network. This helps the network learn any complex relationship between input and output. That is why FFNN and CNN outperform other Non-Deep Learning Algorithms.

Advantages of Convolution Neural Network (CNN) over Feed Forward Neural Network:

- CNN learns the filters automatically without mentioning it explicitly. These filters help in extracting the right and relevant features from the input data
- CNN does automatic **Feature Extraction** with the help of



- CNN captures the **spatial features** from an image. Spatial features refer to the arrangement of pixels and the relationship between them in an image. They help us in identifying the object accurately, the location of an object, as well as its relation with other objects in an image



- CNN also follows the concept of parameter sharing. A single filter is applied across different parts of an input to produce a feature map:

References:

1. Lectures of ML class by Prof Mayank Vatsa
2. https://towardsdatascience.com/handwritten-digit-mnist-pytorch-977b5338e627
1. https://github.com/L1aoXingyu/pytorch-beginner/tree/master/08-AutoEncoder
2. https://www.analyticsvidhya.com/blog/2017/01/t-sne-implementation-r-python/
3. https://www.analyticsvidhya.com/blog/2018/08/dimensionality-reduction-techniques-python/
4. https://towardsdatascience.com/visualizing-clusters-with-pythons-matplolib-35ae03d87489
5. https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798