TAMILZH-AI

A08 DIGITALISATION OF TAMIL HANDWRITTEN TEXT

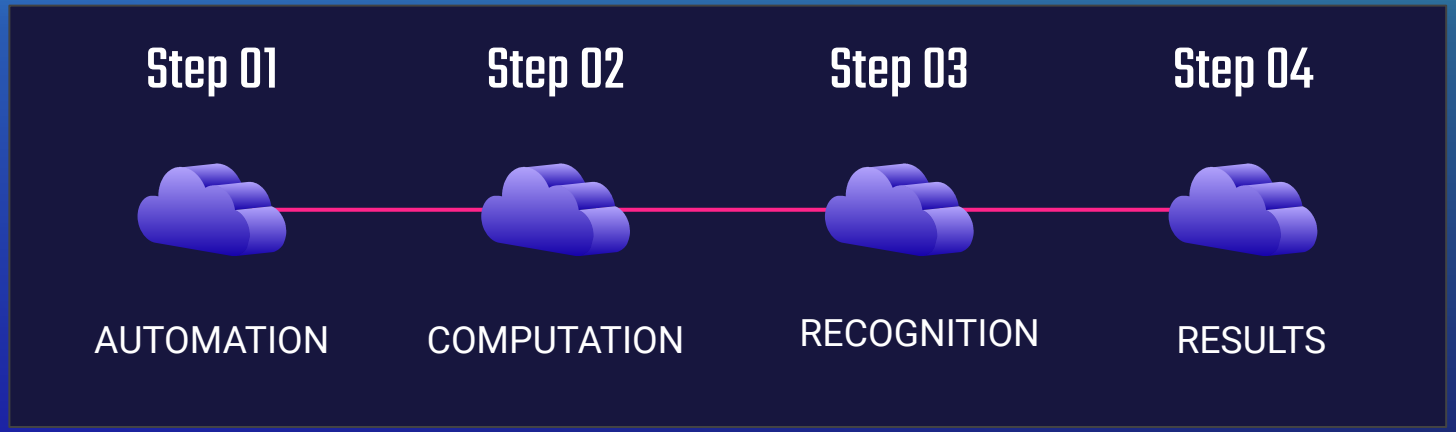
A PROJECT BY: AzureRs

S. SNEKHA & L.N.PRIYANKA BHARATHI
St. Joseph's College of Engineering,
Chennai

APPROACH OF THE PROJECT:

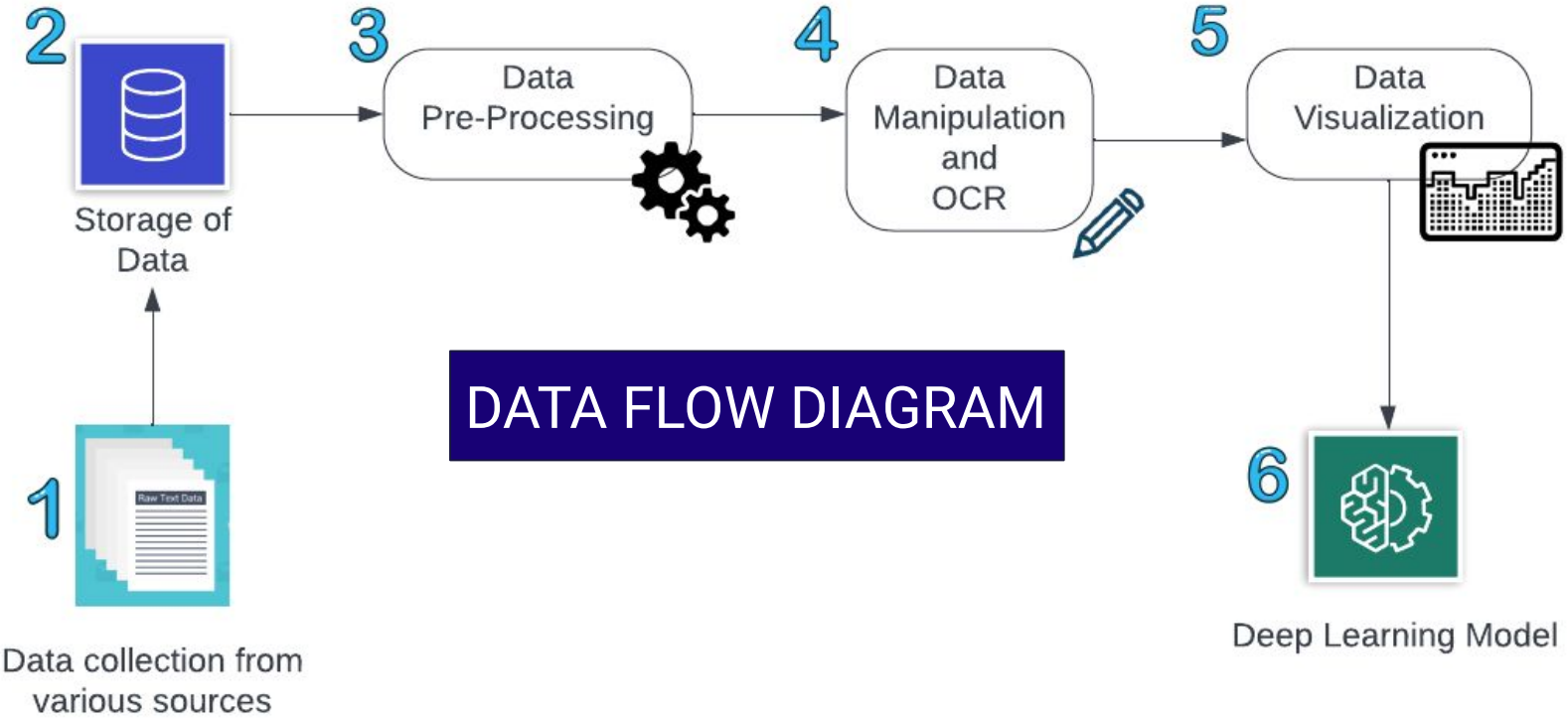
- Recognize **handwritten Tamil Characters**.
- Handwritten Tamil character into printed Tamil character.
- Difficult-great variations
- Writing **styles, different size and orientation angle** of the characters.
- The uploaded image is **preprocessed**
- Dimensions reduction is done to achieve **maximum accuracy**
- **OCR - OPTICAL CHARACTER RECOGNITION**
- Trained deep learning model -**CNN** Algorithm
- The prediction is saved in a **pdf**.

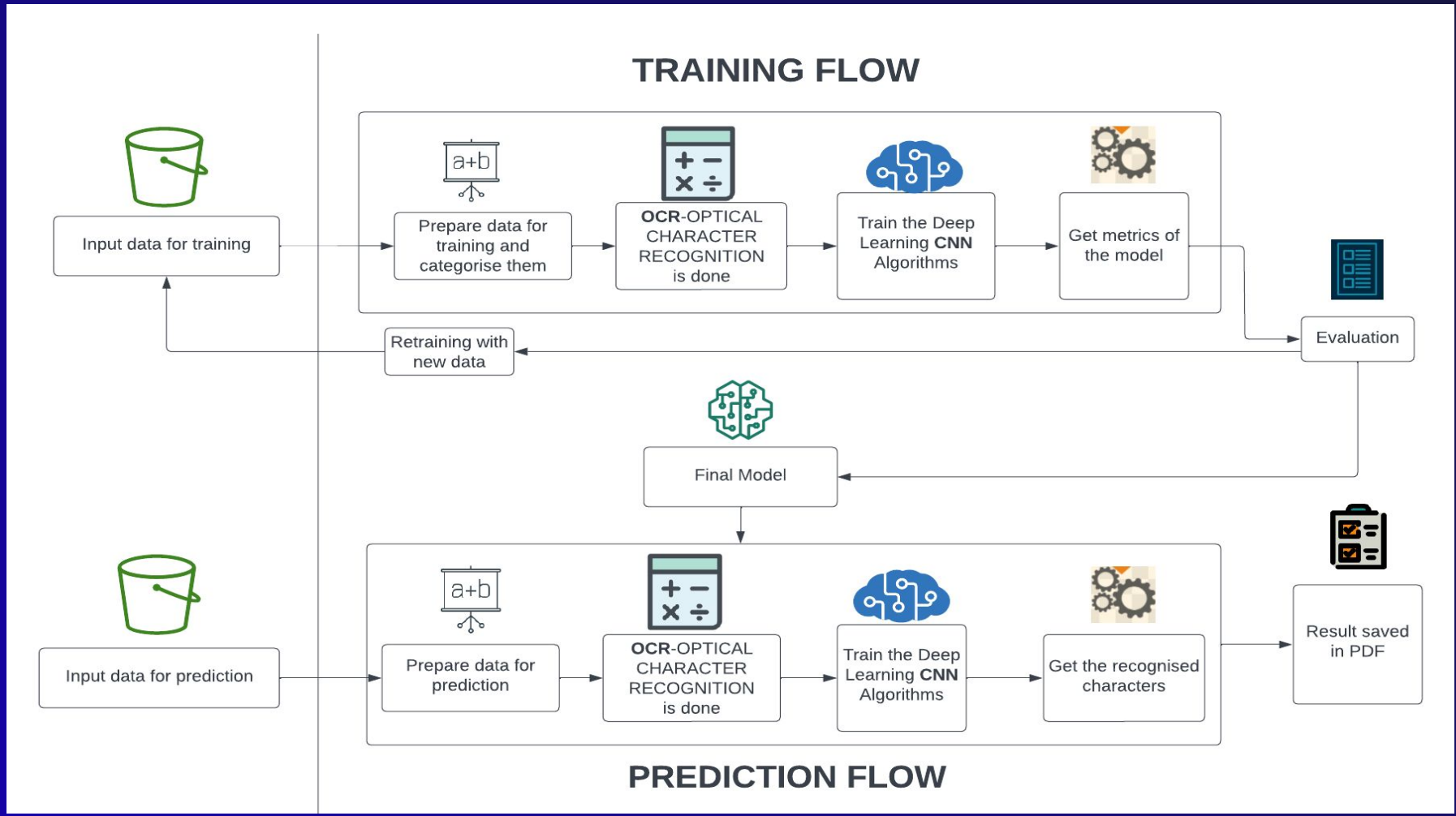
AIM OF THE PROJECT

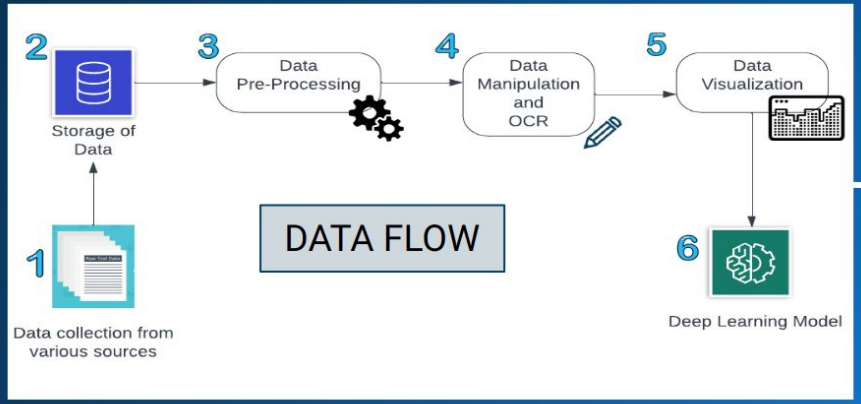


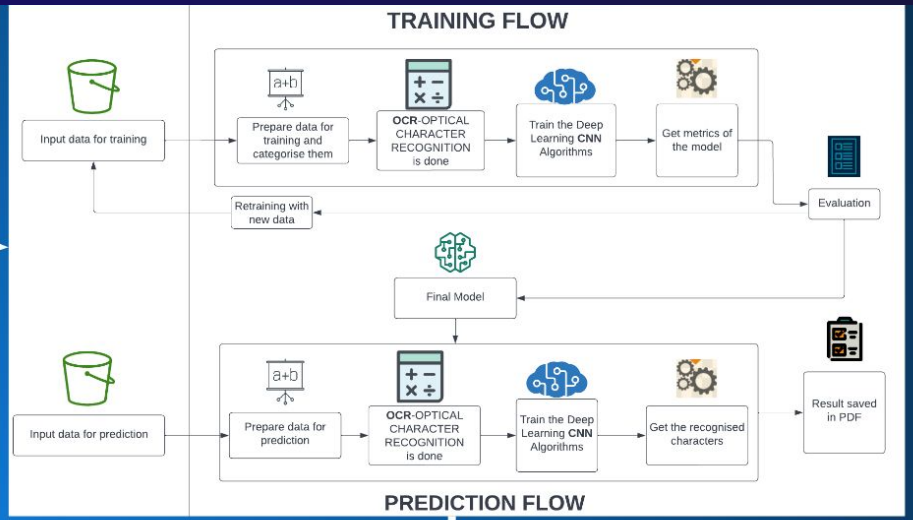
PREDICTION OF TAMIL CHARACTERS FROM IMAGES

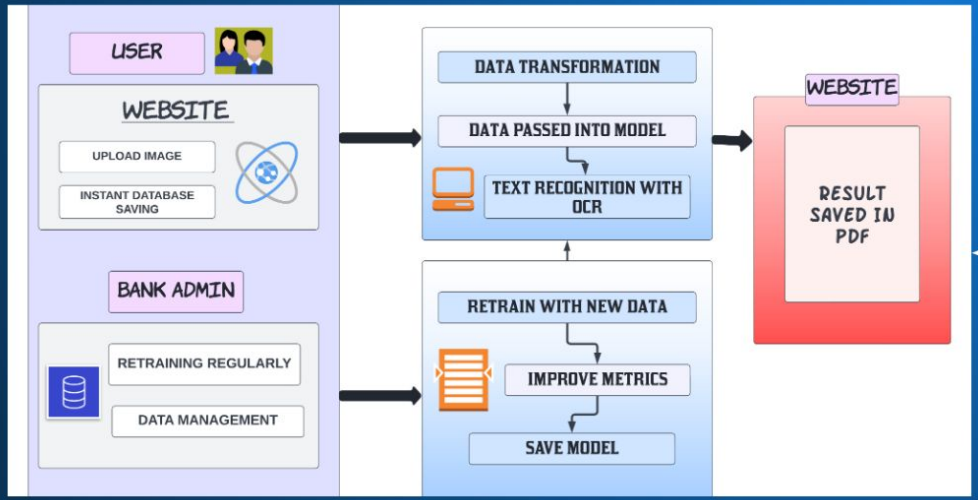
DATA FLOW DIAGRAM




TRAINING FLOW
Input data for training
a+b
Prepare data for training and categorise them
OCR-OPTICAL CHARACTER RECOGNITION is done
Train the Deep Learning CNN Algorithms
Get metrics of the model
Evaluation
Retraining with new data
Final Model
Input data for prediction
a+b
Prepare data for prediction
OCR-OPTICAL CHARACTER RECOGNITION is done
Train the Deep Learning CNN Algorithms
Get the recognised characters
Result saved in PDF
PREDICTION FLOW

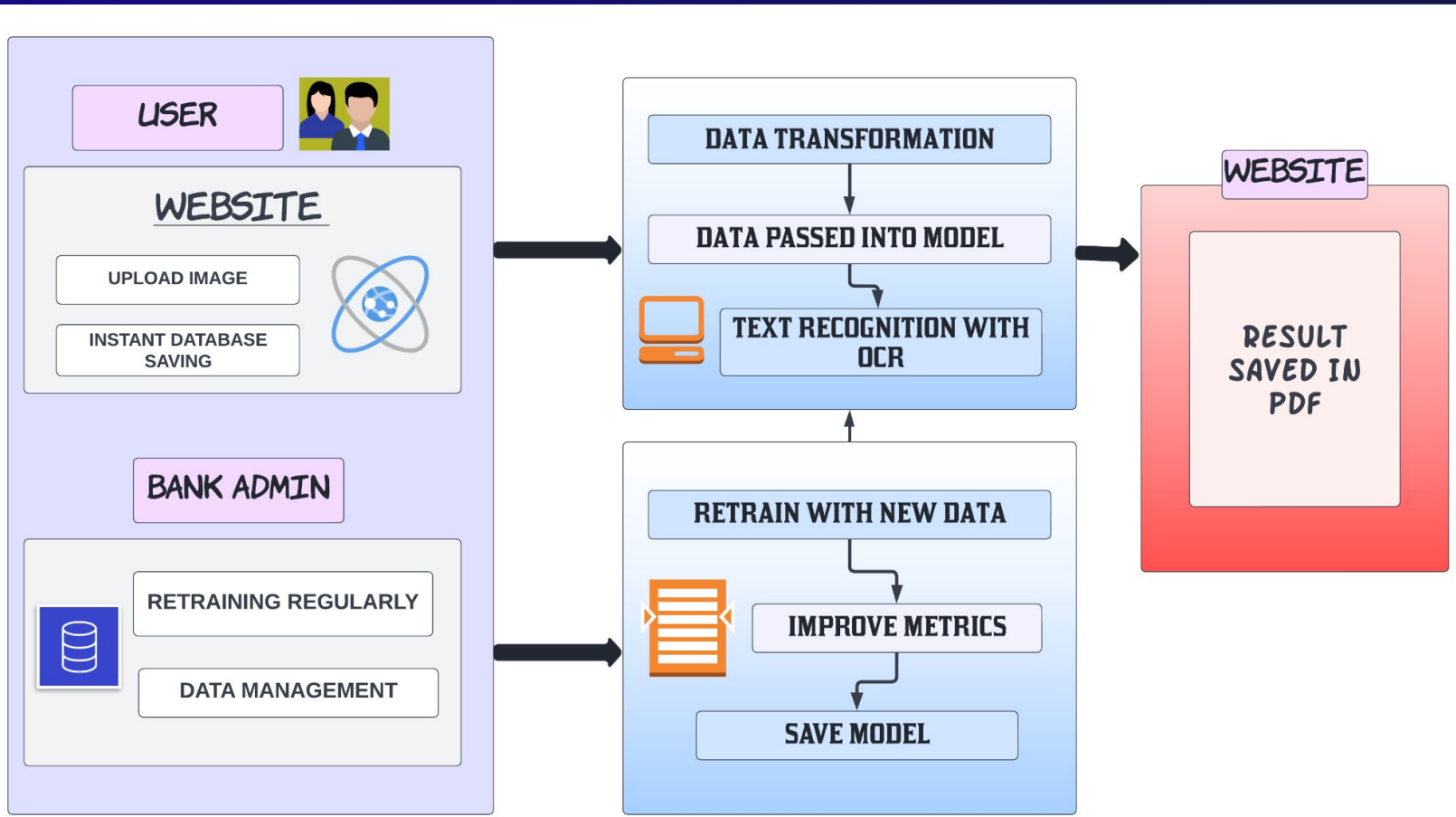




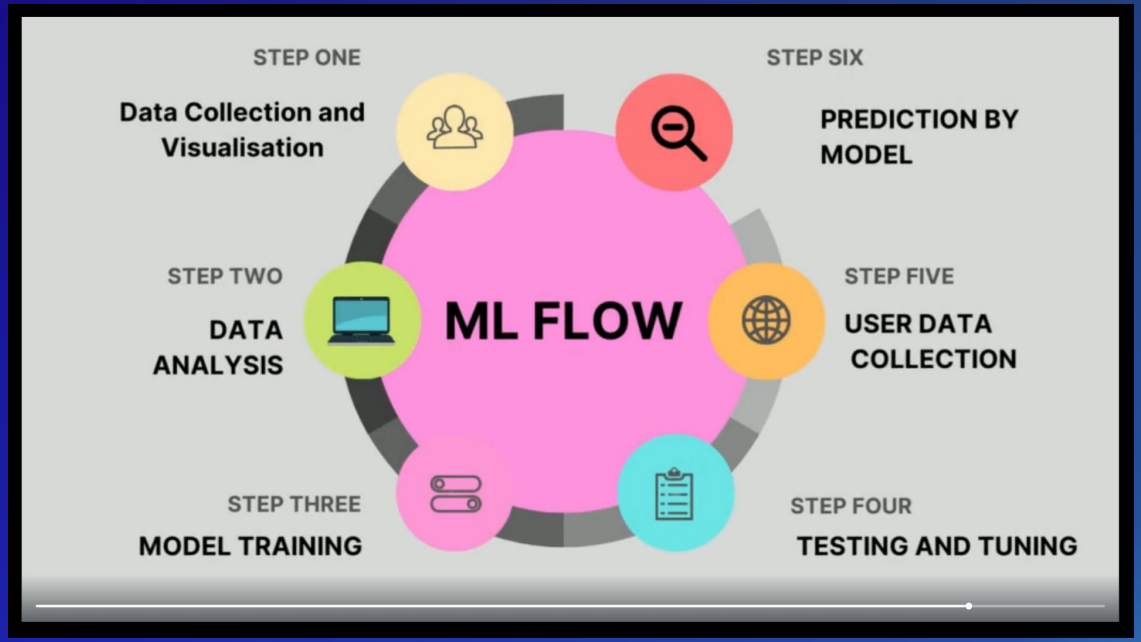


ARCHITECTURE DIAGRAM

WORKFLOW AND APPROACH


USER
WEBSITE
UPLOAD IMAGE
INSTANT DATABASE SAVING
BANK ADMIN
RETRAINING REGULARLY
DATA MANAGEMENT
DATA TRANSFORMATION
DATA PASSED INTO MODEL
TEXT RECOGNITION WITH OCR
RETRAIN WITH NEW DATA
IMPROVE METRICS
SAVE MODEL
WEBSITE
RESULT SAVED IN PDF

ML IMPLEMENTATION



- **Dataset source:**
  GOOGLE
- **Model Analysis:**
  Images were recognised using OCR
  Applied **CNN ALGORITHM**
  Model trained with the obtained dataset

Fig: Features of the project

Input shape	Layer	Output shape
128x128	Convolution Layer- 64C5x5	124x124x64
124x124x64	MaxPooling Layer - P2x2	62x62x64
62x62x64	Convolution Layer - 32C5x5	58x58x32
58x58x32	MaxPooling Layer - P2x2	29x29x32
29x29x32	Convolution Layer - 32C5x5	25x25x32
25x25x32	MaxPooling Layer - P2x2	12x12x32
12x12x32	Convolution Layer - 32C5x5	8x8x32
8x8x32	MaxPooling Layer - P2x2	4x4x32
4x4x32	Flatten Layer	1x512
1x512	Hidden Layer	1x300
1x300	Output Layer	1X156

ARCHITECTURE

- Input Images taken from the dataset, reshape. The same images used and of size 128x128x1.
- Conv-1 The first convolutional layer consists of 64 kernels of size 5x5 applied with a stride of 1 and padding of 0.
- MaxPool-1 The max-pool layer following Conv-2 consists of pooling size of 2x2 and a stride of
- Conv-2 The second convolution layer consists of 32 kernels of size 5x5 applied with a stride of 1 and padding of 0.
- MaxPool-2 The max-pool layer following Conv-2 consists of pooling size of 2x2 and a stride of
- Conv-3 The third conv layer consists of 32 kernels of size 5x5 applied with a stride of 1 and padding of 0.
- MaxPool-3 The max-pool layer following Conv-3 consists of pooling size of 2x2 and a stride of
- Conv-4 The fourth conv layer consists of 32 kernels of size 5x5 applied with a stride of 1 and padding of 0.
- MaxPool-4 The maxpool layer following Conv-4 consists of pooling size of 2x2 and a stride of 0.
- Flattening Layer The output of CNN is flattened to get 1x512 output.
- FC-1 (Dense Layer 1) The flattened output is fed to a hidden layer of 300 neurons.
- Output (Dense Layer 2) Finally, the output of hidden layer is fed to the output layer of 156 to get the final output.

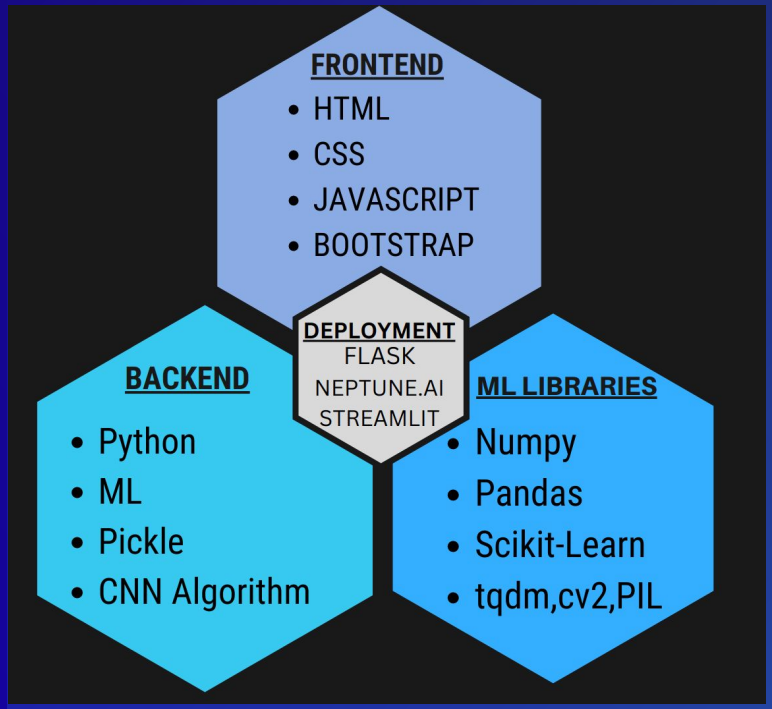


Fig. TECH-STACK

WHY CNN APPROACH?

- High **Accuracy**
- Efficient **Memory** utilisation
- Compares the image piece by piece
- Automatic feature extractors from the image
- Less **error** chances

In [8]:

```
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=20, batch_size=100, verbose=1)
TrainAccuracy = model.evaluate(X_train, y_train, verbose=1)
TestAccuracy = model.evaluate(X_test, y_test, verbose=1)
```

```
Epoch 1/20
556/556 [==============================] - 90s 78ms/step - loss: 3.6002 - accuracy: 0.2181 - val_loss: 0.8129 - val_accuracy: 0.7557
Epoch 2/20
556/556 [==============================] - 42s 75ms/step - loss: 0.8371 - accuracy: 0.7439 - val_loss: 0.5194 - val_accuracy: 0.8397
Epoch 3/20
556/556 [==============================] - 42s 75ms/step - loss: 0.5595 - accuracy: 0.8212 - val_loss: 0.4010 - val_accuracy: 0.8745
Epoch 4/20
556/556 [==============================] - 42s 75ms/step - loss: 0.4270 - accuracy: 0.8597 - val_loss: 0.3542 - val_accuracy: 0.8879
Epoch 5/20
556/556 [==============================] - 42s 75ms/step - loss: 0.3700 - accuracy: 0.8781 - val_loss: 0.3524 - val_accuracy: 0.8862
Epoch 6/20
556/556 [==============================] - 46s 83ms/step - loss: 0.3295 - accuracy: 0.8888 - val_loss: 0.3114 - val_accuracy: 0.9050
Epoch 7/20
556/556 [==============================] - 46s 83ms/step - loss: 0.2922 - accuracy: 0.8996 - val_loss: 0.3035 - val_accuracy: 0.9068
Epoch 8/20
556/556 [==============================] - 42s 75ms/step - loss: 0.2628 - accuracy: 0.9105 - val_loss: 0.3067 - val_accuracy: 0.9013
Epoch 9/20
556/556 [==============================] - 42s 75ms/step - loss: 0.2397 - accuracy: 0.9190 - val_loss: 0.2822 - val_accuracy: 0.9113
Epoch 10/20
556/556 [==============================] - 42s 75ms/step - loss: 0.2268 - accuracy: 0.9217 - val_loss: 0.2653 - val_accuracy: 0.9176
Epoch 11/20
556/556 [==============================] - 42s 75ms/step - loss: 0.2115 - accuracy: 0.9253 - val_loss: 0.2644 - val_accuracy: 0.9188
Epoch 12/20
556/556 [==============================] - 41s 75ms/step - loss: 0.1968 - accuracy: 0.9317 - val_loss: 0.2740 - val_accuracy: 0.9131
Epoch 13/20
556/556 [==============================] - 46s 83ms/step - loss: 0.1971 - accuracy: 0.9295 - val_loss: 0.2574 - val_accuracy: 0.9225
Epoch 14/20
556/556 [==============================] - 46s 83ms/step - loss: 0.1807 - accuracy: 0.9369 - val_loss: 0.2624 - val_accuracy: 0.9218
Epoch 15/20
556/556 [==============================] - 42s 75ms/step - loss: 0.1713 - accuracy: 0.9400 - val_loss: 0.2707 - val_accuracy: 0.9178
Epoch 16/20
556/556 [==============================] - 42s 75ms/step - loss: 0.1681 - accuracy: 0.9398 - val_loss: 0.2859 - val_accuracy: 0.9165
Epoch 17/20
556/556 [==============================] - 41s 75ms/step - loss: 0.1677 - accuracy: 0.9398 - val_loss: 0.2646 - val_accuracy: 0.9227
Epoch 18/20
556/556 [==============================] - 42s 75ms/step - loss: 0.1549 - accuracy: 0.9444 - val_loss: 0.2522 - val_accuracy: 0.9243
Epoch 19/20
556/556 [==============================] - 42s 75ms/step - loss: 0.1456 - accuracy: 0.9486 - val_loss: 0.2587 - val_accuracy: 0.9239
Epoch 20/20
556/556 [==============================] - 41s 75ms/step - loss: 0.1487 - accuracy: 0.9477 - val_loss: 0.2729 - val_accuracy: 0.9229
1736/1736 [==============================] - 16s 9ms/step - loss: 0.0501 - accuracy: 0.9830
856/856 [==============================] - 8s 9ms/step - loss: 0.2729 - accuracy: 0.9229
```

Training accuracy - 98.3 %
Test accuracy - 92.29%

CONCLUSION

THUS MANY MANUSCRIPTS AND TAMIL DOCUMENTS THAT HASN'T BEEN DIGITALISED CAN BE CONVERTED AUTOMATICALLY THROUGH THIS SIMPLE MODEL THUS ANSWERING THE PROBLEM STATEMENT

THANK YOU!