CS771: Introduction to Machine Learning
Assignment-3

Group No. : 35
Karishma Laud - 19111041
Aakrati Jain - 19111001
Apoorva Jain - 19111016
Chaman Jangra - 19111024
Sharwari Samdekar - 19111083
Vasudha - 19111419
Department of Computer Science and Engineering
Indian Institute of Technology Kanpur

Abstract

The document discusses the algorithm for solving the DeCAPTCHA problem of identifying the captcha characters from the given image. To solve this problem CNN model is used after some image preprocessing steps.

1 Solution

Algorithm

1. Convert image from BGR to HSV format using OpenCv library function cvtColor.

2. Get the mask image for conversion from HSV to grayscale by keeping only brightness values from 150 to 255 using OpenCv library function inRange.

3. Invert the mask image.

4. Find extreme outer contours of the mask image using OpenCv library function findContours.

5. For each contour obtained do the following:

   (a) Get the rectangle that contains the contour using OpenCv library function boundingRect.

   (b) If (width of the rectangle) / (height of the rectangle) > 1.5
   Split the rectangle into two letter regions.
   else
   Make the entire rectangle as a single letter region.

6. For each of the letter regions:
   (a) Grab the coordinates of the letter in the image.

   (b) Extract the letter from the original image with a 2-pixel margin around the edge.

Preprint. Under review.

(c) Make prediction using our trained CNN model which has been described below.

(d) Convert the encoded prediction back to a normal letter text.

(e) Append the current letter to the predicted list of letters(predicted_list).

7. Number of characters(number_of_characters) =Number of contours obtained.

8. return (number_of_characters,predicted_list)

Building and Training CNN model

Keras sequential model is used to build CNN model with relu activation at input and hidden layer, softmax at output layer. 5*5 strides are used.

```
#building model
model = Sequential()
model.add(Conv2D(20, (5, 5), padding="same", input_shape=(30, 30, 1), activation="relu"))
model.add(MaxPooling2D(pool_size=(5, 5), strides=(5, 5)))
model.add(Conv2D(50, (5, 5), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(5, 5), strides=(5, 5)))


model.add(Flatten())
model.add(Dense(80, activation="relu"))


model.add(Dropout(0.3))
model.add(Dense(26, activation="softmax"))

model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])

# using early stoping for avoiding overfitting
estop = EarlyStopping(patience=10, mode='min', min_delta=0.001, monitor='val_loss')

model.fit(train_x, train_y, validation_data=(val_x, val_y), batch_size=32, epochs=30, verbose=1, callbacks = [estop])
model.save('my_model.h5')
```

Figure 1: CNN model code

1. Initially 2D convolution layer is used which creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs. We have used 20 output channels, kernel size of 5*5 moving window followed by the strides of 5*5 in the x and y directions.

2. Then, 2D max pooling layer of size 5*5 is used.

3. Next, we add another convolutional + max pooling layer, with 50 output channels.

4. After this, we flatten the output from these to enter our fully connected layers.

5. Next, we specify 80 dimensional output space as unit parameter in Dense function with relu as activation function.

6. 30% dropout is used to regularize the model.

7. Finally we apply softmax activation function and get 26 dimensional output (representing 26 classes A-Z) at the output layer.

8. To avoid over-fitting, this model uses early stopping.

   This trained CNN model will be used for testing.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 30, 30, 20)        520
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 15, 15, 20)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 15, 15, 50)        25050
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 7, 50)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2450)              0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               313728
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 26)                3354
=================================================================
Total params: 342,652
Trainable params: 342,652
Non-trainable params: 0
_________________________________________________________________
None
```

Figure 2: CNN model summary

Reference We have taken help from the code available at medium article [1] by Shubham Chauhan whose reference link is given on last page under the reference section.

2 Hyperparameter Tuning

We have tuned our hyperparameters using Cross Validation.

1) We have varied the number of nodes in the last hidden layer and the stride length.
For 4th observation we were getting least model size and best accuracy. Hence we chose the no of nodes in the last hidden layer to be 80. We have listed our observations in table 1.

	Model	Accuracy(%)
No of nodes in the last hidden layer =256 and stride =(2X2)	2MB	100
No of nodes in the hidden last layer=64 and and stride =(2X2)	2.14MB	99.25
No of nodes in the hidden last layer=256 and and stride =(5X5)	565KB	100
No of nodes in the hidden last layer=80 and and stride =(5X5)	404KB	100

Table 1: Observations varying no of nodes at last hidden layer

2) We have varied the stride length. for Observation 2 . We selected a stride length of (5X5) as we were getting the lest model size for stride of (5X5).
We have listed our observations in table 2.

	Model	Accuracy(Percent)	Avergae Prediction Time on 8 images (secs)
stride (2X2)	3.95MB	100	0.655372
stride (5X5)	404KB	100	0.680

Table 2: Observations varying dropout and epoch

3) We have varied dropout for epoch=30 and stride (5X5). We were getting best prediction and accuracy when we chose a dropout of 0.3.
We have listed our observations in table 3.

	Model(KB)	Accuracy(Percent)	Prediction Time on 8 images (secs)
Dropout = 0.1 and epoch =30	404	99.5	0.655372
Dropout = 0.5 and epoch =30	404	100	0.642787
Dropout = 0.3 and epoch =30	404	100	0.635952

Table 3: Observations varying dropout and epoch

4) For Dropout = 0.3 , epoch =30 and number of nodes in the last hidden layer = 80,
we increased number of hidden layers to 4. Our model size became 1.31MB with negligible improvement in performance.
So we didn't change no of hidden layers and kept it as 3.

Final Hyperparameter values and observation:

For Dropout = 0.3 and epoch =30 and number of nodes in the last hidden layer = 80 and number of hidden layers =3 .We obtained the following results :
Accuracy = 100%
Model Size=404KB
Average Predition Time for 8 images =0.68 secs

3 Solution

Zipped Code folder is uploaded at: `https://www.cse.iitk.ac.in/users/apoorvaj/iitkassignmentml/submit.zip`

References

[1] https://medium.com/analytics-vidhya/cnncaptchasolver-5625b189a14f.