CS771

Assignment 2
Support Vectors

Asjad Raza	Mallik Zadah Irfan	Shreyansh Pachauri
210225	210576	200954
Vishnu Gaur	Gourav Jaiswal	Krishn Kumar Choudhary
201132	22123009	200522

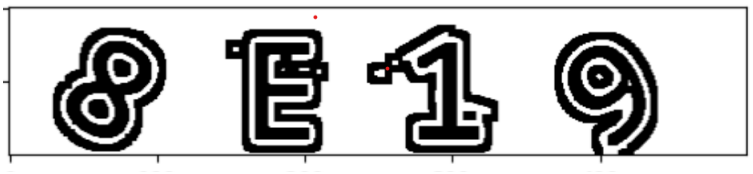
July 2023

Problem 2.1

`HexaCAPTCHA`
Your Task. Your task is to design an algorithm that can take a set of such 500×100 images and return the parity of the hexadecimal number in that image. 1. Describe the method you used to find out what characters are present in the image. Give all details such as details of the model being used (e.g. linear, decision tree, neural network, kernel etc), the training algorithm used, including hyperparameter search procedures, validation procedures. You have to give a detailed explanation even if you used an algorithm/implementation from the internet – make sure to give proper credit to the person/source from where you took code/algorithm. There is no penalty for using someone else's code/algorithm but there would be heavy penalty for doing so without giving proper credit to that person/source.
Answer: The model operates in two distinct stages. Initially, it employs computer vision techniques to separate the individual characters, followed by the utilization of a convolutional neural network (CNN) based deep learning model to accurately predict each character. Let's delve into each of these stages in detail.
During the first stage, various processes are applied to the image. One of these processes involves the elimination of "stray lines." To achieve this, the image is subjected to a dilation operation followed by an erosion operation. By applying these operations, the stray lines are effectively removed, and the

letters within the image are restored to a similar state as before the removal of the stray lines. An example of an image after this operation is presented below:

In the subsequent morphological transformation, the focus is on separating and emphasizing the margins of the characters. This is achieved by applying a morphological kernel to make the margins bold. The resulting image, after this operation, is displayed below:



In the preceding image, the white areas can be considered as a collection of connected components. Notably, if we turn the outermost connected component black, the letters will be separated. Given that the letters are relatively smaller in size compared to the overall image, this component will also be the largest one. Hence, our objective is to identify and extract the largest connected component in the image.
To accomplish this task, we make use of an OpenCV function that computes the connected components within the image while providing relevant metadata. In this case, we employ 8-connectivity, which defines a strong notion of connectivity, ensuring that two sections cannot be connected if they are only linked at a corner. As a result, the image now appears as follows:

After segmenting the letters, our next objective is to eliminate the random white splotches present in the image. These splotches can be considered as contours. To achieve this, we obtain the contours and their hierarchy. The contours can be organized into a tree structure, where the contours enclosed by a contour are its children.
Initially, we only consider the highest level contours, which correspond to the roots or the children of the roots. This step is crucial to address the issue of

letters like or 8 or 9 which may have multiple levels of elliptical contours. In our case, we are interested in capturing only the first two levels.
Additionally, we observe that the useful contours tend to have a close resemblance to relvent shapes. Hence, we approximate each contour with a suitable shape. The characteristics of the contour are utilized to define the polygon at runtime. We then compare the areas of the approximation and the original contour. If the areas are very close, it indicates that the contour is useful, as the useful contours in our use case are typically well approximated by that shape. However, we do not outright reject the contour if this condition is not met.
Finally, we proceed to examine and remove the tiny splotches. The remaining contours are sorted based on their area in descending order. We then traverse the contours in this sorted order. If the ratio of areas between two successive contours is too high, we halt the traversal and discard the remaining contours. Only the traversed contours are accepted.
To visualize the accepted contours, they are drawn on an image. An example of such an image is provided here:

To further process the image, we divide it into four parts. Initially, we randomly sample N points and collect the white pixels from those points. Subsequently, we sort these white pixels based on their x-coordinates. As we examine the sorted list, we can observe three significant jumps in the x-coordinates. These jumps indicate the points at which we will divide the pixels into four groups.
After dividing the image into four groups, we calculate the average position for each group. This average position corresponds to a square that is centered at the calculated average position. These squares act as segments, providing localized regions for further analysis.
Since you mentioned that the model needs to be trained only for the last image number, we can assign a label to each segment based on whether the number is even or odd. We store all these segments in a list, preserving their order. To obtain the last image segment, we simply return the segment located at the third index of this list.

To create a training dataset for the CNN, we can use the last indexed seg-

ment as the training set and assign its corresponding label (even or odd) as the target. By doing so, we create a labeled dataset that can be utilized for training the CNN model.
Finally, the segments obtained from this processing step are fed into a neural network that is built using TensorFlow. The neural network can then perform various tasks such as character recognition or any other relevant analysis based on the specific application.

```
# Define the CNN model
model = keras.Sequential([
    keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64, 1)),
    keras.layers.MaxPooling2D((2, 2)),
    keras.layers.Conv2D(64, (3, 3), activation='relu'),
    keras.layers.MaxPooling2D((2, 2)),
    keras.layers.Conv2D(64, (3, 3), activation='relu'),
    keras.layers.Flatten(),
    keras.layers.Dense(64, activation='relu'),
    keras.layers.Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# Train the model
model.fit(x_train, y_train, epochs=10, batch_size=32, validation_data=(x_val, y_val))

# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print('Test Loss:', test_loss)
print('Test Accuracy:', test_accuracy)
```

The model architecture consists of multiple Conv2D layers, which perform 2D convolutions with different filter sizes and activations to extract features from the input images. MaxPooling2D layers are used for spatial dimension reduction. The Flatten layer converts the 2D output into a 1D vector, followed by Dense layers for fully connected classification.
In terms of hyperparameter tuning respective loss functions as mentioned in the code worked best for us we verified this using a grid search of various combinations of loss function vs accuracy and the above combination worked best for us.
The binary cross-entropy loss function is employed for binary classification tasks. It measures the dissimilarity between predicted probabilities and true binary labels, encouraging the model to output probabilities close to 1 for one class(ODD) and close to 0 for the other(EVEN). The goal is to minimize this loss function during training. The statistics for the split of 1600 training samples and 400 testing samples are as follows:
•Size of the Model: 7.5MB
•Testing Time per Image: 0.06825984300000527 seconds
•Test Loss: 0.02926153875887394
•Test Accuracy: 0.9975000023841858
•Parity Score: 1

```
Model: "sequential_15"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_45 (Conv2D)          (None, 62, 62, 32)        320

 max_pooling2d_30 (MaxPoolin  (None, 31, 31, 32)       0
 g2D)

 conv2d_46 (Conv2D)          (None, 29, 29, 64)        18496

 max_pooling2d_31 (MaxPoolin  (None, 14, 14, 64)       0
 g2D)

 conv2d_47 (Conv2D)          (None, 12, 12, 64)        36928

 flatten_15 (Flatten)        (None, 9216)              0

 dense_30 (Dense)            (None, 64)                589888

 dense_31 (Dense)            (None, 1)                 65

=================================================================
Total params: 645,697
Trainable params: 645,697
Non-trainable params: 0
_________________________________________________________________
```

Reference: concept of morphological splitting and image splitting through erosion and dilation, as well as the utilization of K means, were derived from the following sources:
An article by Shubham Chauhan, available at: https://shubhamchauhan125.medium.com/cnncaptchasolver-5625b189a14f.
The GitHub repository by cliche-niche, specifically the CS771 directory, located at: https://github.com/cliche-niche/CS771/tree/main/assn3.