DEEP LEARNING BASED CAPTCHA SOLVER

PRESENTED TO - DR. MANJUSHA PANDEY

TEAM MEMBERS -

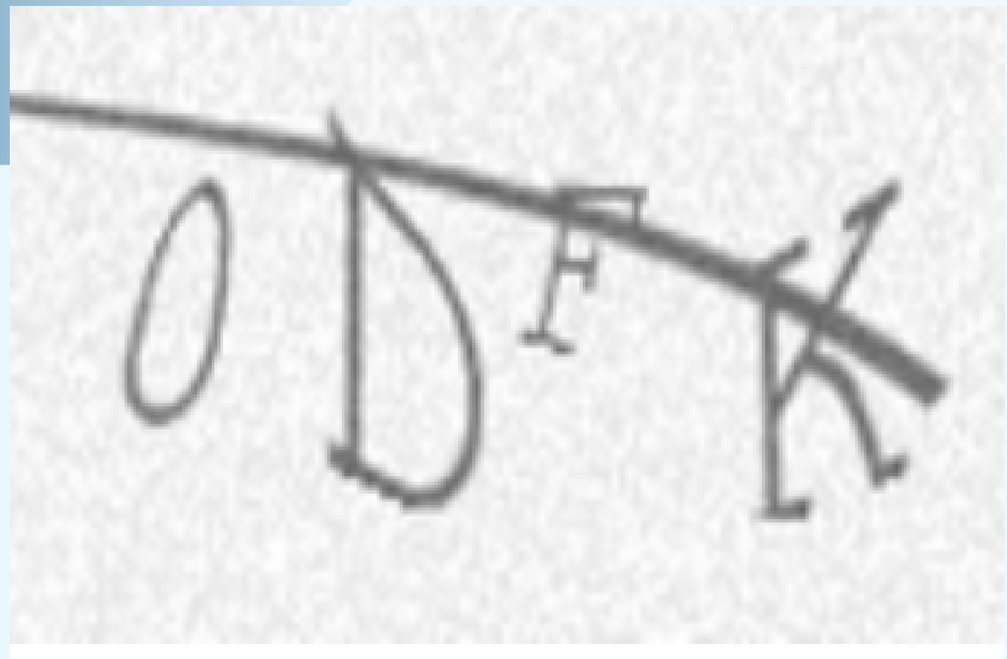
AMIT PRAKASH	21051236
RAJARSHI DEY	21051237
ANUSMITA SAMANTA	21051207
KUMAR SNEHDEEP	21051224

CONTENTS

INTRODUCTION

Captchas, essential for distinguishing humans from bots online, emerged as vital security measures against automated abuse. Despite their importance, captchas pose challenges like user accessibility issues and potential vulnerabilities. Automating captcha solving addresses these concerns, enhancing user experience and facilitating automated systems. Responsible development ensures a balance between security and accessibility, driving the evolution of online systems.

PROBLEM STATEMENT



- There are many important websites which still use the traditional Captcha which is a combination of Letters and Numbers and end up being at the risk of their data being updated automatically by bots
- Captchas have defeated machines but things are changing rapidly as technology improves. Hence research into Optical Character Recognition(OCR) is required to strengthen Captchas against machine based attacks.
- The goal of this project is to accurately classify the captchas which is a combination of letters and numbers along with some ‘noise’ included

METHODOLOGY

1. Approach
2. Data Generation
3. Data Segmentation
4. Data Exploration
5. CNN Architecture
6. Training the Model
7. Testing the model
8. Insights

APPROACH

- For a 4 sequence captcha there is 36x36x36x36 combinations that is **1,679,616** unique combinations increasing captcha length increases combinations.
- To train captchas directly with Neural networks would require us to generate huge amount of data and computing resources to train the model.
- Alternative approach is to make the model identify characters rather than mapping entire captcha.

DATA GENERATION

- We wrote a random character generator and used ImageCaptcha library in python to generate our own set of Captchas
- Captcha we have generated is of a 4 character sequence and has random noise

Data Generation-- Generating Captchas

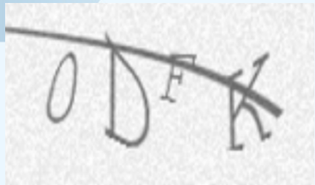
In [238]:
```
def randomString():
    '''random string generator to generate captchas with letters and numbers'''
    rndLetters = (random.choice(string.ascii_uppercase+string.digits) for _ in r
    return "".join(rndLetters)
```

In [ ]:
```
captcha_text = []
## Referenced from https://github.com/kuszaj/claptcha
## Generating captchas by mentioning the count as a range- Download and use the
n = 2000 #number of images to be generated
for i in range(1,n):
    text = randomString()
    # Below code line is used from https://github.com/kuszaj/claptcha
    c = Claptcha(text, "FreeMono.ttf", (150,90),
             resample=Image.BICUBIC, noise=0.2)
    c.margin = (25,25)
    text, _ = c.write('Generated_captchas\\'+text+'.png')
    captcha_text.append(text)
```

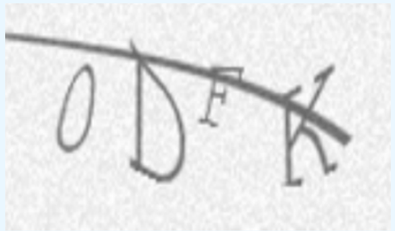
In [13]:
```
## Saving the captchas texts separately as they are the labels for our identific
a = pd.Series(captcha_text)
a.to_csv('Captcha_label.csv')
```

DATA SEGMENTATION

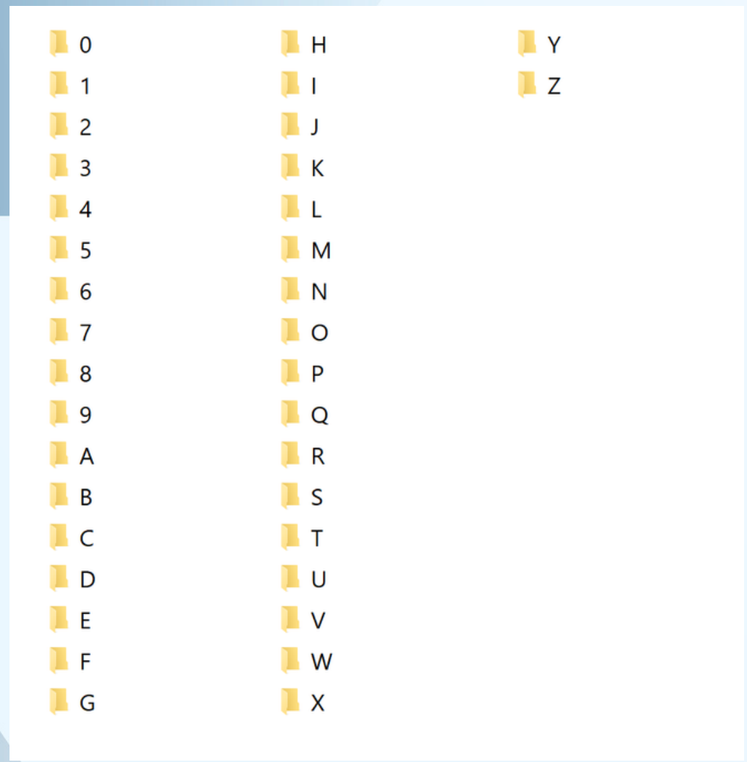
- Because of the random line which is generated, spliting into characters based on contour technique is arduous.



- We binarized the image using(black or white) using OTSU thresholding to remove background noise and then split the captcha into segments based on positioning and spacing of the characters in the captcha.




DATA EXPLORATION



- We have generated our labeled data by splitting our captchas into characters
- Data contains 36 characters 0-9 and A-Z with random noise in the images to make our classification complicated

CNN ARCHITECTURE

- Implemented using Keras library
- Model used - Sequential
- Features – 32
- Convolution
- Maxpooling
- Flattening
- Dense layer
- Optimizer - Adam

CNN Architecture

```
## Below architecture was chosen after iterations and based on https://elitedatascience.com/keras-
classifier = Sequential()
classifier.add(Conv2D(32, (5, 5), input_shape = (32, 32, 1), activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (2, 2),strides=(2,2)))
classifier.add(Conv2D(32, (5, 5), activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (2, 2),strides=(2,2)))
classifier.add(Flatten())
classifier.add(Dense(units = 500, activation = 'relu'))
classifier.add(Dense(units = 36, activation = 'softmax'))
# Compiling the CNN
classifier.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
```

```
## Fitting the model
classifier.fit(training_set,steps_per_epoch=1000,epochs=10)
```

TRAINING THE MODEL

```
train_datagen = ImageDataGenerator(rescale = 1./255)
```

```
## Training data generation - from directory
training_set = train_datagen.flow_from_directory('Preprocessed/',
target_size = (32, 32),
class_mode = 'categorical',color_mode='grayscale')
```

```
Found 3970 images belonging to 36 classes.
```

```
## Printing the different classes - 0 to 35 classes - refers to 0-9 and A-Z
training_set.classes
```

```
array([ 0,  0,  0, ..., 35, 35, 35])
```

```
## Fitting the model
classifier.fit_generator(training_set,steps_per_epoch=2000,epochs=10)
```

```
Epoch 1/10
2000/2000 [==============================] - 133s - loss: 0.4045 - acc: 0.8867
Epoch 2/10
2000/2000 [==============================] - 128s - loss: 0.0043 - acc: 0.9997
Epoch 3/10
2000/2000 [==============================] - 129s - loss: 0.0041 - acc: 0.9997
```

- Rescaled and standardized the image using image data generator
- Fitting the model using 10 epochs
- Batch size – 32
- Saved the trained model built in hierarchal data format

TESTING THE MODEL

- Loaded the save model
- Processed the test images – Grey scaling, thresholding and slicing
- Achieved overall model accuracy of 98%

Check the trained model on Test data

```
# Load the saved model
captcha_model = load_model('captchaclassifier.hdf5')
```

```
#Get list of all files in a directory
captcha_image_files =  glob.glob(os.path.join('Test_data', "*
```

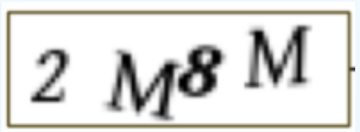
```
dict_class = {v:k for k,v in d.items()}
```

```
## Predicting using the trained model
```

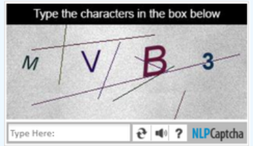
```
# Counter to count the number of correct predictions
count=0
for file in captcha_image_files:
    combine = []
    #Get the file name
    mystr = file.split('\\')
```

INSIGHTS

- The applications containing Captchas with no ‘noise’ included are at the highest risk of being automated.



- The Captchas with noise such as lines across the text are at a medium risk of a breach.



- Observing the different captchas and its history we can find that there is always a space separating the letters in a captcha which makes these captchas vulnerable for machines to break them.
- If the intensity of the noise is less than the actual text and if not converging on the text, the noise can be removed by thresholding or binarizing

RESULTS

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
4PAI
4PAI
------------
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
4QMD
4QMD
------------
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
4QNV
4QNV
------------
```

```
VDEV
VDEV
------------
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
VEYO
VEDF
------------
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
VEE3
VEE3
------------
```

Metrics - Accuracy of prediction

```
## Number of correct predictions is stored in count variable
Accuracy = count/len(captcha_image_files)
print(Accuracy)
```

```
0.8118118118118118
```

ADVANTAGES

1. Improved User Experience
2. Increased Accessibility
3. Efficient Automated Systems
4. Advancement of Technology
5. Enhanced Security

DISADVANTAGES

1. Potential Misuse
2. Ethical Considerations
3. Legal Implications

CONCLUSION AND FUTURE SCOPE

In conclusion, our research introduces Deep-CAPTCHA, a groundbreaking approach leveraging convolutional neural networks to automate CAPTCHA solving. This innovation significantly improves user experience, accessibility, and system efficiency in online interactions. However, ethical and legal considerations, alongside potential misuse, underscore the need for responsible development and regulatory oversight. Looking forward, further advancements in algorithm refinement and collaboration with stakeholders promise to enhance security while maintaining accessibility in the digital landscape.

CONTRIBUTIONS

AMIT PRAKASH -

- Used Conv2D for feature extraction.
- Applied MaxPooling2D for dimension reduction.
- Added Flatten layer for data preparation.
- include Dense layers for classification.
- Employed ImageDataGenerator.
- Set categorical class mode.
- Specified target size for resizing.
- Accessed classes attribute for exploration.

RAJARSHI DEY

- Defined CNN model architecture involving the combination of various CNN architecture.
- Compiled model with Adam optimizer and categorical cross-entropy loss.
- Saved trained model to Keras compatible file.
- Loaded saved model.
- Loaded test image files.
- Created dictionary mapping class indices to labels.

KUMAR SNEHDEEP

- Loop through test images.
- Preprocess each image.
- Segment image into character regions.
- Recognize characters using the trained model.
- Combine predicted characters into a string.
- Compare with actual value and count correct predictions.
- Print predictions and actual values.

ANUSMITA SAMANTA

- Generated random strings for captcha text..
- Processed captcha images by converting to grayscale and applying thresholding technique.
- Segmented captcha images into character regions.
- Saved individual character images into folders based on their labels.
- Defined a function to save preprocessed images.
- Iterated through captcha images, processed them, and saved character images accordingly.