BTech Final Project

Handwriting Analysis

Gourav Kumar Shaw (2020CSB010)

Soumyadeep Sinha (2020CSB044)

Mentor: Dr. Samit Biswas

Objective

- **Author recognition** from the **handwritten** text.
- Using **Convolutional Neural Network** and **train** using a **softmax classification** loss function .
- **Instead of traditional** way to establish **features** like **curvature** of **letters**, **spacing** between letters. And feed them into a strong **classifier** like **SVM** to distinguish between the **writers**.

Solution

Data Gathering

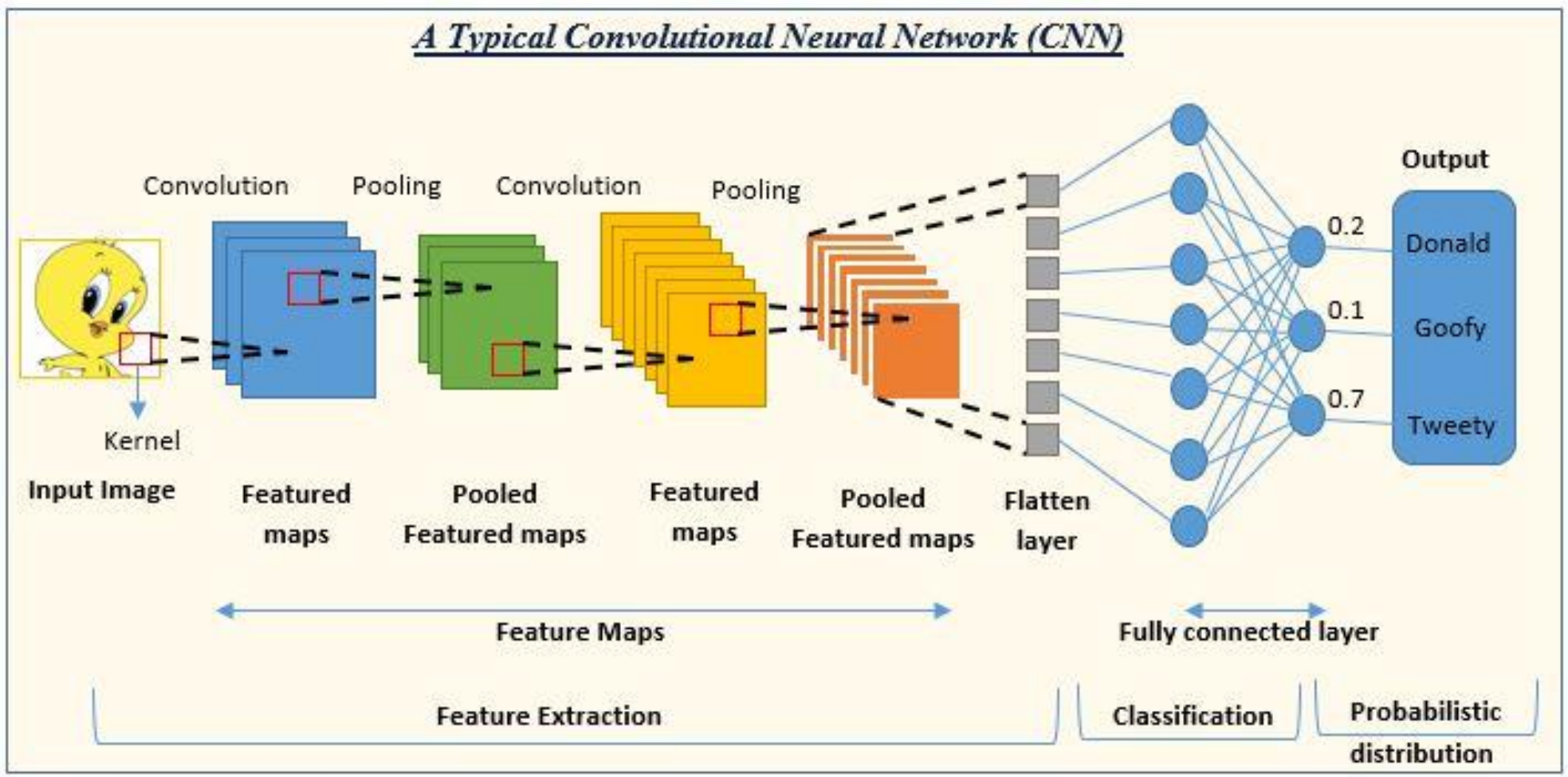
- The database used contains 1539 pages of scanned text sentences written by 600+ writers.

- This project uses the top 50 writers with most amount of data. Data is grouped by writers having written a collection of sentences

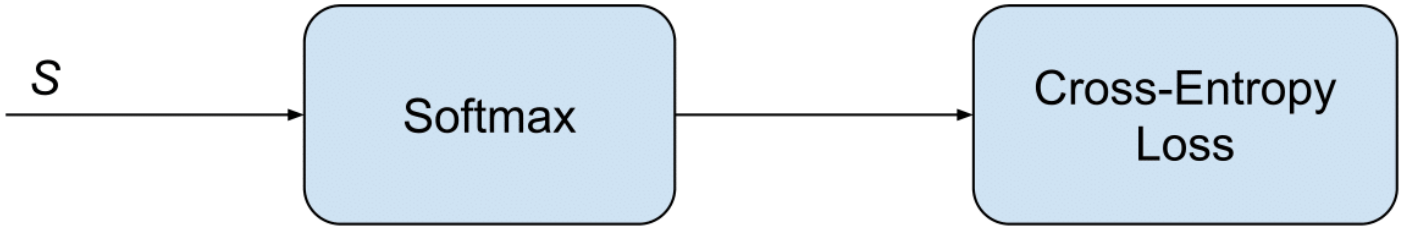
Preprocessing

- For our **CNN** to understand the writing style, language is not a restriction, so we pass patches of text having image size **113x113** from each sentence.
- We **don't break** them **w.r.t. sentences** or **words**, but we **break** them down into **smaller image sets**.
- For serving the purpose, a **generator function** is implemented to **scan through each sentence** and **generate random** patches with same patch size.
- **CNN** doesn’t even need to take the full data, so we’ve limited the number of patches to be **30% of the total patches.**
- **Data-set** is shuffled.

Convolutional Neural Network



Softmax Function



$$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}} \qquad CE = -\sum_i^C t_i log(f(s)_i)$$

Here, si: The *i*-th element of the input vector.
e: Euler's number (approximately 2.71828).

t: is the actual label (0 or 1 in binary classification, a one-hot vector in multi-class classification).

f(s)i: predicted probability

- **SoftMax function** is a mathematical function that **takes a vector** of real numbers as input and **transforms** it into a **probability distribution**.
- **Used** in machine learning, particularly in **multiclass classification** problems
- **Cross-Entropy Loss** is used for **classification tasks**.

Self-designed CNN Model

- We’ve used **Keras** with **TensorFlow** backend.
- A standard **CNN Model** is designed with **multiple convolution** and **maxpool layers**, a **few dense layers** and a **final output layer** is the **softmax activation**.
- **ReLU activation** was also used **between** the **convolution and dense layers**.
- The resultant model was **optimized using Adam Optimizer**.

Design of the model

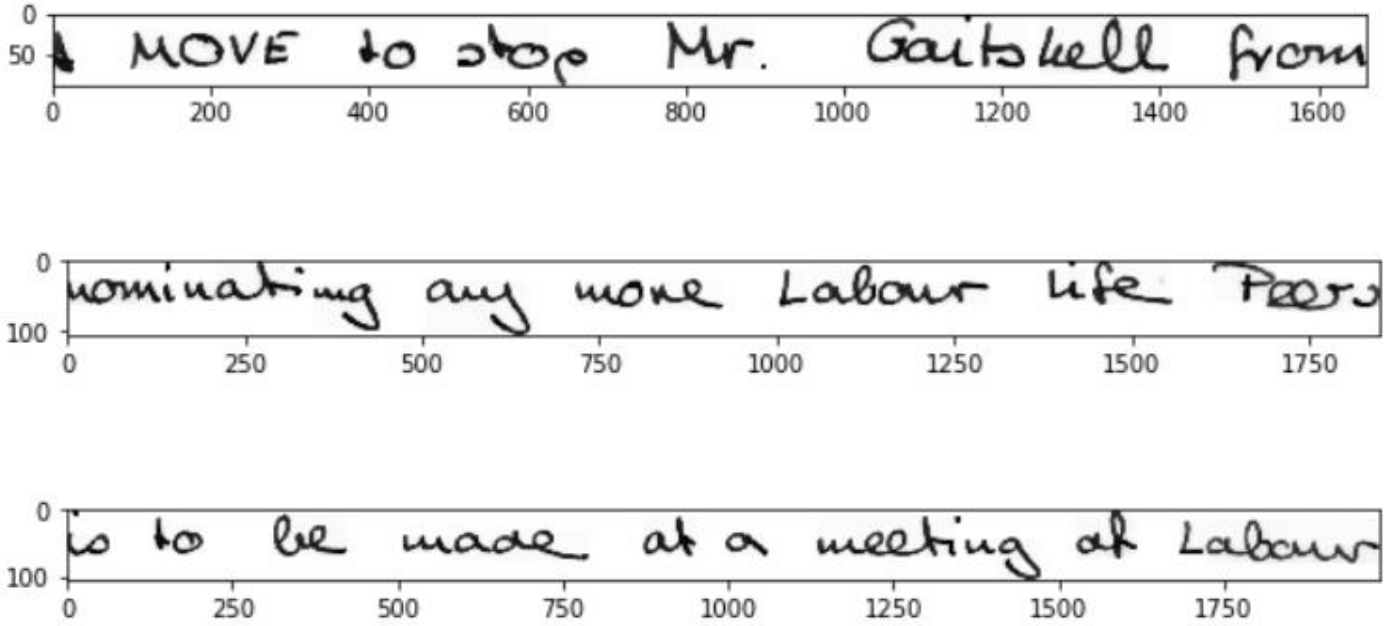
Following is the design of the model:

Layer (type)	Shape	Params
zero_padding2d_2 (Zero Padding)	(None, 115, 115, 1)	0
lambda_2 (Lambda)	(None, 56, 56, 1)	0
conv1 (Conv2D)	(None, 28, 28, 32)	832
activation_7 (Activation)	(None, 28, 28, 32)	0
pool1 (MaxPooling2D)	(None, 14, 14, 32)	0
conv2 (Conv2D)	(None, 14, 14, 64)	18496
activation_8 (Activation)	(None, 14, 14, 64)	0
pool2 (MaxPooling2D)	(None, 7, 7, 64)	0
conv3 (Conv2D)	(None, 7, 7, 128)	73856
activation_9 (Activation)	(None, 7, 7, 128)	0
pool3 (MaxPooling2D)	(None, 3, 3, 128)	0
flatten_2 (Flatten)	(None, 1152)	0
dropout_4 (Dropout)	(None, 1152)	0
dense1 (Dense)	(None, 512)	590336
activation_10 (Activation)	(None, 512)	0
dropout_5 (Dropout)	(None, 512)	0
dense2 (Dense)	(None, 256)	131328
activation_11 (Activation)	(None, 256)	0
dropout_6 (Dropout)	(None, 256)	0
output (Dense)	(None, 50)	12850
activation_12 (Activation)	(None, 50)	0

Visualize the image data.

In [4]:

```
for filename in img_files[:3]:
    img=mpimg.imread(filename)
    plt.figure(figsize=(10,10))
    plt.imshow(img, cmap ='gray')
```



Taking 8 epoch

```
Epoch 1/8
409/408 [==============================] - 870s 2s/step - loss: 3.4767 - acc: 0.1948 - val_loss:
2.9207 - val_acc: 0.2410

Epoch 00001: saving model to low_loss.hdf5
Epoch 2/8
409/408 [==============================] - 645s 2s/step - loss: 2.7719 - acc: 0.2515 - val_loss:
2.3124 - val_acc: 0.3520

Epoch 00002: saving model to low_loss.hdf5
Epoch 3/8
409/408 [==============================] - 646s 2s/step - loss: 2.3043 - acc: 0.3374 - val_loss:
1.8966 - val_acc: 0.4336

Epoch 00003: saving model to low_loss.hdf5
Epoch 4/8
409/408 [==============================] - 653s 2s/step - loss: 1.9469 - acc: 0.4199 - val_loss:
1.5183 - val_acc: 0.5308

Epoch 00004: saving model to low_loss.hdf5
Epoch 5/8
409/408 [==============================] - 651s 2s/step - loss: 1.7120 - acc: 0.4794 - val_loss:
1.3303 - val_acc: 0.5825

Epoch 00005: saving model to low_loss.hdf5
Epoch 6/8
409/408 [==============================] - 631s 2s/step - loss: 1.5410 - acc: 0.5280 - val_loss:
1.3000 - val_acc: 0.5906

Epoch 00006: saving model to low_loss.hdf5
Epoch 7/8
409/408 [==============================] - 609s 1s/step - loss: 1.4084 - acc: 0.5678 - val_loss:
1.1805 - val_acc: 0.6291

Epoch 00007: saving model to low_loss.hdf5
Epoch 8/8
409/408 [==============================] - 615s 2s/step - loss: 1.3068 - acc: 0.5991 - val_loss:
1.0342 - val_acc: 0.6785
```

Accuracy of the Model

Test model performance on the Test Set

1. Accuracy on test set
2. Samples predicted to be from the same writer

In [21]:

```
# Load save model and use for prediction on test set
model.load_weights('low_loss.hdf5')
scores = model.evaluate_generator(test_generator,842)
print("Accuracy = ", scores[1])
```

```
('Accuracy = ', 0.94013787749041677)
```

Link to Code: https://www.kaggle.com/code/gkshaw/handwriting-recognition-cnn/edit

Future Work

More Languages Support

- **Currently** we test this model with only **English dataset**. Will test this model with **regional languages** like **Bengali** and **Hindi**.

Accuracy

- Will try to **improve** the **accuracy** by applying other **advanced techniques** like **LSTM**.

References

https://towardsdatascience.com/handwriting-recognition-using-tensorflow-and-keras-819b36148fe5

https://www.tensorflow.org/api_docs

https://keras.io/

https://www.linkedin.com/pulse/handwritten-text-recognition-using-deep-learning-cnn-rnn-dikhit/

Thank You!

Feel free to ask any further questions. ☺