CS771A – Project 3

Project Title : Automatic Captcha Recognition

The given captcha code images contained well structured alphabet images but the background contained some obsfuscating lines which posed difficulty to the process alphabets detection. We used basic image processing techniques like dilation, thresholding, etc to preprocess the images eliminating noise and background. The model for detecting captcha codes was designed using a convolutional neural network. Brief outline of method used for developing the model:

Image preprocessing:

We first converted the given image into hsv space. We observed that the saturation of background and the unwanted lines was much lower than the alphabet images. This distinctively eliminated the unwanted colours and converted the image into a single channel.

Background removal:

We identified the color of the corner point(0,0) of the given image. All pixels of the same color were removed from the image.

Dilation:

Dilation added pixels to the boundaries of objects in the image. The number of pixels added from the objects in the image depends on the size and shape of the structuring element used to process the image.

The value of the output pixel came out to be the maximum value of all pixels in the neighborhood. This made objects more visible and fills in small holes in objects.

Resizing Since our input data is structured so we resized the image to a optimum size. Initially the images had a high dimension which if were used to train than the model size would have been large and also time taken would be large. So we decreased the size to an optimum value where we could extract all the information from the images without wasting extra memory and time. For resizing we used a file helper.py [1] for resizing the image as compared to cv2.resize() because it was time effective.

Contour Detection

Contours detection is a process which can be explained simply as a curve joining all the continuous points (along with the boundary), having same colour or intensity. The contours are a useful tool for shape analysis and object detection and recognition.

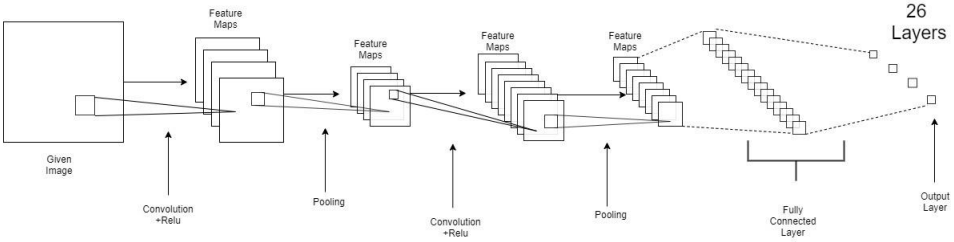
An edge is a point in an image where there is a sharp change in the pixel colour value which doesn't make it continuous in nature and sometimes makes it hard to determine the shape of the object. Contours can do a bit more than "just" detect edges. The algorithm does indeed find edges of images but also puts them in a hierarchy. This means that we can request outer borders of objects detected in our images.

Since it detects all the contours that could possibly exist in our image. Thus it also detects contours of noises. Noises contour will definitely have less area as compared to the segments we are planning to extract. So we calculate the area of all the contours detected and put a threshold on that area. After

Preprint. Under review.

applying contour detection on our training data we came to a threshold that all the noises detected will be less than 200 pixels in area. So we selected those contours only which has area more than 200 pixels.

CNN Architecture



We used a 6 layered neural network having 2 convolution layers, 2 max pooling layers, 1 fully connected layer and an output layer containing 26 neurons. The output layer is programmed as multi-class classifier, with each output neuron corresponding to one alphabetical class. After running various iterations with different architectures the final architecture looks something like this :

Layer 1

First layer is a convolution Layer which takes the input image of dimension **(25X25)**. The padding is set to be such that output volume is same as the input volume. The activation function of the layer is chosen to be the **Relu** function.

Layer 2

Second Layer is a Max pooling layer with a **pool _size** to be set as **(2, 2)**. The strides of the layer is also set as **(2, 2)**

Layer 3

Third Layer is also a Convolutional Layer which takes input a vector of dimension **20**. The padding is set to be such that the output of this layer is to be same as the input of next layer. The activation function is set to be the **Relu** function.

Layer 4

Fourth Layer is identical to second layer. It is a Max. pooling layer with a **pool _size** to be set as **(2, 2)**. The strides of the layer is also set as **(2, 2)**

Layer 5

Fifth layer is a fully connected layer containing **50** nodes. The activation function taken for this layer is taken to be **Relu**

Layer 6

The output layer is programmed as multi class classifier, with each output neuron corresponding to one alphabetical class. So, the output layer contain 26 nodes. Since, it a multi class classifier. So, its activation function is set to be **softmax**

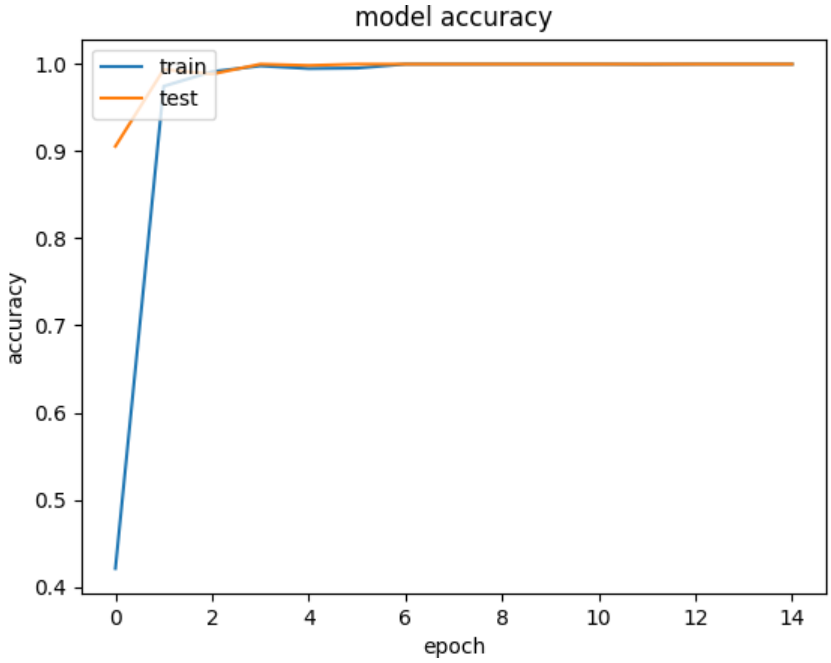
Training

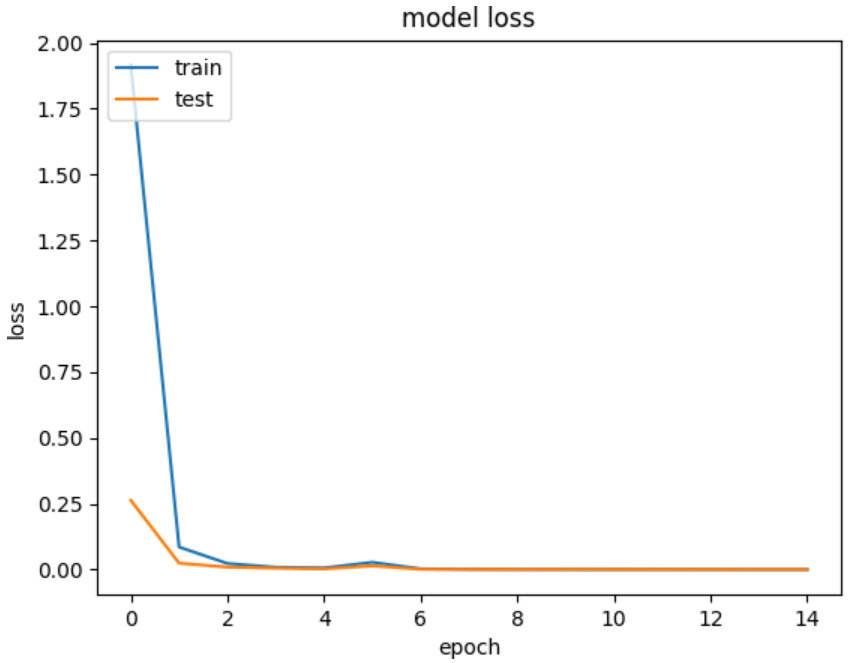
After designing the architecture of CNN we compiled our model setting out Loss metric as **cross-entropy loss** and optimization metric set as **Accuracy** and **adan** as optimizer.

For training we splitted out training data in ratio of 75:25. We trained the model on 75% of data and tested it on rest 25%.

After preparing the model we trained the model by taking a batch size of **32** and initally we set the epoch to **15** to visualize the comparison of model accuracy and loss of training data and validation data. The graaph for 15 epoch is been visualized and shown below.

After looking at the graphs we got observation that our NN is very strong started converting from epoc 4 only. So after this we trained our model for 4 epoch for submission.





After visualizing, we trained it on the whole dataset and tested in whole dataset, and the results we got are: time-5.2 secs, model size- 642kB

Effect of parameters in Image Processing

1. Threshold

Threshold	Prediction Time	Fraction Length accuracy	Code Match Score
230	2.362	1.00	1.00
200	2.404	1.00	1.00
250	2.405	1.00	1.00

2. Kernel Size

Kernel Size	Prediction Time	Fraction Length accuracy	Code Match Score
(3,3)	2.517	1.00	0.989
(4,4)	2.616	1.00	1.00
(5,5)	2.358	1.00	1.00
(6,6)	2.401	1.00	0.981

3. Iterations

Iterations	Prediction Time	Fraction Length accuracy	Code Match Score
1	2.567	1.00	1.00
2	2.416	1.00	0.77

4. Hyperparameters

0.1 Convolution Layer 1

Activation Function	Kernel size	Padding and stride
ReLU	5x5	Input volume = output volume

0.2 Max Pooling Layer 1

Pool Size	Stride
2x2	2x2

0.3 Convolution Layer 2

Activation Function	Kernel size	Number of kernels	Padding and stride
ReLU	5x5	20	Input volume = output volume

0.4 Max Pooling Layer 2

Pool Size	Stride
2x2	2x2

0.5 Dense Layers

Activation Dense 1	Number of Neurons Dense 1	Activation Dense 2	Number of Neurons Dense 2
ReLU	50	SoftMax	26

0.6 Others

Loss Function : Categorical cross entropy
Optimizer : Adam
Batch size : 32
Epoch : 15
Verbose : 1

Dependencies used in the code i) OpenCV ii) numpy library iii) os iv) keras.models v) imutils

Results
The code results was evaluated on a 75 : 25 split of training data and validation data. We used held out validation to check the performance of our code Prediction time : 2.13 s on 500 captcha images
Code match score : 1.00
Fraction length accuracy : 1.00
Model size : 642.2 kB

References

[1] URL: https://github.com/lxw0109/ML-Experiments/tree/master/CAPTCHA/solving_captchas_code_examples.