PROJECT REPORT

JULY, 2022

HANDWRITTEN EQUATION RECOGNITION AND SOLVER USING CONVOLUTIONAL NEURAL NETWORKS

Submitted in partial fulfillment of the requirements for the degree of

Bachelor of Technology

in

ELECTRONICS AND COMMUNICATION ENGINEERING

SHUBHAM KUMAR

ECE A

SEMESTER 6TH

ROLL: 251901004

Under the guidance of

MS. SHEFALI DHINGRA

Assistant Professor

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

UNIVERSITY INSTITUTE OF ENGINEERING AND TECHNOLOGY

KURUKSHETRA UNIVERSITY

CERTIFICATE

UNIVERSITY INSTITUTE OF ENGINEERING AND TECHNOLOGY
KURUKSHETRA UNIVERSITY

It is certified that **Shubham Kumar**, student of Bachelors of Technology (Electronics and Communication Engineering), under the class roll number 251901007, for the session 2019-23, has completed the project entitled **"HANDWRITTEN EQUATION RECOGNITION AND SOLVER USING CONVOLUTIONAL NEURAL NETWORKS"** under my supervision. The project report is, in my opinion, worthy of consideration in accordance with the rules and the regulations of the University Institute of Engineering and Technology, Kurukshetra University, Kurukshetra.

I wish them all success in their future endeavors.

MS. SHEFALI DHINGRA
ASSISTANT PROFESSOR
UIET-KU

ACKNOWLEDGEMENT

A journey is easier when you are traveling together in the right direction. Interdependence is certainly more valuable than interdependence. This project is the result of work whereby we have been accomplished and supported by many people. It is a pleasant aspect that now we have the opportunity to express our gratitude to all of them.

Foremost, we would like to express our deepest gratitude to our professor Ms. Shefali Dhingra, Department of Electronics and Communication Engineering, University Institute of Engineering and Technology, for providing us with valuable guidance and support. We are immensely grateful to her for their cooperation and encouragement in the completion of the project.

We wish to express our sincere regards to Dr. Sunil Dhingra, Director, UIET-KU, for continuous encouragement and supervision throughout the course of the present work.

We would also like to take this opportunity to express our thanks to our parents and friends for their constant support, understanding, and encouragement throughout this entire period.

Finally, we would like to thank our instituUniversitysity Institute of Engineering and Technology, Kurukshetra University, for providing us with the learning opportunity.

ABSTRACT

The aim of this project is to give a general overview of handwritten mathematical expression recognition and its applications. Mathematical expressions are frequently entered by hand from a computer, which is substantially slower than writing them down on paper with a pen. We'll use deep learning technology to identify a handwritten expression on a piece of paper or a drawing software like Microsoft Paint. In this work, we go over the various processes that take to recognize mathematical expressions in handwriting using CNN. The Convolutional Neural Network (CNN) Method offers the greatest accuracy for handwritten mathematical expression recognition. We may be able to enhance overall expression accuracy considerably with more time and computer resources. The future scope of this project involves creation of an improved user interface.

Contents

Chapter 1 INTRODUCTION 7
Chapter 2 LITERATURE REVIEW 8
Chapter 3 METHODOLOGY 9
Chapter 4 DATASET PREPARATION 10
Chapter 5 MODEL BUILDING AND EVALUATION 14
Chapter 6 RESULTS ANDF CONCLUSIONS 21

Chapter 1 INTRODUCTION

4.1. OVERVIEW

Mathematical expressions and equations are often encountered in day-to-day life. Although humans are capable of solving them, they can solve manually only to an extent. Solution of complex expressions and equations holds prime importance in fields ranging from natural sciences to social sciences. This project is all about developing a *machine(deep) learning model* that recognizes the handwritten mathematical expression, analyze it, and returns a solution to the expression.

4.2. OBJECTIVE

Objective of the project is to develop an Assistance System that would take image of a handwritten mathematical expression on a physical paper or a digital snapshot of it and process it to recognize the characters in the expression and provide us the solution.

4.3. SCOPE

For now, the scope of the project is limited to recognition and solution of very simple mathematical expression that spans 16 symbols viz., 0,1,2,3,4,5,6,7,8,9, +, -, *, =, x, y.

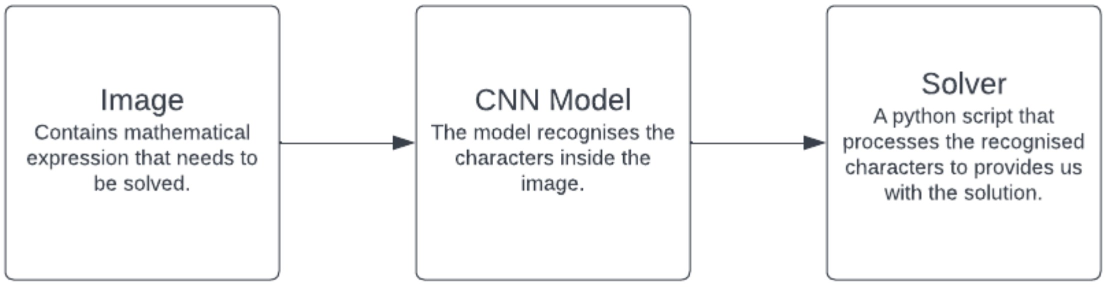


Image
Contains mathematical expression that needs to be solved.
CNN Model
The model recognises the characters inside the image.
Solver
A python script that processes the recognised characters to provides us with the solution.

Chapter 2 LITERATURE REVIEW

Convolutional Neural Network (CNN), a common deep neural network architecture, may be used to accomplish HME Recognition. Traditional CNN classifiers are capable of learning and classifying essential 2D characteristics in pictures, with the classification accomplished using the soft-max layer.

A CNN model includes following layer:

a) **Input Layer**

   The Input Layer serves as a buffer for the input before it is sent on to the following layer.

b) **Convolutional Layer**

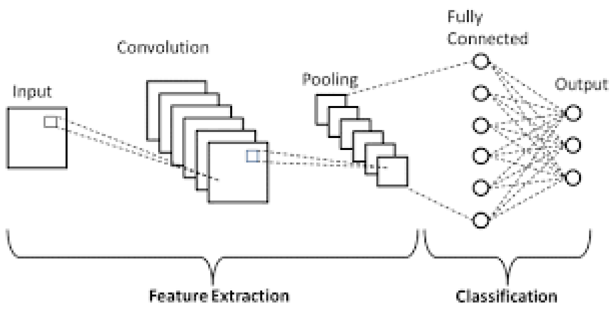
   The main operation of feature extraction is performed by the Convolution Layer. It uses the supplied data to conduct a Convolution process. Sliding the kernel across the input and performing the sum of the product at each position is how the convolution process is carried out. Stride is the size of the step in which the kernel slides. The number of feature maps created in a convolutional layer is also known as the depth of the layer. Several convolutional operations are done on the input using various kernels, resulting in distinct feature maps.

c) **Rectified Linear Unit (ReLU)**

   It is an activation function that is used to inject nonlinearity into a system. Negative values are replaced with zero, which speeds up the learning process. Every convolution layer's output is routed via the activation function.

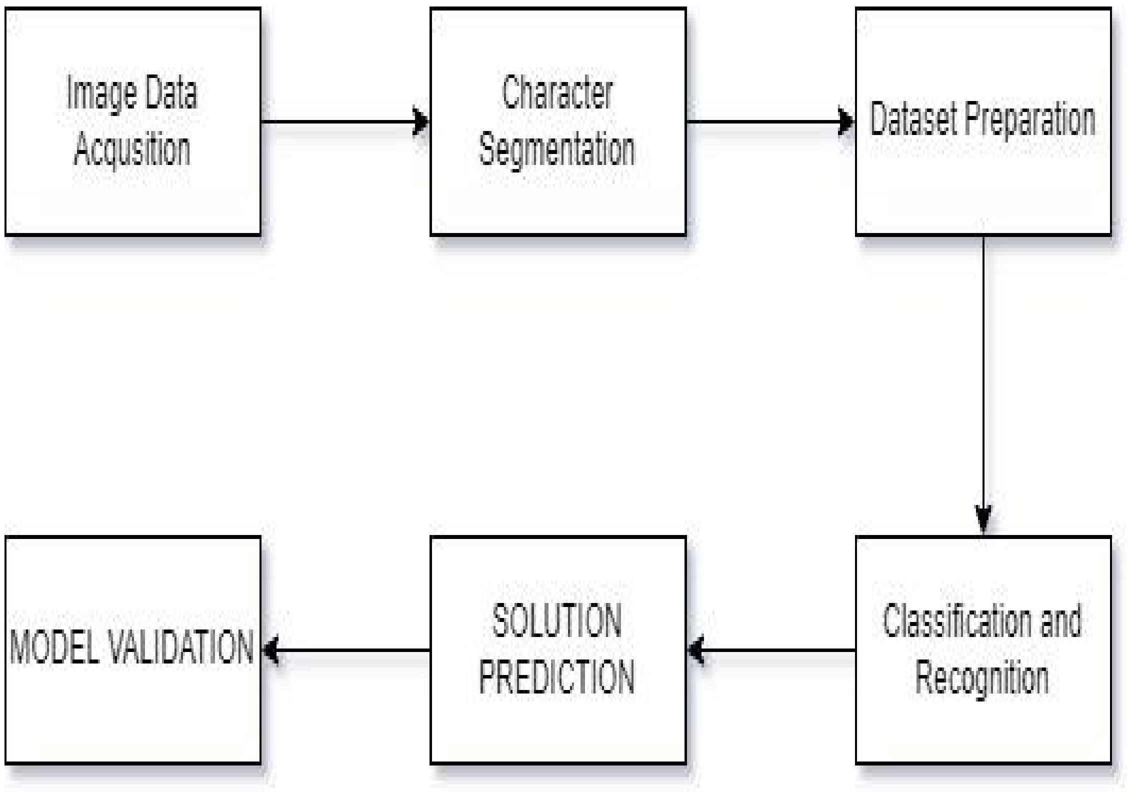
d) **Pooling Layer**

   Pooling employs a sliding window that moves in sync with the feature map, converting it into representative values. It also minimizes the spatial size of each feature map, which reduces the network's computation. The terms "minimum pooling," "average pooling," and "max pooling" are often used.



Chapter 3 METHODOLOGY

The project starts with gathering image data of numbers and symbols, i.e., characters over which model has to be trained. The characters in the images are separated using *Character Segmentation*. The entire image dataset is converted into a comma-separated-value(csv) file dataset. The dataset is trained using CNN network. A program is written using the trained model, that stores the characters recognized from the input image and processes it to provide a solution.



The entire project has been divided into two parts: *Dataset Preparation* and *Model Building and Evaluation.*

The Dataset preparation part includes gathering image data of handwritten numbers and mathematical symbols, segmenting the characters out of them and converting all the characters into NumPy arrays, with merging all the arrays into a Pandas data frame at the end.

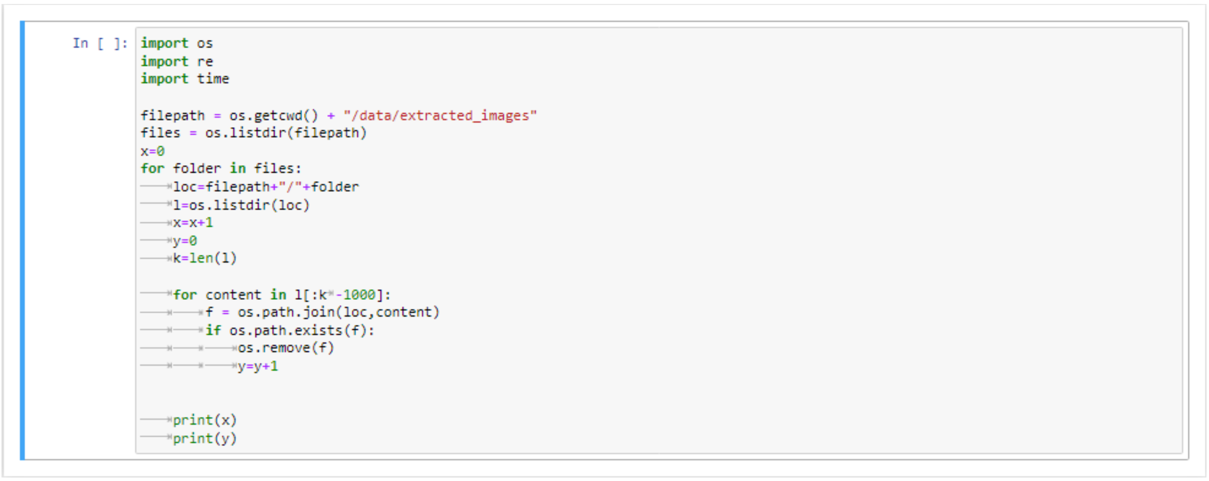
The Model Building and Evaluation part includes developing a CNN model and training it over the data frame so that it could recognize and predict fresh numbers given to it as an input. It also includes developing a script that would process the recognized numbers and symbols to give us the final solution. This part also aims for evaluating/validating the model.

Chapter 4 DATASET PREPARATION

4.1. IMAGE DATA GATHERING

We have used a dataset called the "***Handwritten math symbols dataset***" obtained from *Kaggle*. The dataset contains over 100,000 image samples as ***jpg*** files of **45 x 45** pixels. The dataset includes *English alphanumeric symbols, all mathematical and set operators, predefined math functions* like log, Lim, cos, sin, tan, etc., *mathematical symbols* like \int, \sum, \sqrt, \ delta, etc., and *Greek alphabets* like alpha, beta, gamma, mu, sigma, etc. The dataset is parsed, extracted and modified version of the **CROHME** dataset.

The dataset contains mathematical characters spread over more than 80 folders. The name of each folder is the name of the symbol it contains. All the folders are present in a single folder named, *data*. For our purpose, we used just 16 folders based on the scope of our project. The folders are those containing the symbols: ***0 to 9*** (10)**, +, -,** X, =, x, y. Following script was used to separated desired images out entire dataset into a folder named ***extracted_images***:

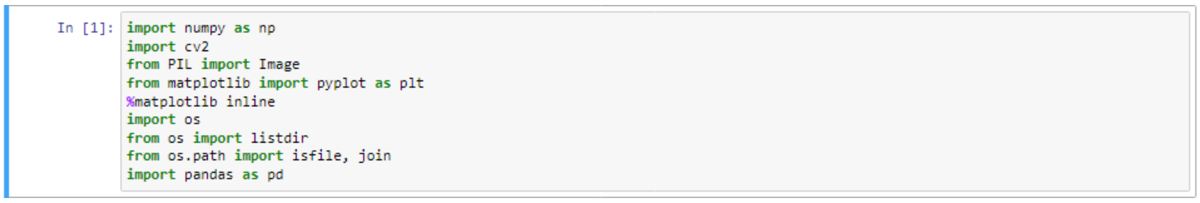


```
In [ ]: import os
        import re
        import time

        filepath = os.getcwd() + "/data/extracted_images"
        files = os.listdir(filepath)
        x=0
        for folder in files:
            loc=filepath+"/"+folder
            l=os.listdir(loc)
            x=x+1
            y=0
            k=len(l)

            for content in l[:k-1000]:
                f = os.path.join(loc,content)
                if os.path.exists(f):
                    os.remove(f)
                    y=y+1


            print(x)
            print(y)
```

4.2. CHARACTER SEGMENTATION

Once, we were done with the selection of image data, our next objective was Character Segmentation, i.e., extraction of individual characters from the image. In other words, *feature extraction,* that includes extraction of only those pixels that compose character images.

For this, we uploaded the *extracted_image* folder on google drive, so that we could work on Google Collaboratory, and use its *TPU* and *GPU* as our dataset is quite large and complex. Then we mounted the folder in a google Collaboratory file and imported the required dependencies. The dependencies included *NumPy* (for processing image arrays), *pandas* (for dataset preparation), *matplotlib* (for data visualization), *cv2*(for processing images), *os* (for working with folders).



```
In [1]: import numpy as np
        import cv2
        from PIL import Image
        from matplotlib import pyplot as plt
        %matplotlib inline
        import os
        from os import listdir
        from os.path import isfile, join
        import pandas as pd
```

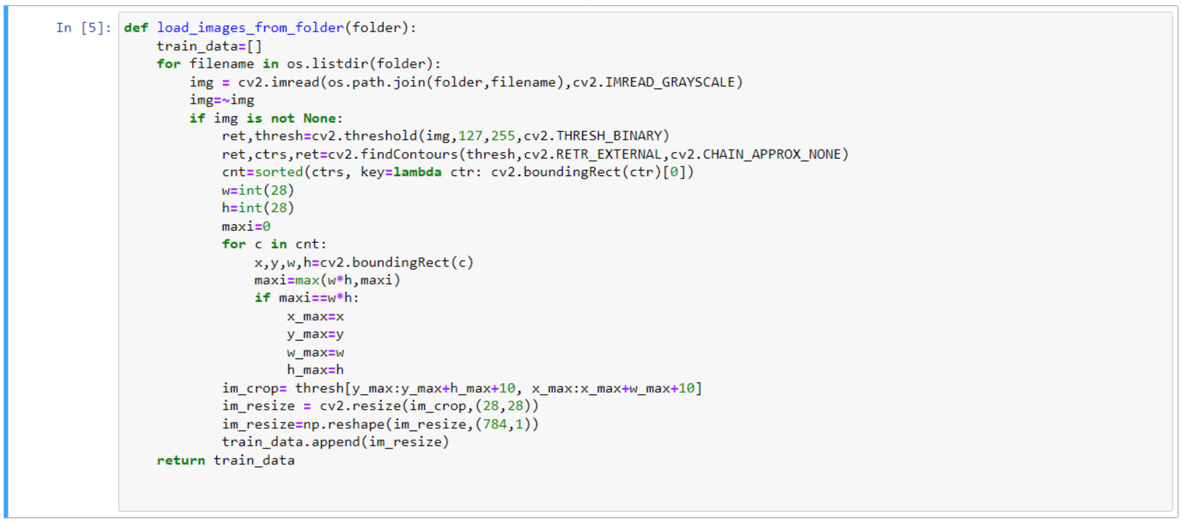
Then we constructed a function named ***load_images*** that would take a folder as an input, iterate through subfolders and images in subfolders, perform the character segmentation tasks, and return a list of NumPy arrays, named ***train_data***, such that each array corresponds to each of the images.

The character segmentation tasks include taking images as *Grayscale* input using **cv2.imread** function and using the technique **cv2.IMREAD_GRAYSCALE**. Then the grayscale images are inverted for better character segmentation. Then, for each image, we use the threshold function of OpenCv2, **cv2.threshold**, so that the characters, as a white foreground, are separated from the black background.

We use *Simple Thresholding*, with **cv2.THRESH_BINARY** technique. Hence, after thresholding, all the character pixels have the same intensity(white). After image thresholding, we need to extract the characters with their exact boundaries for character detection. Precisely, we need to extract *contours of the* characters. For that, we use the **cv2.findContours** function. The contour retrieval mode and the contour approximation methods used are cv2.RETR**_EXTERNAL** and **cv2.CHAIN_APPROX_NONE.**

Once the various contours are obtained for an image, we sort those contours, considering all of them as *integer* values, using **key = lambda.** Then, we iterate over each contour without considering rotation of the character inside the contours (using **bounding_rect**), and based on conditions, we *crop out* the contours containing the character into a **28 x 28-pixel** image. Finally, we reshape the cropped image into a **784 x 1** NumPy array.

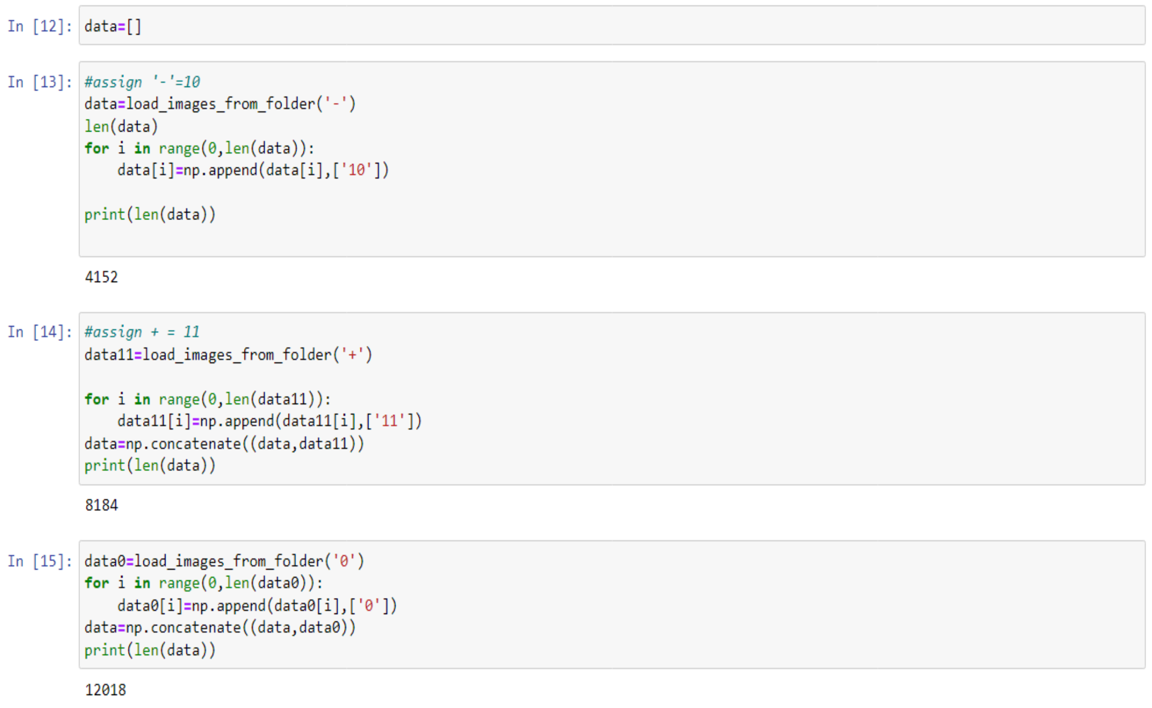
As we had run a loop, for all the images, we wrote a code that would *append* the 784 x 1 NumPy array of all images into a list named **train_data**, initiated at the beginning of the function.



```
In [5]: def load_images_from_folder(folder):
            train_data=[]
            for filename in os.listdir(folder):
                img = cv2.imread(os.path.join(folder,filename),cv2.IMREAD_GRAYSCALE)
                img=~img
                if img is not None:
                    ret,thresh=cv2.threshold(img,127,255,cv2.THRESH_BINARY)
                    ret,ctrs,ret=cv2.findContours(thresh,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_NONE)
                    cnt=sorted(ctrs, key=lambda ctr: cv2.boundingRect(ctr)[0])
                    w=int(28)
                    h=int(28)
                    maxi=0
                    for c in cnt:
                        x,y,w,h=cv2.boundingRect(c)
                        maxi=max(w*h,maxi)
                        if maxi==w*h:
                            x_max=x
                            y_max=y
                            w_max=w
                            h_max=h
                    im_crop= thresh[y_max:y_max+h_max+10, x_max:x_max+w_max+10]
                    im_resize = cv2.resize(im_crop,(28,28))
                    im_resize=np.reshape(im_resize,(784,1))
                    train_data.append(im_resize)
            return train_data
```

4.3. DATASET PREPARATION

We initialize a list named **data.** Using the function, **load_images_from_folder,** we obtain the NumPy array of an image representing minus sign **(-),** and store it the temporary variable *data10*, which mean minus belongs to *class* **10**. The plus sign will belong *class* **11** and multiplication sign(times) will belong to *class* **13.** The numbers will belong to a class name of themselves. Then, to the array of the image, we add another entry, which is nothing but the class name of the number or symbol for that image. We *append* the final array as an element of the list *data*. We do this for all the image folders in 16 iterations for the 136classes(numbers/symbols). For example:



```
In [12]: data=[]
```

```
In [13]: #assign '-'=10
         data=load_images_from_folder('-')
         len(data)
         for i in range(0,len(data)):
             data[i]=np.append(data[i],['10'])

         print(len(data))

         4152
```

```
In [14]: #assign + = 11
         data11=load_images_from_folder('+')

         for i in range(0,len(data11)):
             data11[i]=np.append(data11[i],['11'])
         data=np.concatenate((data,data11))
         print(len(data))

         8184
```

```
In [15]: data0=load_images_from_folder('0')
         for i in range(0,len(data0)):
             data0[i]=np.append(data0[i],['0'])
         data=np.concatenate((data,data0))
         print(len(data))

         12018
```

```
data1=load_images_from_folder('1')

for i in range(0,len(data1)):
    data1[i]=np.append(data1[i],['1'])
data=np.concatenate((data,data1))
print(len(data))
```

```
16074
```

```
data2=load_images_from_folder('2')

for i in range(0,len(data2)):
    data2[i]=np.append(data2[i],['2'])
data=np.concatenate((data,data2))
print(len(data))
```

```
20334
```

```
data3=load_images_from_folder('3')

for i in range(0,len(data3)):
    data3[i]=np.append(data3[i],['3'])
data=np.concatenate((data,data3))
print(len(data))
```

```
23850
```

```
data4=load_images_from_folder('4')

for i in range(0,len(data4)):
    data4[i]=np.append(data4[i],['4'])
data=np.concatenate((data,data4))
print(len(data))
```

```
27882
```

```
data5=load_images_from_folder('5')

for i in range(0,len(data5)):
    data5[i]=np.append(data5[i],['5'])
data=np.concatenate((data,data5))
print(len(data))
```

```
31426
```

```
data6=load_images_from_folder('6')

for i in range(0,len(data6)):
    data6[i]=np.append(data6[i],['6'])
data=np.concatenate((data,data6))
print(len(data))
```

```
34543
```

```
data7=load_images_from_folder('7')

for i in range(0,len(data7)):
    data7[i]=np.append(data7[i],['7'])
data=np.concatenate((data,data7))
print(len(data))
```

```
37451
```

Finally, using **pd.DataFrame** we convert the list of NumPy array of images into a *csv* file name **train_final.csv.**

```
In [27]: df=pd.DataFrame(data,index=None)
         df.to_csv('train_final.csv',index=False)
```

Chapter 5 MODEL BUILDING AND EVALUATION

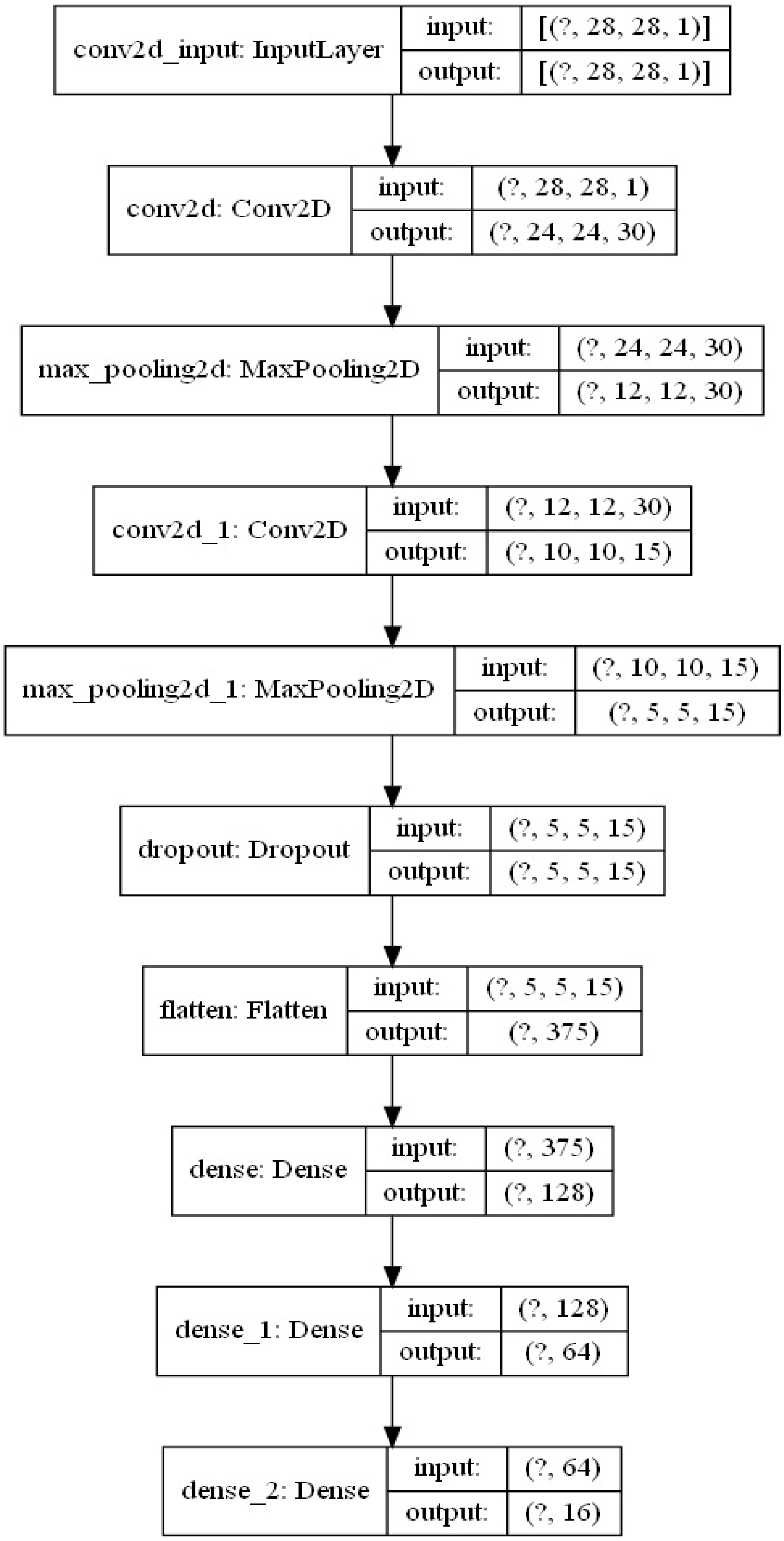
5.1. RECOGNITION AND CLASSIFICATION

For this part, we created a new *ipynb* file. Then we imported essential dependencies and the dataset. After that, we preprocessed the data. At first, we did *Input-Target Split.* Then we performed *Categorical Encoding* on the target column as it was categorical in nature. Then we converted the input data into a 4D NumPy array of shape `(47504, 28, 28, 1)` for training purposes. 47504 is the number of rows representing number of images. Finally, we performed *Train-Test-Split* on the input data.

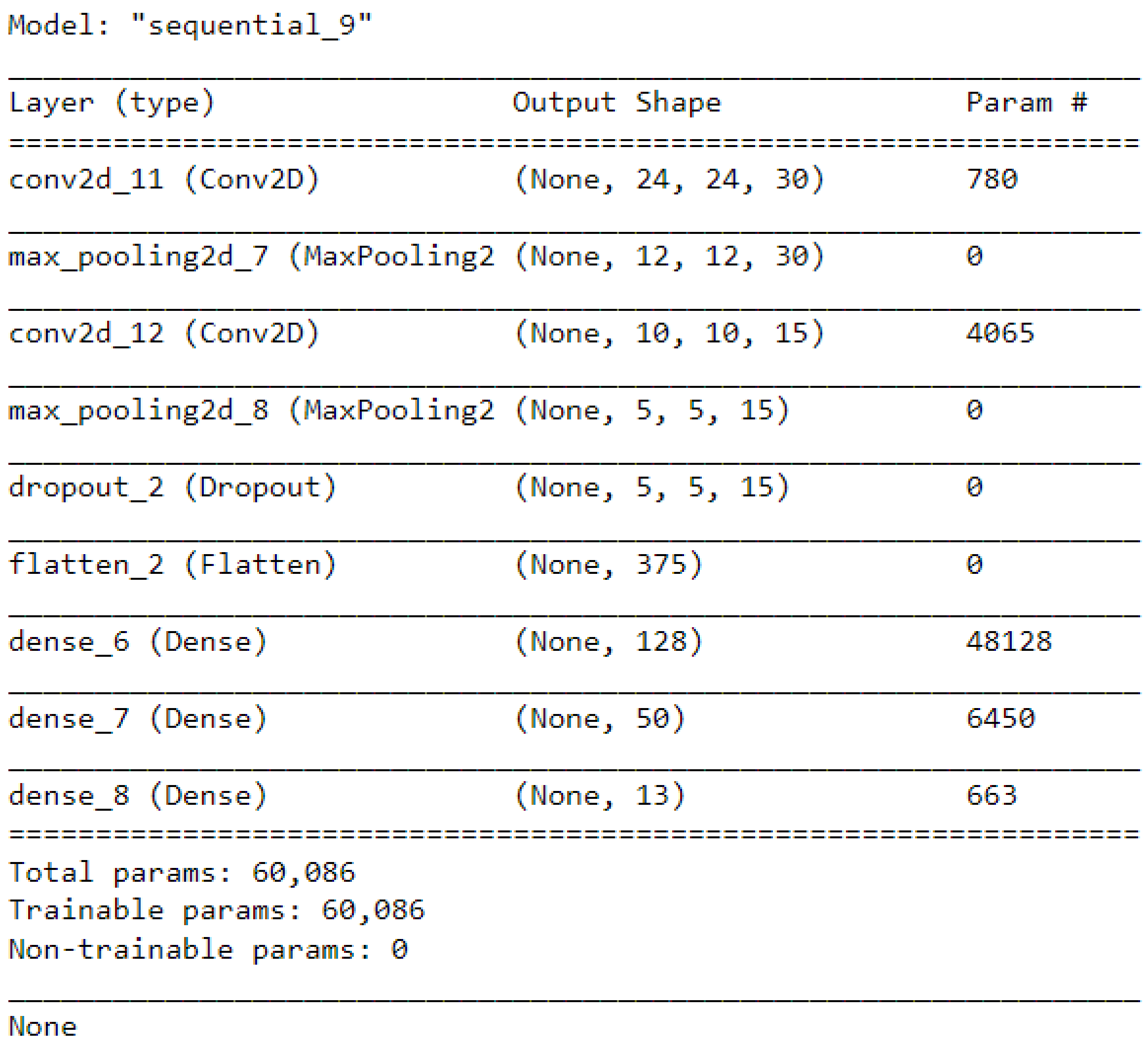
Once data had been split into training and test inputs and outputs respectively, the model is then built. A ***Sequential*** model named **model** is initialized using Keras's ***sequential()*** module of ***Model*** class.

Finally, we compiled our model with following parameters: Loss Function: *Categorical Cross Entropy;* Optimizer = *Adam*; Metrics = *Accuracy*. We got a total of 60,086 trainable parameters.

We added several layers into our model. Following is the plot of Neural Network Model graph :

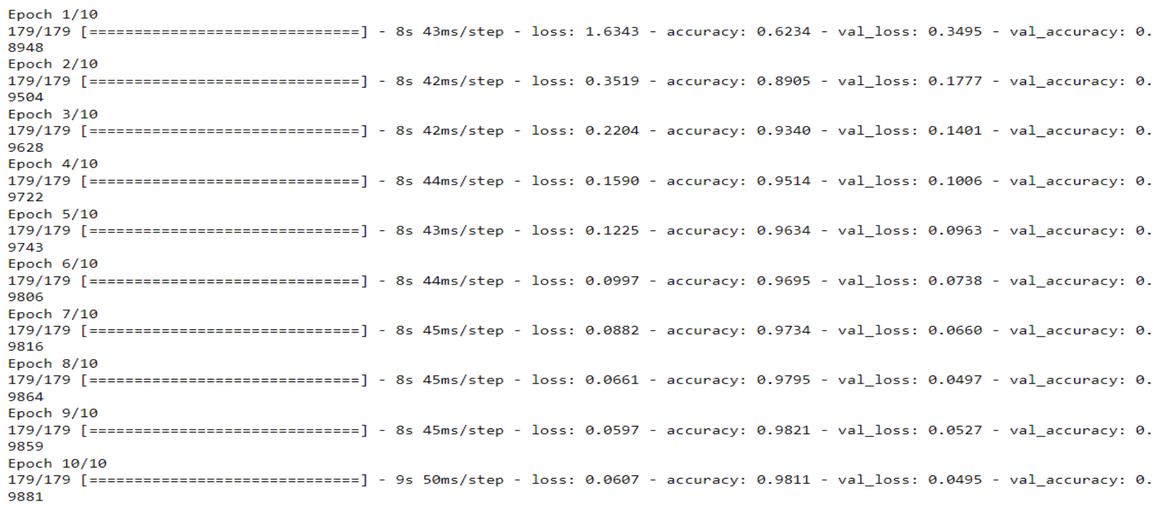


Model Summary



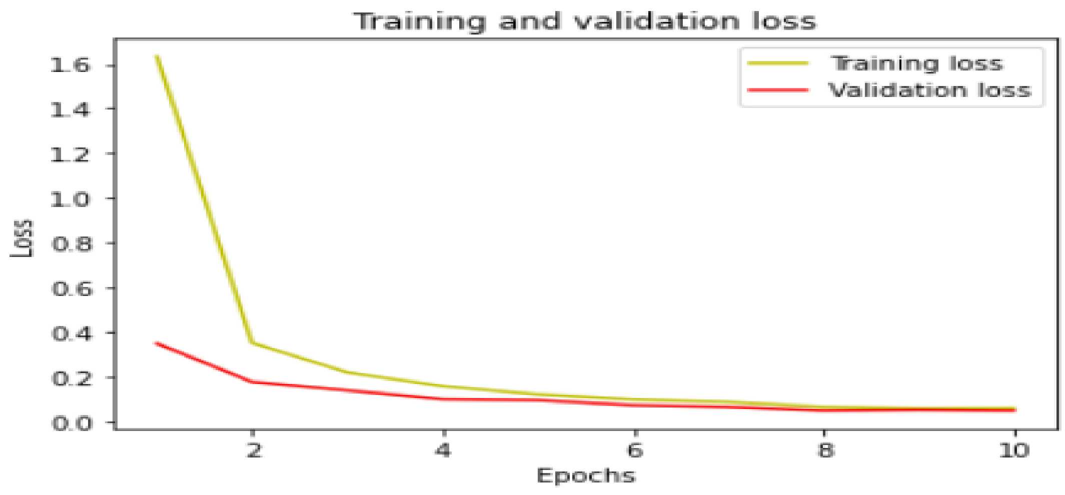
```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_11 (Conv2D)           (None, 24, 24, 30)        780
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 12, 12, 30)        0
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 10, 10, 15)        4065
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 5, 5, 15)          0
_________________________________________________________________
dropout_2 (Dropout)          (None, 5, 5, 15)          0
_________________________________________________________________
flatten_2 (Flatten)          (None, 375)               0
_________________________________________________________________
dense_6 (Dense)              (None, 128)               48128
_________________________________________________________________
dense_7 (Dense)              (None, 50)                6450
_________________________________________________________________
dense_8 (Dense)              (None, 13)                663
=================================================================
Total params: 60,086
Trainable params: 60,086
Non-trainable params: 0
_________________________________________________________________
None
```

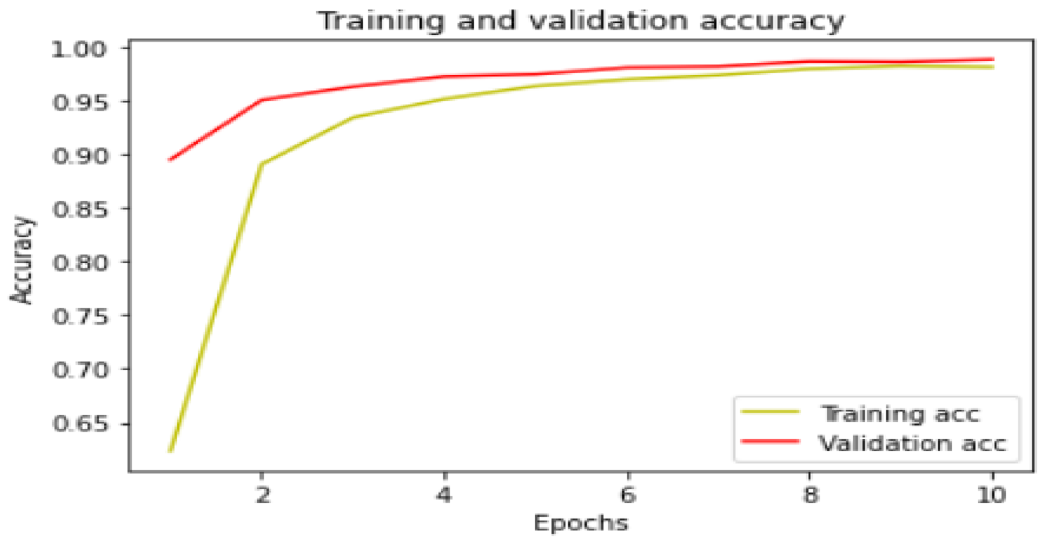
Then, the training data and validation data was fed into the model. The epoch was set to 10 and the batch size for training was set to 200. It took around 9 seconds for the model to get rained with an accuracy of **98.11 %.**



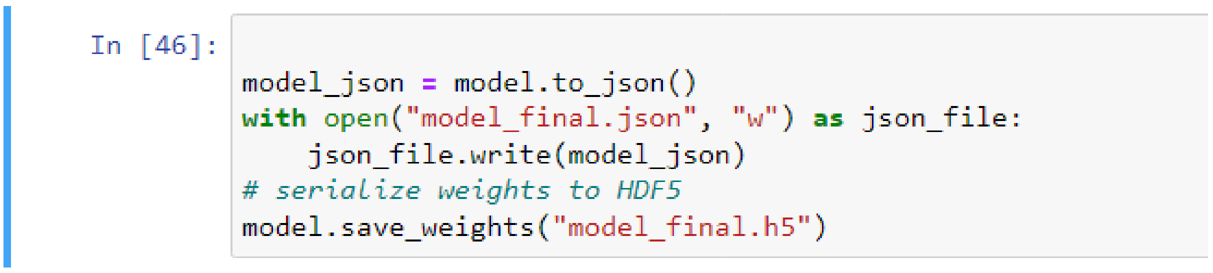
```
Epoch 1/10
179/179 [==============================] - 8s 43ms/step - loss: 1.6343 - accuracy: 0.6234 - val_loss: 0.3495 - val_accuracy: 0.
8948
Epoch 2/10
179/179 [==============================] - 8s 42ms/step - loss: 0.3519 - accuracy: 0.8905 - val_loss: 0.1777 - val_accuracy: 0.
9504
Epoch 3/10
179/179 [==============================] - 8s 42ms/step - loss: 0.2204 - accuracy: 0.9340 - val_loss: 0.1401 - val_accuracy: 0.
9628
Epoch 4/10
179/179 [==============================] - 8s 44ms/step - loss: 0.1590 - accuracy: 0.9514 - val_loss: 0.1006 - val_accuracy: 0.
9722
Epoch 5/10
179/179 [==============================] - 8s 43ms/step - loss: 0.1225 - accuracy: 0.9634 - val_loss: 0.0963 - val_accuracy: 0.
9743
Epoch 6/10
179/179 [==============================] - 8s 44ms/step - loss: 0.0997 - accuracy: 0.9695 - val_loss: 0.0738 - val_accuracy: 0.
9806
Epoch 7/10
179/179 [==============================] - 8s 45ms/step - loss: 0.0882 - accuracy: 0.9734 - val_loss: 0.0660 - val_accuracy: 0.
9816
Epoch 8/10
179/179 [==============================] - 8s 45ms/step - loss: 0.0661 - accuracy: 0.9795 - val_loss: 0.0497 - val_accuracy: 0.
9864
Epoch 9/10
179/179 [==============================] - 8s 45ms/step - loss: 0.0597 - accuracy: 0.9821 - val_loss: 0.0527 - val_accuracy: 0.
9859
Epoch 10/10
179/179 [==============================] - 9s 50ms/step - loss: 0.0607 - accuracy: 0.9811 - val_loss: 0.0495 - val_accuracy: 0.
9881
```

The curves for *loss-vs-epochs* and *accuracy-vs-epochs* were as follows for the training and validation set:





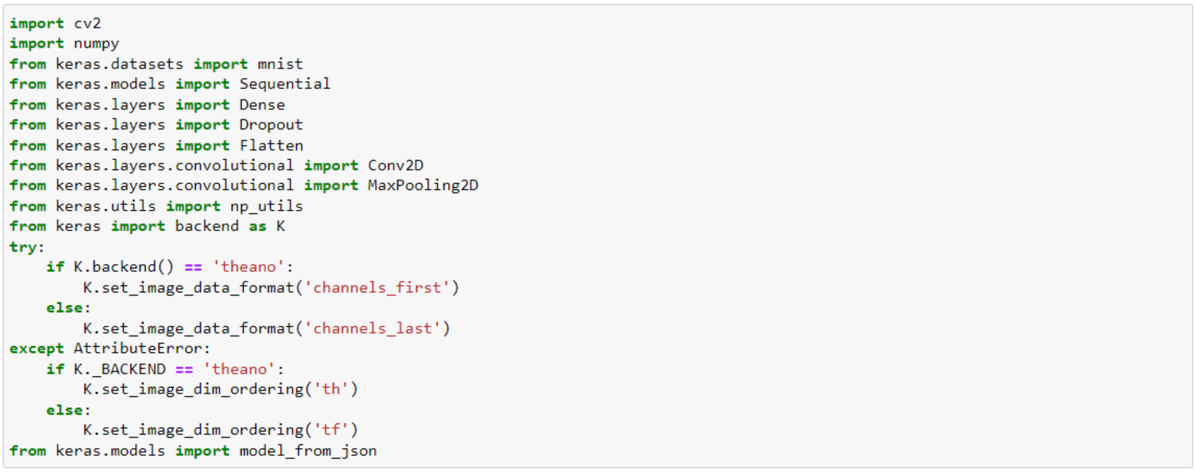
After the model was trained, we dumped the weights of the model into a file named ***model_final.h5.***


In [46]:
```
model_json = model.to_json()
with open("model_final.json", "w") as json_file:
    json_file.write(model_json)
# serialize weights to HDF5
model.save_weights("model_final.h5")
```

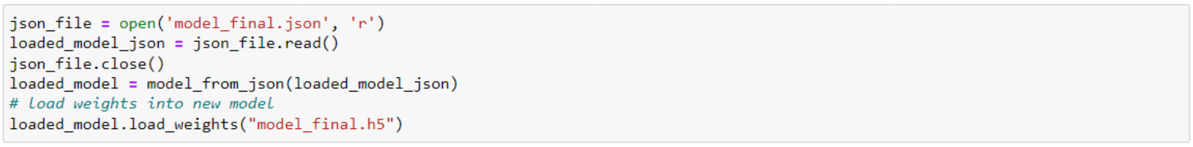
5.2. EXPRESSION SOLVER

Once the model was prepared, we had to write a script that would take an input image containing an expression and process it to recognize characters using the model and then feed those characters into a script that would generate solution to that expression.

Aa separate file named ***CNN_test was and*** required dependencies were initialized. The trained model was also loaded.

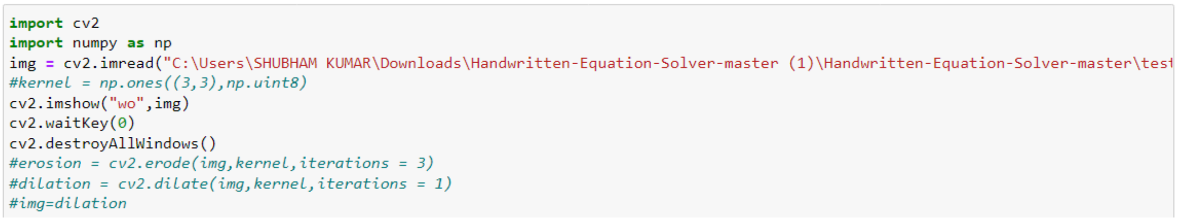


```
import cv2
import numpy
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Flatten
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.utils import np_utils
from keras import backend as K
try:
    if K.backend() == 'theano':
        K.set_image_data_format('channels_first')
    else:
        K.set_image_data_format('channels_last')
except AttributeError:
    if K._BACKEND == 'theano':
        K.set_image_dim_ordering('th')
    else:
        K.set_image_dim_ordering('tf')
from keras.models import model_from_json
```



```
json_file = open('model_final.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json)
# load weights into new model
loaded_model.load_weights("model_final.h5")
```

Then a program was written that takes the input image and processes it. The processing includes Character Segmentation, Feature Extraction, Contour Detection, and conversion of the extracted image into NumPy array into the dimension **1x28x28,** which was the dimension of the data over which the model has been trained.



```
import cv2
import numpy as np
img = cv2.imread("C:\Users\SHUBHAM KUMAR\Downloads\Handwritten-Equation-Solver-master (1)\Handwritten-Equation-Solver-master\test
#kernel = np.ones((3,3),np.uint8)
cv2.imshow("wo",img)
cv2.waitKey(0)
cv2.destroyAllWindows()
#erosion = cv2.erode(img,kernel,iterations = 3)
#dilation = cv2.dilate(img,kernel,iterations = 1)
#img=dilation
```

```python
if img is not None:
    #images.append(img)
    img=~img
    _,thresh=cv2.threshold(img,127,255,cv2.THRESH_BINARY)
    ctrs,_=cv2.findContours(thresh,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
    cnt=sorted(ctrs, key=lambda ctr: cv2.boundingRect(ctr)[0])
    w=int(28)
    h=int(28)
    train_data=[]
    #print(len(cnt))
    rects=[]
    for c in cnt :
        x,y,w,h= cv2.boundingRect(c)
        rect=[x,y,w,h]
        rects.append(rect)
    #print(rects)
    bool_rect=[]
    for r in rects:
        l=[]
        for rec in rects:
            flag=0
            if rec!=r:
                if r[0]<(rec[0]+rec[2]+10) and rec[0]<(r[0]+r[2]+10) and r[1]<(rec[1]+rec[3]+10) and rec[1]<(r[1]+r[3]+10):
                    flag=1
                l.append(flag)
            if rec==r:
                l.append(0)
        bool_rect.append(l)
    #print(bool_rect)
    dump_rect=[]
    for i in range(0,len(cnt)):
        for j in range(0,len(cnt)):
            if bool_rect[i][j]==1:
                area1=rects[i][2]*rects[i][3]
                area2=rects[j][2]*rects[j][3]
                if(area1==min(area1,area2)):
                    dump_rect.append(rects[i])
```

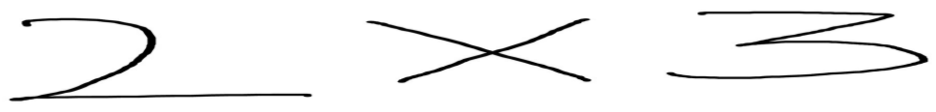
```python
    #print(len(dump_rect))
    final_rect=[i for i in rects if i not in dump_rect]
    #print(final_rect)
    for r in final_rect:
        x=r[0]
        y=r[1]
        w=r[2]
        h=r[3]
        im_crop =thresh[y:y+h+10,x:x+w+10]


        im_resize = cv2.resize(im_crop,(28,28))
        cv2.imshow("work",im_resize)
        cv2.waitKey(0)
        cv2.destroyAllWindows()

        im_resize=np.reshape(im_resize,(1,28,28))
        train_data.append(im_resize)
```

At last, a script was written that processes the recognised characters to solve the equation/expression.

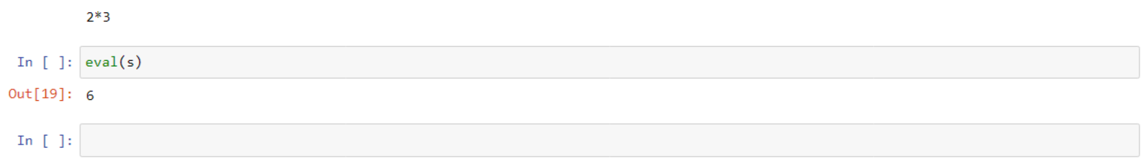
```python
s=''
for i in range(len(train_data)):
    train_data[i]=np.array(train_data[i])
    train_data[i]=train_data[i].reshape(1,28,28,1)
    result=loaded_model.predict_classes(train_data[i])
    if(result[0]==10):
        s=s+'-'
    if(result[0]==11):
        s=s+'+'
    if(result[0]==12):
        s=s+'*'
    if(result[0]==0):
        s=s+'0'
    if(result[0]==1):
        s=s+'1'
    if(result[0]==2):
        s=s+'2'
    if(result[0]==3):
        s=s+'3'
    if(result[0]==4):
        s=s+'4'
    if(result[0]==5):
        s=s+'5'
    if(result[0]==6):
        s=s+'6'
    if(result[0]==7):
        s=s+'7'
    if(result[0]==8):
        s=s+'8'
    if(result[0]==9):
        s=s+'9'

print(s)
```

5.3. MODEL VALIDATION

For model validation, a simple mathematical expression is written on Microsoft Paint and its image is exported into *jpeg* file.



Then the image is fed into the Solver program.



```
         2*3

In [ ]:  eval(s)
Out[19]: 6

In [ ]:
```

As one can see, the model successful recognizes the characters in the image, viz., **2 * 3** and stores it in a variable ***s.*** When the variable is passed into the ***eval()*** *function* from the solver script, it returns the correct solution as **6.**

Chapter 6 RESULTS AND CONCLUSIONS

Handwriting has been one of the most fundamental methods for communication for ages, since it is an important component of the learning process. This project addresses useful methods of character recognition.

The major goal was handwritten mathematical expression recognition, as printed text recognition did not require a large number of datasets to train the algorithm. Of various methods for recognition of handwritten mathematical expressions and the one that has the highest accuracy is the Method of Convolutional Neural Network (CNN), hence it was chosen.

CNNs are a strong way to tackle handwritten expression recognition. With more time and greater computational resources, it may be able to dramatically improve overall expression accuracy. We learned about the various processes done by CNN to identify handwritten mathematical expression. We have studied how handwritten mathematical model improves accuracy and what improvements can be done. We also explored a variety of datasets for training and testing of CNN models.

In the next phase we could work on finding the solutions of advance mathematical expressions by combining mathematical expression recognition capabilities with existing algebra solving software, graphing tools, and simulation systems would be a first step toward developing a superior user interface for performing the task. In the future by implementing the technologies, the percentage accuracy of the model can be further be increased.