Machine Learning Engineer Nanodegree

Capstone Project

Radheshyam verma

July 30th, 2019

I. Definition

Project Overview

Captcha have been one of the bottlenecks in data accumulation/scraping. Most of the websites, to prevent data scraping keep public information behind captchas. Railway booking info, scorecards, Business info with PAN card number are some of the examples where public data is present behind captchas. Although such measures prevent public data scraping, but it hinders data accumulation for data scientist enthusiasts.

Other problems related to captcha images is the delay it causes while filling forms. Unlike other common form-fields which are automatically filled by smart browsers, captchas are ignored.

Captcha reading problems fall in the category of OCR or object localisation networks. Where position of an object also affects the output. So captcha reading problem can be solved with object localization networks as well as LSTM networks present in keras as presented in the article. OCR techniques are well suited for the captcha reading problem. Which are basically optimisations over RNN.

There are already many solutions developed for breaking captchas following wikipedia section enlists some of them.

Problem Statement

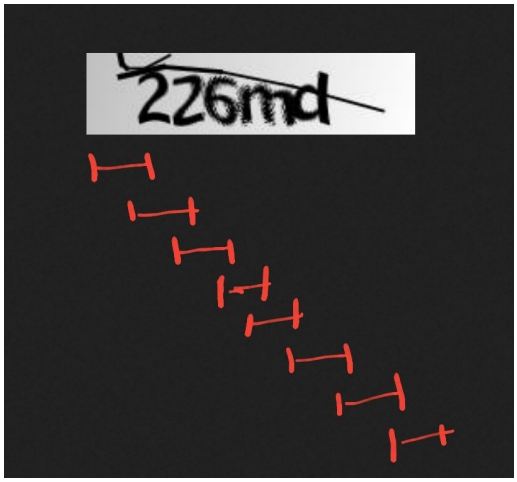
For this project I will be using image dataset from kaggle captcha dataset. This dataset contains 1070 files of different captchas. Scope of the problem that this project intends to solve is limited to correctly predicting text in the captcha. Any additional use-case built on top of that such as browser add on, data scraper is not part of scope.

Solution presented in this document will be relevant only to the images present in the dataset, solution is not intended to work on all types of obscure captchas that are unlike captchas present in the dataset.

This problem involves image classification but the important thing is that here sequence and repetition of character also matters, So even though we identify that a, b are present in the image, we also have to identify the sequence. Or if a is present 2 times and b is present 3 times.

I will be using a combination of neural networks and manual python coding to assess the text written in the image. This problem is quite similar to handwriting reading problem with extra difficulty in terms of noise present in the image. Usually any object localization algorithm can clearly define the boundaries of the text, which can then be sorted to get the text written in the image.

CNN part of the solution will identify the character present in the image. While using python program I will move window from left to right to construct the text present in the image.
This is illustrated in the below image



Some implementations of object localizations also use dividing image in grid, my implementation is similar with just exception that I will be using sliding window manually.

Metrics

There are 2 metrics involved here, first metrics is of the CNN, to correctly identify character present in the image. While other metrics is of the manual python code which uses moving window to determine the correct sequence of the characters.

It can be seen that 1st metrics(CNN) will have a direct impact on the overall correct prediction. Incorrect classification in any one sliding window will result in incorrect output of complete image.

For CNN **accuracy** is chosen metrics as we want out CNNs output to be as close as possible to the output vector. For loss function **categorical_crossentropy** and **binary_crossentropy** will be evaluated. These loss functions are suited for multi-class classifications. As these loss functions punish for misclassification of each class.

However overall we would want to improve our overall efficiency of the model, which includes accuracy of CNN as well as our manual window selection algorithm

II. Analysis

Data Exploration

A typical captcha in the dataset is shown below.

All the images of the captchas are present in a single folder. All files have extension PNG. Text present in the captcha is written in the name of the file. So the name of the above image will be 226md.png

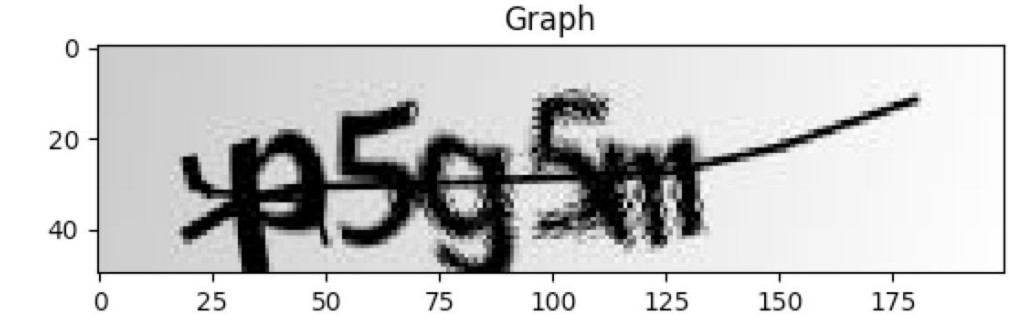
Another important characteristic of these images is that all the characters start at x-axis value=28. So this dataset is pretty consistent. Also almost all characters have similar width. With certain programming we can extract a single character from image because of consistency of the character width.

Number of characters in the image are exactly 5. All images are 200x50. All images are also grayscale. Average character width of all characters is about 22, while range of width is about 18-25 pixels.

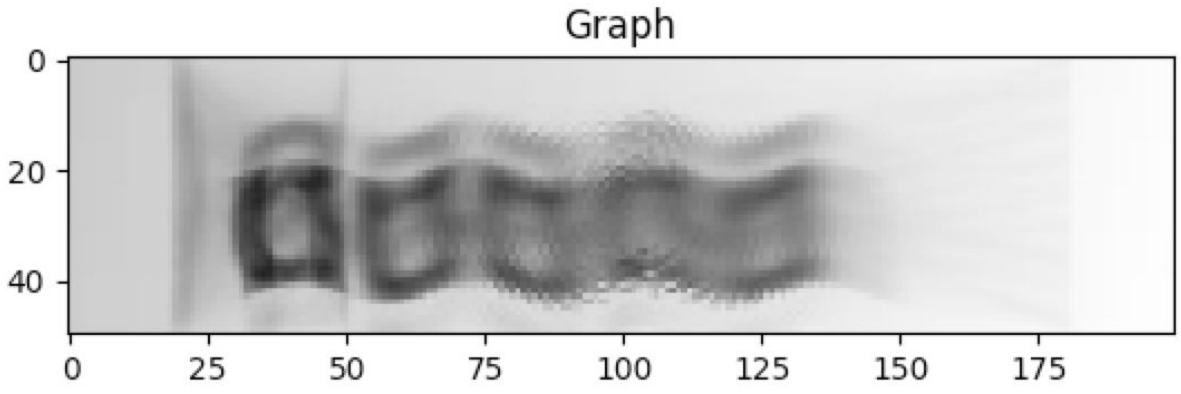
Only problem with this dataset is the absence of some characters. Here is the list of all characters that aren't present in any of the captcha image: a,h,i,j,k,l,o,q,r,s,t,u,v,z,0,1,9

Exploratory Visualization

Here are some samples of images.



Another visualization I have done is taking an average of all images and visualize resultant image, which is basically summing up all the values for each 2D cell and dividing them by total number of addition. Which results in the following image.



This image makes it pretty clear that there is a pattern in the position of characters and they almost follow an average width and place. All the captchas start at the same position and end at almost the same position. Almost rectangle like shapes show corresponding character positions.

Slightly darker tone in the beginning of the first rectangle in above averaged image shows higher probabilities of character lying there, while a more blurred tone at the end of the above image represents a dangling but still high probabilities of character to be found in that rectangle.

Following tables contain number of times each character occurs in captcha images. We see that some characters if they do occur more than 200 times otherwise they don't occur at all. With exception that n occurs 540 times.

a	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**	**k**	**l**
0	247	276	269	245	277	281	0	0	0	0	0

m	**n**	**o**	**p**	**q**	**r**	**s**	**t**	**u**	**v**	**w**	**x**
282	540	0	259	0	0	0	0	0	0	244	271

y	**z**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
240	0	0	0	270	271	289	288	267	261	272	0

Algorithms and Techniques

For CNN part of the problem, I need to identify characters in the image. But mere identifying characters is not enough. Their sequence and frequency is also important. Which it thought to be achieved by manual sliding windows.

Input vector of the CNN is the array of image vectors while output vector is an array of size 36. Where 1 represents presence of a character and 0 vice versa. So for a captcha with "abcde" in it output vector will be [1,1,1,1,1,0,0…..,0]. If an image has 5 a's then its output vector will be [1,0,0…,0].

CNN used for above problem has 3 convolutional layers, with MaxPool2D layers immediately following it. Dropout layers with value .2 are also added after each maxpool layer to evenly distribute learning in internal layers. At the end of the neural network a fully connected layer of value 36 is attached. Which is followed by Activation layer.

1st layer of the convolution layer extracts basic patterns such as horizontal, vertical, diagonal lines of the image. While the 2nd layer of the convolution identifies slightly more complex

patterns such as circles, ovals, arcs, curves. 3rd layer of the convolution finally able to identify complex shapes of characters, it for example identifies patterns of arcs, curves. 3rd layer of convolution basically identifies if multiple shapes that 2nd convolutional network found are present together, thus it is able to identify complex patterns in the image.

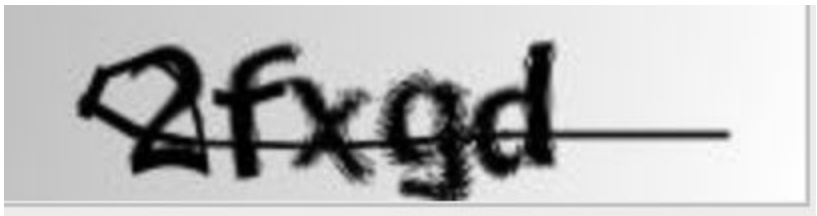
Based on the presence of these complex patterns our loss function help us choose which patterns to look for in the image for a particular class to be present. For example if out CNN is to search for 8. Then it will look for a pattern where 2 circles are together one over other.

Since patterns in the characters are not as complex as some real life objects such as cats ships trees etc, neural network with 3 convolutional network should sufficiently classify characters.

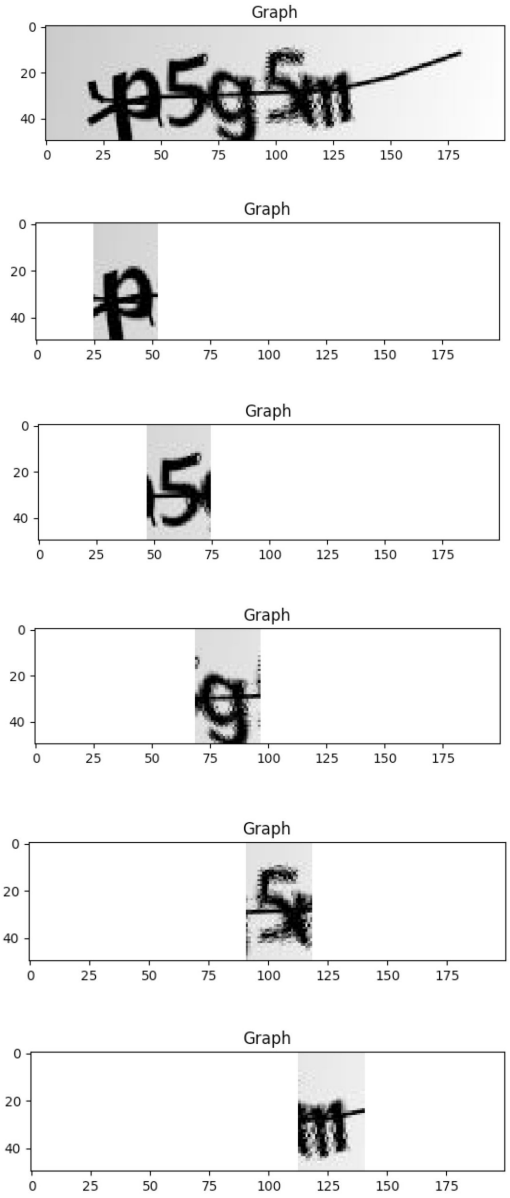
Test train and validation sets each have 20%, 60%, 20% of the data respectively. All the trained models are saved in the folder named “saved_models”

Sliding window algorithm basically starts with a small cut which gradually increases in size towards positive x-axis until CNN identifies a character with significant probability inside the window. Once the character is identified, it is added to the output text and window sliding continues in the rest of the image.

For CNN I set on to follow 2 approaches to see which one yields better results.
In **method-1** I took the whole image in one go and passed it through neural network, with corresponding output vector.
In **method-2** I used the property of the image dataset, that all the characters have almost equal width and start at the same x-value. This property can be seen in the images below.



Using this property I extracted 5 character images from single image. This way I had total 1070*5 character images to train on. I used tolerance of +-3 pixels for each character in case some characters have larger/smaller widths. Images extracted this way were trained on the CNN. the output vector of such images only had one value in the array set to 1 and all others were set to 0. For example if I extracted image of ‘b’ then corresponding output vector is [0,1,0,0….,0]
Some samples of the extracted and the original image are attached below.



In all I had several variables to work on which are: Activation function (“sigmoid”, “softmax”), loss function(“binary_crossentropy”, “categorical_crossentropy”) and classification type(single character, multi image) OR (multi-class--single label, multi-class--multi-label).

Benchmark

I will be benchmarking **method-2** against **method-1** of the previous section. I.e. I will measure performance of single label vs multi-label classification.

I will run all 3 parameters(activation function, loss function, classification-type(single vs multi label)) to see which parameters get along well.

So in the end with benchmarking, we will be able to assess things such as which activation function and loss function is better suited for multi-class, single label classification.

III. Methodology

Data Preprocessing

Data preprocessing was only needed in the form of extracting out characters from the image, and forming input and output vectors to the CNN. Otherwise this dataset has consistent data, with all images in the same format and resolution and having the same number of characters in each image. Dataset is also clean in respect that all the characters on average are within a small range of width.

While extracting image of a character rest of the image is turned white by setting grayscale value 255. This is also shown in the images in section 2.

Implementation

For the first part of the problem, that is identifying right characters in the image, CNN is trained on the image dataset by varying 3 parameters-(activation function, loss function, multi-label vs single label classification).

One advantage single label classification has over multi-label classification is the presence of more data points because one image is divided into 5 images of individual characters. However we see that it is just part of the same dataset.

With above parameters CNN was trained and following results were obtained.

Single/multi character	Activation fn	loss fn	Training loss	Training accuracy	Test loss	Test accuracy
multi	softmax	categorical	69.25159	0.0058139535	68.38892	0.004672897
multi	softmax	binary	2.2401984	0.8607880 5	2.2473454	0.8603323
multi	sigmoid	categorical	21.764206	0.1569767 4	20.673409	0.1822429 9
multi	sigmoid	binary	0.3111148 2	0.8603036	0.3155950 3	0.8627985
single	softmax	categorical	15.308409	0.0502336 44	15.410083	0.0439252 33
single	softmax	binary	0.8905695	0.9444433 5	0.8905699	0.9444432
single	sigmoid	categorical	5.766441	0.0490654 22	5.564681	0.0439252 33
single	sigmoid	binary	0.0151853 95	0.9958787	0.0145130 41	0.9958203

Referring above table we see that single character extraction with sigmoid activation function and binary_crossentropy loss function gives best results. Thus for CNN character identification, single character images with sigmoid activation and binary_crossentropy loss function is chosen as out model. Which gives pretty decent results.
Point to note here is that since we are going to use neural network in sliding window fashion multiple times thus accuracy thus obtained is further attenuated in the final results.

To assess the sequence and of the characters following approaches were used.

In the first approach I started with a window of minimal size from the start of the image. A window is basically a part between 2 vertical cuts in the image. Then I used CNN to identify if a character is present in the image. If a character with significant probability is suggested by CNN it is added to the output list, and window is again initialized for the rest of the image. This approach is gave a really bad accuracy of around **20** percent. Which is much less considering that CNN is pretty accurate in classifying characters.

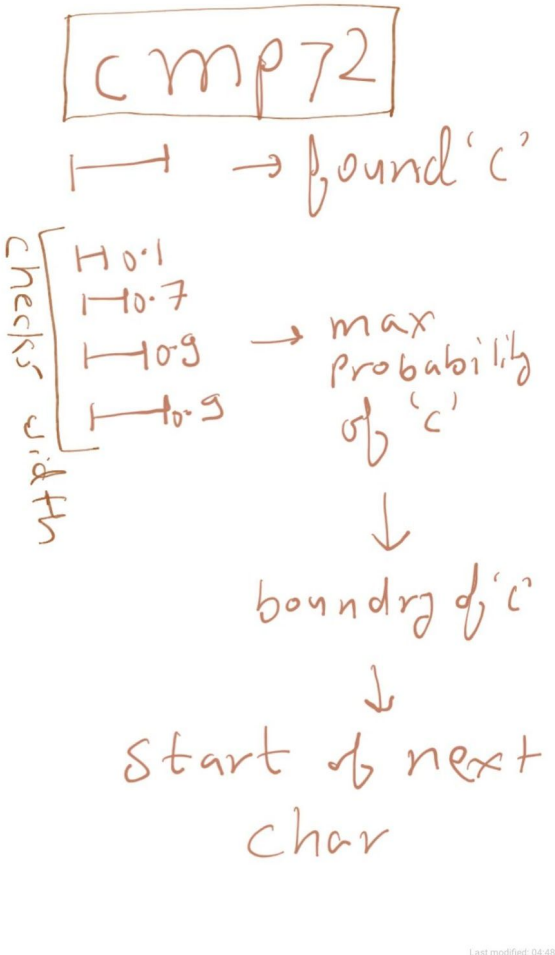
On further investigating it was found that some characters such as **m** and **n** are very similar looking so as soon as sliding window encounters initial part of **m** it classified it as **n** and continued with the rest of the window. This is shown in the images below.

Same is also true for a lot of other characters such P and l, because as soon as CNN encountered vertical line of P it classified it as L and. Presence of noise also makes it hard for classifier to extract character from the limited window because it is hard to distinguish between noise and part of character, For example below window of image of P is classified as 7 or Y depending on the cut.

Thus to identify correct image of the character it becomes necessary to have a wider view to separate out noise as well as to distinguish between characters such as m and n.

That brings to 2nd approach, In this approach similar to single character extraction from image, each character is assumed to be of average length and character is extracted from the image using properties of the dataset. So in all 5 character images are extracted and run through CNN. This algorithm is basically using our CNN in a similar way as that of we trained it.
Output of all the 5 CNN is combined to form the final output. This algorithm gives the accuracy of around **70** percent, still much less than what is acceptable.
On investigating it was found that some characters are slightly larger such as m, while some are smaller in width such as f, x, 7, c which is causing some characters to bleed into wrong window. So even if this behaviour occur once, the entire output of the captcha becomes wrong.
Presence of noise also caused issues in some of the character identification.
Problem with 2nd approach is shown in below image, which is basically wrong assumption of character width. Representation of bleeding of m in the image of p is shown.

Finally to further improve accuracy a combination of above 2 approaches was used. Where First a character is read using 2nd approach. Once the character is identified its width is assessed by sliding window gradually until the character probability reaches maximum. Once the character probability reaches maximum, that is thought to be the boundary of the character. Now next character is searched from the boundary identified of the previous character. This method solves the raging problem of 2nd approach where we encountered problems due to incorrect assessment of character width. Following this approach accuracy of **92.71** percent is achieved. This approach is shown in the image below.



On investigating some of the incorrectly read images, presence of noise is found to be the primary reason. As shown in the images below where image of e looks like x because of strike throughs.

Refinement

As stated in the previous section, much of the improvements were made in the window selection and character width assessment procedure. For CNN 3 set of parameters were run in loop one after the other to arrive at most suitable parameters.

However window selection algorithm was beset by a lot of edge cases because of character similarities, slightly varying widths and presence of noise. With changes in window selection algorithm the accuracy was improved from around 20 percent to 92.71 percent. Which is significant improvement.

IV. Results

Model Evaluation and Validation

To prevent model from bias to the input; train,validation and test were divided in 60, 20, 20 percent respectively. Corresponding test and validation accuracies were measured to ensure they aren't far apart.

In the final model single character images were given as input. Since the character images were not present at a single location, for example image of a character 'p' is present in the beginning in some captchas while it is present in the middle in some other captchas. This ensured that our model doesn't look for images of 'p' at just one location.

Dropout layers were added after maxpooling layers to train all the filters in the convolution layer.

The results were validated with test data, which wasn't used for learning and as shown in the table above test and validation loss and accuracies are quite close.
The weights which were obtained after learning, are saved in the saved_models directory. Those are the weights which generated data present in the comparison table of parameters above.

Although accuracy of the dataset which was provided are quite good. Presence of characters at some very specific locations might make our model too specific to those locations and might ignore the rest of the image where characters are usually not present if we run our model for a very high number of iterations.

Some cases where this model might not work as efficiently as for images provided in the dataset is when characters are of smaller/bigger size. Because while training our model translation, rotation, and scaling(zooming) wasn't used for images. Another case this model can backfire is when color images for captchas are used, provided dataset only has grayscale images.

To accommodate such cases data set can be augmented by translating, zooming in and out, rotating already provided images. That will make our model much more robust to the slight changes in the input.

Justification

Results of single-label classification were found to be more accurate than multi-label classification, which is expected because in single-label classification rest of the image other than character was masked, so it made easier for single-label classification to focus on key characteristics of the character.

Overall in all cases single-label classification was every time ahead of multi-label classification, however when multi-label classification was used with sigmoid activation and binary loss function it achieved accuracy of around 86 percent.

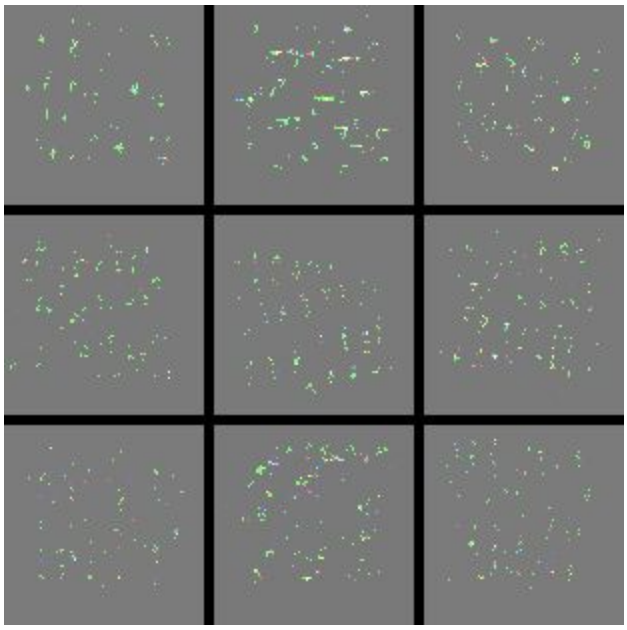
V. Conclusion

Free-Form Visualization

For a multi-class--multi-label classification binary_crossentropy loss with sigmoid activation works really well. This is also validated in this article.
For multi-class--single-label classification also binary_crossentropy with sigmoid classification works well as it is just a special case of multi-class--multi-label classification. In fact a lot of article term sigmoid with binary loss as multi-label

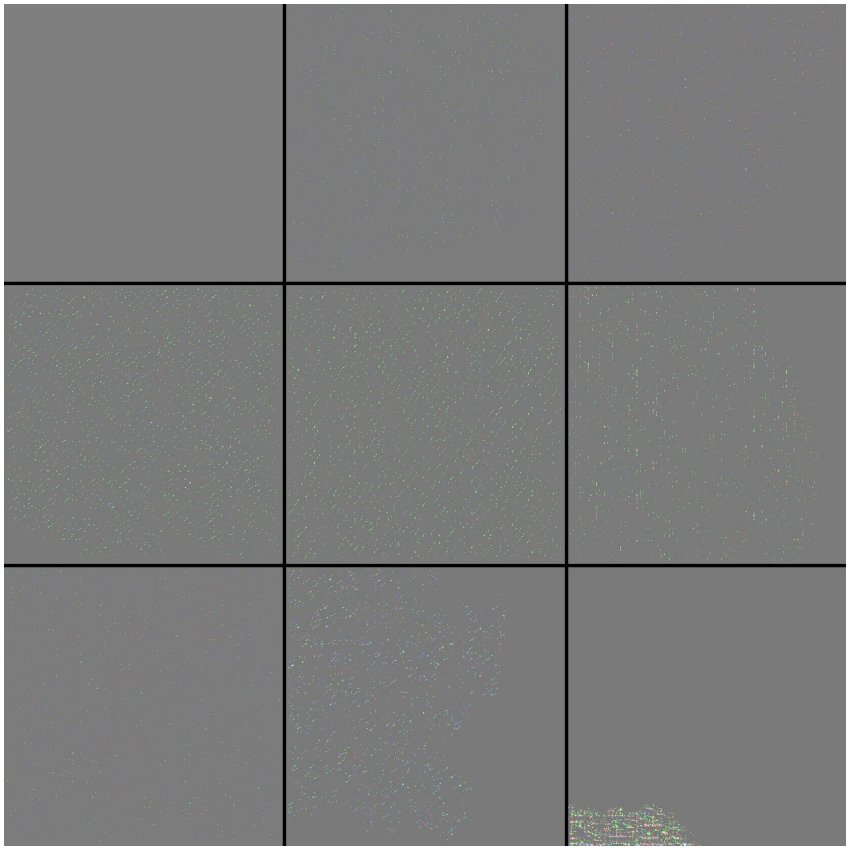
classification. Which is also validated by our findings recorded in the comparison table in the methodology section.

Following image is obtained on evaluating final convolutional filter.



Because images aren't as complex as real life objects, relatively simpler patterns are observed in the filter, unlike very complex patterns observed in VGG model filter.

On evaluating 1st convolutional layer following images is obtained.



Here on extreme zooming we see that it has simple horizontal, vertical and diagonal patterns which is expected from initial layers of CNN.

Other thing to notice here is how sparsely filled these filters are. Probably this could mean that CNN that we have used can recognize a lot many objects of more complex shapes, than what we have used it for.

Reflection

Without using neural networks with object localization has its own challenges and edge cases. Object localization neural networks are free from these edge cases and noises, which makes them a lot robust.
Because number of classes were a lot less, I was able to find the cause of missing edge cases, which becomes difficult when a lot of complex classes(objects) are involved in classification.
I found coming up with final set of parameters to be quite difficult for neural network, initially it seemed like this problem can't be solved with neural networks. However entering right parameters fixed it for me. I learnt that just making neural network with

convolutional layers is not enough, we also have to think of correct metrics to be used for problem or classification at hand.
Another difficulty I faced is in coming up with better window selection or character boundary selection. Which is a problem of object localization, which I manually coded by moving window around.
From initially getting 20 percent accuracy to reaching 92.71 percent is significant improvement.

Interesting thing about this project is that I came across object detection, localization, semantic segmentation and instance segmentation algorithms. Which also use neural networks extensively.

Improvement

I would definitely like to dive more into object localization and segmentation algorithms as well as noise reduction in the image.
I can think of following improvements to the solution I have implemented.

- Preprocessing images using noise reduction algorithms.
- Enriching dataset by translating images.
- By directly using object localization algorithms.

References

1. https://www.dlology.com/blog/how-to-choose-last-layer-activation-and-loss-function/
2. https://hackernoon.com/latest-deep-learning-ocr-with-keras-and-supervisely-in-15-minutes-34aecd630ed8?gi=221309870724
3. https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/object_localization_and_detection.html
4. https://www.kaggle.com/fournierp/captcha-version-2-images
5. https://en.wikipedia.org/wiki/CAPTCHA#Machine_learning-based_attacks