CS771 Assignment 2

D-Bot

Aarish Khan, Emaad Ahmed, Krishnansh Agarwal, Rishi Agarwal, Vansh Raj Sachan, Zoya Manjer

1 Method Used to Solve the Problem

1.1 Image Pre-processing:

The initial idea was to perform contouring on the image using OpenCV to form clusters and single out the digits. We tried greyscaling the image and using inbuilt implementations of contour extraction to get the digits, but the background lines came along with the digits and in some cases disturbed the contour detection as well, so we abandoned this approach.

Next, we tried HSV-based color extraction method to single out parts of image based on colour, and applied a thickness threshold on the obtained coloured regions owing to the fact that the obfuscating lines were not as thick, and were able to reliably extract the images. But, since later on for training purposes we anyways binarized the image, we resorted to the following faster and reliable method.

First, we cropped the image and isolated the last digit, and resized it to a dimension of 90x90 pixels. For this we observed that the last digit remains in roughly the same position across all train images, allowing us to use the same bounding box that captured this region. We then converted the cropped image to grayscale using the OpenCV library to focus solely on grayscale intensity. Then we applied binary thresholding to leave the image with only two kinds of pixels, purely black and white. Still, there was noise from the straight lines in the background. To remove these we used eroded the image using the dilate method of openCV. After this processing, we obtained the following images:

1.2 CNN Approach:

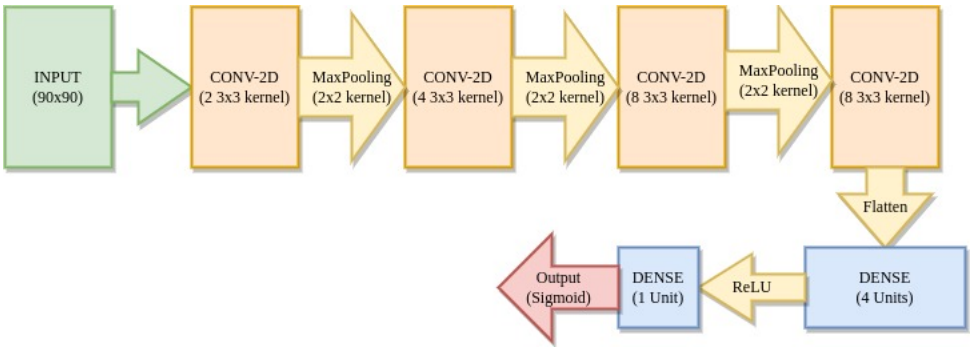
Since this is an Image Processing task, the first intuition was to use Convolutional Neural Networks. We formulated an architecture first and then whittled down the size of the network for hyperparameter tuning to achieve the objective of minimizing the space occupied the model. Following is a description of the approach and the advantages and disadvantages we felt it held.

The training dataset was split into a training set and a validation set with a test size of 0.25. The CNN architecture consisted of multiple 2D convolutional layers with total 22 3x3 filters, and an input dimension of 90x90x1 (width, height, and channel, respectively). After each convolutional

Preprint. Under review.

layer, we utilized a MaxPooling 2D layer to downsample the feature maps. Then we flatten the 2D layers to a single dimension.

To obtain the desired output, we incorporated one Dense layer in the CNN architecture. The Dense layer had 4 nodes with the Rectified Linear Unit (ReLU) activation function. The final layer consisted of a single node with the sigmoid activation function, producing a binary output indicating the presence or absence of characters. During training, we employed the Adam optimizer and used binary cross-entropy as the loss function. Additionally, we monitored the model's performance using an accuracy metric. We used the Tensorflow library for defining the model. The final model architecture looks like:



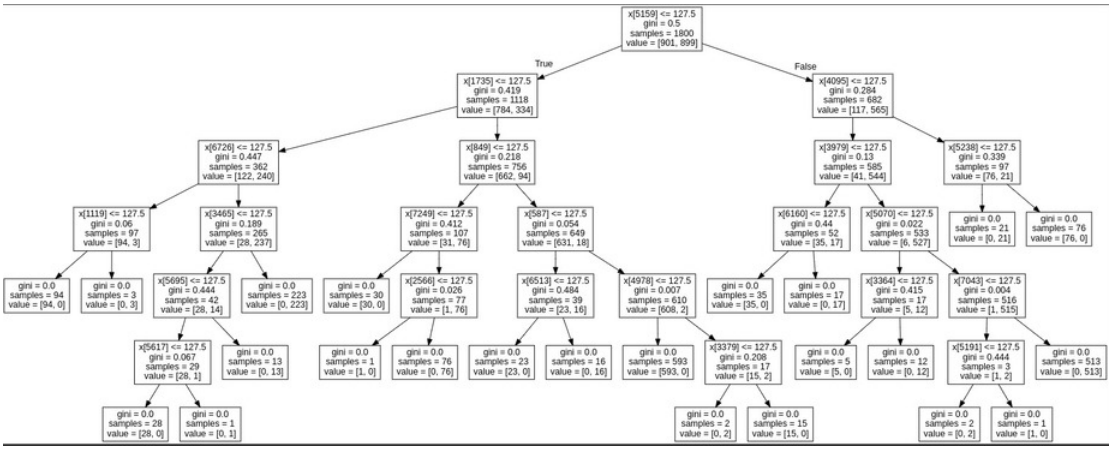
For the training process, we conducted only two epochs with a batch size of 32. Remarkably, the model achieved an impressive accuracy of 100% and a loss of 0.0084, indicating its effectiveness in character recognition. We then utilized the trained model to predict the characters in the validation dataset, achieving another outstanding accuracy of 100%. The model size and inference time were 25kB and 14ms/ image respectively

Noticing the ease with which the CNN was able to detect the characters and the numerous simplifications this problems had from a real challenge that a CNN would typically be used in, we decided to explore lighter architectures since even though the model size was very small thanks to the aggressive parameter reduction as evident in the model architecture, the inference time according to us was a little too long.

1.3 Decision Tree Approach:

The motivation behind using decision trees was the fact that this was a binary classification problem and that we were able to reduce our input to a relatively small size. We also noticed that it was not important to actually recognize *what* the digit was, and that a brute-force like classification (as commonly done by a DT) could suffice to capture the necessary information.

We used Scikitlearn's implementation of the Decision tree and tuned it manually by mainly tweaking the parameter of the max depth. This implementation uses the gini impurity as a loss function by default. This loss worked fine for our case so we did not change it. After training and tuning, we found the standard parameters with a maximum tree depth of 8 gave brilliant outcomes. We achieved 100% accuracy both on train and test sets with the model size of only **5.6kB** and inference time of only **1.6ms** per image.
We trained the following decision tree for the problem:



The convergence, as in the case of CNNs occurred within a few epochs of training and since the learning parameters were a few split conditions only, we concluded that the result was satisfactory. It is worth noting that the superiority of the DT approach stems from the fact that the problem was very simplified and presented in a manner that the local features of the digit were not necessary to be captured. Had that not been the case, we believe that this approach would have proven to be inferior to the CNN approach, which does indeed take into account the local features of the digits.

2 Code Used (& not used)

2.1 Image Processing

```
import numpy as np
import cv2
from PIL import Image

def cropper(png_file):
    image = Image.open(png_file)
    width, height = image.size
    crop_region = (width - 140, 10, width - 50, height)
    ci = image.crop(crop_region)
    cv2_imshow(ci)
    ci = np.asarray(ci)
    _,img = cv2.threshold(ci,165,255,cv2.THRESH_BINARY)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    _,img = cv2.threshold(img,254,255,cv2.THRESH_BINARY)
    kernel = np.ones((5,5), np.uint8)
    img_erode = cv2.dilate(img, kernel)
    cropped_train.append(img_erode)
```

This code is also used in the final submission.

2.2 CNN approach

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(2, kernel_size=(3, 3), activation='relu', input_shape=(90
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
    tf.keras.layers.Conv2D(4, kernel_size=(3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
    tf.keras.layers.Conv2D(8, kernel_size=(3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
    tf.keras.layers.Conv2D(8, kernel_size=(3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
```

```
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(4, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.summary()
model.fit(X_trn, Y_trn, epochs=30, batch_size=32)
# model.save_weights('model.h5')

weights_file = "final.pkl"
with open(weights_file, "wb") as f:
    pickle.dump(model, f)

predictions = model.predict(x_tst)
count = 0
for i in range(500):
    if predictions[i] == y_tst[i]:
        count = count + 1
count/5.0
```

The code for the decision tree approach only included fitting the scikitlearn's implementation on the preprocessed dataset and some manual tweaking so it is not provided. The final predict.py makes use of the weights obtained from this model to make predections.

3 References

1) OpenCV Documentation
2) HSV Color extraction using OpenCV
3) CNN Woking and Intuition
4) Keras Docs
5) SciKit Learn Docs