INFO 5502

Principles and Techniques for Data Science

**Prof. Dr. Ting Xiao**

**IDENTIFICATION OF HANDWRITTEN DIGITS-USING THE MNIST DATASET**

**Group Members:**

|  |  |  |
| --- | --- | --- |
| Full Name | UNT ID | E-Mail |
| Bharath Kumar Sunkari | 11504574 | bharathkumarsunkari@my.unt.edu |
| Neeraj Mothukuri | 11549636 | dhamodarneerajmothukuri@my.unt.edu |
| Sai Krishna Dasineni | 11549138 | saikrishnadasineni@my.unt.edu |
| Sampath Potluri | 11516629 | sampathpotluri@my.unt.edu |

**CONTENTS**

1. **Abstract**
2. **Group Collaboration**
3. **Workflow**
4. **Data Specification**
5. **Project Design**
6. **User Interface Development**
7. **Project Milestone**
8. **Project Results**
9. **Repository**
10. **Code**
11. **Reference Material**

**Abstract:**

Humans detect and visually experience the world around them using their eyes and brains. Computer vision aims to make computers capable of observing and evaluating images in the same way that humans do. One of the difficult approaches in pattern recognition applications is recognizing handwritten numbers. Digit recognition applications include postal codes or data form filling, to mention a few. The primary goal is to create a model that can better recognize and determine a handwritten number based on its appearance. We're using an MNIST, which is a dataset of the MNIST database for our project.

The MNIST database (Modified [National Institute of Standards and Technology](https://en.wikipedia.org/wiki/National_Institute_of_Standards_and_Technology) database) is a large database of handwritten digits that is commonly used for training various image processing systems. The database is also widely used for training and testing in the field of machine learning[[1]](https://en.wikipedia.org/wiki/MNIST_database#cite_note-1).

We'll look at a dataset that has a set of pixel values, and we'll utilize visualization techniques to display them as indicated in the diagram below. Following the digit visualization, we will create a machine-learning model that includes the Cross Validation technique for the best accuracy of the graphically represented digits.

Graphical user interface, application

Description automatically generated

Digits Visualized based on pixel value from the MNIST Dataset

**Group Collaboration:**

**Roles:**

|  |  |  |
| --- | --- | --- |
| Full Name |  | Role Overview |
| Bharath Kumar Sunkari |  | Choosing the Dataset and Pre-Processing |
| Neeraj Mothukuri |  | Developing the suitable Machine Learning Model |
| Sai Krishna Dasineni |  | Developing the Model and User Interface |
| Sampath Potluri |  | Calculating the performance and accuracy of the developed model |

**Bharat Kumar Sunkari:** Worked effectively in choosing the dataset and pre-processing the data and involved while developing the model.

**Neeraj Mothukuri:** Took initiative to develop the machine learning model from the stage of the choosing the right algorithm and training and testing the data for the best results.

**Sai Krishna Dasineni:** Involved in developing the model and in addition developed a User Interface using web-frameworks, where the user can upload the image and predict the hand-written digit.

**Sampath Potluri:** worked on evaluating the model performance like Plotting the Confusion Matrix, Accuracy Score, Precision, Recall and F1 Score.

In Overall, all the members contributed to documentation for the project and created a GIT Repository to access the project files.

**WORKFLOW:**

**Choosing the Dataset and Pre-Processing:**

For this project instead of going through multiple datasets to choose and then assess if the dataset is suitable for our project purpose which would take up a lot of time so we decided to use “TensorFlow MNIST” which has “Keras API” which imports the dataset directly that we chose. By doing this we have saved the processing time of the system.

Our data set consists of an array of images hence in the pre-processing phase we did the re-shaping of the images as required for grey scaling as the images are generally in color. And then we changed the values from int type to float type.

**Developing the suitable Machine Learning Model:**

After the pre-processing phase, we came to the development phase of the model. In this phase, we made an elaborate study on the various algorithms which will give the most accurate result for image classification.

We chose to go with the CNN algorithm, a Convolutional neural network that is a deep learning algorithm, this algorithm takes images as input, and it is capable to differentiate between the images given. We used this algorithm and defined a model.

We assigned 75% of the data for training the dataset and we assigned the rest of the 25% of the dataset for testing the model. In the phase, we have compiled the whole model and trained the model. Then came predicting the class label of every single image. Then we Predicted the values of the class label of the test data.

**Calculating the performance and accuracy of the developed model :**

Once the model was developed and trained, we started by testing its performance of the model.

To test our model and observe whether the results are accurate enough, we calculated the performance of the accuracy of actual label values and predicted label values. For that, we plotted a confusion matrix for actual label values and predicted label values, where we tried to find out how well did the model predict the values when it came to visualizing the values.

Once we observed that the model is accurate enough which means the predicted label values and actual values had very few wrong predictions giving out a very high success rate from the confusion matrix and then we calculated precision, recall, and f1 score.

**Data Specification :**

The dataset we chose for the project is the MINST data set. Our data set consists of an array of images. In simple terms, our data set has images that are in an array of values from 0 to 255 in which 0 represents black and 255 represents white. The dataset consists of 70,000 images and we used 52,500 images for the purpose of training the dataset and the rest of 17,500 images were used for the purpose of testing the model where the images are applied to the developed model and predicting the values to the trained data.

The size of each image in the dataset is 28\*28 pixels. The images are stored in the form of array of integers. We are using a supervised leaning technique of machine learning which is CNN algorithm.

The dataset we are using consists of images as array of values, each image has 28 rows and 28 columns. When the value appears to be 0 or any closer value to 0 that means the color is black or light shade of black as it moves further away and when the value is closer to 255 or 255 it will be white and darker shade of white as it moves further away.

**Project Design:**

In our project development the one of the tools we used are Anaconda, we used anaconda as we know we will in the need of numerous data science packages for computing, management of the development and deployment of the project and in anaconda we used Jupiter notebook for writing the code.

Diagram

Description automatically generated

**Loading and Splitting the Dataset:** First we will load the dataset and further we concatenate the 60000 training images and 10000 test images to total 70000 images. Then, we will split the images using with rule 75/25 using Scikit-Learn. That is 75% of images for training purpose and the remaining 25% of images for testing purpose.

Graphical user interface, text, application, email

Description automatically generated

Selecting 9 random images and displaying them with class labels:

A picture containing text

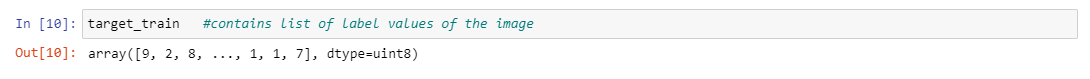
Description automatically generated

The below matrix shows how the grayscale images are stored in an array. Here the 0 represent as black and the maximum value 255 represent white color:

A picture containing chart

Description automatically generated

Showing the labels data of training images:



**Preprocessing the Dataset:**

Initially, we will perform reshaping the images as (52500, 28,28, 1) that model requires. Here the 52500 training images and 28, 28 represents array size and 1 represent the grayscale. Another 17500 test images are reshaped (17500, 28, 28, 1) and converted similarly into float for both testing and training images. Now, for both training and testing images will undergo normalization which means all the image values are divided by 255.0 and stored all the values between 0 to 1.

Graphical user interface, text, application

Description automatically generated

We do encode the labels for both train and test images, where in an array of ten 0’s the label number will be stored in that position. For example, label 7 will be stored as [0,0,0,0,0,0,0,1,0,0]



**Developing the Model:**

Our input images are defined by default into 28x28 pixel size. . Pixel array serves as input layer which has 28\*28=784 pixels. Further, it will process and divide into two hidden layers. Where each hidden layer consists of 512 nodes and then it will be processed into output layer having ten values.

Table

Description automatically generated

**Training the model:**

Here the batch\_size defines the number of samples to be used for the model and epochs defines the number of times that we want to iterate the whole training set. So, a greater number of epochs gives us the best accuracy.

Table

Description automatically generated

**Prediction:**

Prediction is done for the test images

Background pattern

Description automatically generated

**Model Performance:**

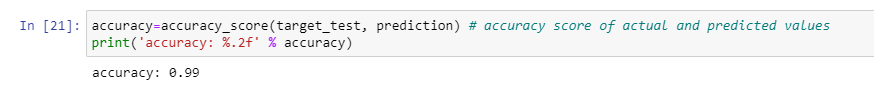
**Confusion Matrix:**

It is an N x N matrix used for evaluating the performance of a classification model, where N is the number of target classes.

Chart

Description automatically generated

**Accuracy:** Accuracy score is calculated as the ratio between the number of correctly predicted values to the total number of values between predicted and actual values.



**User Interface Development:**

We have taken of advantage of developing the User-Interface where we can present the project easily understandable.

To develop the web page, we have used **React.js** which is well known for developing the single page web applications and **Flask** web-framework, because the react and flask is having very good response for their combination in developing the web applications.

To begin with, we have created a basic html file with all the requirements to be displayed on the web page such as Choose File to upload the image, to show the image what has been uploaded such as preview and a Predict button to submit and few animations such as Fade In, Loader to look good.

Text

Description automatically generated

This is the Predict.py file (can be referred from GIT Repository). It has number of required libraries imported and made use of them. Using the Flask, we trying to communicate the between python and HTML using the template. We make use of GET and POST methods to communicate with each other for the actions performed. Initially, we run the Predict.py file with the model and when model is successfully executed that means the model is trained and ready to predict the values. To decrease the run time while predicting the values, we will store the model code separately in another .py file and the model method is stored in a .h5 file “Keras saves models in this format as it can easily store the weights and model configuration in a single file”. So, whenever we need to predict a new value, we don’t need to run the model again and again. We can simply load the .h5 file and predict the value which is lot more time saving and very much responsive in terms of web applications.

Text

Description automatically generated

We make use of AJAX, to receive the outputs from python where the prediction is done and represents the output on the web page. This work both ways, as sending actions from HTML to Python.

Text

Description automatically generated

**User Interface Outputs:**

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

**Project Milestones** :

1. Convert the images from color to grayscale.
2. Select an algorithm for development of the model.
3. Receiving the accuracy of 99%.
4. Developing an User-Interface to easily upload and predict the value.

**Project Results:**

All images in the form of array values ranging from 0-255 are transformed to values 0-1. The image is given as an input and the model we developed predicts the digit value. Assuming we had more time and more data to work on we can increase the functioning of our model. As our current design can take the input of one image at a time and predict a single digit at a time, we think our design can take an image that consists of more than 1 digit and predict all the digits in the image for an example the image can be a number plate of a vehicle.

**GIT Repository:** <https://github.com/saikrishna59/python_final_project>

**Code:**

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

import random

import tensorflow

import seaborn as sns

from tensorflow import keras

from keras.datasets import mnist

from keras.utils.np\_utils import to\_categorical

from keras.models import Sequential

from keras import layers

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import precision\_score, recall\_score, f1\_score, accuracy\_score

from keras.preprocessing.image import load\_img

from keras.preprocessing.image import img\_to\_array

from sklearn.metrics import confusion\_matrix

# ## \*\*Loading and splitting the dataset\*\*

(trainX, trainy), (testX, testy) = mnist.load\_data()

train = np.concatenate((trainX,testX))

test = np.concatenate((trainy,testy))

print(len(train))

print(len(test))

data\_train, data\_test, target\_train, target\_test = train\_test\_split(train,test,stratify=test, test\_size=0.25,random\_state=42)

print(len(data\_train),len(target\_train))

print(len(data\_test),len(target\_test))

# Selecting 9 random images and displaying them with class labels

plt.rcParams['figure.figsize'] = (9,9)

for i in range(9):

plt.subplot(3,3,i+1)

r = random.randint(0, len(data\_train))

plt.imshow(data\_train[r], cmap='gray', interpolation='none')

plt.title("Class {}".format(target\_train[r]))

plt.tight\_layout()

# Matrix shows how grayscale images are stored in a array

def matprint(mat, fmt="g"):

col = [max([len(("{:"+fmt+"}").format(x)) for x in col]) for col in mat.T]

for x in mat:

for i, y in enumerate(x):

print(("{:"+str(col[i])+fmt+"}").format(y), end=" ")

print("")

r = random.randint(0, len(data\_train))

matprint(data\_train[r])

target\_train #contains list of label values of the image

# ## \*\*Preprocessing the dataset\*\*

def preproc(data\_train,data\_test):

data\_train = data\_train.reshape((data\_train.shape[0], 28, 28, 1))

data\_test = data\_test.reshape((data\_test.shape[0], 28, 28, 1))

data\_train = data\_train.astype('float32') # change integers to 32-bit floating point numbers

data\_test = data\_test.astype('float32')

data\_train /= 255.0 # normalize each value for each pixel for the entire vector for each input

data\_test /= 255.0

return data\_train, data\_test

data\_train,data\_test= preproc(data\_train,data\_test)

print("Training matrix shape", data\_train.shape)

print("Testing matrix shape", data\_test.shape)

classes =len(np.unique(target\_train))

y\_train = to\_categorical(target\_train,classes) #Categorically encode the labels

y\_test = to\_categorical(target\_test,classes)

# ## \*\*Defining the model\*\*

model = keras.Sequential(

[

keras.Input(shape=(28,28,1)),

#first hidden layer

layers.Conv2D(32, kernel\_size=(3, 3), activation="relu"),

layers.MaxPooling2D(pool\_size=(2, 2)),

#second hidden layer

layers.Conv2D(64, kernel\_size=(3, 3), activation="relu"),

layers.MaxPooling2D(pool\_size=(2, 2)),

layers.Flatten(),

#output layer

layers.Dropout(0.5),

layers.Dense(classes, activation="softmax"),

]

)

model.summary()

# # \*\*Compilation\*\*

# compile model

model.compile(loss="categorical\_crossentropy", optimizer="adam", metrics=["accuracy"])

# ## \*\*Train the model\*\*

batch\_size = 128

epochs = 15

model.fit(data\_train, y\_train, batch\_size=batch\_size, epochs=epochs, validation\_split=0.1)

# ## \*\*Model Performance\*\*

score = model.evaluate(data\_test,y\_test)

print('Test loss:', score[0])

print('Test accuracy:', score[1])

predicts = model.predict(data\_test)

prediction = []

for i in predicts:

prediction.append(np.argmax(i))

print(prediction)

print(target\_test)

print("predicted values Actual values") # comparing first 10 predicted and actual values

for i in range(10):

print(prediction[i],"\t\t\t",target\_test[i])

accuracy=accuracy\_score(target\_test, prediction) # accuracy score of actual and predicted values

print('accuracy: %.2f' % accuracy)

plt.figure(figsize=(12,7))

cm = confusion\_matrix(target\_test,prediction)

ax = sns.heatmap(cm, annot=True, fmt="d")

plt.xlabel('True label')

plt.ylabel('Predicted label')

#Of all the labels that model predicted, what is the percentage of them are correct

precision = precision\_score(target\_test, prediction, average='micro')

print('precision: %.3f' % precision)

#Of all the actual labels, what is the percentage of them are predicted correctly

recall=recall\_score(target\_test, prediction, average='micro')

print('Recall: %.3f' % recall)

score = f1\_score(target\_test, prediction, average='micro')

print('F-Measure: %.3f' % score)

# # \*\*Predicting class label of a single image\*\*

img = load\_img("number.png", grayscale=True, target\_size=(28, 28))

# convert to array

img = img\_to\_array(img)

# reshape into a single sample with 1 channel

img = img.reshape(1,28, 28, 1)

# prepare pixel data

img = img.astype('float32')

img = img / 255.0

val = model.predict(img)

predicted\_value = np.argmax(val)

print(predicted\_value)

**PDF:**

****

**Reference Material:**

[1] Model Performance: <https://www.analyticsvidhya.com/blog/2020/04/confusion-matrix-machine-learning/#:~:text=A%20Confusion%20matrix%20is%20an,by%20the%20machine%20learning%20model>.

[2] Flask File Uploading: <https://www.javatpoint.com/flask-file-uploading>

[3] H5 file: <https://www.edureka.co/community/77292/what-is-h5-model-in-keras>