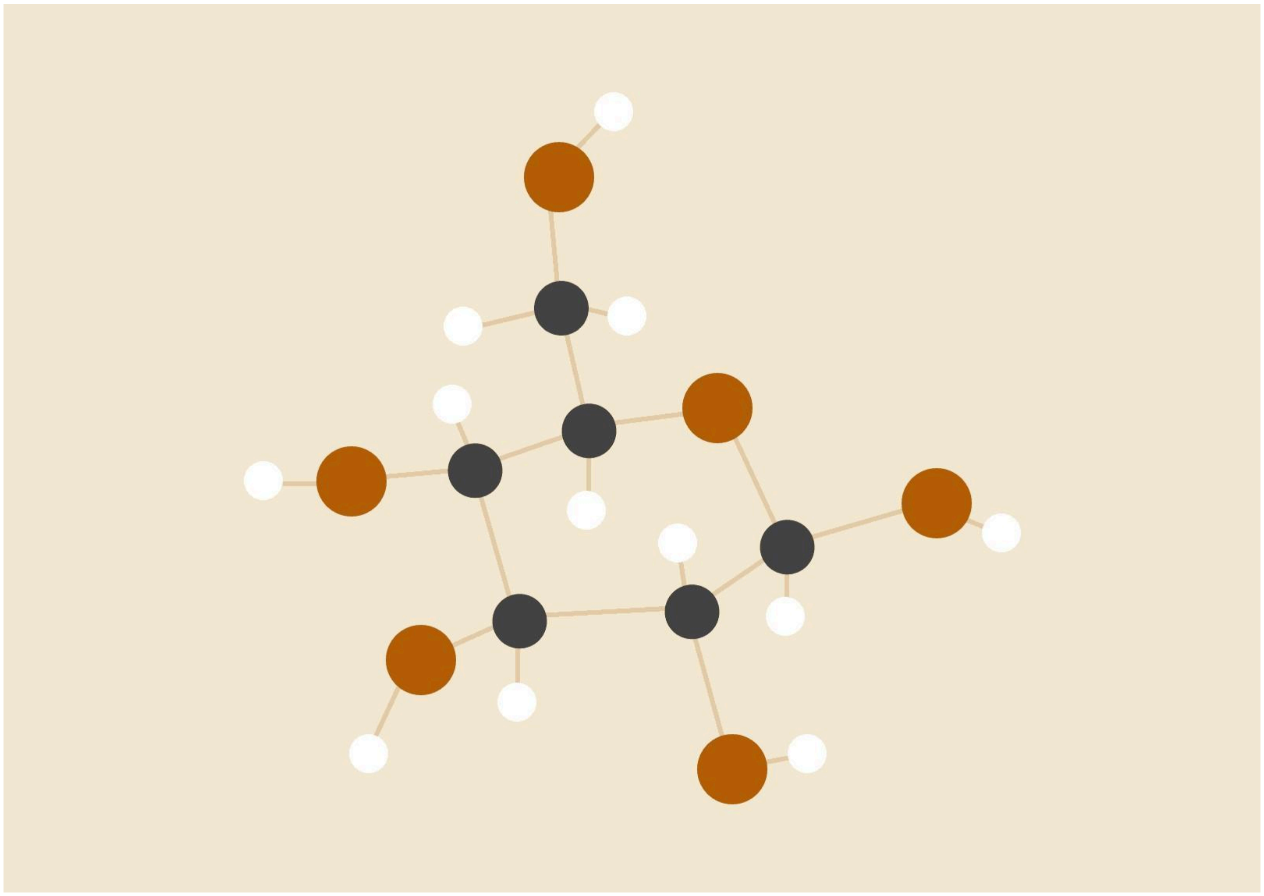
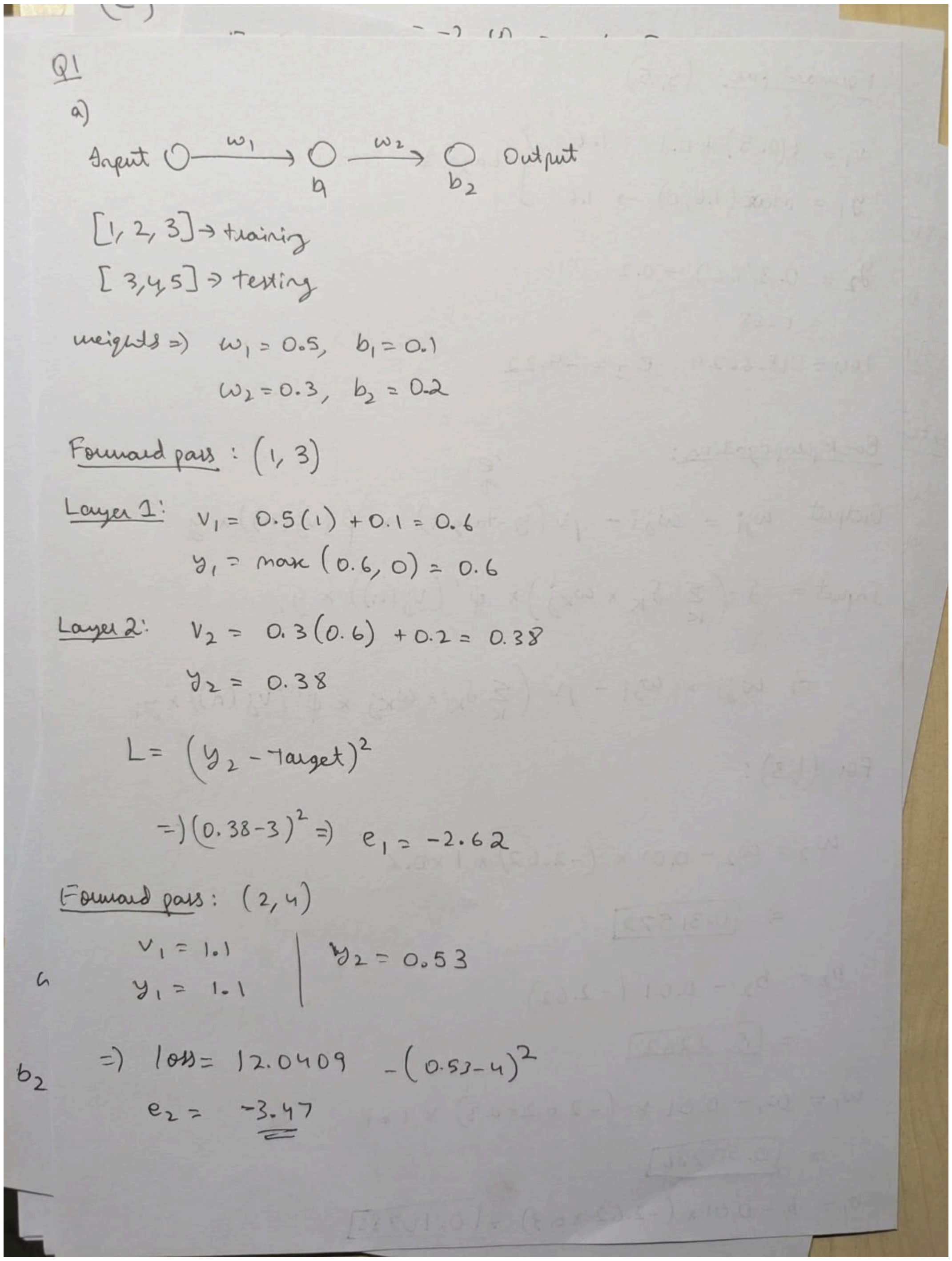
ML ASSIGNMENT 3 REPORT



Vikranth Udandarao

ROLL NUMBER - 2022570

THEORY



Q1

a)

$$\begin{tikzcd} \text{Input } \bigcirc \arrow[r, "w_1"] & \underset{b_1}{\bigcirc} \arrow[r, "w_2"] & \underset{b_2}{\bigcirc} \text{ Output} \end{tikzcd}$$

$[1, 2, 3] \rightarrow$ training

$[3, 4, 5] \Rightarrow$ testing

weights $\Rightarrow$ $w_1 = 0.5, \quad b_1 = 0.1$

$w_2 = 0.3, \quad b_2 = 0.2$

Forward pass : $(1, 3)$

Layer 1: $v_1 = 0.5(1) + 0.1 = 0.6$

$y_1 = \max(0.6, 0) = 0.6$

Layer 2: $v_2 = 0.3(0.6) + 0.2 = 0.38$

$y_2 = 0.38$

$L = (y_2 - \text{Target})^2$

$\Rightarrow (0.38 - 3)^2 \Rightarrow \quad e_1 = -2.62$

Forward pass : $(2, 4)$

$v_1 = 1.1$ | $y_2 = 0.53$

a $\quad y_1 = 1.1$ |

b_2 $\quad \Rightarrow$ loss $= 12.0409 \quad -(0.53-4)^2$

$e_2 = \underline{\underline{-3.47}}$

Forward pass: $(3,5)$

$$\left.\begin{aligned} v_1 &= 3(0.5) + 0.1 = 1.6 \\ y_1 &= \max(1.6, 0) \rightarrow 1.6 \end{aligned}\right\} \text{Layer 1}$$

$$y_2 = 0.3(1.6) + 0.2$$
$$= 0.68$$

$$\text{loss} = 18.6624, \quad e_3 = -4.32$$

Backpropogation:

$$\text{Output} \quad w_{ji} = w_{ji} - \eta\,\overset{(e)}{\overset{\uparrow}{(y\text{-target})}} \times \varphi(v_j(n)) \times y$$

$$\text{Input} = \delta = \left(\sum_K \delta_K \times w_{Kj}\right) \times \varphi'(v_j(n)) \times y_i$$

$$\Rightarrow w_{ji} = w_{ji} - \eta \left(\sum_K \delta_K \times w_{Kj} \times \varphi'(v_j(n)) \times y_i\right.$$

For $(1,3)$:

$$w_2 = w_2 - 0.01 \times (-2.62) \times 1 \times 0.6$$
$$\Rightarrow \boxed{0.31572}$$

$$b_2 = b_2 - 0.01(-2.62)$$
$$= \boxed{0.2262}$$

$$w_1 = w_1 - 0.01 \times (-2.62 \times 0.3) \times 1 \times 1$$
$$= \boxed{0.50786}$$

$$b_1 = b_1 - 0.01 \times (-2.62 \times 0.3) = \boxed{0.10786}$$

For (2,4):

$w_1 = 0.50786$

$b_1 = 0.10786$

$w_2 = 0.31572$

$b_2 = 0.7262$

$w_2 = w_2 - \eta\,(\text{output} - \text{target}) \times 1 \times y_1$

$w_2 - 0.01 \times (-3.47) \times 1 \times 1.1$

$\Rightarrow \boxed{0.35389}$

$b_2 = 0.2262 - 0.01 \times (-3.47)$

$\Rightarrow \boxed{0.2609}$

$w_1 = w_1 - \eta\,(w_2 \times \delta_2) \times y_1$

$w_1 = 0.53241$

$b_1 = 0.1201$

For (3,5)

$w_1 = 0.532 \quad , b_1 = 0.1201$

$w_2 = 0.3538, \quad b_2 = 0.2609$

$w_2 = w_2 - 0.01 \cancel{\phi} \times e \times 1 \times 1.6$

$= 0.42292$

$b_2 = 0.3041$

$-2x_1 + 5 = 0$

$w_1 = w_1 - 0.01 \times (0.42292 \times -4.32) \times 1 \times 3$

$= \boxed{0.5868}$

$b_1 = b_1 - 0.01 \times (0.42292) \times (-4.32)$

$= \boxed{0.1383}$

<u>After 1 iteration:</u>

$w_1 = 0.5868$, $b_1 = 0.1383$

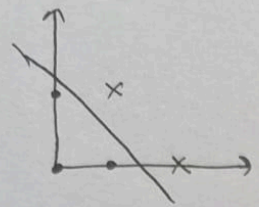
$w_2 = 0.42292$, $b_2 = 0.3041$

<u>Original:</u>

$w_1 = 0.5$, $b_1 = 0.1$

$w_2 = 0.3$, $b_2 = 0.2$

(b)



$\rightarrow$ Linearly separable

$\min \frac{1}{2}||w||^2$, $y_i(w x_i + b) \geq 1$

$L(w, b, \alpha) = \frac{1}{2}||w||^2 - \sum \alpha_i (y_i(w x_i + b) - 1)$

$\max \sum \alpha_i - \frac{1}{2} \sum_{i=1, j=1} \alpha_i \alpha_j y_i y_j x_i x_j$, $\sum \alpha_i y_i = 0$,

$\alpha_i > 0$

$+1$: $(0,0)$, $(1,0)$, $(0,1)$

-1 : $(1,1)$, $(2,2)$, $(2,0)$

Gram matrix:

~~$K_{ij} = y_i y_j x_i$~~

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & -2 & -2 \\ 0 & 0 & 1 & -1 & -2 & 0 \\ 0 & -1 & -1 & 2 & 4 & 2 \\ 0 & -2 & -2 & 4 & 8 & 4 \\ 0 & -2 & 0 & 2 & 4 & 4 \end{bmatrix}$$

$\frac{1}{2}\alpha^T P \alpha - 1^T \alpha$

$w = \sum \alpha_i y_i x_i$, points near $x_1 + x_2 = 1.5$

$(1,0)$, $(0,0)$ $(+1, 1)$ $(2,0)$

Graphically! line $\rightarrow$ $(0,1)(1,0)$ $\therefore$ $x_1 + x_2 = 1$
$(1,1)(2,0)$ $x_1 + x_2 = 2$

same $\uparrow$

$x_1 + x_2 = 1.5$

$w \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, $b = -1.5$

(c) $wx + b = 0, \quad w_1 = -2, \; w_2 = 0, \; b = 5$

$\therefore \; -2x_1 + 5 = 0$

$$S: \left.\begin{matrix} 1 \rightarrow 3 \\ 2 \rightarrow 1 \\ 3 \rightarrow -1 \\ 4 \rightarrow -3 \end{matrix}\right\} \quad y = \begin{cases} 1 \\ 1 \\ -1 \\ 1 \end{cases}$$

(a) Margin $= \dfrac{2}{||w||} = \dfrac{2}{\sqrt{2^2 + 0}} = \boxed{1}$

(b) Sample $(2, 3) = 1$ as $y(wx + b) = 1$

$\therefore$ they are support vectors

(c) $-2(1) + 0(3) + 5 \Rightarrow 3$ which is $> 0 \; \therefore$

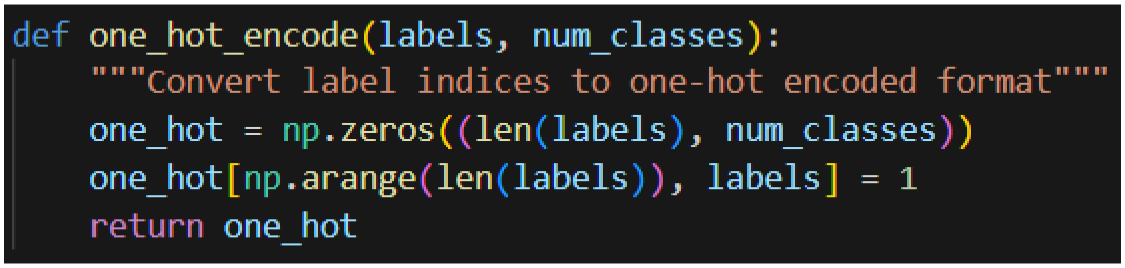
it belongs to $\boxed{+1}$ class

CODING

SEC - B:

1) Done in code
2) Done in code
3) Done in code

4) For pre-processing, I have done normalization by dividing the pixel value by 225, to get the values in a range of 0 to 1 which is better for the neural network to learn.



```python
def one_hot_encode(labels, num_classes):
    """Convert label indices to one-hot encoded format"""
    one_hot = np.zeros((len(labels), num_classes))
    one_hot[np.arange(len(labels)), labels] = 1
    return one_hot
```

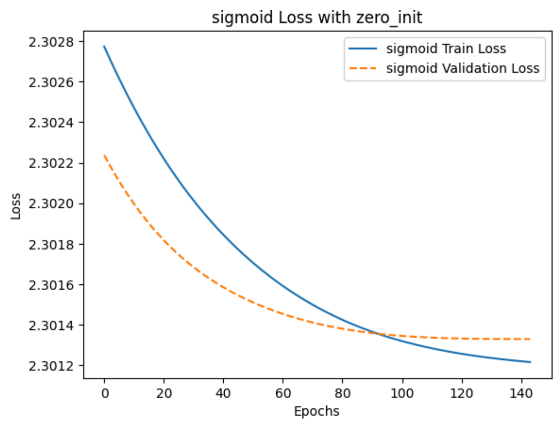
I have also used one hot encoding for multi class labels.

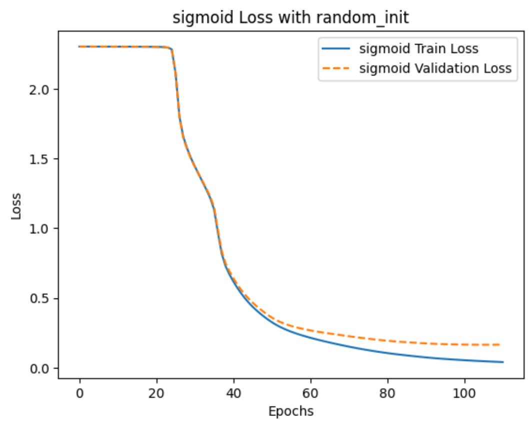
Parameters that i have taken:

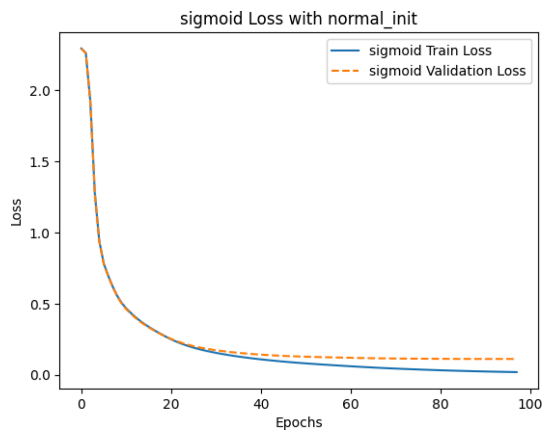
a) Number of hidden layers = 4.
b) Layer sizes = [256,128,64,32]
c) Number of epochs = 200
d) Batch size = 128
e) Learning rate = 0.1

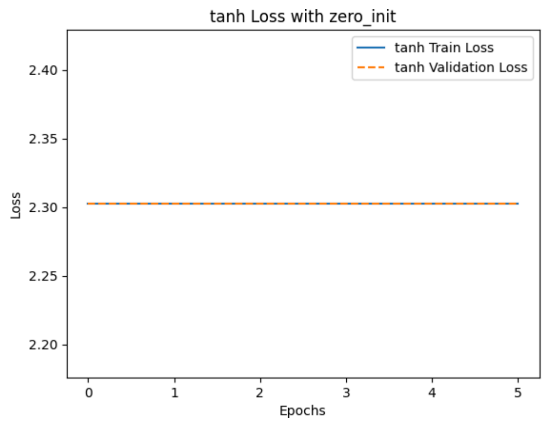
I have implemented early stopping which will stop training the model as soon as there are 5 consecutive increases in the val loss.

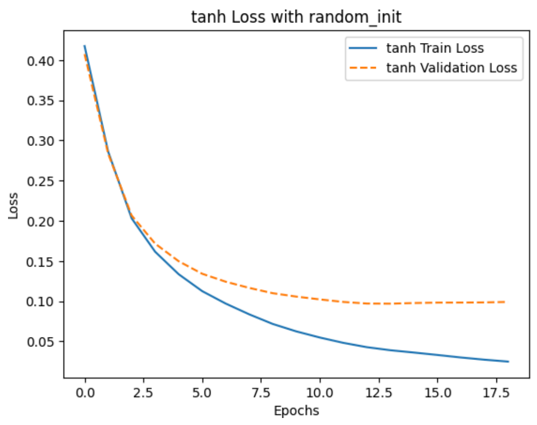
Results and Observations:

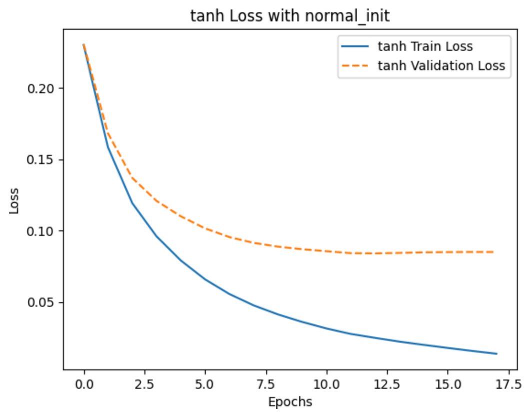

sigmoid Loss with zero_init
sigmoid Train Loss
sigmoid Validation Loss
Loss
Epochs

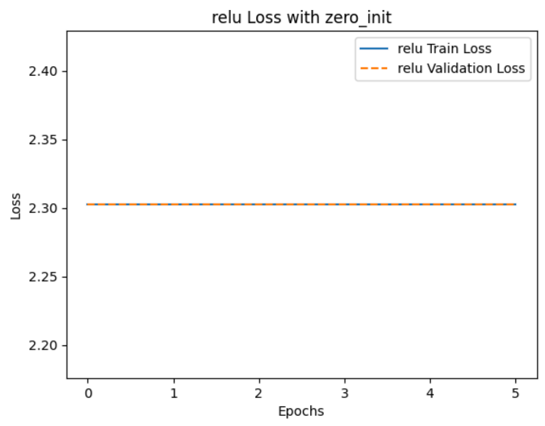

sigmoid Loss with random_init
sigmoid Train Loss
sigmoid Validation Loss
Loss
Epochs

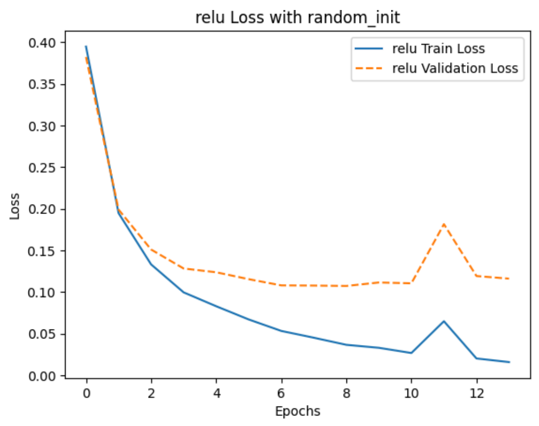

sigmoid Loss with normal_init
sigmoid Train Loss
sigmoid Validation Loss
Loss
Epochs

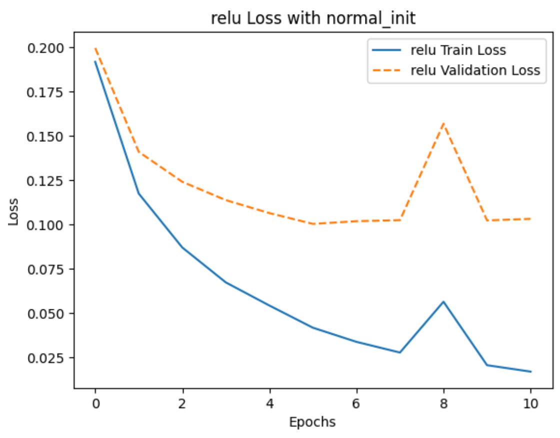

tanh Loss with zero_init
tanh Train Loss
tanh Validation Loss
Loss
Epochs

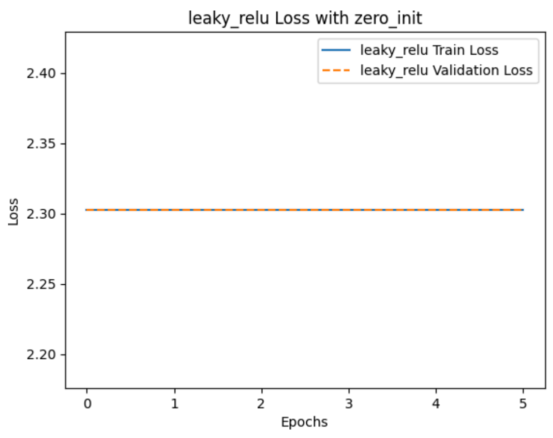

tanh Loss with random_init
tanh Train Loss
tanh Validation Loss
Loss
Epochs

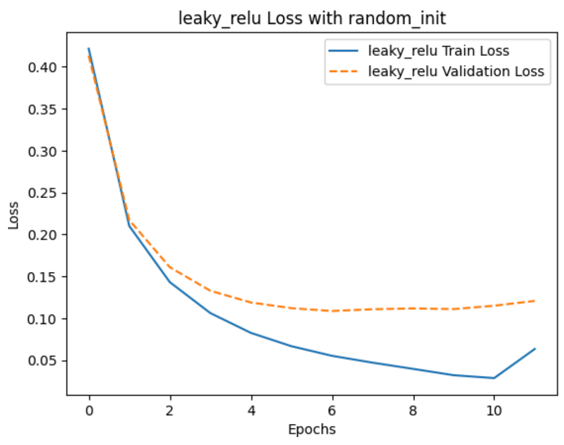

tanh Loss with normal_init
tanh Train Loss
tanh Validation Loss
Loss
Epochs

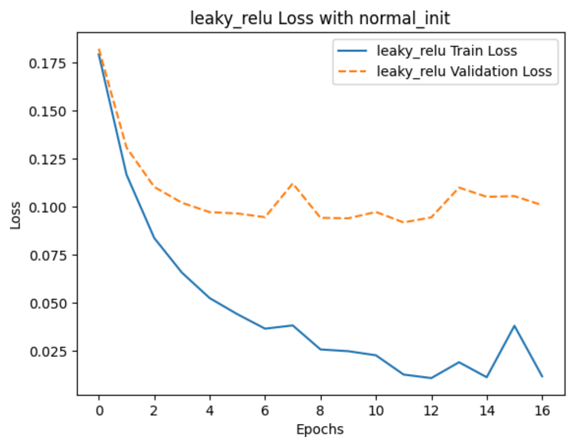












The images illustrate the training and validation loss curves for different loss functions under various initialization schemes.

Observations:

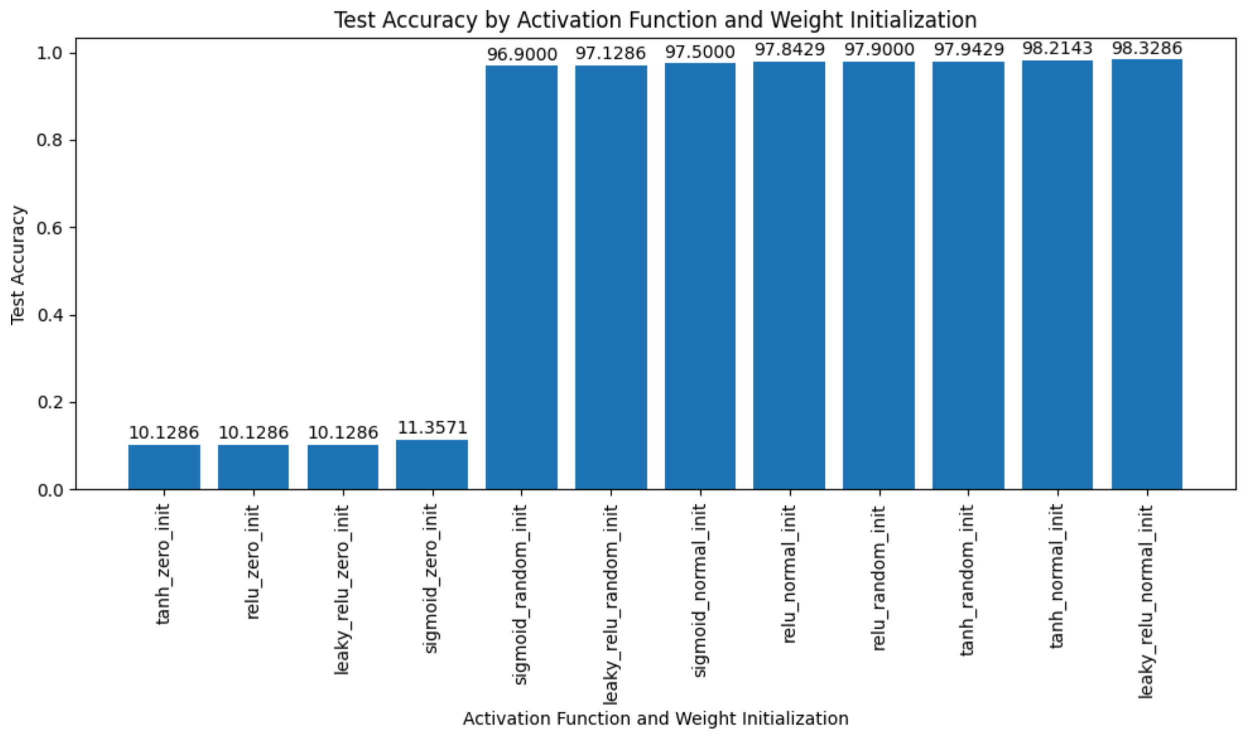
Sigmoid and Tanh: Both loss functions struggle with vanishing gradients, leading to slow convergence and unstable training. This is particularly evident with zero initialization.

ReLU: Shows improved performance over sigmoid and tanh, especially with random and normal initialization. However, it can suffer from the "dying ReLU" problem.

Leaky ReLU: Generally outperforms ReLU, mitigating the "dying ReLU" issue and leading to faster convergence and better generalization.

Overall, Leaky ReLU with random or normal initialization appears to be the most effective combination for training neural networks in these scenarios.

Model Performance Summary

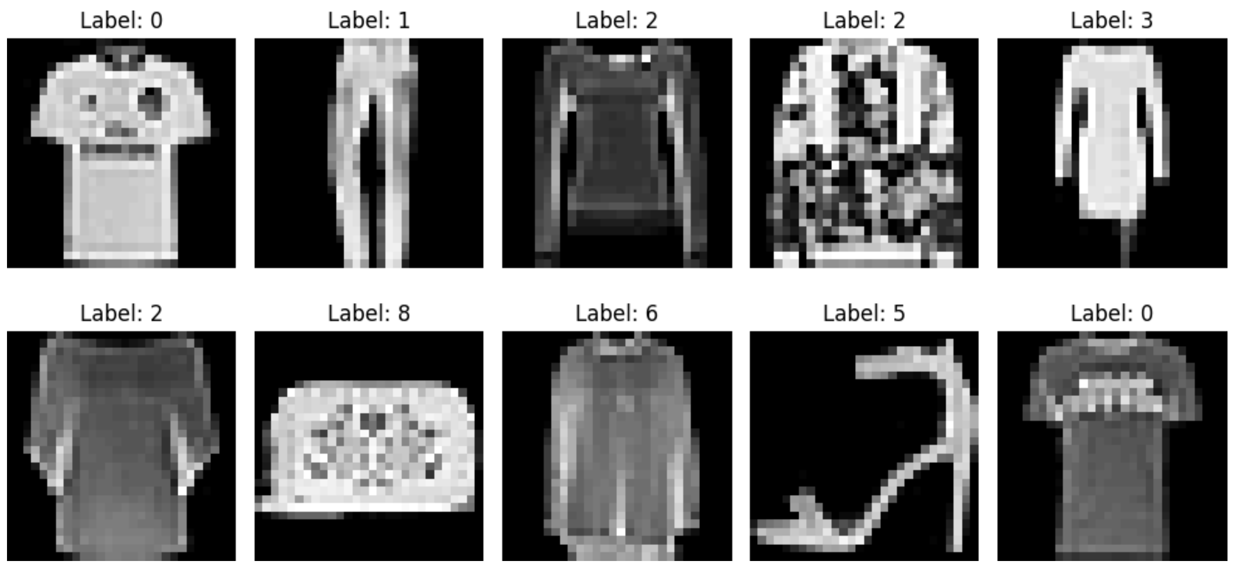


<u>SEC - C:</u>

1. (1 point) Perform appropriate preprocessing on the data (for eg:

normalization) and visualize any 10 samples from the test dataset.

For pre-processing, I have done normalization by dividing the pixel value by 225, to get the values in a range of 0 to 1 which is better for the neural network to learn.



We can see that there are 10 classes of different images related to fashion wearables. The labels are from 0 to 9, therefore this will be a multiclass classification problem.

2. (4 points) Train a MLP Classifier from sklearn’s neural network module

on the training dataset. The network should have 3 layers of size [128, 64, 32],

should be trained for 100 iterations using an ‘adam’ solver with a batch size of

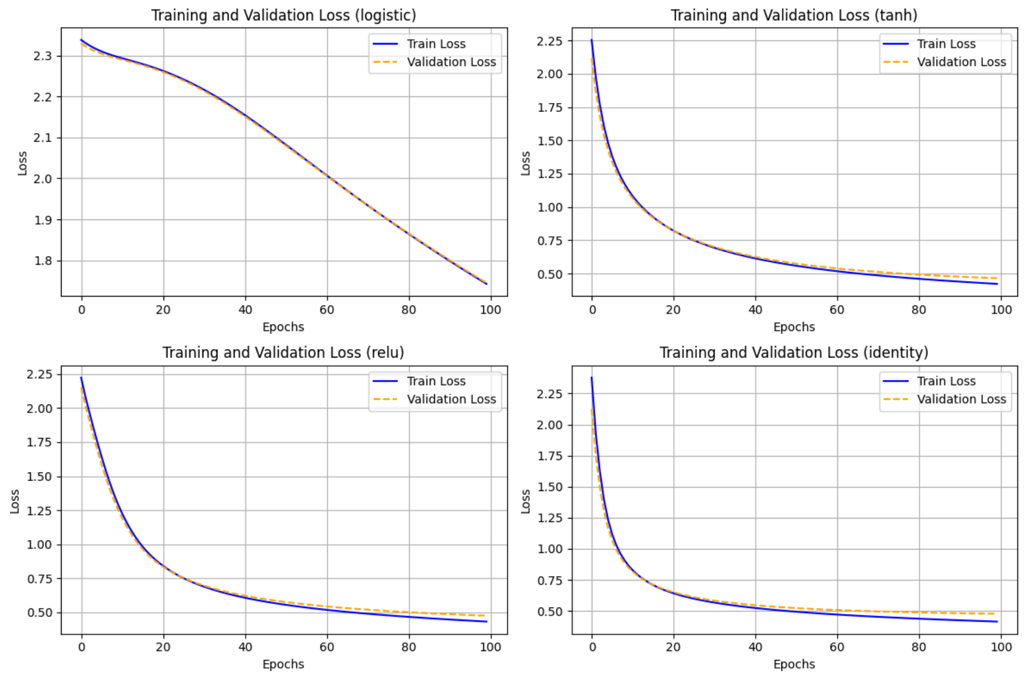
128 and learning rate of 2e-5. Train it using all the 4 activation functions i.e.

‘logistic’, ‘tanh’, ‘relu’ and ‘identity’. For each activation function, plot the training

loss

vs epochs and validation loss vs epochs curves and comment on which activation

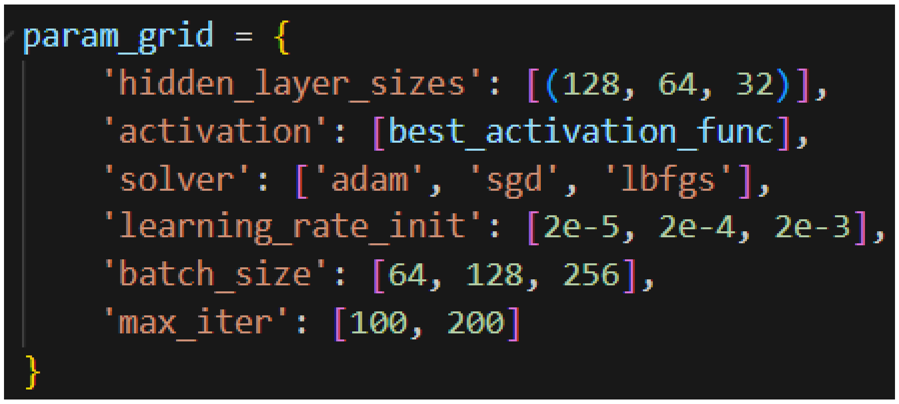
function gave the best performance on the test set in the report.



The above plots are for the training loss vs epochs and validation loss vs epochs curves. Taking accuracy as the performance metric for comparing the different models we can see the result as:

Test Accuracy with logistic: 0.4410
Test Accuracy with tanh: 0.8305
Test Accuracy with relu: 0.8225
Test Accuracy with identity: 0.8225
Best Activation Function: tanh with Accuracy: 0.8305

3. (3 points) Perform grid search using the best activation function from part 2 to find the best hyperparameters (eg: solver, learning rate, batch size) for the MLP classifier and report them in the report.



```
param_grid = {
    'hidden_layer_sizes': [(128, 64, 32)],
    'activation': [best_activation_func],
    'solver': ['adam', 'sgd', 'lbfgs'],
    'learning_rate_init': [2e-5, 2e-4, 2e-3],
    'batch_size': [64, 128, 256],
    'max_iter': [100, 200]
}
```

The above image is of the parameter grid that I have used in the code for grid search

```
Best Parameters: {'activation': 'tanh', 'batch_size': 256,
'hidden_layer_sizes': (128, 64, 32), 'learning_rate_init':
0.002, 'max_iter': 100, 'solver': 'adam'}

Best Cross-Validation Score:

0.854061077080782
```

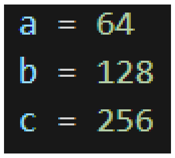
We can see that the best accuracy that we are getting is 0.854 with the above parameters

Test Accuracy with Best Parameters: 0.855

4. (4 points) For this part, you need to train a MLPRegressor from sklearn's neural network module on a regeneration task:

(a) This means you will need to design a 5 layer neural network with layer sizes following the format: [c, b, a, b, c] where c > b > a.

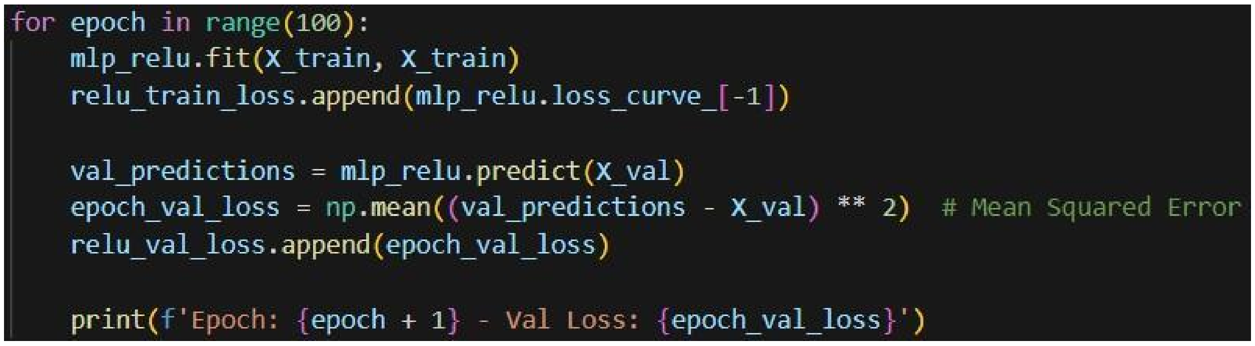
Implemented in code: values for a,b,c are



```
a = 64
b = 128
c = 256
```

(b) By regeneration task, it means that you will try to regenerate the input image using your designed neural network and plot the training and validation losses per epoch to see if your model is training correctly.

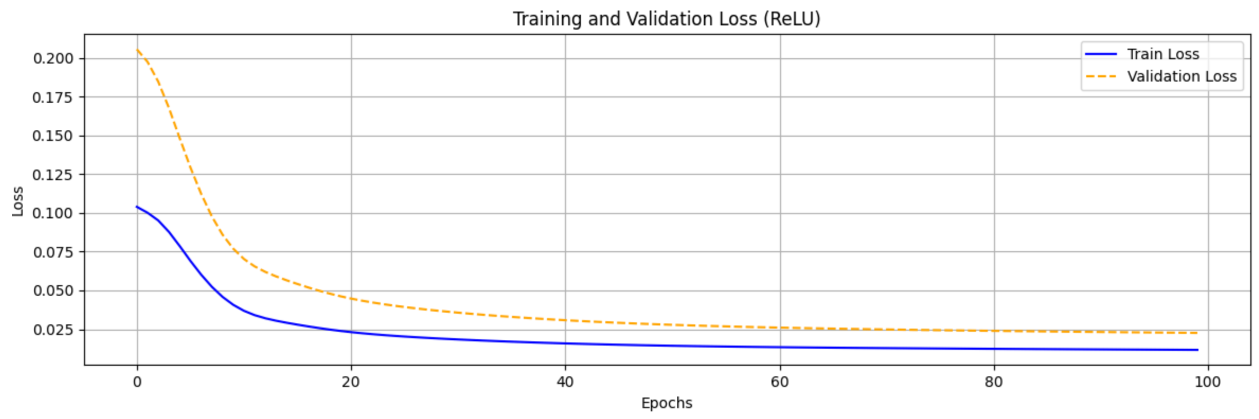
Done in code

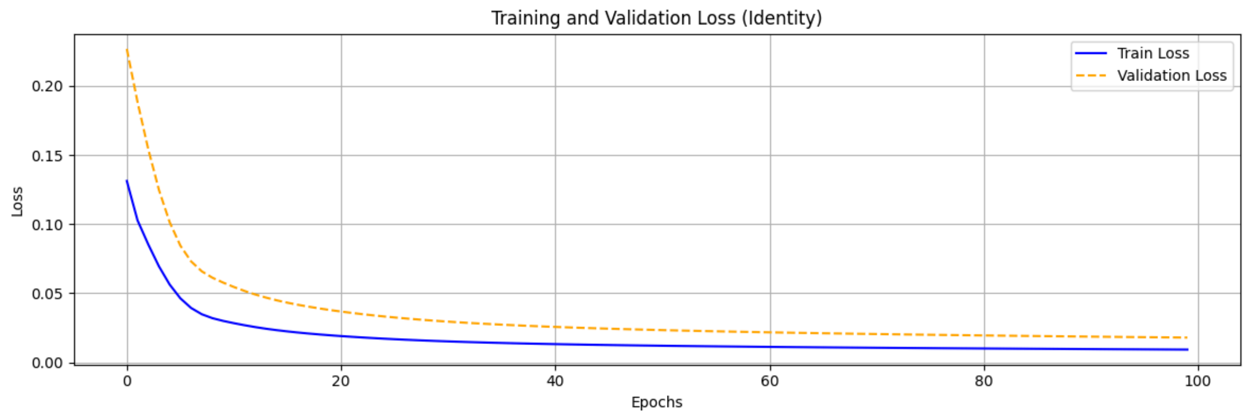


```
for epoch in range(100):
    mlp_relu.fit(X_train, X_train)
    relu_train_loss.append(mlp_relu.loss_curve_[-1])

    val_predictions = mlp_relu.predict(X_val)
    epoch_val_loss = np.mean((val_predictions - X_val) ** 2)  # Mean Squared Error
    relu_val_loss.append(epoch_val_loss)

    print(f'Epoch: {epoch + 1} - Val Loss: {epoch_val_loss}')
```

(c) Train 2 neural networks on the task above. One using a ‘relu’ activation and the other using the ‘identity’ activation function. Set the solver as adam and use a constant learning rate of 2e-5.

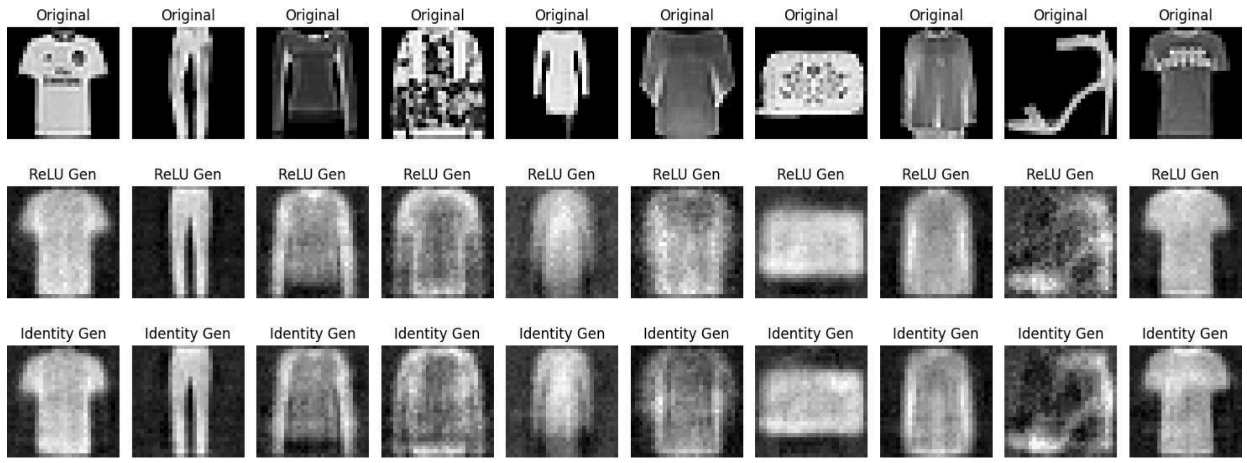
Done in code





From the above plots we can observe that for the task of regeneration, the models are doing quite well, reaching a val loss of around 0.02 for relu and 0.017 for identity.

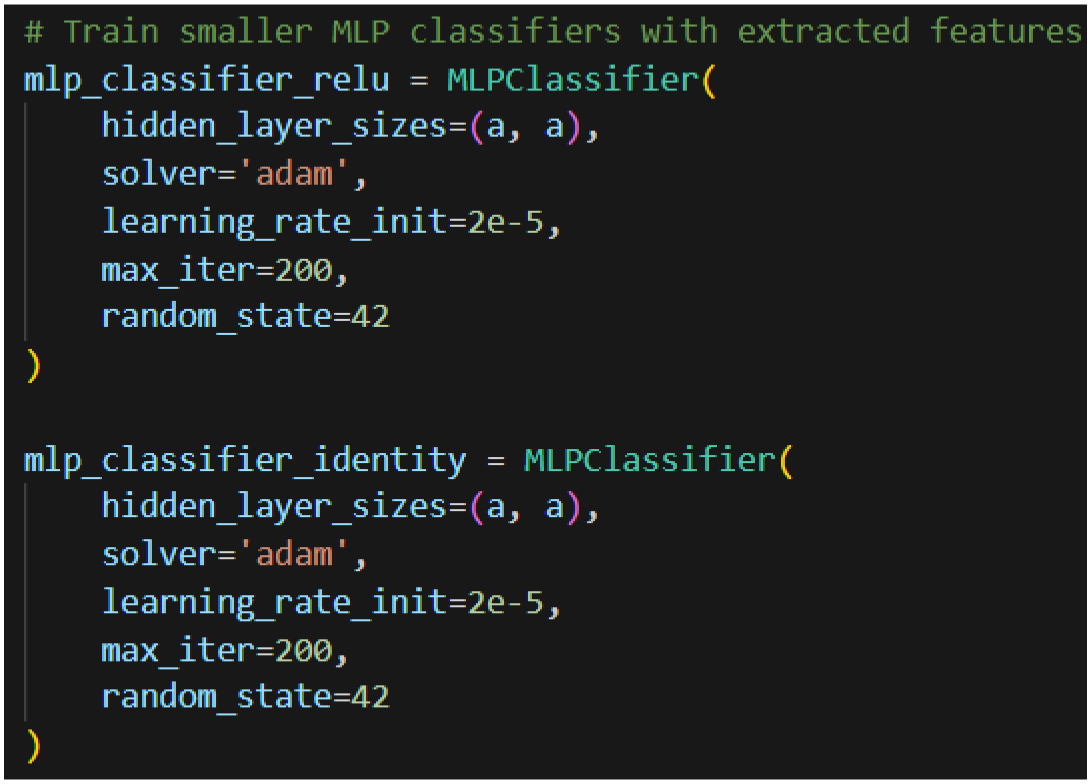
(d) Post training both the networks, visualize the generations for the 10 test samples you visualized in part 1 and describe your observations in the report.



Identity activation function preserves more of the original image details compared to ReLU, which seems to be more blur and there is less contrast, the reason for this could be that the val loss of identity is less than relu during the training process.

5. (3 points) Lastly, from the two neural networks trained above extract the feature vector of size ‘a’ for the train and test data samples. Using this vector as your new set of image features, train two new smaller MLP Classifiers with 2 layers, each of size ‘a’ on the training dataset and report accuracy metrics for both these

classifiers. Train it for 200 iterations with the same solver and learning rate as part 2. Contrast this with the MLP Classifier you trained in part 2 and report possible reasons why this method still gives you a decent classifier?



```
# Train smaller MLP classifiers with extracted features
mlp_classifier_relu = MLPClassifier(
    hidden_layer_sizes=(a, a),
    solver='adam',
    learning_rate_init=2e-5,
    max_iter=200,
    random_state=42
)

mlp_classifier_identity = MLPClassifier(
    hidden_layer_sizes=(a, a),
    solver='adam',
    learning_rate_init=2e-5,
    max_iter=200,
    random_state=42
)
```

Using the same solver and learning rate as part 2, we get the following results:

ReLU MLP Classifier Accuracy: 0.7050

Identity MLP Classifier Accuracy: 0.8010

In part 2 we got the model accuracies as:

```
Test Accuracy with relu: 0.8225

Test Accuracy with identity: 0.8225
```

Feature extraction uses the learned weights of the neural networks to transform raw images into compact, informative vectors. These vectors encapsulate key patterns and features identified by the networks, simplifying the data for further classification.

The reason for similar accuracies is that the feature vectors extracted from the initial neural networks already capture the essential patterns of the images. This allows the smaller MLP classifiers to train on filtered high quality data representations, making it easier to classify accurately without needing large networks or complex learning.