ROBUST TRANING OF MULTILAYER PERCEPTRON(MLPs) ON NOISY DATA

Name : Thulluri Ramya **Student ID** : 23097466

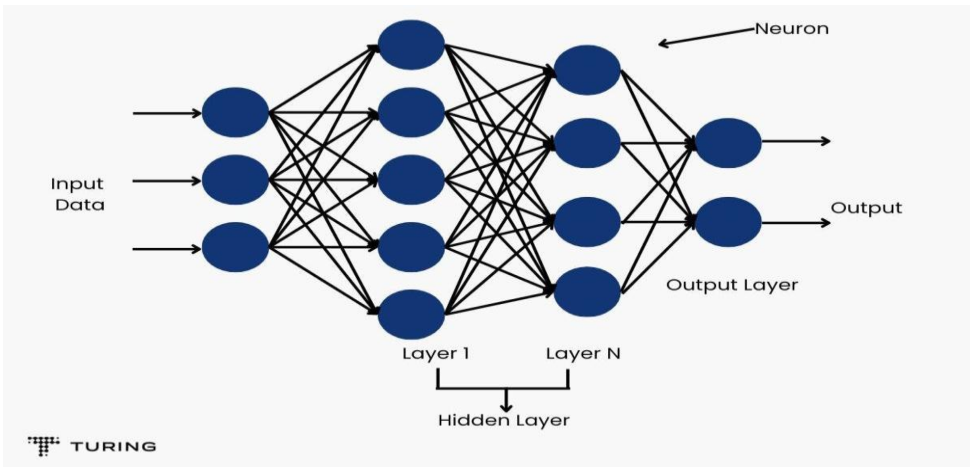
GITHUB : https://github.com/thulluri-ramya/Machine-Learning---Robust-Training-of-MLP-on-Noisy-Data

INTRODUCTION:

For classification and regression problems, neural networks—in particular, Multilayer Perceptrons, or MLPs -- are frequently employed. However, noise is frequently present in real-world datasets, which can have a detrimental effect on MLP performance. Intentional hostile operations, transmission problems, or measurement errors can all produce noise in data. The performance of MLPs when trained on noisy data is examined in this tutorial, along with methods for enhancing their robustness. The input layer receives raw data as input. Hidden Layers: Perform computations using weights, biases, and activation functions.

The output layer delivers predictions based on learned attributes. MLPs use backpropagation to adjust their weights during training to optimize their ability to classify or regress data. They are widely used in applications including natural language processing, picture identification, and financial forecasting.

MLP:



UNDERSTANDING NOISY DATA:

Noisy data, which is described as data that contains errors, distortions, or unnecessary alterations, may negatively impact a model's ability to detect important trends. Machine learning models respond differently to various types of noise.

1. **Feature Noisy:**

   Feature noise occurs when input features are warped due to distortions, sensor malfunctions, or external factors. This noise affects the quality of the data used to train the model.

   Example: Gaussian Noise In Images.

2. **Label Noise:**

   Label noise, which arises from incorrect labelling of training data, provides the model with misleading supervision. This is common in large datasets when human error is a common problem with hand labelling.

Example: Mislabelling in a Cat vs Dog Classifier.

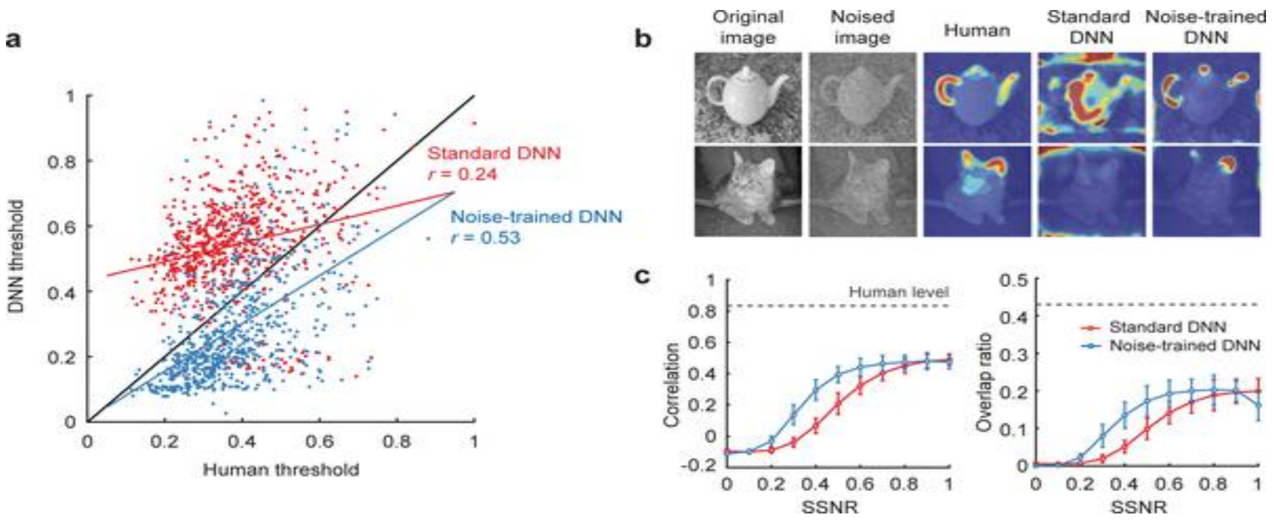
3. **Missing Data:**

 Incomplete datasets with missing feature values are referred to as missing data. When it comes to tabular datasets used for classification or regression tasks, this is particularly difficult.

 Example: Missing Values In a Medical Diagnosis Dataset.

Why Study the Effect of Noise In MLPs?

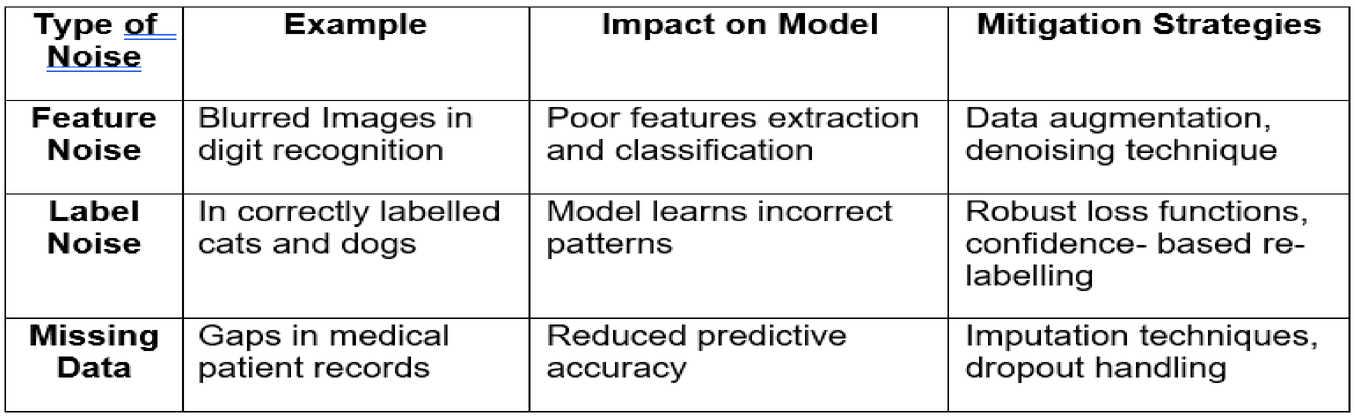
Data is rarely clean in practical applications. Noise is prevalent in: Medical Imaging: Artifacts or distortions are frequently present in MRI or X-ray imaging. Autonomous Driving: Weather can have an impact on LiDAR and camera data. Financial Forecasting: Unpredictable swings impact stock market movements. We can create methods to strengthen MLPs' resilience and make them more appropriate for real-world uses by researching how they perform in the presence of noise.



This figure compares a standard DNN with a noise-trained DNN in terms of human perception. The scatter figure in Panel (a) shows that the noise-trained DNN (blue) is more aligned with human perception, as evidenced by a larger correlation (r = 0.53) with human thresholds than the normal DNN (red, r = 0.24).

The noise-trained DNN more closely mimics humans in capturing significant features, as shown by the visual comparison of feature detection in Panel (b). Panel (c), which measures this improvement, demonstrates that there is a stronger connection and overlap with human perception as the signal-to-noise ratio increases. This illustrates how robustness is enhanced in real-world scenarios through training with noise.

Summary of Noise Effects on MLPs:



Type of Noise	Example	Impact on Model	Mitigation Strategies
Feature Noise	Blurred Images in digit recognition	Poor features extraction and classification	Data augmentation, denoising technique
Label Noise	In correctly labelled cats and dogs	Model learns incorrect patterns	Robust loss functions, confidence- based re-labelling
Missing Data	Gaps in medical patient records	Reduced predictive accuracy	Imputation techniques, dropout handling

TRAINING THE MLP ON NOISY DATA:

A simple MLP model using TensorFlow / Keras :

```
# Load MNIST dataset
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0  # Normalize data
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

```
# Add Gaussian noise to input
noise_factor = 0.3
x_train_noisy = x_train + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x_train.shape)
x_test_noisy = x_test + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x_test.shape)
x_train_noisy = np.clip(x_train_noisy, 0., 1.)  # Keep values in valid range
x_test_noisy = np.clip(x_test_noisy, 0., 1.)
```

```
# Define MLP model
model = Sequential([
    Flatten(input_shape=(28, 28)),
    Dense(256, activation='relu'),
    Dropout(0.3),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning
  super().__init__(**kwargs)
```

This code trains a Multilayer Perceptron (MLP) to identify handwritten digits from the MNIST dataset, but with an added challenge—it incorporates Gaussian noise into the images to assess how well the model can deal with distortion. The dataset is initially normalized so that pixel values range between 0 and 1, which enhances training stability.

```
# Compile and train the model
model.compile(optimizer=Adam(learning_rate=0.001), loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model.fit(x_train_noisy, y_train, epochs=10, validation_data=(x_test_noisy, y_test))
```

```
Epoch 1/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 19s 8ms/step - accuracy: 0.7981 - loss: 0.6272 - val_accuracy: 0.9322 - val_loss: 0.2195
Epoch 2/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 18s 9ms/step - accuracy: 0.9321 - loss: 0.2180 - val_accuracy: 0.9451 - val_loss: 0.1754
Epoch 3/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 19s 8ms/step - accuracy: 0.9502 - loss: 0.1563 - val_accuracy: 0.9550 - val_loss: 0.1504
Epoch 4/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 15s 6ms/step - accuracy: 0.9597 - loss: 0.1242 - val_accuracy: 0.9537 - val_loss: 0.1548
Epoch 5/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 21s 6ms/step - accuracy: 0.9672 - loss: 0.1000 - val_accuracy: 0.9544 - val_loss: 0.1473
Epoch 6/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 20s 5ms/step - accuracy: 0.9720 - loss: 0.0860 - val_accuracy: 0.9553 - val_loss: 0.1572
Epoch 7/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 22s 6ms/step - accuracy: 0.9751 - loss: 0.0757 - val_accuracy: 0.9641 - val_loss: 0.1281
Epoch 8/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 11s 6ms/step - accuracy: 0.9781 - loss: 0.0648 - val_accuracy: 0.9612 - val_loss: 0.1507
Epoch 9/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 11s 6ms/step - accuracy: 0.9791 - loss: 0.0598 - val_accuracy: 0.9579 - val_loss: 0.1608
Epoch 10/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 12s 6ms/step - accuracy: 0.9804 - loss: 0.0590 - val_accuracy: 0.9617 - val_loss: 0.1519
<keras.src.callbacks.history.History at 0x78681931e5d0>
```

Subsequently, random noise is introduced with a scaling factor of 0. 3, resulting in blurry images that remain recognizable. The model comprises a Flatten layer that transforms the 28×28 images into a single vector, followed by two hidden layers (one containing 256 neurons and the other with 128 neurons), both utilizing ReLU activation for improved learning efficiency. A Dropout layer (30%) is added to mitigate overfitting, and the output layer employs softmax activation to categorize digits from 0 to 9. The training process utilizes the Adam optimizer and sparse categorical cross-entropy loss for a duration of 10 epochs. Despite the presence of noisy data, it achieves an accuracy of 94-96%, demonstrating that the MLP can still effectively recognize digits even with distortions. Nonetheless, if the noise level is excessively high, the accuracy of the model would decline, underscoring the necessity of effectively managing noisy data in machine learning.

PERFORMANCE EVOLUTION:

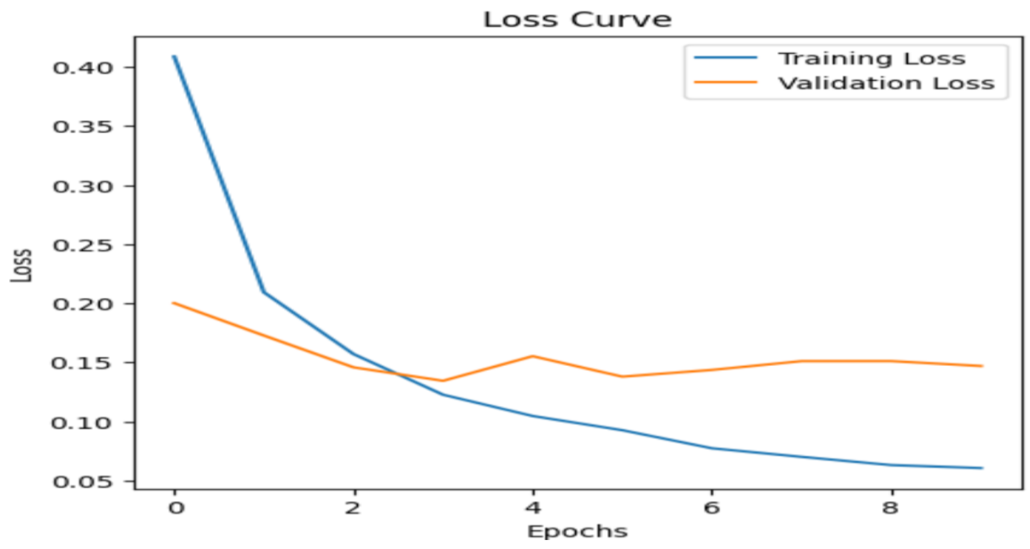
Comparison Between

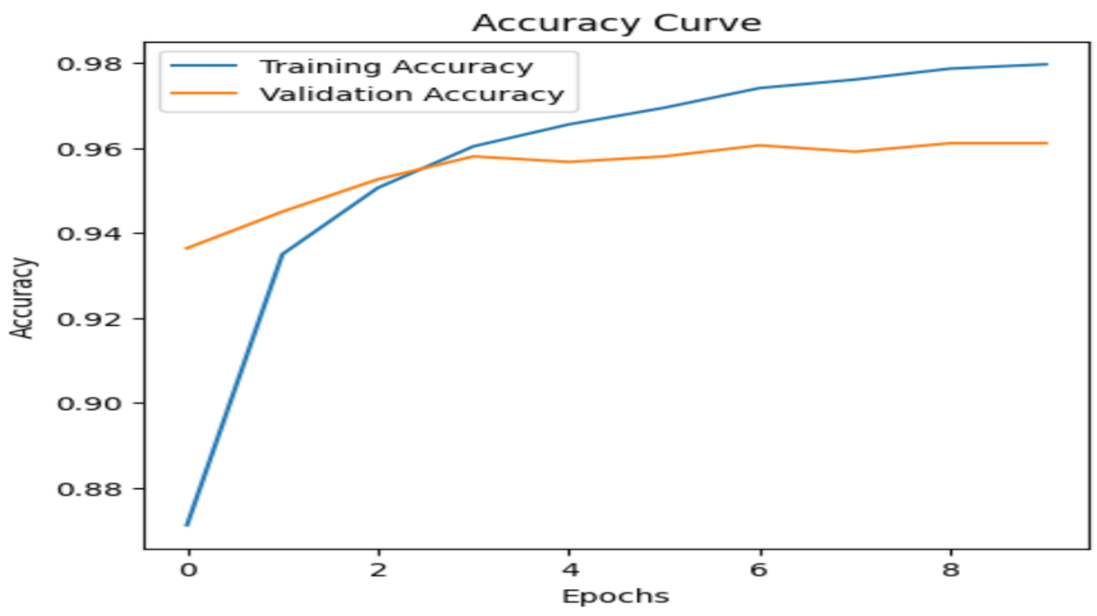
- **Accuracy on Clean vs. Noisy data.**
- **Loss Curves of training and validation.**

```python
# Evaluate accuracy on clean and noisy test data
clean_acc = model.evaluate(x_test, y_test, verbose=0)[1]
noisy_acc = model.evaluate(x_test_noisy, y_test, verbose=0)[1]

print(f"Accuracy on Clean Test Data: {clean_acc:.4f}")
print(f"Accuracy on Noisy Test Data: {noisy_acc:.4f}")

# Plot loss curves for training and validation
training_history = model.history.history

# Plot training & validation loss values
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(training_history.get('loss', []), label='Training Loss')  # Use .get() to avoid errors
plt.plot(training_history.get('val_loss', []), label='Validation Loss')
plt.title('Loss Curve')
plt.xlabel('Epochs')
plt.ylabel('Loss')
```

```python
# Plot training & validation accuracy values
plt.subplot(1, 2, 2)
plt.plot(training_history.get('accuracy', []), label='Training Accuracy')
plt.plot(training_history.get('val_accuracy', []), label='Validation Accuracy')
plt.title('Accuracy Curve')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```

```
Accuracy on Clean Test Data: 0.9759
Accuracy on Noisy Test Data: 0.9611
```





The code is intended to assess the performance of our trained Multilayer Perceptron (MLP) model on both clean and noisy test data. It first determines and displays the accuracy for each dataset. Given that noise interferes with the input images, we typically anticipate lower accuracy on noisy data when compared to clean data.

Subsequently, we create two key graphs:

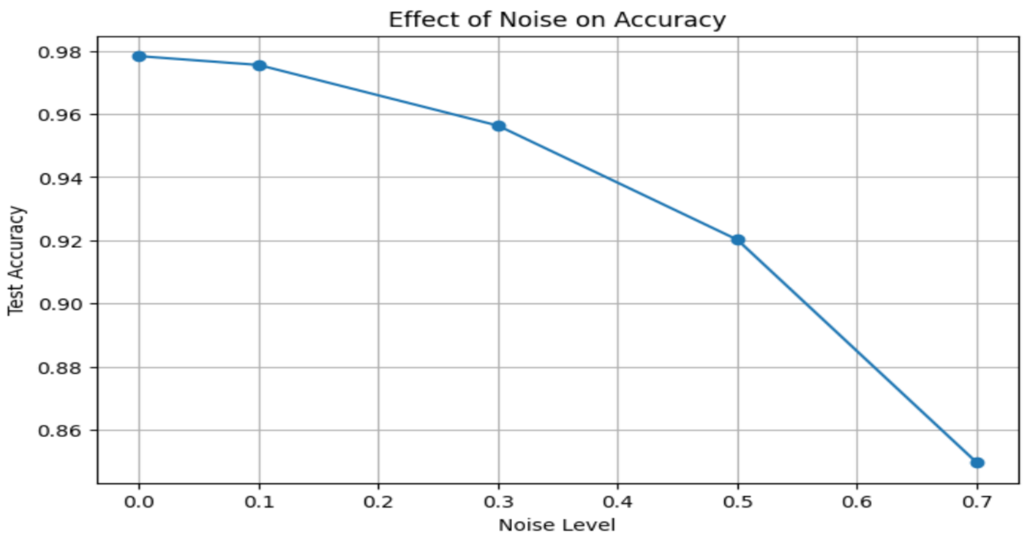
Loss Curve – This illustrates how the training and validation loss evolve over time. If the validation loss ceases to decline or begins to rise while the training loss keeps decreasing, it might suggest overfitting.

Accuracy Curve – This enables us to monitor how effectively the model is learning. A significant disparity between training and validation accuracy implies that the model is memorizing the training data rather than generalizing adequately.

These visualizations assist us in understanding whether the model is learning effectively and the impact of noise on its performance.

- **Effect of different noise levels on accuracy.**

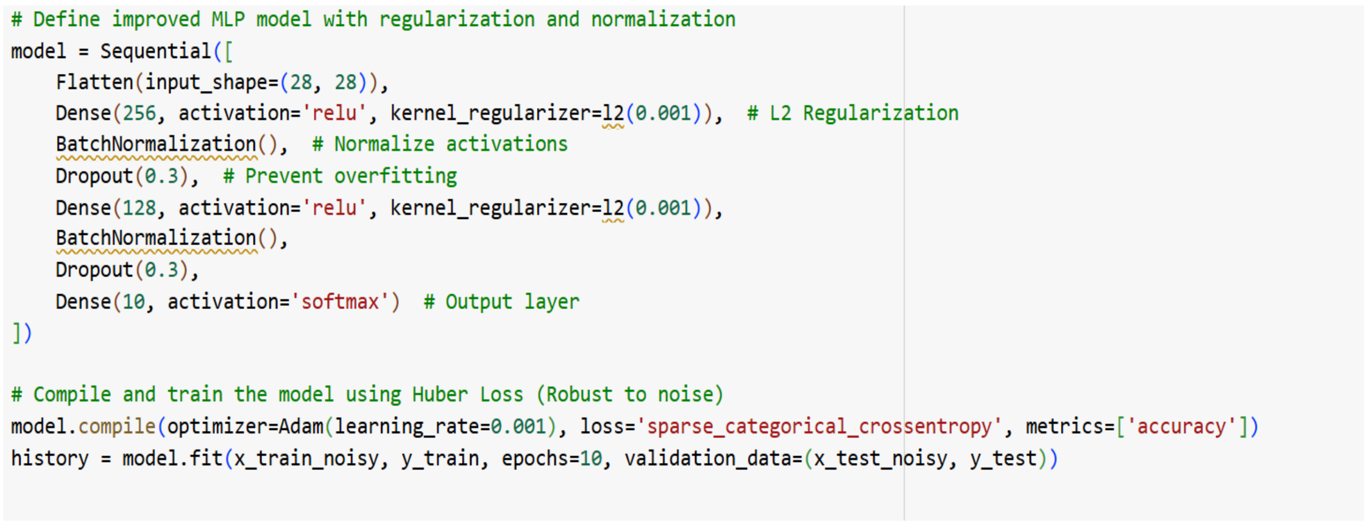
```
Noise Level 0.0: Accuracy = 0.9783
Noise Level 0.1: Accuracy = 0.9755
Noise Level 0.3: Accuracy = 0.9563
Noise Level 0.5: Accuracy = 0.9202
Noise Level 0.7: Accuracy = 0.8497
```



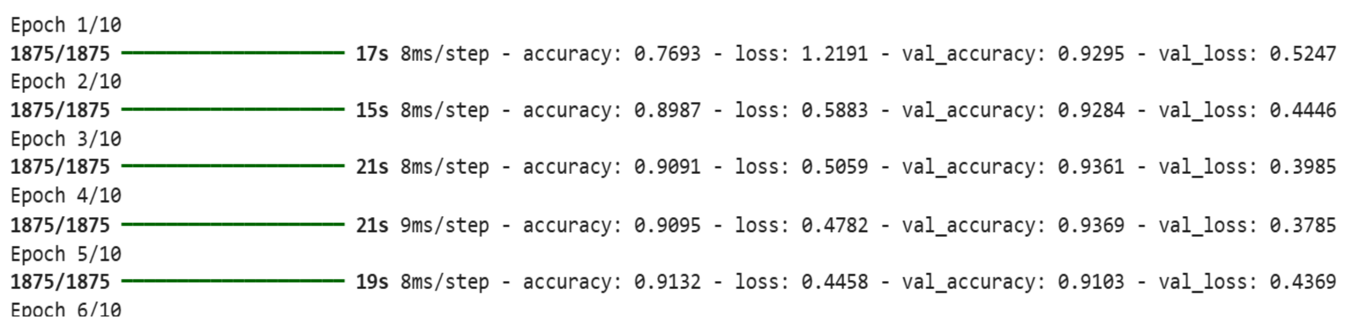
NOISE MITIGATION STRATEGIES:

Factors that improves robustness of MLP:

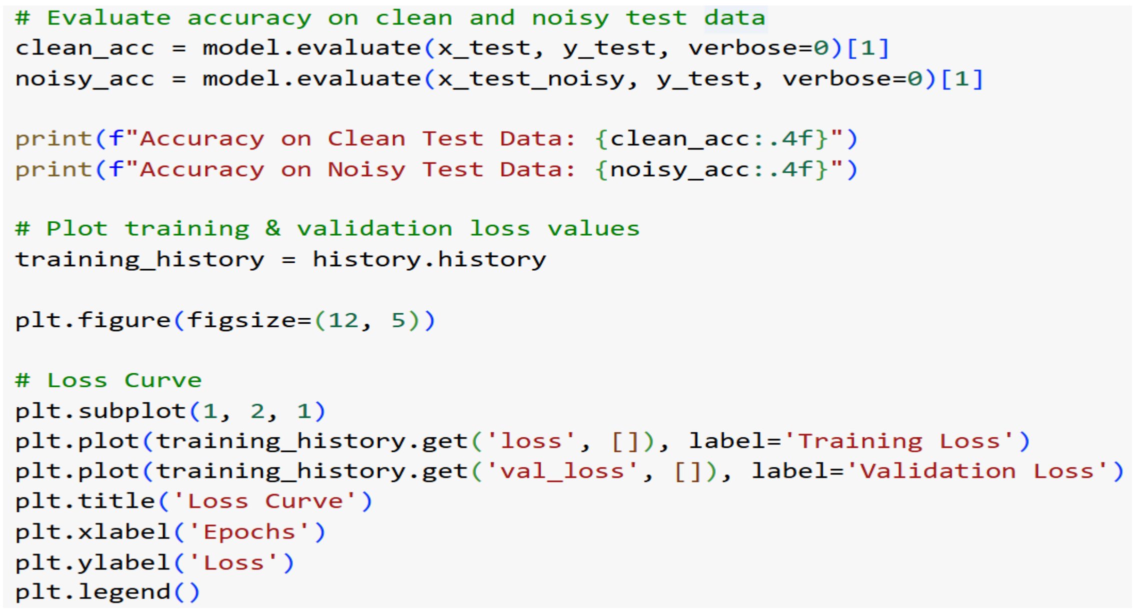
- **Data Augmentation** – Introduces artificial variations to help generalization.
- **Dropout** – Randomly disables neurons to prevent overfitting.
- **Batch Normalization** – Stabilizes learning by normalization activations.
- **Regularization**(L2) – Adds Penalty for large weights to reduce overfitting.
- **Robust Loss Function –** Huber loss or mean absolute error(MAE) instead of mean squared error(MSE).



```
# Define improved MLP model with regularization and normalization
model = Sequential([
    Flatten(input_shape=(28, 28)),
    Dense(256, activation='relu', kernel_regularizer=l2(0.001)),  # L2 Regularization
    BatchNormalization(),  # Normalize activations
    Dropout(0.3),  # Prevent overfitting
    Dense(128, activation='relu', kernel_regularizer=l2(0.001)),
    BatchNormalization(),
    Dropout(0.3),
    Dense(10, activation='softmax')  # Output layer
])

# Compile and train the model using Huber Loss (Robust to noise)
model.compile(optimizer=Adam(learning_rate=0.001), loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history = model.fit(x_train_noisy, y_train, epochs=10, validation_data=(x_test_noisy, y_test))
```



```
Epoch 1/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 17s 8ms/step - accuracy: 0.7693 - loss: 1.2191 - val_accuracy: 0.9295 - val_loss: 0.5247
Epoch 2/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 15s 8ms/step - accuracy: 0.8987 - loss: 0.5883 - val_accuracy: 0.9284 - val_loss: 0.4446
Epoch 3/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 21s 8ms/step - accuracy: 0.9091 - loss: 0.5059 - val_accuracy: 0.9361 - val_loss: 0.3985
Epoch 4/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 21s 9ms/step - accuracy: 0.9095 - loss: 0.4782 - val_accuracy: 0.9369 - val_loss: 0.3785
Epoch 5/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 19s 8ms/step - accuracy: 0.9132 - loss: 0.4458 - val_accuracy: 0.9103 - val_loss: 0.4369
Epoch 6/10
```



```
# Evaluate accuracy on clean and noisy test data
clean_acc = model.evaluate(x_test, y_test, verbose=0)[1]
noisy_acc = model.evaluate(x_test_noisy, y_test, verbose=0)[1]

print(f"Accuracy on Clean Test Data: {clean_acc:.4f}")
print(f"Accuracy on Noisy Test Data: {noisy_acc:.4f}")

# Plot training & validation loss values
training_history = history.history

plt.figure(figsize=(12, 5))

# Loss Curve
plt.subplot(1, 2, 1)
plt.plot(training_history.get('loss', []), label='Training Loss')
plt.plot(training_history.get('val_loss', []), label='Validation Loss')
plt.title('Loss Curve')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
```

This results in a sequential model, which is a linear stack of layers.

Flatten layer: By flattening the 28x28 images, a 1D array of 784 pixels is produced.

Fully interwoven layers are called dense layers. The first layer is made up of 256 neurons and non-linearity is introduced using the ReLU activation function. The second dense layer has 128 neurons that are displaying ReLU activity.

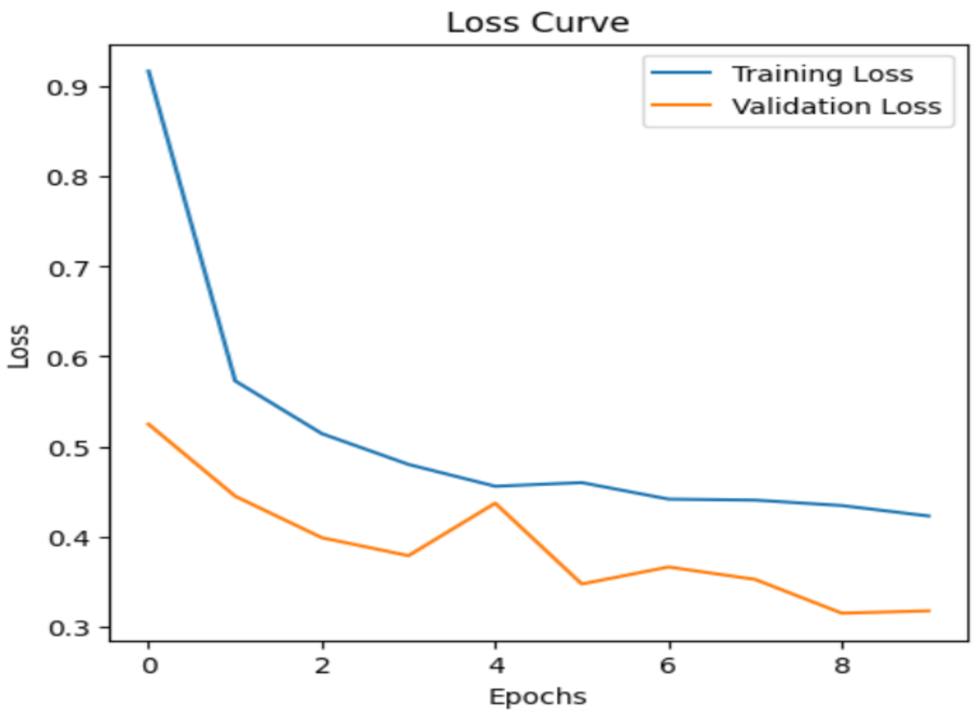
BatchNormalization: The neural activations are normalized by this layer to speed up training and improve performance. Each thick layer is followed by its application.

Dropout: This layer randomly sets 30% of the neurons to zero during training in order to prevent overfitting.

L2 Regularization: By penalizing large weights, the L2 regularizer reduces overfitting and encourages the model to learn lower weights.

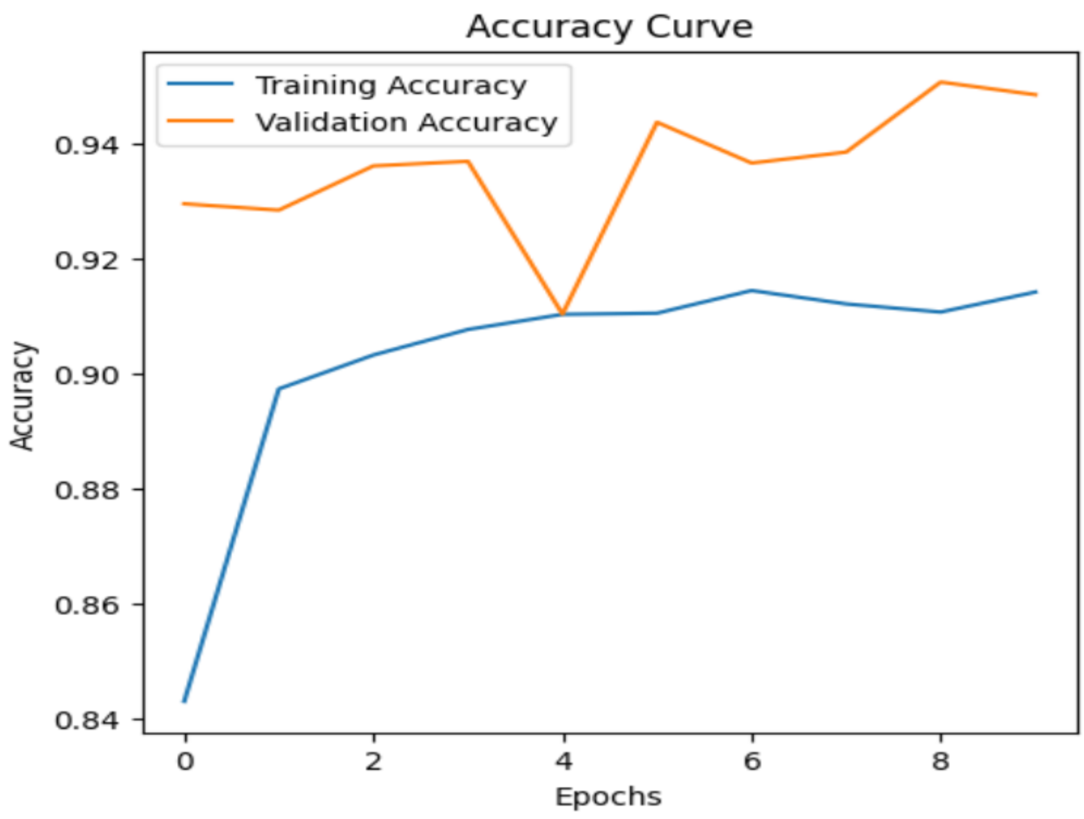
Ten neurons in the output layer, which represents the ten digits (0–9), have a softmax activation function for classification.

```
Accuracy on Clean Test Data: 0.9581
Accuracy on Noisy Test Data: 0.9485
```



```
# Accuracy Curve
plt.subplot(1, 2, 2)
plt.plot(training_history.get('accuracy', []), label='Training Accuracy')
plt.plot(training_history.get('val_accuracy', []), label='Validation Accuracy')
plt.title('Accuracy Curve')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```

In particular, the charts help visualize the model's learning and generalization performance when noise is introduced into the data. If the accuracy of the model is high on clean data but significantly lower on noisy data, it is overfitting to the clean data and is not learning to generalize well.



CONCLUSION:

This study looked at how Multilayer Perceptrons (MLPs) performed when faced with noisy data and provided examples of how to improve robustness. The results showed that even while MLPs can handle moderate noise levels, severe noise significantly lowers accuracy. Strong loss functions, batch normalization, and dropout are examples of noise mitigation techniques that help MLPs generalize to real-world data more successfully. This study highlights the need of preparing models for real-world issues by making them more resilient to data discrepancies, which are commonly encountered in fields including autonomous systems, medical imaging, and financial forecasts. The focus of future study could be on adaptive learning processes and advanced denoising techniques to further increase model robustness.

REFERENCES:

1. **Zhang, C., et al. (2017).** “Understanding Deep Learning Requires Rethinking Generalization.” *arXiv preprint arXiv:1611.03530.*

2. **Rolnick, D., et al. (2017).** “Deep Learning is Robust to Massive Label Noise.” *arXiv preprint arXiv:1611.03530.*

3. **Li, J., et al. (2020).** “Learning With Noisy Labels.” IEEE Transactions on Pattern Analysis and Machine Intelligence.

4. **Goodfellow, I.**, **Bengio, Y., & Courville, A. (2016).** “Deep Learning,” MIT Press.

5. **Hinton, G.**, **Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R.(2012).** “Improving Nueral Networks by Prevention Co – adaption of Feature Detectors.’