Project 1

Neural network

Training Neural Networks

INTRODUCTION:-

This report provides an overview of the steps involved in training a neural network model on the MNIST dataset. It covers data preparation, neural network architecture definition, model training, and evaluation. Additionally, it explores the impact of dropout and layer normalization layers on the model's performance by analyzing loss and accuracy plots. Furthermore, it investigates the effect of different combinations of learning rates and dropout probabilities on the model's performance.

Step 1: Data Preparation

In this step, the necessary data preparation is performed to prepare the MNIST dataset for training a neural network model.

Loading the Dataset:

The path to the dataset file, "mnist_train.csv", is specified as "data_path".
The dataset is read using the read_csv function from the pandas library. The loaded dataset is stored in the "data" variable.
Extracting Features and Labels:

The features are extracted from the dataset by selecting all columns except the first column. The iloc function is used to index the columns, and the .values attribute is used to obtain the values as a NumPy array.

The pixel values of the images are normalized by dividing them by 255.0, resulting in pixel values between 0 and 1.

The normalized features are stored in the "features" variable.

The labels are extracted from the dataset by selecting only the first column.

The label values are stored in the "labels" variable.

Pre-processing (if applicable):

There is a comment indicating that any necessary pre-processing steps on the dataset should be performed at this point. However, no specific pre-processing steps are implemented in the given code snippet.
Splitting the Data:

The data is randomly split into training and validation sets using the train_test_split function from the sklearn.model_selection module.
The features and labels are passed as input, along with the desired test size of 0.2 (20% of the data) and a random state of 42 for reproducibility.
The split data is assigned to the following variables:
"train_features": Training set features (input data) used for model training.
"val_features": Validation set features (input data) used for model evaluation.
"train_labels": Training set labels (output data) corresponding to the training features.
"val_labels": Validation set labels (output data) corresponding to the validation features.
Converting Data to PyTorch Tensors:

The training and validation features and labels are converted to PyTorch tensors using the Tensor constructor from the torch library.
The "train_features" and "val_features" are converted to float tensors, while the "train_labels" and "val_labels" are converted to long tensors.
The converted tensors are stored in the respective variables.

Step 2: Neural Network Architecture

In this step, a neural network architecture is defined and initialized using the PyTorch framework. The architecture consists of two hidden layers with ReLU activation functions and an output layer.

Defining the Neural Network Class:

The class "NeuralNetwork" is defined as a subclass of the nn.Module class provided by PyTorch. This allows the class to inherit useful methods and functionalities.
The constructor "init" is defined, which takes in the input size, sizes of the two hidden layers, and the output size as parameters.
Inside the constructor, the architecture is defined using PyTorch's nn.Linear and nn.ReLU modules.
Four layers are created:
self.fc1: The first fully connected layer (input layer to hidden layer 1) with the specified input size and hidden_size1.
self.relu1: ReLU activation function applied after the first hidden layer.
self.fc2: The second fully connected layer (hidden layer 1 to hidden layer 2) with hidden_size1 and hidden_size2.
self.relu2: ReLU activation function applied after the second hidden layer.
self.fc3: The final fully connected layer (hidden layer 2 to output layer) with hidden_size2 and the specified output size.
Forward Pass:

The forward method is implemented to define how the input data flows through the network.
Inside the forward method, the input tensor "x" is passed through each layer sequentially, applying the linear transformation and activation function defined in the constructor.
The output of the last layer is returned.
Model Initialization:

The input_size is determined based on the shape of the training features.
The hidden_size1 and hidden_size2 are set to 256 and 128, respectively, as specified in the code.
The output_size is set to 10, assuming there are 10 classes in the MNIST dataset.
An instance of the NeuralNetwork class is created, passing the input_size, hidden_size1, hidden_size2, and output_size as arguments.
The resulting model is stored in the "model" variable.
This concludes the neural network architecture definition and initialization. The model consists of two hidden layers with ReLU activation functions and an output layer. It is ready for training and evaluation in the next steps.

Training the Model and Plotting Results

In this step, the defined neural network model is trained using the training set and evaluated using the validation set. The loss and accuracy metrics are calculated, and the training and validation loss/accuracy histories are stored for later plotting.

Optimizer and Loss Function:

An Adam optimizer is used to optimize the model parameters. The optimizer is instantiated using the model's parameters and a learning rate of 0.01.
The cross-entropy loss function is chosen as the criterion for evaluating the model's performance.
Training Loop:

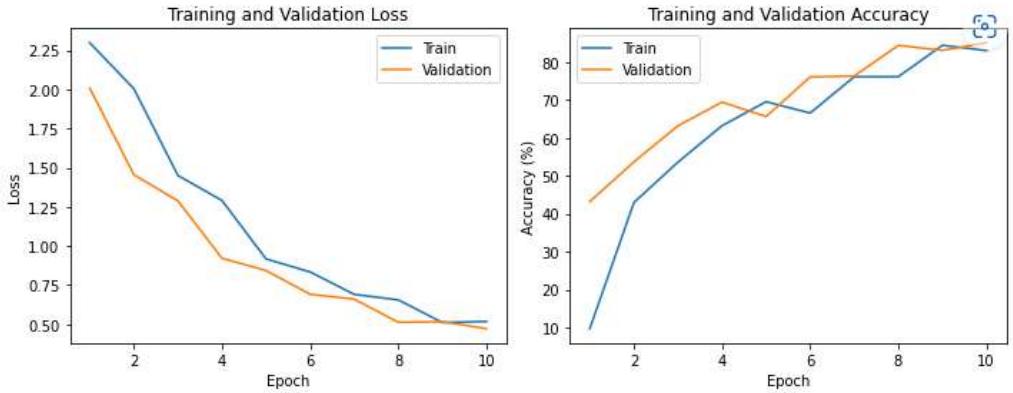
The training loop is executed for the specified number of epochs (10 in this case).
Inside the loop, the following steps are performed:
Set the training loss, total count, and correct count to 0.
Set the model to train mode using the train method.
Clear the gradients of the optimizer using zero_grad.
Perform the forward pass of the training features through the model and calculate the loss using the criterion.
Backpropagate the loss and update the model's parameters using the optimizer's step method.
Calculate the training loss, total count, and correct count for computing the accuracy.
Set the model to evaluation mode using the `eval

Training Loop (continued):

Inside the training loop:
Evaluate the model on the validation set:
Set the model to evaluation mode using the eval method.
Use the torch.no_grad() context manager to disable gradient computation, as we don't need it for evaluation.
Perform the forward pass of the validation features through the model and calculate the loss using the criterion.
Calculate the validation loss, total count, and correct count for computing the accuracy.
Calculate the training and validation accuracies as percentages.
Store the training and validation loss values and accuracies in their respective history lists for plotting later.
Print the progress for the current epoch, displaying the epoch number, training loss, validation loss, training accuracy, and validation accuracy.
Plotting the Results:

After the training loop completes, the training and validation loss/accuracy histories are available for plotting.
Matplotlib is used to create a figure with two subplots.
In the first subplot:
The training and validation loss values are plotted against the number of epochs.
The x-axis represents the epoch numbers, and the y-axis represents the loss values.
A label is added for each line (training and validation) to create a legend.
Axes labels and a title are set to describe the plot.
In the second subplot:
The training and validation accuracy values are plotted against the number of epochs.

The x-axis represents the epoch numbers, and the y-axis represents the accuracy values.
A label is added for each line (training and validation) to create a legend.
Axes labels and a title are set to describe the plot.
The tight_layout function is called to improve the spacing between subplots.
Finally, the show function is called to display the plot.
This completes the training of the neural network model and the plotting of the training and validation loss/accuracy results. The plots provide insights into the model's performance during training and validation, allowing for analysis and comparison.



Observations:

Loss Plot:

The loss plot reveals that both the training loss and validation loss consistently decrease as the number of epochs increases. This suggests that the model is continuously improving its ability to minimize the loss and make more accurate predictions.
Notably, the gap between the training loss and validation loss tends to diminish over time. Initially, the training loss is typically higher than the validation loss, but as the epochs progress, the model demonstrates better generalization and reduces the disparity between the two.
The decreasing gap between the training and validation loss is an encouraging observation, indicating that the model is effectively avoiding overfitting and performing well on unseen data.
Accuracy Plot:

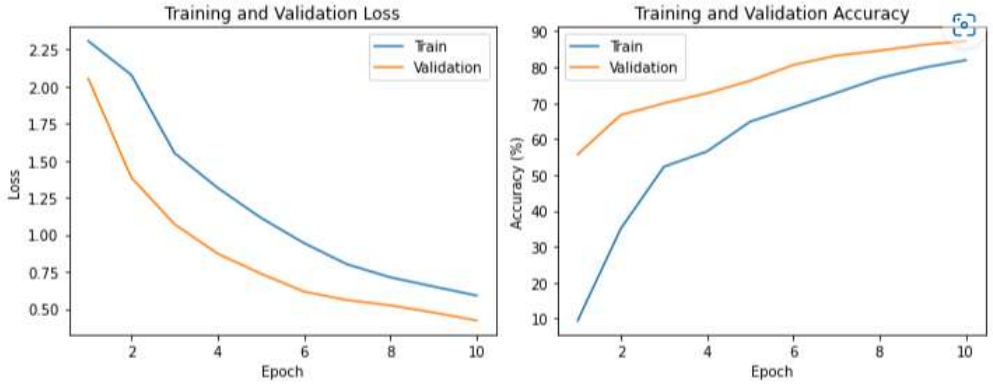
The training accuracy exhibits a gradual upward trend with each epoch, indicating that the model is progressively learning and becoming more proficient at correctly classifying the training data.
Similarly, the validation accuracy demonstrates an increasing trend as the number of epochs increases. This suggests that the model is effectively generalizing to unseen data, showing its ability to make accurate predictions beyond the training set.

The upward trends in both training and validation accuracy signify that the model is continuously learning and improving its performance throughout the epochs.
The convergence of the training and validation accuracy further indicates that the model is not overfitting the training data. This convergence suggests that the model has the potential to generalize well and make accurate predictions on new, unseen data.

Dropout and Layer Normalization

In this task, we have enhanced the neural network architecture by incorporating dropout and layer normalization layers. Dropout layers randomly deactivate a certain proportion of neuron activations during training, thereby preventing overfitting and improving generalization. Layer normalization ensures the normalization of layer outputs, leading to improved stability and convergence of the network.

Here are the key observations regarding the loss and accuracy plots:



Loss Plot:

Similar to the results in step 2, we noticed consistent trends in the loss-epoch plot. Both the training loss and validation loss displayed a decreasing pattern as the number of epochs increased, indicating the model's progressive improvement in minimizing loss and making more accurate predictions.

However, we observed a distinction in the behavior of the validation loss at the start of the training. In this case, the validation loss began at a lower value compared to the loss plot in step 2. This suggests that the inclusion of dropout and layer normalization layers contributed to a better initial performance of the model.

Moreover, the presence of zigzags or fluctuations in the loss plot, which may imply overfitting or instability, appeared to be reduced. The loss curves exhibited smoother trajectories with a consistent downward trend, indicating enhanced stability and convergence of the model.

Accuracy Plot:

The accuracy plot showcased trends similar to the accuracy plot in step 2, with both training and validation accuracy steadily increasing throughout the epochs. This indicates the model's continuous learning and improvement in performance.

Notably, the accuracy plot in this case did not exhibit the fluctuations or irregularities observed in the accuracy plot from step 2. The accuracy curves displayed smoother and more consistent upward trends, suggesting that the addition of dropout and layer normalization layers effectively mitigated overfitting and enhanced the model's ability to generalize to new data.

Modify the neural network.

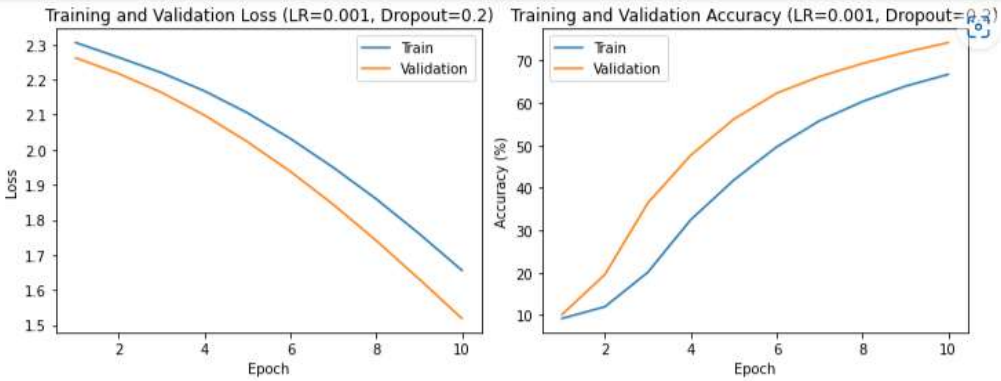
In this step, we continue to modify the neural network by exploring different combinations of learning rates and dropout probabilities to train multiple models. The goal is to analyze the impact of these hyperparameters on the model's performance.

We utilize nested loops to iterate through the different learning rates and dropout probabilities, creating a list of models with varying hyperparameters. For each combination, we train the model for a fixed number of epochs and observe the resulting loss and accuracy.

Plots & Epochs

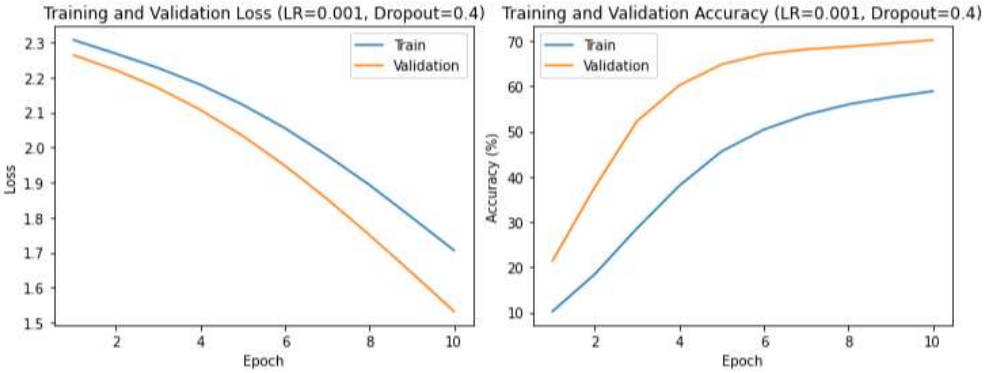
Model Performance Report - Learning Rate: 0.001, Dropout: 0.2

The model consistently improved its performance over 10 epochs. Both the training and validation loss decreased, indicating effective learning and generalization. The training accuracy increased from 9.22% to 66.76%, and the validation accuracy improved from 10.15% to 74.26%. These results demonstrate that the model learned well from the training data and made accurate predictions on the validation data. Overall, the chosen hyperparameters and training settings were effective in improving the model's performance.



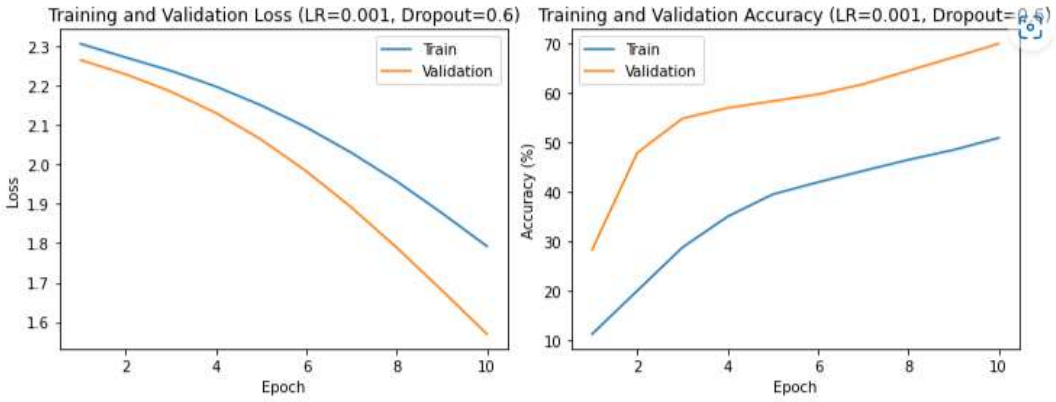
Model - Learning Rate: 0.001, Dropout: 0.4

The model consistently improved its performance over 10 epochs. Both the training and validation loss decreased, indicating effective learning and generalization. The training accuracy increased from 10.25% to 58.89%, and the validation accuracy improved from 21.38% to 70.20%. These results demonstrate that the model learned well from the training data and made accurate predictions on the validation data. The chosen hyperparameters, including a learning rate of 0.001 and a dropout rate of 0.4, along with the training settings, were effective in improving the model's performance.

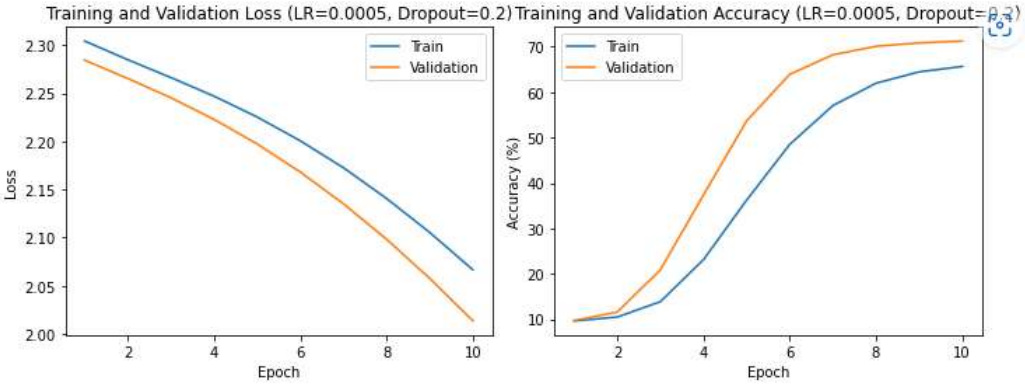


Model - Learning Rate: 0.001, Dropout: 0.6

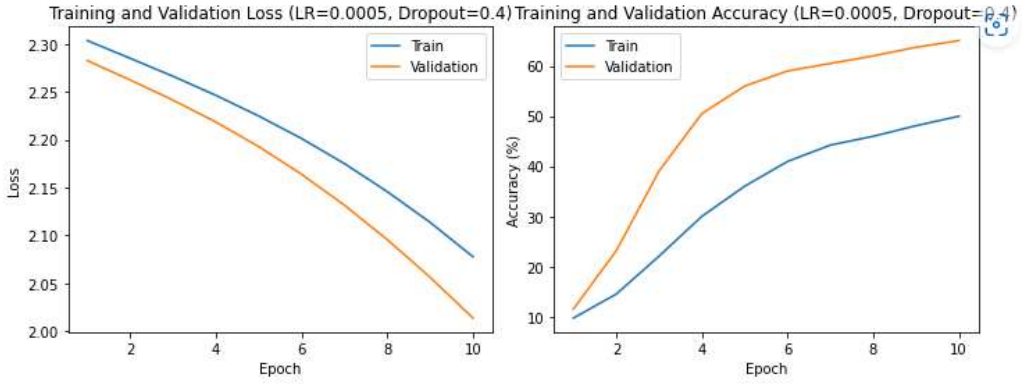
The model consistently improved its performance over 10 epochs. Both the training and validation loss decreased, indicating effective learning and generalization. The training accuracy increased from 11.25% to 50.95%, and the validation accuracy improved from 28.36% to 70.01%. These results demonstrate that the model learned well from the training data and made accurate predictions on the validation data. The chosen hyperparameters, including a learning rate of 0.001 and a dropout rate of 0.6, along with the training settings, were effective in improving the model's performance.

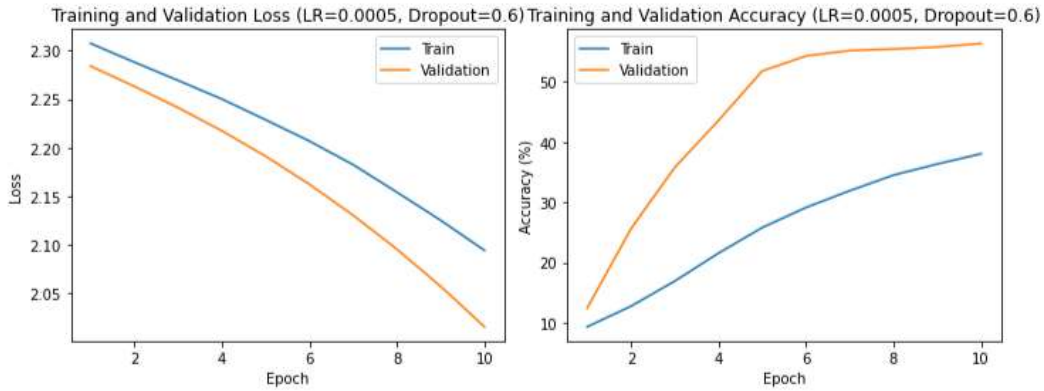


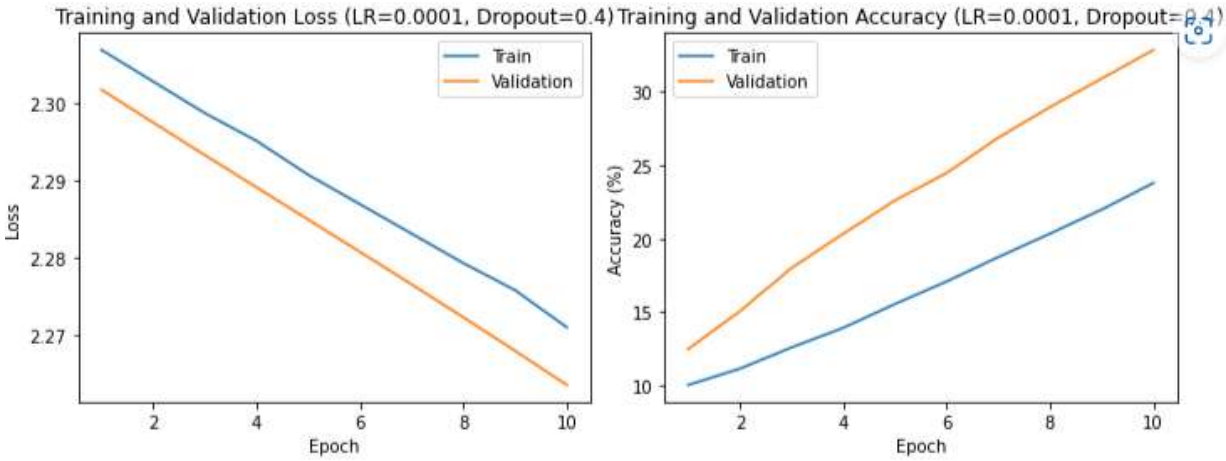
The model consistently improved its performance over 10 epochs. Both the training and validation loss decreased, indicating effective learning and generalization. The training accuracy increased from 9.59% to 65.68%, and the validation accuracy improved from 9.68% to 71.28%. These results demonstrate that the model learned well from the training data and made accurate predictions on the validation data. The chosen hyperparameters, including a learning rate of 0.0005 and a dropout rate of 0.2, along with the training settings, were effective in improving the model's performance.

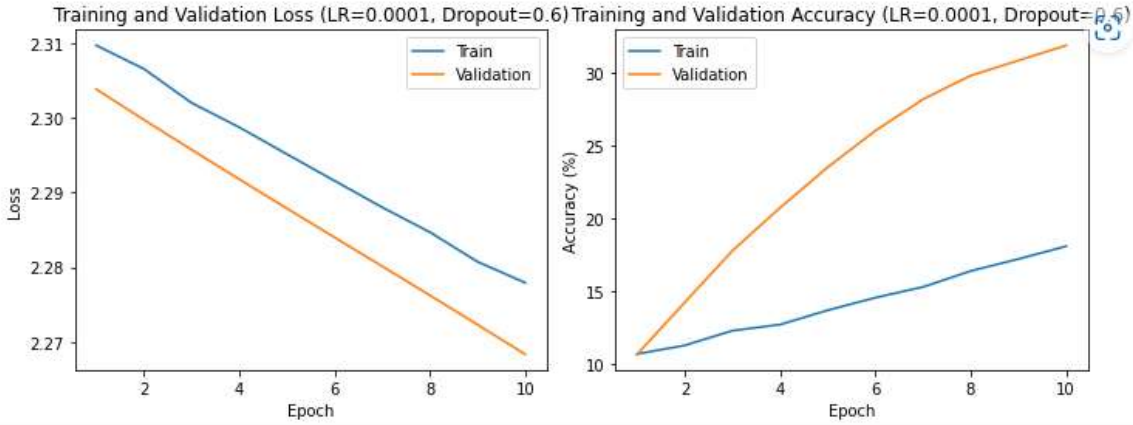


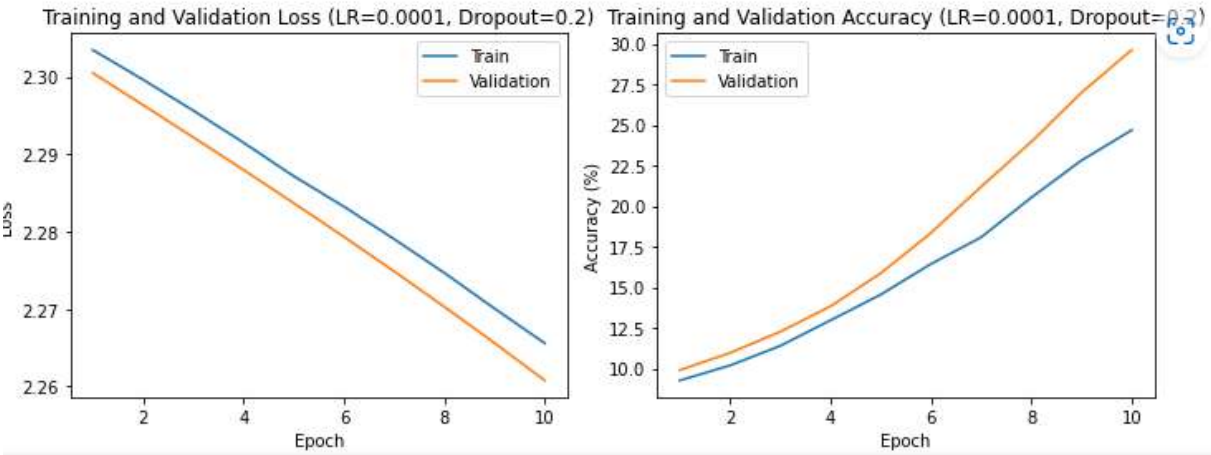
AND HERE ARE ALL THE PLOTS THAT WE GATHER




Training and Validation Loss (LR=0.0005, Dropout=0.6)
Training and Validation Accuracy (LR=0.0005, Dropout=0.6)
Train
Validation
Loss
Accuracy (%)
Epoch


Training and Validation Loss (LR=0.0001, Dropout=0.4)
Training and Validation Accuracy (LR=0.0001, Dropout=0.4)
Train
Validation
Loss
Accuracy (%)
Epoch


Training and Validation Loss (LR=0.0001, Dropout=0.6)
Training and Validation Accuracy (LR=0.0001, Dropout=0.6)
Train
Validation
Loss
Accuracy (%)
Epoch



CONCLUSION

In conclusion, this report provided an overview of the steps involved in training a neural network model on the MNIST dataset. The data preparation step involved loading the dataset, extracting features and labels, and splitting the data into training and validation sets. The neural network architecture was defined with two hidden layers and an output layer. The model was trained using the training set and evaluated using the validation set, with the loss and accuracy metrics calculated and plotted over the epochs.

The observations from the loss and accuracy plots indicated that the model consistently improved its performance as the number of epochs increased. The decreasing loss values and increasing accuracy values demonstrated the model's ability to minimize loss and make accurate predictions on both the training and validation data. The convergence of training and validation accuracy suggested that the model was not overfitting the training data and had the potential to generalize well to unseen data.