Department of Computer Engineering and Science,
Florida Institute of Technology, Melbourne, Florida

English Handwriting Optical Character Recognition with Deep Learning

By

Sheetal Ghodake

Supervisor

Dr. Debasis Mitra

Spring 2021

ACKNOWLEDGMENT

We take this opportunity to express our Gratitude and Deep Regards to Prof. Debasis Mitra for his exemplary guidance, monitoring and constant encouragement throughout the course of this project.

We are obliged to the Dr. Debasis Mitra, for the valuable information provided by him in their respective field. We are grateful for the cooperation during the period of our assignment.

ABSTRACT

Optical character recognition (OCR) technology has been revolutionized, allowing it to become one of the inventions that has a wide range of applications across industries.

It is a method of digitizing printed texts so that they can be electronically searched and used in machine processes. It converts the images into machine-encoded text that can be used in machine translation, text-to-speech and text mining.

While OCR is now available in a variety of languages and has the potential to identify characters in real time, there are still several languages for which this technology has not progressed significantly. Many of these advances have been made possible by the application of artificial intelligence and deep learning principles. When it comes to tasks requiring recognition, Deep Neural Networks have proved to be the best option. This can be accomplished using a variety of algorithms and models. This project aims to develop and improve simple, efficient, and less costly a deep learning-based model that can identify English handwritten characters in real time.

TABLE OF CONTENTS

INTRODUCTION

The most common way to communicate with a machine is by character recognition. For scholars and scientists, it has piqued their curiosity. Character recognition is the mechanism by which a computer identifies and understands characters in a text picture and transforms the interpreted data into a code that the machine can understand. In the field of pattern recognition, it is a basic but difficult activity.

Since it deals with characters that are optically encoded rather than magnetically processed, it's called optical character recognition (OCR). It can be used to solve a wide range of computer vision issues, including identifying car number plate characters and recognizing the address.

In today's world, there is an increasing need for applications that can identify characters in a computer system as material is scanned from a paper text. As we all know, there are many newspapers and books available in written format that cover a wide range of topics. There is a massive need these days for "storing the information available in these paper records on a data retrieval disk and then reusing this information into a search process."

Scanning the papers and then storing them as IMAGES is an easy way to archive material from these paper documents in a database device. However, reading individual contents and searching the contents of these documents line-by-line and word-by-word makes it impossible to reuse this material.

We occasionally need to handle data that is related to languages other than English around the world. A software system called CHARACTER RECOGNITION SYSTEM is needed for this.

PROJECT SCOPE

The aim of our Optical Character Recognition on a Grid Infrastructure product is to provide an effective and enhanced software tool for users to perform Document Image Analysis, document processing, and character recognition in scientific, academic, governmental, and business organizations with large pools of registered, scanned images. Regardless of the size or form of characters in documents, the product recognizes, searches, and processes them faster in accordance with the needs of the environment.

PROJECT PREREQUISITES

Technologies

Python (3.7.4)

The key language we used to write the implementation of this project was Python. The program's components include data preprocessing, a classification model, and a neural network. Python is a very versatile programming language that supports a large number of libraries and extensions. For example, NumPy, which supports high-level math functions on matrices and multidimensional arrays, is one of the libraries used in this project. This library aids in the construction of a simple neural network by assisting with sigmoid function multiplication and large dimensional arrays. Keras, OpenCV, TensorFlow, Matplotlib, and other libraries were used in this project.

TensorFlow (2.0.0)

It is an opensource architecture that consists of tools built into a scalable ecosystem, libraries, and other resources that help developers build and deploy machine learning-based applications quickly and easily. It provides several levels of abstraction and allows the development and training of models using the Keras high-level API. It enables Deep Learning Based Real Time Devanagari Character Recognition 26 users to easily train and deploy models regardless of language or platform. The language we used to implement our model for this project was Python.

Pycharm (IDE)

PyCharm is a JetBrains hybrid framework that serves as a Python IDE. It's a popular tool for creating Python applications. It also includes modules and packages that aid programmers in developing Python software in less time and with less effort. It can also be tailored to meet the needs of developers.

Libraries

Keras (2.3.1)

It's a high-level neural network API written in Python. It is written in Python and is based on TensorFlow. It allows for fast experimentation. It's a user-friendly, modular library that lets you understand models as graphs or sequences. It aids in reducing the number of actions taken by the consumer for common use cases. Keras' error feedback system is an additional bonus. This is the reason Keras is so easy to use.

NumPy (1.16.5)

NumPy is a Python module that supports matrices and multidimensional arrays for high-level math functions. This library aids in the construction of a simple neural network by assisting with sigmoid function multiplication and large dimensional arrays. It can be used as an n-dimensional container for generic data and allows for the definition of arbitrary datatypes. This enables NumPy to easily integrate and combine with a variety of databases.

OpenCV (3.4.2)

Python is slower than its equivalents C and C++, but it has the ability to be expanded with C and C++, which means that these faster languages can be used to write computationally intensive code, and then a python wrapper class that can be used as a python module is developed. This provides benefits such as fast-running code that can be used with the simple Python programming language. It's used to solve computer vision issues. The NumPy library is often used to combine OpenCV since the operations available in NumPy can be combined with it. It is one of the most efficient libraries for performing mathematical operations. OpenCV is a program that can be used to solve computer vision problems. For this reason,

Pandas (0.25.1)

Pandas is a Python open-source library that allows for high-performance data manipulation. It is a Data Representation package built on top of the NumPy package.

Matplotlib (3.1.1)

Matplotlib is a Python library that allows you to create static, animated, and interactive visualizations. It's a plotting library for Python and NumPy, Python's numerical math extension.

Dataset

In this project, I have used **Kaggle** dataset, and it contains 372450 images of alphabets.

- Data type: Greyscale images (.csv format)
- Dataset size: 372450 sample images
- Training data: 297960
- Test data: 74490
- Train labels: 297960
- Test labels: 74490
- Size of image: 32x32 pixels
- Actual Size: 28x28 pixels

ARCHITECTURE

A. Pre-Processing

The image is taken and is converted to gray scale image. The gray scale image is then converted to binary image. This process is called Digitization of image (Binarization).Practically any scanner is not perfect; the scanned image may have some noise. This noise may be due to some unnecessary details present in the image. By applying suitable methods the denoised image is produced. The denoised image thus obtained is saved for further
processing.

B. Character Extraction

The pre-processed image serves as the input to this and each single character in the image is found out.

C. Recognition

The image from the extraction stage is correlated with all the templates which are preloaded into the system. Once the correlation is completed, the template with the maximum correlated value is declared as the character present in the image.

D. Post Processing

After the recognition stage, if there are some unrecognized characters found, those characters are given their meaning in the post-processing stage. Extra templates can be added to the system for providing a wide range of compatibility checking in the systems database.

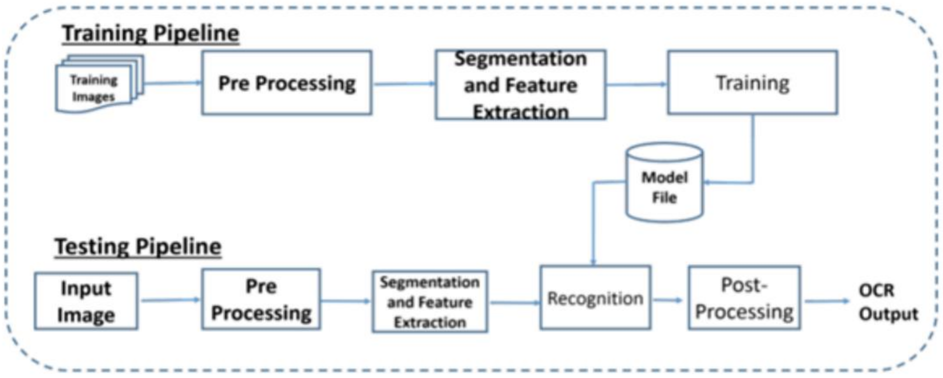
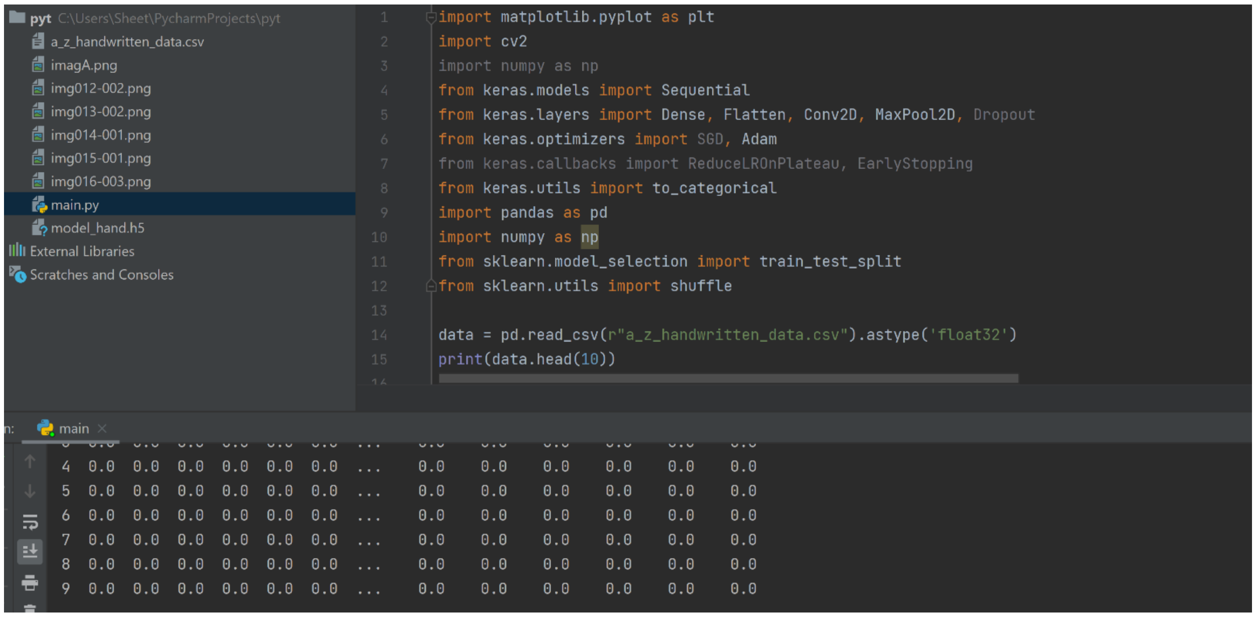


Figure 1: Training and Testing pipeline of a CNN Model

TECHNICAL IMPLEMENTATION

1. First of all, we import all the necessary libraries and we will see the use of all the imports as we use them.
2. Read the data from the CSV file using **pd.read_csv()**.

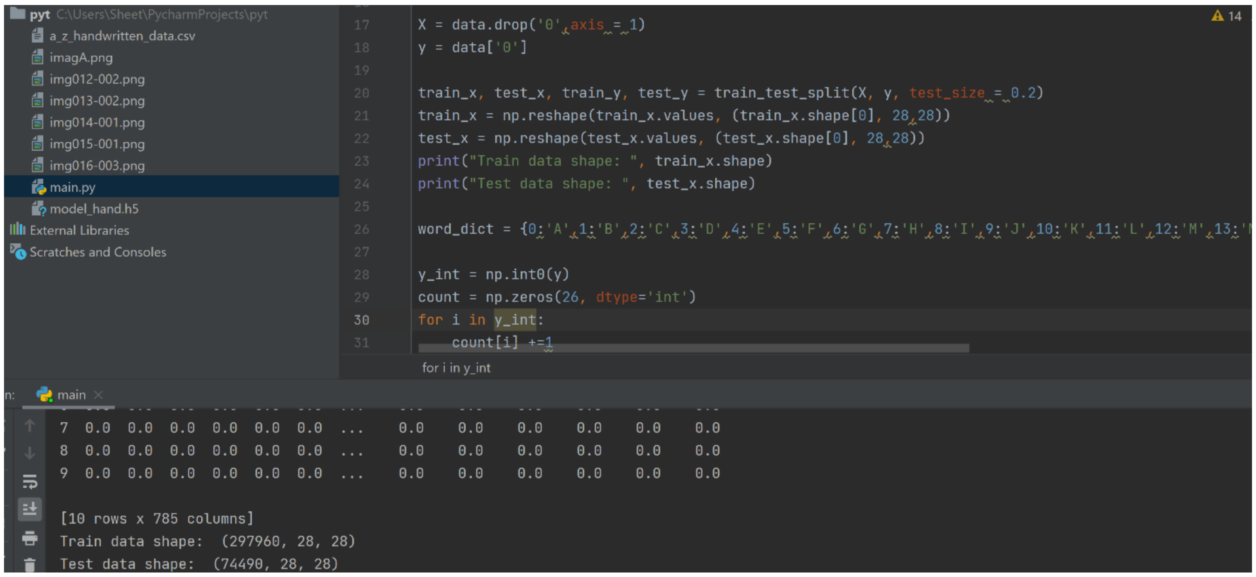


```
import matplotlib.pyplot as plt
import cv2
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Flatten, Conv2D, MaxPool2D, Dropout
from keras.optimizers import SGD, Adam
from keras.callbacks import ReduceLROnPlateau, EarlyStopping
from keras.utils import to_categorical
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle

data = pd.read_csv(r"a_z_handwritten_data.csv").astype('float32')
print(data.head(10))
```

```
4  0.0  0.0  0.0  0.0  0.0  0.0  ...   0.0    0.0    0.0    0.0    0.0    0.0
5  0.0  0.0  0.0  0.0  0.0  0.0  ...   0.0    0.0    0.0    0.0    0.0    0.0
6  0.0  0.0  0.0  0.0  0.0  0.0  ...   0.0    0.0    0.0    0.0    0.0    0.0
7  0.0  0.0  0.0  0.0  0.0  0.0  ...   0.0    0.0    0.0    0.0    0.0    0.0
8  0.0  0.0  0.0  0.0  0.0  0.0  ...   0.0    0.0    0.0    0.0    0.0    0.0
9  0.0  0.0  0.0  0.0  0.0  0.0  ...   0.0    0.0    0.0    0.0    0.0    0.0
```

Figure 2: Reading the data from CSV file

3. Splitting the data read into the images and their corresponding labels. The ‘o’ contains the labels and we drop the ‘o’ column from the data dataframe and use it in the y to form the labels.
4. Splitting the data into Training & Testing dataset using **train_test_split()** method and Reshaping the Train & Test image data so that they can be displayed as an image.
5. All labels are present in the form of floating point value, that we convert to integer value and we create a dictionary **word_dict** to map the integer value with the character (Figure 3).

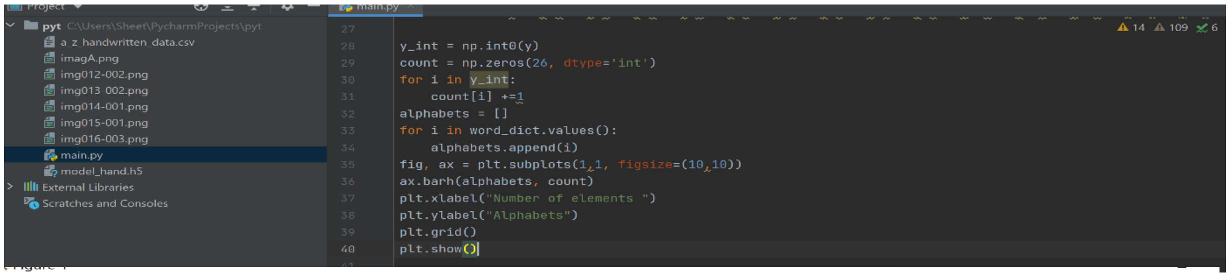


```
X = data.drop('0',axis = 1)
y = data['0']

train_x, test_x, train_y, test_y = train_test_split(X, y, test_size = 0.2)
train_x = np.reshape(train_x.values, (train_x.shape[0], 28,28))
test_x = np.reshape(test_x.values, (test_x.shape[0], 28,28))
print("Train data shape: ", train_x.shape)
print("Test data shape: ", test_x.shape)

word_dict = {0:'A',1:'B',2:'C',3:'D',4:'E',5:'F',6:'G',7:'H',8:'I',9:'J',10:'K',11:'L',12:'M',13:'

y_int = np.int0(y)
count = np.zeros(26, dtype='int')
for i in y_int:
    count[i] +=1
```

```
7  0.0  0.0  0.0  0.0  0.0  0.0  ...   0.0   0.0   0.0   0.0   0.0   0.0
8  0.0  0.0  0.0  0.0  0.0  0.0  ...   0.0   0.0   0.0   0.0   0.0   0.0
9  0.0  0.0  0.0  0.0  0.0  0.0  ...   0.0   0.0   0.0   0.0   0.0   0.0

[10 rows x 785 columns]
Train data shape:  (297960, 28, 28)
Test data shape:  (74490, 28, 28)
```

Figure 3: Labels for the characters

6. Plotting the number of alphabets in the dataset (Figure 4).



```
y_int = np.int0(y)
count = np.zeros(26, dtype='int')
for i in y_int:
    count[i] +=1
alphabets = []
for i in word_dict.values():
    alphabets.append(i)
fig, ax = plt.subplots(1,1, figsize=(10,10))
ax.barh(alphabets, count)
plt.xlabel("Number of elements ")
plt.ylabel("Alphabets")
plt.grid()
plt.show()
```

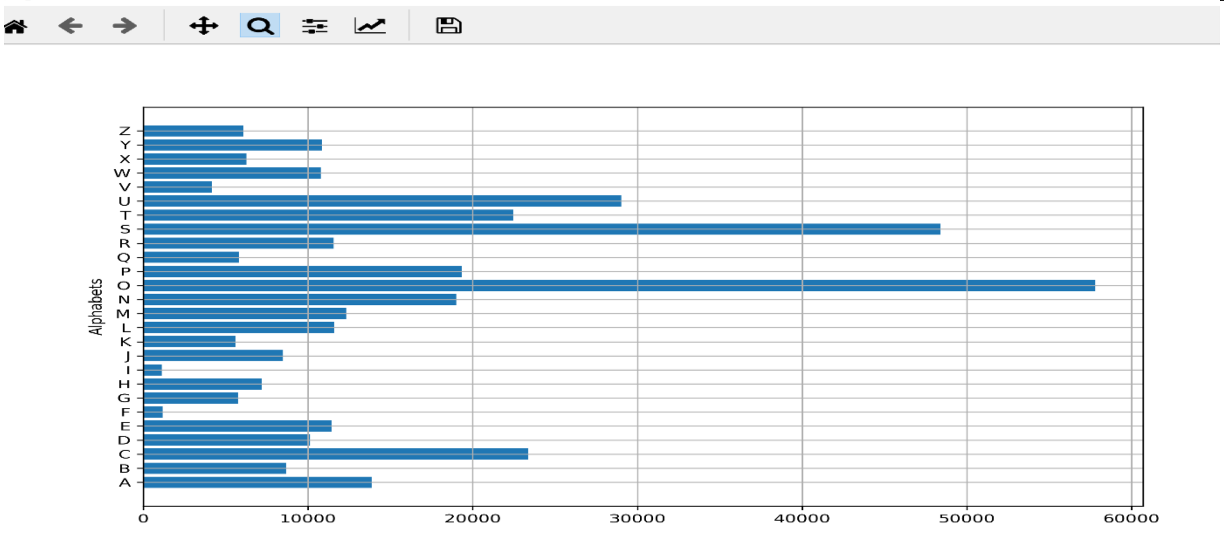


Figure 4: Count of alphabets

7. Shuffle some of the images of the Train set using **shuffle()** function so that we can display some random images.

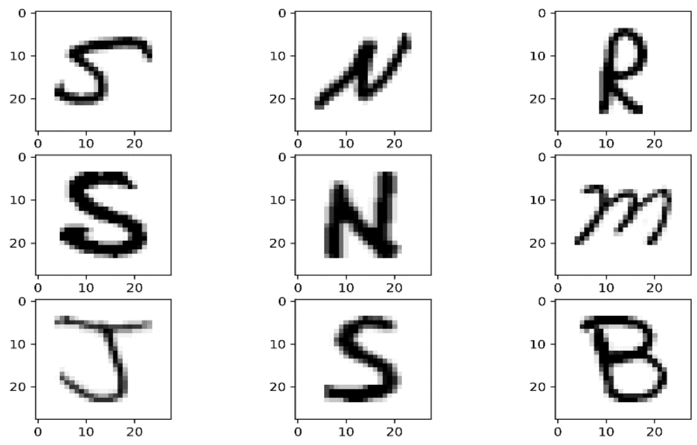


Figure 5: Shuffled Images

8. Reshape the Train and Test image dataset so that they can be put in the model. Total there are 372450 images, we split the dataset into two parts, part of the sample for actual Training and remaining samples for Testing the model. Train dataset size to be 80% and Test dataset size to be 20%. This will give good idea on the accuracy of the model.
9. Convert the single float values to categorical values. As the CNN model takes input of labels & generates the output as a vector of probabilities.
10. Design a CNN model. CNN performs best with image recognition issues because it works well on imagery that is interpreted as grid structures. The dropout layer is used to deactivate certain neurons, which limits the model's bid fitting during practicing. The convolution layers are mostly followed by maxpool layers that are used to lessen the number of features extracted and finally the output of the maxpool and layers and convolution layer are flattened into a vector of single dimension and are given as an input to the Dense Layer.

CNN

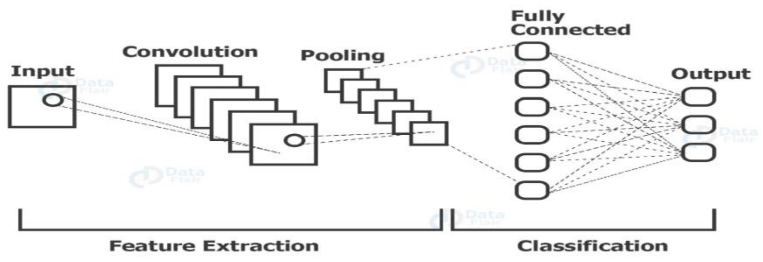
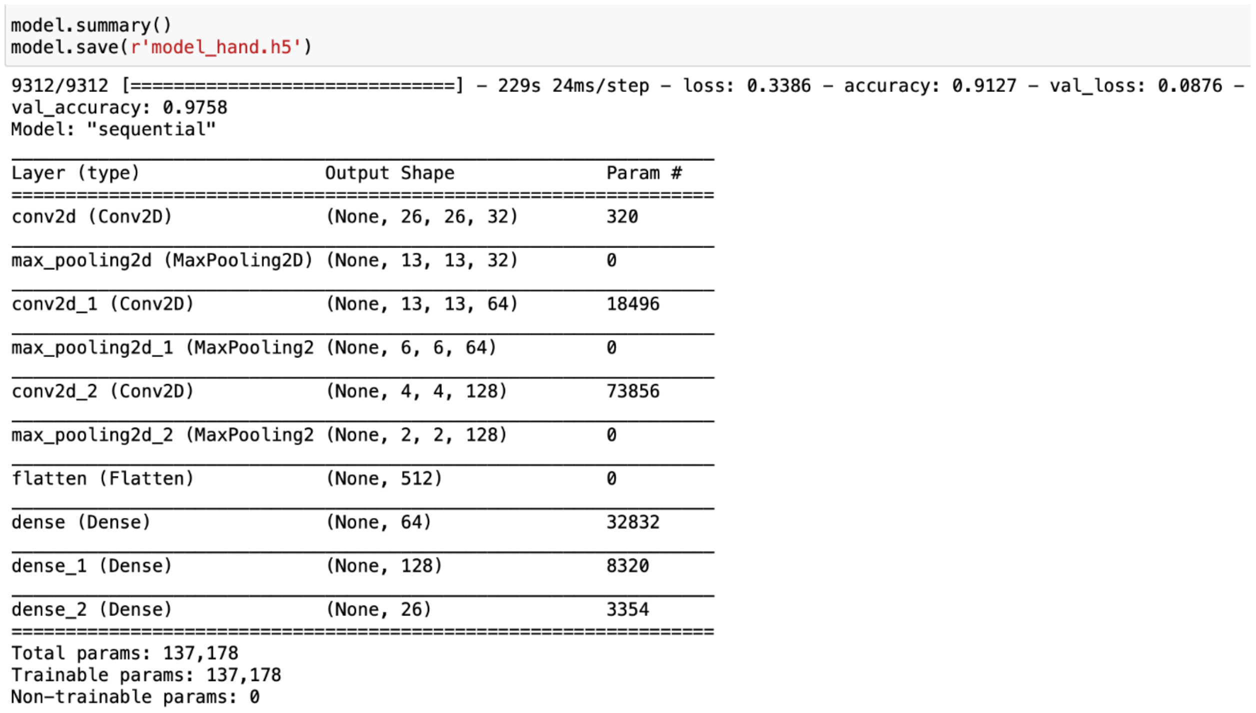


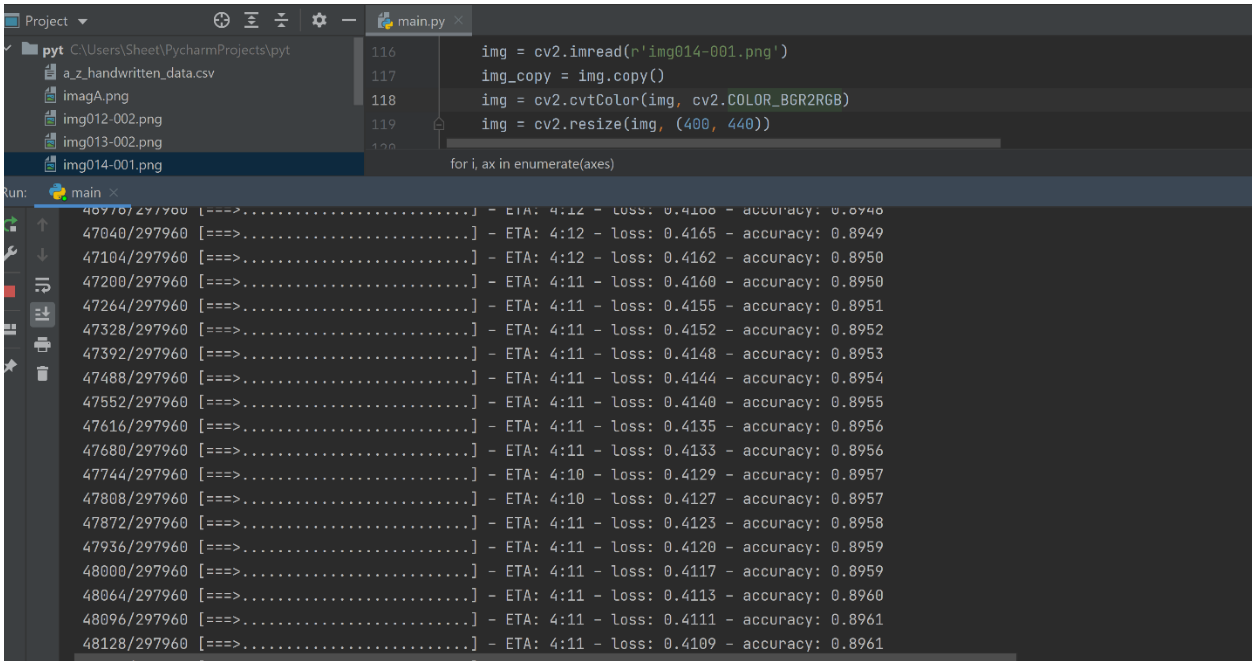
Figure 6: Architecture of CNN Model



```
model.summary()
model.save(r'model_hand.h5')
```

```
9312/9312 [==============================] - 229s 24ms/step - loss: 0.3386 - accuracy: 0.9127 - val_loss: 0.0876 -
val_accuracy: 0.9758
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 13, 13, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 4, 4, 128)         73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 2, 2, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 512)               0
_________________________________________________________________
dense (Dense)                (None, 64)                32832
_________________________________________________________________
dense_1 (Dense)              (None, 128)               8320
_________________________________________________________________
dense_2 (Dense)              (None, 26)                3354
=================================================================
Total params: 137,178
Trainable params: 137,178
Non-trainable params: 0
```

Figure 7: Architecture for the neural network

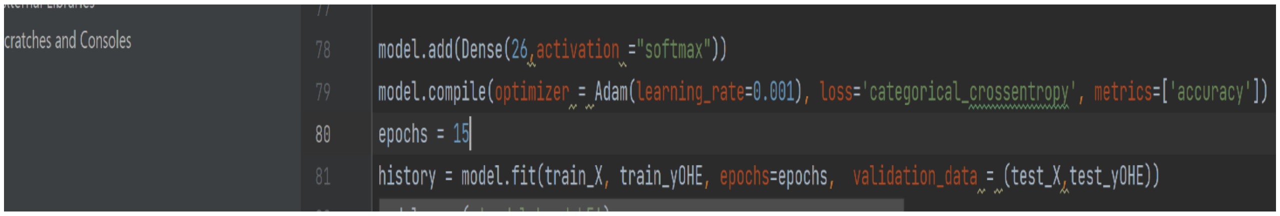


```python
img = cv2.imread(r'img014-001.png')
img_copy = img.copy()
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
img = cv2.resize(img, (400, 440))
```

```
46976/297960 [===>..........................] - ETA: 4:12 - loss: 0.4168 - accuracy: 0.8948
47040/297960 [===>..........................] - ETA: 4:12 - loss: 0.4165 - accuracy: 0.8949
47104/297960 [===>..........................] - ETA: 4:12 - loss: 0.4162 - accuracy: 0.8950
47200/297960 [===>..........................] - ETA: 4:11 - loss: 0.4160 - accuracy: 0.8950
47264/297960 [===>..........................] - ETA: 4:11 - loss: 0.4155 - accuracy: 0.8951
47328/297960 [===>..........................] - ETA: 4:11 - loss: 0.4152 - accuracy: 0.8952
47392/297960 [===>..........................] - ETA: 4:11 - loss: 0.4148 - accuracy: 0.8953
47488/297960 [===>..........................] - ETA: 4:11 - loss: 0.4144 - accuracy: 0.8954
47552/297960 [===>..........................] - ETA: 4:11 - loss: 0.4140 - accuracy: 0.8955
47616/297960 [===>..........................] - ETA: 4:11 - loss: 0.4135 - accuracy: 0.8956
47680/297960 [===>..........................] - ETA: 4:11 - loss: 0.4133 - accuracy: 0.8956
47744/297960 [===>..........................] - ETA: 4:10 - loss: 0.4129 - accuracy: 0.8957
47808/297960 [===>..........................] - ETA: 4:10 - loss: 0.4127 - accuracy: 0.8957
47872/297960 [===>..........................] - ETA: 4:11 - loss: 0.4123 - accuracy: 0.8958
47936/297960 [===>..........................] - ETA: 4:11 - loss: 0.4120 - accuracy: 0.8959
48000/297960 [===>..........................] - ETA: 4:11 - loss: 0.4117 - accuracy: 0.8959
48064/297960 [===>..........................] - ETA: 4:11 - loss: 0.4113 - accuracy: 0.8960
48096/297960 [===>..........................] - ETA: 4:11 - loss: 0.4111 - accuracy: 0.8961
48128/297960 [===>..........................] - ETA: 4:11 - loss: 0.4109 - accuracy: 0.8961
```

Figure 8: Training of CNN

11. Compiling the model where we define the **optimizing function** and **loss function** to be used for fitting. We have trained a dataset for multiple epochs.

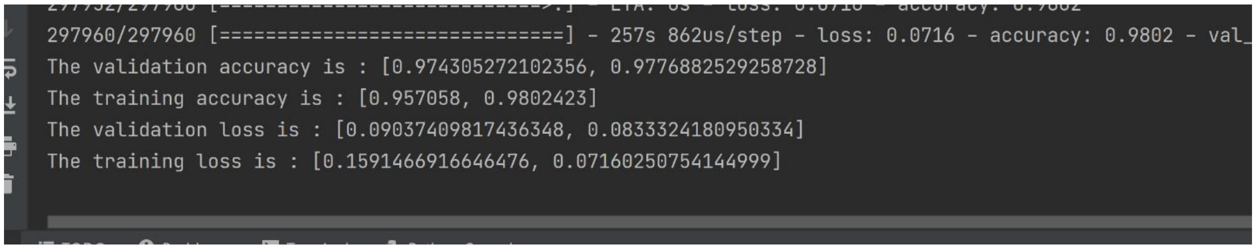


```python
model.add(Dense(26,activation ="softmax"))
model.compile(optimizer = Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])
epochs = 15
history = model.fit(train_X, train_yOHE, epochs=epochs,  validation_data = (test_X,test_yOHE))
```

Figure 9: Compiling the model with multiple epochs

12. Getting the Model Summary. It tells us that what were the different layers defined in the model and save the model using **model.save()** function.

13. Getting the training & validation accuracies and losses for every epoch.



```
297960/297960 [==============================] - 257s 862us/step - loss: 0.0716 - accuracy: 0.9802 - val_
The validation accuracy is : [0.974305272102356, 0.9776882529258728]
The training accuracy is : [0.957058, 0.9802423]
The validation loss is : [0.09037409817436348, 0.0833324180950334]
The training loss is : [0.1591466916646476, 0.07160250754144999]
```

Figure 10: Training & validation accuracies and losses for every epoch

14. Then we plot graphs for accuracy vs epoch and loss vs epoch for the model

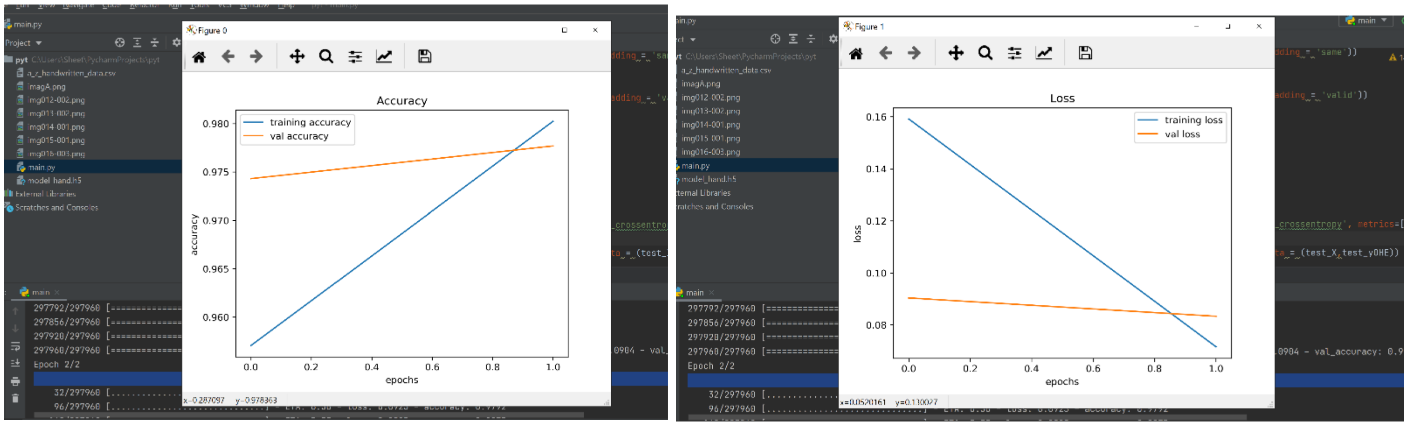


Figure 11: Plot graphs for accuracy vs epoch and loss vs epoch for the model

15. Doing some Prediction on Test Data. We are creating 9 subplots of (3,3) shape and visualize some of the test dataset alphabets along with their predictions using **model.predict()** function.

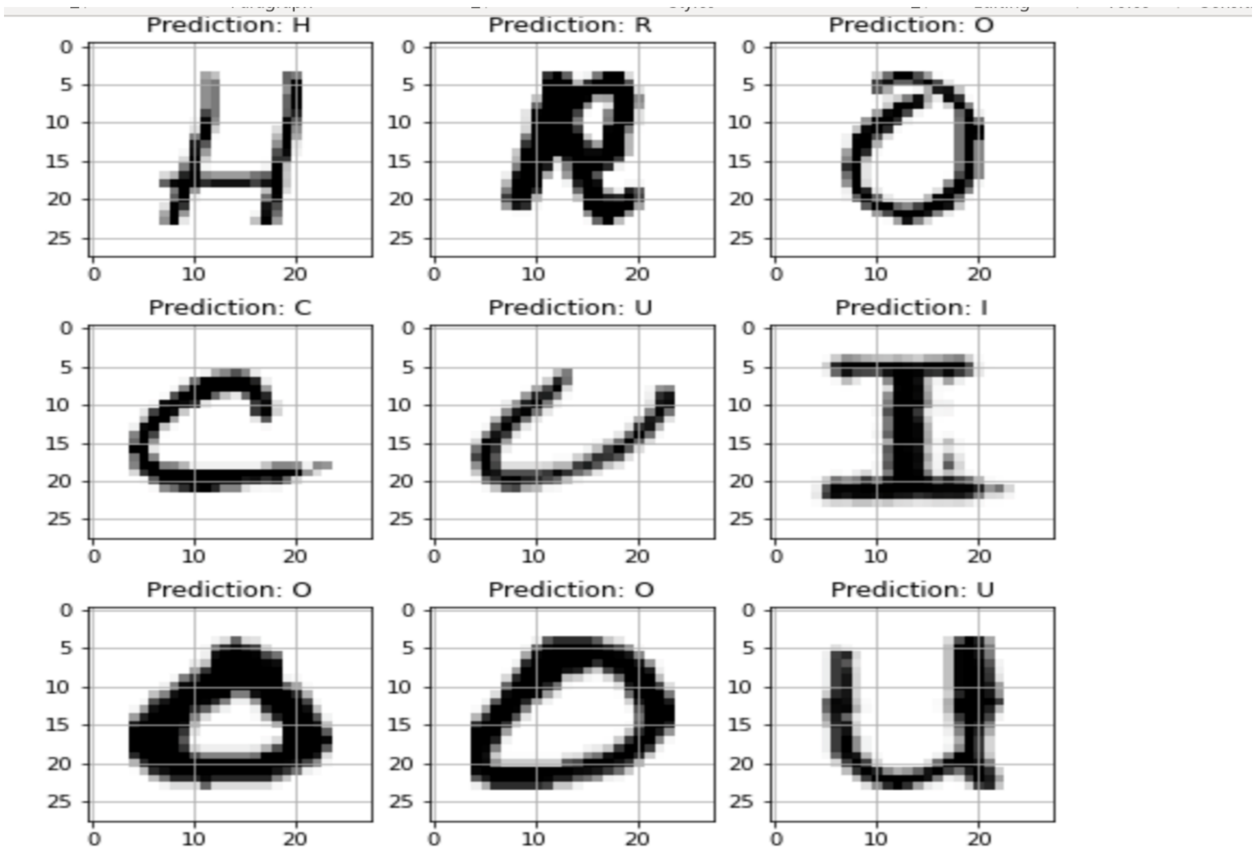


Figure 12: Output of prediction on test data

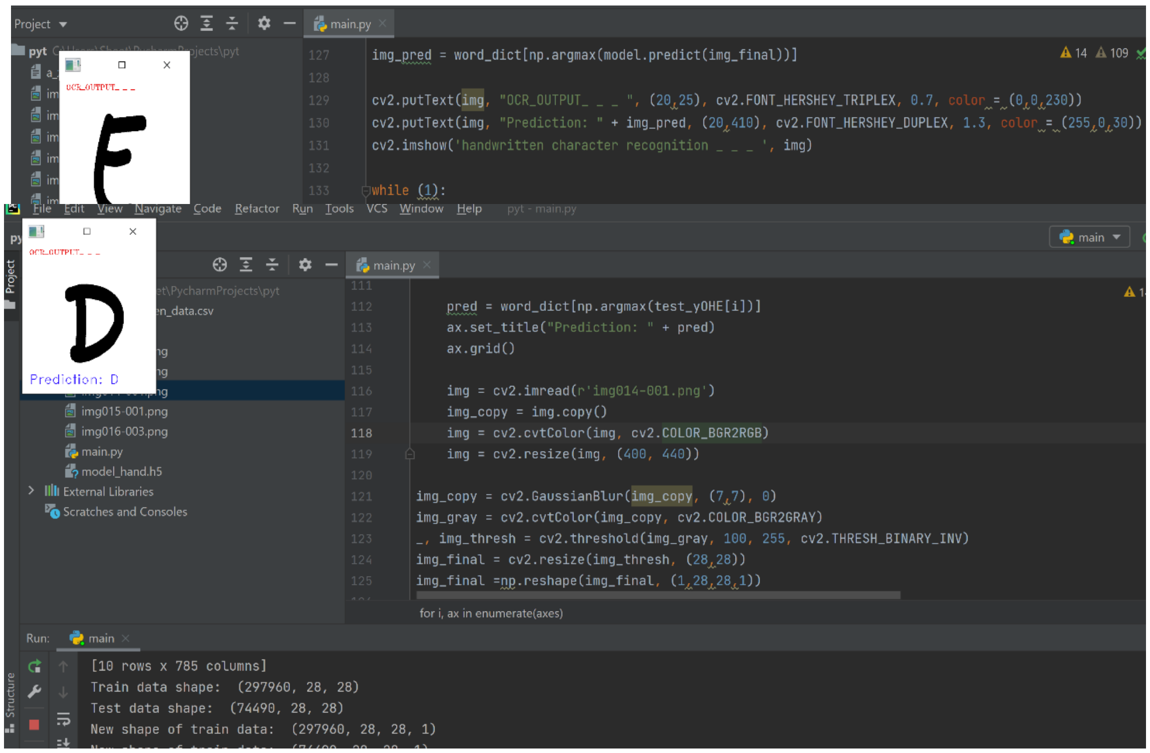
16. Doing prediction on External Image. Here we have read an external image that is originally an image of alphabet "E" and made a copy of it using **img_copy()** function that is to go through some processing to be fed to the model for the prediction.

OUTPUT

Figure 13: Prediction on External Image Character ‘E’

Figure 14: Prediction on External Image Character ‘D’

CONCLUSION



```
img_pred = word_dict[np.argmax(model.predict(img_final))]

cv2.putText(img, "OCR_OUTPUT_ _ _ ", (20,25), cv2.FONT_HERSHEY_TRIPLEX, 0.7, color = (0,0,230))
cv2.putText(img, "Prediction: " + img_pred, (20,410), cv2.FONT_HERSHEY_DUPLEX, 1.3, color = (255,0,30))
cv2.imshow('handwritten character recognition _ _ _ ', img)

while (1):
```

```

    pred = word_dict[np.argmax(test_yOHE[i])]
    ax.set_title("Prediction: " + pred)
    ax.grid()

    img = cv2.imread(r'img014-001.png')
    img_copy = img.copy()
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = cv2.resize(img, (400, 440))

img_copy = cv2.GaussianBlur(img_copy, (7,7), 0)
img_gray = cv2.cvtColor(img_copy, cv2.COLOR_BGR2GRAY)
_, img_thresh = cv2.threshold(img_gray, 100, 255, cv2.THRESH_BINARY_INV)
img_final = cv2.resize(img_thresh, (28,28))
img_final =np.reshape(img_final, (1,28,28,1))
```

```
[10 rows x 785 columns]
Train data shape:  (297960, 28, 28)
Test data shape:  (74490, 28, 28)
New shape of train data:  (297960, 28, 28, 1)
```

For English character recognition, the report proposed a deep learning-based neural network model. On the Kaggle dataset, we trained a CNN classification model that had a 97.5 percent accuracy. To the best of our knowledge, this is the best accuracy on this dataset. We also discussed how we came up with the concept, how we went about implementing it, and how our tests helped us learn more about English character recognition. In order to increase the accuracy of identification and classification, we used data augmentation techniques.

In the feature extraction part, we began with a simple convolution neural network with two convolution layers and two max pool layers, followed by a SoftMax layer for final classification. Character recognition and classification were possible with the model. We improved this model by experimenting with various convolutional parameter combinations. We used a 3x3 kernel size for the first convolution layer and a 5x5 kernel size for the Deep Learning Based Real Time Devanagari Character Recognition 59 layer. The other set of kernel size values for the first and second layers were 5x5 and 7x7, respectively.

On this dataset, the previous researchers were able to achieve an accuracy of 96.4 percent. This was the model's highest level of accuracy to date. This record was broken by our experiment, which used a cross combination of different kernel sizes. The model with 3x3 and 5x5 kernel sizes for the first and second layers, respectively, was able to achieve a 97.5 percent accuracy. This was followed by a model with a 7x7 kernel dimension. This model was able to achieve a 97.25 percent accuracy rate.

PROJECT MANAGEMENT

Task	**Sub-Task**	**Completed By**
Literature Review	Paper research for OCR	Sheetal & Parth
	Understanding how OCR Works	Sheetal & Parth

	OCR Architecture	Sheetal & Parth
	Code Source	Sheetal
	Finding appropriate input character dataset	Sheetal
Implementation Technologies Selection	Python Tensorflow keras opencv	Sheetal & Parth
Presentations & Reports	Project Proposal Presentation	Sheetal & Parth
	Project Intermediate Report	Parth
	Final Report	Sheetal
Technical Implementation	Importing and Installation of all necessary libraries in the system	Sheetal
	Reading, Splitting, Shuffling, Reshaping (test and training dataset), Labelling Characters of the dataset	Sheetal & Parth
	Training the model with CNN	Sheetal
	Increasing the accuracy using multiple epochs along with graphs	Sheetal
	Successful prediction on test and external data	Sheetal

KNOWLEDGE GAINED

I researched and understood basic machine learning concepts as well as technologies associated with it. Furthermore, I understood how it can be used in optical character recognition (OCR).

Along the way, I started learning about machine learning tools and its characteristics.

I also learned how to import and use all the Python libraries like Pandas, NumPy, Matplotlib, Keras that are used for Character recognition. I studied more about computer vision and OpenCV library of python. I also studied the implementation of OpenCV library. I learned about Convolution Neural Networks (CNN) as well as how it fits perfectly into my project. I also built and trained the Convolutional neural network which is very effective for image classification purposes. I discovered the main reason why CNN works well for image classification problems and trained the model with all the above information.

While going through the entire setup of importing libraries, frameworks, and IDEs, I faced multiple errors and I started troubleshooting them. While solving these errors and issues, I came across multiple new sources which increased my machine learning technical knowledge and this will be really useful to me in the future.

REFERENCES

1. Sable, A. (2021, April 15). Building custom deep learning based OCR models. Retrieved April 19, 2021, from https://nanonets.com/blog/attention-ocr-for-text-recogntion/

2. Namysl, M., &amp; Konya, I. (2019). Efficient, lexicon-free ocr using deep learning. 2019 International Conference on Document Analysis and Recognition (ICDAR). doi:10.1109/icdar.2019.00055
3. Rawls, S., Cao, H., Sabir, E., &amp; Natarajan, P. (2017). Combining deep learning and language modeling For SEGMENTATION-FREE OCR from raw pixels. 2017 1st International Workshop on Arabic Script Analysis and Recognition (ASAR). doi:10.1109/asar.2017.8067772
4. Abadi, Martín; Barham, Paul; Chen, Jianmin; Chen, Zhifeng; Davis, Andy; Dean, Jeffrey; Devin, Matthieu; Ghemawat, Sanjay; Irving, Geoffrey; Isard, Michael; Kudlur, Manjunath; Levenberg, Josh; Monga, Rajat; Moore, Sherry; Murray, Derek G.; Steiner, Benoit; Tucker, Paul; Vasudevan, Vijay; Warden, Pete; Wicke, Martin; Yu, Yuan; Zheng, Xiaoqiang (2016). "TensorFlow: A System for Large-Scale Machine Learning"
5. Team, K. (n.d.). Keras documentation: Why Choose keras? Retrieved April 19, 2021, from https://keras.io/why_keras/
6. Deep learning project - handwritten digit recognition using python. (2021, March 14). Retrieved April 27, 2021, from https://data-flair.training/blogs/python-deep-learning-project-handwritten-digit-recognition/
7. J. Schmidhuber, "Deep Learning in Neural Networks: An Overview," Neural Networks, vol.
8. V. Ganapathy and C. C. H. Lean, "Optical Character Recognition Program for Images of Printed Text using a Neural Network," 2006 IEEE International Conference on Industrial Technology, Mumbai, 2006, pp. 1171-1176.doi: 10.1109/ICIT.2006.372591

9. H. Zhao, Y. Hu and J. Zhang, "Character Recognition via a Compact Convolutional Neural Network," 2017 International Conference on Digital Image Computing: Techniques and Applications (DICTA), Sydney, NSW, 2017, pp. 1-6.doi: 10.1109/DICTA.2017.8227414