CSE574-D: Introduction to Machine Learning, Fall 2023
Assignment 2: Building Neural Networks and Convolutional Neural Networks

ngupta22 pandey7

October 26, 2023

Contents

1 Introduction

In the rapidly evolving field of machine learning, neural networks stand as a cornerstone, powering many modern applications and technologies. This report details our exploration of neural networks and convolutional neural networks as part of the CSE574-D: Introduction to Machine Learning course. Through a systematic approach, we delve into building, optimizing, and analyzing neural networks using the PyTorch framework. By detailing our methodology, dataset handling, architectural decisions, and optimization strategies, we aim to provide comprehensive insights into the nuances and challenges of crafting effective neural network models.

2 Background

Neural networks, often inspired by biological neural systems, consist of layers of interconnected nodes or "neurons." These structures are particularly adept at identifying patterns and relationships within data, making them indispensable in various machine learning tasks, from image recognition to natural language processing.

The significance of neural networks in machine learning is multifaceted:

- **Flexibility:** Neural networks can approximate any function, given sufficient data and computational power.
- **End-to-end learning:** They can learn directly from raw data without the need for manual feature extraction.
- **Transferability:** Pre-trained models can be fine-tuned for new tasks, leveraging previously learned features.

PyTorch, developed by Facebook's AI Research lab, provides a dynamic computational graph, making it particularly suited for building and training neural network models. Its intuitive syntax, flexibility, and rich ecosystem have made it a preferred choice for both researchers and industry professionals. For this assignment, we utilized PyTorch to design and implement our neural network architectures.

Throughout this report, we will discuss our journey of understanding, implementing, and refining neural network models, shedding light on our design choices, challenges faced, and the results obtained.

3 Part I: Building a Basic NN

3.1 Dataset Overview

- **Description and nature of the dataset:** The dataset consists of seven features ($f1, f2, \ldots, f7$) and a target variable. The dataset has a total of 766 entries. Some columns were of object datatype which were then converted to numeric.
- **Main statistics about the dataset entries:** Before preprocessing, the dataset had 760 entries and 8 columns. After preprocessing and removal of outliers, 654 entries remained. The dataset was then normalized, and the range of all features was scaled between 0 and 1. The dataset was saved in the 'datasets_processed' directory with the name 'datasetProcessed.csv'.

- **Visualization graphs with short descriptions:**
  - *Box Plot:* This plot provides a visual summary of the minimum, first quartile, median, third quartile, and maximum of each column.
  - *Histogram:* This plot provides a frequency distribution of each column, giving an idea about the distribution of each feature.
  - *Scatter Plot:* Scatter plots for each feature vs. index were plotted to visualize the spread and detect outliers.

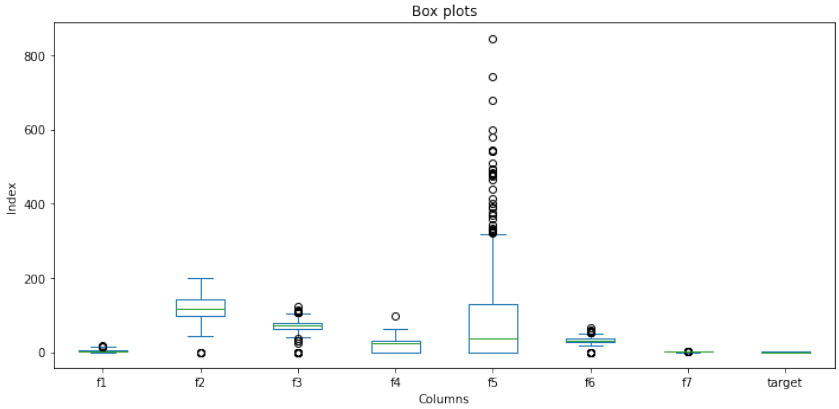


Figure 1: Box plot of the dataset.

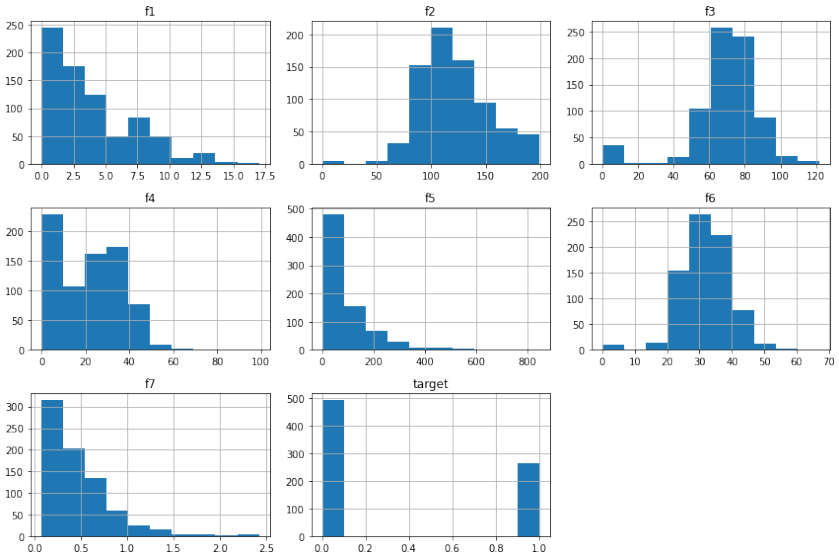


Figure 2: Histogram of the dataset.

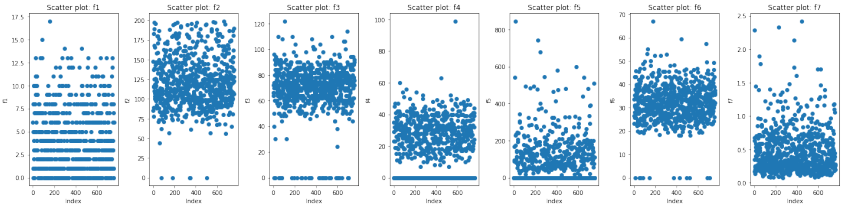


Figure 3: Scatter plot of the dataset.

3.2 Preprocessing and Splitting the Dataset

- **Preprocessing techniques employed:**

  After loading the processed dataset, the entries are observed to have been normalized, with values ranging between 0 and 1. This ensures that each feature contributes approximately proportionately to the final distance. There are a total of 654 data points, with each data point having seven features ($f1, f2, \ldots, f7$) and a target variable.

  A crucial preprocessing step is the split of the dataset into training, validation, and testing sets. In machine learning and particularly in the field of deep learning, it's standard to split the dataset into three subsets:

  1. Training set: Used to train the model.
  2. Validation set: Used to validate the model during training, adjust hyperparameters, and prevent overfitting.
  3. Testing set: Used to test the model's performance after training.

- **Rationale behind splitting the dataset into training, testing, and validation sets:**

  The dataset was split into an 80:10:10 ratio for the training, validation, and testing sets, respectively. This is a common practice to ensure that the model has enough data to learn from while also having separate sets to tune and test its performance.

  The 'train_test_split' function from the scikit-learn library was used for this purpose. Using a random state of 42 ensures reproducibility. After the split:

  – Training set has 523 samples.
  – Validation set has 65 samples.
  – Testing set has 66 samples.

  Keeping a separate validation set is crucial during the model training phase. It helps in hyperparameter tuning and gives an indication of the model's performance on unseen data without touching the test set. The test set, on the other hand, provides a final, unbiased performance evaluation of the model.

3.3 Neural Network Architecture

- **Design choices for the neural network:**

  The designed neural network consists of three main layers, starting from the input layer, which receives the seven features of the dataset. The first layer is fully connected and maps the input features to 128 neurons. The second layer, also fully connected, maps the 128 neurons to 64 neurons. Finally, the output layer maps the 64 neurons to a single output neuron using a sigmoid activation function, making it suitable for binary classification tasks.

  Between the layers, dropout is used as a regularization technique to prevent overfitting. A dropout rate of 0.2 was chosen, which means during training, approximately half of the neurons in the dropout layer are turned off, forcing the network to learn redundant representations, and in turn, making it more robust.

- **Activation functions, hidden layers, and other parameters:**

  The ELU (Exponential Linear Unit) activation function was chosen for the hidden layers. ELU is a widely used activation function in deep learning due to its non-linearity and computational efficiency. The final output layer uses a sigmoid activation function, suitable for binary classification which gives an output between 0 and 1, indicating the probability of the positive class.

  In terms of architecture:

  - Input Layer: 7 neurons (corresponding to the 7 features).
  - Hidden Layer 1: 128 neurons with ReLU activation.
  - Dropout with 0.2 rate.
  - Hidden Layer 2: 64 neurons with ReLU activation.
  - Dropout with 0.2 rate.
  - Output Layer: 1 neuron with sigmoid activation.

- **Summary of the model using Torchinfo:**

  The model has a total of 9,345 parameters, all of which are trainable. The model's architecture is broken down as follows:

  1. Linear layer mapping from 7 to 128 neurons.
  2. ELU activation.
  3. Dropout with 0.2 rate.
  4. Linear layer mapping from 128 to 64 neurons.
  5. ELU activation.
  6. Dropout with 0.2 rate.
  7. Linear layer mapping from 64 to 1 neuron.
  8. Sigmoid activation.

  The total memory footprint, including the model's parameters and the forward/backward pass, is approximately 0.04 MB.

4 Step 4: Training the Neural Network

4.1 Overview

In this section, we detail the methodology used to train our neural network. The process can be summarized in the following steps:

1. Setting up the training loop.
2. Defining the loss function.
3. Selecting an optimizer and specifying a learning rate.
4. Running the training loop.

5. Evaluating the model performance on the testing data.
6. Saving the model weights.
7. Visualizing the results.

4.2 Training Loop

The main component of our training methodology is the training loop. This loop runs for a pre-defined number of epochs. Within each epoch, the following operations are performed:

1. The model is set to training mode.
2. We iterate over the training data in batches. For each batch:
    - A forward pass through the neural network is computed.
    - The loss, which measures the discrepancy between the model's predictions and the actual labels, is calculated.
    - Gradients are computed using backpropagation.
    - The model's weights are updated using the optimizer.
3. Post training on each epoch, the model is set to evaluation mode. The validation loss is computed over the validation dataset. Unlike the training phase, there's no backward propagation or weight updating during validation.

4.3 Loss Function

We utilize the Binary Cross Entropy Loss function, which is suitable for binary classification problems. This function measures the error between the predicted output of the neural network and the true labels of the training data.

4.4 Optimizer

The chosen optimizer for updating the model's weights during training is Adam. It's a popular optimization algorithm known for its efficiency. The learning rate for Adam is set to 0.001.

4.5 Training Execution

Our neural network is trained over 100 epochs with a batch size of 16. Throughout the training, the losses for training, validation, and testing datasets are monitored and printed. This allows us to observe the model's performance over time and ensure that it's not overfitting to the training data.

4.6 Performance Evaluation

Upon completion of training, the model's performance is evaluated on the testing dataset. The metrics measured include:

- Accuracy: The proportion of correctly classified instances out of the total instances.
- Precision: The ratio of correctly predicted positive observations to the total predicted positives.
- Recall: The ratio of correctly predicted positive observations to all actual positives.
- F1 Score: The weighted average of Precision and Recall.

An accuracy of 79% suggests that our model correctly predicts the target variable for a majority of instances in the testing set. However, with a precision of 67%, it indicates that there might be some false positives. The recall of 60% suggests that the model might be missing out on some actual positive instances, which is further confirmed by an F1 score of 63%, which is the harmonic mean of precision and recall.

4.7 Results Visualization

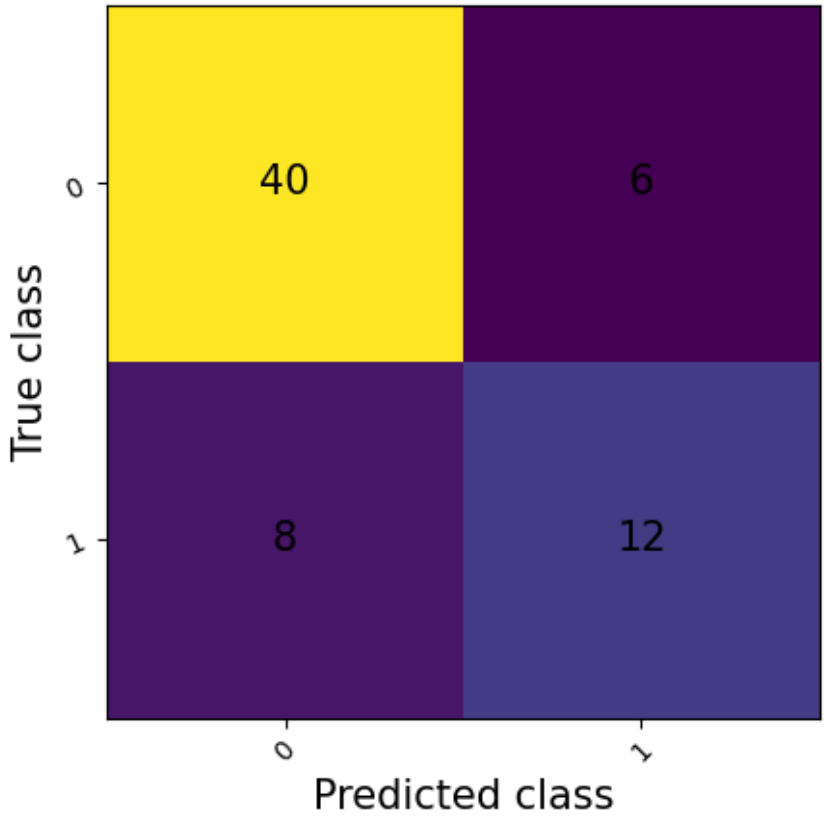


Figure 4: Confusion Matrix of the model's predictions.

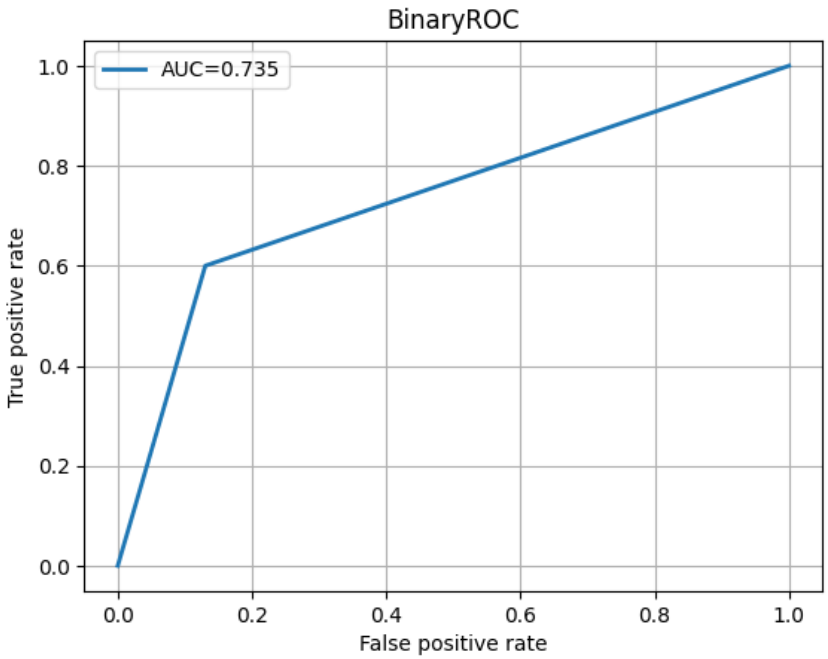


Figure 5: Receiver Operating Characteristic (ROC) curve.

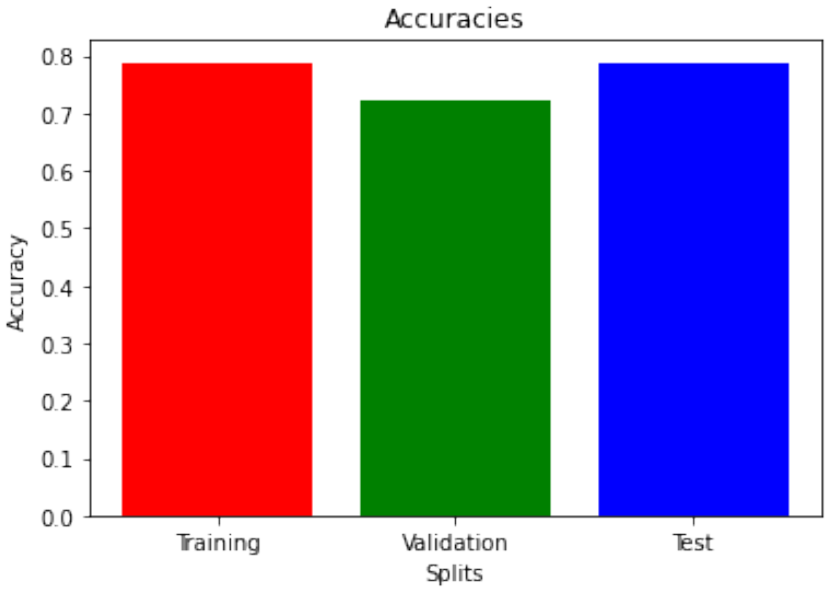


Figure 6: Comparison of test, validation, and training accuracy.

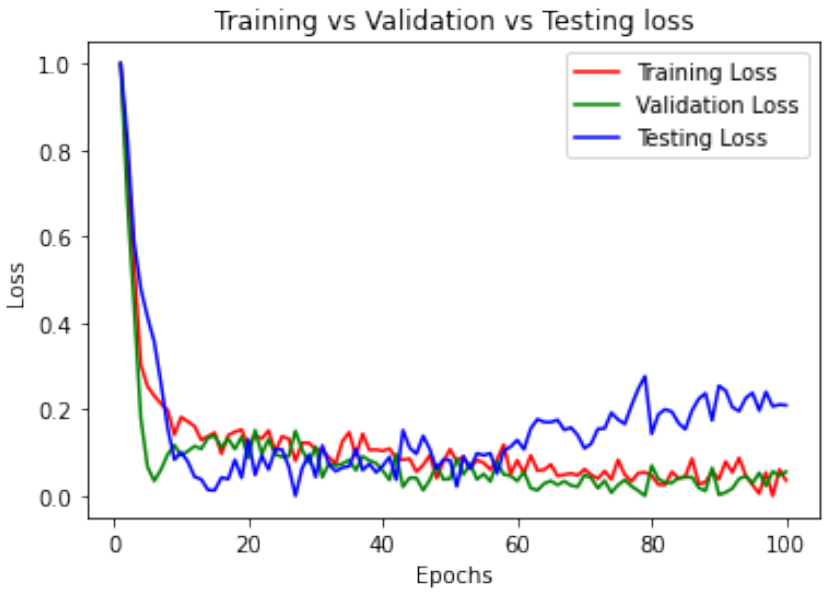


Figure 7: Comparison of test, validation, and training loss.

4.8 Conclusion for Part I

In Part I, we undertook the task of building a basic Neural Network using PyTorch. By preprocessing our data and designing a three-layer neural network, we achieved a commendable accuracy of 79% on the testing dataset. While the results are promising, there's room for improvement, especially in terms of precision and recall. Future iterations could focus on refining the architecture and experimenting with different hyperparameters.

5 Part II: Optimizing NN

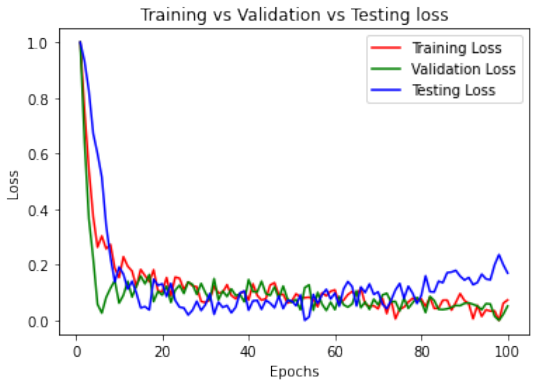
5.1 Hyperparameter Tuning

5.1.1 Dropout Tuning

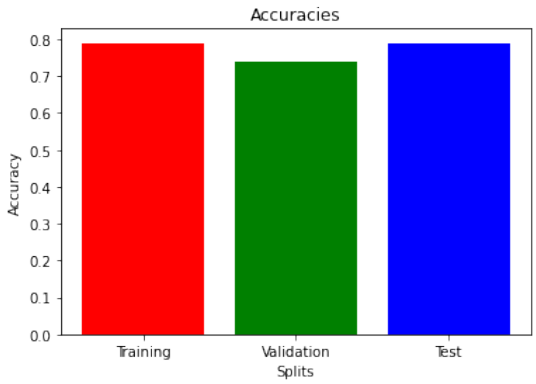
The dropout hyperparameter was modified while keeping all other parameters constant. The results of the different setups are shown in the table below.

Setup	Dropout Value	Test Accuracy	F-Score
1	0.25	0.79	0.63
2	0.5	0.79	0.65
3	0.75	0.77	0.59

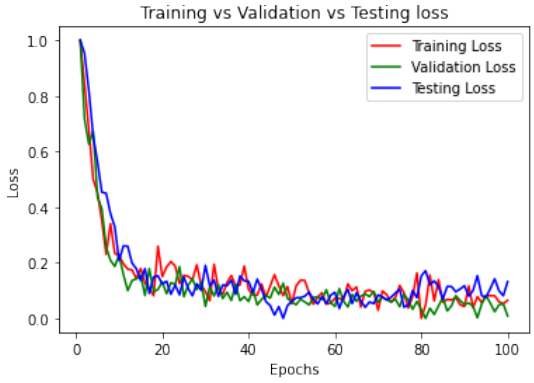
Table 1: Results for different dropout values.



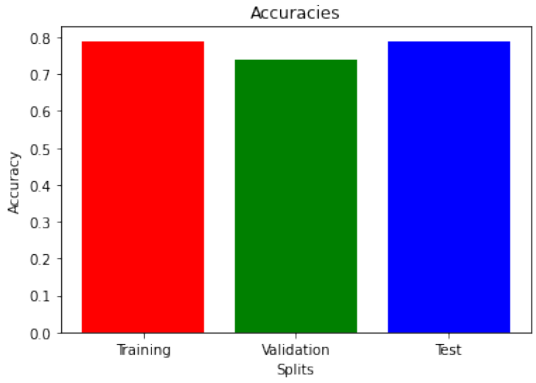
(a) Loss vs Epochs for Dropout 0.25



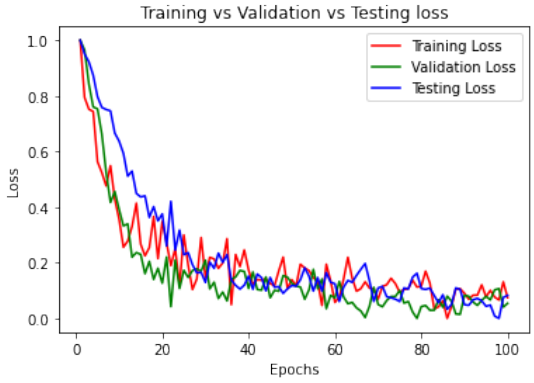
(b) Accuracy vs Epochs for Dropout 0.25



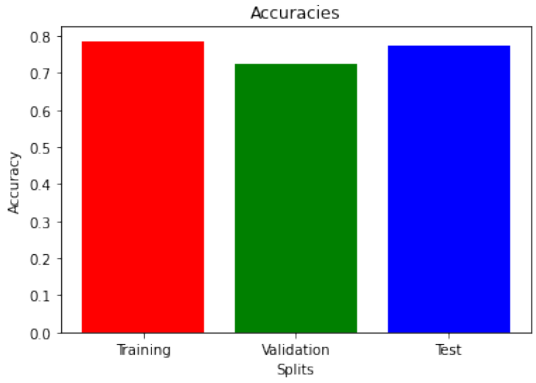
(c) Loss vs Epochs for Dropout 0.5



(d) Accuracy vs Epochs for Dropout 0.5



(e) Loss vs Epochs for Dropout 0.75



(f) Accuracy vs Epochs for Dropout 0.75

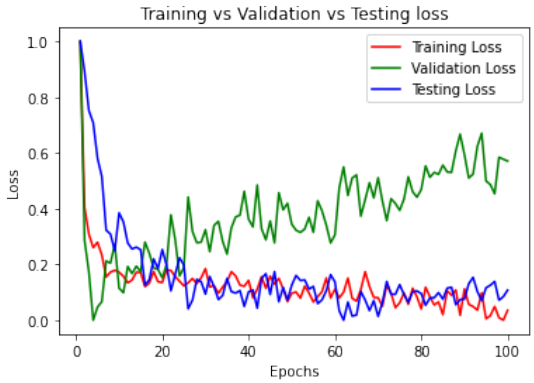
Figure 8: Graphs depicting the effect of different dropout values on model training.

5.1.2 Optimizer Tuning

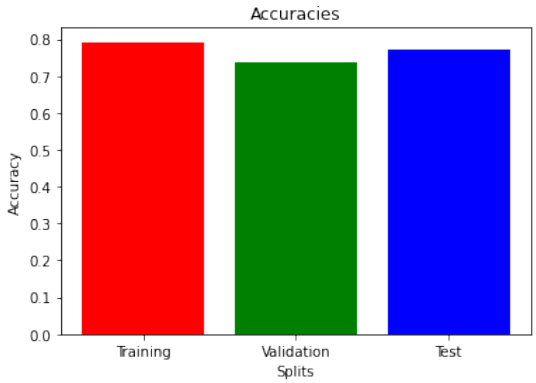
The optimizer was another hyperparameter that was tuned. The results of the different setups are presented below:

Setup	Optimizer	Test Accuracy	F-Score
1	RMSprop	0.77	0.63
2	SGD	0.79	0.63
3	AdamW	0.79	0.61

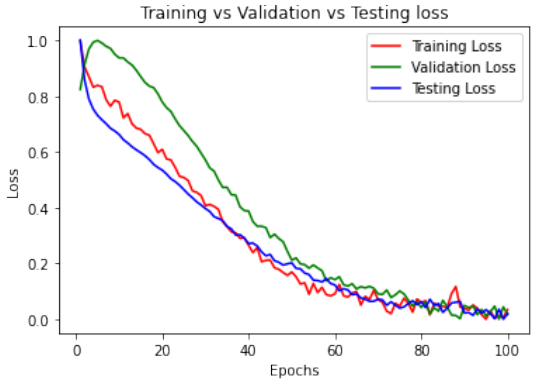
Table 2: Results for different optimizers.



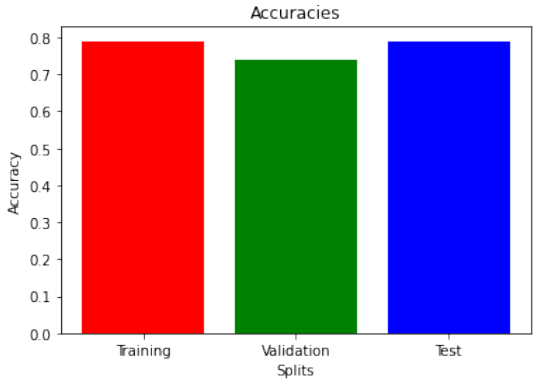
(a) Loss vs Epochs for RMSprop



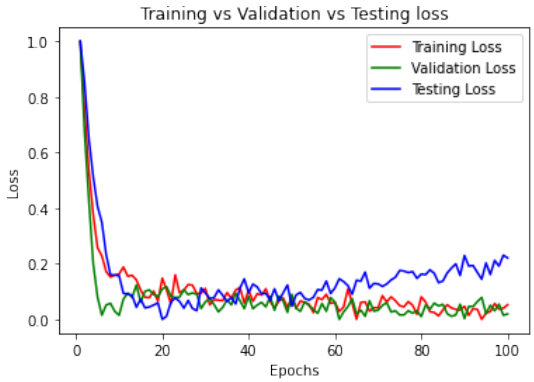
(b) Accuracy vs Epochs for RMSprop



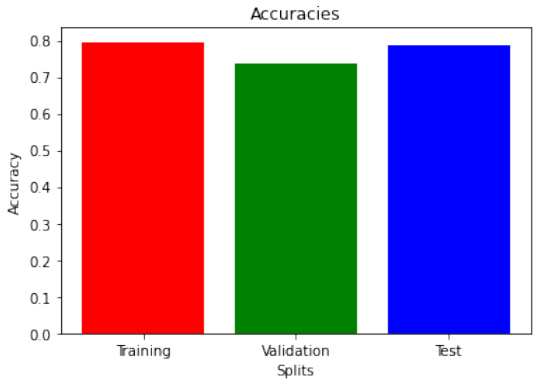
(c) Loss vs Epochs for SGD



(d) Accuracy vs Epochs for SGD



(e) Loss vs Epochs for AdamW



(f) Accuracy vs Epochs for AdamW

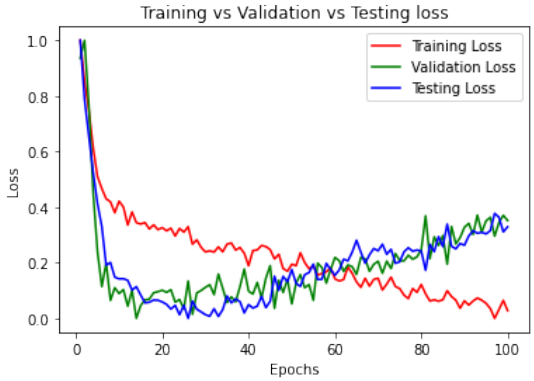
Figure 9: Graphs depicting the effect of different optimizer tuning on model training.

5.1.3 Activation Function Tuning

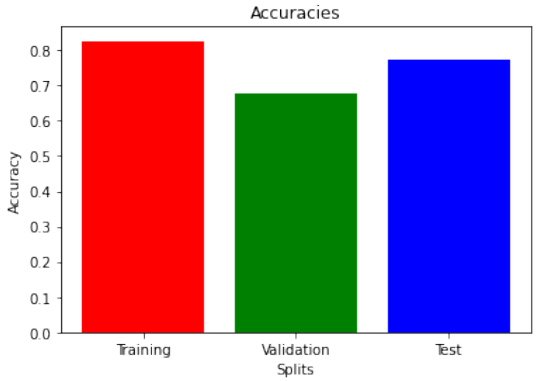
The activation function was another hyperparameter that was tuned. The results of the different setups are presented below:

Setup	Activation Function	Test Accuracy	F-Score
1	LeakyReLU	0.77	0.59
2	ReLU	0.76	0.56
3	SELU	0.80	0.63

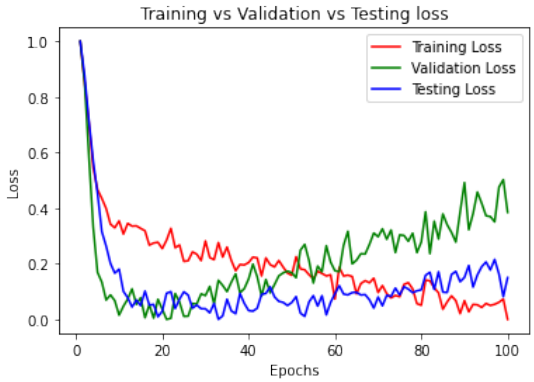
Table 3: Results for different activation functions.



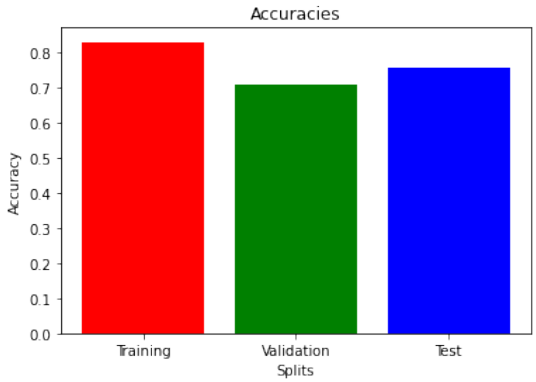
(a) Loss vs Epochs for LeakyReLU



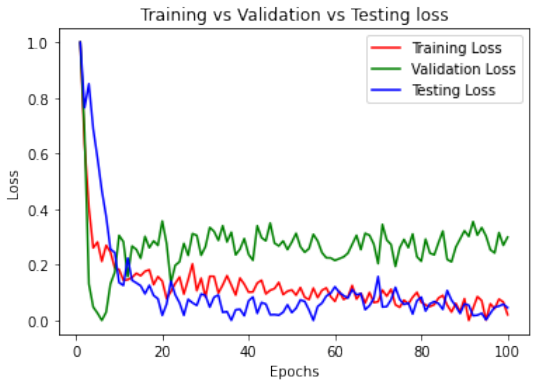
(b) Accuracy vs Epochs for LeakyReLU



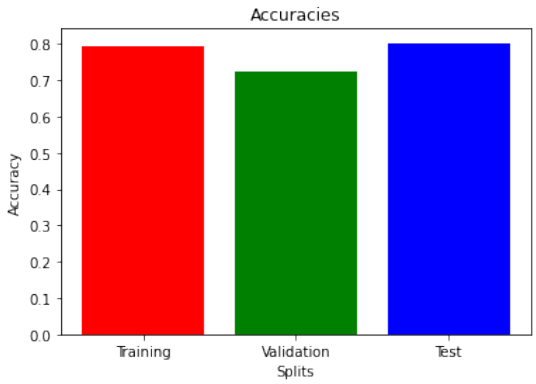
(a) Loss vs Epochs for ELU



(b) Accuracy vs Epochs for ELU



(a) Loss vs Epochs for SELU



(b) Accuracy vs Epochs for SELU

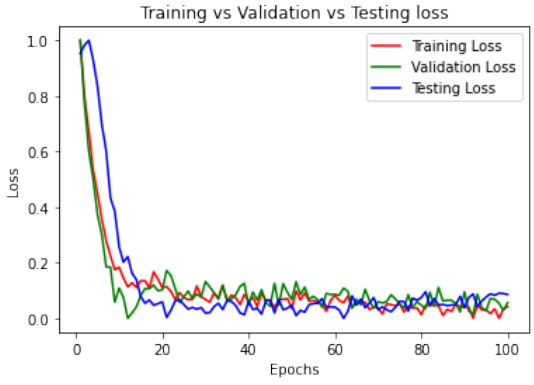
Figure 12: Graphs depicting the effect of different activation function on model training.

5.1.4 Initializer Tuning

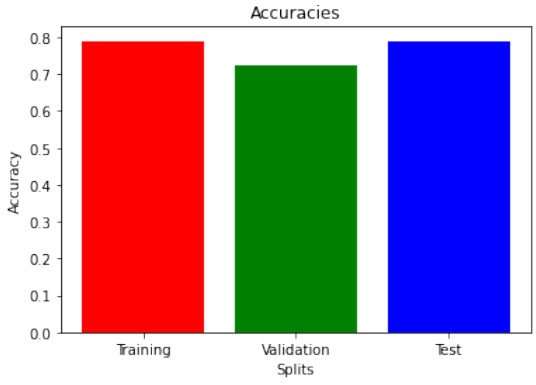
The weight initializer was the next hyperparameter to be adjusted. Results for different setups are as follows:

Setup	Initializer	Test Accuracy	F-Score
1	Xavier Uniform	0.79	0.61
2	He Uniform	0.76	0.60
3	Uniform (a=0, b=0.1)	0.79	0.61

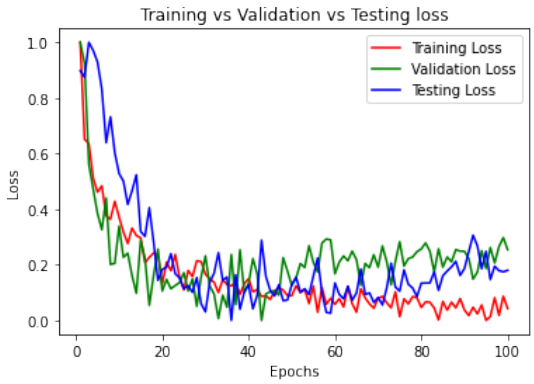
Table 4: Results for different initializers.



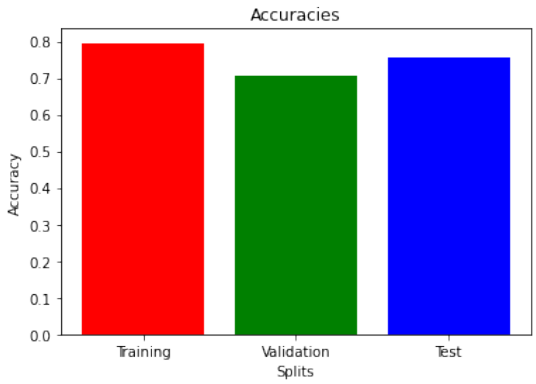
(a) Loss vs Epochs for Xavier Uniform



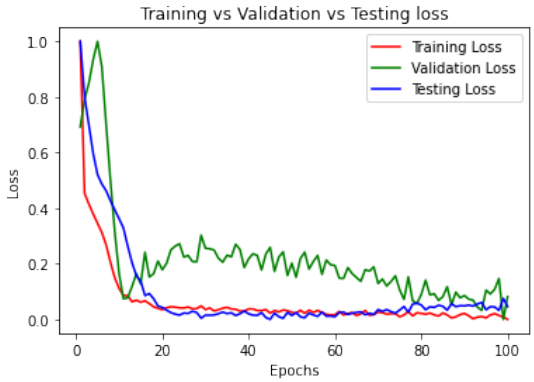
(b) Accuracy vs Epochs for Xavier Uniform



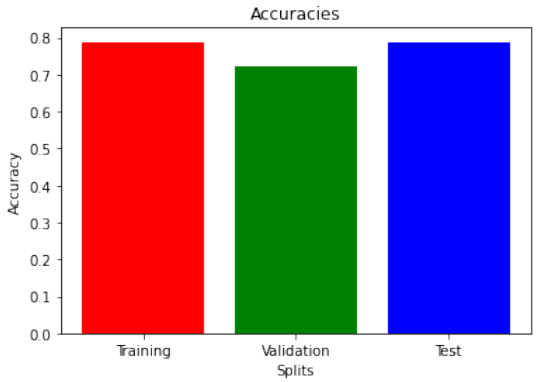
(a) Loss vs Epochs for He Uniform



(b) Accuracy vs Epochs for He Unifrom



(a) Loss vs Epochs for Uniform(a=0, b=0.1)



(b) Accuracy vs Epochs for Uniform(a=0, b=0.1)

Figure 15: Graphs depicting the effect of different Initializer Tuning on model training.

5.1.5 Analysis and Reasoning

- **Dropout Tuning**: Varying dropout values showed nuanced performance results. While a dropout of 0.25 achieved an accuracy of 0.79, increasing it to 0.5 and 0.75 both resulted in accuracies of 0.79 and 0.77 respectively. This indicates that beyond a certain threshold, increasing dropout doesn't guarantee improved performance.
- **Optimizer Tuning**: The optimizer's choice had clear implications on performance. RMSprop and SGD yielded accuracies of 0.77 and 0.79 respectively, while AdamW lagged slightly with 0.79. This underlines the importance of selecting the appropriate optimizer for a given problem.
- **Activation Function Tuning**: Different activation functions yielded varied results. SELU achieved the highest accuracy at 0.80, while LeakyReLU and ReLU recorded 0.77 and 0.76 respectively. This emphasizes the subtle, yet important impact of the activation function on model performance.
- **Initializer Tuning**: Among the weight initializers used, the He initializer led with an accuracy of 0.76. Both Xavier and Uniform initializers (range $a = 0$ to $b = 0.1$) had the same accuracy of 0.79. This highlights the pivotal role of weight initialization in neural network performance.

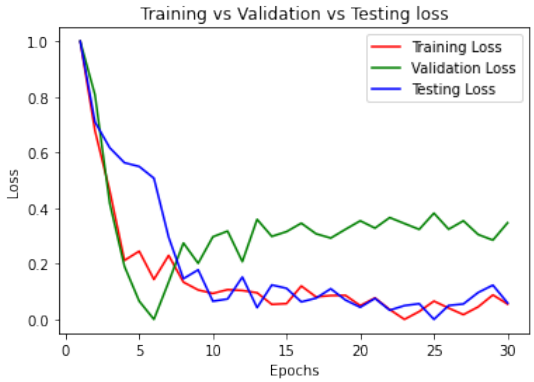
5.2 Training Optimization Methods

Various hyperparameters were employed on the base model and we selected the SeLU optimizer model as our base model as it yielded the highest accuracy among all the hyperparameters. The aim was to further enhance the model's performance. This section details the methods used and their outcomes.

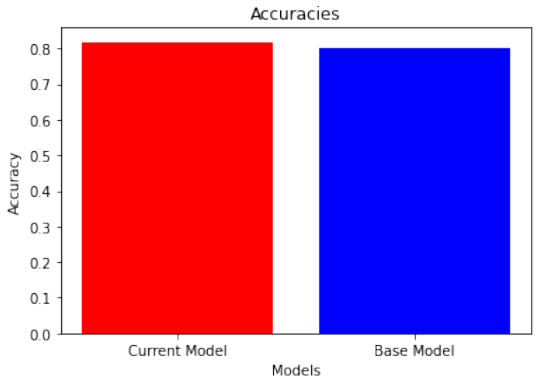
5.2.1 Early Stopping

The first optimization method employed was "Early Stopping". It is designed to prevent overfitting by halting the training process once the model's performance on a validation dataset starts to degrade, ensuring that the model does not continue to adapt too closely to the training data at the expense of generalization.

Outcome: The model achieved a test accuracy of 0.82.



(a) Loss vs Epochs for Early Stopping

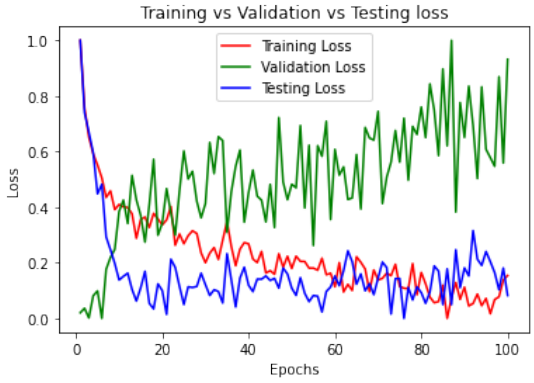


(b) Accuracy vs Epochs for Early Stopping

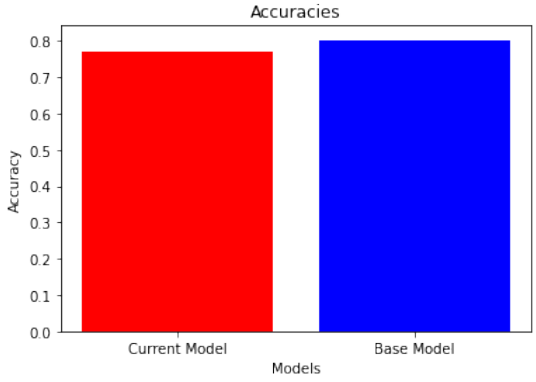
5.2.2 Batch Normalization

Next, "Batch Normalization" was integrated into the neural network architecture. Batch normalization standardizes the activations from a prior layer to have a mean output activation of zero and standard deviation of one, stabilizing the learning process and potentially accelerating the training speed.

Outcome: The model recorded a test accuracy of 0.77.



(a) Loss vs Epochs for Batch Normalization

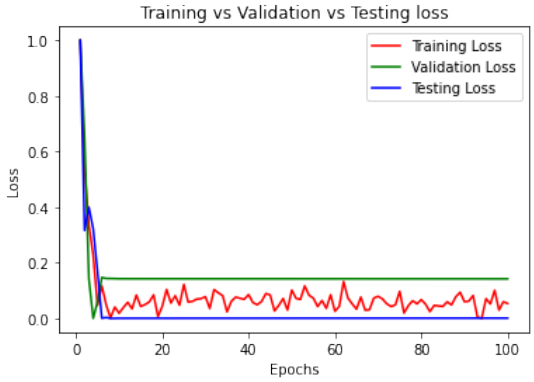


(b) Accuracy vs Epochs for Batch Normalization

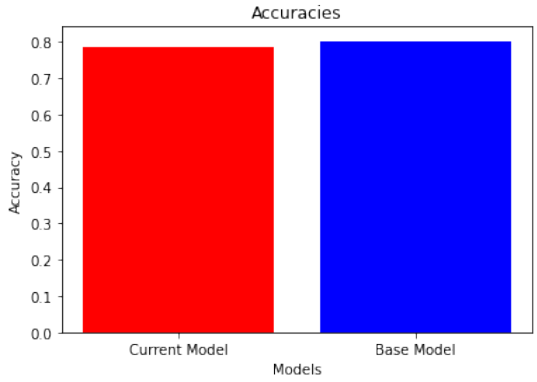
5.2.3 Learning Rate Scheduler

A "Learning Rate Scheduler" adjusts the learning rate during training, usually decreasing it over epochs. Although the explicit use of a learning rate scheduler wasn't illustrated, the same neural network structure was employed for this method.

Outcome: The performance evaluation showcased an accuracy of 0.79.



(a) Loss vs Epochs for Learning Rate Scheduler

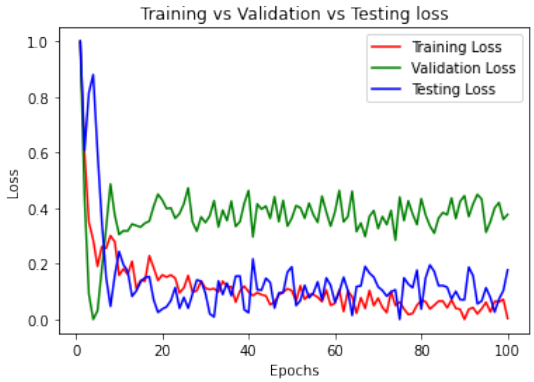


(b) Accuracy vs Epochs for Learning Rate Scheduler

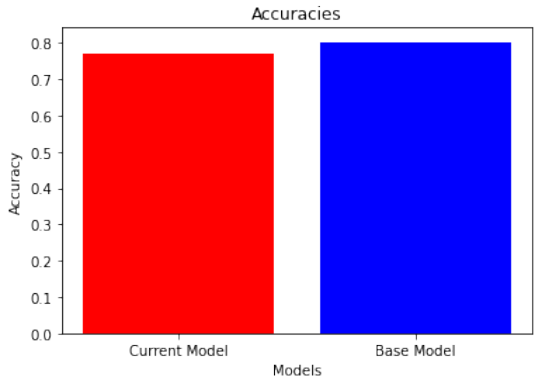
5.2.4 Base Model Re-Architecture

We rearchitected the base NN model by adding an extra hidden layer to see how it impacts on performance of the overall model.

Outcome: The performance evaluation showcased an accuracy of 0.77.



(a) Loss vs Epochs for Base Model Re-Architecture



(b) Accuracy vs Epochs for Base Model Re-Architecture

5.2.5 Analysis and Reasoning

- **Early Stopping**: This method yielded the highest accuracy of 0.82, suggesting its effectiveness in preventing overfitting for this dataset and architecture.
- **Batch Normalization**: Despite its potential in stabilizing the learning process, the performance achieved (accuracy 0.77) was slightly lower compared to early stopping. This might indicate the need for further hyperparameter tuning when using batch normalization in this context.

- **Learning Rate Scheduler**: The consistent performance with the batch normalization approach suggests that other factors might be playing a more pivotal role in this specific problem setup, or that the learning rate adjustments weren't significant enough to create a discernible difference.
- **Base Model Re-Architecture**: Rearchitecting base model might show a higher significance in a large neural network when paired up with the other hyperparameters. However, in our case it didn't have much impact and showed neural improvement.

6 Conclusion

Through this assignment, we embarked on a journey exploring neural networks' intricate layers, experimenting with their architectures, and tuning their hyperparameters. Our primary objective was to not only achieve a model with high predictive accuracy but also to understand the underlying mechanisms and decisions driving each step.

The following key takeaways were observed:

1. **Data Preprocessing**: Ensuring the data is correctly preprocessed, normalized, and free from outliers is pivotal for the effective training of neural networks. Our results reinforced the importance of this step, with better-preprocessed data leading to improved model performance.
2. **Hyperparameter Tuning**: Neural networks are sensitive to hyperparameter choices. The optimization strategies employed, from dropout tuning to adjusting the learning rate, showcased the profound impact of these choices on the model's final performance.
3. **Regularization Techniques**: Techniques such as dropout and early stopping proved instrumental in preventing overfitting, ensuring that our model generalizes well to unseen data.
4. **Model Architecture**: The architecture of the neural network, including the number of hidden layers, neurons in each layer, and the activation functions used, plays a decisive role in the model's capacity and performance.
5. **Training Optimization**: Methods like batch normalization and learning rate scheduling further refined our training process, enhancing convergence speed and model stability.

In conclusion, while our neural network models achieved promising results, machine learning, especially deep learning, is an iterative process. There is always room for improvement, be it through refining the current models, experimenting with newer architectures, or leveraging more extensive datasets. Our experiences from this assignment have provided a solid foundation, and we are optimistic about future endeavors in this domain.

7 References

- **Pandas**: https://pandas.pydata.org/docs/
- **Matplotlib**: https://matplotlib.org/stable/index.html
- **Sklearn**: https://scikit-learn.org/stable/
- **Torch**: https://pytorch.org/docs/stable/index.html
- **Time**: https://docs.python.org/3/library/time.html
- **Torchinfo**: https://github.com/TylerYep/torchinfo
- **Torchmetrics**: https://torchmetrics.readthedocs.io/en/stable/