**MINSIT classification**

**Assessment 1**

Abhinav Pillai

MAIB-JAN 23

Roll No.- aj23syd015

Contents

[Introduction 2](#_Toc147961113)

[Literature Review 2](#_Toc147961114)

[Methods 3](#_Toc147961115)

[Result: 7](#_Toc147961116)

[Challenges and problem during project 9](#_Toc147961117)

[**Reference:** 9](#_Toc147961118)

# Introduction

In the realm of computer vision, image classification is a fundamental and crucial task, with applications ranging from medical diagnostics to autonomous cars. Machines' capacity to correctly recognise and classify images is evidence of the effectiveness of neural networks, particularly Multilayer perceptron’s (MLP), in handling challenging visual identification issues.

This project aims to build an image classifier on the well-known MNIST (Modified National Institute of Standards and Technology) dataset utilising basic neural networks, specifically MLP. Each grayscale picture in the MNIST collection shows a handwritten digit from 0 to 9. It provides an excellent standard for assessing the efficiency of image classification systems, with 60,000 training photos and 10,000 testing images.

# Literature Review

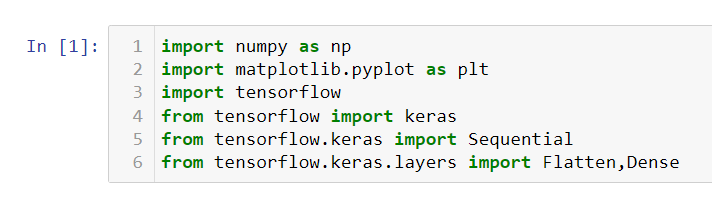
Deep learning approaches have been primarily responsible for the significant breakthroughs in picture classification that have occurred in recent years. Notably, because to their capacity to automatically learn hierarchical features from images, convolutional neural networks (CNNs) have elevated to the status of industry standard for image classification tasks. MLPs, on the other hand, provide a more straightforward yet efficient method for picture categorisation and serve as the foundation of our study.

Previous work on image classification often involves the exploration of different neural network architectures, optimization algorithms, and preprocessing techniques. Commonly used optimizers include Stochastic Gradient Descent (SGD), Adam, and RMSprop. These optimizers play a crucial role in adjusting the model's weights during training to minimize the loss function.

# Methods

**Task 1**: Import Libraries:

1. Import required libraries

****

1. import numpy as np: NumPy for numerical operations.
2. import matplotlib.pyplot as plt: Matplotlib for data visualization.
3. import tensorflow: TensorFlow for machine learning.
4. from tensorflow import keras: Keras API from TensorFlow.
5. from tensorflow.keras import Sequential: Sequential class for building neural networks.
6. from tensorflow.keras.layers import Flatten, Dense: Importing Flatten and Dense layers from Keras.

**Task 2**: Import Dataset:

1. Load dataset using keras API.

A screen shot of a computer

Description automatically generated

1. Dataset must be pre-processed before training the network, if you check image in

the training, you will see pixel value range from 0 to 255, scale these values range

from 0 to 1.



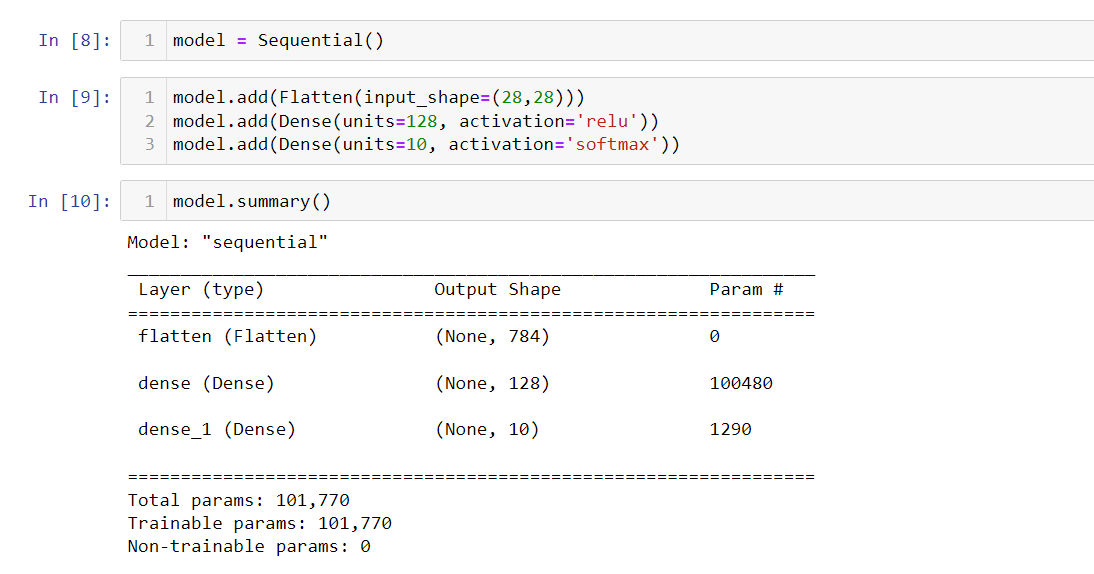
The normalization of pixel values in the provided code snippet is a crucial preprocessing step for machine learning models, particularly neural networks. In the context of image classification, pixel values ranging from 0 to 255 are normalized by dividing each value by 255.0. This normalization serves multiple purposes, including ensuring scale consistency across features, expediting convergence during optimization, preventing activation function saturation, and enhancing numerical stability. Dividing by 255.0 specifically scales the pixel values to a standardized range between 0 and 1, making them compatible with common activation functions and promoting floating-point precision. By normalizing the data, the machine learning model becomes more robust and efficient, contributing to improved training performance and overall effectiveness in handling diverse image datasets.

**Task 3:** Build a Classifier using MLP

1. The layer is the most fundamental component of a neural network. Data is put into

layers, and they extract representations from it. Choose the number of stacking

layers so that model representations should, be useful for the given task.



1. model = Sequential(): Creates a Sequential model for building feedforward neural networks.
2. model.add(Flatten(input\_shape=(28, 28))): Adds a Flatten layer as the input layer, transforming the 28x28 input into a 1D array (784 elements).
3. model.add(Dense(units=128, activation='relu')): Adds a hidden dense layer with 128 ReLU-activated neurons.
4. model.add(Dense(units=10, activation='softmax')): Adds the output layer with 10 units and softmax activation for multi-class classification.

**Task 4:** Compile the Model

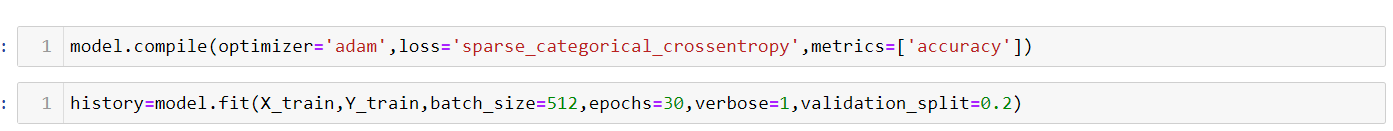
a. A few more parameters are required before the model can be used for training.

These are added at the build step of the model: Choose these parameters.

1 Loss function

2 Optimizer

3 Metrics



1. **model.compile(...)**: This line compiles the model, specifying the optimizer, loss function, and metrics to be used during training.

* **optimizer='adam'**: The Adam optimizer is chosen. Adam is an adaptive optimization algorithm that is widely used in training neural networks. It adjusts the learning rates of each parameter individually.
* **loss='sparse\_categorical\_crossentropy'**: This is the loss function used for training. In a classification task with integer labels (as opposed to one-hot encoded labels), 'sparse\_categorical\_crossentropy' is a suitable choice. It measures the difference between the true labels and the predicted probabilities.
* **metrics=['accuracy']**: During training, the accuracy metric is used to monitor the performance of the model.Top of Form

1. model.fit(...): This line trains the model on the training data.

* X\_train, Y\_train: The input features (X\_train) and corresponding labels (Y\_train) are provided for training.
* batch\_size=512: The training data is divided into batches of size 512. This is a common practice to speed up training and reduce memory requirements.
* epochs=30: The number of epochs specifies how many times the entire training dataset is processed by the model. In this case, training is performed for 30 epochs.
* verbose=1: This parameter controls the verbosity of the training output. Setting it to 1 means that progress bars will be displayed during training.
* validation\_split=0.2: 20% of the training data is used as a validation set. The model's performance on this set is monitored during training, providing insights into its generalization to unseen data.

**Task 5:** Train and Test the model.

a. Feed the training data to the built model.

b. Ask model to make predictions about a test set.

c. Verify that the predictions match the labels from the test labels.

A screenshot of a computer code

Description automatically generated

1. predictions = model.predict(X\_test): Uses the trained model to predict class probabilities for the test dataset.

2.np.argmax(predictions, axis=1): Finds the index of the maximum probability along axis 1, obtaining predicted class indices.

3.predicted\_classes: Array containing predicted class indices for each test sample.

4. print(predicted\_classes): Prints the array of predicted class indices.

A computer code with colorful text

Description automatically generated with medium confidence

1. num\_images\_to\_display = 5: Specifies the number of test images to display (set to 5).

2.for i in range(num\_images\_to\_display): Iterates over the specified number of images.

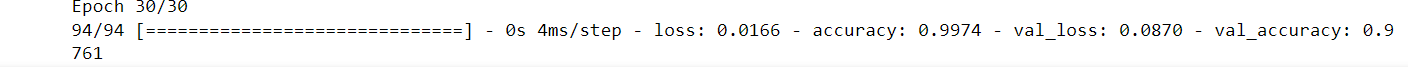
3.plt.imshow(X\_test[i].reshape(28, 28), cmap='gray'): Displays the i-th test image in grayscale.

4. plt.title(f'Predicted: {predicted\_classes[i]}'): Sets the title to include the predicted class.

5. plt.show(): Displays the image with its title.

# Result:

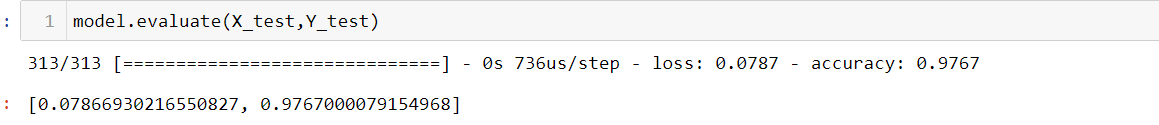




The training of the neural network was conducted using the provided training dataset (**X\_train** and **Y\_train**). After 30 epochs (iterations through the entire training dataset), the model achieved the following results:

* **Training Loss:** 0.0166
* **Training Accuracy:** 99.74%
* **Validation Loss:** 0.0870
* **Validation Accuracy:** 97.61%

These metrics indicate that the model performed exceptionally well on both the training and validation datasets. The low training loss and high training accuracy suggest effective learning on the training data. The validation results indicate strong generalization to new, unseen data, as evidenced by a relatively low validation loss and high validation accuracy



he **evaluate** method is used to assess the performance of the trained neural network on a specified test dataset. The line of code you provided, **model.evaluate(X\_test, Y\_test)**, returned the following results:

* **Test Loss:** 0.0787
* **Test Accuracy:** 97.67%

Here's a brief explanation:

* **Test Loss (0.0787):** This is a measure of how well the model is performing on the test data. It represents the average difference between the predicted and actual values. A lower loss indicates better performance.
* **Test Accuracy (97.67%):** This is the proportion of correctly classified instances in the test dataset. In this case, the model achieved an accuracy of 97.67%, meaning it correctly predicted the digit labels for approximately 97.67% of the test samples.

A screenshot of a computer code

Description automatically generated

It was able to predict it properly.

# Challenges and problem during project

1. Overfitting:
   * Overfitting occurs when the model performs well on the training data but fails to generalize to new, unseen data. This can be addressed by using techniques such as dropout, regularization, or reducing model complexity.
2. Data Preprocessing:
   * Ensuring that the dataset is preprocessed correctly is crucial. Issues such as inconsistent image sizes, label mismatches, or missing data can lead to unexpected errors.
3. Hyperparameter Tuning:
   * Selecting the right hyperparameters, such as the learning rate, number of layers, and neurons in each layer, is a trial-and-error process. Tuning these hyperparameters can significantly impact the model's performance.

# **Reference:**

1. <https://www.analyticsvidhya.com/blog/2021/11/newbies-deep-learning-project-to-recognize-handwritten-digit/>
2. <https://www.analyticsvidhya.com/blog/2021/06/a-beginners-guide-to-codeless-deep-learning-mnist-digit-classification/>
3. <https://medium.com/analytics-vidhya/multi-layer-perceptron-using-keras-on-mnist-dataset-for-digit-classification-problem-relu-a276cbf05e97>
4. <https://towardsdatascience.com/improving-accuracy-on-mnist-using-data-augmentation-b5c38eb5a903>