

INTERNSHIP: PROJECT REPORT

Name of the Student	SAUBHAGYAJEET SAHOO
Internship Project Title	AUTOMATE IDENTIFICATION AND RECOGNITION OF HANDWRITTEN TEXT FROM AN IMAGE
Name of the Company	TCS iON
Name of the Industry Mentor	ANAMIKA CHATTERJEE
Name of the Institute	DHANESWAR RATH INSTITUTE OF ENGINEERING AND MANAGEMENT STUDIES (DRIEMS), TANGI, CUTTACK, ODISHA

Start Date	End Date	Total Effort (hrs.)	Project Environment	Tools used
01/06/2020	01/08/2020	210	PYTHON- 3.6, GOOGLE COLAB – Jupyter Notebook, Anaconda Navigator- Jupyter Notebook.	IAM Dataset, TensorFlow 2.2.0, Keras-2.3.0, Google Colab Virtual GPU, numpy-1.18.5.

Project Synopsis:

- This is TCS iON-RIO 210-Industry Project. This project is basically focused on identification and recognition of handwritten text from an image. It is based on enhancement of optical character recognition system. Optical Character Recognition (OCR) or Optical Character Reader is the electronic or mechanical conversion of images of handwritten or printed text into machine-encoded text, whether from a scanned document, a photo of a document, a scene-photo or from subtitle text superimposed on an image. An optical character recognition problem is basically a type of image-based sequence recognition problem. For an image-based problem most suited are convolution neural networks (CNN) while for sequence recognition problem, most suited neural networks are recurrent neural networks (RNN). To cope up with the OCR problems we need to combine both of this CNN and RNN. The purpose of OCR in text identification and recognition from an image is to extract handwritten data to digital data. So, one can easily handle this digital data by editing, adding new information in that text. OCR is developing field of Computer Vision, Pattern Recognition and Artificial Intelligence. The main objective of this project is to develop machine learning algorithm in order to enable entity and knowledge extraction from documents with handwritten annotations, with an aim to identify handwritten words on an image.

Solution Approach:

The solution provided in this project is highly inspired from the method that is described in the paper [DeepWriter: A Multi-Stream Deep CNN for Text-independent Writer Identification](#). The main aim of the paper was to use a deep multi-stream CNN in order to predict hand written text. The images used in this project are the scanned images of various writers that are provided in the IAM Database.

The steps that are taken for the solution of the project are

- 1) Providing the samples of the Image and form that are required for making the database.
- 2) Creating a dictionary to store form ID and its writer.
- 3) Selecting the writers ID which will be used for making the dataset.
- 4) Selecting the form that are associated with the selected writers ID.
- 5) Creating a temporary directory which will store the writer's image from the selected writer ID.
- 6) Creating arrays of the input file and visualize it.
- 7) Encoding writer's ID.
- 8) Splitting the dataset into train, validation and tests sets.
- 9) Defining constants that will be used throughout the project.
- 10) Cropping the Inputted Data and converting it into array and visualizing sample from the training set.
- 11) Creating a generating method that will used to call the trainset while training.
- 12) Building a network architecture in Keras using CNN.
- 13) Training the model.
- 14) Checking the accuracy of the trained model.

The following steps are explained briefly below:

1) Providing the samples of the image and form that are required for the making of the database

Step 1 - Download the data from the IAM Database "<http://www.fki.inf.unibe.ch/databases/iam-handwriting-database>"

Download the IAM Handwriting Database

Structure

The IAM Handwriting Database is hierarchically structured into forms (The name of the files correspond to the naming scheme of the LOB Corpus [5]):

- data
 - data/ascii - Contains summarized meta information in ascii format.
 - data/forms - Contains form images (example: a01-122.png).
 - data/lines - Contains text lines (example: a01/a01-122/a01-122-02.png).
 - data/sentences - Contains sentences, one for each line (example: a01/a01-122/a01-122-s01-02.png).
 - data/words - Contains words (example: a01/a01-122/a01-122-s01-02.png).
 - data/xml - Contains the meta-information in XML format (example: a01-122.xml).







You can look at the images in each directory or download a compressed archive in each top level directory.

Terms of usage

The IAM Handwriting Database is publicly accessible and freely available for non-commercial research purposes. If you are using data from the IAM Handwriting Database, we request you to register, so we are aware of who is using our data. If you are publishing scientific work based on the IAM Handwriting Database, we request you to include a reference to the paper

Step 2 - Download the text file of the data you have downloaded from the ascii folder.

Index of /DBs/iamDB/data/ascii

Name	Last modified	Size	Description
 Parent Directory	-		
 ascii.tgz	2004-10-21 14:26	2.5M	
 forms.txt	2004-03-02 14:49	44K	
 lines.txt	2004-03-02 14:49	1.1M	
 sentences.txt	2004-03-02 14:49	1.3M	
 words.txt	2004-03-02 14:49	5.1M	

Apache/2.4.10 (Debian) Server at www.fki.inf.unibe.ch Port 80

Step 3 - Make a Folder

Step 4 - Unzip the downloaded data into a folder inside the created folder.

Step 5 - Move the “text file” to the folder.

The structure of the folder must look like

```
Data_Example_Folder
├── forms.txt
└── sentences
    ├── a01
    │   ├── a01-000u
    │   │   ├── a01-000u-s00-00.png
    │   │   └── a01-000u-s00-01.png
    │   └── a01-000x
    │       ├── a01-000x-s00-00.png
    │       └── a01-000x-s00-01.png
    └── a02
        ├── a02-000
        │   ├── a02-000-s00-00.png
        │   └── a02-000-s00-01.png
        └── a02-004
            ├── a02-004-s00-00.png
            └── a02-004-s00-01.png
```

2) Creating a dictionary to store form ID and its writer

This is the first step of the program. The first step include the creation of dictionary which will map each form ID to a writer. This information is available in the “text file” that have been downloaded from the ascii folder in the IAM Database. Each line of the text file (except for the first 16 lines, which are documentation) defines the form ID at index 0, and its writer at index 1.

```
# Create a dictionary to store each form ID and its writer
import os
from itertools import islice
form_writer = {}
forms_file_path = "gdrive/My Drive/dataset_rio/forms.txt" # Enter the path of the form file.
with open(forms_file_path) as f:
    for line in islice(f, 16, None):
        line_list = line.split(' ')
        form_id = line_list[0]
        writer = line_list[1]
        form_writer[form_id] = writer
list(form_writer.items())[0:5] # Visualize dictionary (as array for simplicity)
```

```
[('a01-020x', '010'),
 ('a01-026', '009'),
 ('a01-026u', '000'),
 ('a01-026x', '008'),
 ('a01-030', '005')]
```

3) Selecting the writers ID which will be used for making the dataset

From the created dictionary we have to select a few writers in order to increase the efficiency of the program. In this program we have used 50 writers from the 221 writers present in the dataset.

```
# Select the 50 most common writer
from collections import Counter

top_writers = []
num_writers = 50 # Enter the number of writers from 221 you want to make dataset of.
writers_counter = Counter(form_writer.values())
for writer_id, _ in writers_counter.most_common(num_writers):
    top_writers.append(writer_id)
print(top_writers[0:5]) # Visualize the writer id of the top 50 writers.
```

```
['000', '150', '151', '152', '153']
```

4) Selecting the form that are associated with the selected writers ID

From the above selected writers, we need to select the images associated with them.

```
# From the 50 most common writers we have selected, we'll now need to select the forms (sentences) they have written:
top_forms = []
for form_id, author_id in form_writer.items():
    if author_id in top_writers:
        top_forms.append(form_id)
print(top_forms[0:5]) # Visualize the form id of the top 50 writers:
```

```
['a01-026u', 'a01-030u', 'a01-043u', 'a01-049u', 'a01-049x']
```

5) Creating a temporary directory which will store the writer's image from the selected writer ID

Creating a temporary directory which will contain the images which are associated with the selected writers ID.

```
import os
import glob
import shutil

# Create temp directory to save writers' forms in (assumes files have already been copied if the directory exists)
temp_sentences_path = "gdrive/My Drive/dataset_r10/temp_sentences"
if not os.path.exists(temp_sentences_path):
    os.makedirs(temp_sentences_path)
# Copy forms that belong to the top 50 most common writers to the temp directory
original_sentences_path = "gdrive/My Drive/dataset_r10/sentences/**/*/*.png"
for file_path in glob.glob(original_sentences_path):
    image_name = file_path.split('/')[-1]
    file_name, _ = os.path.splitext(image_name)
    form_id = '-'.join(file_name.split('-')[0:2])
    if form_id in top_forms:
        shutil.copy2(file_path, temp_sentences_path + "/" + image_name)
```

6) Creating arrays of the input file and visualize it.

Creating the array of the selected image in the temporary folder and the writer ID associated with it.

```
import numpy as np

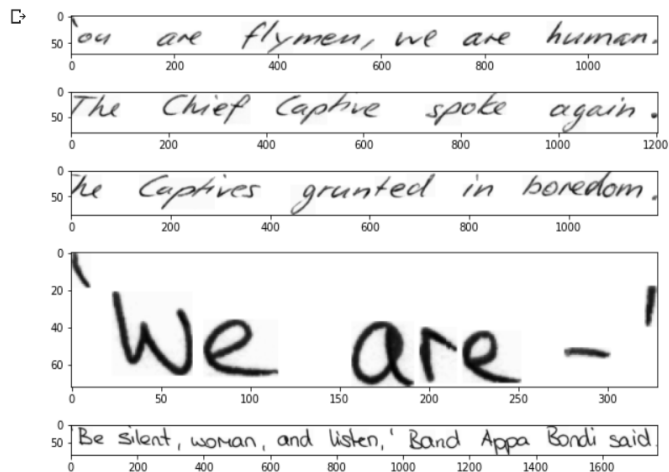
img_files = np.zeros((0), dtype=np.str)
img_targets = np.zeros((0), dtype=np.str)
path_to_files = os.path.join(temp_sentences_path, '')
for file_path in glob.glob(path_to_files):
    img_files = np.append(img_files, file_path)
    file_name, _ = os.path.splitext(file_path.split('/')[-1])
    form_id = '-'.join(file_name.split('-')[0:2])
    for key in form_writer:
        if key == form_id:
            img_targets = np.append(img_targets, form_writer[form_id])
print(img_files[0:15])
print(img_targets[0:15])
```

```
[ 'gdrive/My Drive/dataset_rio/temp_sentences/m06-019-s09-00.png'
'gdrive/My Drive/dataset_rio/temp_sentences/m06-019-s06-00.png'
'gdrive/My Drive/dataset_rio/temp_sentences/m06-019-s05-00.png'
'gdrive/My Drive/dataset_rio/temp_sentences/m06-031-s02-00.png'
'gdrive/My Drive/dataset_rio/temp_sentences/m06-031-s03-00.png'
'gdrive/My Drive/dataset_rio/temp_sentences/m06-031-s05-00.png'
'gdrive/My Drive/dataset_rio/temp_sentences/m06-031-s04-00.png'
'gdrive/My Drive/dataset_rio/temp_sentences/m06-031-s01-00.png'
'gdrive/My Drive/dataset_rio/temp_sentences/m06-031-s05-01.png'
'gdrive/My Drive/dataset_rio/temp_sentences/m06-031-s00-00.png'
'gdrive/My Drive/dataset_rio/temp_sentences/m06-031-s06-00.png'
'gdrive/My Drive/dataset_rio/temp_sentences/m06-031-s06-01.png'
'gdrive/My Drive/dataset_rio/temp_sentences/m06-031-s07-00.png'
'gdrive/My Drive/dataset_rio/temp_sentences/m06-031-s09-00.png'
'gdrive/My Drive/dataset_rio/temp_sentences/m06-031-s08-01.png']
['551' '551' '551' '552' '552' '552' '552' '552' '552' '552' '552' '552'
'552' '552' '552']
```

Visualizing the image

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
%matplotlib inline

for file_name in img_files[:5]:
    img = mpimg.imread(file_name)
    plt.figure(figsize = (10,10))
    plt.imshow(img, cmap = 'gray')
```



7) Encoding writer's ID.

Encoding the writes with a value between 0 and n_classes-1.

```
from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()
encoder.fit(img_targets)
encoded_img_targets = encoder.transform(img_targets)

print("Writer ID      : ", img_targets[:15])
print("Encoded writer ID: ", encoded_img_targets[:15])
```

```
Writer ID      : ['551' '551' '551' '552' '552' '552' '552' '552' '552' '552' '552' '552' '552' '552' '552']
Encoded writer ID: [42 42 42 43 43 43 43 43 43 43 43 43 43 43 43]
```

8) Splitting the dataset into train, validation and tests sets

Splitting the dataset into train, validation and test sets so that these sets could use for training and testing purpose.

```
from sklearn.model_selection import train_test_split

# Split dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(img_files, encoded_img_targets, test_size=0.2, shuffle = True)

# Further split training set into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, shuffle = True)

print(X_train.shape, X_val.shape, X_test.shape)
print(y_train.shape, y_val.shape, y_test.shape)
```

```
(3096,) (775,) (968,)
(3096,) (775,) (968,)
```

9) Defining constants that will be used throughout the project

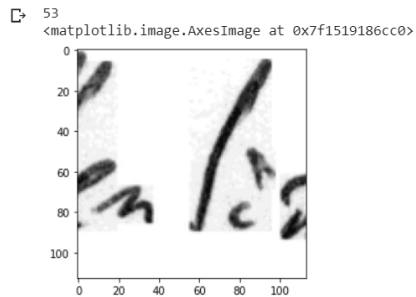
Defining some constants they are crop size, number of labels and batch size that will be used in the project.

```
CROP_SIZE = 113
NUM_LABELS = 100
BATCH_SIZE = 160
```

10) Cropping the Inputted Data and converting it into array and visualizing sample from training set

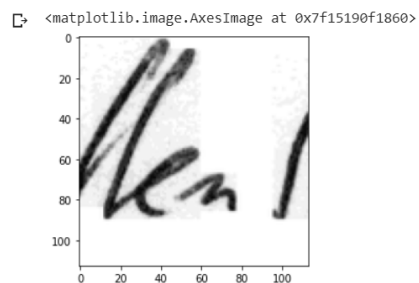
As suggested in the paper, the input to the model are not unique sentences but rather random patches cropped from each sentence. The `get_augmented_sample` method is in charge of doing so by resizing each sentence's height to 113 pixels, and its width such that original aspect ratio is maintained. Finally, from the resized image, patches of 113x113 are randomly cropped.


```
print(len(images)) # The images returned by it are the random patches created from the original image (only two samples shown for simplicity)
plt.imshow(images[0], cmap = 'gray')
```



Visualizing another sample of the cropped image of the original text

```
plt.imshow(images[1], cmap = 'gray')
```



11) Creating a generating method that will used to call the trainset while training

The program uses a generator method in order to call sample of image created in the above method when training the model.

```
[ ] import operator
    from functools import reduce
    from keras.utils import to_categorical

    def generate_data(samples, labels, batch_size, sample_ratio):
        while 1:
            for offset in range(0, len(samples), batch_size):
                batch_samples = samples[offset:(offset + batch_size)]
                batch_labels = labels[offset:(offset + batch_size)]

                # Augment each sample in batch
                augmented_batch_samples = []
                augmented_batch_labels = []
                for i in range(len(batch_samples)):
                    sample = batch_samples[i]
                    label = batch_labels[i]
                    augmented_samples, augmented_labels = get_augmented_sample(sample, label, sample_ratio)
                    augmented_batch_samples.append(augmented_samples)
                    augmented_batch_labels.append(augmented_labels)
```



```
# Flatten out samples and labels
augmented_batch_samples = reduce(operator.add, augmented_batch_samples)
augmented_batch_labels = reduce(operator.add, augmented_batch_labels)

# Reshape input format
X_train = np.array(augmented_batch_samples)
X_train = X_train.reshape(X_train.shape[0], CROP_SIZE, CROP_SIZE, 1)

# Transform input to float and normalize
X_train = X_train.astype('float32')
X_train /= 255

# Encode y
y_train = np.array(augmented_batch_labels)
y_train = to_categorical(y_train, NUM_LABELS)

yield X_train, y_train
```


Creating training, validation and test generators

```
train_generator = generate_data(X_train, y_train, BATCH_SIZE, 0.3)
validation_generator = generate_data(X_val, y_val, BATCH_SIZE, 0.3)
test_generator = generate_data(X_test, y_test, BATCH_SIZE, 0.1)
```

Importing Tensor Flow

```
import tensorflow as tf

config = tf.compat.v1.ConfigProto()
tf.compat.v1.Session(config = config)
```

 <tensorflow.python.client.session.Session at 0x7f15190ae6d8>

Resizing the image to size

```
def resize_image(img): # Function to resize image.
    size = round(CROP_SIZE/2)
    return tf.image.resize(img, [size, size])
```

12) Building a network architecture in Keras using CNN.

Building a network in Keras using CNN

```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Lambda, Activation
from keras.layers.convolutional import Convolution2D, ZeroPadding2D, MaxPooling2D
from keras.optimizers import Adam
from keras import metrics

model = Sequential()

# Define network input shape
model.add(ZeroPadding2D((1, 1), input_shape=(CROP_SIZE, CROP_SIZE, 1)))
# Resize images to allow for easy computation
model.add(Lambda(resize_image))

# CNN model - Building the model suggested in paper
model.add(Convolution2D(filters= 32, kernel_size=(5,5), strides=(2, 2), padding='same', name='conv1'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), name='pool1'))

model.add(Convolution2D(filters= 64, kernel_size=(3, 3), strides=(1, 1), padding='same', name='conv2'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), name='pool2'))

model.add(Convolution2D(filters= 128, kernel_size=(3, 3), strides=(1, 1), padding='same', name='conv3'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), name='pool3'))

model.add(Flatten())
model.add(Dropout(0.5))

model.add(Flatten())
model.add(Dropout(0.5))

model.add(Dense(512, name='dense1'))
model.add(Activation('relu'))
model.add(Dropout(0.5))

model.add(Dense(256, name='dense2'))
model.add(Activation('relu'))
model.add(Dropout(0.5))

model.add(Dense(NUM_LABELS, name='output'))
model.add(Activation('softmax'))

model.compile(loss='categorical_crossentropy', optimizer=Adam(), metrics=['acc'])

print(model.summary())
```

13) Training the model

The model is trained using GPU acceleration.

```
from keras.callbacks import ModelCheckpoint

# Create directory to save checkpoints at
model_checkpoints_path = "gdrive/My Drive/model_checkpoints"
if not os.path.exists(model_checkpoints_path):
    os.makedirs(model_checkpoints_path)

# Save model after every epoch using checkpoints
create_checkpoint = ModelCheckpoint(
    filepath = "gdrive/My Drive/model_checkpoints/check_{epoch:02d}_{val_loss:.4f}.hdf5",
    verbose = 1,
    save_best_only = False
)
```

```
# Fit model using generators
history_object = model.fit_generator(
    train_generator,
    steps_per_epoch = round(len(X_train) / BATCH_SIZE),
    validation_data = validation_generator,
    validation_steps = round(len(X_val) / BATCH_SIZE),
    epochs = 200,
    verbose = 1,
    callbacks = [create_checkpoint]
)
```

```
Epoch 00156: saving model to gdrive/My Drive/model_checkpoints/check_156_0.4628.hdf5
Epoch 157/200
194/194 [=====] - 106s 544ms/step - loss: 0.2951 - acc: 0.9087 - val_loss: 0.5390 - val_acc: 0.8612

Epoch 00157: saving model to gdrive/My Drive/model_checkpoints/check_157_0.5390.hdf5
Epoch 158/200
194/194 [=====] - 107s 550ms/step - loss: 0.2963 - acc: 0.9082 - val_loss: 0.3148 - val_acc: 0.8674

Epoch 00158: saving model to gdrive/My Drive/model_checkpoints/check_158_0.3148.hdf5
Epoch 159/200
194/194 [=====] - 108s 555ms/step - loss: 0.2937 - acc: 0.9089 - val_loss: 0.3129 - val_acc: 0.8671

Epoch 00159: saving model to gdrive/My Drive/model_checkpoints/check_159_0.3129.hdf5
Epoch 160/200
194/194 [=====] - 107s 553ms/step - loss: 0.2954 - acc: 0.9082 - val_loss: 0.5459 - val_acc: 0.8708

Epoch 00160: saving model to gdrive/My Drive/model_checkpoints/check_160_0.5459.hdf5
Epoch 161/200
194/194 [=====] - 105s 540ms/step - loss: 0.2917 - acc: 0.9096 - val_loss: 0.3148 - val_acc: 0.8688

Epoch 00161: saving model to gdrive/My Drive/model_checkpoints/check_161_0.3148.hdf5
Epoch 162/200
194/194 [=====] - 109s 562ms/step - loss: 0.2893 - acc: 0.9099 - val_loss: 0.3588 - val_acc: 0.8724
```

14) Checking the accuracy of the trained model

Selecting the best model from the trained models and checking the model for the accuracy.

```
[ ] model_weights_path = "gdrive/My Drive/model_checkpoints/check_184_0.1748.hdf5"
if model_weights_path:
    model.load_weights(model_weights_path)
    scores = model.evaluate_generator(test_generator, steps=round(len(X_test)/BATCH_SIZE))
    print("Accuracy: ", scores[1], " :: ", len(scores))
else:
    print("Set model weights file to load in the 'model_weights_path' variable")
```

```
Accuracy: 0.92054682970047 :: 2
```

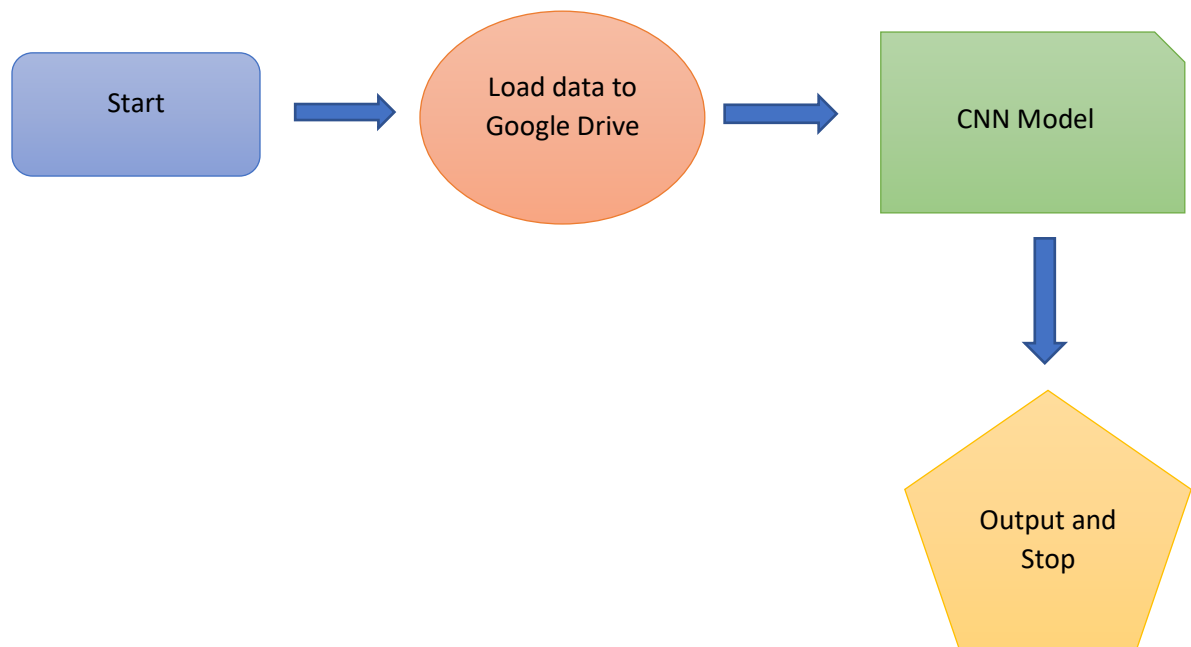
For better accuracy fine tune model hyper parameters on the dataset.

Assumptions: The assumptions considered as follows:

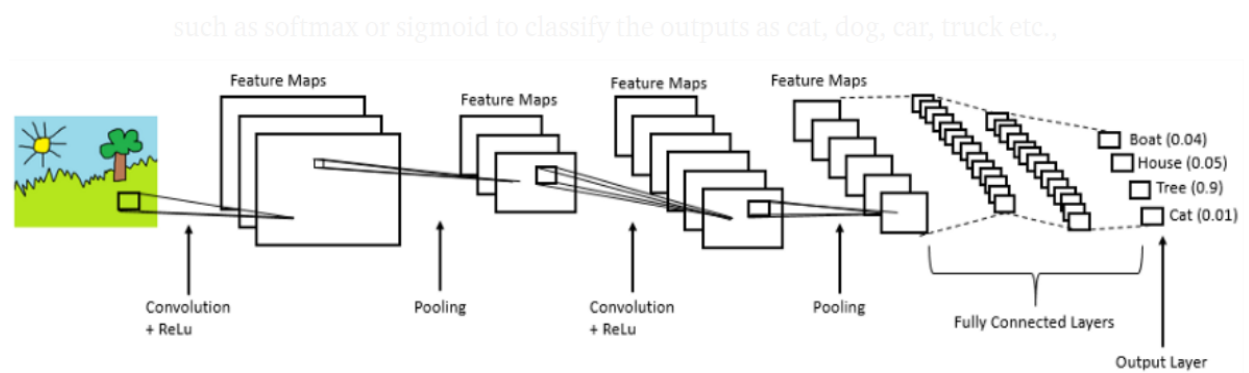
- 1) The handwritten text across the image must be in English.
- 2) The image should not be tilted.
- 3) Only image is provided for text recognition.
- 4) All machine dependencies must be installed properly

Project Diagrams:

1) Block Diagram



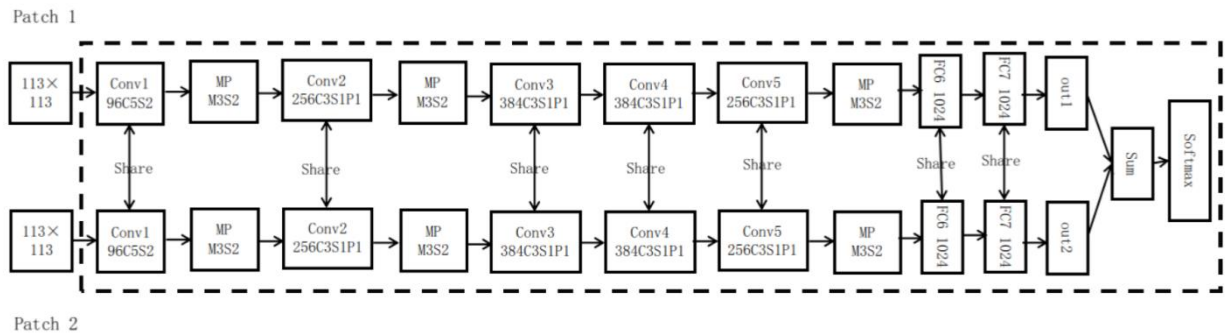
2) CNN Model



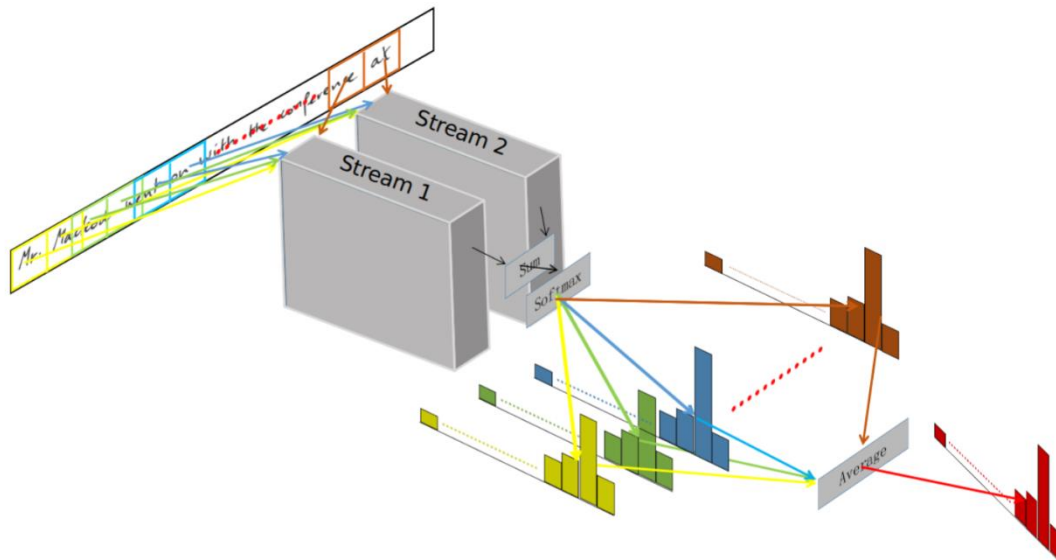
Perform convolution on the image and apply ReLU activation to the

A Simple CNN Model

3) CNN model used in the project



CNN Used



A multi-stream CNN network

Algorithms:

Model = Image + CNN + CTC loss

Our model consists of three parts

- 1) Image set
- 2) The Convolution Neural Network for extraction of Image
- 3) CTC loss function which is transcription layer used to predict output for each time step.

Model Architecture

Layer (type)	Output Shape	Param #
zero_padding2d_1 (ZeroPaddin	(None, 115, 115, 1)	0
lambda_1 (Lambda)	(None, 56, 56, 1)	0
conv1 (Conv2D)	(None, 28, 28, 32)	832
activation_1 (Activation)	(None, 28, 28, 32)	0
pool1 (MaxPooling2D)	(None, 14, 14, 32)	0
conv2 (Conv2D)	(None, 14, 14, 64)	18496
activation_2 (Activation)	(None, 14, 14, 64)	0
pool2 (MaxPooling2D)	(None, 7, 7, 64)	0
conv3 (Conv2D)	(None, 7, 7, 128)	73856
activation_3 (Activation)	(None, 7, 7, 128)	0
pool3 (MaxPooling2D)	(None, 3, 3, 128)	0
flatten_1 (Flatten)	(None, 1152)	0
dropout_1 (Dropout)	(None, 1152)	0
dense1 (Dense)	(None, 512)	590336
activation_4 (Activation)	(None, 512)	0
dropout_2 (Dropout)	(None, 512)	0
dense2 (Dense)	(None, 256)	131328
activation_5 (Activation)	(None, 256)	0
dropout_3 (Dropout)	(None, 256)	0
output (Dense)	(None, 50)	12850

```
activation_6 (Activation)      (None, 50)      0
=====
Total params: 827,698
Trainable params: 827,698
Non-trainable params: 0
```

Strategy used in the program

- 1) Design and optimize multi-stream structure for writer identification task.
- 2) Introducing data augmentation learning to enhance the performance of the program.
- 3) Introducing a patch scanning strategy to handle text image with different lengths.

Outcome:

The algorithm is able to detect and segment handwritten text from an image. The model successfully able to detect maximum words in a given line of sentence or words, which makes it about 90% accurate while implementation and testing.

Exceptions considered: The exceptions considered are as follows:

- 1) The text across the input image must be of the same color not multicolor handwritten text.
- 2) The image doesn't have too aggressive multicolor backgrounds across the text of the image.
- 3) The image doesn't have any kind's objects in the background across the text of the image.
- 4) The image should not be tilted or rotated.

Enhancement Scope: The enhancement scope of this project are follows:

- 1) The accuracy of the model can increase with predefined models and powerful machine learning GPU processors can be used to attain a good percentage of accuracy.
- 2) In future we can use this algorithm with more than one particular language.
- 3) If we increase the number of epochs, we can get a better accuracy.
- 4) This Model can be used in paragraph extraction if we increase the CNN layers and RNN layers and preprocess the data well.
- 5) This Model can be used in extraction of text from video if we can join CNN and OpenCV concepts together.

References:

- 1) <https://arxiv.org/pdf/1606.06472.pdf>
- 2) <https://software.intel.com/content/www/us/en/develop/training/course-artificial-intelligence.html>
- 3) <https://software.intel.com/content/www/us/en/develop/training/course-machine-learning.html>
- 4) https://www.python-course.eu/machine_learning.php
- 5) <https://numpy.org/doc/>
- 6) <https://software.intel.com/en-us/ai/courses/deep-learning>
- 7) <https://www.tensorflow.org/tutorials/images/classification>
- 9) <https://www.tensorflow.org/tutorials/images/cnn>
- 10) <https://www.tensorflow.org/tutorials/keras/classification>

- 11) <https://www.tensorflow.org/tutorials>
- 12) <https://pandas.pydata.org/pandas-docs/version/0.15/tutorials.html>
- 13) <http://www.fki.inf.unibe.ch/databases/iam-handwriting-database/download-the-iam-handwriting-database>
- 14) <https://towardsdatascience.com/a-gentle-introduction-to-ocr-ee1469a201aa>
- 15) <https://link.springer.com/article/10.1007/s11036-019-01243-5>
- 16) <https://medium.com/@tomhoag/opencv-text-detection-548950e3494c>
- 17) https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_ml/py_knn/py_knn_opencv/py_knn_opencv.html
- 18) https://github.com/RiteshKH/Cursive_handwriting_recognition
- 19) <https://www.pyimagesearch.com/2014/09/22/getting-started-deep-learning-python/>
- 20) <http://hanzratech.in/2015/02/24/handwritten-digit-recognition-using-opencv-sklearn-and-python.html>
- 21) <https://www.learnopencv.com/deep-learning-based-text-recognition-ocr-using-tesseract-and-opencv/>

Link to Code and executable file:

Link to GitHub - <https://github.com/SaubhagyajeetSahoo/TCSiON-Hand-Writing-Extraction-from-an-Image>

GitHub Main Project File - <https://github.com/SaubhagyajeetSahoo/TCSiON-Hand-Writing-Extraction-from-an-Image/tree/master/Main%20Project%20file>

Google Colab Link - https://colab.research.google.com/drive/1wrFa5CAGefQH_J-yBh0KjVrUm5PRI_Ga?usp=sharing