TEXT DETECTION AND RECOGNITION

Yash Shah
yash.shah2020@vitstudent.ac.in
9594131021

Nithin Kodipyaka
kodipyaka.nithin2020@vitstudent.ac.in
7386150868

Jalavadi Shree Nikhila
jalavadishree.nikhila2020@vitstudent.ac.in
9908568845

Swetha B
swetha.b2020a@vitstudent.ac.in
7396245379

Abstract

The goal of text recognition in pictures research is to create a computer system that can automatically interpret text from photographs. Nowadays, there is a significant need for saving information that is now only available in paper documents on a computer storage drive so that it can subsequently be used by a search procedure. Scan the paper papers, then store the images as a straightforward method of storing data from these paper documents in computer systems. However, it is highly challenging to read the individual contents and search the contents of these documents line-by-line and word-by-word in order to reuse this information. The font qualities of the characters in printed texts and the image quality provide difficulties. These difficulties prevent the computer from reading the characters correctly. In order to execute Document Image Analysis (DIA), which converts paper documents into electronic format, character recognition algorithms are required. This study discusses a technique for extracting text from photographs. With the use of a specific order of various processing modules, the goal of this work is to recognize text from an image for the reader's benefit. In this study, we introduce an efficient CNN model that uses attached binary images to recognize Captchas. We further perform registration number and serial id recognition on a given certificate pdf. First, we input a pdf file and them covert it into image. It will perform OCR on given image using bounding boxes. It will then use regex pattern on the extracted text to identify the required text, by giving Registration and Serial ID as the output.

Keywords: recognition, text-based CAPTCHA, convolutional neural network, deep learning , OCR, Regex

1. Introduction

CAPTCHA is an algorithm for regional human behavior and machine behavior . With the rapid development of Internet technology, network security issues continue to expand. CAPTCHA recognition is an effective way to maintain network security and prevent malicious attacks from

computer programs, and it has been widely used in major mainstream websites . CAPTCHA is generally considered to be a reverse turing test to classify humans and computers .

The mainstream CAPTCHA is based on visual representation, including images such as letters and text. Traditional CAPTCHA recognition [4–6] includes three steps: image preprocessing, character segmentation, and character recognition. Traditional methods have generalization capabilities and robustness for different types of CAPTCHA. The stickiness is poor. As a kind of deep neural network, convolutional neural network has shown excellent performance in the field of image recognition, and it is much better than traditional machine learning methods. Compared with traditional methods, the main advantage of CNN lies in the convolutional layer in which the extracted image features have strong expressive ability, avoiding the problems of data preprocessing and artificial design features in traditional recognition technology. Although CNN has achieved certain results, the recognition effect of complex CAPTCHA is insufficient .

In this project, firstly we perform captcha recognition and then we further recognize registration number and serial id registration on a given certificate pdf. First, we input a pdf file and them covert it into image. It will perform OCR on given image to create bounding boxes. It will then use regex pattern on the extracted text to identify the required text, by giving registration and serial id as the output. OCR stands for "Optical Character Recognition." It is a technology that recognizes text within a digital image. It is commonly used to recognize text in scanned documents and images. OCR software can be used to convert a physical paper document, or an image into an accessible electronic version with text.

Pytesseract or Python-tesseract is an OCR tool for python that also serves as a wrapper for the Tesseract-OCR Engine. It can read and recognize text in images and is commonly used in python ocr image to text use cases.

Regular Expression or (Regex) is one of the most powerful, flexible, and efficient text processing approaches. Regex has its own terminologies, conditions, and syntax; it is, in a sense, a mini programming language. Regex can be used to add, remove, isolate, and manipulate all kinds of text and data.

2. Background study

A. Captcha recognition based on deep convolutional neural network :

Aiming at the problems of low efficiency and poor accuracy of traditional CAPTCHA recognition methods, we have proposed a more efficient way based on deep convolutional neural network (CNN). The Dense Convolutional Network (DenseNet) has shown excellent classification performance which adopts cross-layer connection. Not only it effectively alleviates the vanishing-gradient problem, but also dramatically reduce

the number of parameters. However, it also has caused great memory consumption. So we improve and construct a new DenseNet for CAPTCHA recognition (DFCR). Firstly, we reduce the number of convolutional blocks and build corresponding classifiers for different types of CAPTCHA images. Secondly, we input the CAPTCHA images of TFrecords format into the DFCR for model training. Finally, we test the Chinese or English CAPTCHAs experimentally with different numbers of characters. Experiments show that the new network not only keeps the primary performance advantages of the DenseNets but also effectively reduces the memory consumption. Furthermore, the recognition accuracy of CAPTCHA with the background noise and character adhesion is above 99.9%.

The focus of this paper is to compare several common machine learning classification algorithms for Optical Character Recognition of CAPTCHA codes. The main part of a research focuses on the comparative study of Neural Networks, k-Nearest Neighbour, Support Vector Machines and Decision Trees implemented in MATLAB Computing environment. Achieved success rates of all analyzed algorithms overcome 89%. The main difference in results of used algorithms is within the learning times. Based on the data found, it is possible to choose the right algorithm for the particular task.

B. CAPTCHA Recognition Method Based on CNN with Focal Loss :

In order to distinguish between computers and humans, CAPTCHA is widely used in links such as website login and registration. The traditional CAPTCHA recognition method has poor recognition ability and robustness to different types of verification codes. For this reason, the paper proposes a CAPTCHA recognition method based on convolutional neural network with focal loss function. This method improves the traditional VGG network structure and introduces the focal loss function to generate a new CAPTCHA recognition model. First, we perform preprocessing such as grayscale, binarization, denoising, segmentation, and annotation and then use the Keras library to build a simple neural network model. In addition, we build a terminal end-to-end neural network model for recognition for complex CAPTCHA with high adhesion and more interference pixel. By testing the CNKI CAPTCHA, Zhengfang CAPTCHA, and randomly generated CAPTCHA, the experimental results show that the proposed method has a better recognition effect and robustness for three different datasets, and it has certain advantages compared with traditional deep learning methods. The recognition rate is 99%, 98.5%, and 97.84%, respectively.

C. Breaking CAPTCHAs with Convolutional Neural Networks :

This paper studies reverse Turing tests to distinguish humans and computers, called CAPTCHA. Contrary to classical Turing tests, in this case the judge is not a human but a

computer. The main purpose of such tests is securing user logins against the dictionary or brute force password guessing, avoiding automated usage of various services, preventing bots from spamming on forums and many others

3. Proposed Methodology

Preprocessing

Traditional processing methods are used to preprocess the CAPTCHA image, including grayscale, binarization, image denoising, image segmentation, and image annotation.

Creating Dataset for Training and validation purpose of model :

```
df = pd.DataFrame(columns=['filename','extension','label','labelsize','char1','char2','c
har3','char4','char5'])
i = 0
for _, _, files in os.walk(img_folder):
    for f in files:
        df.loc[i,'filename'] = f
        df.loc[i,'extension'] = f.split('.')[1]
        df.loc[i,'label'] = f.split('.')[0]
        df.loc[i,'labelsize'] = len(f.split('.')[0])
        df.loc[i,'char1'] = f.split('.')[0][0]
        df.loc[i,'char2'] = f.split('.')[0][1]
        df.loc[i,'char3'] = f.split('.')[0][2]
        df.loc[i,'char4'] = f.split('.')[0][3]
        df.loc[i,'char5'] = f.split('.')[0][4]
        i = i+1
```

Converting Alphabets to Integers:

```
char_to_num = {'0':0,'1':1,'2':2,'3':3,'4':4,'5':5,'6':6,'7':7,'8':8,'9':9
               ,'a':10,'b':11,'c':12,'d':13,'e':14,'f':15,'g':16,'h':17,'i':18,
              'j':19,'k':20,'l':21,'m':22,'n':23,'o':24,'p':25,'q':26,'r':27,
               's':28,'t':29,'u':30,'v':31,'w':32,'x':33,'y':34,'z':35}
```

We have used two max pool with pool size and strides Hence, down sampled feature maps are 4x smaller. The number of filters in the last layer is 64 --> output volume shape = (50,12,64). Reshape to "split" the volume in 5 time-steps.

Creating train and validation datasets:

```
img_folder = "/Users/Acer/archive/img/"
def create_train_and_validation_datasets(crop=False):
    # Loop on all the files to create X whose shape is (1040, 50, 200, 1) and y whose sha
pe is (1040, 5)
    X, y = [],[]

    for _, _, files in os.walk(img_folder):
        for f in files:
            # To start, let's ignore the jpg images
            label = f.split('.')[0]
            extension = f.split('.')[1]
            if extension=='png':
                img, label = encode_single_sample(img_folder+f, label,crop)
                X.append(img)
                y.append(label)

    X = np.array(X)
    y = np.array(y)
    X_train, X_val, y_train, y_val = train_test_split(X.reshape(64961, 10000), y, test_s
ize=0.1, shuffle=True, random_state=42)
    X_train, X_val = X_train.reshape(58464,200,50,1), X_val.reshape(6497,200,50,1)
    return X_train, X_val, y_train, y_val
```

```
Model: "ocr_classifier_based_model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 image (InputLayer)          [(None, 200, 50, 1)]      0

 Conv1 (Conv2D)              (None, 200, 50, 32)       320

 pool1 (MaxPooling2D)        (None, 100, 25, 32)       0

 Conv2 (Conv2D)              (None, 100, 25, 64)       18496

 pool2 (MaxPooling2D)        (None, 50, 12, 64)        0

 reshape (Reshape)           (None, 6, 6400)           0

 dense1 (Dense)              (None, 6, 256)            1638656

 dense2 (Dense)              (None, 6, 128)            32896

 dense3 (Dense)              (None, 6, 64)             8256

 dense4 (Dense)              (None, 6, 36)             2340

=================================================================
Total params: 1,700,964
Trainable params: 1,700,964
Non-trainable params: 0
_________________________________________________________________
```

4. Implementation

TensorFlow:

The TensorFlow platform helps you implement best practices for data automation, model tracking, performance monitoring, and model retraining. Using production-level tools to automate and track model training over the lifetime of a product, service, or business process is critical to success.

Keras:

Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library. Keras contains numerous implementations of commonly used neural-network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier to simplify the coding necessary for writing deep neural network code.

Sklearn model selection:

Sklearn's model selection module provides various functions to cross-validate our model, tune the estimator's hyperparameters, or produce validation and learning curves.

The Sklearn train_test_split function helps us create our training data and test data. This is because typically, the training data and test data come from the same original dataset. To get the data to build a model, we start with a single dataset, and then we split it into two datasets: train and test.

Regex:

A RegEx, or Regular Expression, is a sequence of characters that forms a search pattern. RegEx can be used to check if a string contains the specified search pattern.

OCR with Tesseract:

OCR = Optical Character Recognition. In other words, OCR systems transform a two-dimensional image of text, that could contain machine printed or handwritten text from its image representation into machine-readable text. OCR as a process generally consists of several sub-processes to perform as accurately as possible.

FC layers (Fully Connected Layers):

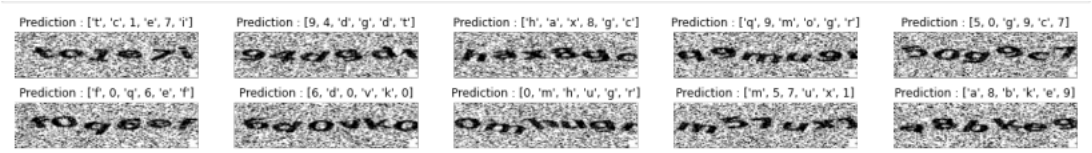
Each individual function consists of a neuron (or a perceptron). In fully connected layers, the neuron applies a linear transformation to the input vector through a weights matrix. A non-linear transformation is then applied to the product through a non-linear activation function f.

5. Results and Discussion

1. Captcha Recognition (OCR Classifier Based Model)
   We have used CNN model to train the CAPTCHAs and we received an accuracy of 99.35%. The model successfully is able predict CAPTCHAs from a given image.

```
y: 0.9379
Epoch 20/30
1827/1827 [==============================] - 289s 158ms/step - loss: 0.0276 - accuracy: 0.9911 - val_loss: 0.4595 - val_accurac
y: 0.9358
Epoch 21/30
1827/1827 [==============================] - 283s 155ms/step - loss: 0.0272 - accuracy: 0.9912 - val_loss: 0.4531 - val_accurac
y: 0.9398
Epoch 22/30
1827/1827 [==============================] - 280s 154ms/step - loss: 0.0252 - accuracy: 0.9920 - val_loss: 0.4801 - val_accurac
y: 0.9374
Epoch 23/30
1827/1827 [==============================] - 280s 153ms/step - loss: 0.0257 - accuracy: 0.9918 - val_loss: 0.4792 - val_accurac
y: 0.9357
Epoch 24/30
1827/1827 [==============================] - 295s 161ms/step - loss: 0.0252 - accuracy: 0.9922 - val_loss: 0.5074 - val_accurac
y: 0.9353
Epoch 25/30
1827/1827 [==============================] - 310s 170ms/step - loss: 0.0225 - accuracy: 0.9930 - val_loss: 0.5156 - val_accurac
y: 0.9374
Epoch 26/30
1827/1827 [==============================] - 313s 171ms/step - loss: 0.0226 - accuracy: 0.9928 - val_loss: 0.5205 - val_accurac
y: 0.9368
Epoch 27/30
1827/1827 [==============================] - 314s 172ms/step - loss: 0.0207 - accuracy: 0.9934 - val_loss: 0.5303 - val_accurac
y: 0.9367
Epoch 28/30
1827/1827 [==============================] - 315s 172ms/step - loss: 0.0218 - accuracy: 0.9933 - val_loss: 0.5048 - val_accurac
y: 0.9376
Epoch 29/30
1827/1827 [==============================] - 315s 172ms/step - loss: 0.0212 - accuracy: 0.9935 - val_loss: 0.5337 - val_accurac
y: 0.9384
Epoch 30/30
1827/1827 [==============================] - 313s 171ms/step - loss: 0.0216 - accuracy: 0.9935 - val_loss: 0.5429 - val_accurac
y: 0.9390
```

Prediction :



2. Registration Number Text Recognition by Tesseract (OCR Model)

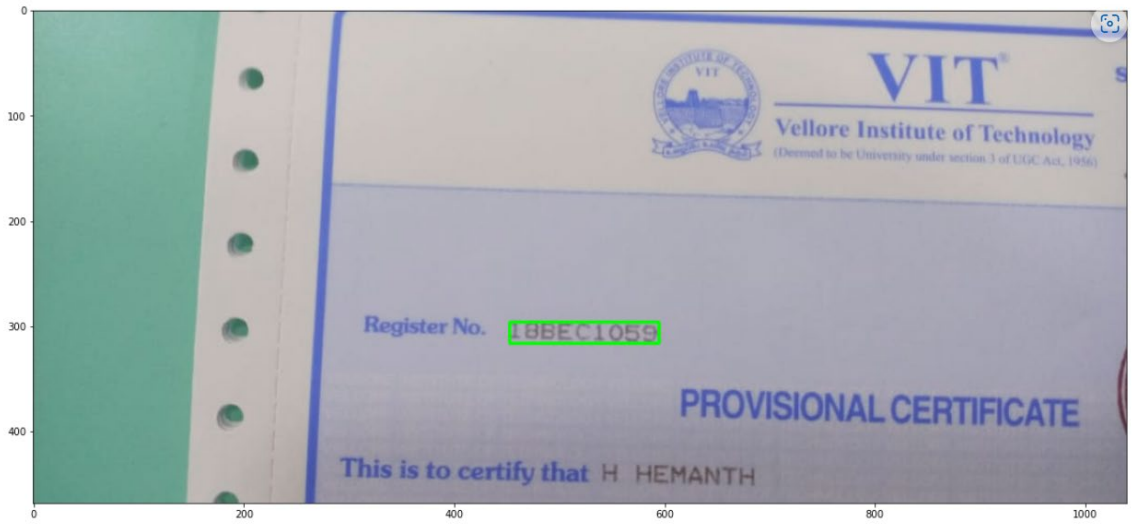
```
In [2]: pytesseract.pytesseract.tesseract_cmd = r'C:\Users\Nithin Kodipyaka\AppData\Local\Tesseract-OCR\tesseract.exe'

In [5]: img = cv2.imread("C:\\Users\\Nithin Kodipyaka\\Downloads\\im11.jpg")
```

We installed Google's Tesseract Application file and accessed the application through our model. Tesseract extracts all the text from a given image and converts Image to Data in form of Dictionaries. It creates Bounding boxes for all text. So, we use RegEx to train the model about the pattern of Registration Numbers, so that it shows output of only it. This OCR Model can recognize Registration Numbers from Certificates and ID Card Images as input.

```
import re

email_pattern = '[0-9]{2}[A-Z]{3}[0-9]{4}'

img = cv2.imread("inimg//im3.jpg")

d = pytesseract.image_to_data(img, output_type=Output.DICT)

n_boxes = len(d['text'])
for i in range(n_boxes):
    if int(d['conf'][i]) >60 :
        if re.match(email_pattern, d['text'][i]) or d['text'][i-1].__contains__('Register'):
            (x, y, w, h) = d["left"][i], d["top"][i], d["width"][i], d["height"][i]
            img = cv2.rectangle(img, (x, y), (x+w, y+h), (0, 255, 0), 2) #Plotting bounding box
            print(f"reg no: {d['text'][i]}")

plt.imshow(img)
```

```
reg no: 18BEC1059
```

```
t[12]: <matplotlib.image.AxesImage at 0x1ffce02f610>
```



3. Serial Number ID Text Recognition (OCR Classifier Based Model)

For this recognition requirement, we used the same working model of CAPTCHA's Recognition.

Importing Packages

In [28]:
```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import seaborn as sns

from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

Reading the image folder from the system

In [29]:
```
%pwd
img_folder='C:\\Users\\Nithin Kodipyaka\\Downloads\\Compressed\\resized\\'
```

Split the image's name into each column as (which is used later for validation) ['filename','extension','label','labelsize','char1','char2','c har3','char4','char5']

In [30]:
```
df = pd.DataFrame(columns=['filename','extension','label','labelsize','char1','char2','char3','char4','char5','char6'])
i = 0
for _, _, files in os.walk(img_folder):
    for f in files:
        df.loc[i,'filename'] = f
        df.loc[i,'extension'] = f.split('.')[1]
        df.loc[i,'label'] = f.split('.')[0]
        df.loc[i,'labelsize'] = len(f.split('.')[0])
        df.loc[i,'char1'] = f.split('.')[0][0]
        df.loc[i,'char2'] = f.split('.')[0][1]
        df.loc[i,'char3'] = f.split('.')[0][2]
        df.loc[i,'char4'] = f.split('.')[0][3]
        df.loc[i,'char5'] = f.split('.')[0][4]
        df.loc[i,'char6'] = f.split('.')[0][5]
        i = i+1
```

Convert the data into csv format and read the dataframe created

In [31]:
```
df.to_csv('ml_proj.csv')
```

In [32]:
```
df=pd.read_csv("ml_proj.csv")
```

The info() method prints information about the DataFrame. The information contains the number of columns, column labels, column data types, memory usage, range index, and the number of cells in each column (non-null values)

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7 entries, 0 to 6
Data columns (total 11 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Unnamed: 0  7 non-null      int64
 1   filename    7 non-null      object
 2   extension   7 non-null      object
 3   label       7 non-null      int64
 4   labelsize   7 non-null      int64
 5   char1       7 non-null      int64
 6   char2       7 non-null      int64
 7   char3       7 non-null      int64
 8   char4       7 non-null      int64
 9   char5       7 non-null      int64
 10  char6       7 non-null      int64
dtypes: int64(9), object(2)
memory usage: 744.0+ bytes
```

The Value_counts() function is used to get a series containing counts of unique values. The resulting object will be in descending order so that the first element is the most frequently-occurring element. Excludes NA values by default.

```
In [34]: df.value_counts()
```

Out[34]:

Unnamed: 0	filename	extension	label	labelsize	char1	char2	char3	char4	char5	char6	
0	101547.png	png	101547	6	1	0	1	5	4	7	1
1	101548.png	png	101548	6	1	0	1	5	4	8	1
2	101549.png	png	101549	6	1	0	1	5	4	9	1
3	101553.png	png	101553	6	1	0	1	5	5	3	1
4	101563.png	png	101563	6	1	0	1	5	6	3	1
5	101564.png	png	101564	6	1	0	1	5	6	4	1
6	126841.png	png	126841	6	1	2	6	8	4	1	1

dtype: int64

Read image file and returns a tensor. Decode and convert to grayscale (this conversion does not cause any information lost and reduces the size of the tensor) This decode function returns a tensor. Scales and returns a tensor. Crop and resize to the original size. top-left corner = offset_height, offset_width in image = 0, 25 .lower-right corner is at offset_height + target_height, offset_width + target_width = 50, 150. Transpose the image because we want the time dimension to correspond to the width of the image. Converts the string label into an array with 5 integers. E.g. '6n6gg' is converted into [6,16,6,14,14].

In [166]:

```
def encode_single_sample(img_path, label, crop):
    # Read image file and returns a tensor with dtype=string
    img = tf.io.read_file(img_path)
    # Decode and convert to grayscale (this conversion does not cause any information los
t and reduces the size of the tensor)
    # This decode function returns a tensor with dtype=uint8
    img = tf.io.decode_png(img, channels=1)
    # Scales and returns a tensor with dtype=float32
    img = tf.image.convert_image_dtype(img, tf.float32)
    # Crop and resize to the original size :
    # top-left corner = offset_height, offset_width in image = 0, 25
    # lower-right corner is at offset_height + target_height, offset_width + target_width
= 50, 150
    if(crop==True):
        img = tf.image.crop_to_bounding_box(img, offset_height=0, offset_width=75, targe
t_height=100, target_width=145)
        img = tf.image.resize(img,size=[50,200],method='bilinear', preserve_aspect_ratio
=False,antialias=False, name=None)
    # Transpose the image because we want the time dimension to correspond to the width o
f the image.
    img = tf.transpose(img, perm=[1, 0, 2])
    # Converts the string label into an array with 5 integers. E.g. '6n6gg' is converted
into [6,16,6,14,14]
    label = list(map(lambda x:char_to_num[x], label))
    return img.numpy(), label
```

Loop on all the files to create X whose shape is (1040, 50, 200, 1) and y whose shape is (1040, 5). To start, let's ignore the jpg images

In [39]:

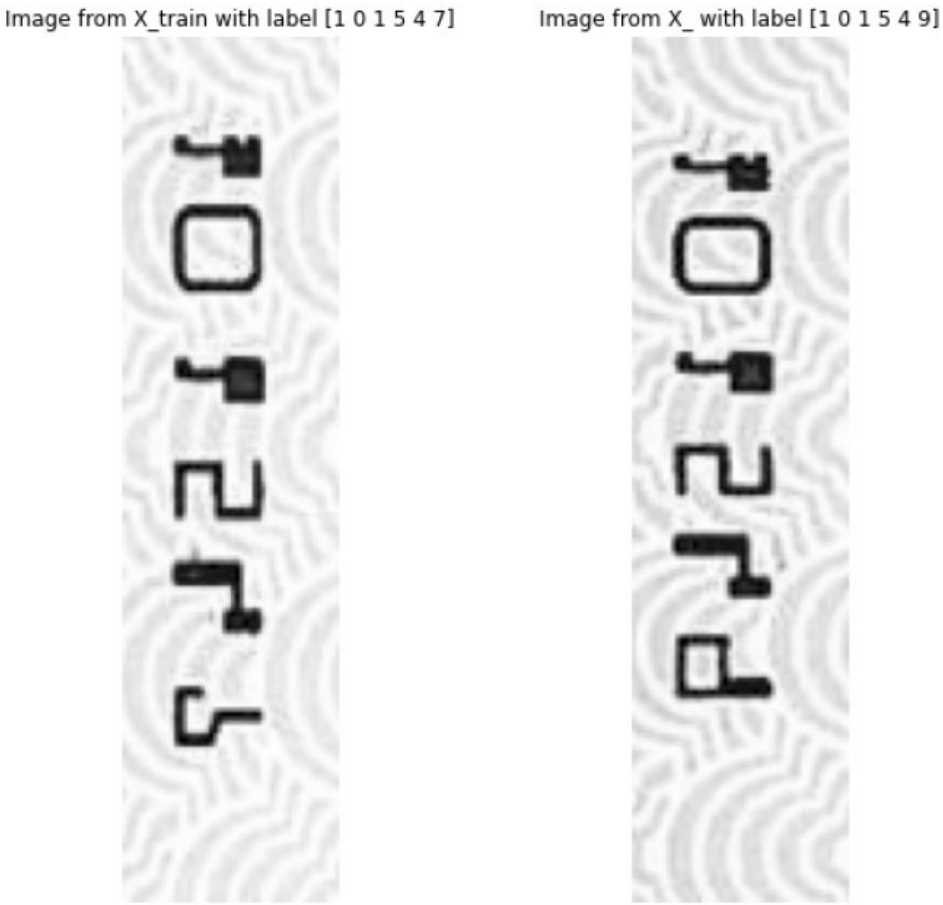
```
img_folder = "C:\\Users\\Nithin Kodipyaka\\Downloads\\Compressed\\resized\\"

X, y = [],[]
crop=False
for _, _, files in os.walk(img_folder):
    for f in files:
        # To start, let's ignore the jpg images
        label = f.split('.')[0]
        extension = f.split('.')[1]
        if extension=='png':
            img, label = encode_single_sample(img_folder+f, label,crop)
            X.append(img)
            y.append(label)

X = np.array(X)
y=np.array(y)
```

Taking 2 images from trained dataset

```
In [40]: fig=plt.figure(figsize=(20, 20))
         fig.add_subplot(2, 4, 1)
         plt.imshow(X[0], cmap='gray')
         plt.title('Image from X_train with label '+ str(y[0]))
         plt.axis('off')
         fig.add_subplot(2, 4, 2)
         plt.imshow(X[2], cmap='gray')
         plt.title('Image from X_ with label '+ str(y[2]))
         plt.axis('off')
```

Images taken for training with labels

```
Out[40]: (-0.5, 49.5, 199.5, -0.5)
```



Inputs to the model. Create 2 convolution layers through deep learning. We have used two max pool with pool size and strides 2. Hence, down sampled feature maps are 4x smaller. The number of filters in the last layer is 64 --> output volume shape = (50,12,64) Reshape to "split" the volume in 5 time-steps. Create FC Layers (Each individual function consists of a neuron (or a perceptron). In fully connected layers, the neuron applies a linear transformation

to the input vector through a weights matrix. A non-linear transformation is then applied to the product through a non-linear activation function f). Finally fit the model.

In [189]:

```
def build_model():

    # Inputs to the model
    input_img = layers.Input(shape=(200,50,1), name="image", dtype="float32")

    # First conv block
    x = layers.Conv2D(32,(3, 3),activation="relu",kernel_initializer="he_normal",padding
="same",name="Conv1")(input_img)
    x = layers.MaxPooling2D((2, 2), name="pool1")(x)

    # Second conv block
    x = layers.Conv2D(64,(3, 3),activation="relu",kernel_initializer="he_normal",padding
="same",name="Conv2")(x)
    x = layers.MaxPooling2D((2, 2), name="pool2")(x)

    # We have used two max pool with pool size and strides 2.
    # Hence, downsampled feature maps are 4x smaller. The number of
    # filters in the last layer is 64 --> output volume shape = (50,12,64)
    # Reshape to "split" the volume in 5 time-steps
    x = layers.Reshape(target_shape=(6, 6400), name="reshape")(x)

    # FC layers
    x = layers.Dense(256, activation="relu", name="dense1")(x)
    x = layers.Dense(128, activation="relu", name="dense2")(x)
    x = layers.Dense(64, activation="relu", name="dense3")(x)
    output = layers.Dense(36, activation="softmax", name="dense4")(x)

    # Define the model
    model = keras.models.Model(inputs=input_img, outputs=output, name="ocr_classifier_ba
sed_model")

    # Compile the model and return
```

```
    model.compile(optimizer=keras.optimizers.Adam(), loss="sparse_categorical_crossentrop
y", metrics="accuracy")
    return model


# Get the model
model = build_model()
model.summary()
```

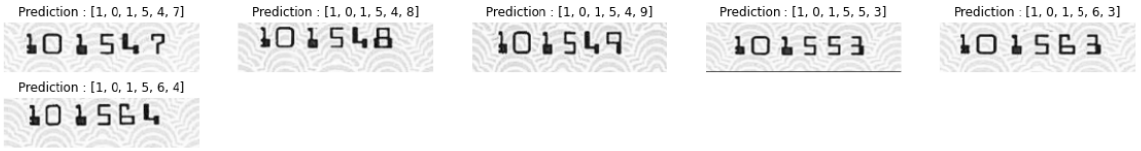
Number of epochs is a hyperparameter that defines the number times that the learning algorithm will work through the entire training dataset. One epoch means that each sample in the training dataset has had an opportunity to update the internal model parameters.

In this model we have run 50 epochs and gained an accuracy of 100 percent.

```
Epoch 1/50
1/1 [==============================] - 0s 96ms/step - loss: 0.2260 - accuracy: 0.9286
Epoch 2/50
1/1 [==============================] - 0s 96ms/step - loss: 0.2125 - accuracy: 0.9286
Epoch 3/50
1/1 [==============================] - 0s 96ms/step - loss: 0.2013 - accuracy: 0.9286
Epoch 4/50
1/1 [==============================] - 0s 96ms/step - loss: 0.1921 - accuracy: 0.9286
Epoch 5/50
1/1 [==============================] - 0s 96ms/step - loss: 0.1832 - accuracy: 0.9524
Epoch 6/50
1/1 [==============================] - 0s 96ms/step - loss: 0.1824 - accuracy: 0.9048
Epoch 7/50
1/1 [==============================] - 0s 96ms/step - loss: 0.1723 - accuracy: 0.9762
Epoch 8/50
1/1 [==============================] - 0s 88ms/step - loss: 0.1592 - accuracy: 0.9524
Epoch 9/50
1/1 [==============================] - 0s 96ms/step - loss: 0.1555 - accuracy: 0.9286
Epoch 10/50
1/1 [==============================] - 0s 96ms/step - loss: 0.1501 - accuracy: 0.9524
Epoch 11/50
1/1 [==============================] - 0s 96ms/step - loss: 0.1420 - accuracy: 0.9286
Epoch 12/50
1/1 [==============================] - 0s 96ms/step - loss: 0.1337 - accuracy: 0.9762
Epoch 13/50
1/1 [==============================] - 0s 96ms/step - loss: 0.1298 - accuracy: 1.0000
Epoch 14/50
1/1 [==============================] - 0s 120ms/step - loss: 0.1266 - accuracy: 0.9524
Epoch 15/50
1/1 [==============================] - 0s 104ms/step - loss: 0.1179 - accuracy: 0.9762
Epoch 16/50
1/1 [==============================] - 0s 104ms/step - loss: 0.1094 - accuracy: 0.9762
Epoch 17/50
1/1 [==============================] - 0s 104ms/step - loss: 0.1046 - accuracy: 1.0000
Epoch 18/50
1/1 [==============================] - 0s 104ms/step - loss: 0.1006 - accuracy: 1.0000
Epoch 19/50
1/1 [==============================] - 0s 96ms/step - loss: 0.0924 - accuracy: 1.0000
Epoch 20/50
1/1 [==============================] - 0s 88ms/step - loss: 0.0864 - accuracy: 1.0000
Epoch 45/50
1/1 [==============================] - 0s 96ms/step - loss: 0.0113 - accuracy: 1.0000
Epoch 46/50
1/1 [==============================] - 0s 88ms/step - loss: 0.0103 - accuracy: 1.0000
Epoch 47/50
1/1 [==============================] - 0s 88ms/step - loss: 0.0095 - accuracy: 1.0000
Epoch 48/50
1/1 [==============================] - 0s 104ms/step - loss: 0.0088 - accuracy: 1.0000
Epoch 49/50
1/1 [==============================] - 0s 96ms/step - loss: 0.0081 - accuracy: 1.0000
Epoch 50/50
1/1 [==============================] - 0s 128ms/step - loss: 0.0075 - accuracy: 1.0000
```

Final Prediction of the Serial Number IDs from images:

```
In [60]: nrow = 1
fig=plt.figure(figsize=(20, 5))
for i in range(0,6):
    if i>4: nrow = 2
    fig.add_subplot(nrow, 5, i+1)
    plt.imshow(X[i].transpose((1,0,2)),cmap='gray')
    plt.title('Prediction : ' + str(list(map(lambda x:num_to_char[str(x)], y_pred[i]))))
    plt.axis('off')
plt.show()
```



6. Conclusion and Future works

In this research, we discussed and proposed a text recognition approach. The Models we have used are successfully able to recognize any text from Image based on requirements. We trained the models to recognize CAPTCHA, Registration Number Detection and Serial Number IDs Detection from Certificate Images. The OCR is a vast field for pattern recognition researchers. Numerous studies on OCR for different languages have been conducted and are continuously being conducted. Researchers are drawn to this difficult topic in increasing numbers. The significance of each stage of optical character recognition should be taken into consideration when designing the system for upcoming demands.

7. References

https://medium.com/@manvi./captcha-recognition-using-convolutional-neural-network-d191ef91330e

https://www.kaggle.com/code/dmitryyemelyanov/receipt-ocr-part-2-text-recognition-by-tesseract

https://www.aimspress.com/article/doi/10.3934/mbe.2019292?viewType=HTML

https://www.hindawi.com/journals/complexity/2021/6641329/

https://ceur-ws.org/Vol-1885/93.pdf

https://www.kaggle.com/datasets/surenbobby/captchas-net **(The dataset we have used for the project)**