Handwritten Digit Recognition

BY:

Name: Tanzeel Khan
Course: B.Tech
Branch: AI&DS
Roll Number: 2017677

Handwritten Digit Recognition

By: Tanzeel Khan

Problem Statement:

MNIST Handwritten Digit Recognition: Here, I have to make a model that takes in an image of a handwritten digit as input, performs some processing operations on it and then classifies the number written in the image. The dataset to be used for this project is called MNIST Dataset. It has 70,000 rows in total with 60,000 rows for training and 10,000 rows for testing our model.

Motivation:

Ever since I started learning Data Science, I wanted to step into deep learning with Computer Vision. The idea of making machines work like human brains makes me curious about the concepts behind it. I wanted to learn the underlying methods and algorithms that can be used to make a deep learning model. Since the burden of academic studies and the need to perform well in my college, took a toll on my extra learning hours, I decided to opt for this project thereby indirectly making it a part of my syllabus and hence it led me to know about Convolutional Neural Networks. I would also like to add that this project would have been harder to make if it wasn't for the immense help and support of my mentor Mr. Vikas Tomer, who was always ready to clear any doubts I had. He was always just a text or a call away. I would like to thank him from the bottom of my heart.

Introduction:

This project is based on Computer Vision, Deep Learning and Optical Character Recognition (OCR). OCR is a part of Computer Vision that deals with recognizing text written on images. It has wide scale applications like recognizing images of bills or receipts and extracting information from it, recognizing a vehicle from the license plate if that vehicle breaks any traffic rules etcetera. This project is an introductory project in the world of Deep Learning and hence it has its limitations. Here, we will we classifying only the digits 0 to 9 written on images. To do this, I will be using a dataset called MNIST Handwritten Digits dataset. It has thousand of images that can be used to train and test our model. Each image is of 28x28 pixels. To create and deploy this model, I have used the following libraries:

1. Tensorflow: It is a deep learning library developed by Google.
2. Keras: It is also a deep-learning library that provides us with numerous functions and methods for easily creating a model.

3. Flask: It is a framework provided by Python that helps in integrating front-end and back-end of any web-based application.

About the dataset:

MNIST is an acronym for Modified National Institute of Standards and Technology. It is a dataset that helps beginners in deep learning learn, understand, and implement the basics of deep learning. It consists of 60,000 rows for training the model that we created and an additional 10,000 rows for testing the accuracy of our model. Each image here is 28x28 pixels and consists of digits from 0 to 9. It is a clean dataset with no missing values and outliers.

Theoretical Study:

Why do we need Optical Character Recognition?

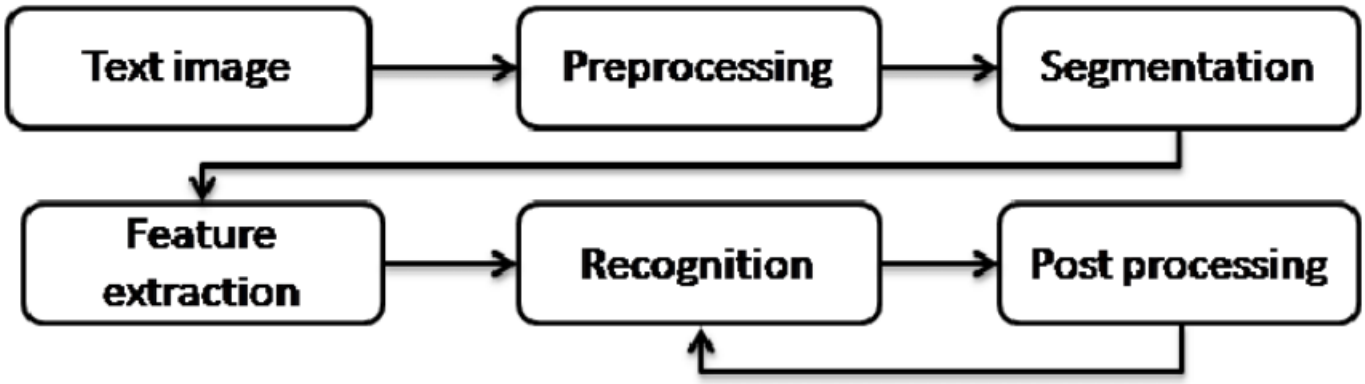
Let's take up an instance: Suppose you just witnessed a horrible hit and run incident. Everyone around you was too shaken up by the accident that no one was able to note down the license plate number of the car that did this. In this situation, the culprit is most likely to run away as even the cameras that recorded the incident will be a bit slow in catching up with the car and the culprit. Here, OCR can play an excellent role to help the situation. We can design a software that detects license plate numbers of vehicles from a live feed and instantly inform the nearest police station if and when such an incident occurs by using video anomaly detection algorithms. Not only this, but we have more such applications like extracting useful information from images with text, helping us to organize our photo galleries in a better way, searching through images based on some text in those images etcetera.

What are the pre-requisites for OCR?

To completely understand OCR, we need to have a prior knowledge of a programming language that can implement machine learning, for example: Python, R, Julia, Java etcetera. We also need to have a good grasp over machine learning and image preprocessing as it is a crucial part of OCR.

Methodology:

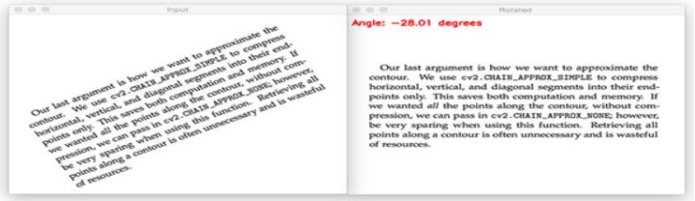
Below is a diagram that describes how an OCR model works, step by step:



Let's briefly discuss each of these steps one by one:

1. *Image acquisition/Text image:* This step involves scanning a picture or a document that contains some text to be recognized and then storing it.
2. *Preprocessing:* Our model will not perform well if we feed the scanned image directly into the model. We need to perform some preprocessing on the scanned image for better results. These preprocessing steps may include:

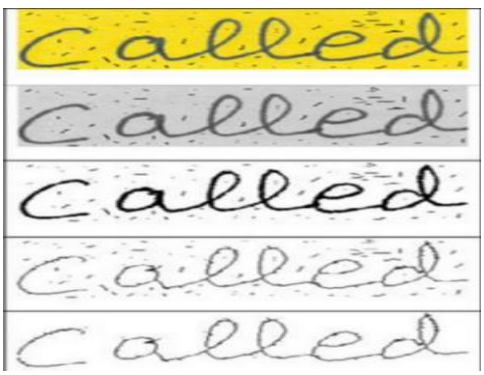
   a. Skew Correction: Sometimes, the image that we scanned might be askew, that means it is not in horizontal orientation. This step is performed to rectify this mistake.



   b. Binarization: This process involves converting the colored scanned image into a binary image with only black and white colors. It is usually done by converting the colored image into an intermediate gray-scale image. It can be implemented using different methods like Adaptive Thresholding, Otsu's Binarization or Local Maxima and Minima method.

   c. Noise Removal: While capturing the image of the document, our picture might be introduced to some small dots or foreground components because of reasons like low quality camera, blurred photo or some shadow on the image. This noise needs to be removed to make our image clean and uniform.

   d. Thinning and Skeletonization: It is not necessary that we will have images that have pen or pencil strokes of the same width. The difference in widths introduces a large variability in the dataset which will require more data to make a good model. Hence, to reduce this, we perform this process where



we make al strokes of equal width, maybe 1 pixel wide.

3. *Segmentation:* Once we acquire a clean and skeletonized image, we move to segmentation. This process involves breaking the image in smaller parts or segments for further processing. There are three types of segmentation in OCR: Line Level Segmentation, Word Level Segmentation and Character Level Segmentation.

4. *Feature Extraction:* In this step, we extract unique characteristics or features from the segmented subcomponents of the image. Nowadays, we use machine learning models like CNN, RNN, LSTM etcetera for feature extraction.

5. *Classification:* This step is the decision-making step of the whole process. It uses the features obtained in the feature extraction stage to correctly identify any text segment. This is done using algorithms like Support Vector Machines.

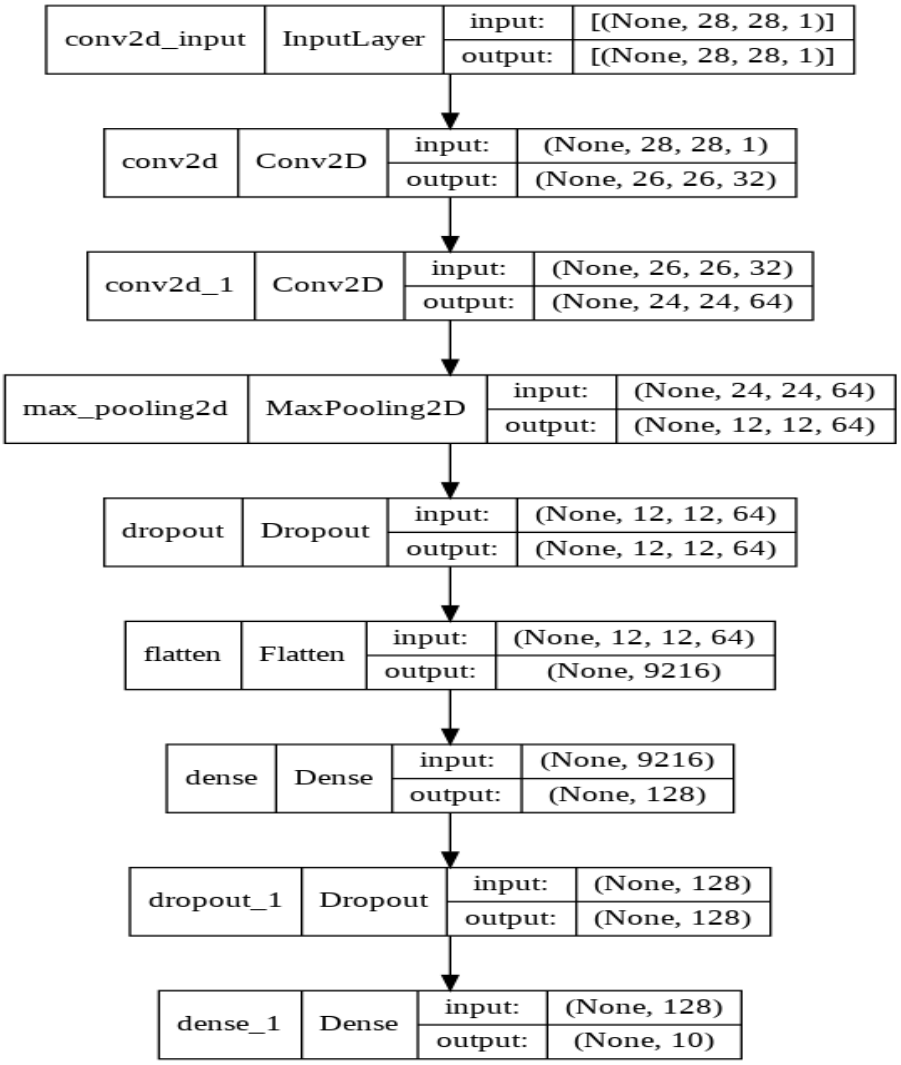
6. *Post-processing:* In OCR model, errors most probably occur due to the wrong prediction during classification which maybe a result of feature extraction, blurry image etcetera. These errors might result in minor spelling mistakes like the word “tall” can be predicted as “toll”. To remove such errors, we can take the help of Natural Language Processing, which itself is a completely different branch of machine learning.

Tools Used:

Following are the tools that I used to complete this project:

1. Google Colab: It is a cloud-based Integrated Development Environment that I used to create my deep-learning model.
2. Visual Studio Code: In this IDE, I wrote the code that will integrate the frontend of my web application to the inner working of my model.
3. GitHub: This is a version control system that I used to maintain my code.

Following is an image of the model that I made:


conv2d_input
InputLayer
input: [(None, 28, 28, 1)]
output: [(None, 28, 28, 1)]
conv2d
Conv2D
input: (None, 28, 28, 1)
output: (None, 26, 26, 32)
conv2d_1
Conv2D
input: (None, 26, 26, 32)
output: (None, 24, 24, 64)
max_pooling2d
MaxPooling2D
input: (None, 24, 24, 64)
output: (None, 12, 12, 64)
dropout
Dropout
input: (None, 12, 12, 64)
output: (None, 12, 12, 64)
flatten
Flatten
input: (None, 12, 12, 64)
output: (None, 9216)
dense
Dense
input: (None, 9216)
output: (None, 128)
dropout_1
Dropout
input: (None, 128)
output: (None, 128)
dense_1
Dense
input: (None, 128)
output: (None, 10)