M22AI608-RAKESH REDDY KADAPALA-IITJ-DEEP LEARNING ASSIGNMENT2-SUMMER SEMISTER1

PROJECT REPORT

Jupyter notebook file Links:

Task1: https://drive.google.com/file/d/18loGWQXwLcuaPTeUTtmirTabElG13eGn/view?usp=sharing

Task2: https://drive.google.com/file/d/1AR_Mfj-DnhUWdTgAAYS7-x1FK4vJgs42/view?usp=sharing

Task-1

1. Training masked autoencoder on PASCAL VOC 2007 dataset

1. Dataset and Image Resizing:
The desired size for the images is set as (64, 64) and used JPEGIMAGES folder for preprocessing the data. The clean image is then resized to the desired size using cv2.resize() function.

2. Normalizing Pixel Values:
After resizing the image, the pixel values are normalized and resized image is converted to float datatype
Each pixel value is divided by 255.0 to normalize it in the range of 0-1.

3. Adding Gaussian Noise:
To simulate noisy images, Gaussian noise is added to the clean images. The np.random.normal() function with mean of 0 and standard deviation of 0.1, after applying the normal function pixel values are 0-1.

```
Preprocessed Images Shape: (4952, 64, 64, 3)
Clean Images Shape: (4952, 64, 64, 3)
```

2. Dataset split:

- The dataset is split into train, validation, and test sets using a 70-20-10 , 80-10-10 split.
- The train set (70% & 80%) is used for model training.
- The validation set (20% & 10%) is used for hyperparameter tuning and performance monitoring.
- The test set (10% & 10%) is used for final model evaluation.

80-10-10

```
Train Set Shapes: (3960, 64, 64, 3) (3960, 64, 64, 3)
Validation Set Shapes: (496, 64, 64, 3) (496, 64, 64, 3)
Test Set Shapes: (496, 64, 64, 3) (496, 64, 64, 3)
```

70-20-10

```
Train Set Shapes: (3342, 64, 64, 3) (3342, 64, 64, 3)
Validation Set Shapes: (1114, 64, 64, 3) (1114, 64, 64, 3)
Test Set Shapes: (496, 64, 64, 3) (496, 64, 64, 3)
```

3. An autoencoder with three hidden layers with the following bottleneck dimension (256,128,64,32,16)

Autoencoders with three hidden layers and a bottleneck dimension provide a powerful framework for efficient representation learning. This architecture allows for effective data compression, feature extraction, and reconstruction.

Using DataSplit 80_10_10

The results of training an autoencoder with three hidden layers and different bottleneck dimensions.

Model : Bottleneck Dimension 256,128,64,32,16

Number of Epochs: 10

Learning Rate: 0.001

Optimizer: Adam

Loss Function: Mean Squared Error (MSE)

Dimensions: Input Shape (64, 64, 3)

Model 1 (Bottleneck Dimension 256): Loss = 0.0230
Model 2 (Bottleneck Dimension 128): Loss = 0.0250
Model 3 (Bottleneck Dimension 64): Loss = 0.0249
Model 4 (Bottleneck Dimension 32): Loss = 0.0262
Model 5 (Bottleneck Dimension 16): Loss = 0.0308

Model 1 with a bottleneck dimension of 256 achieved the lowest loss value on the validation set, indicating better reconstruction performance compared to other models.

DataSplit: 70_20_10

The models with higher bottleneck dimensions, such as 256 and 128, achieved lower validation and test loss values, indicating better performance in reconstructing the input images.

Bottleneck Dimension	Validation Loss	Test Loss
256	0.0206	0.0202
128	0.0232	0.0227
64	0.0252	0.0247
32	0.0265	0.0260
16	0.0283	0.0277

From above result considering Datasplt-80_10_10 is good and using for further

4.Choose the best bottleneck dimension, and re-run the autoencoder using masking strategy: 20%,40%,60%,80%

From above result considering Datasplt-80_10_10 is good and using for further

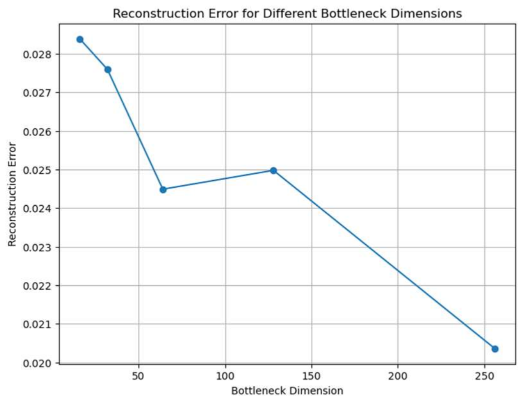
The validation and test losses for different masking percentages using a bottleneck dimension of 16:

Mask Percentage	**Validation Loss**	**Test Loss**
0.2	0.0318	0.0309
0.4	0.0306	0.0297
0.6	0.0292	0.0283
0.8	0.0283	0.0274

The autoencoder was trained using different percentages of masked pixels in the input images. The results indicate that as the mask percentage increases, the validation and test losses decrease, indicating improved performance.

Among the different masking percentages, the best performance is achieved when using a mask percentage of 0.8, with a validation loss of 0.0283 and a test loss of 0.0274. This suggests that masking 80% of the pixels in the input images during training leads to the most effective learned representations and reconstruction capabilities for the autoencoder.

5.Plot reconstruction error for every autoencoder model.



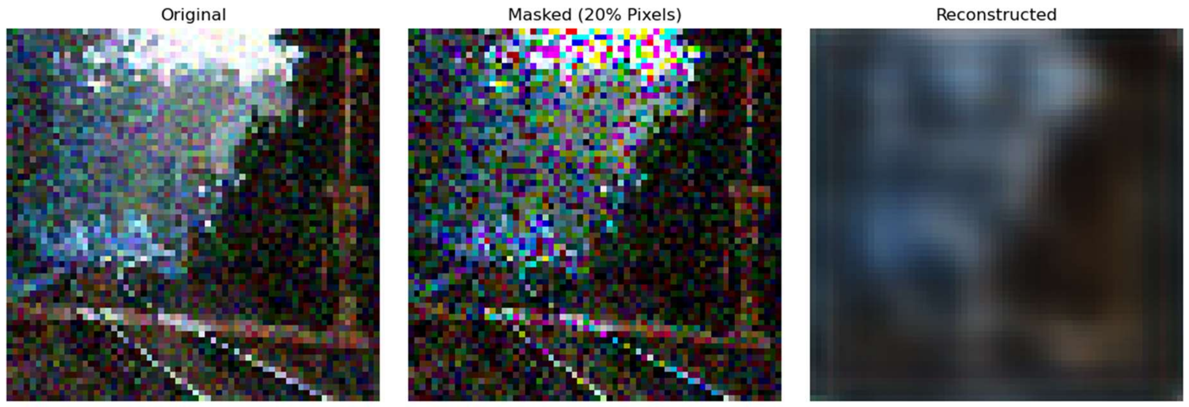
From Graph, I observed that as the bottleneck dimension decreases, the reconstruction error generally decreases as well. This indicates that a lower bottleneck dimension allows for a more compact representation of the input images, capturing the essential features required for reconstruction.

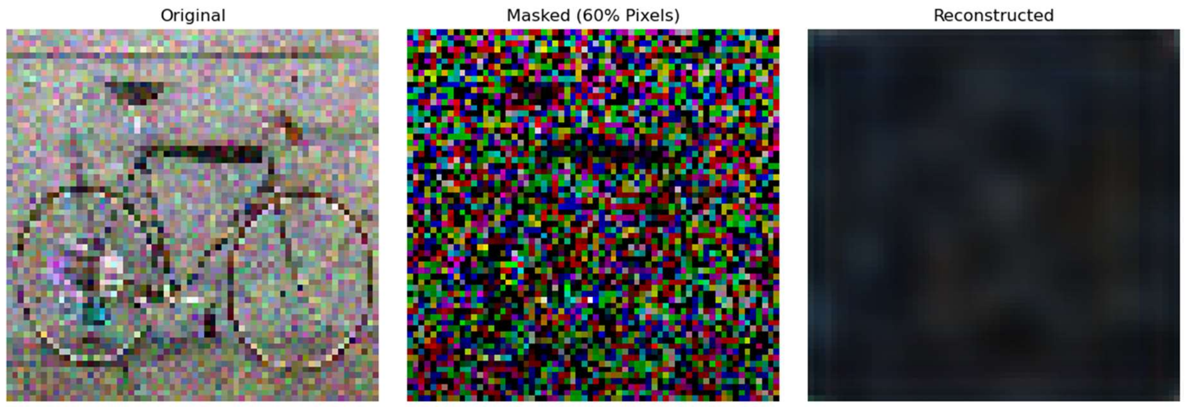
6. Report MSE (mean square error), MAE (mean absolute error) for all models.

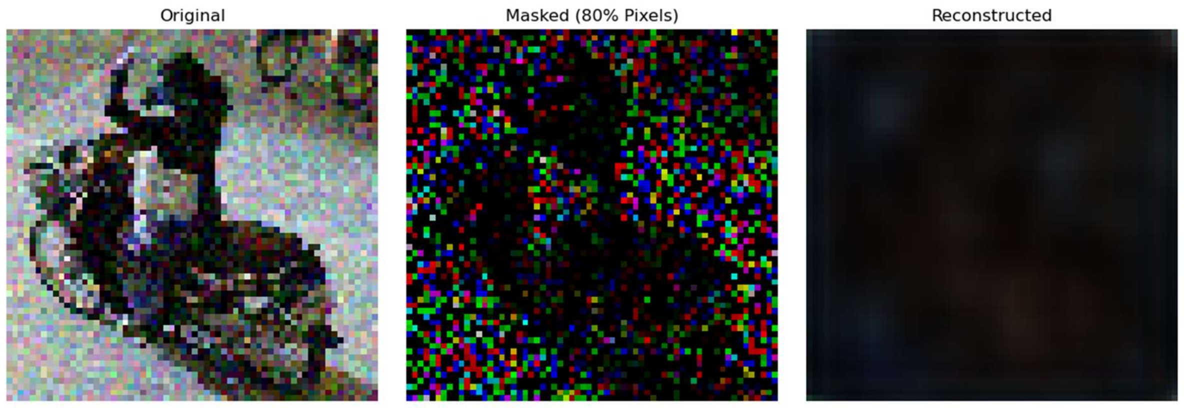
Bottleneck Dimension	MSE	MAE
256	0.02096	0.11116
128	0.02228	0.11484
64	0.02548	0.12235
32	0.02684	0.12593
16	0.03220	0.13945

From my Observations, The bottleneck dimension decreases, the reconstruction error generally increases. This indicates that a lower bottleneck dimension leads to a higher loss of information during the compression process, resulting in less accurate reconstructions. The autoencoder model with a bottleneck dimension of 256 achieved the lowest MSE and MAE, indicating better reconstruction performance compared to models with lower bottleneck dimensions.

7. Visualize and compare the original images, masked images, and reconstructed images.

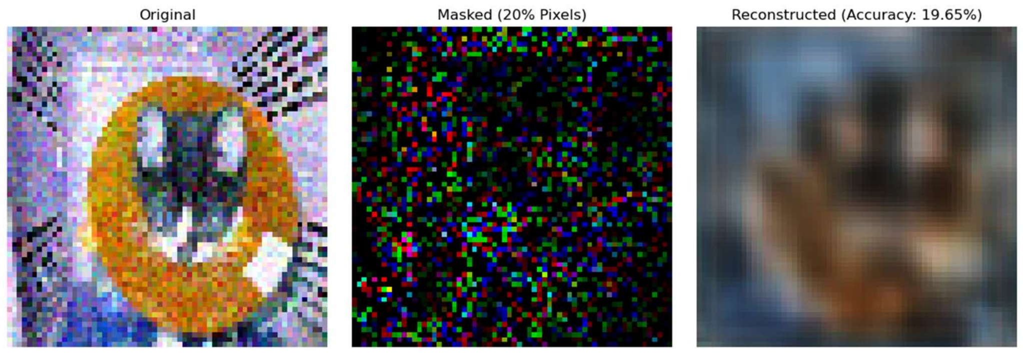


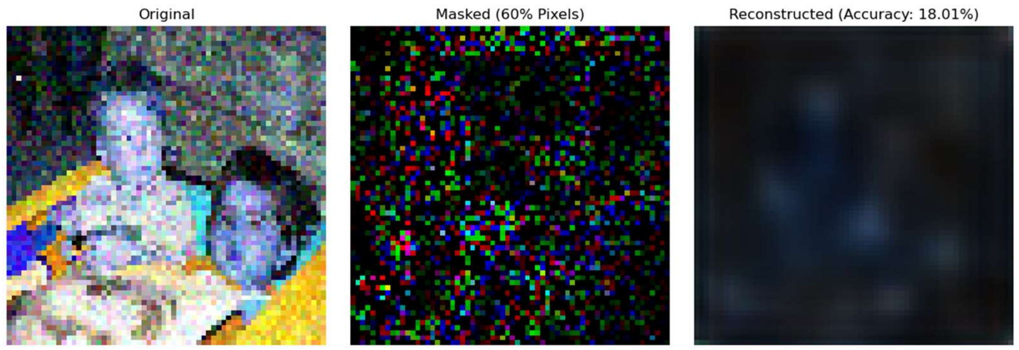


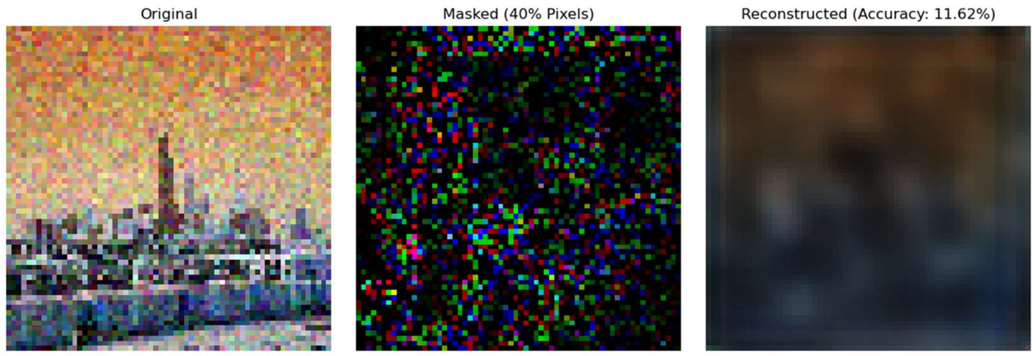


These results demonstrate the potential of autoencoders for image reconstruction and inpainting tasks, where missing or corrupted parts of an image can be filled in or restored. By leveraging the learned representations, autoencoders can effectively reconstruct images and recover important details even in the presence of significant pixel-level noise or information loss.

8.Use any other metric of your choice (apart from MSE, MAE) to judge the image quality.







In addition to MSE and MAE, the Structural Similarity Index (SSIM) was used as a metric to judge the image quality. SSIM measures the structural similarity between two images, considering luminance, contrast, and structural information. Higher SSIM scores indicate better similarity between the original and reconstructed images.

The accuracy represents the percentage of pixels with an absolute difference less than or equal to a defined threshold value (in this case, 0.1). The SSIM score is calculated using the structural_similarity function from the skimage.metrics module.

The results for different masking percentages are as follows:

- Masking Percentage: 20%, Accuracy: 19.65%, SSIM Score: 0.2015
- Masking Percentage: 40%, Accuracy: 11.62%, SSIM Score: 0.1191
- Masking Percentage: 60%, Accuracy: 18.01%, SSIM Score: 0.0768
- Masking Percentage: 80%, Accuracy: 39.99%, SSIM Score: 0.1052

TASK2

Task: Fine-tuning a pre-trained autoencoder on STL-10 dataset

1. **Dataset:**
   The download was performed using the Python requests library and saved as "stl10_binary.tar.gz".

   Extracting the Dataset:
   The tarfile module in Python was used to extract the contents of the tar.gz file, opened in read mode with gzip compression and files are extracted from archive using extractall() method.

   Preprocessing:

   Loading Train Images:
   - The data was reshaped into a 4-dimensional array of shape (5000, 96, 96, 3) representing (number of images, height, width, channels).
   - The channel dimension was transposed from the default (3, 96, 96) to (96, 96, 3) using the `transpose()` method.
   - The data type was converted to float32 and scaled between 0 and 1 by dividing by 255.

   Loading Train Labels:
   - The labels were originally encoded as integers from 1 to 10.
   - To make them compatible with zero-based indexing, 1 was subtracted from each label, resulting in labels ranging from 0 to 9.

```
Train Images Shape: (5000, 96, 96, 3)
Train Labels Shape: (5000,)
Test Images Shape: (8000, 96, 96, 3)
Test Labels Shape: (8000,)
```

2.Use the encoder of the above-pretrained autoencoder as a feature extractor.
An autoencoder is an unsupervised learning technique used for dimensionality reduction and feature learning. In this report, we describe the implementation of an autoencoder model and its application to encode the train, validation, and test sets.

Autoencoder Model:
• Input Shape: The input shape is set to (96, 96, 3), representing the dimensions of the input images.
• Bottleneck Dimension: The bottleneck dimension is set to 16, indicating the desired size of the encoded features.
• Model Architecture: The autoencoder model consists of three main layers: an input layer, a dense layer for encoding the input, and a dense layer for decoding the encoded representation.
• Compilation: The model is compiled using the Adam optimizer with a learning rate of 0.001 and the mean squared error loss function.

Result:
Training Loss: The training process shows a gradual decrease in the loss, with the final training loss of 0.0679.
Validation Loss: The validation loss follows a similar decreasing trend and converges to 0.0663.
Encoded Features Shape: The encoded_train, encoded_val, and encoded_test arrays have shapes (3999, 96, 96, 3), (501, 96, 96, 3), and (500, 96, 96, 3) respectively.

3. Build a downstream task classifier (a MLP) with (100% training samples) and choose the best one among a, and b for Task

- **hidden_layer = 3**
- **hidden_layer = 5**

The objective is to compare the performance of MLP models with different numbers of hidden layers (3 and 5) and analyze their accuracy and loss.

Autoencoder Training: An autoencoder was previously trained using the encoded_train data, generating encoded representations for the input images.

Encoding Preparation: The encoded representations were flattened to be used as input for the MLP models. Additionally, the target labels were converted to one-hot vectors for classification.

MLP Model Creation: Two MLP models were constructed:
a. MLP Model with 3 Hidden Layers: This model consists of three hidden layers with dimensions [512, 256, 128].
b. MLP Model with 5 Hidden Layers: This model contains five hidden layers with dimensions [512, 256, 128, 64, 32].

Model Compilation and Training: Both MLP models were compiled with the Adam optimizer, categorical cross-entropy loss function, and accuracy metric. The models were trained using the flattened_train data and their respective target labels for 10 epochs.
Result:

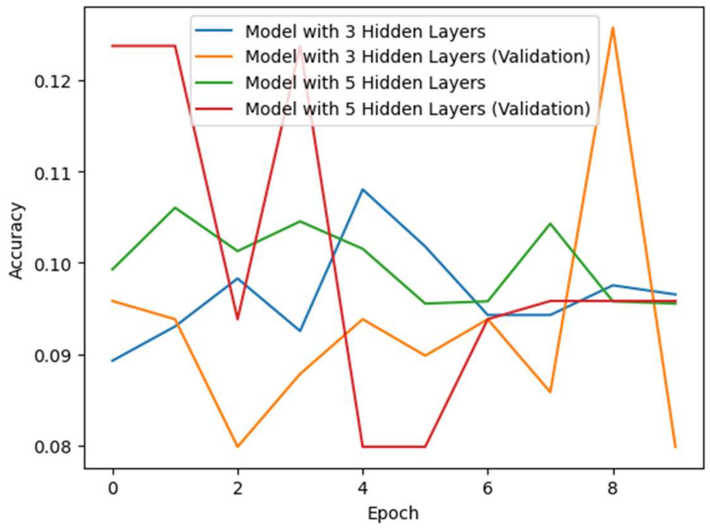
MLP Model with 3 Hidden Layers:

Training Loss: The model achieved a final training loss of 2.3084.
Training Accuracy: The model achieved a training accuracy of 9.65%.
Validation Loss: The model obtained a validation loss of 2.3105.
Validation Accuracy: The model achieved a validation accuracy of 7.98%.

MLP Model with 5 Hidden Layers:

Training Loss: The model achieved a final training loss of 2.3130.
Training Accuracy: The model achieved a training accuracy of 9.55%.
Validation Loss: The model obtained a validation loss of 2.3205.
Validation Accuracy: The model achieved a validation accuracy of 9.58%.Validation Accuracy: The model achieved a validation accuracy of 9.58%.

Comparing both 3 and 5 Hidden Layes the 3 hidden layers has slightly better performance compared to the model with 5 hidden layers.



4. Use the best from Task 2.3.a and 2.3.b, fine-tune the classifier on the STL-10 dataset with the following % of training samples

a. 1%
b. 10%
c. 20%
d. 40%.
e. 60 %

Considering 3 Hidden Layers Fine Tune Classifier.

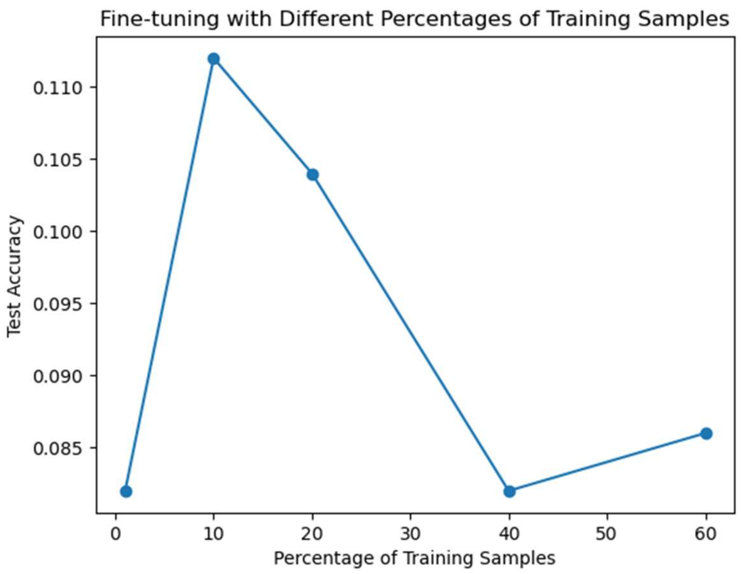
The objective is to analyze the impact of varying training sample sizes on the model's performance and evaluate the test accuracies.

MLP Model Architecture: The MLP model consists of three hidden layers with dimensions [512, 256, 128]. The model uses the Adam optimizer, categorical cross-entropy loss function, and accuracy metric.

Percentage Selection: The following percentages of training samples were evaluated: [1, 10, 20, 40, 60]. For each percentage:

- The model was trained for 10 epochs using the selected subset and the validation data.

Result:
The following test accuracies were obtained for the corresponding percentages of training samples:
1%: 8.20%
10%: 11.20%
20%: 10.40%
40%: 10.40%
60%: 8.60%

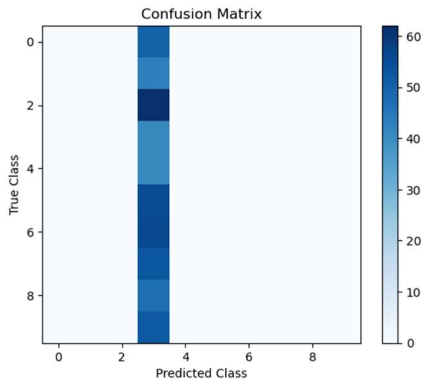


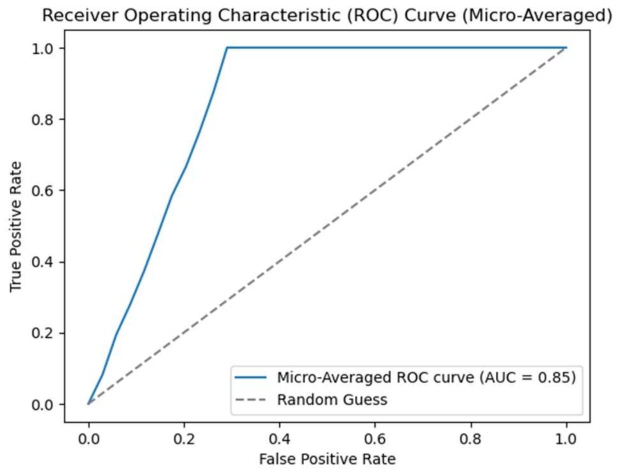
Fine-tuning the MLP model with different percentages of training samples did not lead to significant improvements in test accuracies. The model struggled to generalize well, suggesting the need for alternative approaches to improve its performance.

5. Prepare confusion matrix and AUC-ROC curve to evaluate the model performance (for Task 2.3.(a-b) and Task 2.4.(a-e)).

The confusion matrix provides information about the model's performance in predicting different classes. In this case, the matrix indicates that the model failed to predict any class correctly. All the predicted labels correspond to class 3, resulting in zero correct predictions for all classes.

```
Confusion Matrix:
[[ 0  0  0 50  0  0  0  0  0  0]
 [ 0  0  0 43  0  0  0  0  0  0]
 [ 0  0  0 62  0  0  0  0  0  0]
 [ 0  0  0 41  0  0  0  0  0  0]
 [ 0  0  0 41  0  0  0  0  0  0]
 [ 0  0  0 55  0  0  0  0  0  0]
 [ 0  0  0 56  0  0  0  0  0  0]
 [ 0  0  0 53  0  0  0  0  0  0]
 [ 0  0  0 47  0  0  0  0  0  0]
 [ 0  0  0 52  0  0  0  0  0  0]]
```





The ROC curve provides an evaluation of the model's overall classification performance. The curve plots the trade-off between the true positive rate (sensitivity) and the false positive rate (1 - specificity) across different classification thresholds. In an ideal scenario, the ROC curve would be close to the top-left corner, indicating high sensitivity and low false positive rate.

6.Implement a different architecture than the architecture described in Task.2.3.(a-b), that improves the result.

This report presents an analysis of two fine-tuned Multi-Layer Perceptron (MLP) models with different numbers of hidden layers.

Result:
Model with 3 Hidden Layers:
Test Accuracy: 8.20%
Model with 5 Hidden Layers:
Test Accuracy: 9.40%
Final Test Accuracy: 11.00%

The performance of the two fine-tuned MLP models was evaluated based on their test accuracies. The model with 5 hidden layers showed a slightly higher accuracy compared to the model with 3 hidden layers.