# Semester Project

Write your name
1. Prathyusha Reddy Thumma- (pthumma)
2. Kavya Sri Kasarla -(kkasarla)
3. Bharath chowdary Anumolu-(banumolu)[1]*

**Abstract**
**BACKGROUND:** Face recognition, handwritten digit recognition has become an appreciable area of research for the practical approaches but this task of making new approaches has become difficult due to the existing data problem and defect in the the data.So, this classification of faces and digits make automatic classification a difficult tasks. Classify task with higher accuracy and proposing models for dynamism data sets has become a challenging task.
**OBJECTIVE:** To most noteworthy limitations of low accuracy and data problems this project deals with building classifiers that will classify faces and digits.Based on on different modelings for a given data set we are going to train the data according to the requirements to gain accuracy and give the results.CNN's (Conv-Nets) utilize the foundational methods in mathematics for the purpose of of convolution in the domains such as: Computer Vision tasks.In the project, we use his technique as it provides much lower pre-processing in comparison to the other classification algorithms.With the use of regular NN's when we use the CNN we will have 3 dimensions as : width, Height, Depth.
**METHODS:** We have used CNN architecture for training and validation tasks with data-sets (MNIST). The proposed CNN model extract the features first and then perform classification tasks.
**CONCLUSION:** The proposed best CNN technique has a very simple architecture. This model provides higher accuracy for different data sets an the computational time is low. In comparison to past works the validation accuracy is also higher.

**Keywords**
**Convolution layer, Convolution Neural Network, MNIST data-set**

[1] *Luddy School of Informatics, Computing, and Engineering, Indiana University, Bloomington, IN, USA*

## Contents

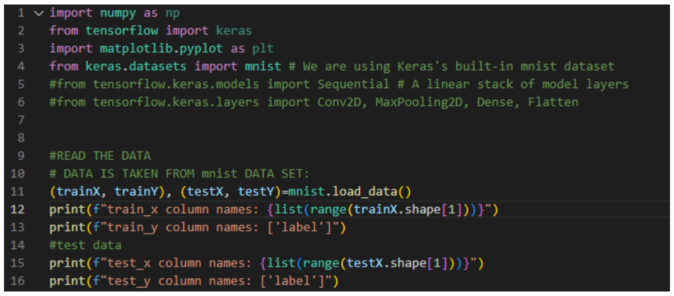
## 1. Problem and Data Description

Face Digit recognition and classification play sin important role in today's life. It has become the medium of communication to connect with people and to present someone's idea in the form of text. So, identification has become a necessary in all kinds of areas. The diversity often makes it difficult to recognize and read/identify. The necessity of recognition of images and digits is necessary and thus several deep learning algorithms were implemented. They provide better results than conventional methods and these algorithms require less time to recognize especially when we have large data-sets. For this project we have taken MNIST data-set , the reason that it is chosen is that, it has all the data pre-processed and when we apply the models then we could trigger the exact output for classification.

**DATA DESCRIPTION:**
MNIST data: 70000 garyscale images of handwritten digits (0-9)
Training data-set: 60000 images
Testing data-set: 10000 images
Image scaling: 28x28- pixel square represented as a 2D array of grayscale pixel range: 0-255
Standardization: pixel values are normalized
**Mean**= 0.1307
**Standard deviation**= 0.3081

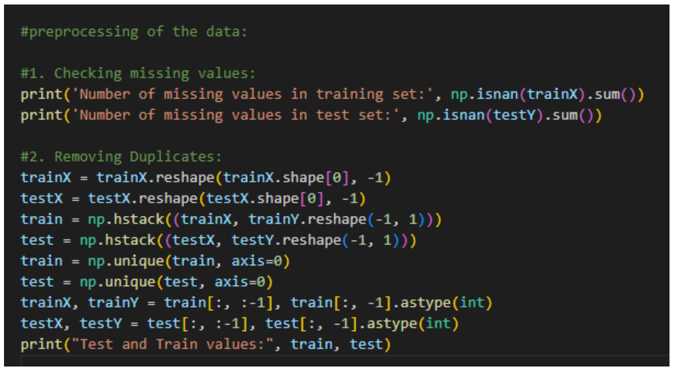
## 2. Data Preprocessing & Exploratory Data Analysis

The following steps are followed on MNSIT data-set for Data Pre-processing:
**1. Load the data:** In this code, we use the mnist.load$_{data}()$ $function from the keras.datasets module to load the MNIST dataset.$



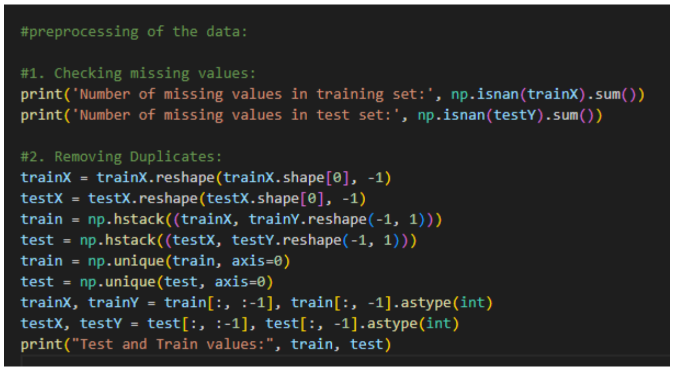
```
import numpy as np
from tensorflow import keras
import matplotlib.pyplot as plt
from keras.datasets import mnist # We are using Keras's built-in mnist dataset
#from tensorflow.keras.models import Sequential # A linear stack of model layers
#from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten


#READ THE DATA
# DATA IS TAKEN FROM mnist DATA SET:
(trainX, trainY), (testX, testY)=mnist.load_data()
print(f"train_x column names: {list(range(trainX.shape[1]))}")
print(f"train_y column names: ['label']")
#test data
print(f"test_x column names: {list(range(testX.shape[1]))}")
print(f"test_y column names: ['label']")
```

**2.Data cleaning:** MNIST dataset is a well-curated dataset that has already undergone some cleaning and preprocessing, there are still some common data cleaning steps that can be taken to improve the quality of the data. Some of these steps include:
1. Checking missing values: Check for any missing or corrupted data in the dataset and either remove the corresponding examples or impute the missing values with appropriate replacements.



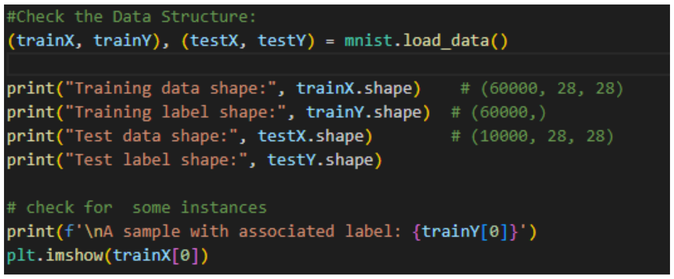
```
#preprocessing of the data:

#1. Checking missing values:
print('Number of missing values in training set:', np.isnan(trainX).sum())
print('Number of missing values in test set:', np.isnan(testY).sum())

#2. Removing Duplicates:
trainX = trainX.reshape(trainX.shape[0], -1)
testX = testX.reshape(testX.shape[0], -1)
train = np.hstack((trainX, trainY.reshape(-1, 1)))
test = np.hstack((testX, testY.reshape(-1, 1)))
train = np.unique(train, axis=0)
test = np.unique(test, axis=0)
trainX, trainY = train[:, :-1], train[:, -1].astype(int)
testX, testY = test[:, :-1], test[:, -1].astype(int)
print("Test and Train values:", train, test)
```

2. Removing duplicates: Check for any duplicate entries in the data-set and remove them to ensure that each image and label is unique.



```
#preprocessing of the data:

#1. Checking missing values:
print('Number of missing values in training set:', np.isnan(trainX).sum())
print('Number of missing values in test set:', np.isnan(testY).sum())

#2. Removing Duplicates:
trainX = trainX.reshape(trainX.shape[0], -1)
testX = testX.reshape(testX.shape[0], -1)
train = np.hstack((trainX, trainY.reshape(-1, 1)))
test = np.hstack((testX, testY.reshape(-1, 1)))
train = np.unique(train, axis=0)
test = np.unique(test, axis=0)
trainX, trainY = train[:, :-1], train[:, -1].astype(int)
testX, testY = test[:, :-1], test[:, -1].astype(int)
print("Test and Train values:", train, test)
```

3. Remove outliers
4. Checking for data consistency
5. Standardization
**3. Check the Data Structure:** The MNIST dataset can be loaded into Python using various libraries such as tensorflow, pytorch, or sklearn

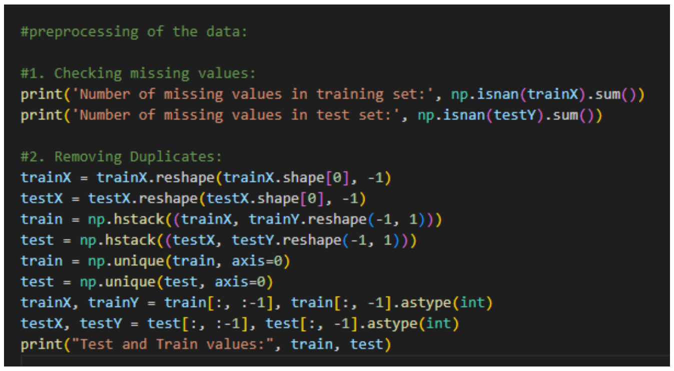


```
#Check the Data Structure:
(trainX, trainY), (testX, testY) = mnist.load_data()

print("Training data shape:", trainX.shape)    # (60000, 28, 28)
print("Training label shape:", trainY.shape)  # (60000,)
print("Test data shape:", testX.shape)        # (10000, 28, 28)
print("Test label shape:", testY.shape)

# check for  some instances
print(f'\nA sample with associated label: {trainY[0]}')
plt.imshow(trainX[0])
```

### 2.1 Handling Missing Values

### 2.2 Exploratory Data Analysis

Checking missing values: Check for any missing or corrupted data in the dataset and either remove the corresponding examples or impute the missing values with appropriate replacements.

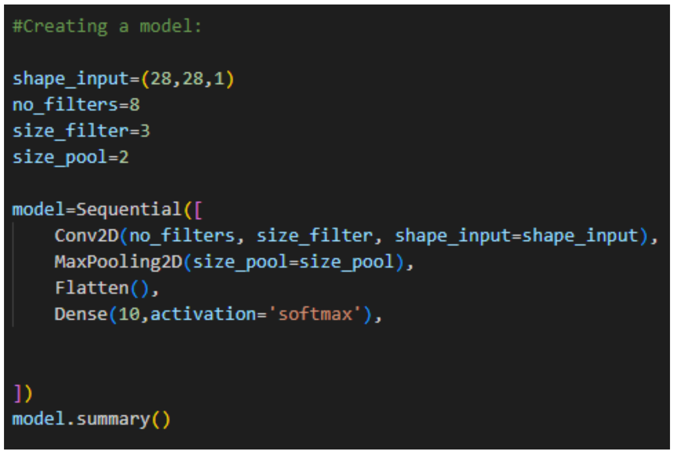


```
#preprocessing of the data:

#1. Checking missing values:
print('Number of missing values in training set:', np.isnan(trainX).sum())
print('Number of missing values in test set:', np.isnan(testY).sum())

#2. Removing Duplicates:
trainX = trainX.reshape(trainX.shape[0], -1)
testX = testX.reshape(testX.shape[0], -1)
train = np.hstack((trainX, trainY.reshape(-1, 1)))
test = np.hstack((testX, testY.reshape(-1, 1)))
train = np.unique(train, axis=0)
test = np.unique(test, axis=0)
trainX, trainY = train[:, :-1], train[:, -1].astype(int)
testX, testY = test[:, :-1], test[:, -1].astype(int)
print("Test and Train values:", train, test)
```

## 3. Algorithm and Methodology

**Creating a Model:** Creating a model on the MNIST data set involves building a machine learning or deep learning model that can accurately classify the handwritten digits in the data set. We use simple Convolution Neural Network (CNN) using ” tensorflow” to classify the data.
**The CNN architecture consists of the following layers:**
**1.Conv2D:** This layer applies convolution to the input image using the specified number of filters and filter size.
**2.MaxPooling2D**: This layer performs max pooling operation over a 2x2 window to reduce the spatial dimensions of the output from the convolution layer.
**3.Flatten**: This layer flattens the output from the previous layer, converting it into a one-dimensional vector.
**4.Dense**: This layer applies a fully connected neural network with 10 output units and softmax activation function. The inputShape variable defines the shape of the input image as 28x28 with one channel (grayscale). The numberOfFilters variable specifies the depth of the convolutional layer, which is set to 8. The filterSize variable specifies the size of the filter, which is set to 3x3. The poolSize variable specifies the size of the max pooling window, which is set to 2x2.

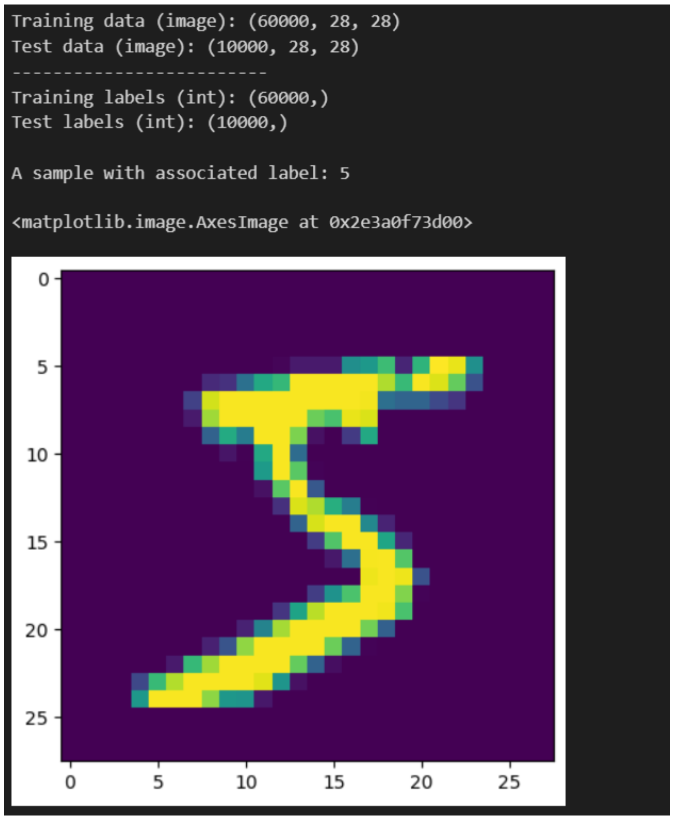
The model.summary() function prints a summary of the CNN architecture, including the shape of the output from each layer and the total number of parameters in the model.



```
#Creating a model:

shape_input=(28,28,1)
no_filters=8
size_filter=3
size_pool=2

model=Sequential([
    Conv2D(no_filters, size_filter, shape_input=shape_input),
    MaxPooling2D(size_pool=size_pool),
    Flatten(),
    Dense(10,activation='softmax'),


])
model.summary()
```

## 4. Experiments and Results

**1. Data Structure:**For every data-set we train we need to check the data structure so as to fit all the attributes without missing values:

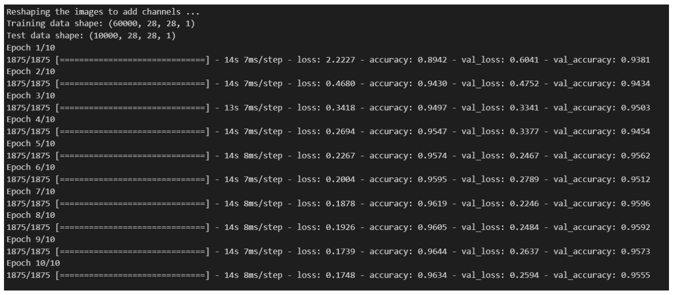


```
Training data (image): (60000, 28, 28)
Test data (image): (10000, 28, 28)
-------------------------
Training labels (int): (60000,)
Test labels (int): (10000,)

A sample with associated label: 5

<matplotlib.image.AxesImage at 0x2e3a0f73d00>
```

**2. Model implementation:** The shape-input variable is a tuple that specifies the shape of the input images. In this case, the input images are 28x28 grayscale images, so shape-input is set to (28, 28, 1).
The no-filters variable specifies the number of filters to use in the convolutional layer. In this case, we use 8 filters.
The size-filter variable specifies the size of the filters. In this case, the filter size is 3x3.
The size-pool variable specifies the size of the max pooling window. In this case, the max pooling window is 2x2.
The model object is created using the Sequential class from Keras. This class allows us to define a linear stack of layers.
The first layer in the model is a Conv2D layer. This layer applies convolutional filters to the input images. We specify the number of filters and the filter size using the no-filters and size-filter variables, and we pass in the shape-input variable to specify the shape of the input images.
The second layer in the model is a MaxPooling2D layer. This layer performs max pooling on the output from the convolutional layer. We specify the size of the max pooling window using the size-pool variable. The third layer in the model is a Flatten layer. This layer flattens the output from the previous layer into a one-dimensional vector.
The fourth and final layer in the model is a Dense layer with 10 output units and a soft max activation function. This layer applies a fully connected neural network to the flattened output from the previous layer, and the softmax activation function produces a probability distribution over the 10 possible classes. The model.summary() function prints a summary of the model

architecture, including the shape of the output from each layer and the total number of parameters in the model. This can be useful for debugging and optimizing the model.

**3. Training the Model:**



```
Reshaping the images to add channels ...
Training data shape: (60000, 28, 28, 1)
Test data shape: (10000, 28, 28, 1)
Epoch 1/10
1875/1875 [==============================] - 14s 7ms/step - loss: 2.2227 - accuracy: 0.8942 - val_loss: 0.6041 - val_accuracy: 0.9381
Epoch 2/10
1875/1875 [==============================] - 14s 7ms/step - loss: 0.4680 - accuracy: 0.9430 - val_loss: 0.4752 - val_accuracy: 0.9434
Epoch 3/10
1875/1875 [==============================] - 13s 7ms/step - loss: 0.3418 - accuracy: 0.9497 - val_loss: 0.3341 - val_accuracy: 0.9503
Epoch 4/10
1875/1875 [==============================] - 14s 7ms/step - loss: 0.2694 - accuracy: 0.9547 - val_loss: 0.3377 - val_accuracy: 0.9454
Epoch 5/10
1875/1875 [==============================] - 14s 8ms/step - loss: 0.2267 - accuracy: 0.9574 - val_loss: 0.2467 - val_accuracy: 0.9562
Epoch 6/10
1875/1875 [==============================] - 14s 7ms/step - loss: 0.2004 - accuracy: 0.9595 - val_loss: 0.2789 - val_accuracy: 0.9512
Epoch 7/10
1875/1875 [==============================] - 14s 8ms/step - loss: 0.1878 - accuracy: 0.9619 - val_loss: 0.2246 - val_accuracy: 0.9596
Epoch 8/10
1875/1875 [==============================] - 14s 8ms/step - loss: 0.1926 - accuracy: 0.9605 - val_loss: 0.2484 - val_accuracy: 0.9592
Epoch 9/10
1875/1875 [==============================] - 14s 7ms/step - loss: 0.1739 - accuracy: 0.9644 - val_loss: 0.2637 - val_accuracy: 0.9573
Epoch 10/10
1875/1875 [==============================] - 14s 8ms/step - loss: 0.1748 - accuracy: 0.9634 - val_loss: 0.2594 - val_accuracy: 0.9555
```

## 5. Deployment and Maintenance

This phase has to be implemented in further check-In's.

## 6. Summary and Conclusions

From above project description we have 2 domains of classification as of : **Digit classification  Face classification.**. In this **Check-in phase-I** we have implemented the model (CNN) for digit classification.The major tasks that we accomplished in this phase are:
1. Data Preprocessing
2. Data cleaning
3. Check Data Structure
4. Check sample instance
5. Create a model
6. Compile the model
7. Train the model
8. Predict the model

**Tasks that should be accomplished in next phase evaluation:**
Deploy the model and check the same for different data sets to ensure the accuracy for the model that we have developed and we need to work from the initial model creation if it fails to work on data-sets.

## Acknowledgments

We would like to thank Professor and all the Teaching Assistants who gave us idea on how to proceed with the outline of the project and detailing on working sources.

## References

**1. https://www.analyticsvidhya.com/blog/2021/07/classification-of-handwritten-digits-using-cnn/**
**2.https://arxiv.org/ftp/arxiv/papers/2004/2004.00331.pdf**
**3. https://link.springer.com/article/10.1007/s42452-019-1161-5**