International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 06 Issue: 09 | September - 2022 Impact Factor: 7.185 ISSN: 2582-3930

Recognizing CAPTCHA using Neural Networks

Santanu Panda

1. ABSTRACT

CAPTCHAs (Completely Automated Public Turing test to tell Computers and Humans Apart) are distortions of texts or images so that they are still recognizable to most humans but can become difficult for a computer to recognize. Additionally, many CAPTCHAs also inject noise or additional structures such as blobs or lines into the image to further make it difficult to recognize. They are primarily employed as security measures on websites to prevent bots from accessing or performing transactions on the site.

On the other hand, machine learning (ML) algorithms, in particular deep learning (DL) neural networks have been trained with significant success on similar problems such as handwritten digit recognition. This motivates us to build an ML-based CAPTCHA breaker that maps CAPTCHAs to their solutions.

2. INTRODUCTION

The Completely Automated Public Turing test to tell Computers and Human Apart (CAPTCHA) is a type of test to differentiate between humans and computer programs on Internet websites. CAPTCHA attempts to provide security against bots and can appear in many forms, including text, image, audio and video. Conducting research on recognizing CAPTCHA images is important because it helps identify weak points and loopholes in the generated CAPTCHAs and consequently leads to the avoidance of these loopholes in newly designed CAPTCHA-generating systems, thus boosting the security of the Internet.

Text-based CAPTCHAs are still a much popular and powerful tool against malicious computer program attacks due to their extensive usability and easy implementation. The majority of text-based CAPTCHAs consist of English uppercase letters (A to Z), English lowercase letters (a to z), and numerals (0 to 9). Other new large character sets, such as Chinese characters, have been used recently in text-based CAPTCHAs . Numerous mechanisms have been developed to secure and strengthen text-based CAPTCHAs including background noise, text distortion, rotating and warping, variable string length, and merging of characters. However, due to the

rapid evolution of deep learning in the past few years, CAPTCHA recognition systems have become more competent than before in breaking most of the current defense mechanisms of text-based CAPTCHAs. As a result, sophisticated security mechanisms need to be developed to make text-based CAPTCHAs more robust against malicious attacks.

Deep learning techniques demonstrate an excellent ability to extract meaningful features from input images and have numerous applications in various areas, such as image restoration and object detection. These powerful characteristics of deep learning techniques make them a good choice for building robust CAPTCHA recognition networks to perform attacks against text-based CAPTCHAs. Although several CAPTCHA recognition algorithms use traditional digital image processing techniques in their implementation, these techniques still suffer from drawbacks (e.g., weak feature extraction ability and easily influenced by noise in input images). As a result, these techniques are being gradually replaced by the powerful deep learning approaches.

In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most applied to analyze visual imagery. CNNs are regularized versions of multilayer perceptrons. Multilayer perceptrons usually mean fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer. The "full connectivity" of these networks make them prone to overfitting data. Typical ways of regularization, or preventing overfitting, include: penalizing parameters during training (such as weight decay) or trimming connectivity (skipped connections, dropout, etc.) CNNs take a different approach towards regularization: they take advantage of the hierarchical pattern in data and assemble patterns of increasing complexity using smaller and simpler patterns embossed in their filters. Therefore, on a scale of connectivity and complexity, CNNs are on the lower extreme.

A random forest is a machine learning technique that’s used to solve regression and classification problems. It utilizes ensemble learning, which is a technique that combines many classifiers to provide solutions to complex problems. A random forest algorithm consists of many decision trees. The ‘forest’ generated by the random forest algorithm is trained through bagging or bootstrap aggregating. Bagging is an ensemble meta-algorithm that improves the accuracy of machine learning algorithms. The (random forest) algorithm establishes the outcome based on

the predictions of the decision trees. It predicts by taking the average or mean of the output from various trees.

Increasing the number of trees increases the precision of the outcome. In this project, we explore a segmentation-based approach that uses Convolutional Neural Network (CNN).

3. LITERATURE REVIEW

1st paper:

(i) Title, Author, Date of publication, and Publisher Information

Y. Shu and Y. Xu, "End-to-End Captcha Recognition Using Deep CNN-RNN Network," 2019 IEEE 3rd Advanced Information Management, Communicates, Electronic and Automation
Control Conference (IMCEC), 2019, pp. 54-58, DOI: 10.1109/IMCEC46724.2019.8983895.

(ii) Methodology

In this paper, an end-to-end deep CNN-RNN network model is constructed by studying the captcha recognition technology, which realizes the recognition of a 4-character text captcha. The CNN-RNN model first constructs a deep residual convolutional neural network based on the residual network structure to accurately extract the input captcha picture features. Then, through the constructed variant RNN network, that is, the two-layer GRU network, the deep internal features of the captcha are extracted, and finally, the output sequence is the 4-character captcha.

(iii) Conclusion

The experimental analysis proves that the model constructed in this paper can achieve a better captcha recognition effect, and the model realized in this paper can also perform better by using the few samples dataset crawled from the online web- site.

2nd paper:

(i) Title, Author, Date of publication and Publisher Information

Y. Hu, L. Chen and J. Cheng, "A CAPTCHA recognition technology based on deep learning," 2018 13th IEEE Conference on Industrial Electronics and Applications (ICIEA), 2018, pp. 617-620, doi: 10.1109/ICIEA.2018.8397789.

(ii) Methodology

This paper proposed a method based on the Convolutional Neural Network (CNN) model to identify CAPTCHA and avoid the traditional image processing technology such as location and segmentation. The adaptive learning rate is introduced to accelerate the convergence rate of the model, and the problem of overfitting and local optimal solution has been solved. The multi task joint training model is used to improve the accuracy and generalization ability of model recognition.

(iii) Conclusion

Experimental results show that the proposed method has a good recognition effect, and the recognition accuracy reaches 96.5%. In the future work, Chinese characters CAPTCHAs recognition can be added.

3rd paper:

(i) Title, Author, Date of publication, and Publisher Information

S. Sachdev, "Breaking CAPTCHA characters using Multi-task Learning CNN and SVM," 2020 4th International Conference on Computational Intelligence and Networks (CINE), 2020, pp. 1-6, DOI: 10.1109/CINE48825.2020.234400.

(ii) Methodology

This paper aims to present similar methods, that aids in breaking security. The paper presents Multi-task Learning CNN (MTL-CNN) and SVM can independently recognize text-based CAPTCHAs. With MTLCNN, no pre-processing of images is required and still, it achieves promising results.

(iii) Conclusion

The proposed models provide reliable means to recognize characters from CAPTCHA. This paper proposes one method where there is no need for segmentation, the whole image is processed at once by the model. CNN which takes segmented characters as input achieves slightly more accuracy than MLT-CNN. MLT-CNN can achieve this accuracy even without any pre-processing, that is the main advantage.

4th paper:

(i) Title, Author, Date of publication, and Publisher Information

T. Azakami, C. Shibata and R. Uda, "Challenge to Impede Deep Learning against CAPTCHA with Ergonomic Design," 2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC), 2017, pp. 637-642, DOI: 10.1109/COMPSAC.2017.83.

(ii) Methodology

An effective text-based CAPTCHA algorithm with amodal completion was proposed by the authors. Their CAPTCHA causes computers a large number of calculation costs while amodal completion helps human beings to recognize characters momentarily. Their CAPTCHA has evolved with aftereffects and combinations of complementary colors. They also evaluated their CAPTCHA with deep learning which is attracting the most attention since deep learning is faster and more accurate than existing methods for recognition with computers.

(iii) Conclusion

In this paper, the authors put characters in their CAPTCHA with jagged edges in order to decrease the accuracy of recognition by deep learning while keeping the visual performance of human beings. As a result, they succeeded in decreasing the accuracy in some percentages.

5th paper:

(i) Title, Author, Date of publication, and Publisher Information

Recognition of CAPTCHA Characters by Supervised Machine Learning Algorithms,2018, Ondrej Bostik Jan Klecka, Department of Control and Instrumentation, Brno University of Technology Brno, Czech Republic.

(ii) Their Proposed methodology

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 the research focuses on the comparative study of Neural Networks, k-Nearest Neighbour, Support Vector Machines, and Decision Trees implemented in the MATLAB Computing environment. Achieved success rates of all analyzed algorithms overcome 89%. The main difference in the results of used algorithms is within the learning times.

(iii) Conclusion

This work compares commonly used supervised machine learning algorithms for Optical Character Recognition of Captcha codes. Our experiments reveal that all of the used algorithms can classify the objects into the right class with a success rate of around 99%. The main difference between algorithms is in computational costs. The overall winner in both categories combined is the Pattern recognition neural network as it has both good precision and low computational cost.

Use only some machine learning algorithms to compare the accuracy.

6th paper:

(i) Title, Author, Date of publication, and Publisher Information

CAPTCHA recognition based on deep convolutional neural network Jing Wang, Jianhua Qin, Xuyu Xiang, Yun Tan, and Nan Pan College of Computer Science and Information Technology,
24 June 2019, Central South University of Forestry and Technology, 498 shaoshan S Rd, Changsha, 410004, China.

(ii) Methodology

Aiming at the problems of low efficiency and poor accuracy of traditional CAPTCHA recognition methods, they have proposed a more efficient way based on deep convolutional neural networks (CNN). The Dense Convolutional Network (DenseNet) has shown excellent classification performance which adopts a cross-layer connection. Not only does it effectively alleviate the vanishing gradient problem, but also dramatically reduces the number of parameters. However, it also has caused great memory consumption. So they improved and constructed a new DenseNet for CAPTCHA recognition (DFCR). They test the Chinese or English CAPTCHAs experimentally with different numbers of characters.

(iii) Conclusion

Defeating the CAPTCHAs is the most effective way to increase its own safety by finding the deficiency. The deep CNNs act as a more robust and useful method. All in all, using deep learning techniques to enhance the security of CAPTCHAs is a promising direction. In this paper, they constructed a deep CNN, which we referred to as DFCR. They compared its effectiveness with the ResNet-50 and the DenseNet-121. The experimental results showed that the DFCR not only kept compelling advantages but also encouraged feature reuse.

7th paper:

(i) Title, Author, Date of publication, and Publisher Information

CAPTCHA Recognition Using Deep Learning with Attached Binary Images Alaa Thobhani 1, Mingsheng Gao, Ammar Hawbani, Safwan Taher Mohammed Ali and Amr Abdussalam, 17 September 2020, College of Internet of Things (IoT) Engineering, Hohai University, Changzhou Campus, Changzhou 213022, China.

(ii) Methodology

In this paper, they describe the basic idea of their proposed CAPTCHA recognition algorithm. Also, they explain the characteristics of the adopted attached binary images. Then, they present the internal structure of the CAPTCHA recognition CNN and its training parameters.

(iii) Conclusion

The proposed algorithm is evaluated on two schemes of datasets with about 70,000 CAPTCHA images in each scheme. The experimental results show that the proposed algorithm has a relatively higher CAPTCHA recognition accuracy than the state-of-the-art multi-label CNN and RNN models in both CAPTCHA scheme datasets. The storage size of the proposed model is also much smaller than that of both models. The proposed algorithm can be considered a new fundamental algorithm for recognizing CAPTCHAs in addition to the multilabel-CNN, CRNN, and segmentation-based CAPTCHA recognition models.

8th paper:

(i) Title, Author, Date of publication, and Publisher Information

Captcha Automatic Segmentation and Recognition Based on Improved Vertical Projection Lili Zhang, Yuxiang Xi, Xidao Luan, Jingmeng He

(ii) Methodology

In this paper, a method of captcha segmentation based on improved vertical projection can efficiently solve the segmentation problem of different types of conglutination characters in captcha combining both numbers and letters, so as to improve the accuracy of captcha recognition. Firstly, take preprocessing to captcha images including removing interfering background, denoising, and binarization, and getting binary captcha images with less noise. Secondly, apply improved vertical projection method to captcha character segmentation, propose targeted segmentation method to different conglutinate type characters, getting split single character image. Thirdly, take the overlap rate of sample and template character pixels as the matching rate, using the template matching algorithm for recognition.

(iii) Conclusion

With the proposed improved vertical projection method can effectively identify and segment not conglutination, part conglutination, and high conglutination characters, help to improve the accuracy of the captcha recognition. The shortcoming is that the robustness of this method is not strong enough. It is easily affected by previous captcha image preprocessing effects.

9th paper:

(i) Title, Author, Date of publication, and Publisher Information

CAPTCHA Recognition Method Based on CNN with Focal Loss, Zhong Wang and Peibei Shi

(ii) Methodology

The paper proposes a CAPTCHA recognition method based on convolutional neural networks with a 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 pixels.

(iii) Conclusion

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.

10th paper:

(i) Title, Author, Date of publication, and Publisher Information

CAPTCHA Recognition Based on Faster R-CNN Feng-Lin Du , Jia-Xing Li , Zhi Yang , Peng Chen , Bing Wang3, and Jun Zhang

(ii) Methodology

Faster R-CNN is an end-to-end object detection framework based on deep-learning and has no repetitive computation. Faster R-CNN uses the classified network of ImageNet [9] as the front network. Faster R-CNN provides three different network models with different level convolutional layers. In this paper, Faster R-CNN was employed to recognize the CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart). Unlike traditional methods, the proposed method is based on a deep learning object detection framework. By inputting the database into the network and training the Faster R-CNN, the feature map can be obtained through the convolutional layers. The proposed method can recognize the character and its location.

(iii) Conclusion

The experimental results show that this method is feasible and also can get satisfactory results. CAPTCHA recognition method based on Faster R-CNN can recognize CAPTCHA with a high accuracy. This method is different from the traditional recognition method in several steps, this makes the method efficient and accurate.

4. METHODOLOGY

Creating our Dataset

To train any machine learning system, we need training data. The dataset is collected from the WordPress plugin where 10,000 captcha images are generated.

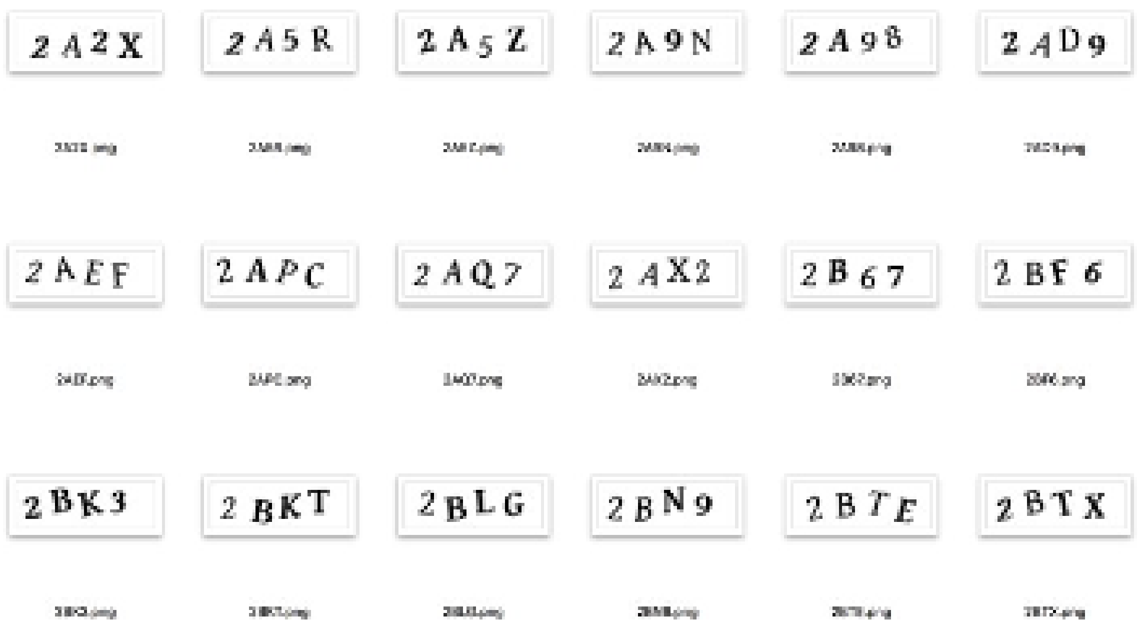


Figure 1: captcha images

Split the image into 4 parts

Luckily the CAPTCHA images are always made up of only four letters. If we can somehow split the image apart so that each letter is a separate image, then we only have to train the neural network to recognize a single letter at a time. In image processing, we often need to detect "blobs" of pixels that have the same color. The boundaries around those continuous pixel blobs are called contours. OpenCV has a built-in findContours() function that we can use to detect these continuous regions.

And then we'll convert the image into pure black and white (this is called thresholding) so that it will be easy to find the continuous regions:

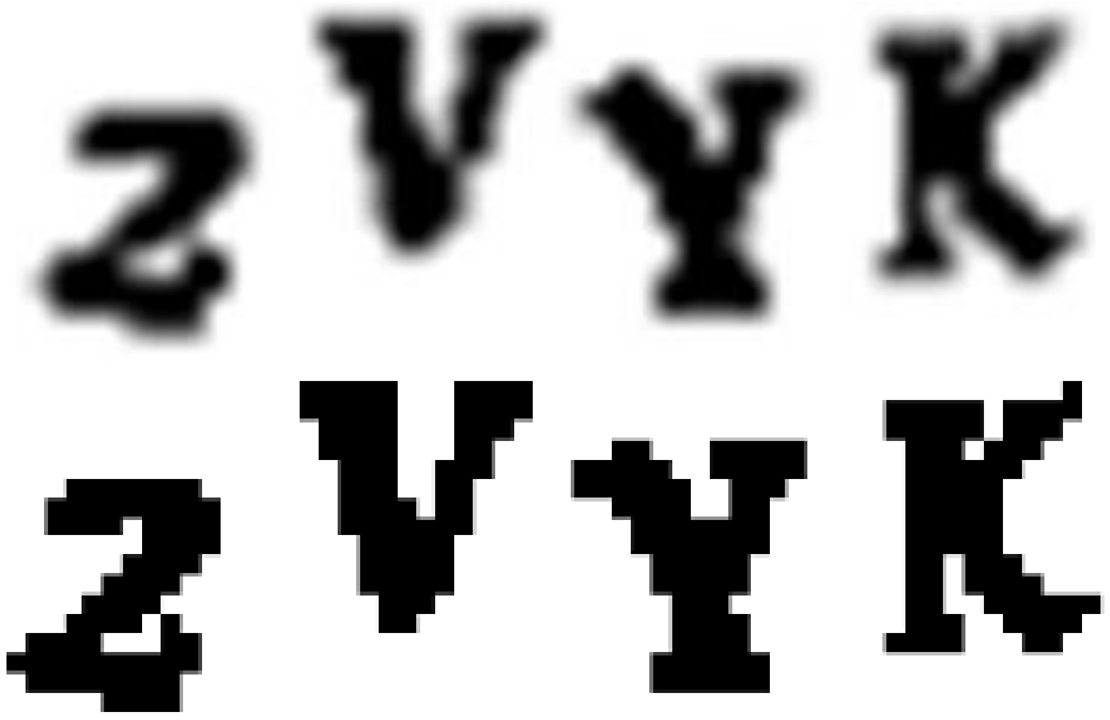


Figure 2: separate letters

Next, we'll use OpenCV's findContours() function to detect the separate parts of the image that contain continuous blobs of pixels of the same color:

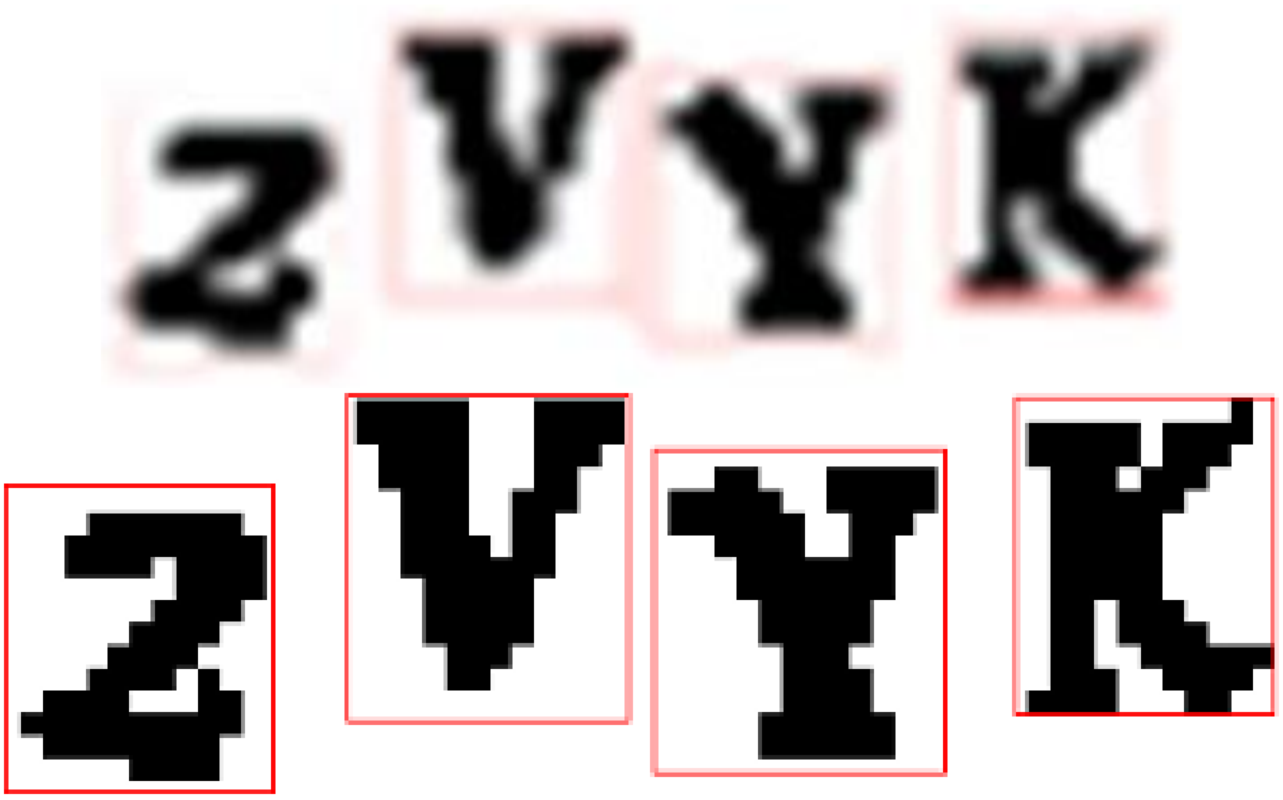


Figure 3: recognising individual letters

Sometimes the CAPTCHAs have overlapping letters like this:

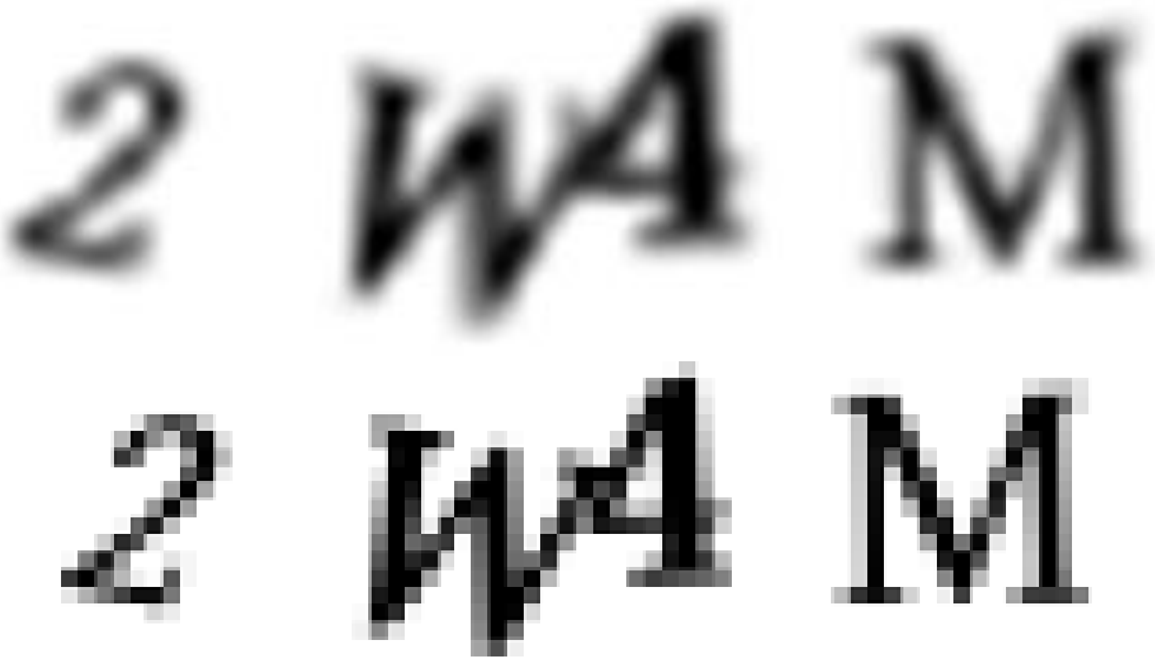


Figure 4: Overlapping letters

That means that we'll end up extracting regions that mash together two letters as one region:

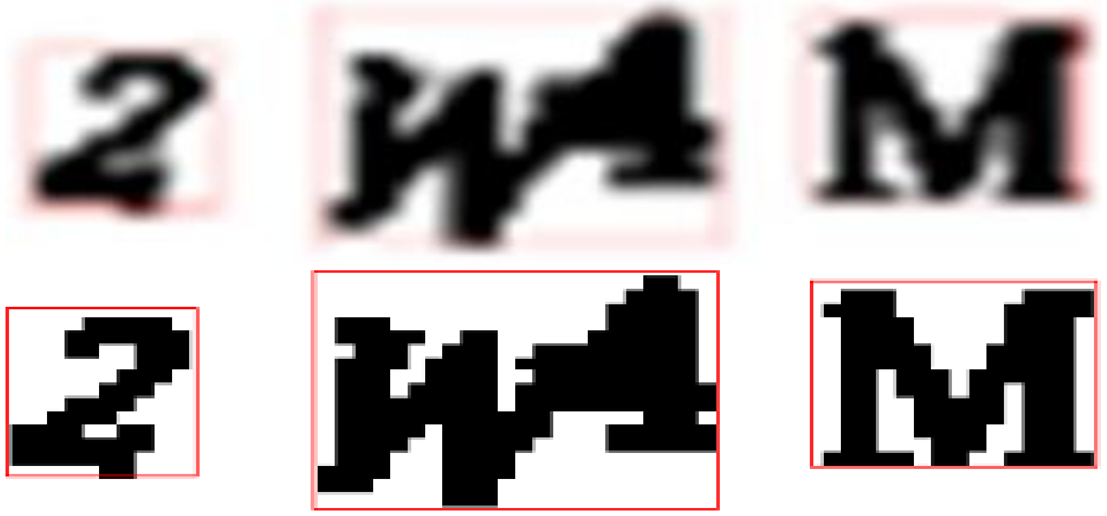


Figure 5: grouping two overlapping letters

If we don’t handle this problem, we’ll end up creating bad training data. We need to fix this so that we don’t accidentally teach the machine to recognize those two squashed-together letters as one letter.

If a single contour area is a lot wider than it is tall, that means we probably have two letters squished together. In that case, we can just split the conjoined letter in half down the middle and treat it as two separate letters:

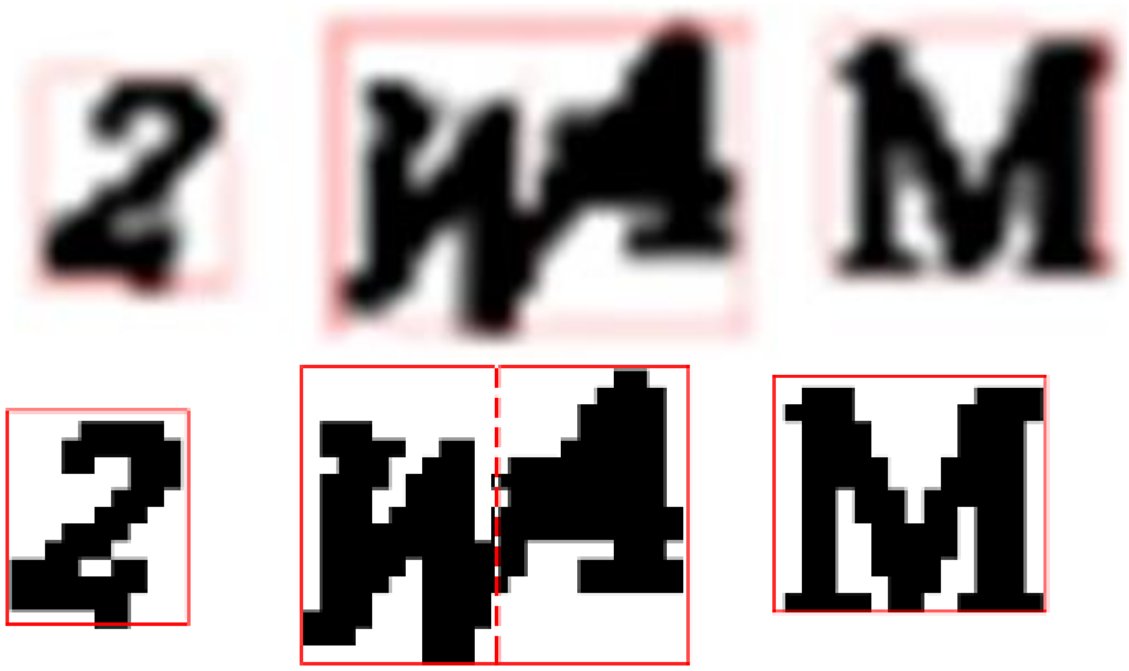


Figure 6: splitting the two overlapping letters

Extract individual characters

Now we have to extract individual letters and store them in their own folder to keep things organized. Here’s a picture of what my “W” folder looked like after I extracted all the letters:

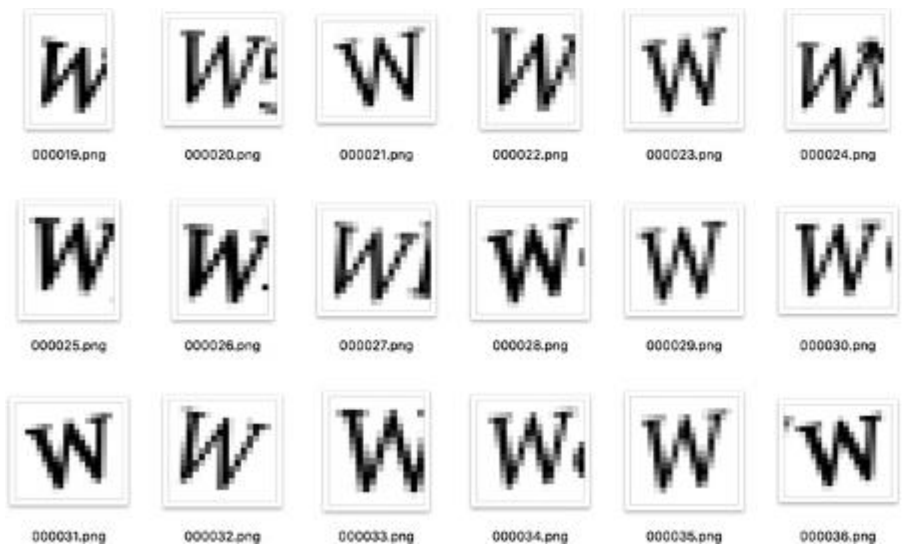


Figure 7: extract individual letters

Building and Training the Neural Network

Since we only need to recognize images of single letters and numbers, we don’t need a very complex neural network architecture. Recognizing letters is a much easier problem than recognizing complex images like pictures like cats and dogs.

We’ll use a simple convolutional neural network architecture with two convolutional layers and two fully connected layers:

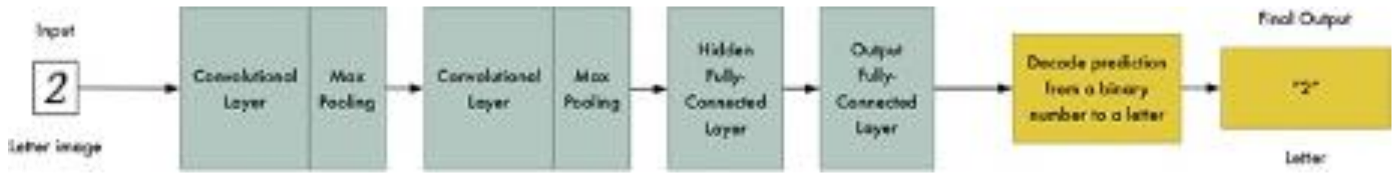

Input
2
Letter image
Convolutional Layer
Max Pooling
Convolutional Layer
Max Pooling
Hidden Fully-Connected Layer
Output Fully-Connected Layer
Decode prediction from a binary number to a letter
Final Output
"2"
Letter

Figure 8: Training neural network

Defining this neural network architecture only takes a few lines of code using Keras.

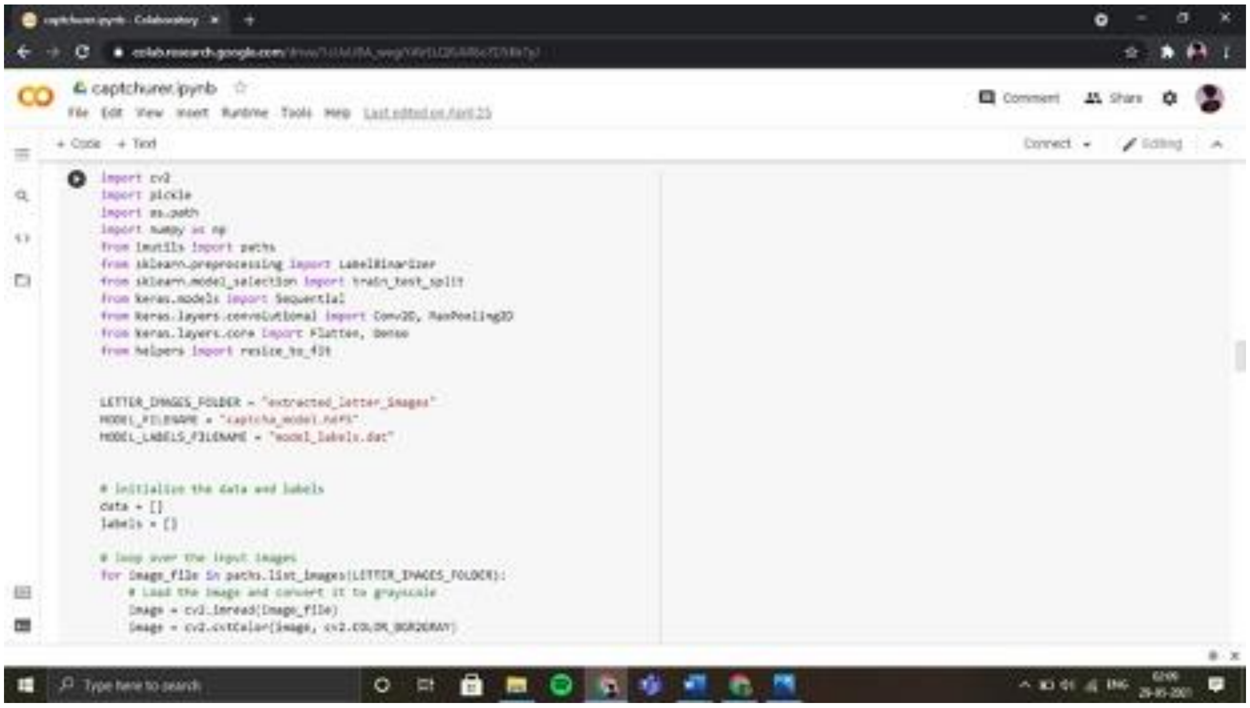
Using the Trained Model to Solve CAPTCHAs

We have a trained neural network, and using it to break a real CAPTCHA is pretty simple:

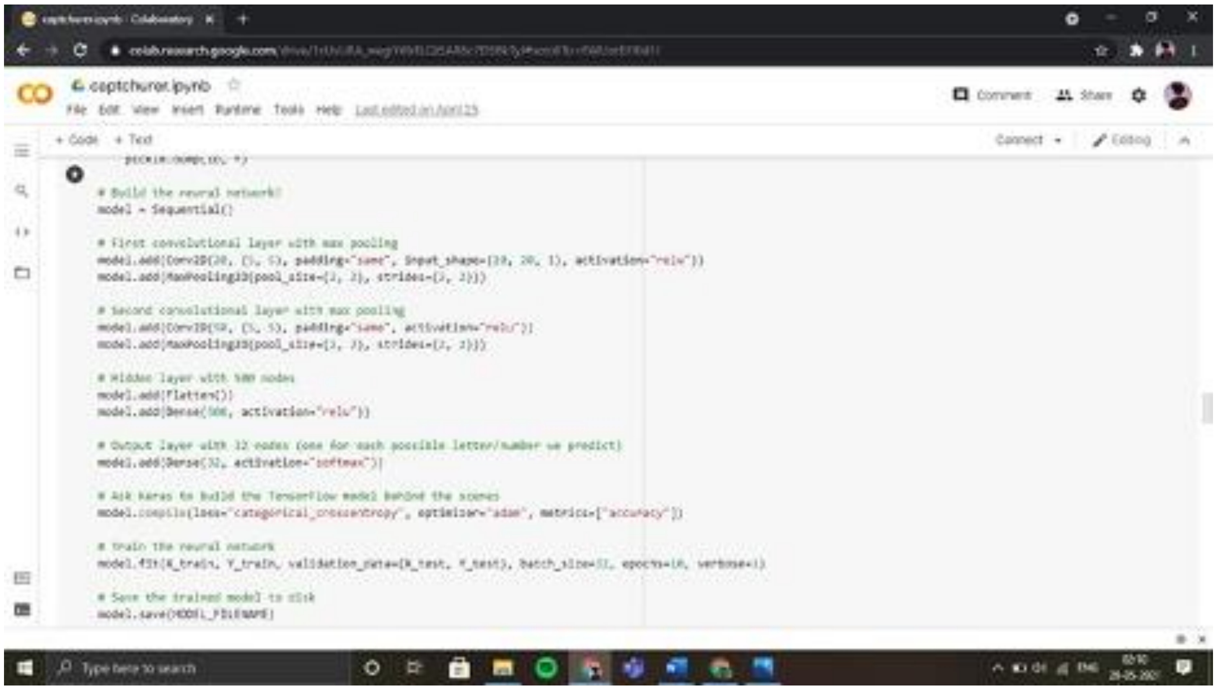
1. Grab a real CAPTCHA image from a website that uses this WordPress plugin.
2. Break up the CAPTCHA image into four separate letter images.
3. Ask our neural network to make a separate prediction for each letter image.
4. Use the four predicted letters as the answer to the CAPTCHA.

5. RESULTS AND DISCUSSION

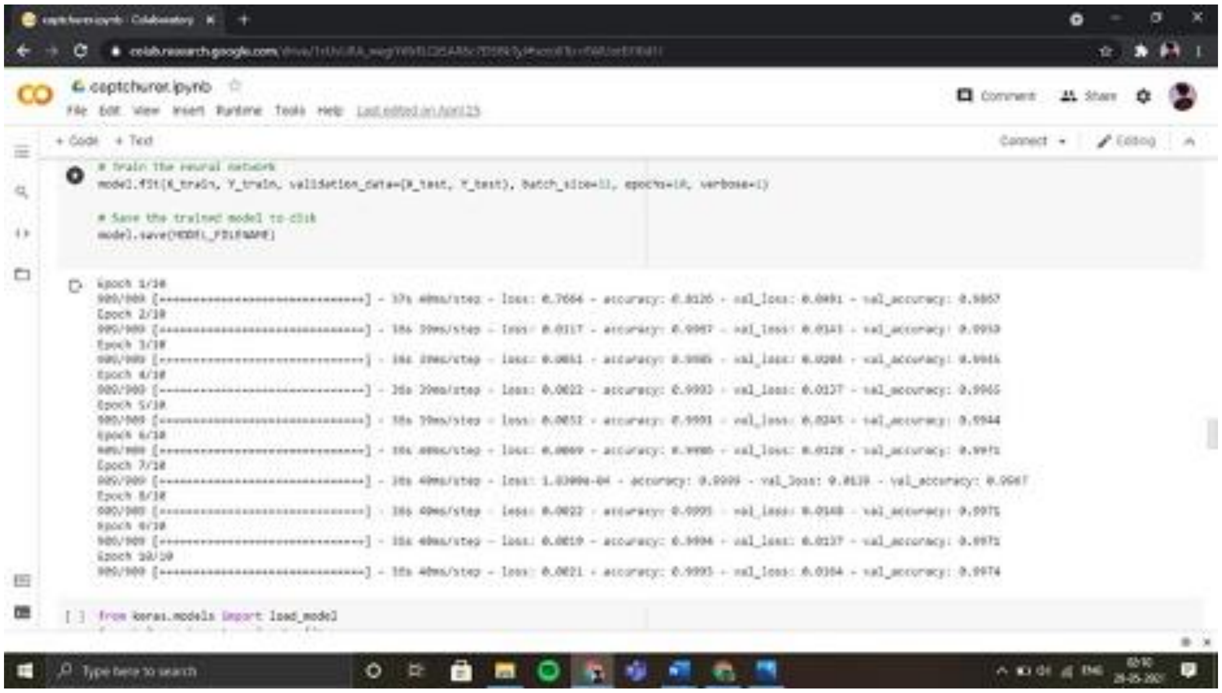
1. Captcha recognition using neural networks:



```
import cv2
import pickle
import os.path
import numpy as np
from imutils import paths
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers.convolutional import Conv2D, MaxPooling2D
from keras.layers.core import Flatten, Dense
from helpers import resize_to_fit


LETTER_IMAGES_FOLDER = "extracted_letter_images"
MODEL_FILENAME = "captcha_model.hdf5"
MODEL_LABELS_FILENAME = "model_labels.dat"


# initialize the data and labels
data = []
labels = []

# loop over the input images
for image_file in paths.list_images(LETTER_IMAGES_FOLDER):
    # Load the image and convert it to grayscale
    image = cv2.imread(image_file)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```



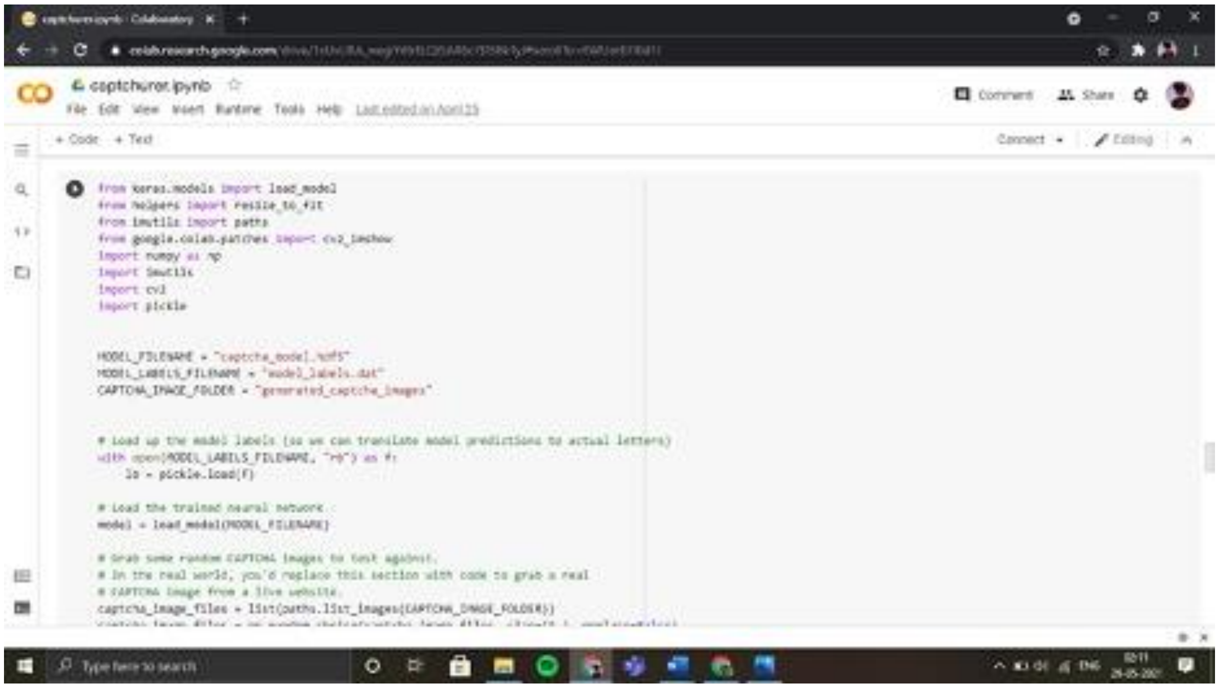
```
    pickle.dump(lb, f)

# Build the neural network!
model = Sequential()

# First convolutional layer with max pooling
model.add(Conv2D(20, (5, 5), padding="same", input_shape=(20, 20, 1), activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

# Second convolutional layer with max pooling
model.add(Conv2D(50, (5, 5), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

# Hidden layer with 500 nodes
model.add(Flatten())
model.add(Dense(500, activation="relu"))

# Output layer with 32 nodes (one for each possible letter/number we predict)
model.add(Dense(32, activation="softmax"))

# Ask Keras to build the TensorFlow model behind the scenes
model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])

# train the neural network
model.fit(X_train, Y_train, validation_data=(X_test, Y_test), batch_size=32, epochs=10, verbose=1)

# Save the trained model to disk
model.save(MODEL_FILENAME)
```



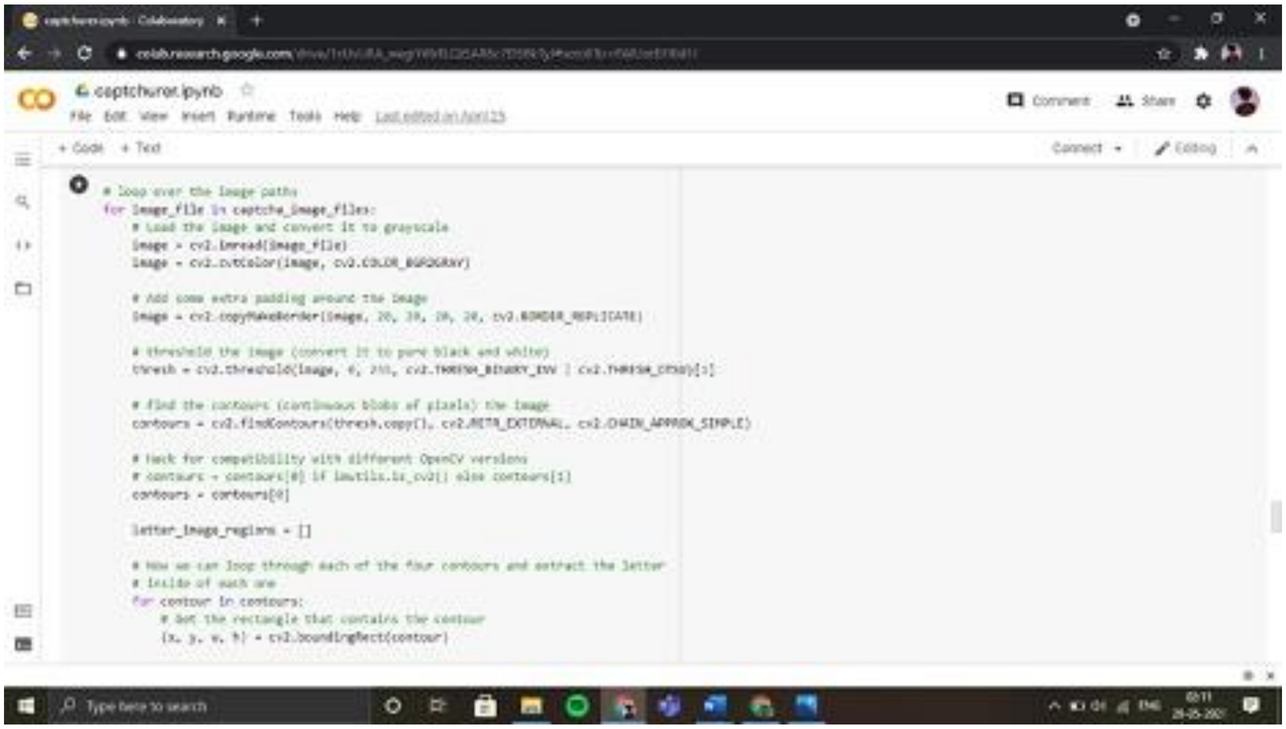
```
# train the neural network
model.fit(X_train, Y_train, validation_data=(X_test, Y_test), batch_size=32, epochs=10, verbose=1)

# Save the trained model to disk
model.save(MODEL_FILENAME)
```

```
Epoch 1/10
909/909 [==============================] - 37s 40ms/step - loss: 0.7664 - accuracy: 0.8126 - val_loss: 0.0491 - val_accuracy: 0.9867
Epoch 2/10
909/909 [==============================] - 36s 39ms/step - loss: 0.0117 - accuracy: 0.9967 - val_loss: 0.0141 - val_accuracy: 0.9953
Epoch 3/10
909/909 [==============================] - 36s 39ms/step - loss: 0.0051 - accuracy: 0.9985 - val_loss: 0.0204 - val_accuracy: 0.9946
Epoch 4/10
909/909 [==============================] - 36s 39ms/step - loss: 0.0022 - accuracy: 0.9993 - val_loss: 0.0137 - val_accuracy: 0.9965
Epoch 5/10
909/909 [==============================] - 36s 39ms/step - loss: 0.0032 - accuracy: 0.9991 - val_loss: 0.0245 - val_accuracy: 0.9944
Epoch 6/10
909/909 [==============================] - 36s 40ms/step - loss: 0.0009 - accuracy: 0.9996 - val_loss: 0.0128 - val_accuracy: 0.9971
Epoch 7/10
909/909 [==============================] - 36s 40ms/step - loss: 1.0300e-04 - accuracy: 0.9999 - val_loss: 0.0139 - val_accuracy: 0.9967
Epoch 8/10
909/909 [==============================] - 36s 40ms/step - loss: 0.0022 - accuracy: 0.9995 - val_loss: 0.0168 - val_accuracy: 0.9975
Epoch 9/10
909/909 [==============================] - 36s 40ms/step - loss: 0.0019 - accuracy: 0.9994 - val_loss: 0.0137 - val_accuracy: 0.9971
Epoch 10/10
909/909 [==============================] - 37s 40ms/step - loss: 0.0021 - accuracy: 0.9993 - val_loss: 0.0164 - val_accuracy: 0.9974
```

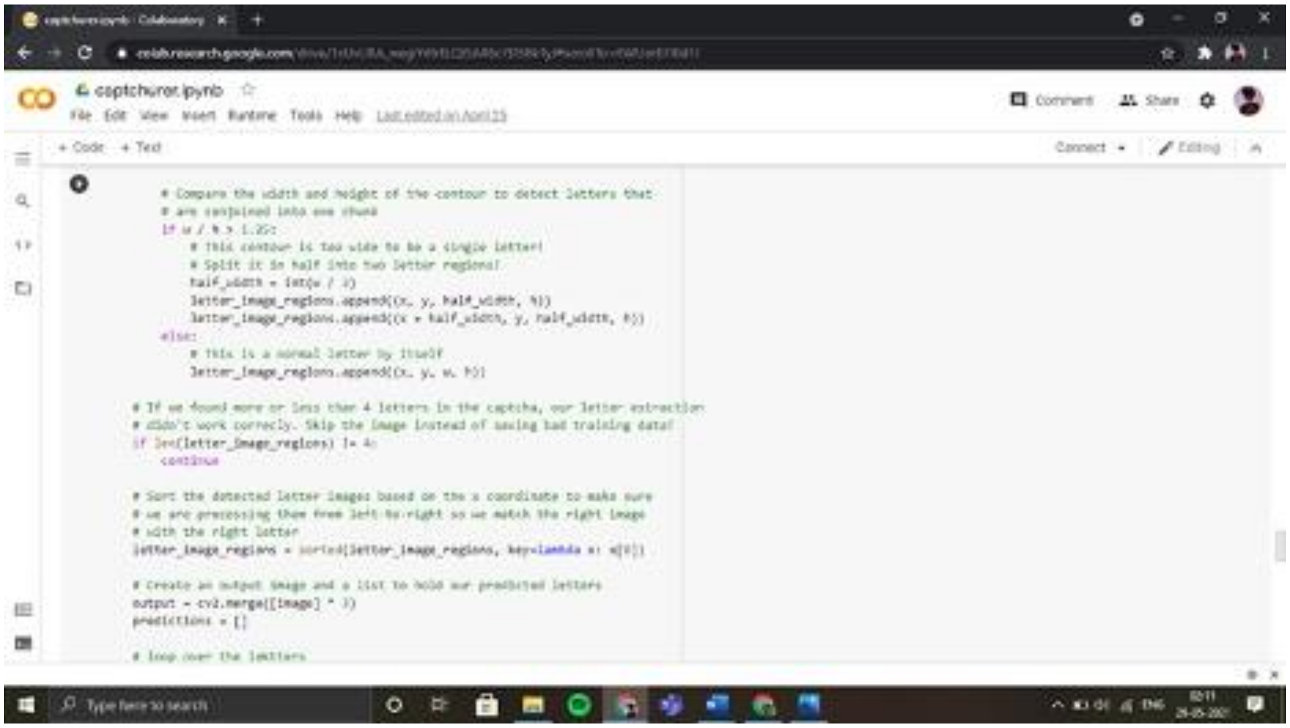
```
from keras.models import load_model
```



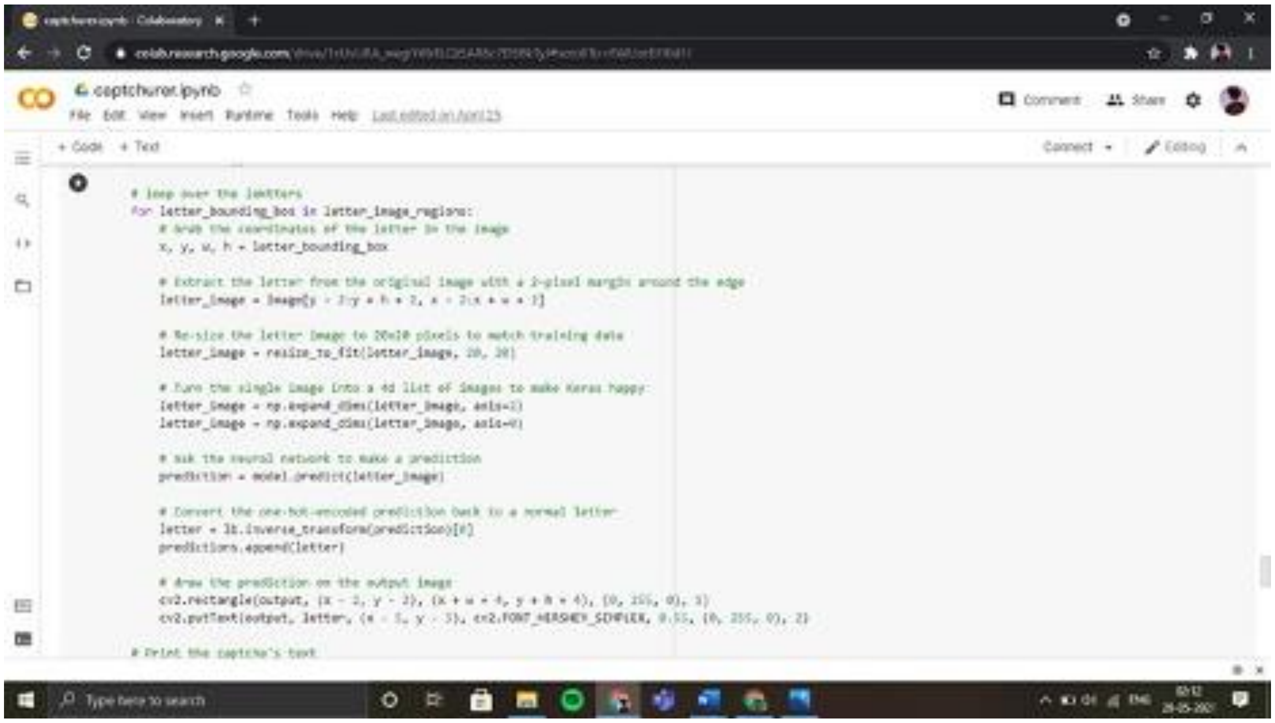
```
from keras.models import load_model
from helpers import resize_to_fit
from imutils import paths
from google.colab.patches import cv2_imshow
import numpy as np
import imutils
import cv2
import pickle


MODEL_FILENAME = "captcha_model.hdf5"
MODEL_LABELS_FILENAME = "model_labels.dat"
CAPTCHA_IMAGE_FOLDER = "generated_captcha_images"


# Load up the model labels (so we can translate model predictions to actual letters)
with open(MODEL_LABELS_FILENAME, "rb") as f:
    lb = pickle.load(f)

# Load the trained neural network
model = load_model(MODEL_FILENAME)

# Grab some random CAPTCHA images to test against.
# In the real world, you'd replace this section with code to grab a real
# CAPTCHA image from a live website.
captcha_image_files = list(paths.list_images(CAPTCHA_IMAGE_FOLDER))
```



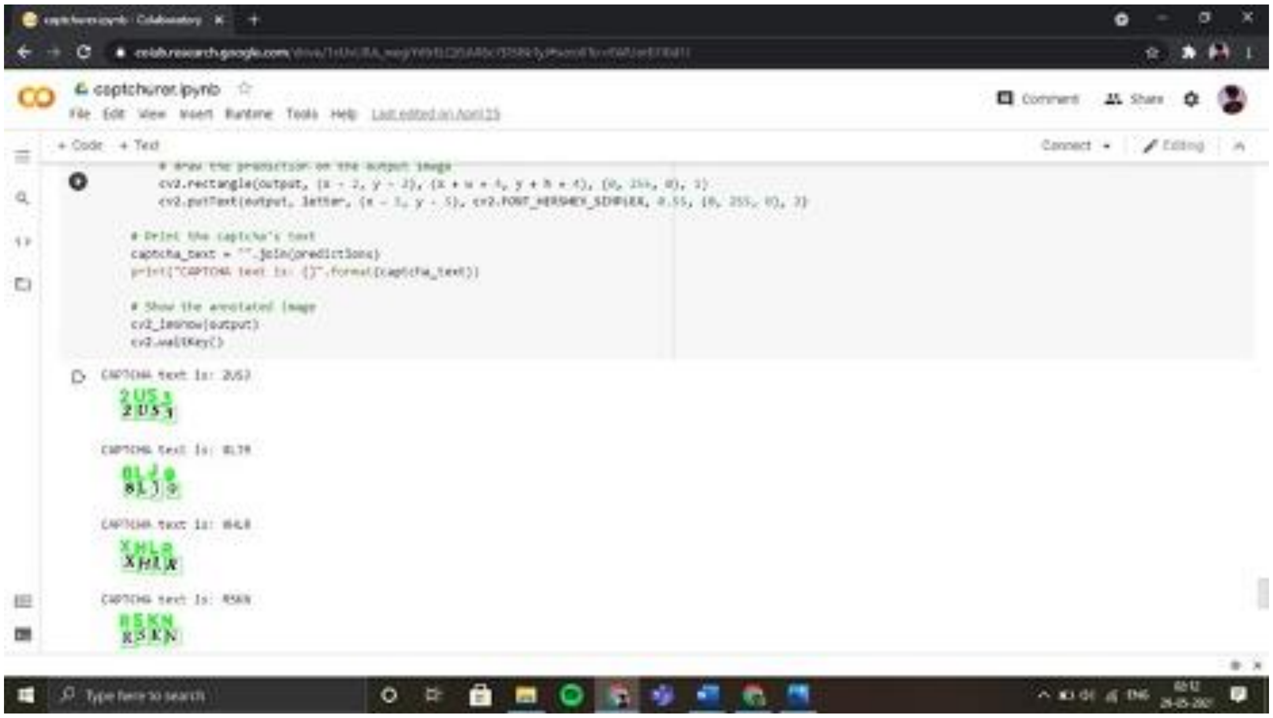
```
# loop over the image paths
for image_file in captcha_image_files:
    # Load the image and convert it to grayscale
    image = cv2.imread(image_file)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Add some extra padding around the image
    image = cv2.copyMakeBorder(image, 20, 20, 20, 20, cv2.BORDER_REPLICATE)

    # threshold the image (convert it to pure black and white)
    thresh = cv2.threshold(image, 0, 255, cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)[1]

    # find the contours (continuous blobs of pixels) the image
    contours = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    # Hack for compatibility with different OpenCV versions
    # contours = contours[0] if imutils.is_cv2() else contours[1]
    contours = contours[0]

    letter_image_regions = []

    # Now we can loop through each of the four contours and extract the letter
    # inside of each one
    for contour in contours:
        # Get the rectangle that contains the contour
        (x, y, w, h) = cv2.boundingRect(contour)
```



```
        # Compare the width and height of the contour to detect letters that
        # are conjoined into one chunk
        if w / h > 1.25:
            # This contour is too wide to be a single letter!
            # Split it in half into two letter regions!
            half_width = int(w / 2)
            letter_image_regions.append((x, y, half_width, h))
            letter_image_regions.append((x + half_width, y, half_width, h))
        else:
            # This is a normal letter by itself
            letter_image_regions.append((x, y, w, h))

    # If we found more or less than 4 letters in the captcha, our letter extraction
    # didn't work correcly. Skip the image instead of saving bad training data!
    if len(letter_image_regions) != 4:
        continue

    # Sort the detected letter images based on the x coordinate to make sure
    # we are processing them from left-to-right so we match the right image
    # with the right letter
    letter_image_regions = sorted(letter_image_regions, key=lambda x: x[0])

    # Create an output image and a list to hold our predicted letters
    output = cv2.merge([image] * 3)
    predictions = []

    # loop over the letters
```

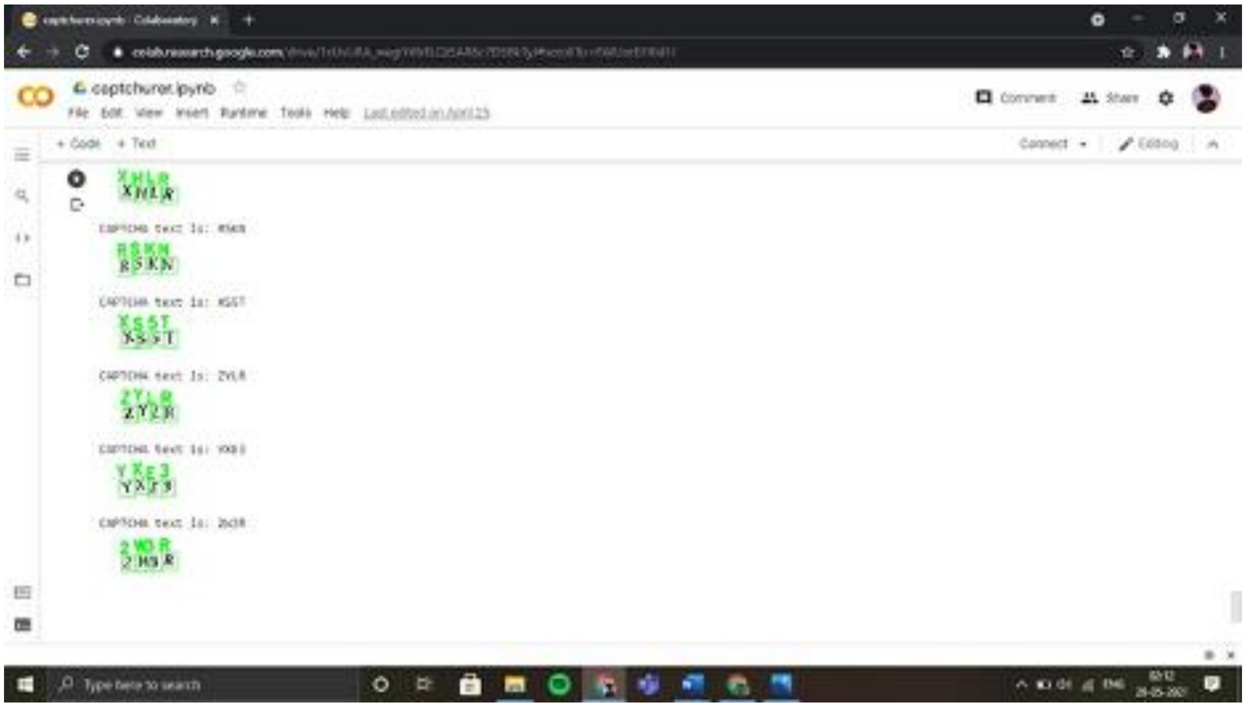


```
    # loop over the letters
    for letter_bounding_box in letter_image_regions:
        # Grab the coordinates of the letter in the image
        x, y, w, h = letter_bounding_box

        # Extract the letter from the original image with a 2-pixel margin around the edge
        letter_image = image[y - 2:y + h + 2, x - 2:x + w + 2]

        # Re-size the letter image to 20x20 pixels to match training data
        letter_image = resize_to_fit(letter_image, 20, 20)

        # Turn the single image into a 4d list of images to make Keras happy
        letter_image = np.expand_dims(letter_image, axis=2)
        letter_image = np.expand_dims(letter_image, axis=0)

        # Ask the neural network to make a prediction
        prediction = model.predict(letter_image)

        # Convert the one-hot-encoded prediction back to a normal letter
        letter = lb.inverse_transform(prediction)[0]
        predictions.append(letter)

        # draw the prediction on the output image
        cv2.rectangle(output, (x - 2, y - 2), (x + w + 4, y + h + 4), (0, 255, 0), 1)
        cv2.putText(output, letter, (x - 5, y - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.55, (0, 255, 0), 2)

    # Print the captcha's text
```



```
        # draw the prediction on the output image
        cv2.rectangle(output, (x - 2, y - 2), (x + w + 4, y + h + 4), (0, 255, 0), 1)
        cv2.putText(output, letter, (x - 5, y - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.55, (0, 255, 0), 2)

    # Print the captcha's text
    captcha_text = "".join(predictions)
    print("CAPTCHA text is: {}".format(captcha_text))

    # Show the annotated image
    cv2_imshow(output)
    cv2.waitKey()
```

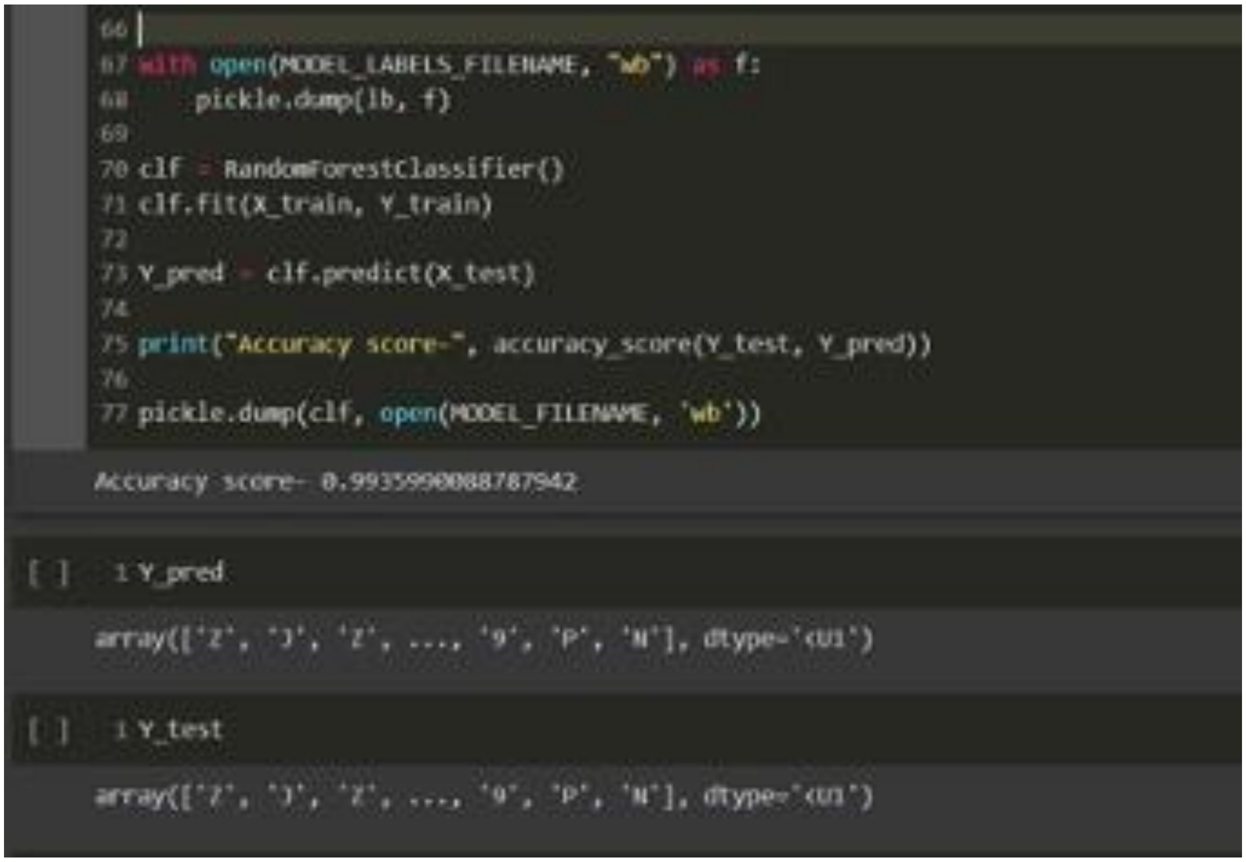
```
CAPTCHA text is: 2US3

CAPTCHA text is: 8LJ9

CAPTCHA text is: XHLR

CAPTCHA text is: R5KN
```



```
CAPTCHA text is: R5KN

CAPTCHA text is: X5ST

CAPTCHA text is: ZYLR

CAPTCHA text is: YXE3

CAPTCHA text is: 2W3R
```

2. Captcha recognition using random forest:

Most of the image pre-processing is the same as the process mentioned in the previous sections and screenshots. Using the random forest classifier for training.



```
with open(MODEL_LABELS_FILENAME, "wb") as f:
    pickle.dump(lb, f)

clf = RandomForestClassifier()
clf.fit(X_train, Y_train)

Y_pred = clf.predict(X_test)

print("Accuracy score-", accuracy_score(Y_test, Y_pred))

pickle.dump(clf, open(MODEL_FILENAME, 'wb'))
```

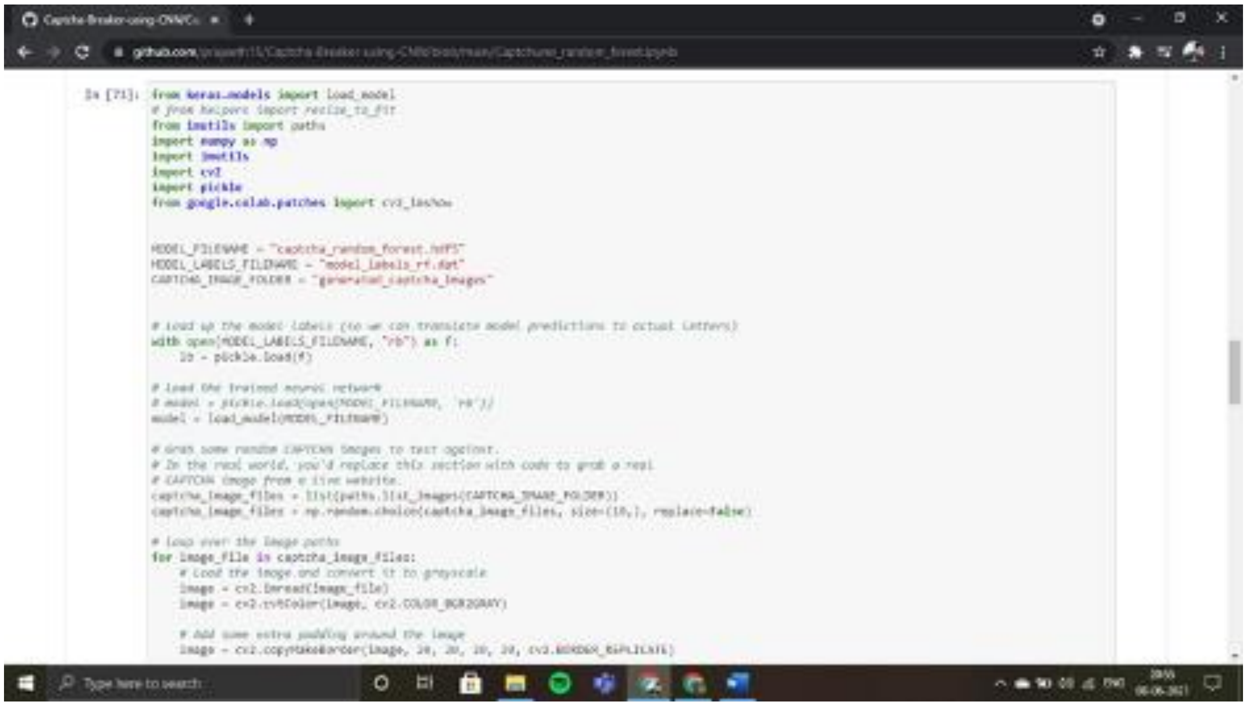
```
Accuracy score- 0.9935990088787942
```

```
[ ]  Y_pred
```

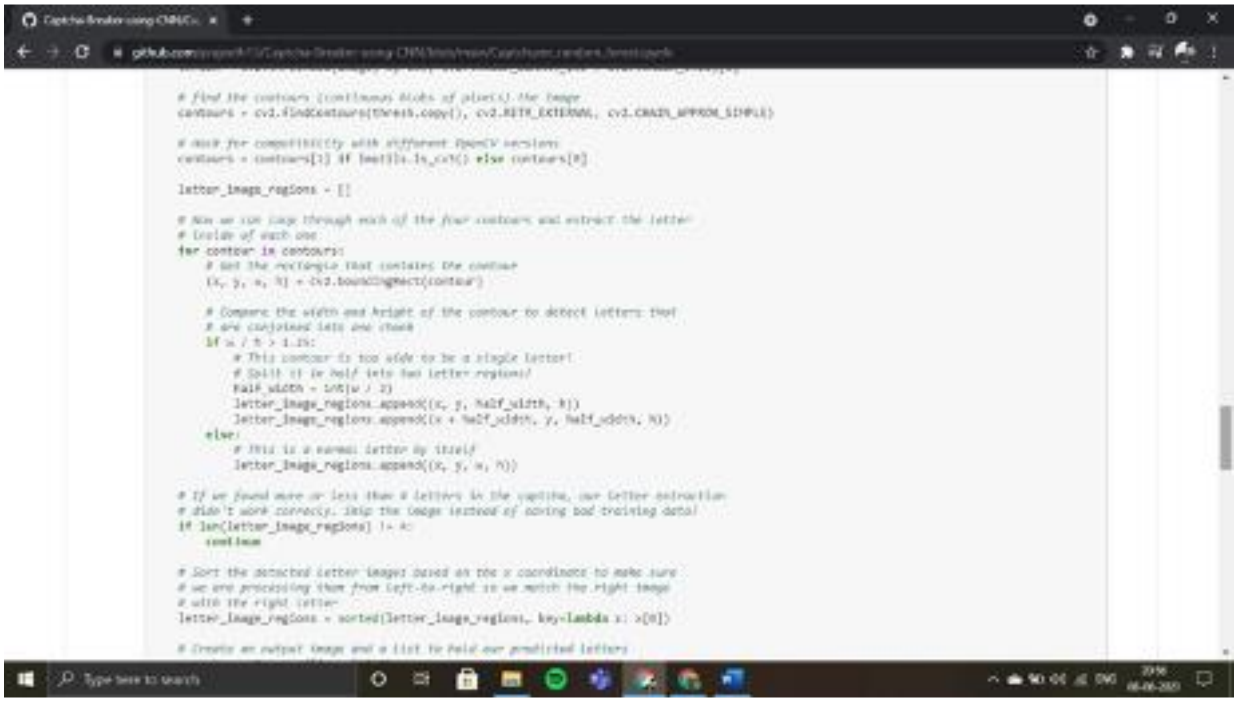
```
array(['Z', 'J', 'Z', ..., '9', 'P', 'N'], dtype='<U1')
```

```
[ ]  Y_test
```

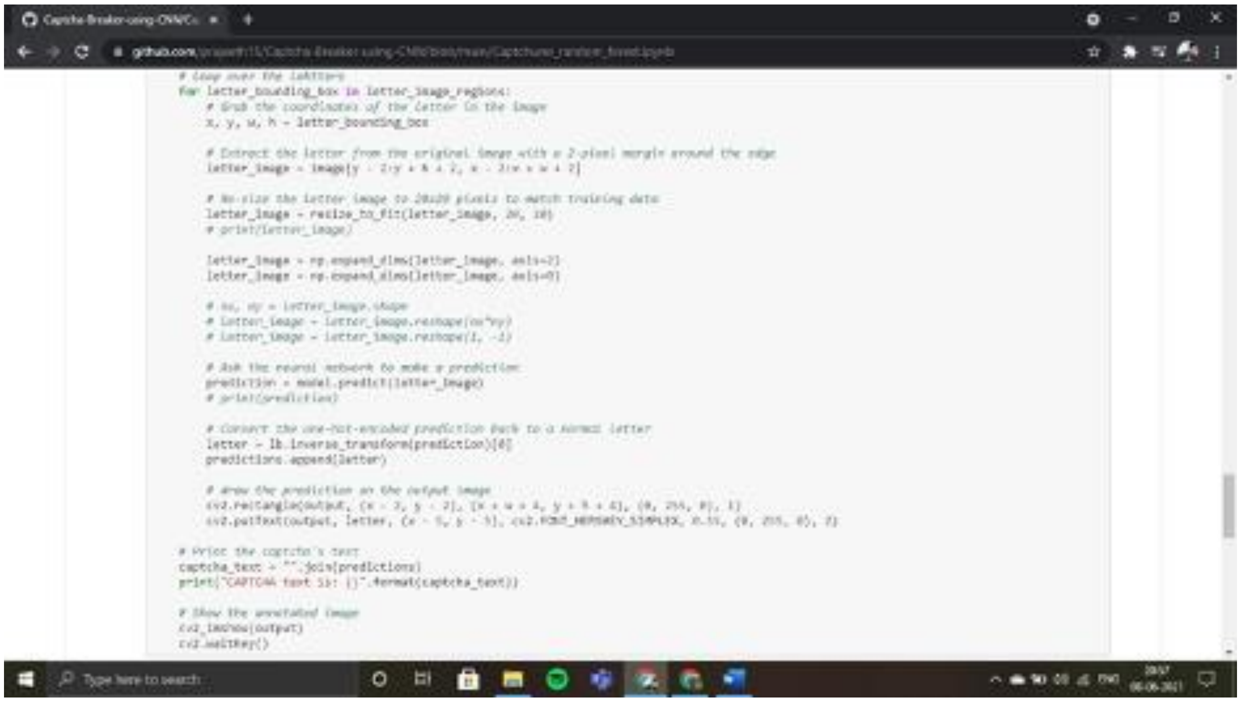
```
array(['Z', 'J', 'Z', ..., '9', 'P', 'N'], dtype='<U1')
```



```
In [71]: from keras.models import load_model
         # from helpers import resize_to_fit
         from imutils import paths
         import numpy as np
         import imutils
         import cv2
         import pickle
         from google.colab.patches import cv2_imshow


         MODEL_FILENAME = "captcha_random_forest.hdf5"
         MODEL_LABELS_FILENAME = "model_labels_rf.dat"
         CAPTCHA_IMAGE_FOLDER = "generated_captcha_images"


         # Load up the model labels (so we can translate model predictions to actual letters)
         with open(MODEL_LABELS_FILENAME, "rb") as f:
             lb = pickle.load(f)

         # Load the trained neural network
         # model = pickle.load(open(MODEL_FILENAME, 'rb'))
         model = load_model(MODEL_FILENAME)

         # Grab some random CAPTCHA images to test against.
         # In the real world, you'd replace this section with code to grab a real
         # CAPTCHA image from a live website.
         captcha_image_files = list(paths.list_images(CAPTCHA_IMAGE_FOLDER))
         captcha_image_files = np.random.choice(captcha_image_files, size=(10,), replace=False)

         # loop over the image paths
         for image_file in captcha_image_files:
             # Load the image and convert it to grayscale
             image = cv2.imread(image_file)
             image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

             # Add some extra padding around the image
             image = cv2.copyMakeBorder(image, 20, 20, 20, 20, cv2.BORDER_REPLICATE)
```

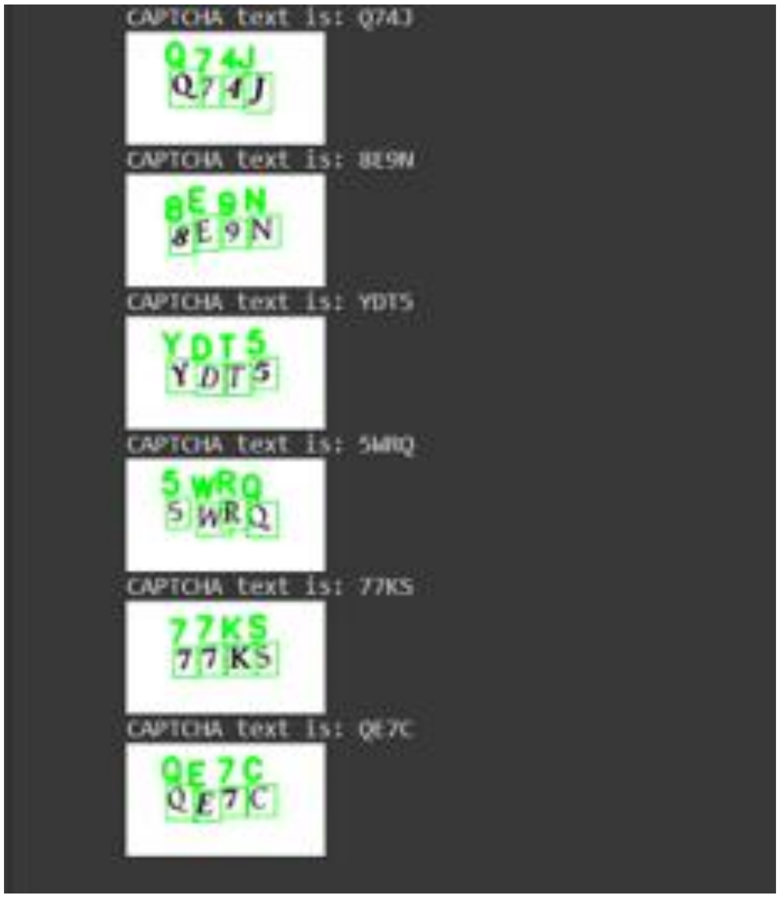


```
    # find the contours (continuous blobs of pixels) the image
    contours = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    # Hack for compatibility with different OpenCV versions
    contours = contours[1] if imutils.is_cv3() else contours[0]

    letter_image_regions = []

    # Now we can loop through each of the four contours and extract the letter
    # inside of each one
    for contour in contours:
        # Get the rectangle that contains the contour
        (x, y, w, h) = cv2.boundingRect(contour)

        # Compare the width and height of the contour to detect letters that
        # are conjoined into one chunk
        if w / h > 1.25:
            # This contour is too wide to be a single letter!
            # Split it in half into two letter regions!
            half_width = int(w / 2)
            letter_image_regions.append((x, y, half_width, h))
            letter_image_regions.append((x + half_width, y, half_width, h))
        else:
            # This is a normal letter by itself
            letter_image_regions.append((x, y, w, h))

    # If we found more or less than 4 letters in the captcha, our letter extraction
    # didn't work correcly. Skip the image instead of saving bad training data!
    if len(letter_image_regions) != 4:
        continue

    # Sort the detected letter images based on the x coordinate to make sure
    # we are processing them from left-to-right so we match the right image
    # with the right letter
    letter_image_regions = sorted(letter_image_regions, key=lambda x: x[0])

    # Create an output image and a list to hold our predicted letters
```



```
# Loop over the letters
for letter_bounding_box in letter_image_regions:
    # Grab the coordinates of the letter in the image
    x, y, w, h = letter_bounding_box

    # Extract the letter from the original image with a 2-pixel margin around the edge
    letter_image = image[y - 2:y + h + 2, x - 2:x + w + 2]

    # Re-size the letter image to 20x20 pixels to match training data
    letter_image = resize_to_fit(letter_image, 20, 20)
    # print(letter_image)

    letter_image = np.expand_dims(letter_image, axis=2)
    letter_image = np.expand_dims(letter_image, axis=0)

    # nx, ny = letter_image.shape
    # letter_image = letter_image.reshape(nx*ny)
    # letter_image = letter_image.reshape(1, -1)

    # Ask the neural network to make a prediction
    prediction = model.predict(letter_image)
    # print(prediction)

    # Convert the one-hot-encoded prediction back to a normal letter
    letter = lb.inverse_transform(prediction)[0]
    predictions.append(letter)

    # draw the prediction on the output image
    cv2.rectangle(output, (x - 2, y - 2), (x + w + 4, y + h + 4), (0, 255, 0), 1)
    cv2.putText(output, letter, (x - 5, y - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.55, (0, 255, 0), 2)

# Print the captcha's text
captcha_text = "".join(predictions)
print("CAPTCHA text is: {}".format(captcha_text))

# Show the annotated image
cv2.imshow(output)
cv2.waitKey()
```

Displaying the predicted values using the random forest classifier,


CAPTCHA text is: Q74J
CAPTCHA text is: 8E9N
CAPTCHA text is: YDT5
CAPTCHA text is: 5WRQ
CAPTCHA text is: 77KS
CAPTCHA text is: QE7C

Result analysis:

In this project, we have used 2 classifiers- one being a random forest classifier and the other is a simple convolutional neural network (CNN) model. The random forest model has an accuracy score of approximately 0.9935 (99.35 % accurate). We used the random forest model only to check how good our dataset preparation was, and it is not used for making any other comparisons. Since we got such a high accuracy score with the random forest model, we can conclude that the dataset preparation was done properly and meticulously.

On the other hand, by the tenth and final epoch, the CNN model has an accuracy score of approximately 0.9998 (99.98 % accurate) on the training data and approximately 0.9981 (99.81 % accurate) on the validation data. Further, the loss in training data is only around 0.0013 (0.13 %) and the loss in the validation data is around 0.0061 (0.61 %).

Following are the graphical plots of both model accuracy and loss on both training and validation data with respect to the number of epochs during the training of the CNN model.

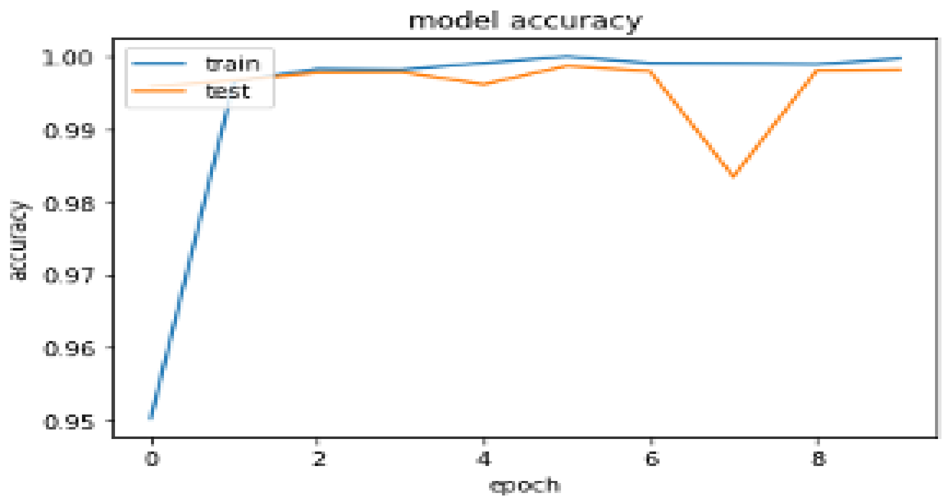


Fig 9: Model accuracy on training and validation data as count of epoch increases

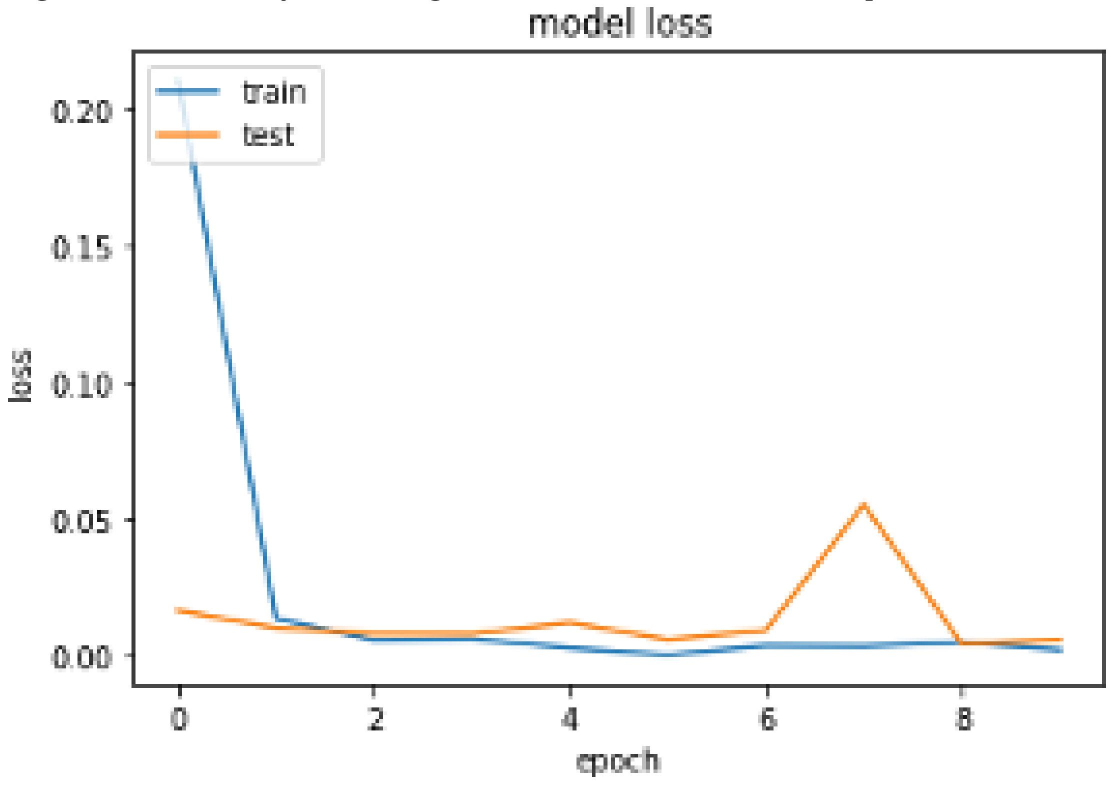


Fig 10: Model loss on training and validation data as count of epoch increases

In the above two graphs, around the seventh epoch, there is a significant difference where the model's accuracy decreases, and loss increases on the validation data. This is most probably due to a case of over-fitting since the model's performance on the training data does not change. But later, the accuracy again increases and the loss decreases, meaning the model is no longer over-fitting and has been trained properly on the created dataset.

The final working and predictions made by the trained CNN model can be seen from the images present in the previous sections.

6. CONCLUSION AND FUTURE WORK

In this project, we have built a deep learning model based on a simple convolutional neural network architecture. The trained model takes a CAPTCHA image as input, and correctly recognizes the characters present in the provided input image. The trained model is approximately 99.81 % accurate in recognizing the characters present in the image. We also used a simple but effective method in order to segment the characters present in the input image, in order to simplify and convert the problem into a basic character recognition task, using very simple techniques of image processing and computer vision.

In this project, we used CAPTCHA images with only four characters present in them. Future work may include using much more complex images with a variable number of characters present in them, as well as images with line distortions and noise impurities added in them, to train and improve and increase the model's performance.

7. REFERENCES

1. L. Von Ahn, M. Blum, and J. Langford, "Telling humans and computers apart automatically," *Communications of the ACM*, vol. 47, no. 2, pp. 56–60, 2004.
2. K. Chellapilla and P. Y. Simard, "Using Machine Learning to Break Visual Human Interaction Proofs (HIPs)," in *Proceedings of the Advances in Neural Information Processing Systems*, pp. 265–272, ofAdvances in Neural Information Processing Systems, 2004.

3. N. Roshanbin and J. Miller, "A survey and analysis of current CAPTCHA approaches," *Journal of Web Engineering*, vol. 12, no. 1-2, pp. 001–040, 2013.
4. J. Yan and A. S. E. Ahmad, "A low-cost attack on a microsoft CAPTCHA," in *Proceedings of the 15th ACM conference on Computer and Communications Security, CCS '08*, pp. 543–554, USA, October 2008.
5. S.-Y. Huang, Y.-K. Lee, G. Bell, and Z.-H. Ou, "An efficient segmentation algorithm for CAPTCHAs with line cluttering and character warping," *Multimedia Tools and Applications*, vol. 48, no. 2, pp. 267–289, 2010.
6. R. A. Nachar, E. Inaty, P. J. Bonnin, and Y. Alayli, "Breaking down Captcha using edge corners and fuzzy logic segmentation/recognition technique," *Security and Communication Networks*, vol. 8, no. 18, pp. 3995–4012, 2015.
7. Naomi S. Altman. An Introduction to Kernel and NearestNeighbor Nonparametric Regression. Am. Stat., 46(3): 175–185, aug 1992. ISSN 0003-1305. doi: 10.1080/ 00031305.1992.10475879.

8. Christopher M. Bishop. Pattern recognition and machine learning. Springer, 2006. ISBN 0387310738.
9. Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In Proc. fifth Annu. Work. Comput. Learn. theory - COLT '92, pages 144–152, New York, New York, USA, 1992. ACM Press. ISBN 089791497X. doi: 10.1145/ 130385.130401.
10. Ondrej Bostik, Karel Horak, Jan Klecka, and Daniel Davidek. Bubble Captcha - A Start of the New Direction of Text Captcha Scheme Development. In Mendel 2017, 23rd Int. Conf. Soft Comput., volume 23 of 23, pages 57–64. Brno University of Technology, 2017.