Machine Learning Assignment

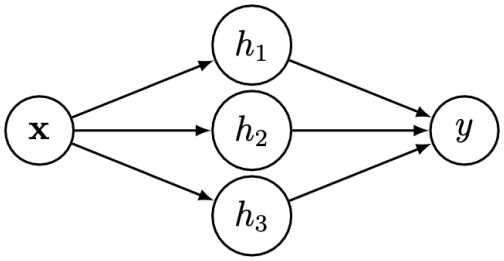
Abhijay Singh
Roll Number: 2021226

Contents

1 Section A

1.1 (a) Training a Multi-Layer Perceptron

Given a regression task, we postulate the presence of three neurons within a solitary hidden layer. Owing to the data and output being unidimensional, a single neuron is employed in both the input and output layers. The topology of our neural network is illustrated in Figure 1, wherein h represents the bias of the hidden layer and b for the output layer. Let's consider the initial weights to be:



$$W_1 = \begin{bmatrix} w_{11} & w_{12} & w_{13} \end{bmatrix} = \begin{bmatrix} 1.0 & 1.0 & 1.0 \end{bmatrix}_{1\times 3} \quad \text{and} \quad W_2 = \begin{bmatrix} w_{21} \\ w_{22} \\ w_{23} \end{bmatrix} = \begin{bmatrix} 1.0 \\ 1.0 \\ 1.0 \end{bmatrix}_{3\times 1}$$

With bias terms initialized to zero and employing ReLU activation, we set the learning rate $\eta = 0.1$. The provided dataset is as follows:

$$\mathbf{x} = \begin{bmatrix} 1.2 \\ 0.8 \\ 2.0 \end{bmatrix}_{3\times 1} \quad \text{and} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 3.0 \\ 2.5 \\ 4.0 \end{bmatrix}_{3\times 1}$$

Iteration 1 (Forward Pass)

During the initial forward pass, we compute:

$$\mathbf{Z}_1 = \mathbf{x}W_1 + h = \begin{bmatrix} 1.2 \\ 0.8 \\ 2.0 \end{bmatrix} \begin{bmatrix} 1.0 & 1.0 & 1.0 \end{bmatrix} + 0 = \begin{bmatrix} 1.2 & 1.2 & 1.2 \\ 0.8 & 0.8 & 0.8 \\ 2.0 & 2.0 & 2.0 \end{bmatrix}$$

Thus, the output $\mathbf{y}$ is:

$$\mathbf{y} = A_1W_2 + b = \text{ReLU}(\mathbf{Z}_1)W_2 + b = \mathbf{Z}_1W_2 + b = \begin{bmatrix} 3.6 \\ 2.4 \\ 6.0 \end{bmatrix}$$

Evidently, $A_2 = \text{ReLU}(\mathbf{y}) = \mathbf{y}$.

Iteration 1 (Loss)

The mean squared error loss for this iteration is computed as:

$$L(W) = \frac{1}{3}\sum_{i=1}^{3}(\hat{y}_i - y_i)^2 = \frac{(3.6-3.0)^2 + (2.4-2.5)^2 + (6.0-4.0)^2}{3} = 1.4567$$

Iteration 1 (Backward Pass)

For the backward pass, we begin by determining the gradient of the loss with respect to the output:

$$\frac{\partial L}{\partial \mathbf{y}} = \frac{1}{3}\left[2(3.6-3.0), 2(2.4-2.5), 2(6.0-4.0)\right] = [0.4, -0.06, 1.34]$$

We proceed to calculate the gradients of the loss with respect to the weights and biases of the output layer:

$$\frac{\partial L}{\partial W_2} = A_1^\top \frac{\partial L}{\partial \mathbf{y}} = \begin{bmatrix}1.2 & 0.8 & 2.0\end{bmatrix}^\top \begin{bmatrix}0.4 & -0.06 & 1.34\end{bmatrix} = \begin{bmatrix}2.016\\1.344\\3.36\end{bmatrix}$$

The updated weights W_2 are then computed as follows:

$$W_2 = W_2 - \eta\frac{\partial L}{\partial W_2} = \begin{bmatrix}1.0\\1.0\\1.0\end{bmatrix} - 0.1\begin{bmatrix}2.016\\1.344\\3.36\end{bmatrix} = \begin{bmatrix}0.7984\\0.8656\\0.6640\end{bmatrix}$$

Similarly, the gradient of the loss with respect to the bias term is found and the bias is updated:

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial \mathbf{y}}\frac{\partial \mathbf{y}}{\partial b} = 1^\top \frac{\partial L}{\partial \mathbf{y}} = 1.68$$

$$b = b - \eta\frac{\partial L}{\partial b} = 0 - 0.1 \cdot 1.68 = -0.168$$

Next, we determine the gradient of the loss with respect to the hidden layer weights:

$$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial \mathbf{Z}_1}W_2^\top = \mathbf{x}^\top \frac{\partial L}{\partial \mathbf{Z}_1} = \begin{bmatrix}1.2 & 0.8 & 2.0\end{bmatrix}^\top W_2^\top$$

The weights W_1 are updated as:

$$W_1 = W_1 - \eta\frac{\partial L}{\partial W_1} = \begin{bmatrix}1.0 & 1.0 & 1.0\end{bmatrix} - 0.1\begin{bmatrix}1.389 & 0.925 & 2.314\end{bmatrix} = \begin{bmatrix}0.86 & 0.90 & 0.76\end{bmatrix}$$

Finally, we compute the gradient of the loss with respect to the hidden layer bias term and update it:

$$\frac{\partial L}{\partial h} = \frac{\partial L}{\partial \mathbf{Z}_1}\frac{\partial \mathbf{Z}_1}{\partial h} = 1^\top \frac{\partial L}{\partial \mathbf{Z}_1} = 1.157$$

$$h = h - \eta\frac{\partial L}{\partial h} = 0 - 0.1 \cdot 1.157 = -0.1157$$

1.2 (b) Nonlinear SVM with Gaussian Kernel

Classification Rule Expression

In the case of a nonlinear SVM using a Gaussian (RBF) kernel, the classification rule for a test sample x is determined by the following decision function:

$$f(x) = \text{sign}\left(\sum_{i=1}^{N} \alpha_i y_i K(x_i, x) + b\right)$$

Where:

- N is the number of support vectors.
- α_i are the Lagrange multipliers.
- y_i are the labels of the training samples.
- $K(x_i, x)$ is the Gaussian kernel function, defined as:

  $$K(x_i, x) = \exp\left(-\gamma \|x_i - x\|^2\right)$$

  with γ controlling the width of the kernel.
- b is the bias term.

Relation Between α_i and Hyperparameter C

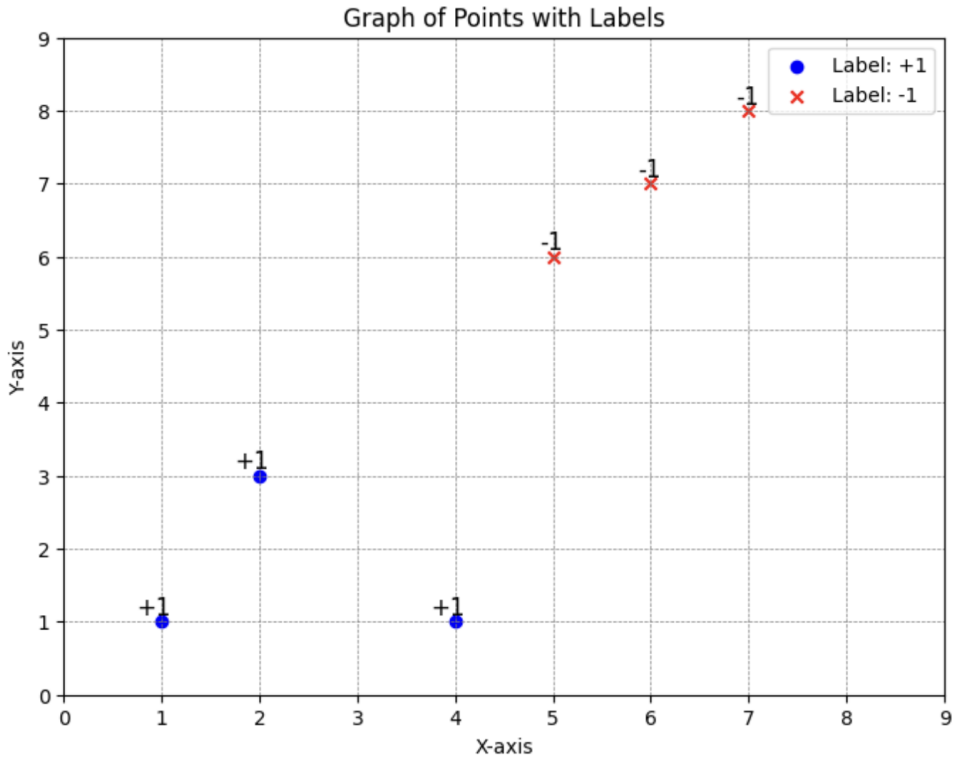
The relationship between the Lagrange multipliers α_i and the SVM hyperparameter C varies based on the positioning of data points with respect to the decision boundary and margin. This can be summarized as follows:

1. **Point** X_1 (Correct side of margin): $\alpha_i = 0$. These points are not support vectors and do not impact the decision boundary.
2. **Point** X_2 (On the margin): $0 < \alpha_i < C$. These points are support vectors and directly influence the decision boundary.
3. **Point** X_3 (Wrong side of margin, correct side of hyperplane): Similar to X_2, $0 < \alpha_i < C$. These are also support vectors.
4. **Point** X_4 (Wrong side of decision hyperplane): $\alpha_i = C$. These points are support vectors and affect the decision boundary and margin.

In essence, the values of α_i indicate whether a point is a support vector and its relative position to the margin and decision hyperplane. Points with $\alpha_i > 0$ are identified as support vectors, and the value of α_i relative to C provides insight into their positioning.

1.3 (c)

Part (a): Linear Separability



Points are linearly separable when there exists a linear decision boundary that can separate the classes of data points without error. In the given scenario, the points (2,3), (5,6), and (4,1) can be separated into distinct classes using a linear boundary. From the graph of these points, it's evident that they do not overlap and can be divided by a line, indicating that they are linearly separable. This is crucial in the context of SVMs as it suggests that a linear kernel is sufficient to classify these points correctly.

Part (b) - Decision Boundary

The decision boundary in a support vector machine (SVM) is a hyperplane that separates the different classes of data points. In our case, we are working with a two-dimensional space, so the boundary is a line. The decision boundary is chosen such that the distance from the nearest data points (support vectors) of each class is maximized. For the given points (2,3), (5,6), and (4,1), the suitable decision boundary is the line represented by the equation $x+y=8$. This line is chosen because it effectively separates the data points into two distinct groups with maximum separation. Graphical analysis of these points helps to visually confirm that this line is indeed an appropriate separator.

Part (c) - Support Vectors

Support vectors are data points that are closest to the decision boundary and are critical in defining the position and orientation of the boundary. In this SVM problem, the support vectors are identified as (5,6), (2,3), and (4,1). These points are crucial because

they are the closest to the decision boundary, and any adjustment to them would directly affect the placement of the boundary. The selection of these support vectors is based on the fact that they are either on the margin or violate the margin, thus playing a pivotal role in maximizing the margin between the classes.

Part (d) - Margin Calculation

The margin in an SVM is defined as the distance between the decision boundary and the nearest points (support vectors) from each class. To calculate the margin for the decision boundary $x + y = 8$, we use the formula for the distance of a point from a line:

$$\text{Distance} = \frac{|ax_1 + by_1 + c|}{\sqrt{a^2 + b^2}}$$

where $ax + by + c = 0$ is the equation of the line, and (x_1, y_1) is a point. For our decision boundary $x + y = 8$ or $x + y - 8 = 0$, $a = 1$, $b = 1$, and $c = -8$. Using the support vector (2,3), the margin calculation is:

$$\text{Distance} = \frac{|1 \times 2 + 1 \times 3 - 8|}{\sqrt{1^2 + 1^2}} = \frac{3}{\sqrt{2}}$$

The margin M is twice the distance:

$$M = 2 \times \frac{3}{\sqrt{2}} = \frac{6}{\sqrt{2}}$$

Part (e): Effect of Removing Support Vectors

In SVM, support vectors are pivotal in defining the decision boundary and its margin. Removing a support vector can alter the optimal margin. In this case, if we remove the point (5,6), the optimal margin will change, indicating its critical role in supporting the current margin. However, removing the other two points, (2,3) and (4,1), does not change the optimal margin. This suggests that (5,6) is a more influential support vector in terms of maintaining the current decision boundary and margin, highlighting the asymmetric importance of different support vectors in SVM classification tasks.

2 Section B Neural Network Implementation

In Section B of our assignment, we implemented a class named `NeuralNetwork` from scratch with the ability to configure multiple parameters during its initialization. The parameters included the number of layers N, the size of each layer specified in a list, the learning rate `lr`, a common activation function for all layers (except the last), a weight initialization function, the number of epochs, and the batch size.

The class provided the following functionalities:

- `fit(X, Y)`: To train the model on input data X and labels Y.
- `predict(X)`: To output predictions for input X.
- `predict_proba(X)`: To provide class-wise probabilities for input X.

- `score(X, Y)`: To calculate the accuracy of the model on input X with labels Y.

Furthermore, we implemented various activation functions such as `sigmoid`, `tanh`, `ReLU`, `Leaky ReLU`, `linear`, and `softmax` (for the last layer), along with their respective gradient functions required for the backpropagation algorithm.

For weight initialization, we created three different functions: `zero_init`, `random_init`, and `normal_init` with a normal distribution, along with appropriate scaling factors to facilitate effective learning.

We trained our neural network on the MNIST dataset with the following configurations:

- 4 hidden layers with sizes [256, 128, 64, 32]
- 100 epochs
- Batch size of 128

As part of the training process, we performed the necessary preprocessing steps which included normalizing the input data and one-hot encoding the labels. We also plotted the training and validation loss against epochs to visually assess the model's learning process. These plots are crucial for understanding the model's behavior over time and ensuring that it learns effectively.

All trained models were saved for potential demonstration and future evaluation.

2.1 Loss Visualization

The training and validation loss plots are shown below, which provide insights into the model's performance and generalization capability over the epochs.

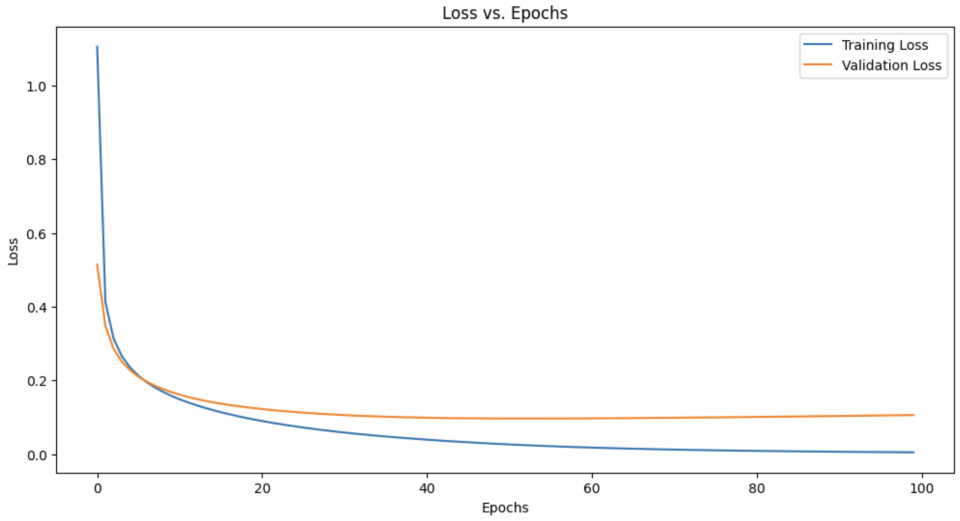


Figure 1: Training and Validation Loss over Epochs (ReLU)

The training and validation loss curves depicted in the graph illustrate the learning process of the neural network over 100 epochs. Initially, there is a sharp decrease in both training and validation losses, indicating rapid learning. This trend is typical in the early stages of training as the model begins to fit the data.

As the epochs progress, the reduction in loss becomes more gradual, suggesting that the model is starting to converge. The close proximity of the training and validation loss curves throughout the training process is a strong indicator of the model's ability to generalize. This is evidenced by the absence of a growing gap between the two curves, which would typically signify overfitting.

The convergence of both losses without significant divergence implies that the model is well-regularized and the capacity is appropriate for the complexity of the task.

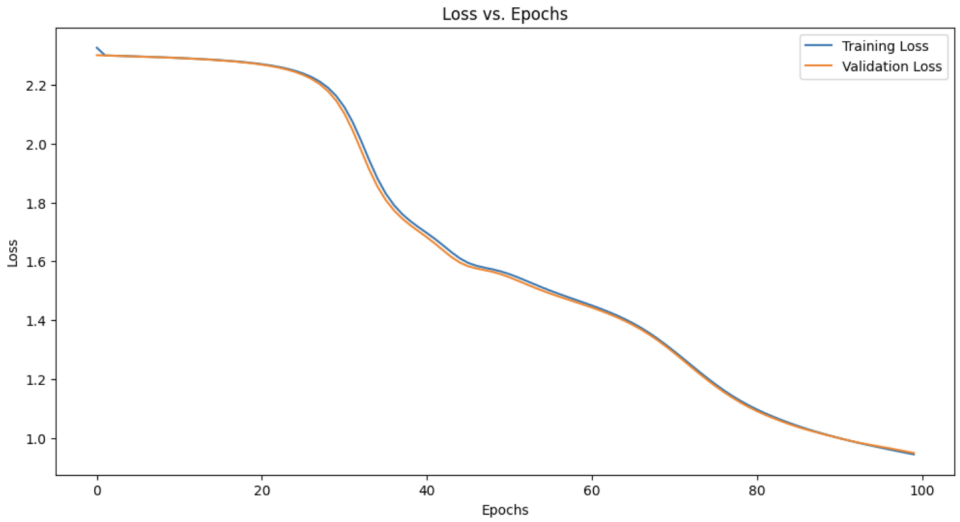


Figure 2: Training and Validation Loss over Epochs (sigmoid)

The graph shows the training and validation losses converging smoothly, indicative of good model generalization. However, the slight flattening of the curve suggests a deceleration in learning, which might indicate the model is nearing its capacity to improve further with the sigmoid function.

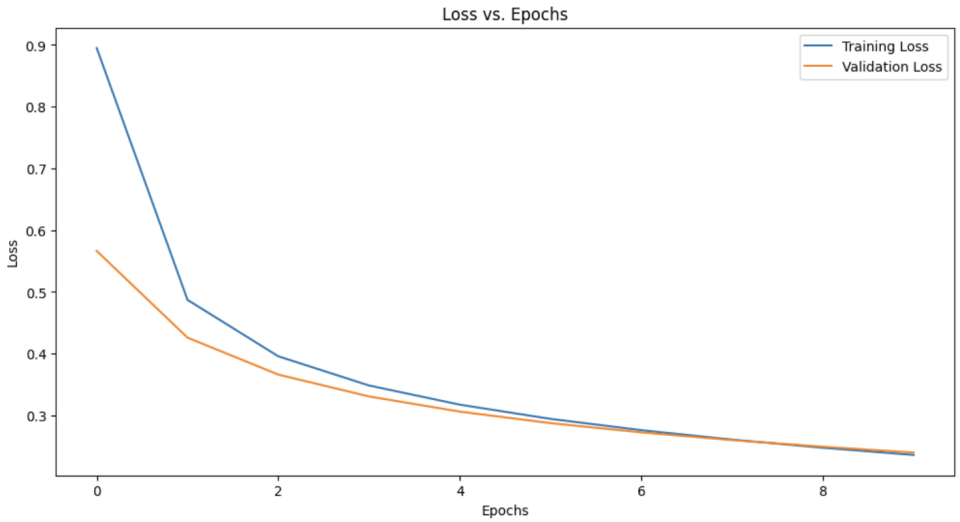


Figure 3: Training and Validation Loss over Epochs (tanh)

The tanh activation function's performance is marked by a steep initial drop in loss, indicating fast learning at the outset. The curves then begin to plateau, with the valida-

tion loss slightly higher than the training loss, hinting at the beginnings of overfitting as the model becomes too tailored to the training data.

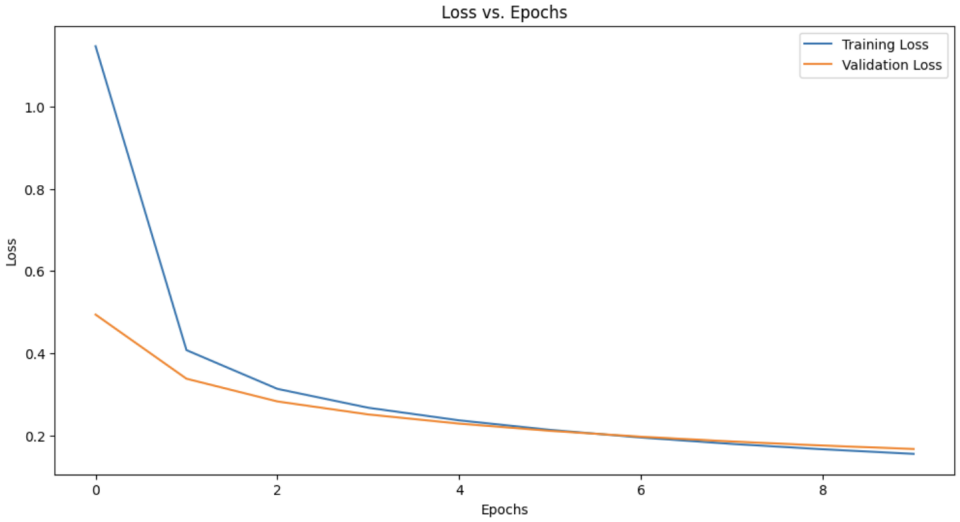


Figure 4: Training and Validation Loss over Epochs (Leaky ReLU)

This graph displays a more pronounced gap between training and validation losses, compared to ReLU. The higher validation loss throughout the training process suggests the model may be overfitting the training data, as it does not generalize as effectively to the validation data.

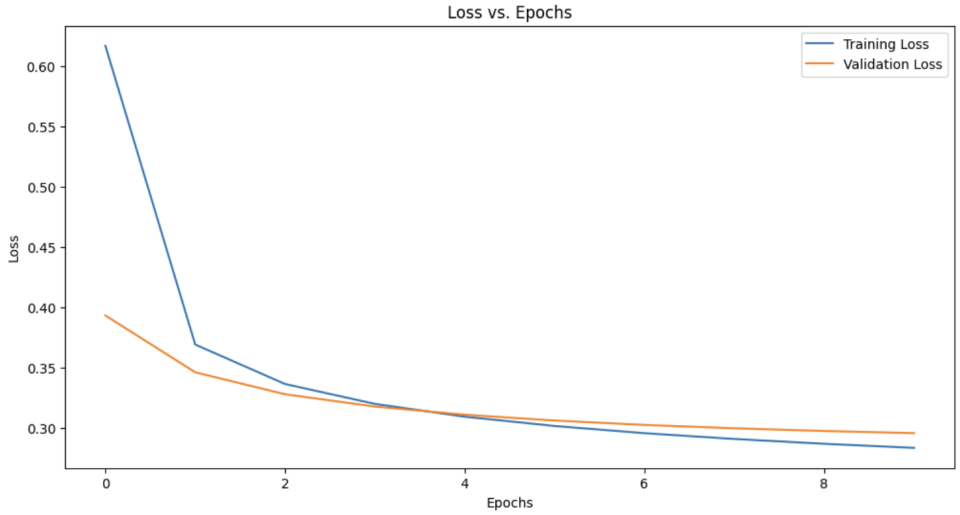


Figure 5: Training and Validation Loss over Epochs (Linear)

The linear activation shows a surprising result with closely tracking training and validation losses, unusual for a task with non-linear patterns like image classification. While both losses decrease steadily, their close proximity may imply the model's capacity for generalization is maintained throughout the training process. However, for more complex tasks, the linear activation might not capture the intricate relationships as effectively as non-linear alternatives.

Each activation function's graph offers insights into the model's learning dynamics and generalization ability, with the ReLU and linear activations showing promising results,

while sigmoid and tanh indicate areas for optimization, especially to combat overfitting and learning saturation.

3 Section C: Application of MLP on SVHN Dataset

3.1 Dataset Preparation

We employed the Street View House Numbers (SVHN) dataset, comprising numerous house number images from Google Street View. Our approach to preparing this dataset for the MLP classifier included loading the data via scipy.io, adjusting labels to represent '0' correctly, and segmenting into distinct training (60%), validation (20%), and test (20%) sets.

3.2 Class Label Distribution Visualization

The class labels' distribution was graphically represented to confirm uniform representation across classes. Histograms depicted this balance, ensuring model training integrity.

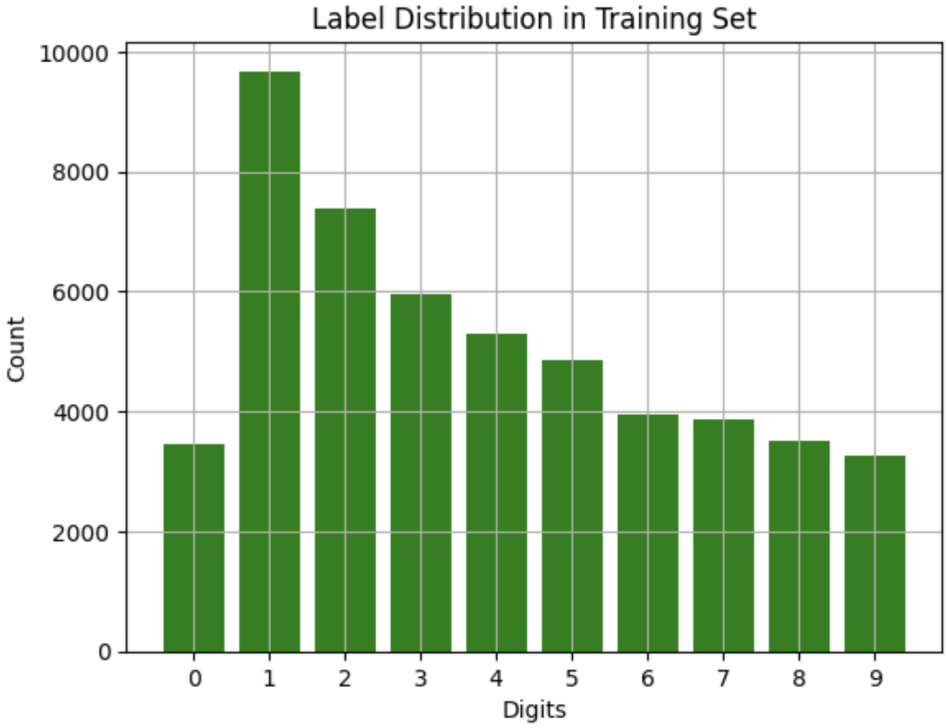


Figure 6: Distribution of class labels in the training set

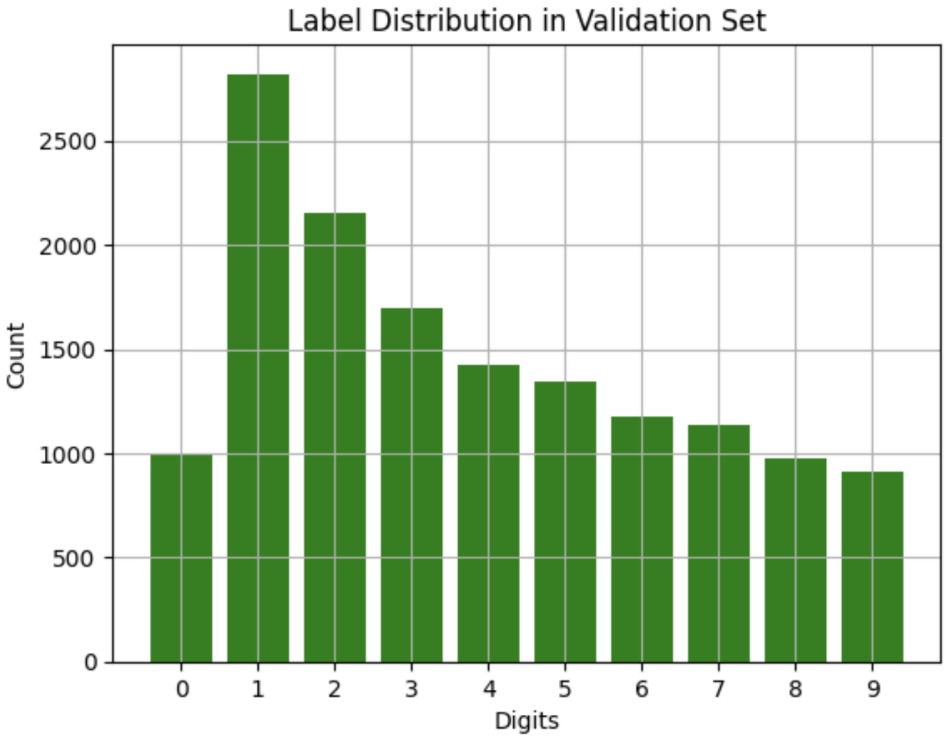


Figure 7: Distribution of class labels in the validation set

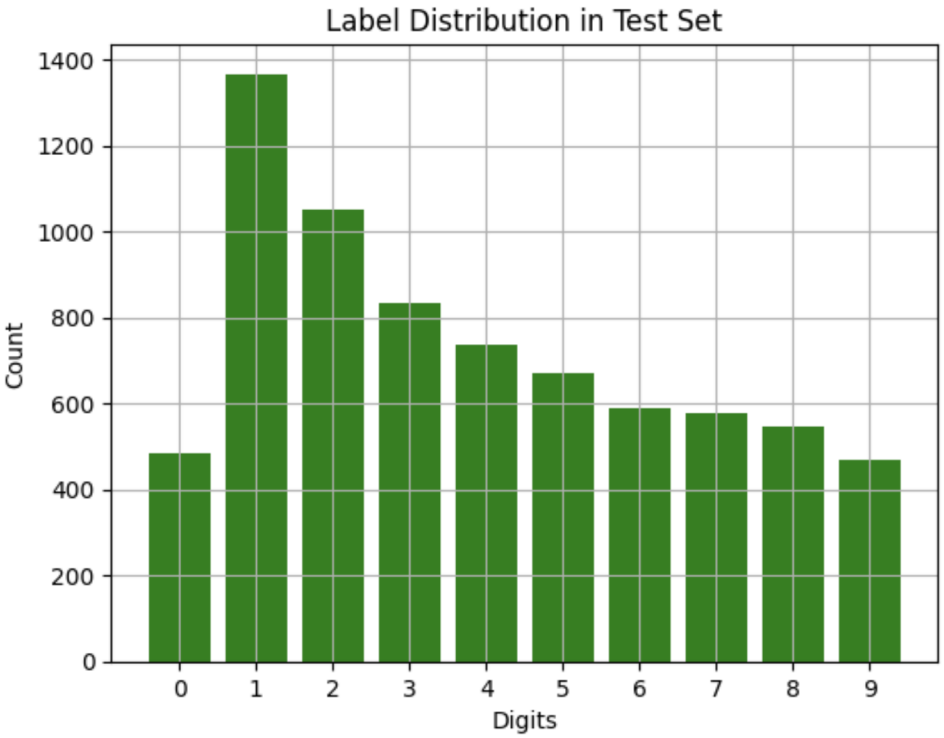


Figure 8: Distribution of class labels in the test set

3.3 Visualization of Training Data Samples

We visualized a subset of unique training samples, demonstrating the dataset's diversity and verifying correct preprocessing.

Figure 9: Five unique samples from the training data with their corresponding labels

3.4 Model Training and Hyperparameter Tuning

3.4.1 Data Preprocessing for MLP

Flattening the 2D image data into 1D vectors and standardizing them ensured that the MLP input was appropriately normalized, improving algorithmic performance. The data was then standardized using StandardScaler to have a mean of zero and a standard deviation of one

3.4.2 MLP Classifier and Pipeline Configuration

An MLPClassifier from the scikit-learn library was utilized to create a neural network with two hidden layers. To facilitate the process, a pipeline incorporating both scaling and classification tasks was established. The network architecture was designed with the first hidden layer containing 100 neurons and the second comprising 50 neurons, with the training set to run for up to 10 iterations.

3.4.3 Hyperparameter Optimization via Grid Search

Grid search, constrained by computational resources, led us to determine optimal hyperparameters, including batch size, learning rate, and L2 regularization term, through a cross-validated process.

3.4.4 Grid Search Results

The systematic grid search pinpointed optimal hyperparameters for the MLP, which are enumerated below:

- Alpha (L2 regularization term): 0.001
- Batch size: 128
- Initial Learning Rate: 0.001

These hyperparameters were established as the most conducive for training the MLP according to the outcomes of the cross-validation grid search.

3.4.5 Experimental Configuration

An MLP classifier was rigorously trained on the SVHN dataset, utilizing a variety of activation functions. Each function was evaluated across 10 training cycles to determine its impact on the classifier's efficacy.

3.4.6 Evaluation of Activation Functions and Loss Metrics

A suite of activation functions was employed during the training process, including:

- Logistic (Sigmoid): Traditionally leveraged for binary classification problems.
- ReLU: Facilitates the passage of positive signals while negating non-positive values.
- Tanh: Essentially a rescaled logistic function that outputs a wider range of output values.
- Identity: Directly transmits the input as output, remaining unaltered.

We scrutinized the progression of training and validation losses over successive epochs to gauge the learning process and the classifier's adaptability. The ensuing observations were:

1. Logistic Activation: Demonstrated a consistent decline in losses, reflective of a model well-tuned to the data.
2. ReLU Activation: Mirrored the logistic function's loss reduction trend, underscoring a steady learning trajectory.
3. Tanh Activation: Showcased an initial sharp loss decrease, leveling off subsequently, which could indicate efficient initial learning.
4. Identity Activation: Revealed a relatively static loss, suggesting its ineffectiveness for this dataset's complex patterns.

3.4.7 Conclusive Insights

Our findings from the MLP classifier's performance with various activation functions revealed differential learning efficiencies. The logistic and ReLU functions exhibited commendable performance, tanh signaled potential that warrants additional exploration, and the identity function's inadequacy suggested the superiority of non-linear alternatives for this classification task.

3.4.8 Graphical Representations

We illustrate the training and validation loss across epochs for each activation function

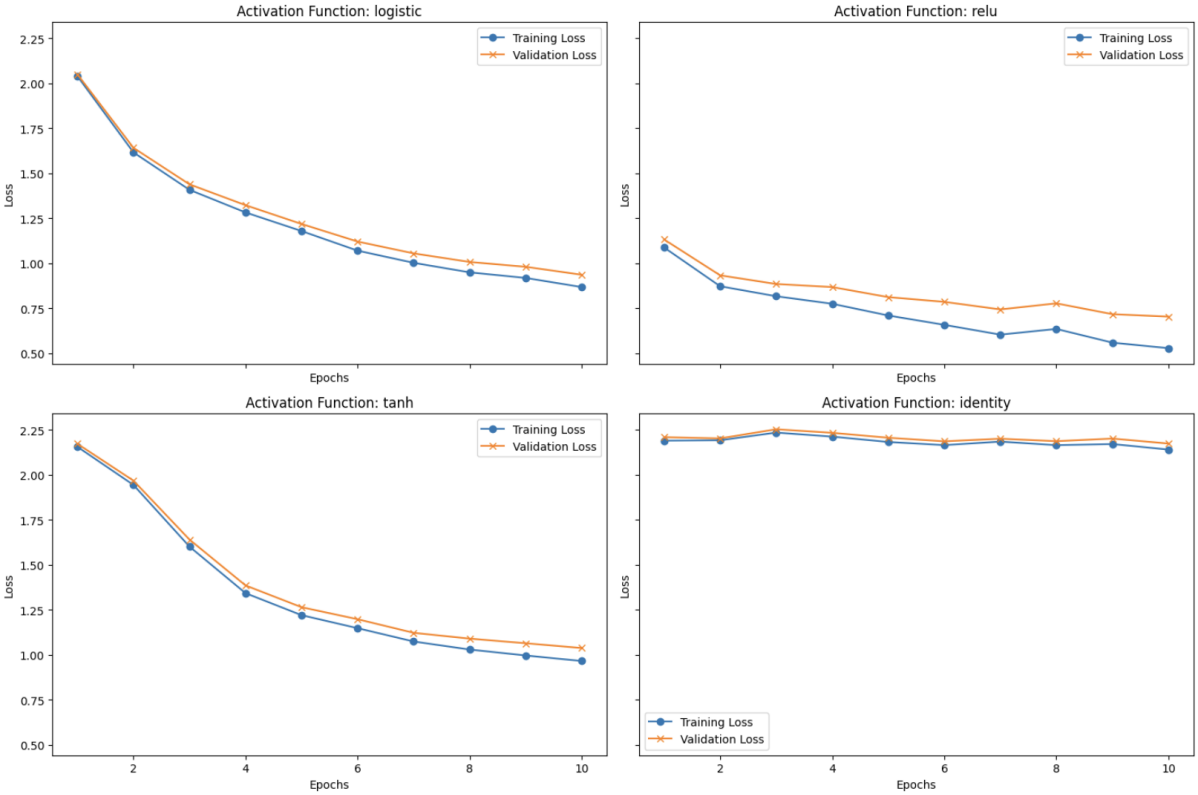


Figure 10: Training and Validation Loss with various activation functions

3.5 Evaluation on Test Set

3.5.1 Accuracy Determination on Test Set

Post-training with optimal hyperparameters, the MLP classifier's efficacy was gauged on the test set utilizing various activation functions, culminating in the following accuracy metrics:

- Logistic (Sigmoid): Secured a test accuracy of 71.39%.
- ReLU: Surpassed other functions with the highest test accuracy of 79.30%.
- Tanh: Managed a test accuracy of 67.89%.
- Identity: Recorded the lowest test accuracy of 23.79%.

3.5.2 Optimal Activation Function

Among the tested functions, ReLU emerged as the most proficient, aligning with prevalent trends in deep neural network training. Its ability to enhance model performance is well-acknowledged in the field.

3.5.3 Contributing Factors to Model Proficiency

The model's satisfactory level of precision can be linked to the ReLU activation function's robustness. The achievement stems from several factors:

1. Implementing a grid search method for the meticulous fine-tuning of hyperparameters resulted in the optimal selection of the learning rate, batch size, and regularization coefficient.

2. Normalizing the input data to ensure uniform influence across all input features, which is a fundamental step in data preprocessing.

3. Electing an activation function adept at averting the vanishing gradient dilemma, thus enhancing the speed of the learning phase.

3.5.4 Overall Model Assessment

The MLP classifier exhibited differential performance across various activation functions. The superior results with ReLU underscore its appropriateness for the dataset and architectural framework in question. This investigation underlines the critical role of methodical hyperparameter optimization and data preparation in bolstering the MLP's learning efficiency and test accuracy.

3.5.5 Tabulated Test Accuracies

Table 1: Test accuracies for different activation functions

Activation Function	**Test Accuracy**
Logistic	0.713896
ReLU	0.793066
Tanh	0.678952
Identity	0.237920

3.6 Visualization and Analysis of Misclassified Images

3.6.1 Missclassification Visualisaton

To assess the model's accuracy, we focused on cases where its predictions diverged from the true labels. We chose three incorrectly predicted images for each digit class, from 0 to 9, and displayed these to highlight the errors made by the model.

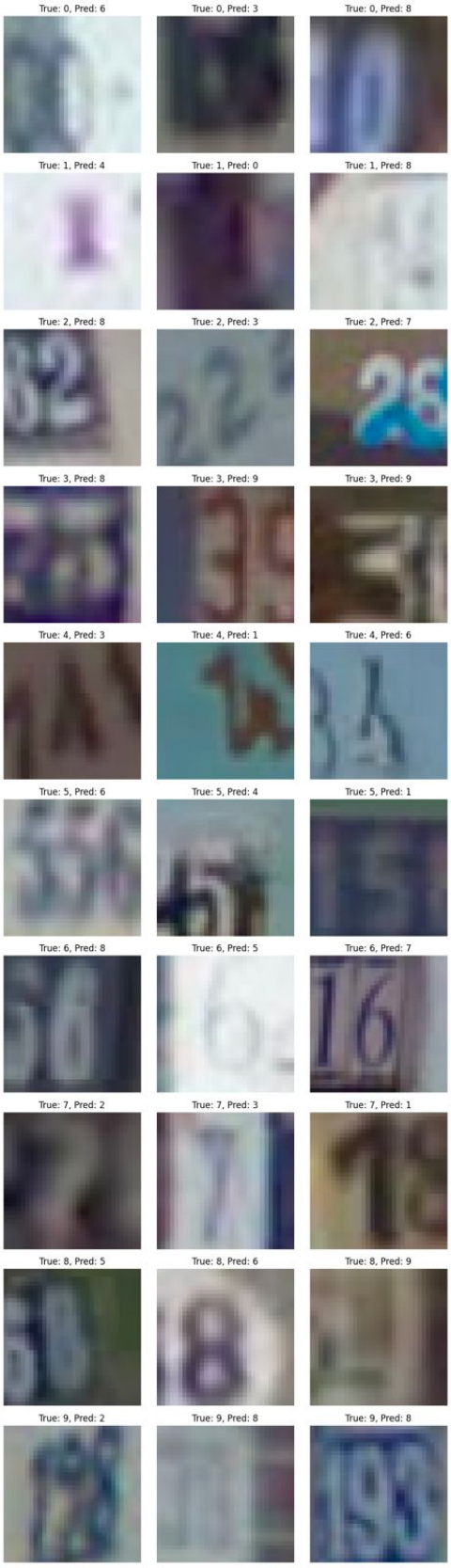


Figure 11: Three misclassified images for each class with their predicted labels

3.6.2 Confusion Mattrix

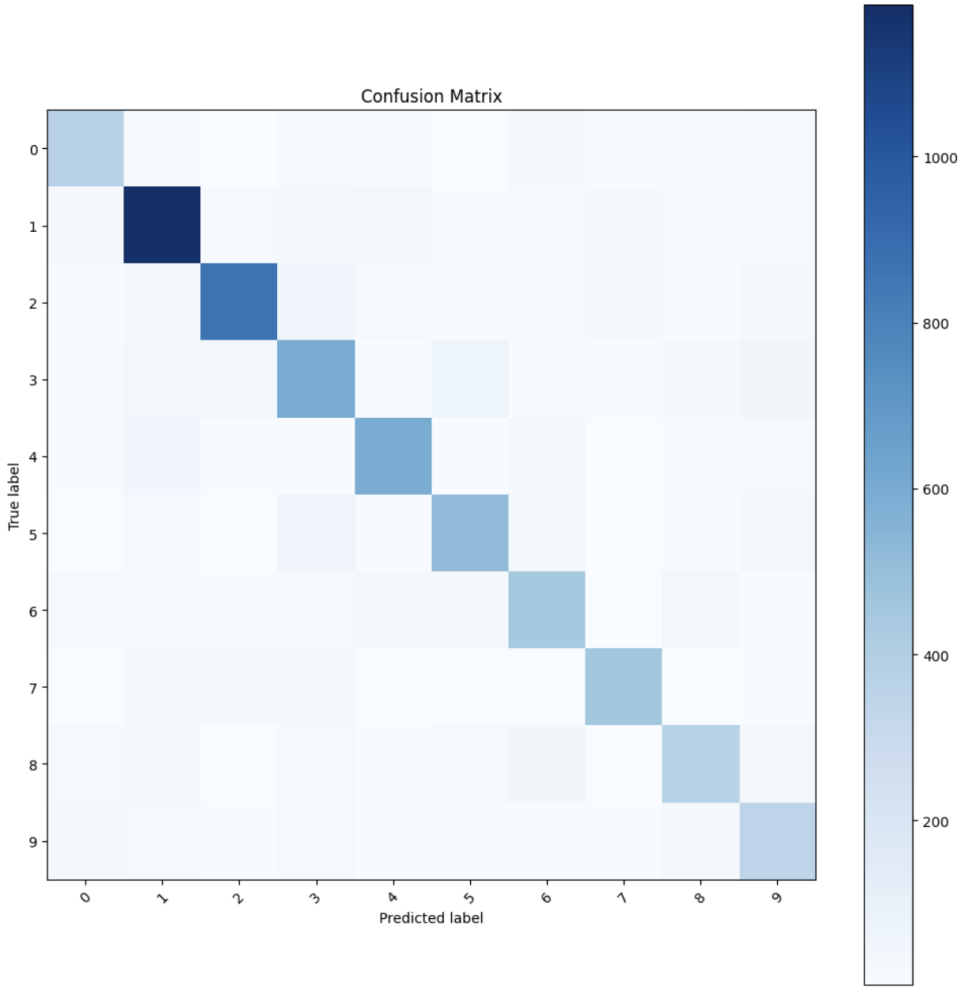


Figure 12: Confusion Mattrix

The confusion matrix provided displays a strong diagonal trend, reflecting a high number of correct predictions by the model for most classes. This pattern signifies effective classification for the majority of instances. The concentration of darker cells along the diagonal line is indicative of the model's ability to accurately identify and categorize the majority of the digits presented to it.

However, the presence of non-negligible darker shades in off-diagonal elements points to consistent misclassification errors for certain digit pairs. These errors are not randomly distributed but seem to occur more between particular classes, suggesting that the model may be mistaking digits that share visual similarities. The lighter shades in the off-diagonal cells imply that such errors are less frequent, yet their consistency warrants further investigation to improve the model's discriminative power between these challenging classes.

3.6.3 Analysis of Misclassifications

The review of images misclassified by the model highlighted a few key factors contributing to the errors in prediction:

- **Image Quality:** Some pictures were not clear, too small, or partially hidden, challenging the model and even people to recognize the numbers correctly.
- **Unclear Digits:** A few numbers were written in a way that they could be mistaken for other numbers, like a '7' that looks much like a '1'.
- **Contextual Cues:** Missing the surrounding information that people usually rely on could cause the model to label the images wrong.
- **Model Constraints:** The basic design of the MLP, which lacks the ability to process image layers like CNNs, might be why it doesn't do as well in sorting images.