Deep Learning
Assignment 2

1. Convolutional Neural Network -

We wrote our own version of the AlexNet layer by layer. This is because the network was simple enough to code and won the 2012 Image Net challenge. Our hypothesis was that it should do a good job on the Beans classification dataset.

Following is our model:

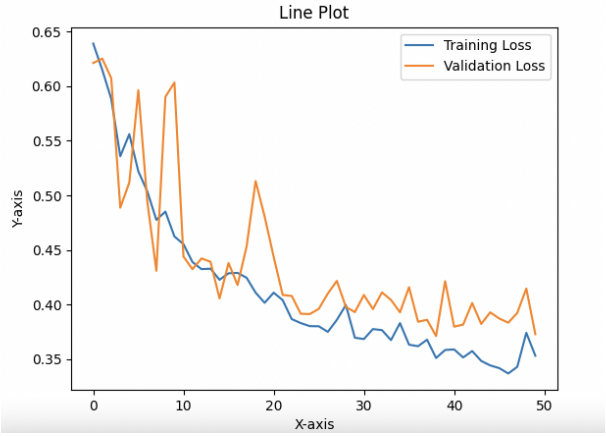
```
AlexNet(
  (head): Sequential(
    (0): Conv2d(3, 96, kernel_size=(11, 11), stride=(4, 4))
    (1): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (2): ReLU(inplace=True)
    (3): Conv2d(96, 256, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): ReLU(inplace=True)
    (6): Conv2d(256, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2))
    (13): ReLU(inplace=True)
  )
  (ann): Sequential(
    (0): Linear(in_features=9216, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Linear(in_features=4096, out_features=4096, bias=True)
    (3): ReLU(inplace=True)
    (4): Linear(in_features=4096, out_features=2, bias=True)
  )
  (classifer): Sequential(
    (0): Sigmoid()
  )
)
```

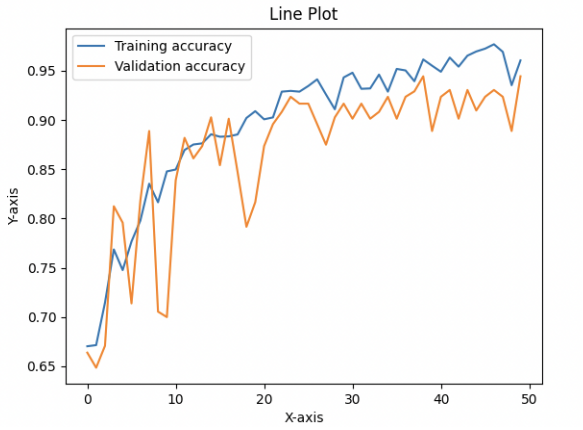
Hyperparameters
Learning rate: 0.01
Batch-Size: 16
Num_epochs: 50
Optimizer: Stochastic Gradient Descent
Loss: Cross Entropy Loss
Layers: 6 Conv layers, 2 max pool Layers, 3 linear layers, 8 ReLU's and 1 final Sigmoid layer

We have saved our results for the blind test dataset as "blind_set_results.csv" along with the codebase

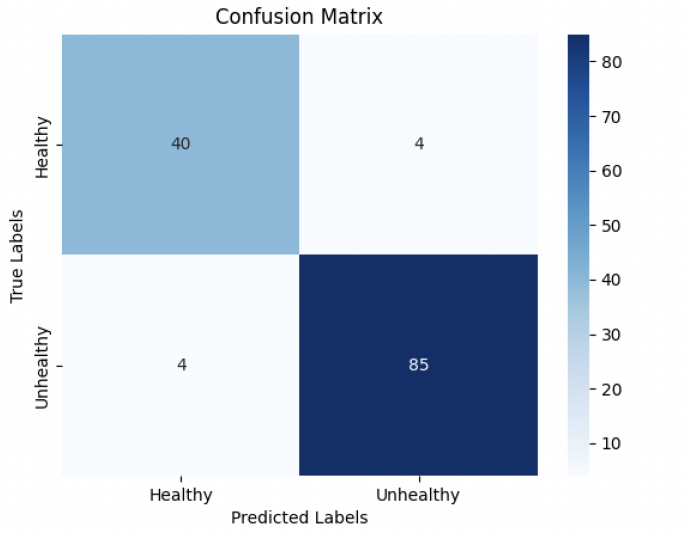
Learning curves:

Learning Curves(Loss and Accuracy):





Confusion matrix for the validation set:



Limitations of our approach:

1. Trained the model from scratch. Using pretrained weights might give better results after fine tuning on our dataset
2. We did not do any data augmentation because our baseline model was giving decent results but augmentation could have improved the performance further.

2. Autoencoder Network -

The implemented autoencoder network is a CNN-based model with an encoder that compresses the input data into a lower-dimensional representation, a decoder that reconstructs the original input from the compressed representation, and a feedforward network that classifies images based on the encoding.

Encoder:

4 convolutional layers with 4 ReLU activation and 1 batch normalization layer
Flattening layer followed by 2 linear layers with ReLU activation

Decoder:

4 convolutional layers with 4 ReLU activation and 1 batch normalization layer
Unflattening layer followed by 2 linear layers with ReLU activation

Classifier:

Consists of the encoder layer, followed by 3 linear layers with ReLU activation
Classifies images based on their encoding into 2 classes

Hyperparameters

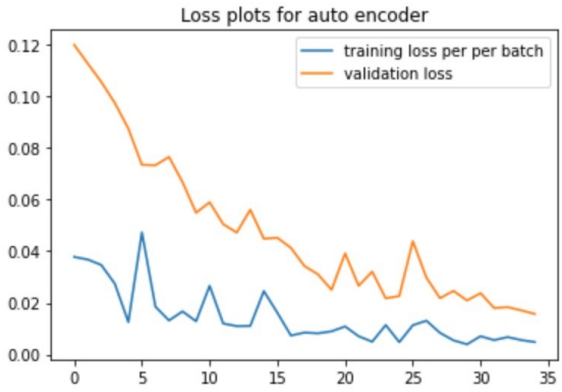
Autoencoder (Encoder and Decoder)	Classifier
Learning rate: 0.01	Learning rate: 0.001
Batch-Size: 4 each	Batch-Size: 16
Num_epochs: 7	Num_epochs: 30
Optimizer: Adam	Optimizer: SGD
Weight_decay: 1e-05	Momentum: 0.9
Loss: MSE	Loss: CrossEntropyLoss
Layers: 4 Conv layers, 2 linear layers, 5 ReLU's each for both encoder and decoder.	Layers: encoder layer, followed by 3 linear layers with ReLU activation.

We have saved our results for the blind test dataset as "blind_results.csv" along with the codebase

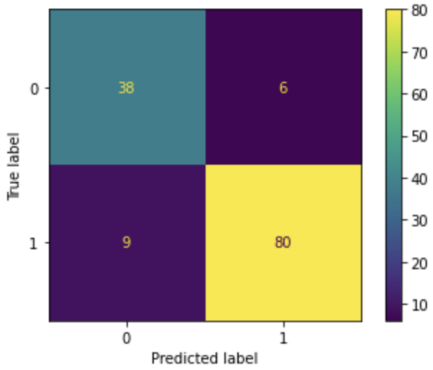
Learning curves:

Classifier Learning Curves(Loss):

Auto-Encoder Learning Curves(Loss):



Confusion matrix for the validation set:



Limitations of our approach:

1. Dependency on input data distribution: The autoencoder is sensitive to the distribution of the input data, which means that it may not work well on datasets that have a different distribution than the training dataset.
2. Limited generalization ability: The autoencoder is trained to reconstruct the input data, which can result in overfitting to the training data and limited generalization ability to new and unseen data.

3. Other Methods -

We can try various approaches if we have a small number of labeled samples for image classification. Following are some of the strategies:

1. **Data Augmentation**: We can augment data to generate more training samples. Various data augmentation techniques like rotation, flip, warping etc can be used.

2. **Transfer Learning**: We can download pre-trained model weights for the model of our choice and fine tune it to the classification objective with the images we have at our disposal. Here is one example: https://pubs.rsna.org/doi/full/10.1148/radiol.212482?casa_token=EC5Lrcts3rQAAAAA%3AbywZflhiyIvYuiAE593mqAMdvAvK-r-daluOCd8pInXxgIe4SVTQHzbG_mt9ZG0Sty1OQKea6Jbe

3. **Using Shallow Networks**: It's good to use models with less parameters if we have less number of labeled samples at our disposal. It is so that we can avoid overfitting of the dataset

4. **Active Learning**: Active Learning can be used when we have a limited amount of labeled data. The algorithm selects the most informative samples to label and trains the model on these labeled samples. This helps to optimize the use of labeled samples.

5. **Ensemble Methods**: Another approach that can be effective for image classification with a small number of labeled samples is ensemble methods. Ensemble methods combine multiple models to improve the classification accuracy, reduce overfitting, and increase the model's robustness to noise in the data.