CAPTCHA Recognition Using Convolutional Neural Networks

A Deep Learning Approach to Surpass Alphanumeric CAPTCHAs for Data Scraping

Summary:

CAPTCHAs are widely used to restrict unauthorized access. This project focuses on automating alphanumeric CAPTCHA recognition using a customized convolutional neural network (CNN) to accurately predict alphanumeric characters. 1700 images were scrapped and annotated and then divide into two chunks of 1450 and 250 for training and testing respectively. The model was trained for 100 epochs with loss of 0.125 on training data and 0.567 on testing data. The model was then used to scrap data(88000 iterations) where it's accuracy turns out to be in the range of 88 and 91 percent with an average processing speed of 14 images per second.

Key Technologies Used:

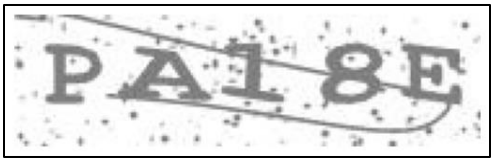
Python, TensorFlow/Keras, Selenium(for captcha and data scraping), Kaggle (for training environment).

Made by:

Ahmad Shafique

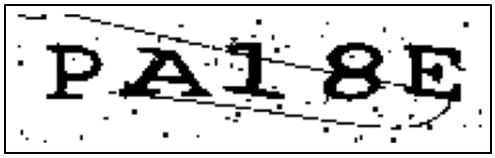
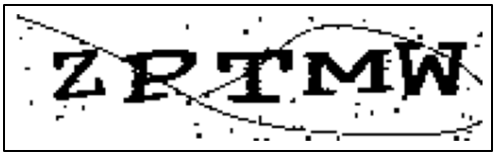
Pre-Processing Flowchart

Original Captcha

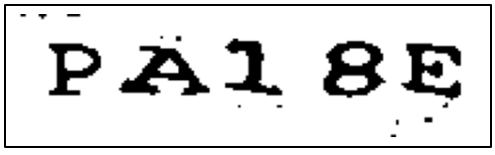
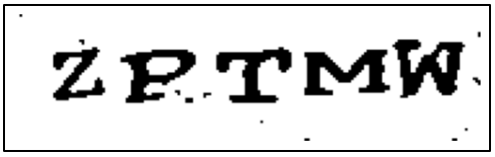



Grayscale Format

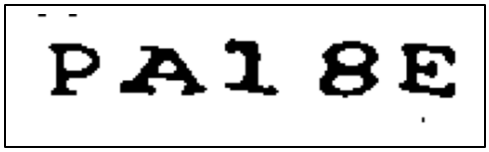
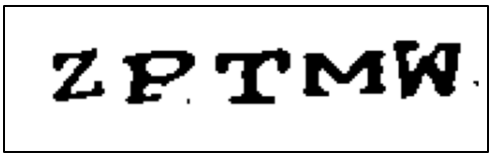

Binary Thresholding

Median Blur

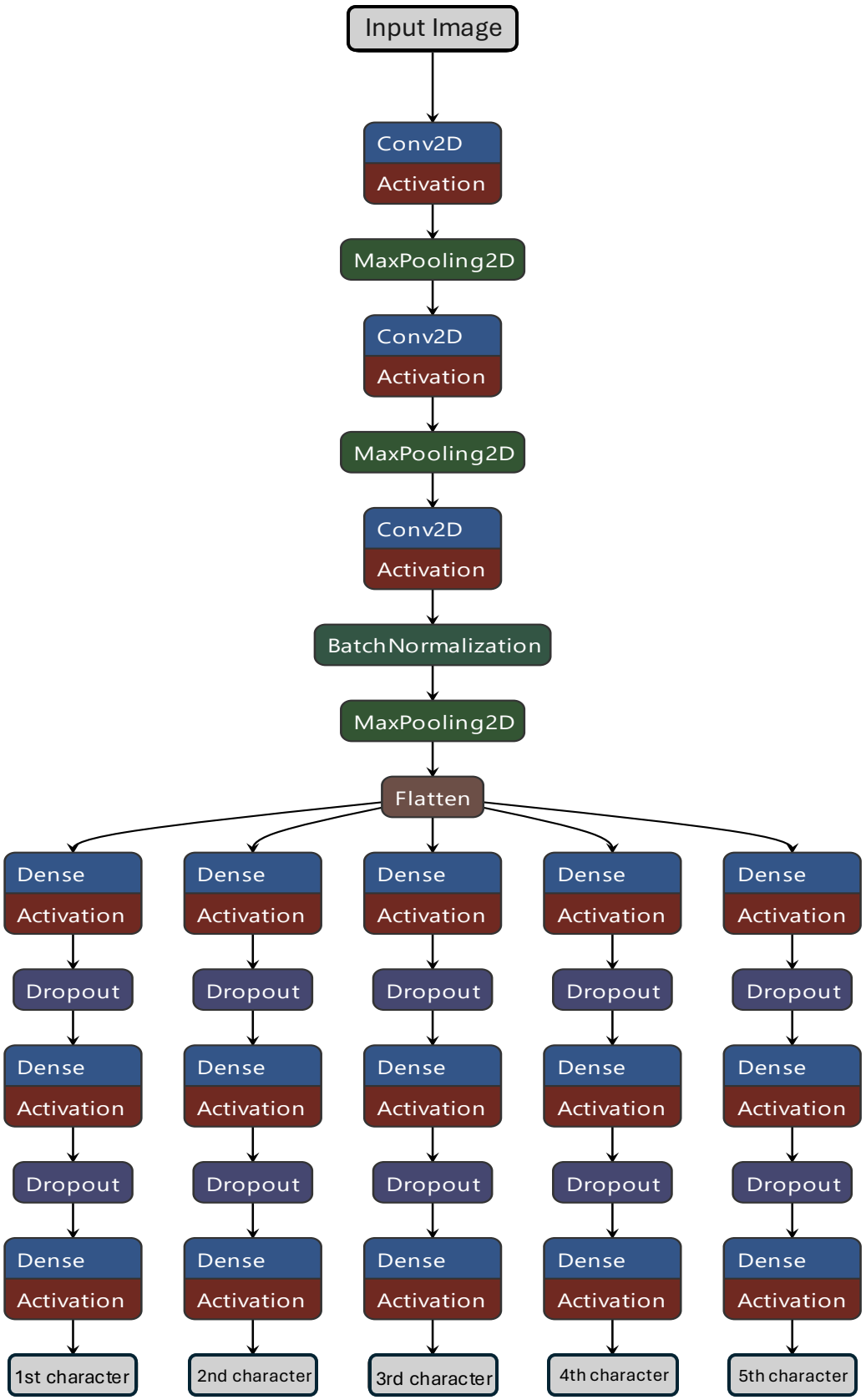
Median Blur

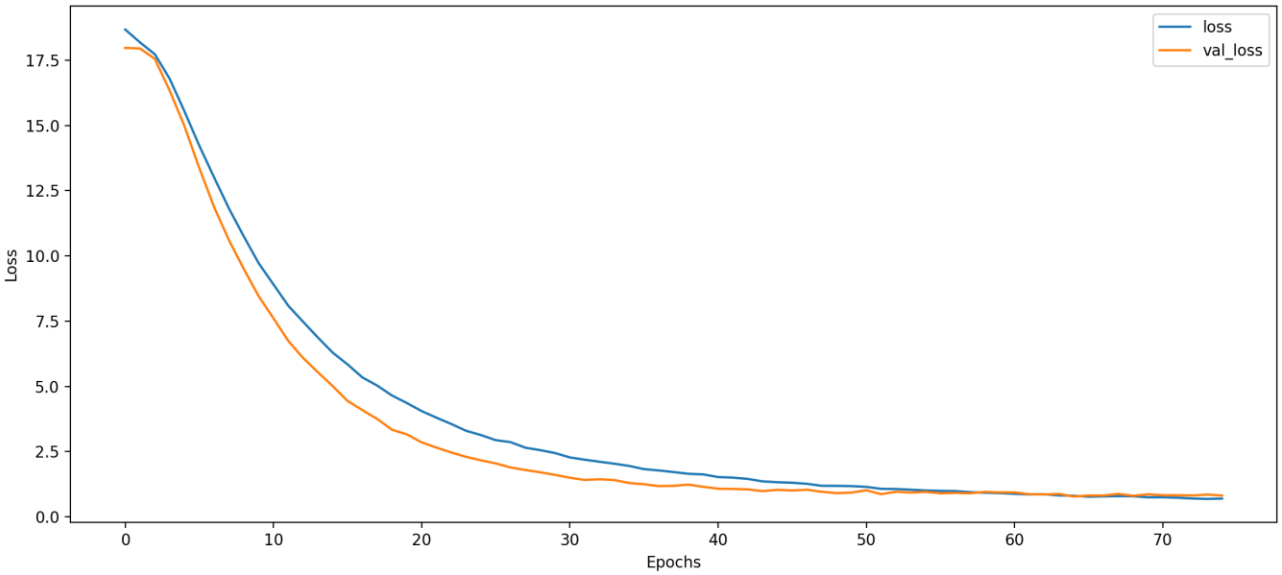
Predicted Labels

PA18E	ZPTMW

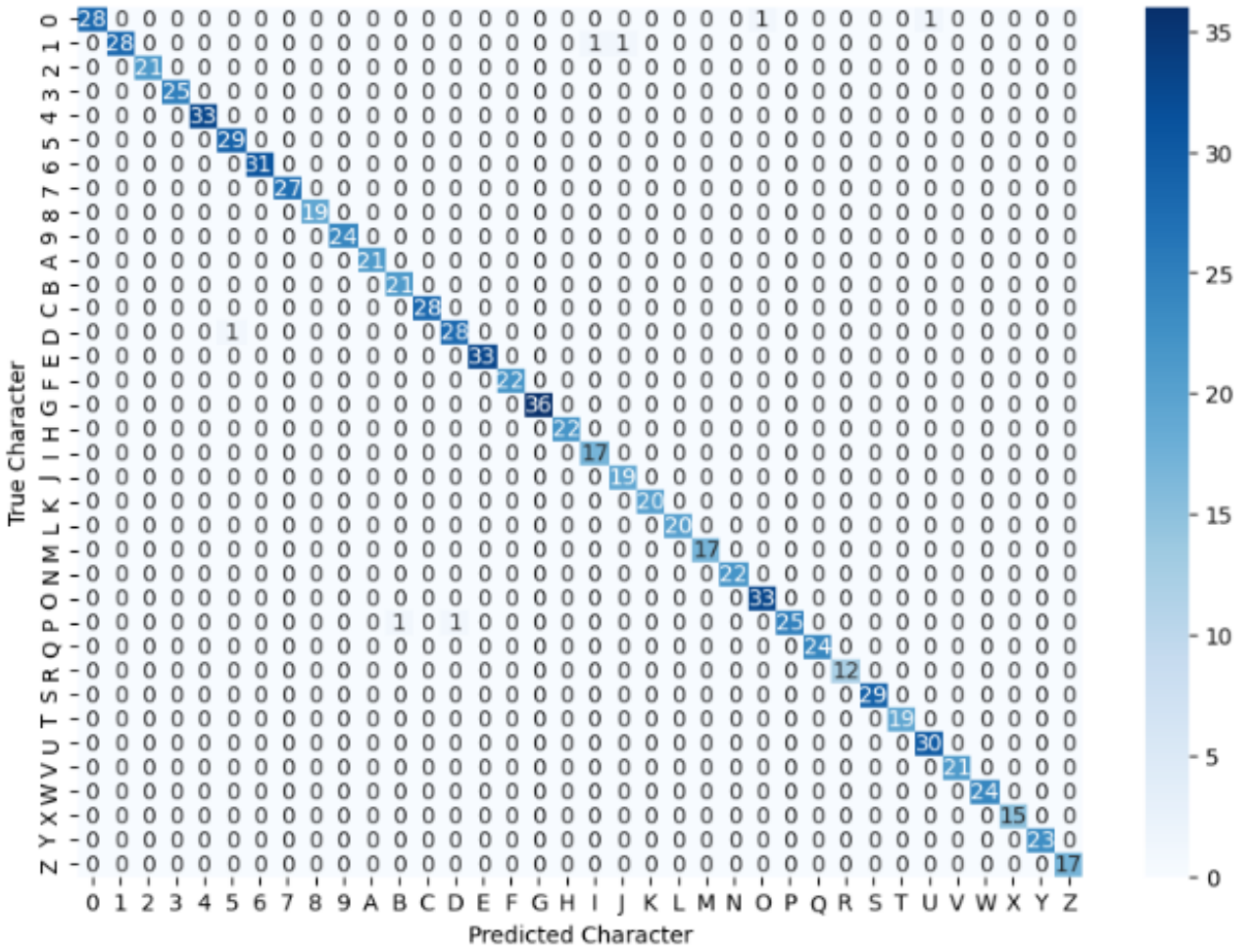
CNN Architecture



Training and validation loss curve



Character-level Confusion Matrix



Conclusion

Initially, when the model was trained on 200 images, It had really high accuracy on training data but low on testing data. The reason was underfitting, where it focused on memorizing the labels instead of learning it. As the training data kept on increasing, the model's accuracy improved. The final model is trained on 1450 images and tested on 250 images before being in data scraping script.

When we talk about the accuracy, it's also important to understand why it matters so much in the production code. In our case, there were a total of 88000 iterations(meaning it had to correctly detect the captcha 88000 times). On average, the code took approximately 1.3 seconds to complete 1 whole iteration. If we assume the model's accuracy is 100%(highly unlikely in real-world applications), the total time taken by the script would be 32 hours. In that case, if the model's accuracy lowers by 3%, the time will increase by 1 hour. This ratio keeps on increasing if the iterations and run time is high.

Another factor that affects accuracy is the characters present in the image. If we look at the confusion matrix, we can see that there were few characters on which the model's accuracy was too low. Now, if those characters keep on repeating in production, the model's accuracy will drop drastically. This is one of the reason, why model's accuracy drop down till 88%(although it went up till 91% and, on average, it's accuracy was not far from 90%).

Some of the characters that were challenging for the model to distinguish are mentioned on next page.

0 VS O

I VS 1

Z VS 2

U VS V

Here are few example where it was hard for the model to differentiate between the mentioned characters. The cherry on the top was the curvy lines randomly drawn on them, which makes it difficult to preprocess the images. Sometimes, the lines were over the text, so when median blur was applied, the lines became part of the characters; ended up changing the character's corners.

Concluding this, this project as a whole was a valuable learning experience for me. Some of the skills that I learnt from this was data scraping(image as well as tabular data), data annotation, model architecture and training.

In case of questions and queries, please comment below or reach out to me at Ahmad Shafique.