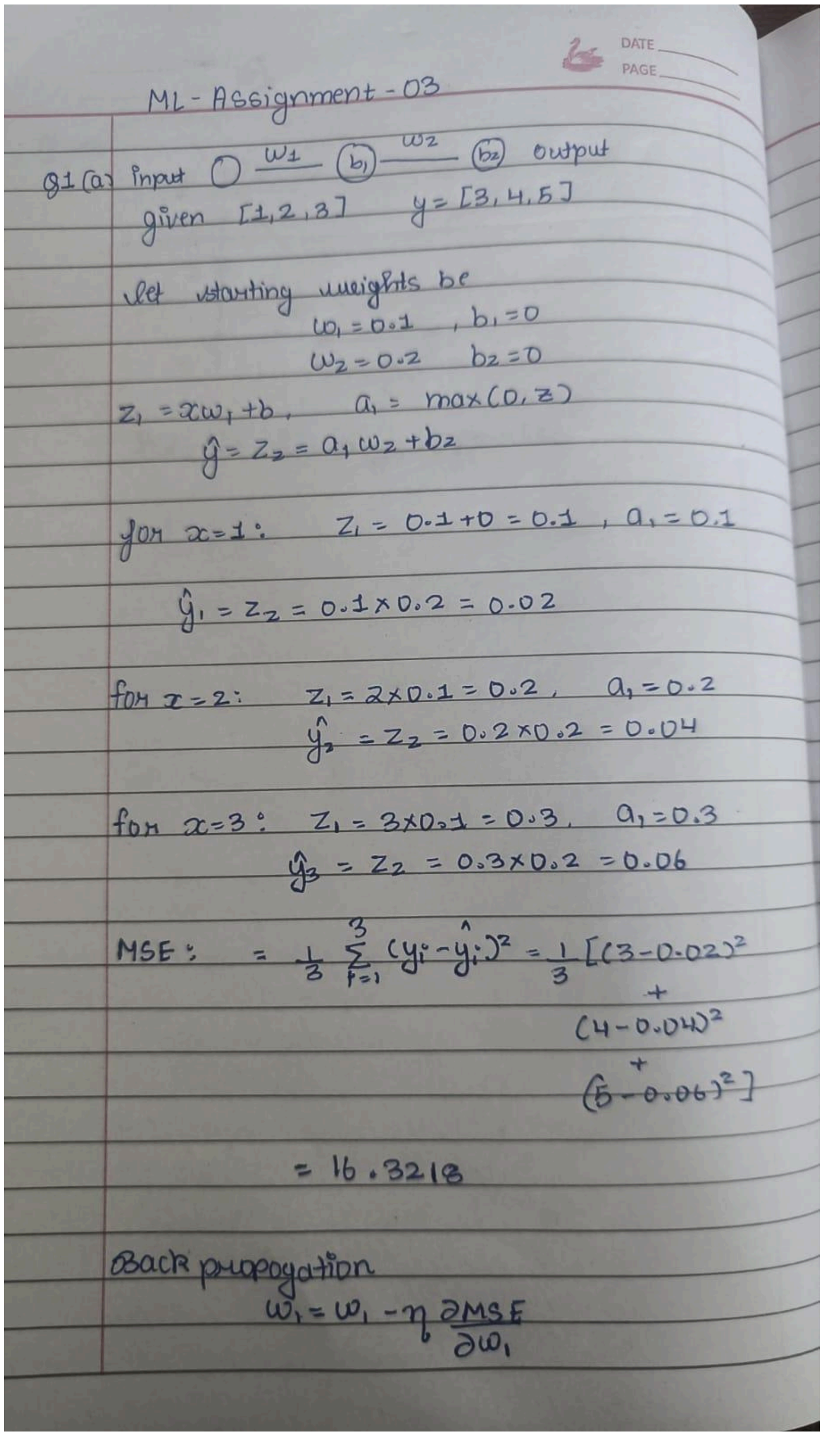
SECTION-A

Part-a



DATE
PAGE

ML - Assignment - 03

Q1 (a) input $$\begin{tikzcd} \bigcirc \arrow[r, dash, "w_1"] & \boxed{b_1} \arrow[r, dash, "w_2"] & \boxed{b_2} \end{tikzcd}$$ output

given [1, 2, 3] $y = [3, 4, 5]$

let starting weights be

$w_1 = 0.1$, $b_1 = 0$

$w_2 = 0.2$ $b_2 = 0$

$z_1 = x w_1 + b_1$, $a_1 = \max(0, z)$

$\hat{y} = z_2 = a_1 w_2 + b_2$

for $x = 1$: $z_1 = 0.1 + 0 = 0.1$, $a_1 = 0.1$

$\hat{y}_1 = z_2 = 0.1 \times 0.2 = 0.02$

for $x = 2$: $z_1 = 2 \times 0.1 = 0.2$, $a_1 = 0.2$

$\hat{y}_2 = z_2 = 0.2 \times 0.2 = 0.04$

for $x = 3$: $z_1 = 3 \times 0.1 = 0.3$, $a_1 = 0.3$

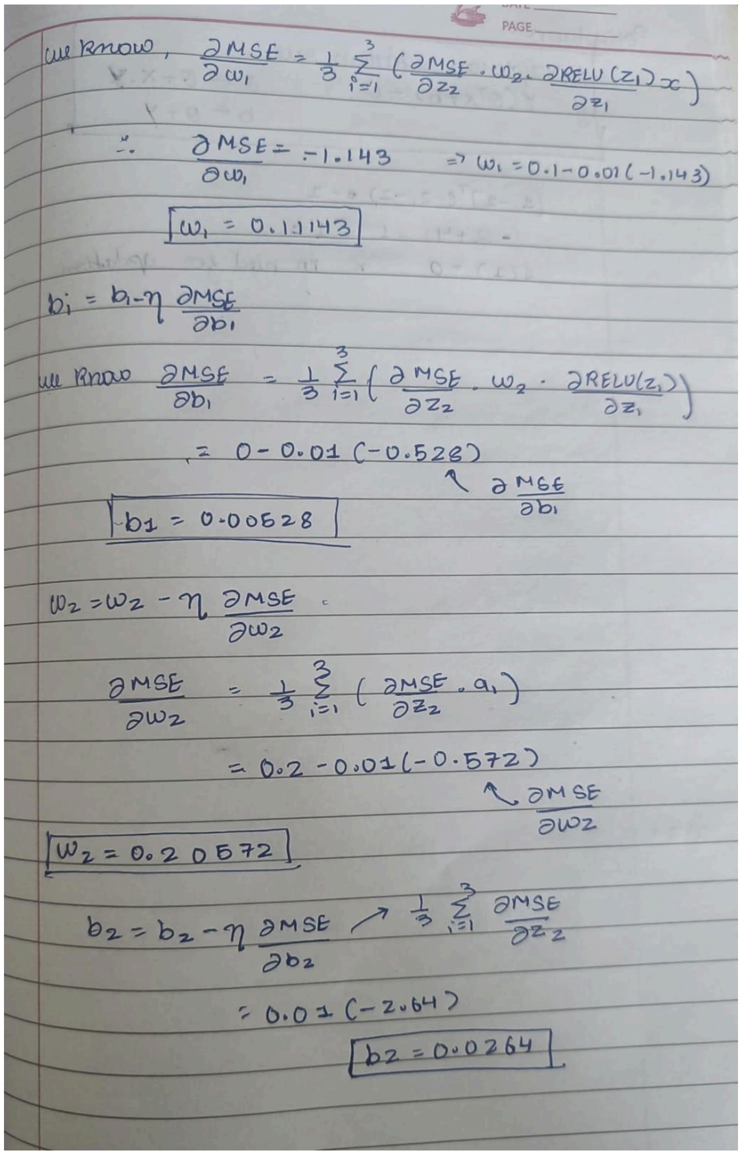
$\hat{y}_3 = z_2 = 0.3 \times 0.2 = 0.06$

$$MSE: = \frac{1}{3} \sum_{i=1}^{3} (y_i - \hat{y}_i)^2 = \frac{1}{3} [(3 - 0.02)^2 + (4 - 0.04)^2 + (5 - 0.06)^2]$$

$= 16.3218$

Back propogation

$$w_1 = w_1 - \eta \frac{\partial MSE}{\partial w_1}$$



we know, $\frac{\partial MSE}{\partial w_1} = \frac{1}{3}\sum_{i=1}^{3}\left(\frac{\partial MSE}{\partial z_2}\cdot w_2 \cdot \frac{\partial RELU(z_1)}{\partial z_1}\, x\right)$

$\therefore \frac{\partial MSE}{\partial w_1} = -1.143 \Rightarrow w_1 = 0.1 - 0.01(-1.143)$

$\boxed{w_1 = 0.11143}$

$b_1 = b_1 - \eta \frac{\partial MSE}{\partial b_1}$

we know $\frac{\partial MSE}{\partial b_1} = \frac{1}{3}\sum_{i=1}^{3}\left(\frac{\partial MSE}{\partial z_2}\cdot w_2 \cdot \frac{\partial RELU(z_1)}{\partial z_1}\right)$

$= 0 - 0.01(\underbrace{-0.528}_{\frac{\partial MSE}{\partial b_1}})$

$\boxed{b_1 = 0.00528}$

$w_2 = w_2 - \eta \frac{\partial MSE}{\partial w_2}$

$\frac{\partial MSE}{\partial w_2} = \frac{1}{3}\sum_{i=1}^{3}\left(\frac{\partial MSE}{\partial z_2}\cdot a_1\right)$

$= 0.2 - 0.01(\underbrace{-0.572}_{\frac{\partial MSE}{\partial w_2}})$

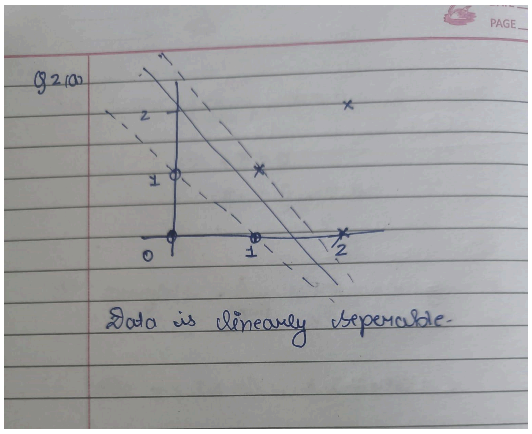
$\boxed{w_2 = 0.20572}$

$b_2 = b_2 - \eta \frac{\partial MSE}{\partial b_2} \nearrow \frac{1}{3}\sum_{i=1}^{3}\frac{\partial MSE}{\partial z_2}$

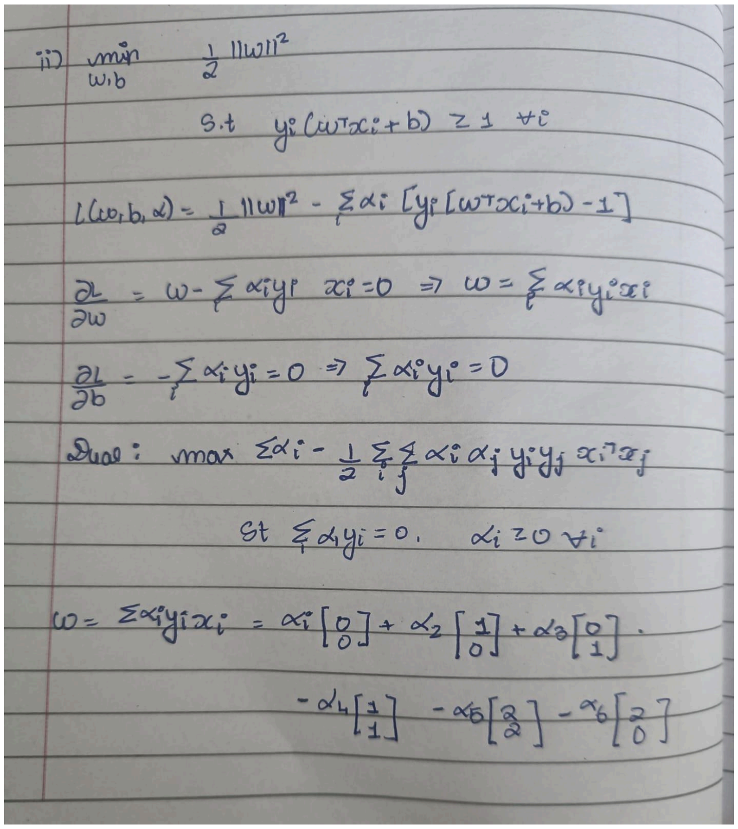
$= 0.01(-2.64)$

$\boxed{b_2 = 0.0264}$

Part-b



Data is linearly seperable.



ii) $\min_{w,b} \quad \frac{1}{2}\|w\|^2$

s.t $\quad y_i(w^T x_i + b) \geq 1 \quad \forall i$

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

$\frac{\partial L}{\partial w} = w - \sum_i \alpha_i y_i \; x_i = 0 \Rightarrow w = \sum_i \alpha_i y_i x_i$

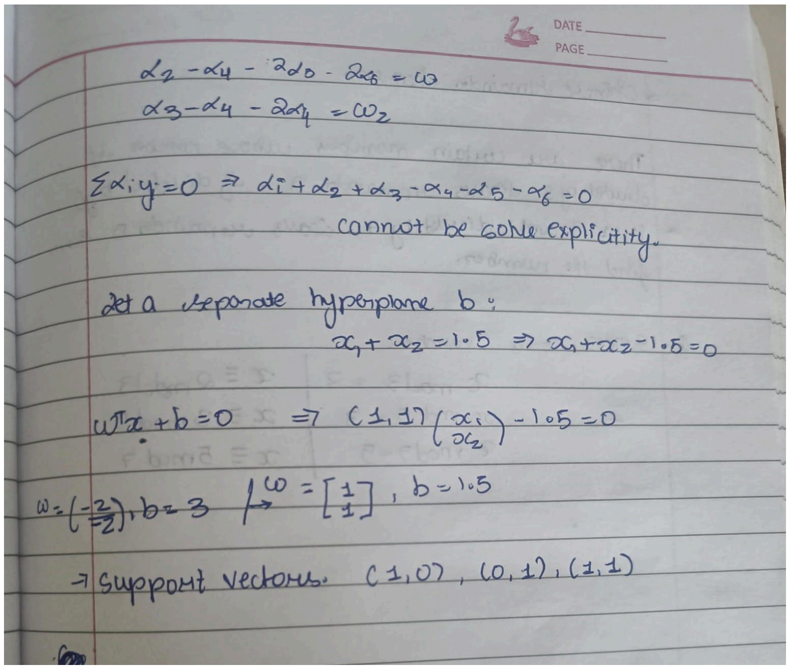
$\frac{\partial L}{\partial b} = -\sum_i \alpha_i y_i = 0 \Rightarrow \sum_i \alpha_i y_i = 0$

Dual: $\max \sum \alpha_i - \frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j y_i y_j x_i^T x_j$

St $\sum \alpha_i y_i = 0. \quad \alpha_i \geq 0 \; \forall i$

$w = \sum \alpha_i y_i x_i = \alpha_i \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \alpha_2 \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \alpha_3 \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$

$- \alpha_4 \begin{bmatrix} 1 \\ 1 \end{bmatrix} - \alpha_5 \begin{bmatrix} 2 \\ 2 \end{bmatrix} - \alpha_6 \begin{bmatrix} 2 \\ 0 \end{bmatrix}$



DATE

PAGE

$\alpha_2 - \alpha_4 - 2\alpha_5 - 2\alpha_6 = w$

$\alpha_3 - \alpha_4 - 2\alpha_4 = w_2$

$\sum \alpha_i y = 0 \Rightarrow \alpha_i + \alpha_2 + \alpha_3 - \alpha_4 - \alpha_5 - \alpha_6 = 0$

cannot be solve explicitity.

let a seperate hyperplane b:

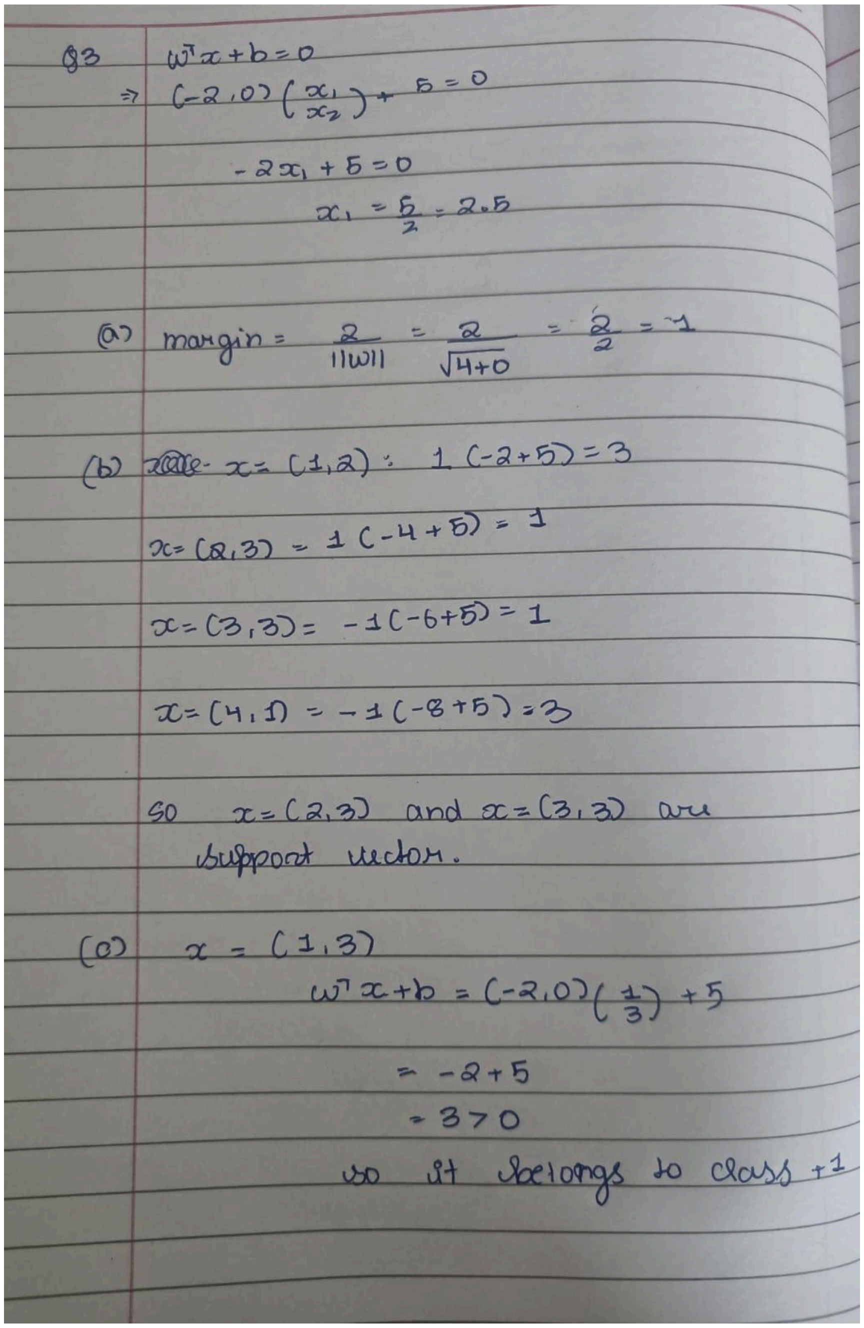
$x_1 + x_2 = 1.5 \Rightarrow x_1 + x_2 - 1.5 = 0$

$w^T x + b = 0 \Rightarrow (1,1)\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} - 1.5 = 0$

$w = \begin{pmatrix} -2 \\ -2 \end{pmatrix}, b = 3 \;\; \nearrow \; w = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \; b = 1.5$

→ support vectors. $(1,0), (0,1), (1,1)$

Part-3



Q3 $w^T x + b = 0$

$\Rightarrow (-2,0)\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + 5 = 0$

$-2x_1 + 5 = 0$

$x_1 = \frac{5}{2} = 2.5$

(a) margin $= \frac{2}{\|w\|} = \frac{2}{\sqrt{4+0}} = \frac{2}{2} = 1$

(b) $x = (1,2) : 1(-2+5) = 3$

$x = (2,3) = 1(-4+5) = 1$

$x = (3,3) = -1(-6+5) = 1$

$x = (4,1) = -1(-8+5) = 3$

so $x = (2,3)$ and $x = (3,3)$ are support vector.

(c) $x = (1,3)$

$w^T x + b = (-2,0)\begin{pmatrix} 1 \\ 3 \end{pmatrix} + 5$

$= -2 + 5$

$= 3 > 0$

so it belongs to class +1

SECTION-B

Data
All the data was given in the format .idx3-ubyte so first I converted these files into csv files and all the images I have sorted in train and test folders respectively.

On reading the data i have found following observations
Data has a total of 60000 rows which is good for training and testing the model.
And there are no null values in the dataset.

And now I have made a new merged_df dataframe in which these columns are there filename, label, image_path, and resized image numpy array for easy splitting data and training data.'

Part-1

Neural Network From Scratch

In neuralNetwok i have implemented these following parameters and functions.
The class is initialized with several important parameters, allowing the user to specify essential model settings:

`N`: Specifies the total number of layers in the network.
`layers`: A list that defines the number of neurons in each layer, where the length of the list equals the number of layers (`N`).
`lr`: Learning rate for the gradient descent optimization.
`activations`: The activation function to be used for all layers except the last layer, which uses softmax for multi-class classification.
weight_init: Defines the weight initialization method to be used (e.g., zero, random, or normal).
`epochs`: The number of epochs (iterations over the entire dataset) for training.
`batch_size`: Batch size to control the number of training samples per gradient update.
Functions

`fit(X, Y)`: Trains the model on input data `X` and labels `Y`. It performs a forward pass, calculates gradients through backward propagation, and updates weights using the specified learning rate and batch size.
`predict(X)`: Predicts class labels for new input data `X` by selecting the class with the highest predicted probability.
`predict_proba(X)`: Returns the class probabilities for input data `X` (useful for evaluating confidence in predictions).
`score(X, Y)`: Calculates accuracy on the test data, comparing predicted labels with true labels.

PART-2

Activation functions implemented from scratch. These functions and their gradients are critical for effective training using backpropagation.

Sigmoid and its gradient: Commonly used for binary classification tasks, but less effective in deep networks due to vanishing gradient issues.
Tanh and its gradient: Similar to sigmoid but zero-centered, making it generally more suitable than sigmoid in deeper networks.
ReLU (Rectified Linear Unit) and its gradient: Popular for hidden layers in deep networks due to its simplicity and ability to mitigate vanishing gradient issues.
Leaky ReLU and its gradient: A variant of ReLU that allows a small gradient when `x < 0`, reducing the risk of dead neurons.
Softmax: Used in the last layer for multi-class classification, providing probabilities across classes.

PART-3

Weight initialization there are different kinds of weight initialization which affect the losses and accuracy so we try different combinations so that to know which weight initialization is better for which activation function.
In this assignment we will define three weight initialization as follows

Zero Initialization: Initializes all weights to zero. This method generally leads to poor performance as it causes neurons in the same layer to learn identical features.

Random Initialization: Sets weights to small random values (multiplied by 0.01). It breaks symmetry and helps layers learn unique features, although it may lead to slow convergence in deep networks.

Normal Initialization: Initializes weights from a standard normal distribution and scales by the square root of the input dimension, which helps maintain gradient flow and reduce the chance of vanishing or exploding gradients.

PART-4

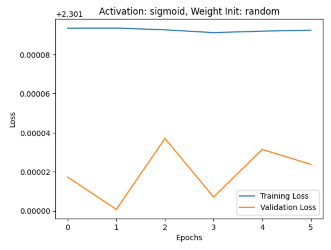
The MNIST dataset, having total 60,000 data, was preprocessed by reshaping each image into a 784-dimensional vector. The data was then split into training, validation, and test sets with an 80:10:10 ratio. To optimize the model's performance, each pixel intensity was normalized, and labels were one-hot encoded.

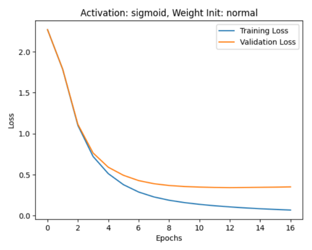
The neural network architecture included four hidden layers with the following configurations:
As the dataset contained 784 pixels so the input layers is of 784 input and output layers have 10 neuron as we have total 10 label classifier.

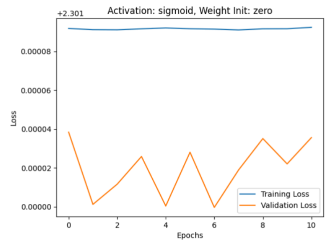
Hidden Layer Sizes: [256, 128, 64, 32]
Activation Functions: Sigmoid, Tanh, ReLU, and Leaky ReLU
Weight Initializations: Random, Normal, and Zero Initializations

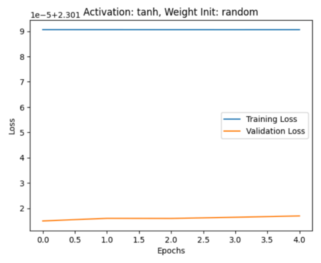
Training was conducted with the following parameters:
Epochs: 50(as computational time was to long so 50) (stopping early if no improvement in validation loss was observed)
Batch Size: 128
Learning Rate: 2e-3

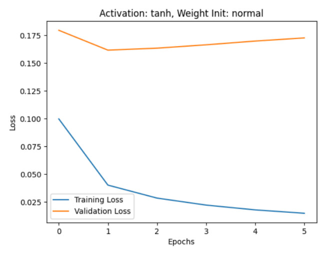
Results and loss curves

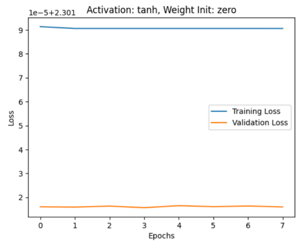


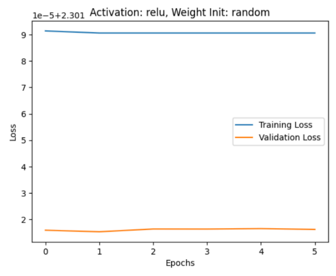


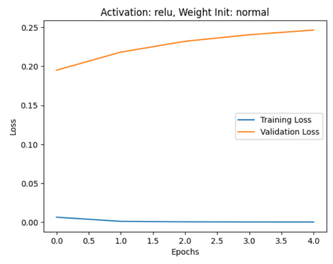


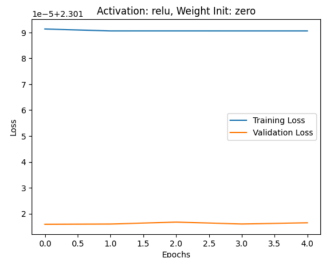


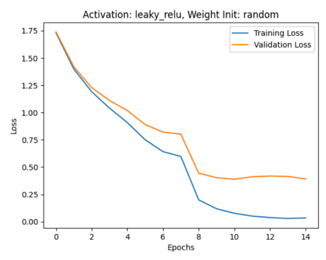


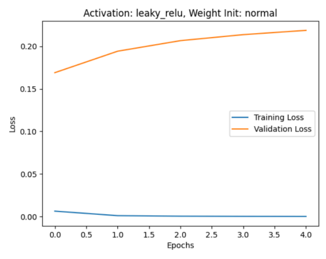


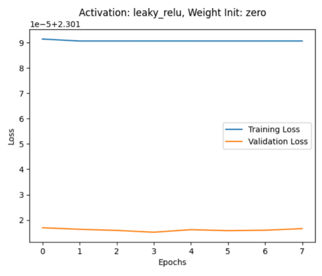












Note: In the above graphs some were having almost the same training loss and validation loss so for that functions the graph was scaled(as written in the top) for better visualization. (This was automatically done by matplotlib)

Sigmoid Activation

1. **Random Initialization:** Convergence was slow, and validation loss plateaued early. The network reached a high validation loss, suggesting that random initialization did not aid the network in escaping from saturation effects typical of sigmoid activations.
2. **Normal Initialization:** Moderate improvement in validation loss but still converged at a high value, with training loss decreasing more effectively than validation loss.
3. **Zero Initialization:** Training loss remained high, showing poor learning due to symmetrical weight updates, resulting in low model performance.

Tanh Activation

1. **Random Initialization:** Model showed minimal reduction in validation loss, suggesting inefficacy in learning meaningful patterns from data.
2. **Normal Initialization:** The model achieved better convergence, reducing validation loss significantly in early epochs before plateauing, indicating the suitability of normal weight initialization with Tanh.
3. **Zero Initialization:** The network failed to learn effectively, with training and validation losses remaining stagnant.

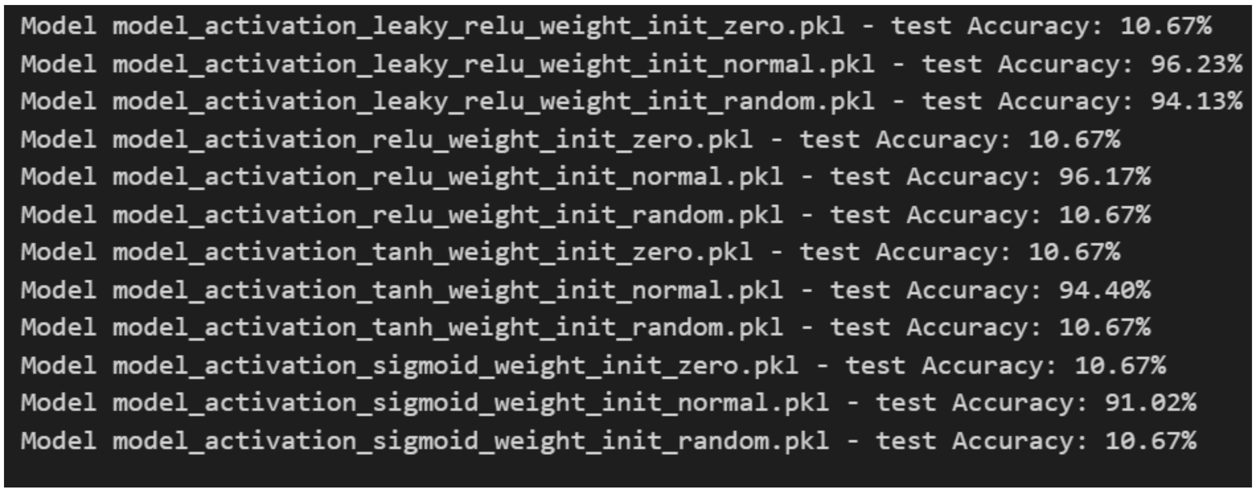
ReLU Activation

1. **Random Initialization:** The network achieved slight reductions in training and validation losses but generally performed poorly, likely due to high sparsity in activation values from random initialization.
2. **Normal Initialization:** Performed the best , quickly reducing training and validation losses and demonstrating the efficacy of ReLU combined with normal weight initialization.
3. **Zero Initialization:** Model struggled to converge due to all neurons receiving similar gradients, leading to poor learning.

Leaky ReLU Activation

1. **Normal Initializations: performed** best among all configurations These configurations showed comparable and stable learning patterns, with normal initialization providing slightly lower validation loss.
2. **Zero Initialization:** Performed poorly as with other activations due to the lack of diversity in initial weights.

The configuration that performed best was the **leaky-ReLU activation** with **normal weight initialization**. This setup achieved the lowest validation loss, showcasing the network's effective learning. Tanh with normal initialization also showed good results but did not converge as quickly as ReLU. Overall, leaky ReLU with normal initialization provided the best balance between training and validation performance, indicating it was the most effective combination for the MNIST classification task.



```
Model model_activation_leaky_relu_weight_init_zero.pkl - test Accuracy: 10.67%
Model model_activation_leaky_relu_weight_init_normal.pkl - test Accuracy: 96.23%
Model model_activation_leaky_relu_weight_init_random.pkl - test Accuracy: 94.13%
Model model_activation_relu_weight_init_zero.pkl - test Accuracy: 10.67%
Model model_activation_relu_weight_init_normal.pkl - test Accuracy: 96.17%
Model model_activation_relu_weight_init_random.pkl - test Accuracy: 10.67%
Model model_activation_tanh_weight_init_zero.pkl - test Accuracy: 10.67%
Model model_activation_tanh_weight_init_normal.pkl - test Accuracy: 94.40%
Model model_activation_tanh_weight_init_random.pkl - test Accuracy: 10.67%
Model model_activation_sigmoid_weight_init_zero.pkl - test Accuracy: 10.67%
Model model_activation_sigmoid_weight_init_normal.pkl - test Accuracy: 91.02%
Model model_activation_sigmoid_weight_init_random.pkl - test Accuracy: 10.67%
```

SECTION-C

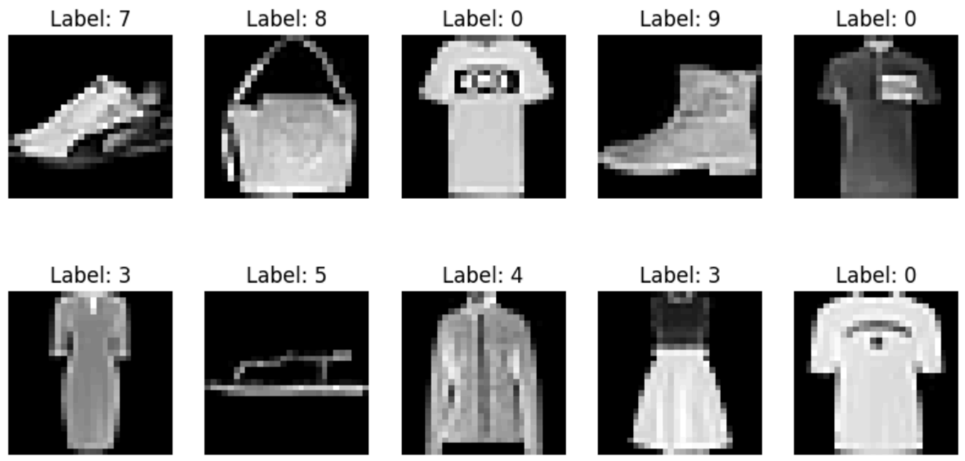
The objective of this task is to implement algorithms for classification and image regeneration using the Fashion-MNIST dataset. Fashion-MNIST, similar in structure to the classic MNIST dataset, contains 28x28 grayscale images of 10 different clothing categories, such as T-shirts, trousers, and coats. Each image has 784 pixel values and an associated label indicating the clothing category, making it an excellent dataset for testing machine learning and neural network models in image classification and reconstruction tasks.

After loading the data, an inspection reveals no missing values in either the training or test set. The training dataset consists of 60,000 images, while the test set comprises 10,000 images. By filtering to the initial 8000 and 2000 samples, respectively, we prepare the data for the tasks of classification, hyperparameter tuning, and regeneration in subsequent sections.

Part-1

The goal of this step is to preprocess the Fashion-MNIST dataset to prepare it for model training and evaluation. Appropriate preprocessing helps enhance the model's performance by normalizing data values to a consistent range. Additionally, visualizing a subset of images provides insight into the dataset's structure and aids in verifying the integrity of the data.

I have scaled the train data and test data using the module called MinMasScaler.
I have also Visualized 10 random pictures from the test dataset.



Part-2

We had a training dataset and test dataset already of size (60000, 785) and (10000, 785) respectively. Now we need a validation set also while training the data so we will extract 10000 rows from the training set for the validation set.

Network Architecture
Network Architecture: 3 hidden layers with sizes [128, 64, 32].
Activation Functions: 'logistic', 'tanh', 'relu', 'identity'.
Solver: 'adam'.
Batch Size: 128.
Learning Rate: 2e-5.
Epochs: 100.

Evaluation Metrics:
Loss: Training and validation loss over epochs, calculated using log loss.
Accuracy: Training and validation accuracy, calculated as the proportion of correct predictions.

Results and observations:

Logistic

Training Accuracy: 76.25%
Validation Accuracy: 75.56%
Observations: The logistic activation achieved relatively lower accuracy compared to other activations. The loss curve displayed slower convergence, likely due to the squashing effect of the logistic function, which restricts gradient flow.

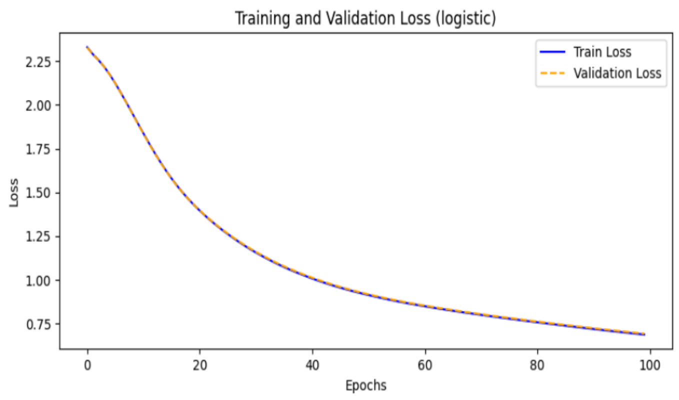
Tanh
Final Training Accuracy: 89.91%
Final Validation Accuracy: 87.76%
Observations: The tanh activation significantly improved accuracy and showed faster convergence in loss curves. This is likely due to its ability to center data around zero, which aids in learning with more balanced gradient flows.

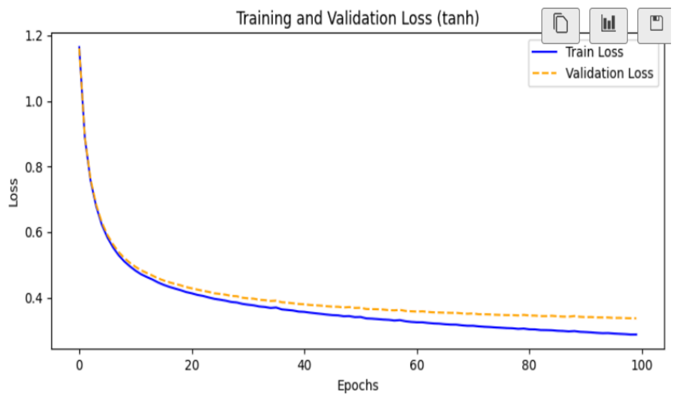
ReLU
Final Training Accuracy: 89.51%
Final Validation Accuracy: 87.83%
Observations: The ReLU activation provided the best validation accuracy overall. It enables better gradient propagation by avoiding vanishing gradient issues commonly seen with logistic and tanh functions, especially in deeper networks. This led to a faster convergence and better generalization.

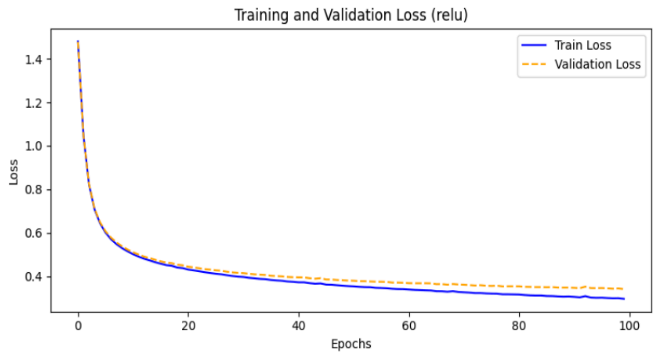
Identity
Final Training Accuracy: 86.97%
Final Validation Accuracy: 85.42%
Observations: The identity function performed relatively well but did not match the performance of tanh and ReLU. Identity activation is generally less effective in capturing non-linear relationships, which limited its accuracy.

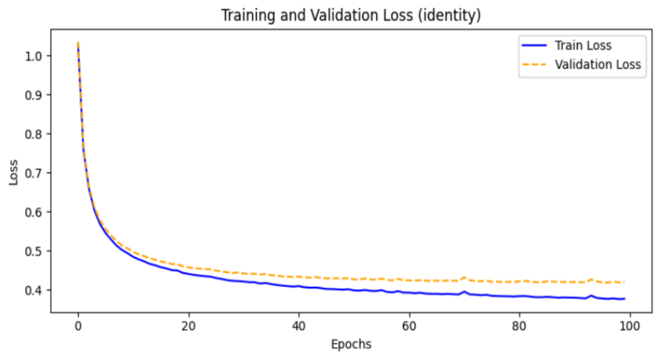
Based on validation accuracy and loss, **ReLU** was the best-performing activation function, achieving the highest final validation accuracy of 87.83%.

In the attached graphs, we can observe that ReLU and tanh activation functions have smoother and faster-converging loss curves compared to logistic and identity. Logistic exhibits a slower decrease in loss, while identity shows moderate performance.









Part-3

From the previous part I found relu is the best activation function for mlp. Now in this part I have tuned various hyperparameters (e.g., solver, learning rate, batch size) for achieving better accuracy using grid search. In the we'll try to maximize the model's cross-validation accuracy on the training dataset.
So fixed parameters are Architecture: [128, 64, 32] hidden layer sizes, **Activation Function:** ReLU (based on previous results), **Max Iterations:** 50 (to limit computation time during grid search), **Random State:** 42 (for reproducibility).

Parameter Grid
Solver: ['adam']
Learning Rate (learning_rate_init): [1e-4, 1e-5]
Batch Size: [128, 256]

By narrowing down the values for each hyperparameter, we aimed to find the best configuration efficiently without testing an exhaustive set of combinations.

Grid Search Configuration:

Cross-Validation: 3-fold cross-validation.
Scoring Metric: Accuracy.
Number of Jobs: -1 (to utilize all available CPU cores)

The grid search tested a total of 4 candidate combinations across 3 folds, totaling 12 fits. After evaluating each combination, the best-performing set of hyperparameters was identified that are batch size:128, Learning Rate (learning_rate_init):0.0001, solver:adam.
Best Cross-Validation Accuracy: 88.12%

Part-4
In this part we have to train a multi-layer perceptron (MLP) regressor to regenerate input images. This task involved constructing a neural network with a specified architecture, training it using different activation functions, and observing the model's performance in terms of its ability to recreate the input images. We evaluated the performance of two models with different activation functions, `ReLU` and `identity`, by analyzing their training and validation losses over epochs and visualizing the regenerated images.

I made a neural network with 5 layers, following the structure `[64, 32, 16, 32, 64]`, where each number represents the number of neurons in the respective layer. This configuration follows the format `[c, b, a, b, c]`, where `c > b > a`. We used two activation functions: `ReLU` and `identity`, and set the solver to `adam` with a constant learning rate of `2e-5`.

And epochs was set to 100 with the target for each sample being the input image itself, creating a regeneration task. For each epoch, the model was evaluated on training and validation data, and the mean squared error (MSE) was recorded.

Results from relu activation:
The MLP model with `ReLU` activation function demonstrated a steady decrease in training and validation losses over the 100 epochs. Initial training and validation losses started at approximately `0.0749` and `0.0745`, respectively, and gradually decreased to around `0.0211` for both by the final epoch.

- The training loss decreased consistently, showing that the model learned to approximate the input image data effectively.
- The validation loss followed a similar trend as the training loss, indicating that the model was not overfitting and generalized well to unseen data.

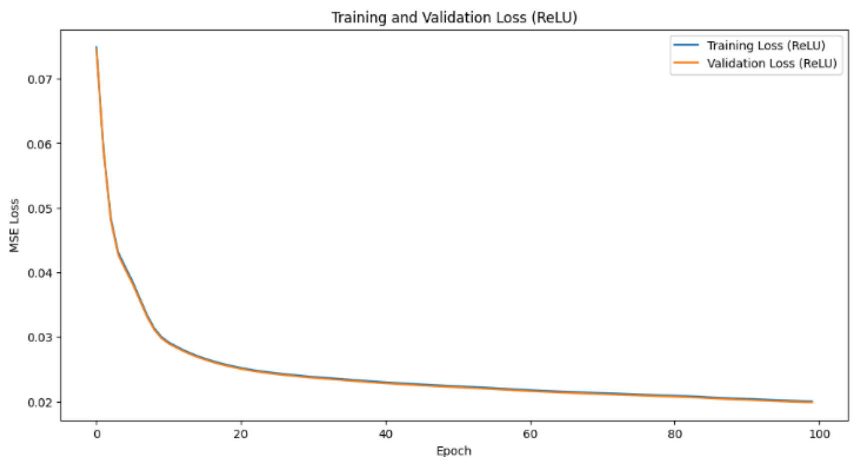
The convergence of the losses suggests that the `ReLU` model was successful in learning to regenerate the input images, with minimal error by the end of training.

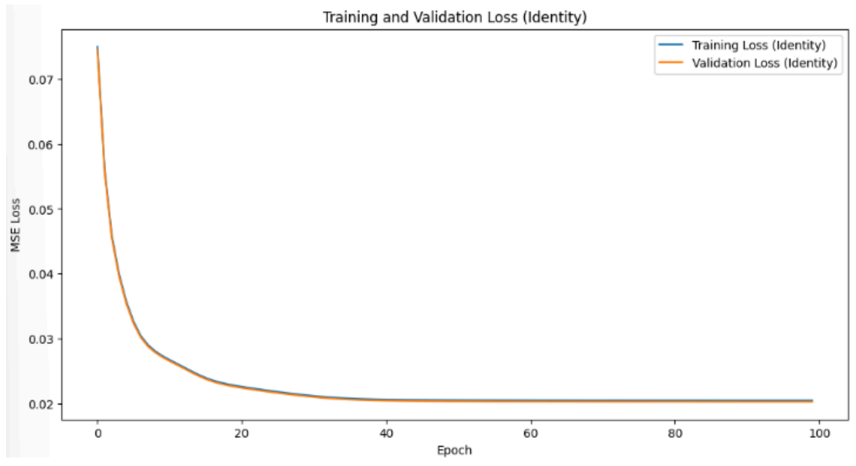
Results from identity activation:

The results of this activation is almost same as the relu activation started with training loss 0.074 and validation loss 0.0744 and decreased to 0.0202.

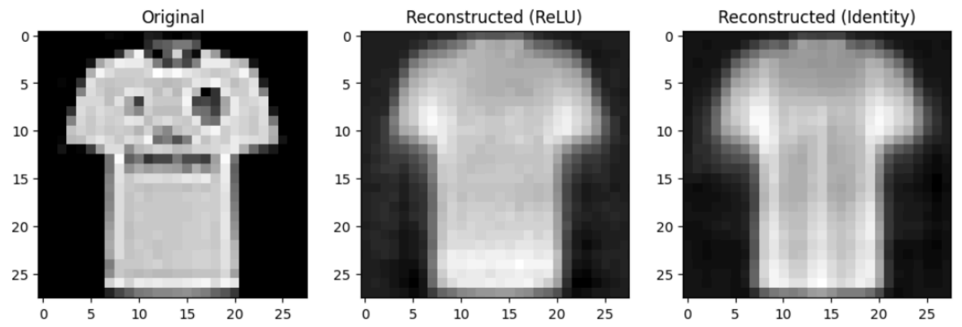
From the above two activation functions the relu activation function is performing better compared to identity function.

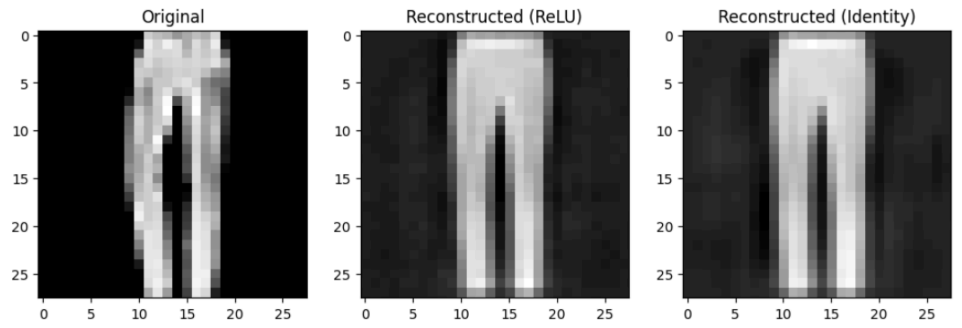
R² metrics for both model
ReLU Model - Test R²: 0.6529935364381618
Iden Model - Test R²: 0.6497690441998021

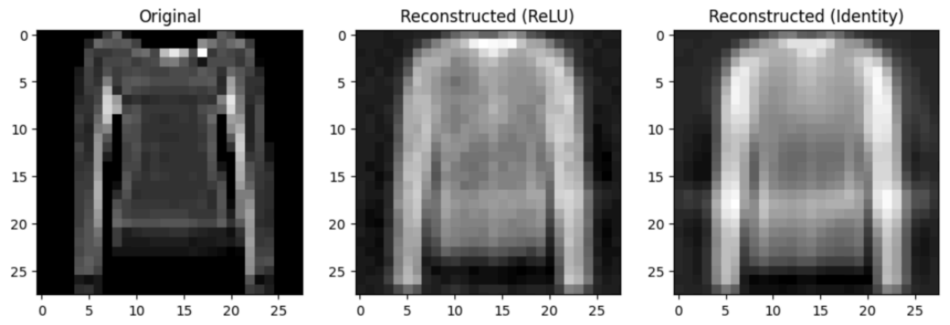


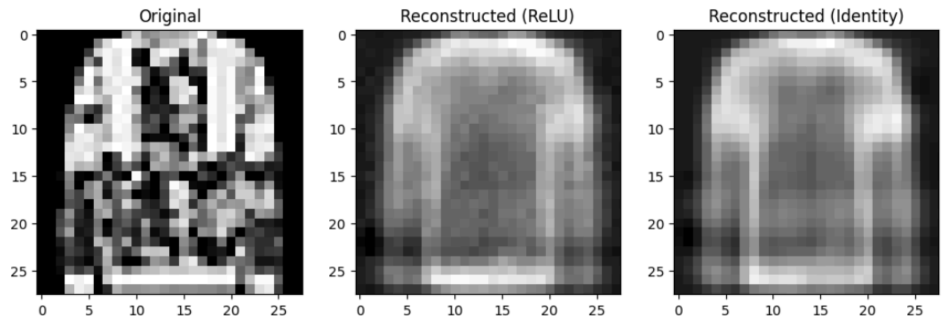


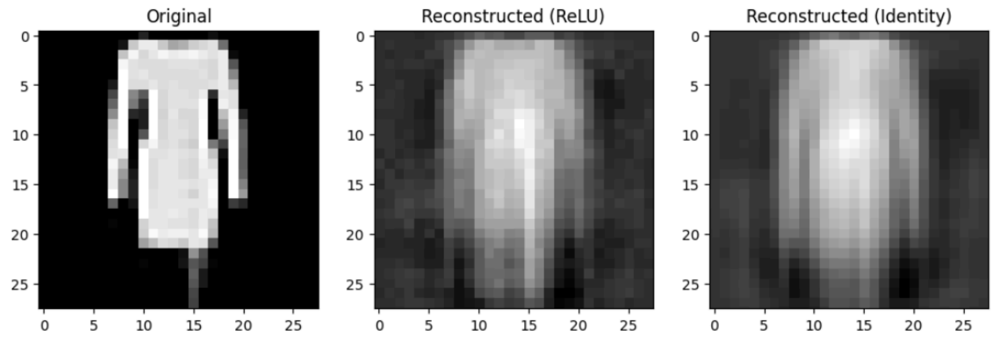
I visualized the regenerated images for 10 test samples to evaluate the quality of the reconstructions. The regenerated images were compared visually against the original inputs.

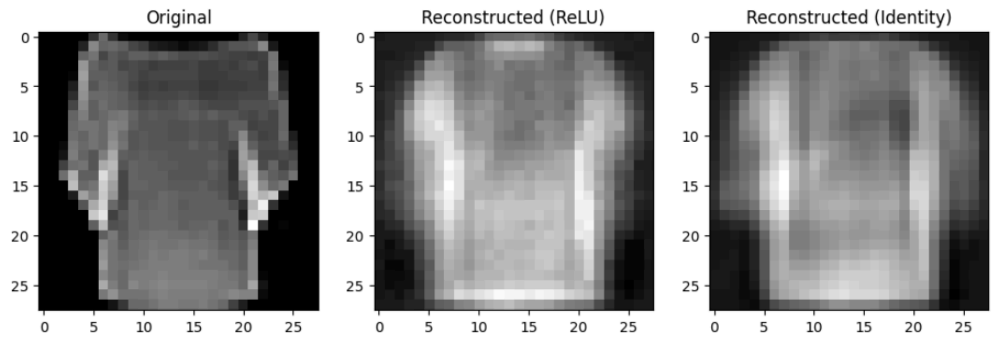

Original
Reconstructed (ReLU)
Reconstructed (Identity)

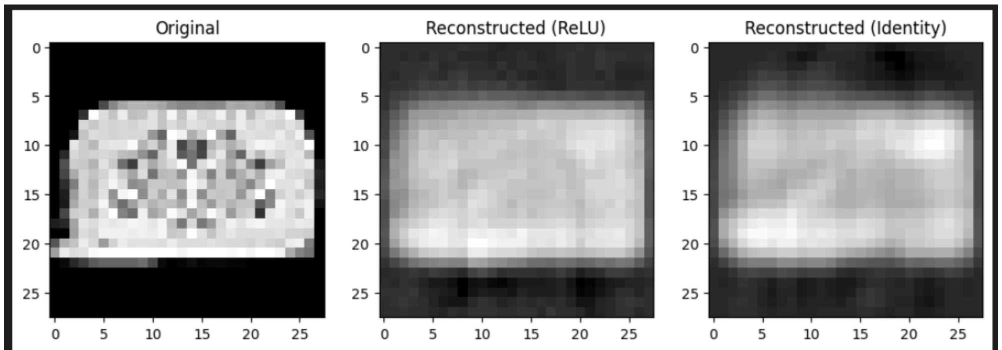

Original
Reconstructed (ReLU)
Reconstructed (Identity)

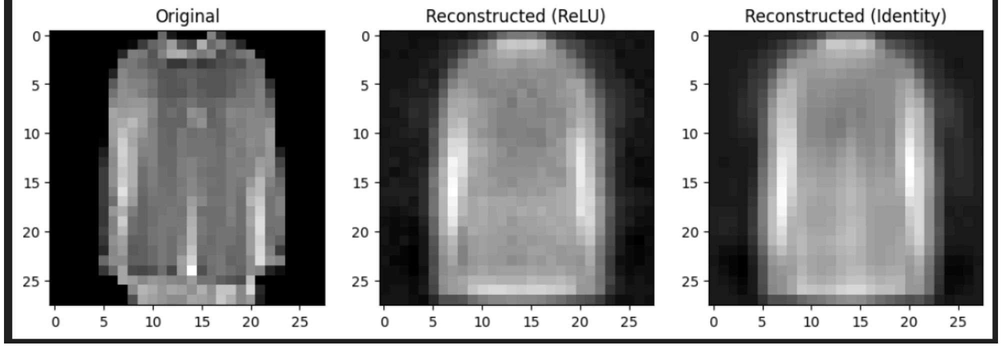

Original
Reconstructed (ReLU)
Reconstructed (Identity)

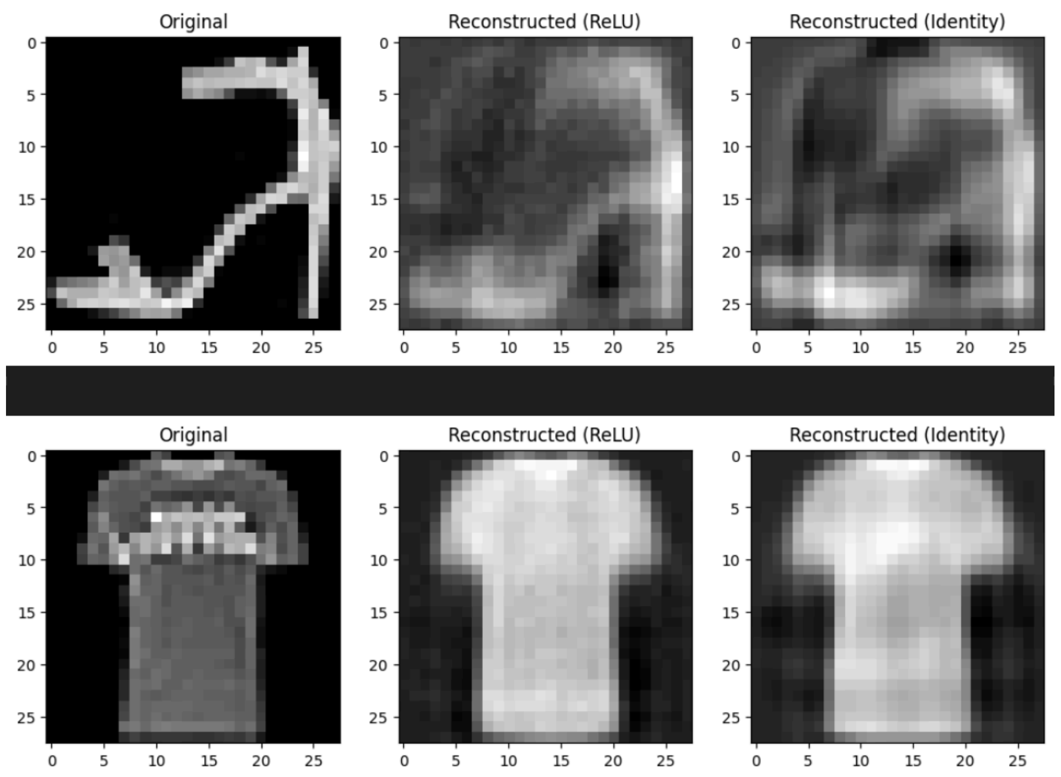

Original
Reconstructed (ReLU)
Reconstructed (Identity)


Original
Reconstructed (ReLU)
Reconstructed (Identity)


Original
Reconstructed (ReLU)
Reconstructed (Identity)


Original
Reconstructed (ReLU)
Reconstructed (Identity)


Original
Reconstructed (ReLU)
Reconstructed (Identity)



From the above images, The relu-activated model produced regenerated images with reasonably high fidelity compared to relu-activation. Details in the reconstructed images were generally well-preserved, especially in the simpler regions. We anticipate that the identity-activated model may have a harder time capturing intricate details compared to the relu model.

Part-5

In this I trained two new Multilayer Perceptron (MLP) classifiers using feature vectors extracted from two previously trained neural networks: one with ReLU activations and another with identity activations.

Feature vectors were obtained from the penultimate layer of the two pre-trained networks (relu_mdl and identity_mdl) for both the training and test datasets, creating compact, lower-dimensional representations of the images.

Using these feature vectors, two smaller MLP classifiers with 2 hidden layers of size a were trained for 200 iterations, using the same solver and learning rate as in part 2, and then evaluated for accuracy on the test set.

ReLU-based Classifier Accuracy: 81.82%
Identity-based Classifier Accuracy: 82.22%