**Mohit Kaushik**

**205320011**

**Pattern Recognition**

**Captcha Recognition**

**Problem Definition**

CAPTCHA (Completely Automated Public Turing Test to tell Computers and Humans Apart) is an automated test created to prevent websites from being repeatedly accessed by an automatic program in a short period of time and wasting network resources. Among all the CAPTCHAs, commonly used types contain low resolution, deformed characters with character adhesions and background noise, which the user must read and type correctly into an input box. This is a relatively simple task for humans, taking an average of 10 seconds to solve, but it presents a difficulty for computers, because such noise makes it difficult for a program to differentiate the characters from them. The main objective of this project is to recognize the target numbers in the captcha images correctly.

The mainstream CAPTCHA is based on visual representation, including images such as letters and text. Traditional CAPTCHA recognition includes three steps: image pre-processing, 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 pre-processing and artificial design features in traditional recognition technology. Although CNN has achieved certain results, the recognition effect of complex CAPTCHA is insufficient

**Dataset**

The dataset contains CAPTCHA images. The images are **5 letter** words, and have noise applied (blur and a line). They are of size **200 x 50**. The file name is same as the image letters.

Link for the dataset: [CAPTCHA Images | Kaggle](https://www.kaggle.com/fournierp/captcha-version-2-images)

****

**Image Pre-Processing**

3 transformations have been applied to the data:

1. Adaptive Thresholding
2. Morphological transformations
3. Gaussian blurring

**1) Adaptive Thresholding**

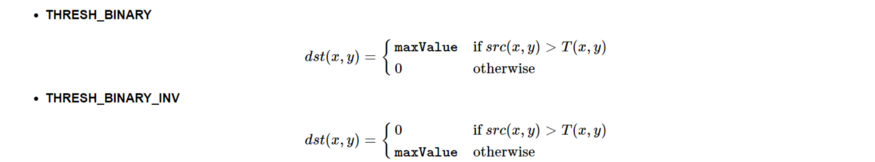
Thresholding is the process of converting a grayscale image to a binary image (an image that contains only black and white pixels). This process is explained in the steps below:

* A threshold value is determined according to the requirements (Say 128).
* The pixels of the grayscale image with values greater than the threshold (>128) are replaced with pixels of maximum pixel value(255).
* The pixels of the grayscale image with values lesser than the threshold (<128) are replaced with pixels of minimum pixel value(0).

But this method doesn’t perform well on all images, especially when the image has different lighting conditions in different areas. In such cases, we go for adaptive thresholding. In adaptive thresholding the threshold value for each pixel is determined individually based on a small region around it. Thus we get different thresholds for different regions of the image and so this method performs well on images with varying illumination.

The steps involved in calculating the pixel value for each of the pixels in the thresholded image are as follows:

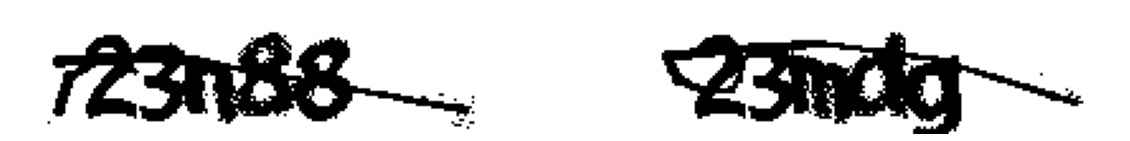
* The threshold value T(x,y) is calculated by taking the mean of the blockSize×blockSize neighborhood of (x,y) and subtracting it by C (Constant subtracted from the mean or weighted mean).
* Then depending on the threshold type passed, either one of the following operations in the below image is performed:



OpenCV provides us the adaptive threshold function to perform adaptive thresholding :

Thres\_img=cv.adaptiveThreshold ( src, maxValue, adaptiveMethod, thresholdType, blockSize, C)

Image after applying adaptive thresholding :



**2) Morphological Transformations**

Morphological transformations are some simple operations based on the image shape. It is normally performed on binary images. Two basic morphological operators are Erosion and Dilation. Then its variant forms like Opening, Closing, Gradient etc also comes into play.

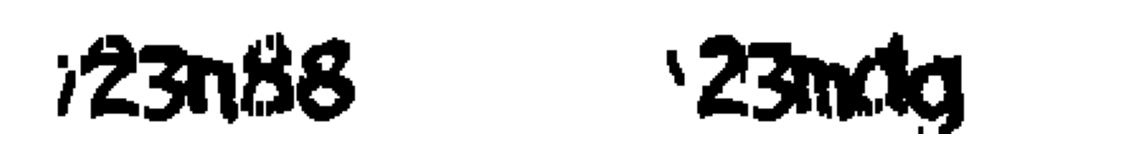
For this project I have used its variant form closing, **closing is a dilation followed by an erosion.** As the name suggests, a closing is used to close holes inside of objects or for connecting components together.

An **erosion** in an image “erodes” the foreground object and makes it smaller. A foreground pixel in the input image will be kept **only if all pixels** inside the structuring element are > 0. Otherwise, the pixels are set to 0 (i.e., background). Erosion is useful for removing small blobs in an image or disconnecting two connected objects.

The opposite of an erosion is a **dilation**. Just like an erosion will eat away at the foreground pixels, a dilation will grow the foreground pixels. Dilations increase the size of foreground objects and are especially useful for joining broken parts of an image together.

Performing the closing operation is again accomplished by making a call to cv2.morphologyEx, but this time we are going to indicate that our morphological operation is a closing by specifying the cv2.MORPH\_CLOSE.

Image after applying morphological transformation:

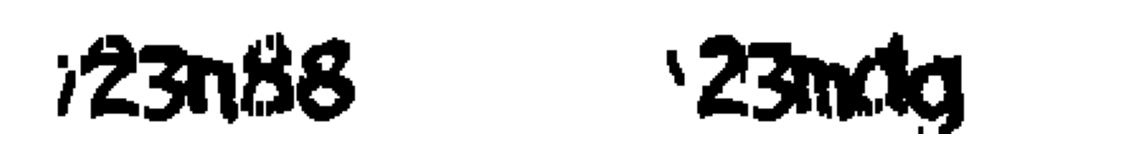


**3) Gaussian Blurring**

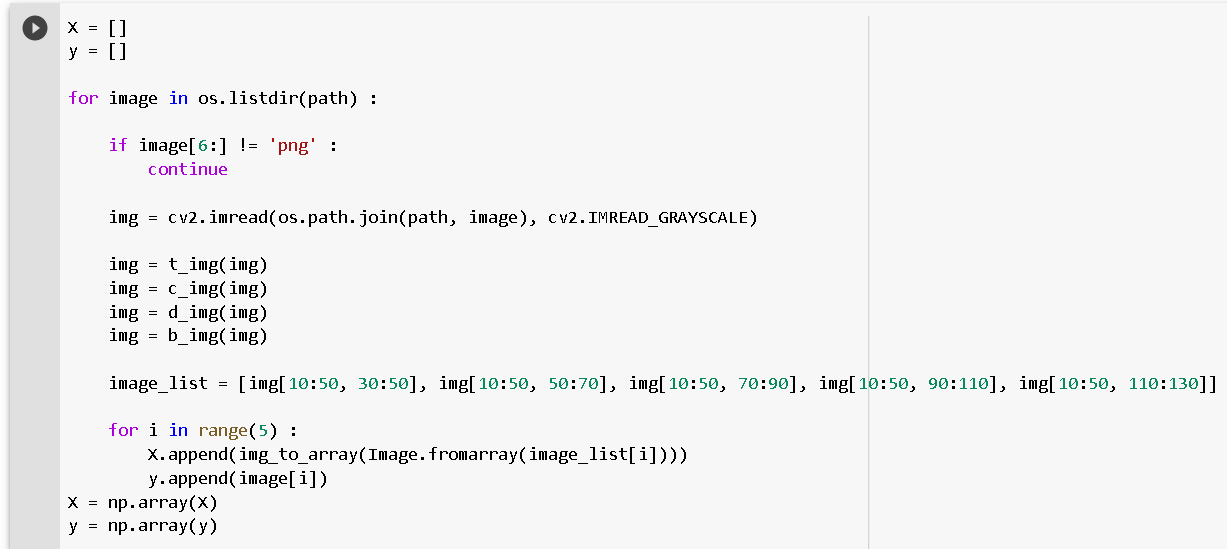
Gaussian smoothing is used to remove noise that approximately follows a Gaussian distribution. The end result is that our image is less blurred, but more “naturally blurred,” than using the average in average blurring. Furthermore, based on this weighting we’ll be able to preserve more of the edges in our image as compared to average smoothing.

**Gaussian blurring** is similar to average blurring, but instead of using a simple mean, we are now using a weighted mean, where neighbourhood pixels that are closer to the central pixel contribute more “weight” to the average. Gaussian smoothing uses a kernel of *M X N*, where both *M* and *N* are odd integers.

Image after applying Gaussian blurring:



After applying all these image pre-processing techniques, images have been converted into n-dimension array

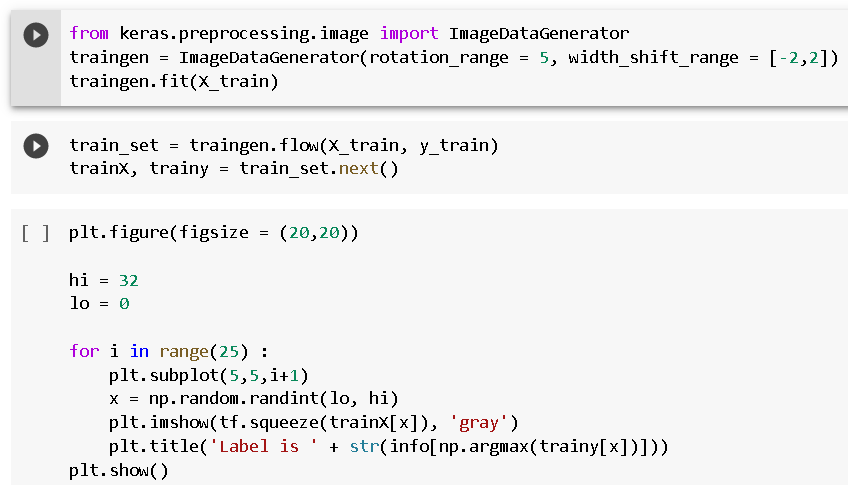


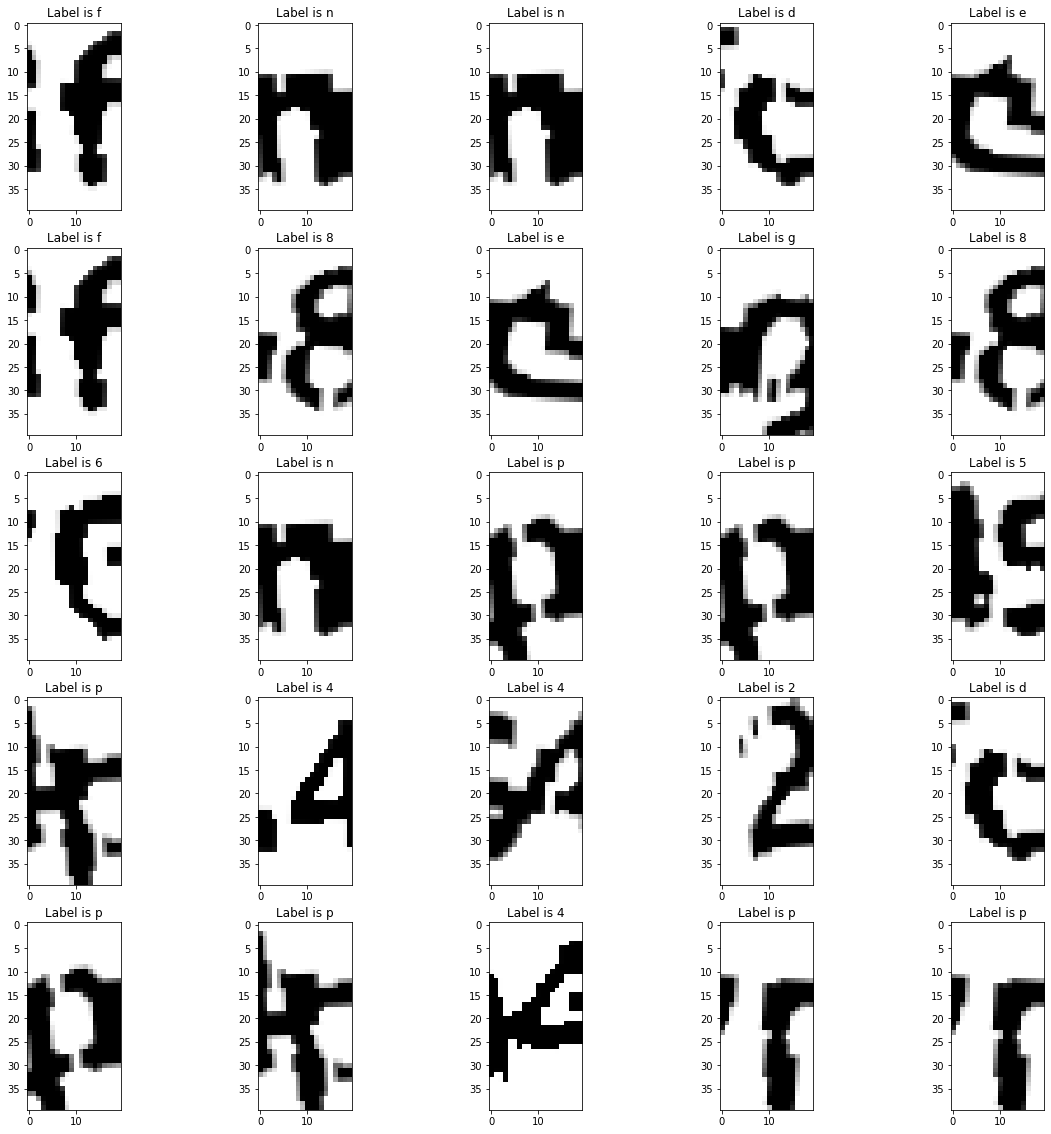
Further 2 more transformations have been applied on this n-dimensional array. The pixel values initially range from 0-255. They are first brought to 0-1 range by dividing all pixel values by 255. Then, they are normalized.

Then, the data is shuffled and splitted into training and validation sets. Since the number of samples is not big enough and in deep learning we need large amounts of data and in some cases it is not feasible to collect thousands or millions of images, so **data augmentation** comes to the rescue.

**Data Augmentation** is a technique that can be used to artificially expand the size of a training set by creating modified data from the existing one. It is a good practice to use **data augmentation** if you want to prevent **overfitting**, or the initial dataset is too small to train on, or even if you want to squeeze better performance from your model. In general, data augmentation is frequently used when building a deep learning model.

To augment images when using **Keras** as our **deep learning** framework we can use **ImageDataGenerator** (**tf.keras.preprocessing.image.ImageDataGenerator**) that generates batches of tensor images with real-time **data augmentation**.





**Modelling and Training**

The CNN architecture developed contains the following layers:

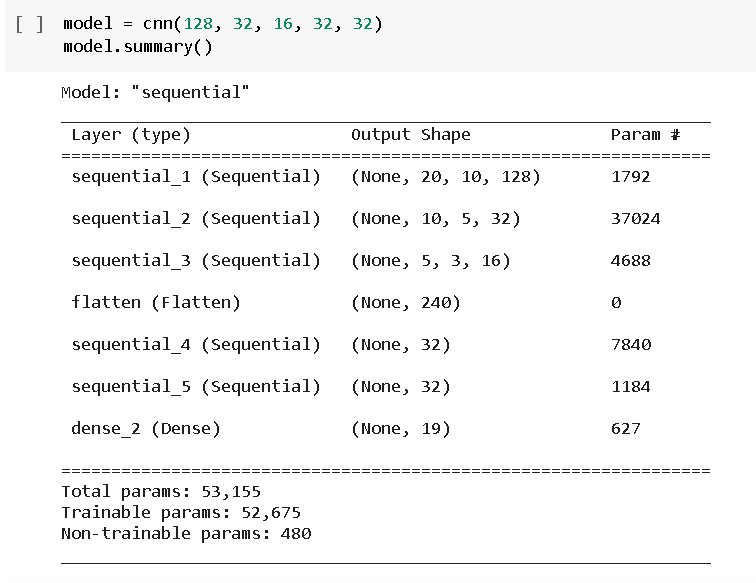
1. **Input Layer:** It’s the layer in which we give input to our model. The number of neurons in this layer is equal to total number of features in our data i.e. 40X20X1 (number of pixels incase of an image).
2. **Hidden Layers:** The input from Input layer is then feed into the hidden layers. Here, I have used 3 hidden convolutional blocks. Each block contains a convolutional layer, a batch normalization layer (Batch normalization is a technique for training very deep neural networks that standardizes the inputs to a layer for each mini-batch. This has the effect of stabilizing the learning process and dramatically reducing the number of training epochs required to train deep networks), a dropout layer with probability 0.2 and a max pooling layer.
3. **Dense Layers**: First the output of the convolutional layer is flatten and then passed into the dense layer with 32 nodes. Then this is connected to another dense layer with 32 nodes.

The output from each layer is computed by matrix multiplication of output of the previous layer with learnable weights of that layer and then by addition of learnable biases followed by activation function which makes the network nonlinear.

1. **Output Layer:** The output from the dense layer is then passed into the output layer with 19 nodes (each node corresponding to a label). The activation function used here is softmax which converts the output of each class into probability score of each class.

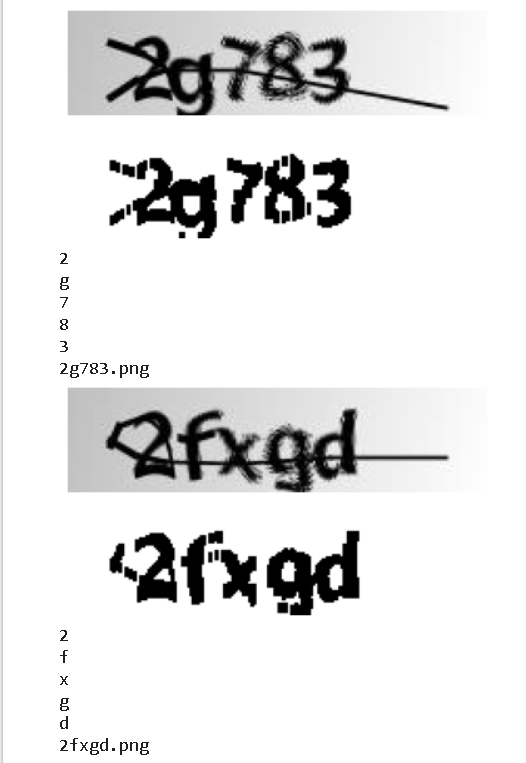
The activation function used throughout is Rectified Linear Unit (ReLU), because of its known efficient performance and fast convergence. The loss function is categorical cross-entropy (since it is a multi-class classification problem) and the optimizer being Adam. Adam is chosen because of its more efficient performance than simple optimizers like Stochastic Gradient Descent.

Model Checkpoint has been used to retain the best perfroming model (in terms of loss) , and ReduceLROnPlateau to reduce the learning in case the model stops improving.

****

**Testing**

A helper function has been made to test the model on test data in which image pre-processing and transformations have been applied to get the final output



**Result**

The model achieves:

1. Accuracy = 89.13%
2. Precision = 91%
3. Recall = 90%
4. F1-score= 90%

Below is the full report:

