

**PROJECT ON:**

**HANDWRITTEN DIGIT RECOGNITION**

**PREPARED BY:**

1. **RUNGSHITA MUKHERJEE (221001002056)**
2. **MD. SAHIL ANSARI (221001002062)**
3. **DAVID ROSHAN (231001012004(L))**
4. **SURANJANA ROY (231001012005(L))**

**ABSTRACT:**

**Handwritten Digit Recognition Using TensorFlow** is a machine learning project that leverages the power of neural networks to classify handwritten digits from 0 to 9. The project utilizes the MNIST dataset, which contains 70,000 grayscale images of handwritten digits, each scaled to 28x28 pixels. The primary goal is to train a model that can accurately recognize and predict digits from unseen data.

The process begins with data preprocessing, including normalization and reshaping of the images for compatibility with TensorFlow. A Convolutional Neural Network (CNN) is typically employed due to its efficiency in extracting spatial features from images. The model undergoes training using techniques like backpropagation and optimization algorithms such as Adam. Accuracy metrics and loss functions guide the model's performance tuning.

After training, the model is evaluated on a test dataset to validate its accuracy. It can be further deployed for real-time digit recognition tasks, such as postal code reading or bank check verification. This project demonstrates the power of deep learning in solving real-world pattern recognition problems with high precision and scalability.

***Problem*:** Handwritten digit recognition is a challenge in machine learning due to variations in writing styles, orientations, and noise in the input images. The goal is to create an accurate model to classify digits (0-9) from images.

**METHODS USED:**

***Dataset*:** MNIST dataset (70,000 grayscale images of handwritten digits).

Preprocessing: Normalization (scaling pixel values)- division by 255 to give a float value <1, reshaping images from 28x28 pixel resolution to 1x784 NumPy array.

***Model*:** Artificial Neural Networks (ANNs) for feature extraction and classification.

Training: Backpropagation, optimization (Adam optimizer), and cross-entropy loss function.

***Results*:** The trained ANN achieved a high accuracy of over 98% on the test dataset, demonstrating strong generalization to unseen data.

***Key Findings:***

Preprocessing (normalization and augmentation) improves performance.

ANNs are highly effective for image-based classification tasks.

Proper hyperparameter tuning (e.g., learning rate, epochs) significantly impacts model accuracy.

This project highlights the efficiency of deep learning for real-world applications like check processing, form digitization, and postal code recognition.

**INTRODUCTION:**

Handwritten digit recognition is a critical task in machine learning, used in numerous real-world applications like postal code recognition, bank check processing, and document digitization. The increasing need for automated systems to accurately interpret handwritten data has driven research into creating reliable models capable of recognizing digits in a variety of styles and formats.

**PROBLEM STATEMENT:**

How can a machine learning model be designed and implemented to accurately recognize handwritten digits from images, despite variations in handwriting styles, noise, and distortions?

**OBJECTIVE:**

The purpose of this project is to develop a machine learning system using TensorFlow that can automatically recognize handwritten digits with high accuracy. The model should generalize well to unseen data and provide a foundation for practical applications.

**SCOPE:**

The project focuses on leveraging the MNIST dataset and Convolutional Neural Networks (CNNs) to create a digit recognition model. It explores data preprocessing, model training, and evaluation techniques. This project is significant in demonstrating the potential of deep learning for automating time-consuming and error-prone manual data entry tasks.

**OVERVIEW OF REPORT STRUCTURE:**

***Introduction*:** Outlines the problem, objective, and scope.

***Background*:** Reviews existing approaches and their limitations.

***Methodology*:** Describes the dataset, preprocessing, and model development**.**

***Results and Analysis*:** Discusses performance, insights, and challenges.

***Conclusion*:** Summarizes findings and suggests future directions.

**LITERATURE REVIEW :**

Several research studies and solutions have contributed to the development of handwritten digit recognition systems:

**LeCun et al. (1998)** introduced the MNIST dataset, a benchmark for handwritten digit recognition. Their work demonstrated the effectiveness of Convolutional Neural Networks (CNNs) in achieving high accuracy for image classification tasks.

* *From the paper "Gradient-Based Learning Applied to Document Recognition" by Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner, published in 1998.*

**Hinton et al. (2006)** developed Deep Belief Networks (DBNs), showcasing how unsupervised pretraining could enhance performance for digit recognition tasks.

* *From the paper "A Fast Learning Algorithm for Deep Belief Nets" by Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh, published in 2006.*

***Recent CNN Advances*:** Modern architectures, such as AlexNet and VGG, have proven their superiority in feature extraction for image data, inspiring efficient solutions for handwritten digit classification.

***Transfer Learning:*** Research shows that pre-trained models on large datasets can be fine-tuned for digit recognition, saving computational resources while maintaining high accuracy.

***Optimization Techniques*:** Studies on optimizers (Adam, SGD) and regularization methods (dropout, batch normalization) have refined the training process, reducing overfitting and improving generalization.

These studies underscore the importance of deep learning in tackling pattern recognition challenges, guiding the development of robust solutions for handwritten digit recognition.

**DATASET DESCRIPTION:**

**SOURCE:** The MNIST dataset is a benchmark dataset for handwritten digit recognition. It can be found at Yann LeCun's page.

**DESCRIPTION:**

***Size*:**

* ***Training Set*:** 60,000 examples.
* ***Test Set*:** 10,000 examples.

**STRUCTURE*:***

Each image is a 28×28 grid of pixels.

This makes a total of 784 pixels per image.

Each pixel represents a grayscale intensity value between 0 (black) and 255 (white).

Labels: Each example is labelled with the digit (0–9) it represents.

**PREPROCESSING STEPS:**

***Normalization***:

Grayscale pixel values are divided by 255 to normalize them to a range of [0, 1].

This helps in faster convergence during training.

***Flattening***:

The 28×28 grid is reshaped into a 1-dimensional array of size 784.

This results in a matrix of size (60,000 × 784) for training data.

***Resizing for Input:***

Each 28×28 NumPy array is reshaped into a vector of shape (784×1).

This is useful for input into machine learning models.

**METHODOLOGY:**

***Exploratory Data Analysis (EDA)***

***Data Overview:***

The MNIST dataset contains 60,000 training images and 10,000 test images, each of size 28x28 pixels. Each image is labelled with a digit (0-9).

***Training labels:***

{0: 5923, 1: 6742, 2: 5958, 3: 6131, 4: 5842, 5: 5421, 6: 5918, 7: 6265, 8: 5851, 9: 5949}

The dataset is relatively balanced across classes.

***Visualization***:

Randomly selected 25 images were plotted to confirm data integrity. All images appeared to be correctly labelled.

*Pixel intensity distribution*: All images were normalized (pixel values scaled between 0 and 1).

**Feature Engineering**

***Normalization***: Pixel values were divided by 255 to scale them to the [0,1] range.

***Reshaping***: Images were flattened from shape (28, 28) to (784,) for use in a fully connected neural network.

***One-hot Encoding***: Labels were converted to categorical format using to\_categorical() function.

***Model Selection***

***Base Model:***

A feedforward neural network with two hidden layers:

* 256 neurons (ReLU activation) in the first layer.
* 40 neurons (ReLU activation) in the second layer.
* 10 output neurons (SoftMax activation) for classification.

***Optimizer***: Adam.

***Loss Function*:** Categorical cross-entropy.

***With Dropout*:**

Added dropout layers (0.5 and 0.25 rates) to reduce overfitting.

***Training and Testing***

***Train-Test Split*:** The default MNIST split was used (60,000 training, 10,000 testing).

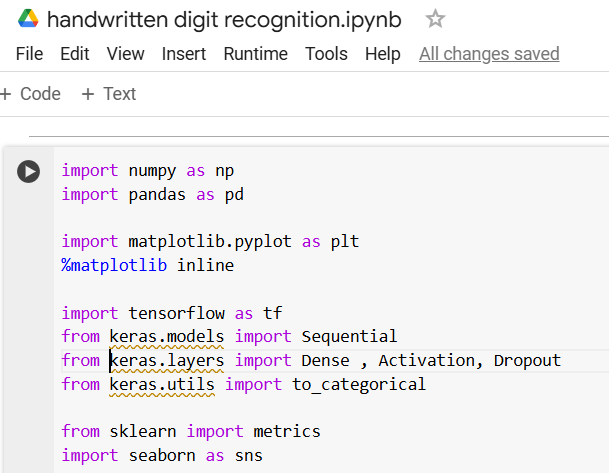
***Batch Size and Epochs*:**

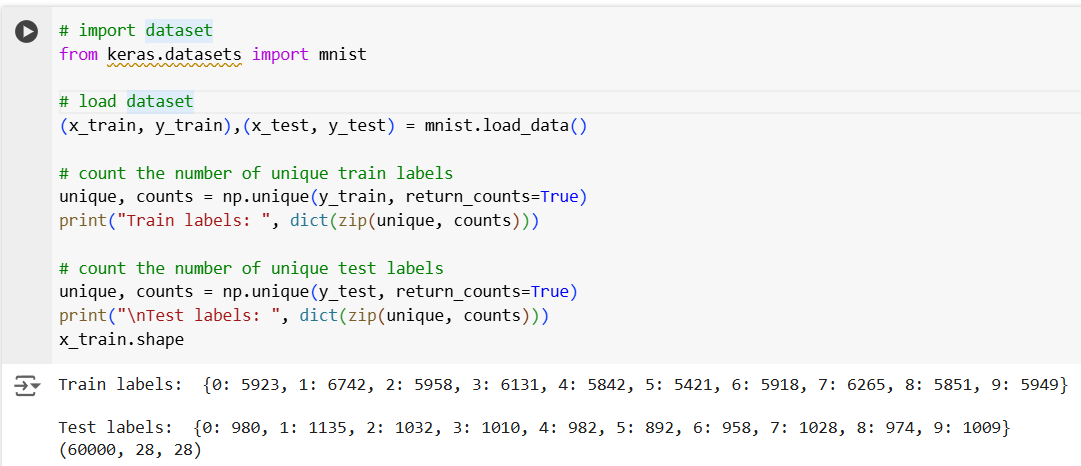
***Batch size:*** 128

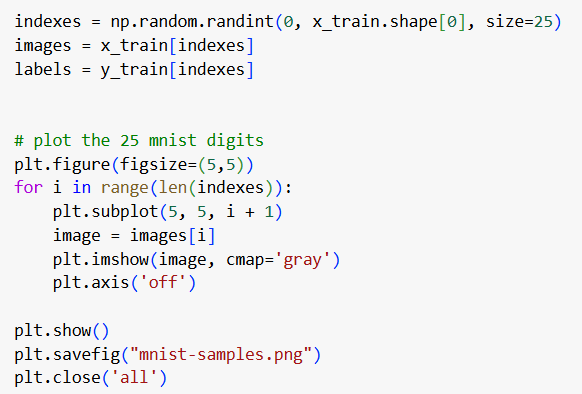
***Epochs*:** 20

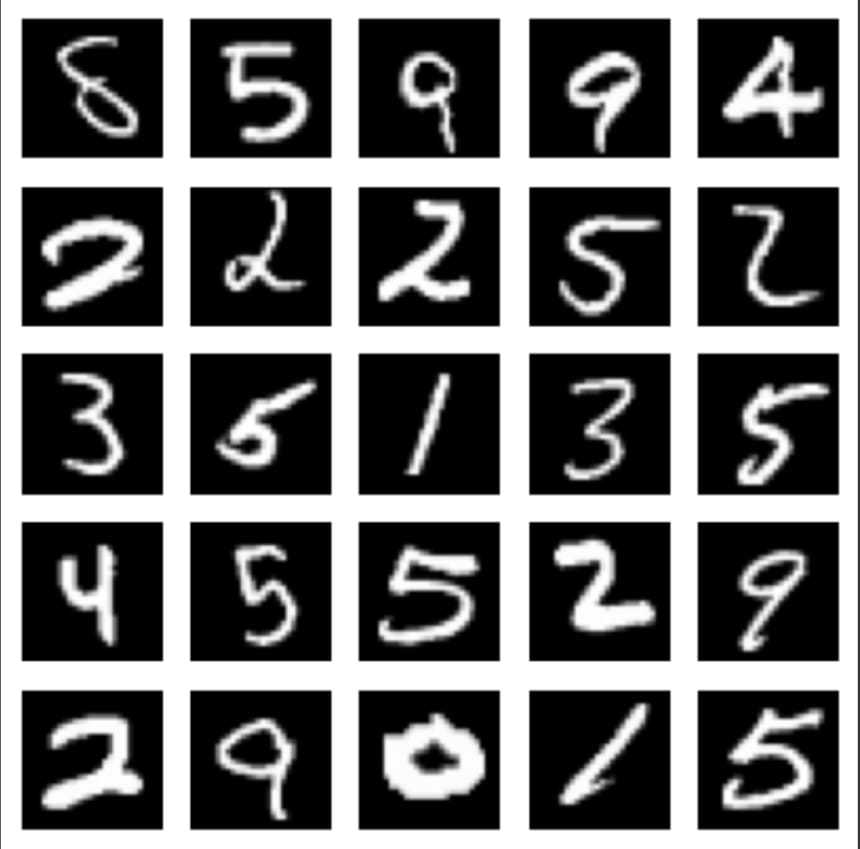
***Evaluation Metrics*:** Accuracy and loss were measured on both training and testing datasets.

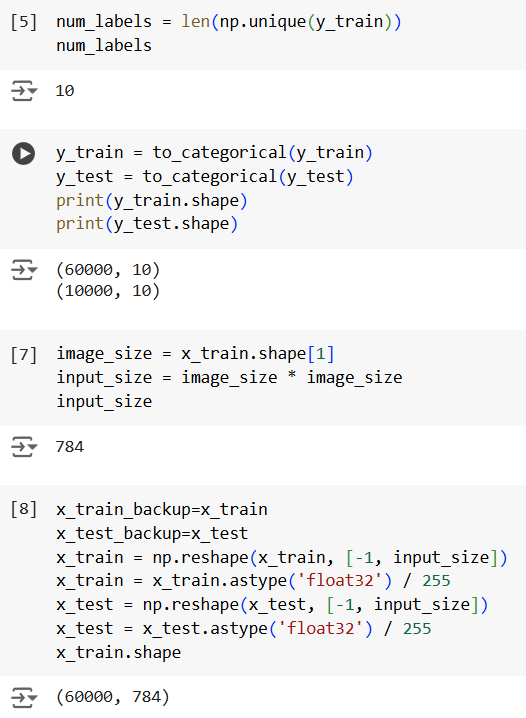
**IMPLEMENTATIONS:**

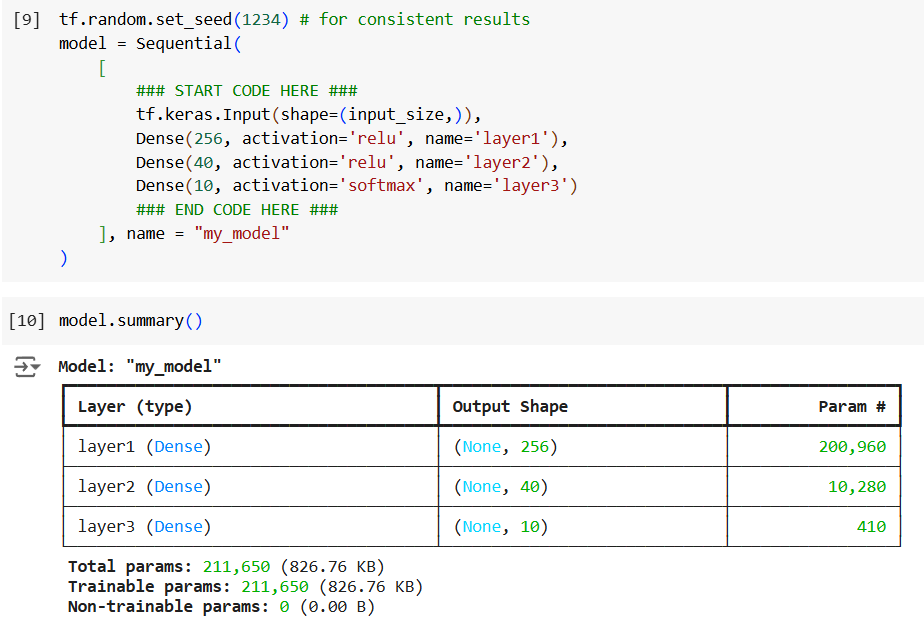
****

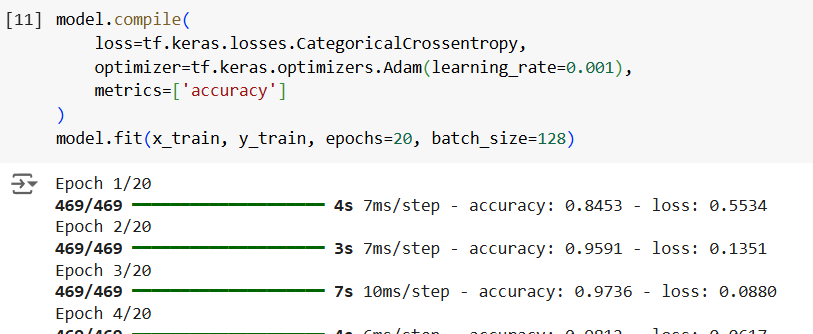
****

****

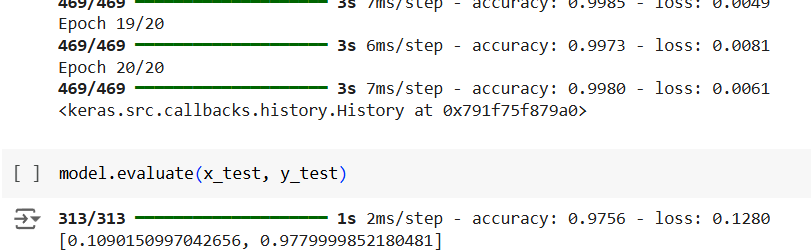
****

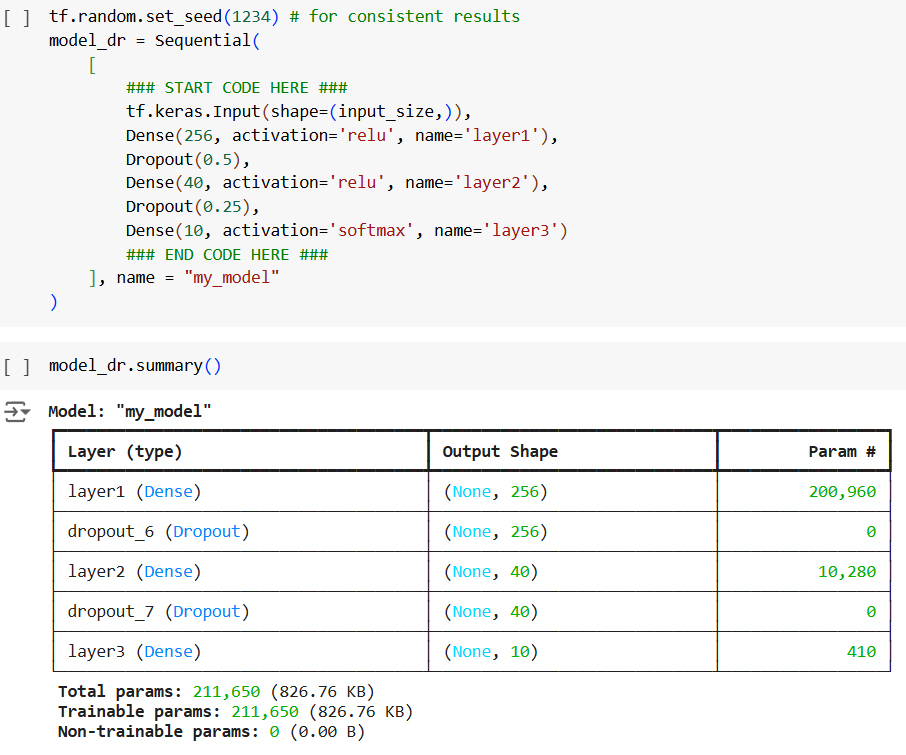
****

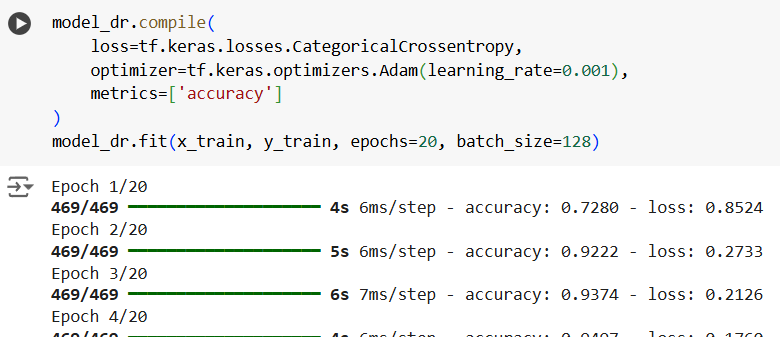
****

****

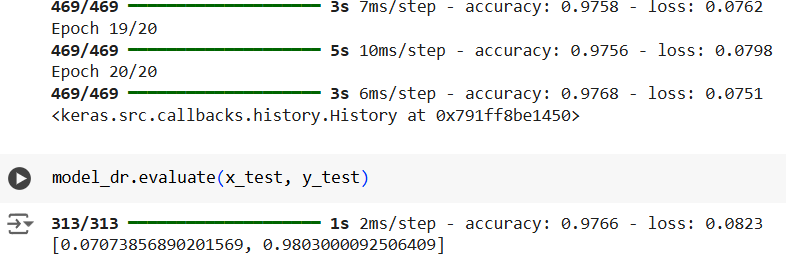
**……**

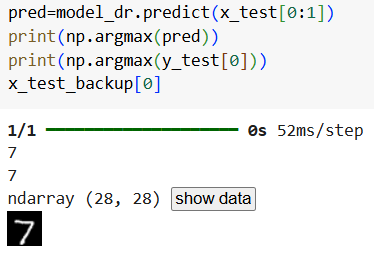
****

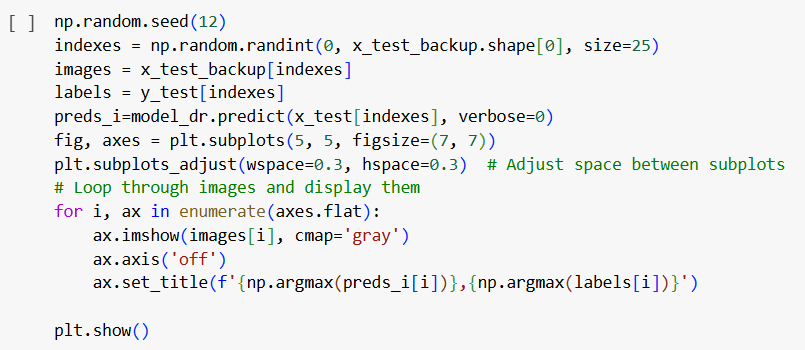
****

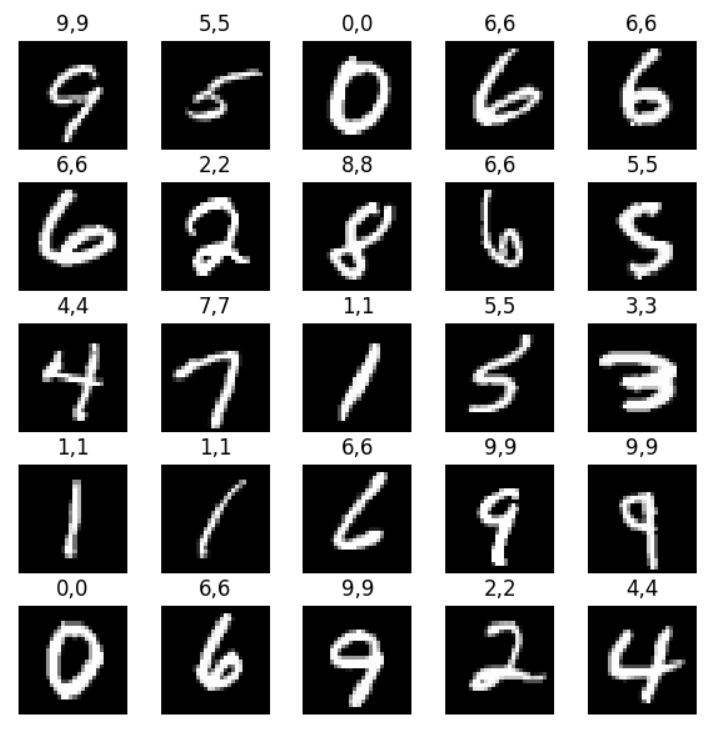
****

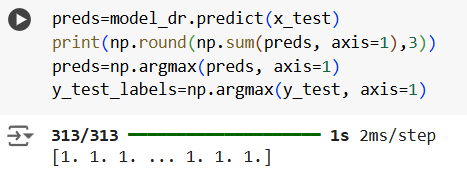
**…..**

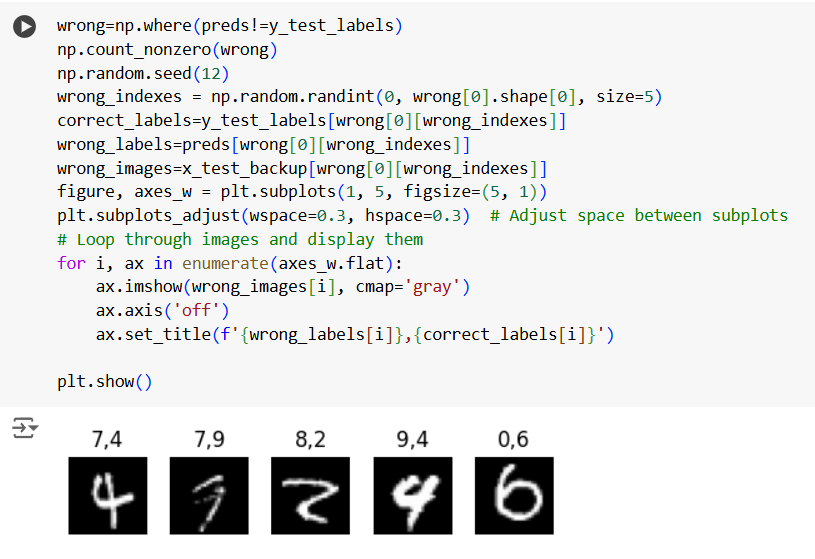
****

****

****

****

****

****

[**https://colab.research.google.com/drive/16IbnKEyqZ2H2X\_5zV4BEkjMuQwHvfD-H?usp=sharing**](https://colab.research.google.com/drive/16IbnKEyqZ2H2X_5zV4BEkjMuQwHvfD-H?usp=sharing) **(Link to the aforementioned code snippets and outputs)**

**PROGRAMMING TOOLS**

***Programming Language:*** Python**.**

***Libraries:*** TensorFlow, Keras, NumPy, Pandas, Matplotlib.

**STEP-BY-STEP MODEL BUILDING AND TRAINING**

1. ***Dataset Loading and Preprocessing:***

* ***Dataset***: The MNIST dataset was loaded using *keras.datasets*.
* ***Labels Distribution***: The *np.unique()* function counted unique occurrences of each digit (0-9) in the training and test datasets.
* ***Normalization***: Pixel values of images were normalized by dividing by 255 to bring them into the range [0, 1].
* ***Reshaping***: Images were reshaped from (28, 28) to (784,) to make them compatible with the dense layers.
* ***One-Hot Encoding***: Labels were converted into one-hot vectors using *to\_categorical() function*.

1. ***Model Architecture (Baseline Model):***

* ***Model***: Built using Keras' *Sequential* API.
* ***Layers***:

1. **Input Layer:** 784 features (28x28 flattened images).
2. **Hidden Layer 1**: 256 neurons with ReLU activation.
3. **Hidden Layer 2**: 40 neurons with ReLU activation.
4. **Output Layer:** 10 neurons (one for each digit) with softmax activation.

* ***Parameters***: A total of 211,650 parameters were trained.

**3. *Compilation and Training***:

* **Loss Function**: Categorical Cross-Entropy (*CategoricalCrossentropy).*
* **Optimizer**: Adam with a learning rate of 0.001.
* **Metrics**: Accuracy.
* **Training**: Model trained for 20 epochs with a batch size of 128. Therefore, each epoch will go through 60000/128 ≈ 469 iterations.

**4. *Evaluation:***

* The baseline model achieved an accuracy of 97.79% on the test set.

**5. *Dropout Model:***

***Modified Architecture:***

* Dropout (50%) after the first hidden layer.
* Dropout (25%) after the second hidden layer.
* This architecture was designed to combat overfitting.
* **Results:** Accuracy improved to 98.03 %.

***Code Overview***

* Loaded ***Baseline Model:***
* the MNIST dataset and performed EDA.
* Pre-processed data (flattening, normalization, and one-hot encoding).
* Defined the model architecture.
* Compiled and trained the model using Adam optimizer.
* Evaluated performance on the test set.

**RESULTS**

***Model Performance Metrics*:**

* ***Final Training Accuracy***: 99.80%
* ***Test Accuracy***: 97.79%
* ***Test loss****:* 0.1090

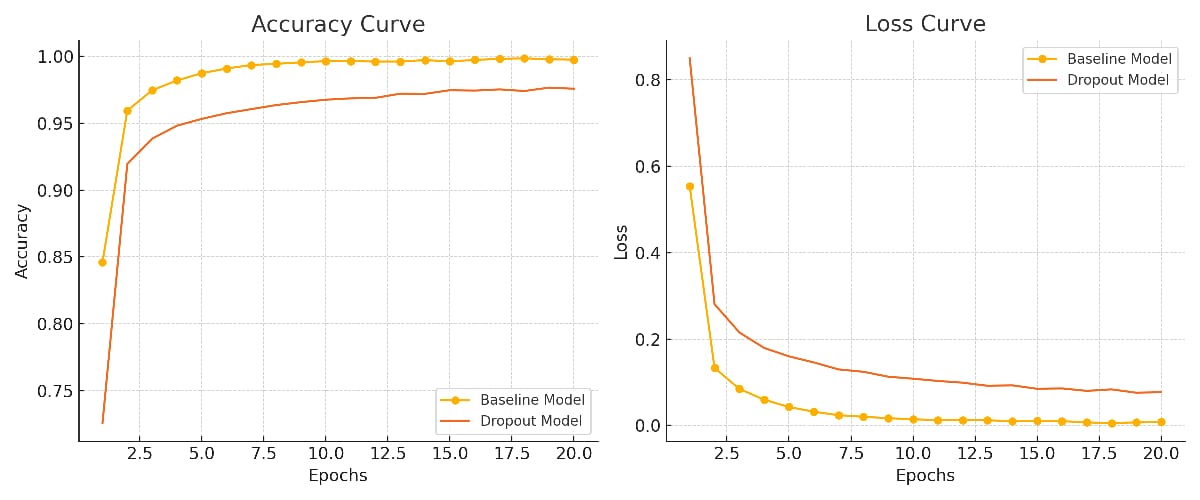
***Dropout Model****:*

* ***Final Training Accuracy***: 97.68%
* ***Test Accuracy***: 98.03%
* ***Test loss:*** 0.0707

Dropout significantly improved generalization by reducing overfitting.

***Visualizations*:**

1. ***Sample data:*** Visualised a grid of 25 MNIST digits.
2. ***Accuracy and Loss curve:***

****

* ***Accuracy Curve:*** Shows improvement over epochs.
* ***Loss Curve***: Demonstrates the decreasing trend of loss over time.

**DISCUSSION**

***Insights*:**

* The model performs very-well, achieving near state-of-the-art accuracy on the MNIST dataset.
* The dropout layers effectively reduce overfitting and improve test performance. Even though they don’t fit the training data as well but significantly improve test accuracy.

***Challenges*:**

* The dataset is well-balanced and relatively simple, so the main challenge was fine-tuning the architecture for optimal performance.
* Some erroneous examples may feed wrong training to model and also results in test error. It reminds us that the benchmark for our model is not test label but human accuracy on the data. Suppose the human accuracy is 99% on the dataset (which means an average human will correctly classify 99% of the entries) then a good model would be which provides around 99% accuracy on test data.

***Limitations*:**

1. ***Dataset*:**

* MNIST is not representative of real-world handwriting variability.
* Model performance might degrade on more challenging datasets.

1. ***Model*:**

* Fully connected networks are computationally expensive and less efficient compared to convolutional neural networks (CNNs) for image tasks.

1. ***Evaluation*:**

* Evaluation metrics are limited to accuracy, loss and multilabel confusion matrix. Additional metrics (e.g. precision, recall, F1-score) could provide deeper insights.

**CONCLUSION:**

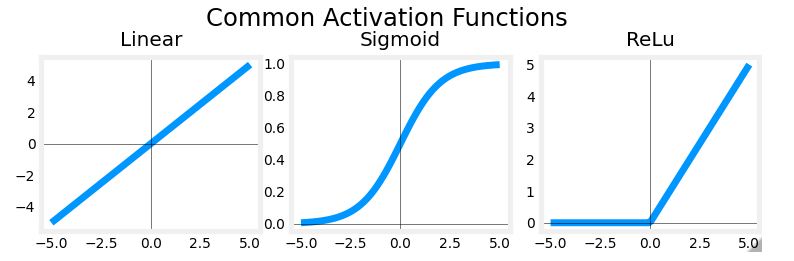
***Key Findings*:**

* The baseline and dropout - enhanced models performed well on the MNIST dataset.
* Dropout improved the model’s generalization ability by reducing overfitting.

***Future Scope*:**

* Extend the model to more challenging datasets, such as EMNIST or CIFAR-10.
* Incorporate a CNN architecture for improved performance on image data.
* Experiment with hyperparameter tuning and cross validation (e.g. learning rates, batch sizes) and other regularization techniques.

**APPENDIX:**

****

These are some common activation functions in TensorFlow library, two of which are employed in our model.

1. Linear Activation Function: f(x)=x. They are generally used in output layers for regression models.
2. Sigmoid activation function: . Used mostly in output layers for binary classification models.
3. ReLu activation function: f(x)= max (0, x). They are used for output layers of regression model where output cannot be negative however they work marvellously well for hidden layers and therefore, are the preferred type.

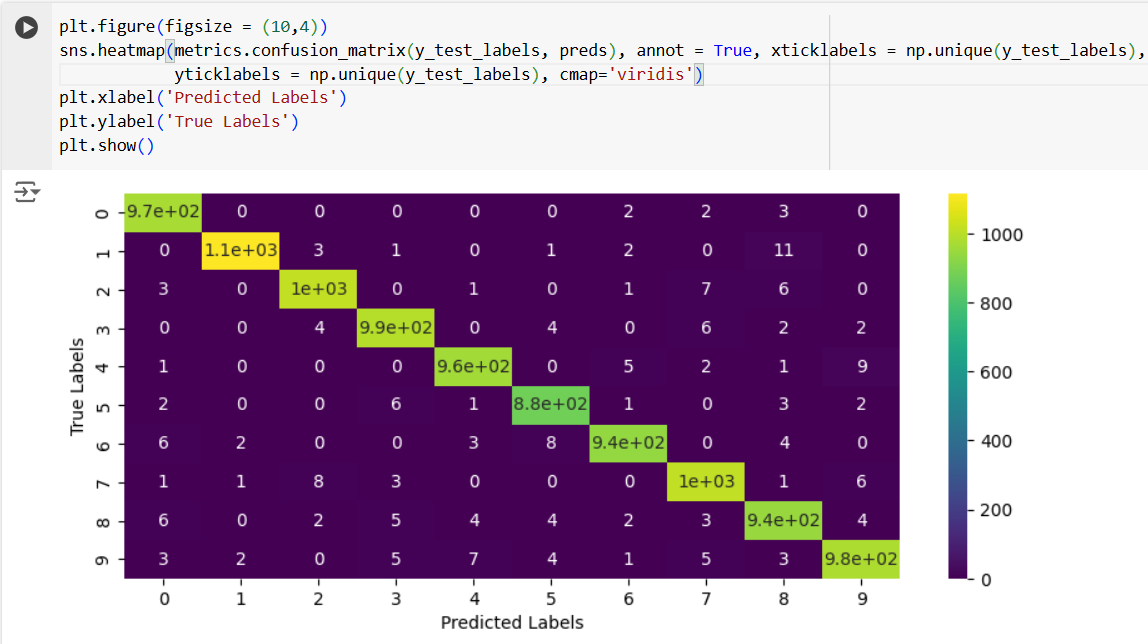
A fourth type of activation function which needs special mention is the SoftMax activation function which is basically a modification of sigmoid function and is used as an output layer activation function for multilabel classification problem chiefly because it does not have the 0 to +1 limit as in the sigmoid function. The upper and lower limit can be set to anything as per the programmer’s requirement. The modified equation is as follows: -

where, σ=SoftMax function

= input vector

K= no. of classes

i= required class

****

We learn from the above code snippet which aims to plot a multilabel confusion matrix using heatmap from seaborn library that, there are common areas where the model makes mistakes such as reading a 9 as a 4 and vice-versa, or reading 6 as a 5 or a 6 as a 0.

**REFERENCES:**

[**https://www.researchgate.net/publication/315053242\_Handwritten\_Character\_Recognition\_in\_English\_A\_Survey**](https://www.researchgate.net/publication/315053242_Handwritten_Character_Recognition_in_English_A_Survey)

[**https://ieeexplore.ieee.org/document/726791**](https://ieeexplore.ieee.org/document/726791)

[**https://www.cs.toronto.edu/~fritz/absps/ncfast.pdf**](https://www.cs.toronto.edu/~fritz/absps/ncfast.pdf)