[idea 20] An Automated Optical Character Recognition of Handwritten English Letters using Artificial Neural Networks

Faculty of Computers and Artificial Intelligence
Artificial Intelligence
Fall semester 2023-2024

Artificial Intelligence project

Team Members

	Name	ID	Department
1	محمد مسعد محمد سليم عبد الخالق	20210833	CS
2	يارا محمد عطيه الطوخي يس	20211041	CS
3	كريم عمرو امام حسين	20210687	CS
4	مله الله عصام عبد العزيز عبد الفتاح	20210959	CS
5	مينا نادى فرج توفيق	20210986	CS
6	يوسف محمد حامد حسن	20211092	IS
7	ساره عبد الغنى حسينى ابراهيم محفوظ	20210396	IS

Link: https://github.com/mohamedmosaadmohed/AI_Project.git

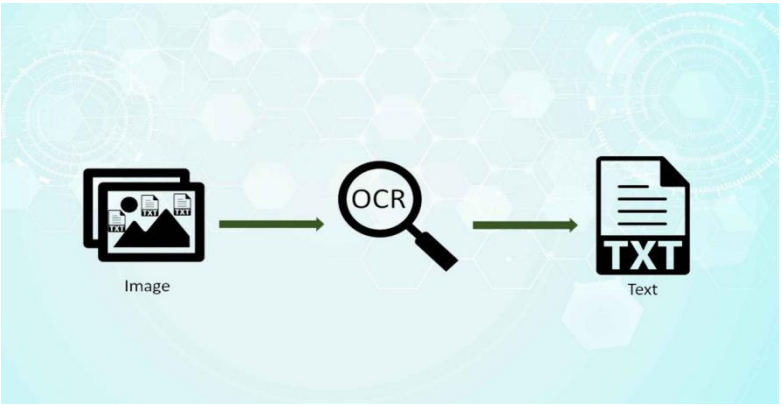
1. Project Idea :-

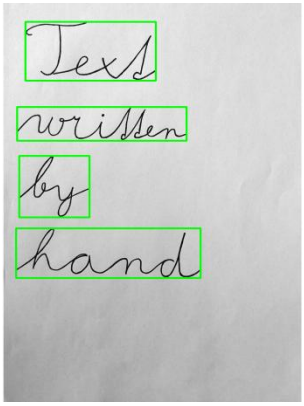
Optial character recognition

OCR stands for Optical Character Recognition. It is a technology that converts different types of documents, such as scanned paper documents, PDFs, or images captured by a digital camera, into editable and searchable data.

The primary purpose of OCR is to recognize and extract text information from images or scanned documents, making it possible to convert these documents into machine-encoded text. This enables users to edit, search, and work with the text as if it originated from a standard digital document rather than a static image.

OCR systems use various algorithms and techniques to identify characters and words within an image. These algorithms analyze the shapes, patterns, and structures of the characters to recognize and convert them into machine-readable text.





Text written by hand

How does OCR work?

The OCR engine or OCR software works by using the following steps:

1-Image Acquisition:

The OCR process starts with acquiring an image containing text. This image could be obtained through various means, such as scanning a document, taking a photograph, or capturing a scene with text.

2-Preprocessing:

The acquired image often undergoes preprocessing to enhance its quality and make it more suitable for character recognition.

- Common preprocessing steps include image normalization, noise reduction, contrast adjustment, and other techniques to improve the clarity of the text.

3-Segmentation

- Text localization involves identifying regions within the preprocessed image that contain text. This step is crucial for isolating the text from the rest of the image.
- Techniques like edge detection or connected component analysis may be employed to locate potential text regions.

4-Feature Extraction:

- After segmentation, the system extracts relevant features from each segmented character. Features are distinctive characteristics that help differentiate one character from another.
- Features could include aspects like the shape, size, and spatial relationships of various components within a character.

5-Classification:

- The extracted features are used for character recognition. The OCR system employs a classification algorithm to assign a label or category to each segmented character based on its features.

- Common classification algorithms include neural networks (e.g., CNNs), support vector machines (SVM), decision trees, and k-nearest neighbors (k-NN).

6-Postprocessing:

- After classification, postprocessing steps may be applied to refine the results and improve overall accuracy.
- Postprocessing can include error correction, context-based corrections, and other techniques to enhance the quality of the recognized text.

These steps collectively allow OCR systems to convert images containing text into machine-readable text, enabling users to edit, search, and work with the text as if it originated from a standard digital document. The success of OCR depends on the quality of the acquired image, the effectiveness of preprocessing, segmentation, feature extraction, and the accuracy of the classification algorithm, with postprocessing serving as a final refinement step.

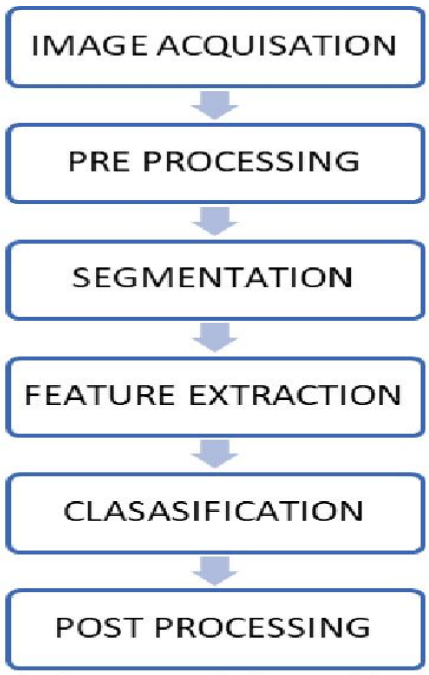
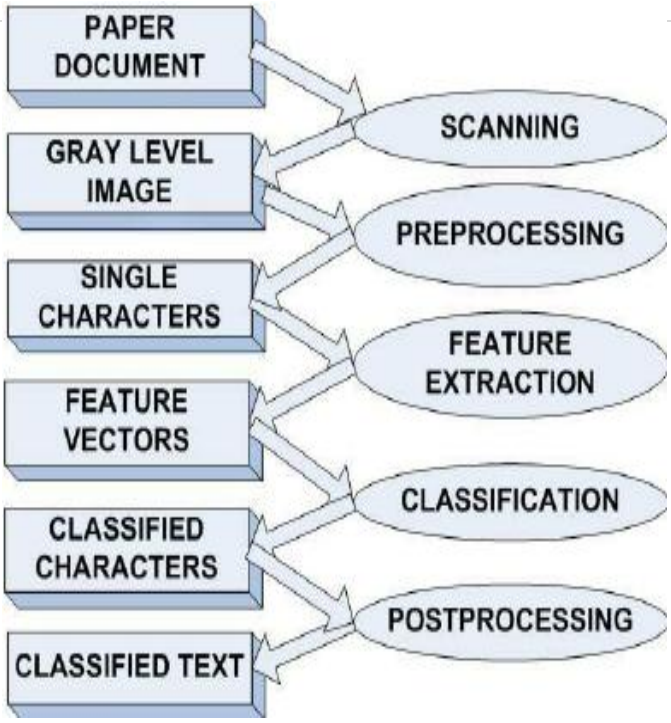


Figure 1: Steps towards Recognition in OCR

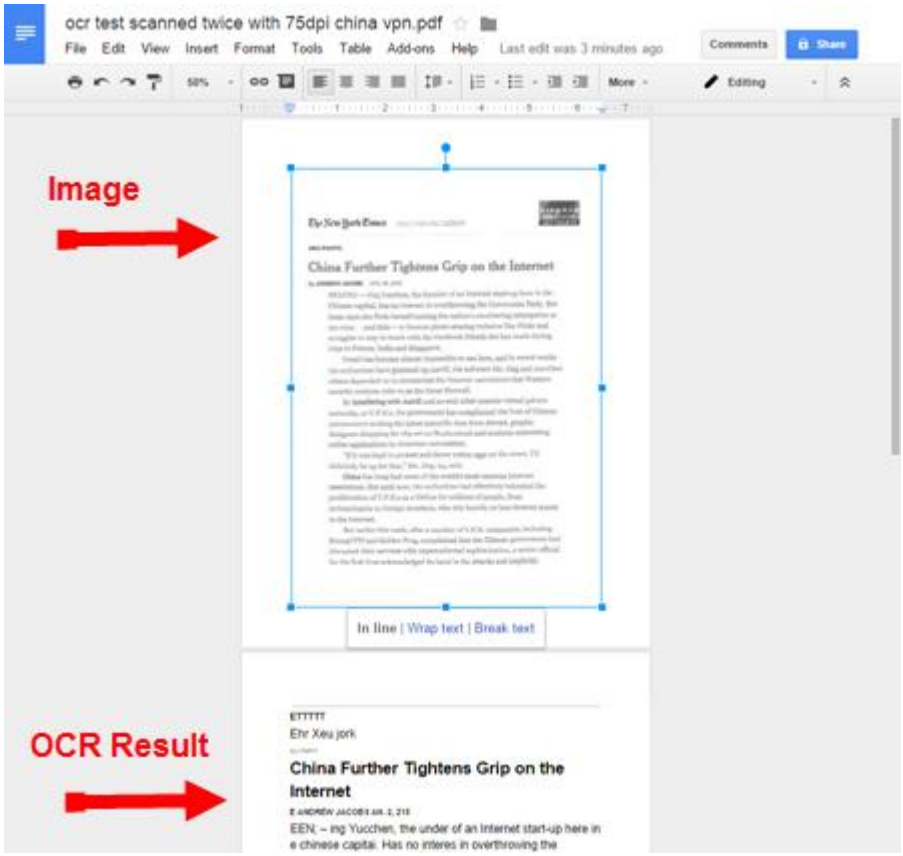


2. Applications :-

1-Google Docs

is a web-based application that uses OCR technology to recognize and convert handwritten or printed text into digital text. It can be used to digitize documents, scan paperwork, and convert images of

text into editable and searchable data.



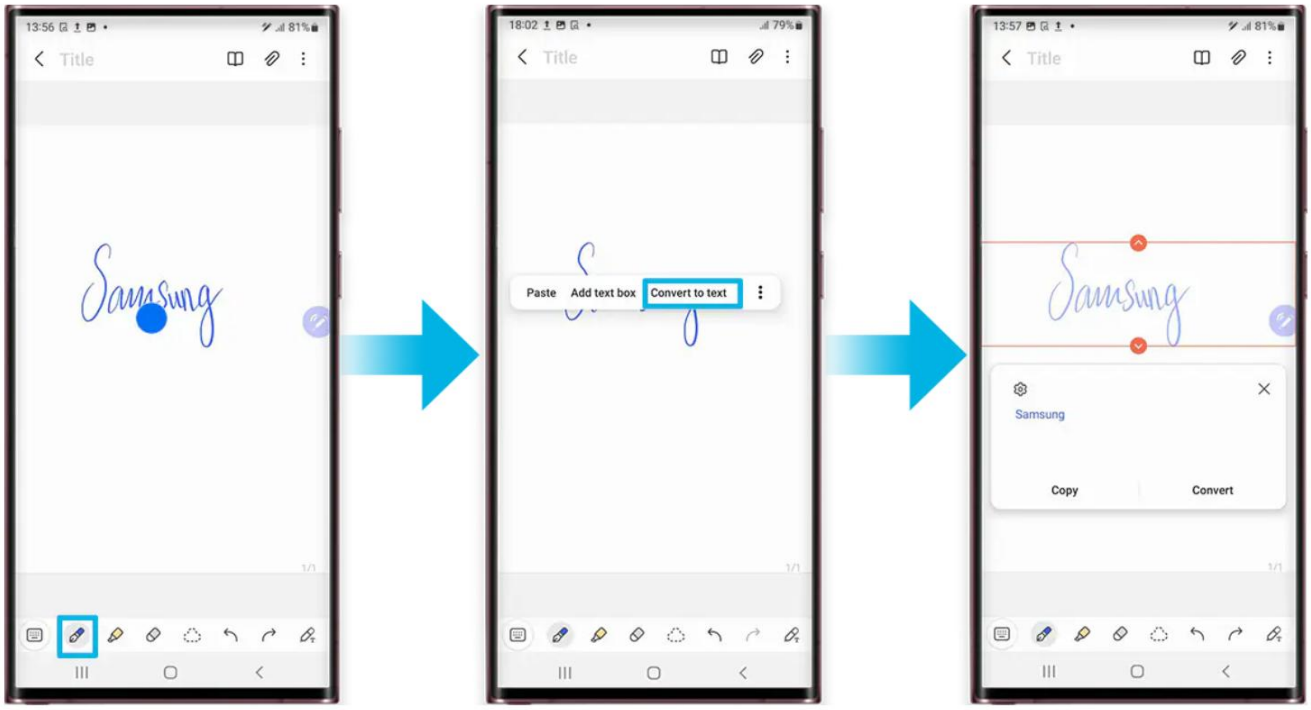
Google Docs OCR is essentially one of the free alternatives offered by Google Drive to convert images to text. The process behind Google Docs OCR is nothing more than uploading images from which you need the data, into Google Docs, and exporting the data as a text format into your computer.

2-Samsung Notes

Samsung Notes is a note-taking application developed by Samsung. It allows users to write, draw, and highlight in a variety of styles and colors with the S Pen.

Samsung Notes supports OCR (Optical Character Recognition) technology, which is used to recognize and convert handwritten or printed text into digital text. This feature allows users to search for

specific words or phrases within their notes.

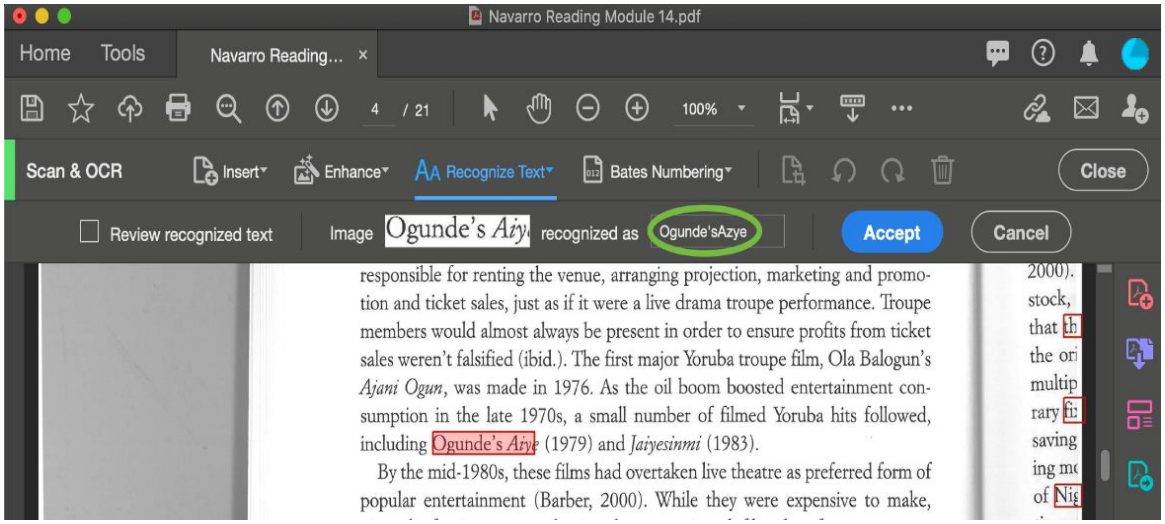


3-Adobe Acrobat Pro

Adobe Acrobat Pro is a software suite that includes a range of tools for creating, editing, and managing PDF files. One of the key features of Adobe Acrobat Pro is its Optical Character Recognition (OCR) technology, which is used to convert scanned files, PDF files, and image files into editable and searchable documents.

Adobe Acrobat Pro's OCR technology can analyze images as they are scanned into the program, or it can analyze already existing images,

PDF files, or other file types after PDF conversion. The program then applies optical character recognition to the document, making the resulting files editable and searchable.



3. A Literature Review of Academic publications

1-Handwriting Arabic Character Recognition LeNet Using

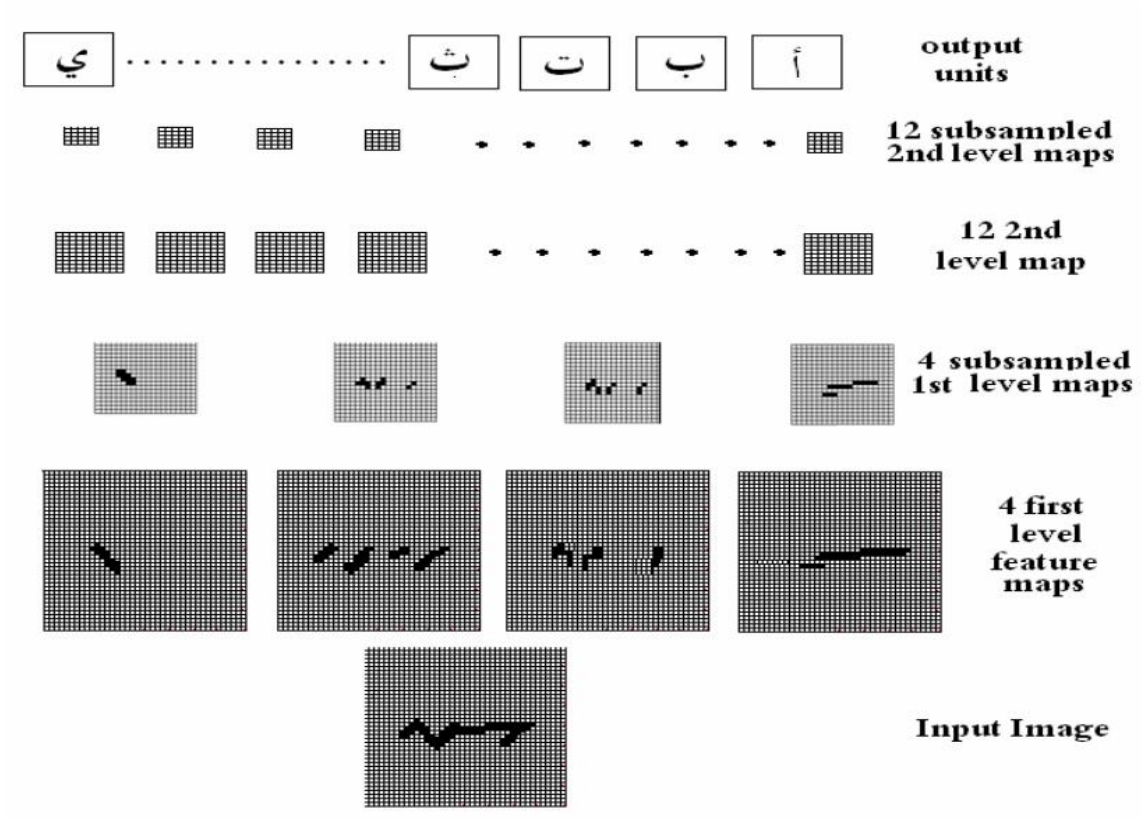


Figure 2. LeNet architecture.

Neural Network

The paper introduces a new neural network designed to recognize handwritten Arabic characters. The network consists of two stages: the first stage recognizes the main shape of the character, and the second stage is dedicated to dots recognition. The document outlines the characteristics, structure, and training algorithm for this network. The LeNet architecture is introduced, recognizing one character at a time. The architecture includes feature maps, weight sharing, and two hidden layers named H1 and H2.

SOURCE: https://iajit.org/PDF/vol.6,no.3/15.pdf

2-A Novel Approach to On-Line Handwriting Recognition Based on

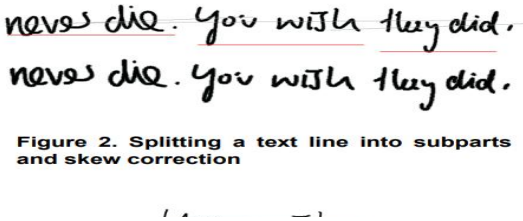


Figure 2. Splitting a text line into subparts and skew correction

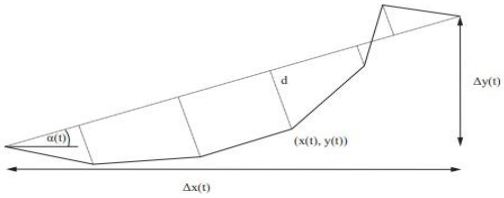


Figure 4. Features of the vicinity

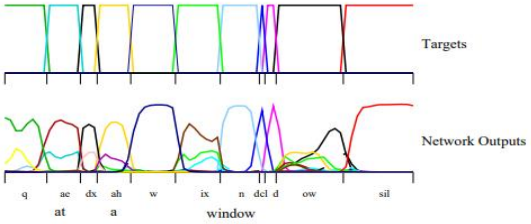


Figure 5. Individual timestep classification with a standard RNN

Bidirectional Long Short-Term Memory Networks

This paper presents a novel approach to online handwriting recognition, focusing on the specific challenge of recognizing handwritten whiteboard notes. The proposed system employs a bidirectional recurrent neural network (RNN) with long short-term memory (LSTM) blocks. The authors use the Connectionist Temporal Classification (CTC) objective function, a recent addition to RNNs, which trains the network to label unsegmented sequence data directly.

SOURCE: https://mediatum.ub.tum.de/doc/1289961/document.pdf

3-Classification and Learning for Character Recognition: Comparison of Methods and Remaining Problems

Table 1. A citation of error rates (%) on the MNIST test set.

Feature	pixel	PCA	grad-4	grad-8
k-NN	3.66	3.01	1.26	0.97
MLP	1.91	1.84	0.84	0.60
RBF	2.53	2.21	0.92	0.69
PC	1.64	N/A	0.83	0.58
SVC-poly	1.69	1.43	0.76	0.55
SVC-rbf	1.41	1.24	0.67	0.42

This paper provides a comprehensive exploration of classification methods in character recognition. it discusses the evolution from traditional OCR methods to learning-based approaches, such as statistical methods, artificial neural networks (ANNs), support vector machines (SVMs), and multiple classifier combination. The paper highlights the successes and remaining challenges in character recognition, emphasizing the need for continuous improvement in accuracy, reliable confidence measures, and rejection capabilities for ambiguous patterns.

It also touches on the comparison between statistical and discriminative classifiers and the distinctions between neural networks and SVMs. The authors conclude by identifying areas for future research, including incremental learning and the importance of standardized benchmarking methods for fair comparisons in character recognition techniques.

SOURCE: https://rb.gy/a22ya2

4-Handwritten English Character Recognition Using Logistic Regression and Neural Network

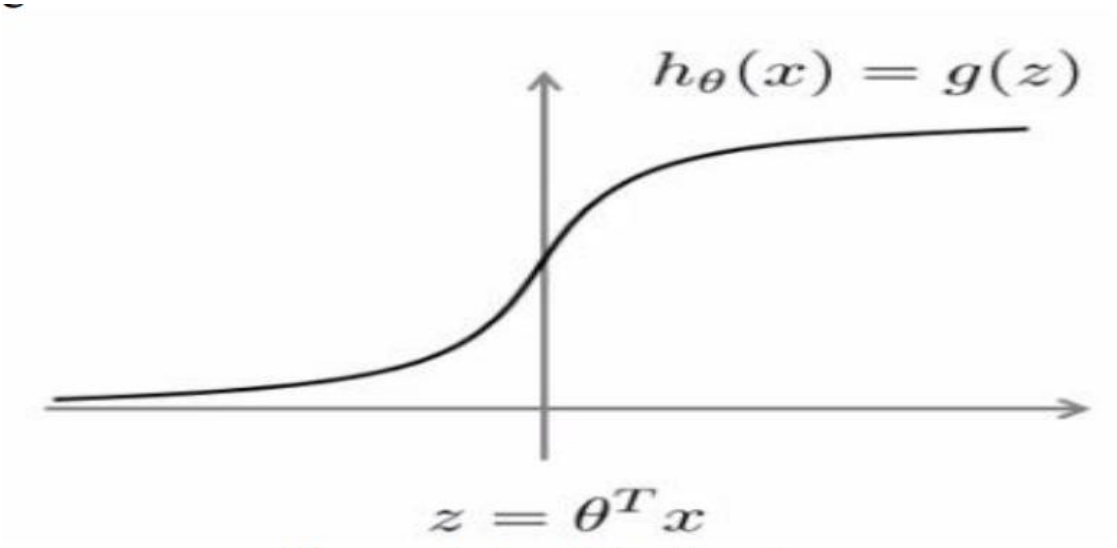


Figure 4: Logistic function

This paper addresses the challenge of handwritten English character recognition using machine learning techniques, specifically Logistic Regression and Neural Network. The authors propose an Optical Character Recognition (OCR) system consisting of two steps: generating a training set using an OCR tool and applying different machine learning algorithms for learning. They emphasize the importance of feature extraction and classification algorithms in the success of OCR and describe their approach to segmenting images into regions containing text. The OCR tool they use involves preprocessing, layout analysis, and data labeling. The paper details the implementation of Regularized Logistic Regression and Neural Network algorithms, discussing the challenges of overfitting and the use of regularization. The authors present their results, showing training and testing accuracy for both algorithms, and conclude by suggesting future improvements and applications, such as feature extraction enhancement and real-time handwriting identification in mobile applications.

SOURCE: https://rb.gy/3wud05

5-Convolutional Neural Network Committees for Handwritten Character Classification

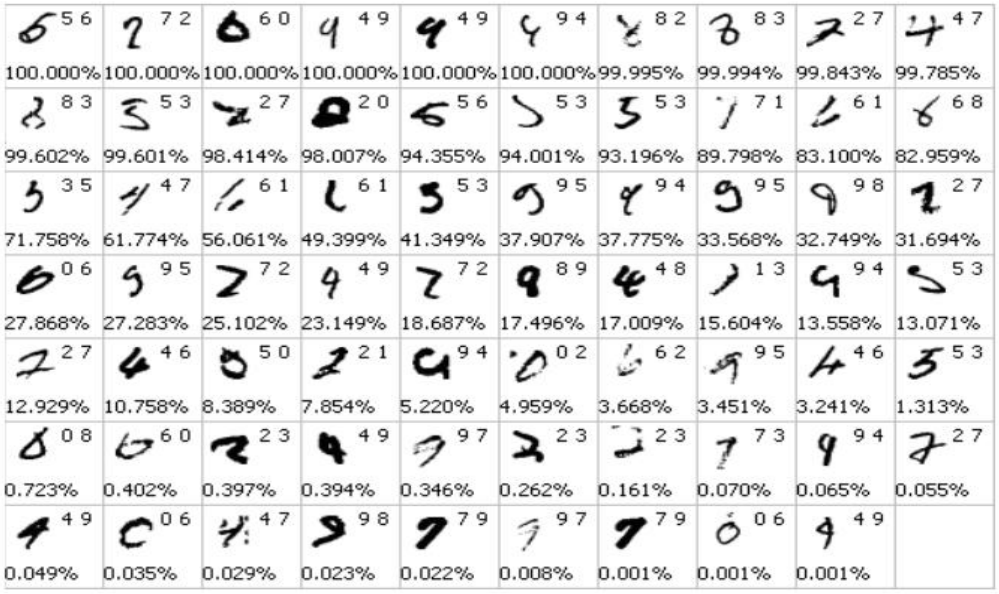


Figure 2. The 69 errors of all committees, the label (up left), the committee majority vote (up right), and the percentage of committees committing a particular error (down left).

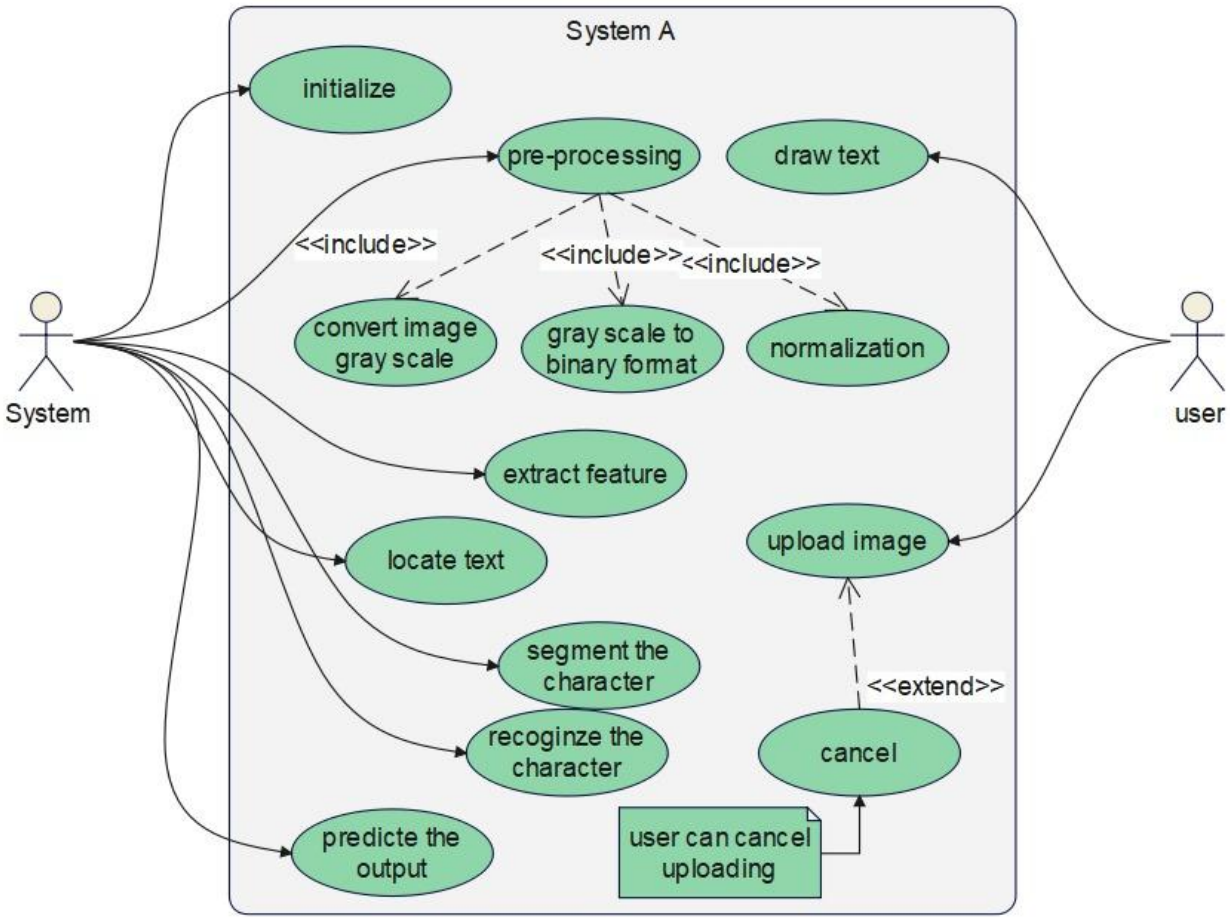
This paper presents a novel approach to improving the accuracy of handwritten character recognition using Convolutional Neural Networks (CNNs).

The paper introduces the concept of committees of neural networks to enhance recognition rates. Instead of relying on a single best-performing classifier, the authors propose training multiple classifiers and forming committees. These committees aim to minimize errors by training identical classifiers on preprocessed/normalized data in different ways, achieving better diversity in error patterns.

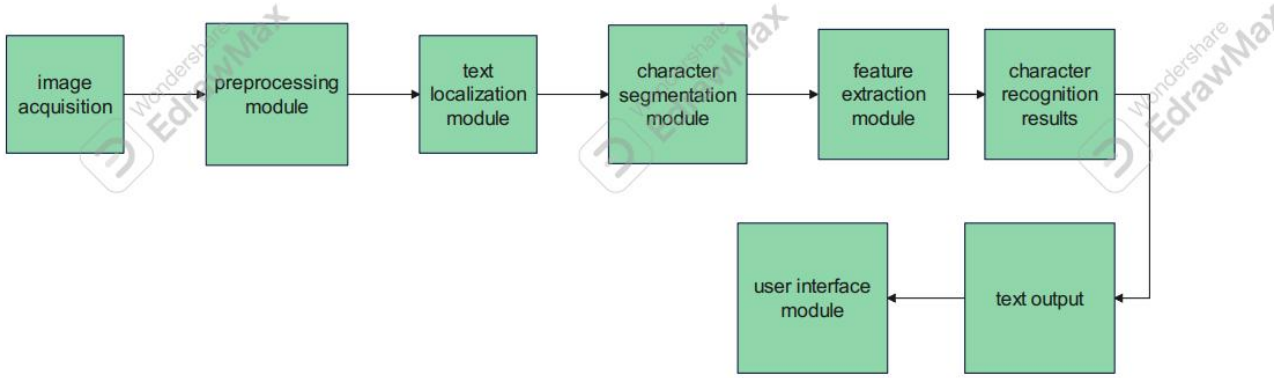
SOURCE: https://ieeexplore.ieee.org/abstract/document/6065487

4. Diagrams :-

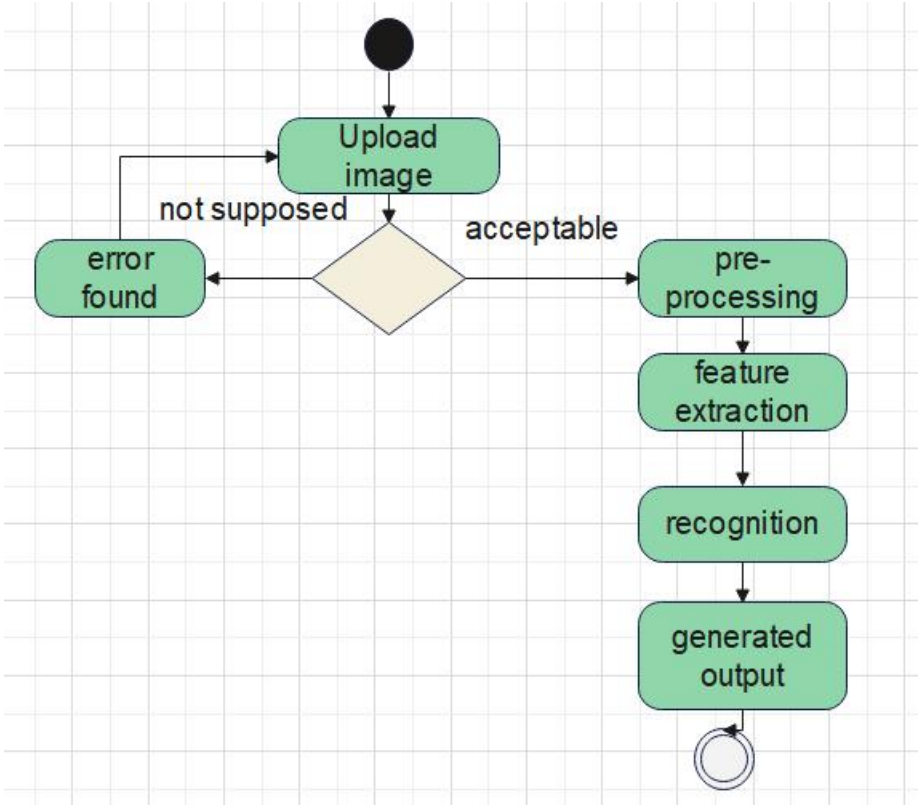
(4.1)use case diagram:



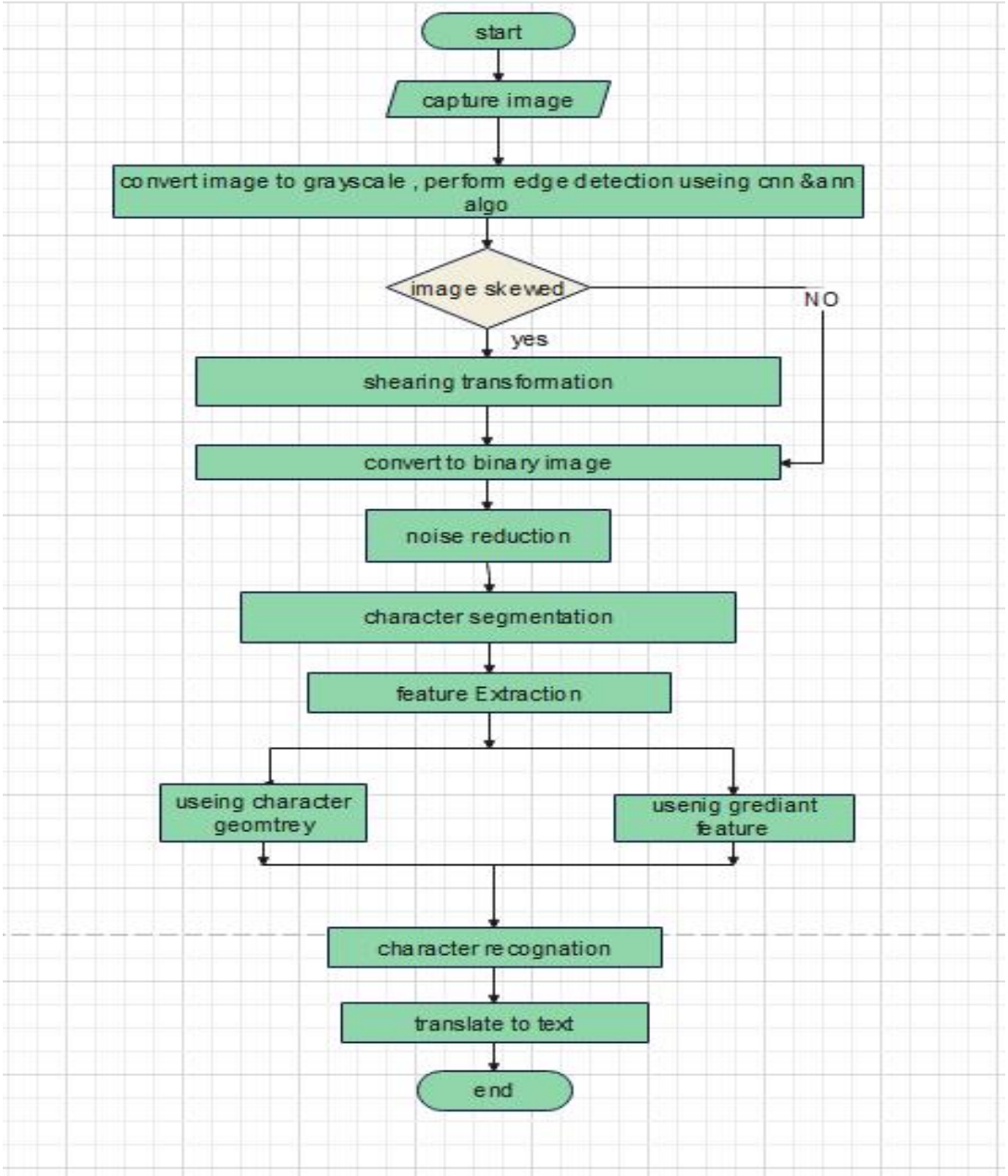
(4.2) Block Diagram:



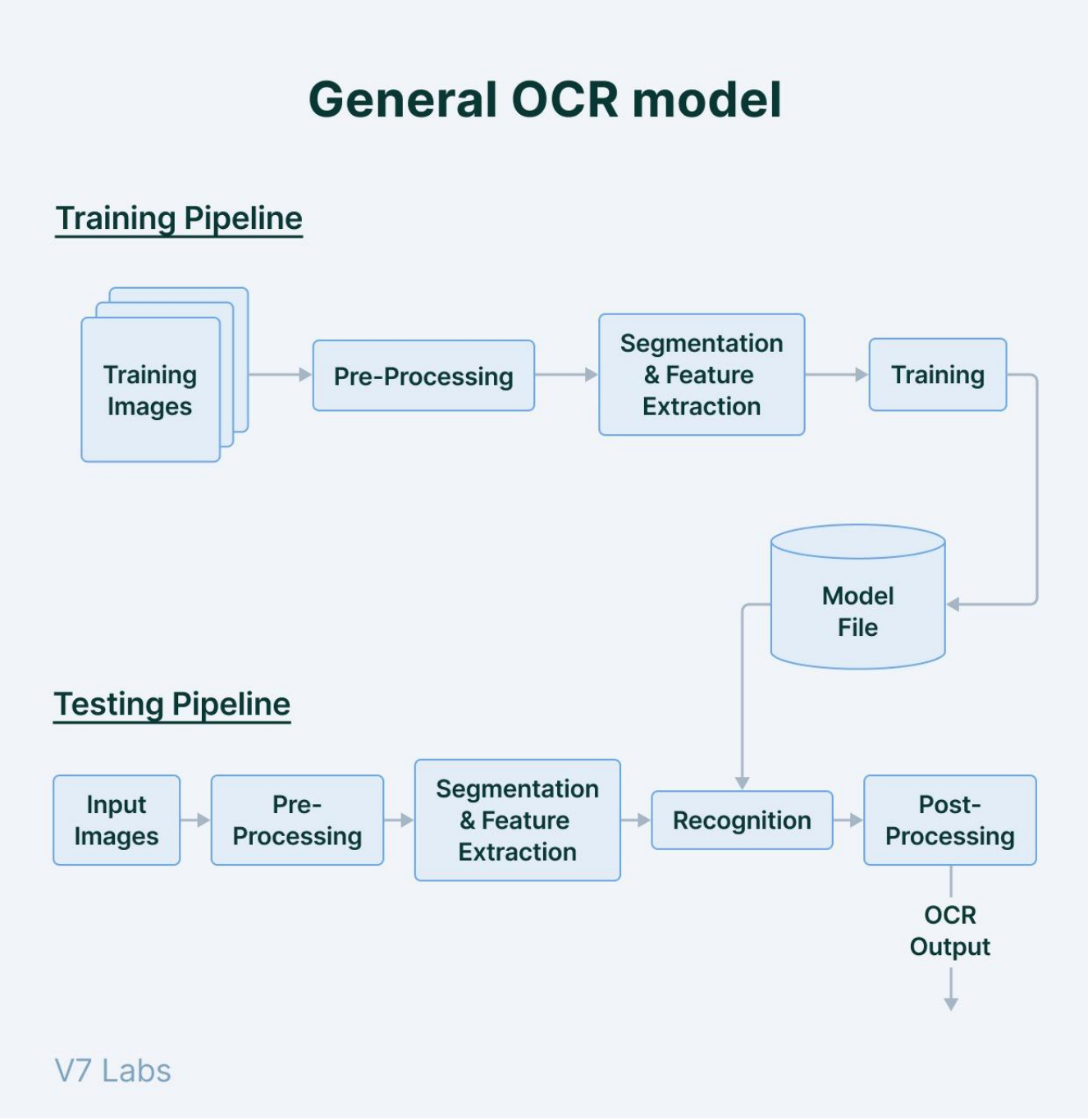
(4.3) Activity Diagram:



(4.4) Flow Chart :



(4.5) Other Diagrams:



Datasets:

https://www.kaggle.com/datasets/crawford/emnist?select=emnist-letters-test.csv

https://www.kaggle.com/datasets/crawford/emnist?select=emnist-letters-train.csv

5. Model :-

Optical Character Recognetion Models :

CNN & ANN :

There are specific instances in which ANN could be preferred over CNN and vice versa.
They are both unique in how they work mathematically, and this causes them to be better
at solving specific problems.
In general, CNN tends to be a more powerful and accurate way of solving classification
problems.
ANN is still dominant for problems where datasets are limited, and image inputs are not necessary.

Let's go through the steps of creating, training, and testing the model.

We used all the letters to train our model with 88800 images, we split the images of each letter

into training and testing. We use 88800 images to train and 14800 to test.

5.1) Preparing The Data :

The training dataset is explored to understand its structure and

```
[72]: #Read the CSV file (dataset)
      train = pd.read_csv("Datasets/emnist-letters-train.csv", header=None)
      test = pd.read_csv("Datasets/emnist-letters-test.csv", header=None)

[73]: train.head()
```

[73]:

	0	1	2	3	4	5	6	7	8	9	...	775	776	777	778	779	780	781	782	783	784
0	23	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	7	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	16	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	15	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	23	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 785 columns

content. The first few rows of the training dataset are displayed to provide an overview.

5.2) Data Preprocessing :

in here we handle the loading and preprocessing of the training and testing datasets for the OCR project:

First, each image in the training dataset is rotated 90 degrees and flipped vertically to enhance variability. Then images are reshaped to a standard 28x28 format, compatible with neural network models. Labels are extracted from the first column of each row.
Label values are adjusted by subtracting 1 so labels become in range (0,25) instead of (1,26). Then the processed images and labels are converted into NumPy arrays (x_train and y_train). And the same goes for the testing dataset.

[74]:
```
images = []
labels = []
for i in range(len(train)):
    image = np.flipud(np.rot90(train.iloc[i, 1:].to_numpy().reshape(28,28))) # iloc all Image Exclude the First
    images.append(image)
    label = train.iloc[i, 0] # First column only
    labels.append(label)
X_train = np.array(images)
y_train = np.array(labels)
y_train = y_train - 1
```

[ ]:
```
images = []
labels = []
for i in range(len(test)):
    image = np.flipud(np.rot90(test.iloc[i, 1:].to_numpy().reshape(28,28)))
    images.append(image)
    label = test.iloc[i, 0]
    labels.append(label)
X_test = np.array(images)
y_test = np.array(labels)
y_test = y_test - 1
```

5.3) Expanding Dimensions :

we added an extra dimension to the input data arrays, transforming them from a 2D shape (height x width) to a 3D shape (height x width x channels). For grayscale images, the channel dimension is set to 1. This adjustment ensures compatibility with neural network architectures, especially convolutional neural networks (CNNs), which typically expect input data in the shape (height, width, channels). This step is crucial for making the data compatible with frameworks like TensorFlow or Keras.

```
[ ]: print(X_train.shape)
     print(y_train.shape)

     print(X_test.shape)
     print(y_test.shape)
```

```
(88800, 28, 28)
(88800,)
(14800, 28, 28)
(14800,)
```

```
[ ]: X_train = np.expand_dims(X_train, axis=-1)
     X_test = np.expand_dims(X_test, axis=-1)
```

```
[ ]: print(X_train.shape)
     print(y_train.shape)

     print(X_test.shape)
     print(y_test.shape)
```

```
(88800, 28, 28, 1)
(88800,)
(14800, 28, 28, 1)
(14800,)
```

5.4) Building the model:

We use the Sequential model from Keras to create a sequential neural network, allowing neurons of each layer to be connected to the neurons of the next layer. Our architecture consists of multiple dense (fully connected) layers, each contributing to the model's ability to learn and recognize handwritten English letters.

1. Input Layer:

We start with an input layer that flattens the 28x28 images into a 1D array with a shape of (784,). This transformation is essential to convert the 2D image structure into a format suitable for neural network processing.

2. Dense Layers (2nd to 5th Layer):

We incorporate four dense layers, each designed to capture and learn different levels of features from the input data. The activation function used in these layers is ReLU (Rectified Linear Unit). ReLU outputs the input directly if it is positive, otherwise, it outputs zero. This allows the network to focus on pixels contributing to important features while discarding less relevant ones.

3. Output Layer:

The final layer is tailored for our specific classification task of recognizing handwritten English letters. It has an output shape of (26), corresponding to the number of classes (letters). The activation function in this layer is softmax, which predicts a multinomial probability distribution. The output of the softmax is a vector with probabilities assigned to each possible letter.

ANN(Artificial Neural Network) Model

```
[ ]: model = keras.models.Sequential([
        keras.layers.Flatten(input_shape=(28, 28)),
        keras.layers.Dense(1024, activation='relu'),
        keras.layers.Dense(512,  activation='relu'),
        keras.layers.Dense(256,  activation='relu'),
        keras.layers.Dense(128,  activation='relu'),
        keras.layers.Dense(64,   activation='relu'),
        keras.layers.Dense(26,   activation='softmax')
    ])
```

5.5) Optimization and training:

Here we optimized the neural network by implementing backpropagation optimization. Configuring with Adam optimizer, sparse categorical crossentropy loss, and accuracy metric. Implemented early stopping with patience set to 5, meaning that if the validation loss does not improve for 5 consecutive epochs, training will be stopped. Logging verbosity (verbose) Controls the

logging of messages. If set to 1, it will print a message when training is stopped due to early stopping.. Mode 'min' Specifies whether the monitored quantity should be minimized or maximized, In this case, it's set to 'min', indicating that training will stop when the quantity being monitored (validation loss) stops decreasing. Model checkpointing saves the best model for optimal weights in letter classification.

Then we trained the neural network using the model.fit function. This involved feeding the training data (X_train) and corresponding labels (y_train) into the model for 20 epochs, with each batch containing 50 samples.
The process was made transparent with progress bars (verbose=1), and 10% of the training data was set aside for validation to assess

```
[ ]: # Optimizing Algorithm for Backpropagation
optimizer_name = 'adam'
model.compile(
        optimizer=optimizer_name,
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy']
    )
early_stopping = EarlyStopping(monitor='val_loss', patience=5, verbose=1, mode='min')
mcp_save = ModelCheckpoint('ANN.model', save_best_only=True, monitor='val_loss', verbose=1, mode='auto')
```

performance during training (validation_split=0.1)

```
[ ]: history = model.fit(X_train,
                      y_train,
                      epochs=20,
                      batch_size=50,
                      verbose=1,
                      validation_split=0.1,
                      callbacks=[early_stopping, mcp_save])
```

5.6) Plots:

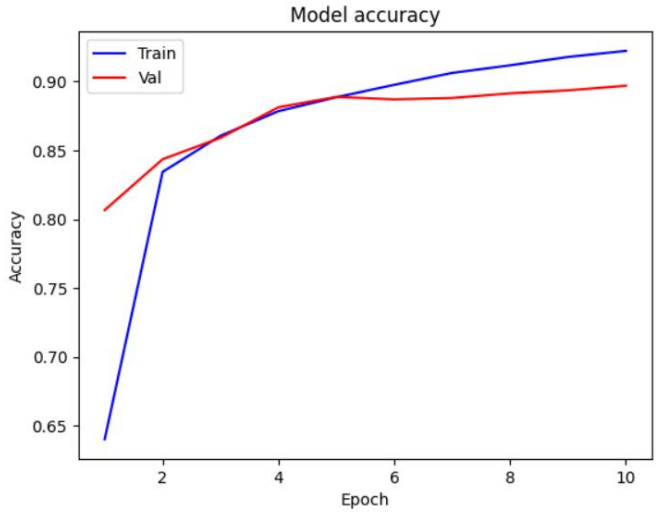
```
[ ]: def plotgraph(epochs, acc, val_acc):
    # Plot training & validation accuracy values
    plt.plot(epochs, acc, 'b')
    plt.plot(epochs, val_acc, 'r')
    plt.title('Model accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Val'], loc='upper left')
    plt.show()

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1,len(acc)+1)
```

We defined a function named plotgraph. This function takes as input the number of epochs, training accuracy, and validation accuracy. Its purpose is to visually display the progression of training and validation accuracy over epochs.

Subsequently, key metrics such as accuracy, validation accuracy, loss, and validation loss are extracted from the training history (history). These metrics will be employed to generate insightful plots illustrating the performance of the neural network during training.

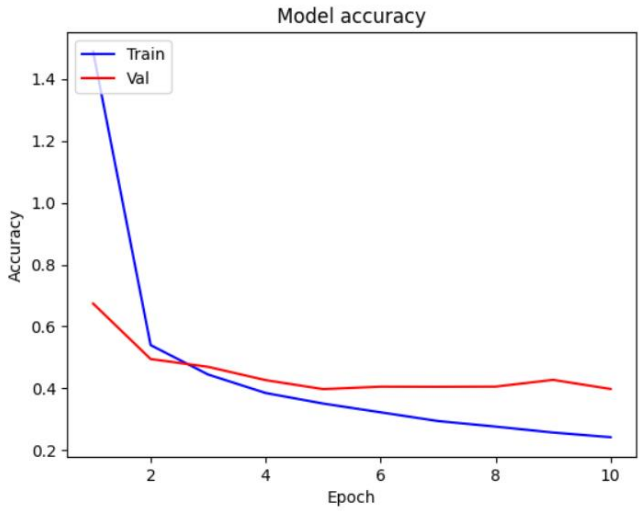
Now, let's proceed to the actual plots:

```
[ ]: plotgraph(epochs, acc, val_acc)
```

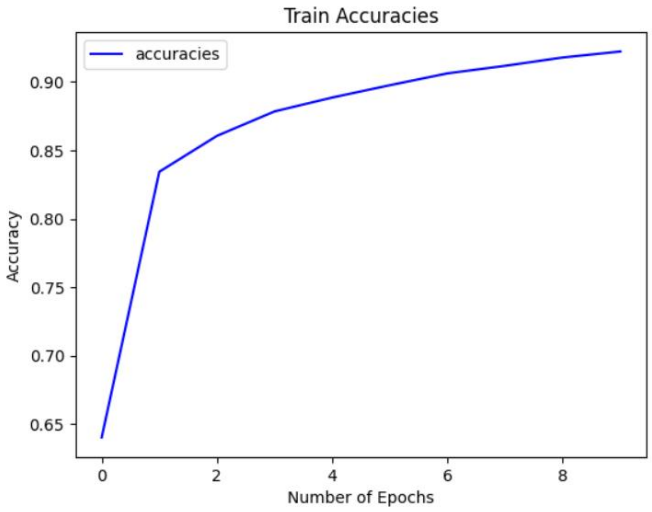


[ ]:
```
# loss curve
print('Loss Curve')
plotgraph(epochs, loss, val_loss)
```

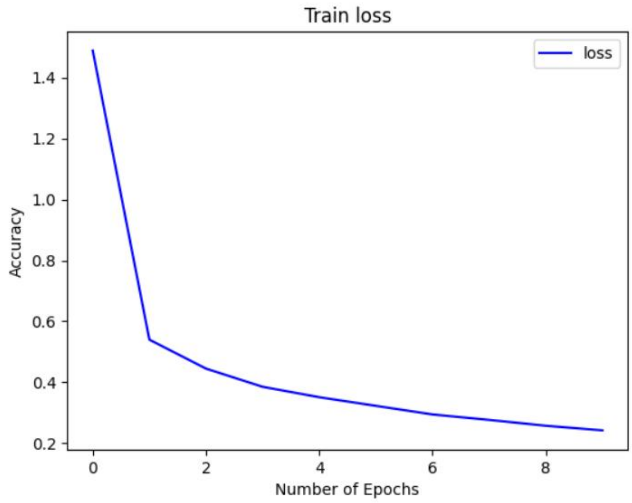
```
Loss Curve
```



[ ]:
```
plt.plot(history.history['accuracy'], color='b', label="accuracies")
plt.title("Train Accuracies")
plt.xlabel("Number of Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```

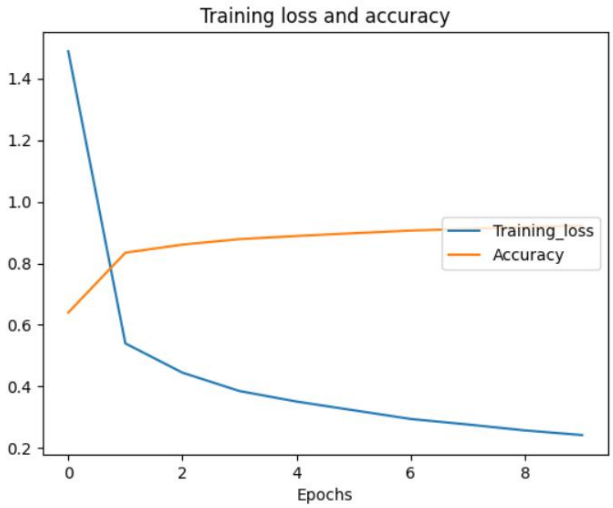


```
[ ]: plt.plot(history.history['loss'], color='b', label="loss")
     plt.title("Train loss")
     plt.xlabel("Number of Epochs")
     plt.ylabel("Accuracy")
     plt.legend()
     plt.show()
```




```
[ ]: N=len(history.epoch) #epoch
     plt.plot(np.arange(0,N),history.history['loss'],label='Training_loss')
     plt.plot(np.arange(0,N),history.history['accuracy'],label='Accuracy')
     plt.title('Training loss and accuracy')
     plt.xlabel('Epochs')
     plt.legend(loc='right')
```

```
[67]: <matplotlib.legend.Legend at 0x1e830796290>
```



5.7) Save and load model:

```
[ ]: # Save the model
     model.save('ANN.model')
```

```
[ ]: # Load best model
     model = load_model("ANN.model")
     model.summary()
```

We save the model after generating the weights and checkpoints, to use it at any time
without training from scratch. To load the model we load it into variable through
ANN_model.h5

5.8) Evaluation:

Advantages and disadvantages of epochs and early stopping :

Advantages:

Number of Epochs (Set to 20):
The decision to train the model for 20 epochs, enables a comprehensive exploration of the learning dynamics. This duration strikes a balance between thorough learning and computational efficiency, observed in accuracy trends.

Early Stopping (Patience Set to 5):
Prospective Overfitting Mitigation: The inclusion of early stopping with a patience of 5 in the code acts as a crucial defense against overfitting. By monitoring the validation loss, the training halts if further improvements are unlikely, ensuring a well-generalized model.

Optimal Model Preservation: Early stopping safeguards the model by preserving optimal weights. The ModelCheckpoint callback reinforces the retention of the best-performing model.

Disadvantages:

Number of Epochs:

While the chosen 20 epochs showcase effective learning, it's essential to acknowledge that training for excessively high numbers might incur computational costs without substantial gains.

Early Stopping:

While mitigating overfitting, an overly aggressive early stopping strategy might risk underfitting. Fine-tuning parameters like patience becomes crucial for striking the right balance.

Future Modifications:

Fine-Tuning Epoch Selection:

Consider exploring dynamic methods for determining the optimal number of epochs based on the unique characteristics of the OCR handwritten character recognition task.

Enhancing Early Stopping Strategy:

Investigate potential adjustments to early stopping parameters, specifically the patience value, to fine-tune its

impact on model convergence and overfitting prevention within the context of handwritten character recognition.

6) Development platforms:

We used google Kaggle and Jupyter notebook to write and execute arbitrary python code.

- **Numpy**: is used to convert the image into an array and to deal with it.

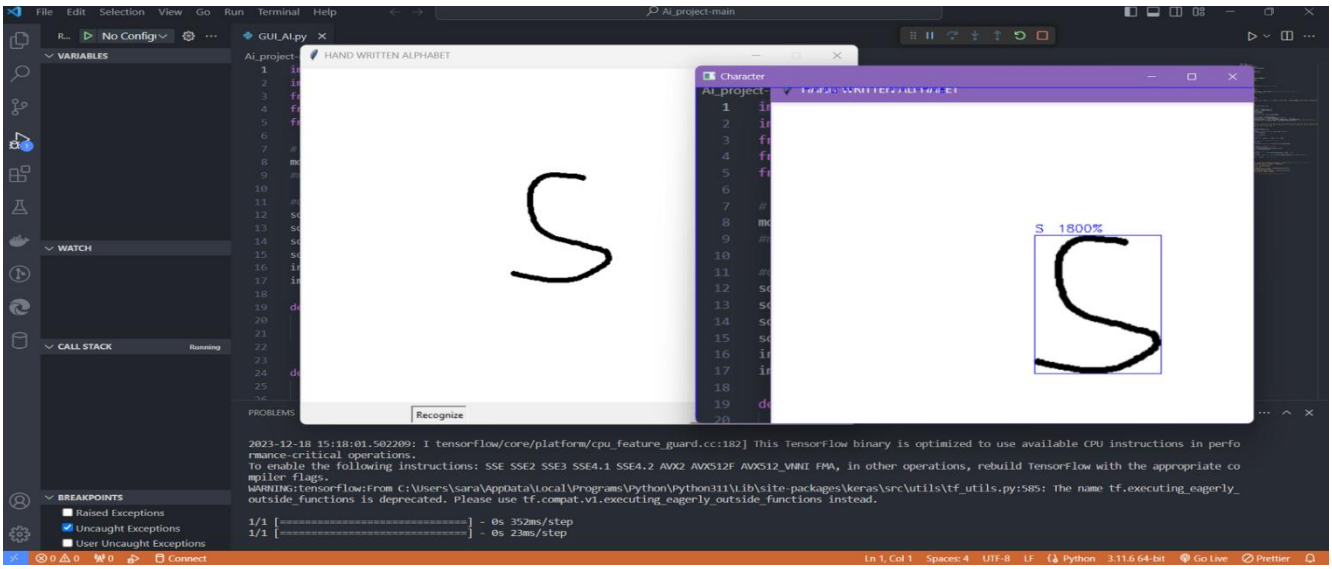
- **Sklearn**: we used it for splitting the dataset.

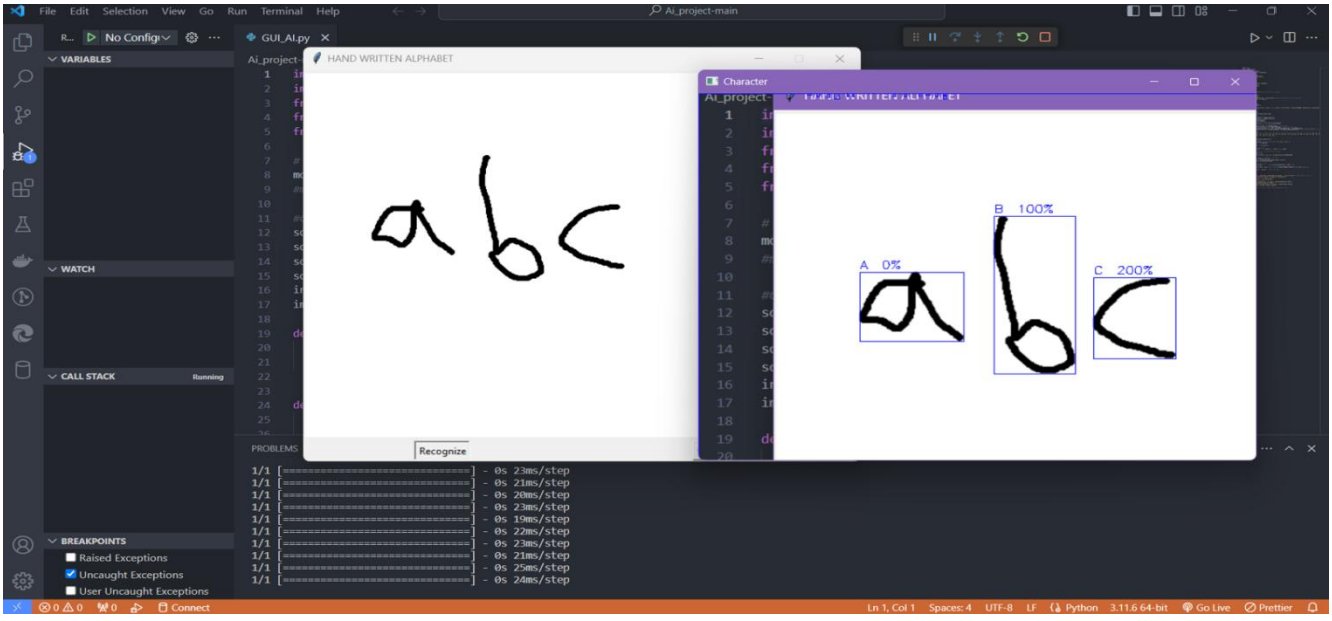
- **Keras**: deep learning API written in Python, running on top of the machine learning platform
- **TensorFlow**: TensorFlow is the open-source library for several various tasks in machine learning.
- **Matplotlib**: Matplotlib is a comprehensive library for creating static and interactive visualizations in **Python to virtualize data.**
- **Tkinter**: Tkinter is the standard GUI library for Python.
- **Pandas**: we used this library to read the csv file.

In [129]:

```
#import instructions
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import tensorflow as tf
from keras.models import Sequential, load_model
from keras.layers import Dense
from keras.callbacks import EarlyStopping, ModelCheckpoint
from sklearn.model_selection import train_test_spli
from sklearn.metrics import classification_report
from sklearn.metrics import roc_curve
from sklearn.metrics import auc
```

GUI Samples :


HAND WRITTEN ALPHABET
Recognize
Character
S 1800%
2023-12-18 15:18:01.502209: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in perfo
rmance-critical operations.
To enable the following instructions: SSE SSE2 SSE3 SSE4.1 SSE4.2 AVX2 AVX512F AVX512_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate co
mpiler flags.
WARNING:tensorflow:From C:\Users\sara\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\utils\tf_utils.py:585: The name tf.executing_eagerly_
outside_functions is deprecated. Please use tf.compat.v1.executing_eagerly_outside_functions instead.
1/1 [==============================] - 0s 352ms/step
1/1 [==============================] - 0s 23ms/step


HAND WRITTEN ALPHABET
Recognize
Character
A 0%
B 100%
C 200%
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step