IITB Research Internship Assignment Report

Handwritten Form Reader Web App

Rohan Kapoor
Batch 2
22/08/2020 - 31/08/2020
rkchamp25@gmail.com
GitHub: https://github.com/rkchamp25/Rohan_Kapoor_9599023170-IITB-Assignment-Jul-Dec2020-Batch2

Contents

1. Introduction

- Optical character recognition(OCR) is a popular computer vision problem. There are many challenges associated with solving this problem. A document image may come from a scanner or it could (commonly) be a mobile camera picture or some app (such as camscanner etc). Mobile camera pictures have many variations in them, such as orientation, zoom-factor, lighting conditions, camera quality, focus, lens-blur, background of the document such as table etc. It is also possible that the actual physical document is slightly damaged as well which makes the problem even harder. Such issues are very common in real-world scenarios and hence require a robust solution for the industrial application.
- There are many ways to solve this problem, one such common approach is a multi-stage approach where the system first locate all the region within image where text is written and then use high quality OCR to read the text which optionally further can be classified as one of the entities of interest such as Name, DOB (date of birth),Designation, Company Name, ID, URL, EmailID Address, City Name, Contact Number etc,.. There are many applications of such a system such as in Scanned Resume Parsing, Invoice Reading, Business Card Reader, ID Card Reader etc.

2. The Problem Statement

To make a high quality and robust OCR module similar to Google's Tesseract, Microsoft Form Reader etc. The input of this module will be an image which is a cropped region of a form's field and the output has to be the predicted text on the image. Also these fields will have boxes for each character.

Instructions

- First generate a dataset of around 50k images which are suitable for the given task i.e images should be similar to a form's different fields such as names, roll numbers, phone numbers, dates etc and the images should have enough variation.
- Use any synthetic text renderer of choice to generate images like SynthText(i), text renderer(ii) etc.
- Convert the dataset into tfrecords format.
- Train any attention based ocr model or a ctc based encoder decoder on this dataset and tune the hyperparameters.
- Change the model architecture and then again train it on the dataset.
- Compare the two models on public test dataset
- Create a web app using Flask and deploy it using Heroku

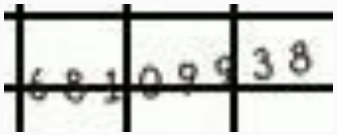
3. Basic Requirements

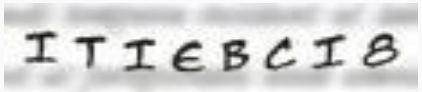
- I started by installing some of the things and on the way modified the packages according to the requirements of that step.
- I used anaconda for this assignment and had it already installed on my machine.
- So I created a new environment so that the packages do not interfere with others and so that I could use any version of a package I wanted. Also I used Python 3.7 for this environment.
- I used VSCode as my code editor throughout this assignment.
- These are some of the main packages I installed initially:
    1. tensorflow-gpu - as the base of the model
    2. numpy - for mathematical operations on higher dimensional arrays
    3. pandas - for taking care of the dataset, creating and manipulating data files
    4. matplotlib - for plotting graphs of loss and accuracy

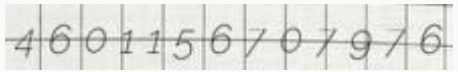
4. Data Generation

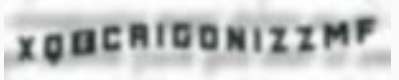
- For data generation I used the TextRecognitionDataGenerator(iii). This generator had a lot of options inbuilt and I experimented with almost all of the options. It also had an option of generating handwritten text but it was quite slow as it further used another model for this and the handwriting was also not upto the mark so I didn't use this option.
- First I installed all the given requirements of this tool. There was a conflict in the required version of tensorflow so I had to downgrade my version to tensorflow version 1.15. For normal text I added some fonts to its default font library and also removed some of the fonts which were very bold because in reality we don't fill forms in bold.
- Next I changed the length of the generated text and set minimum to 5 and maximum to 15.
- The generator generated around 130 images per sec when using all of the threads of the CPU and did not require the use of GPU. I could simply use all threads by using the '-t' argument.
- I used the '-m' argument for creating the images with padding as given to us in the sample set and used the '-k' argument for tilting the text at random angles. Finally I generated two datasets of 60k each (one more random and one similar to public test set) + 10k(random test set).

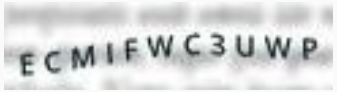
Samples

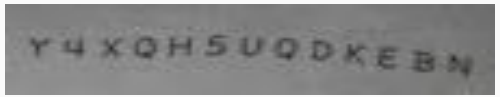










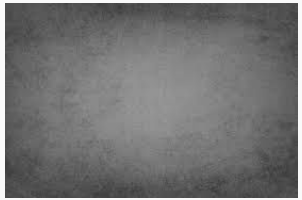


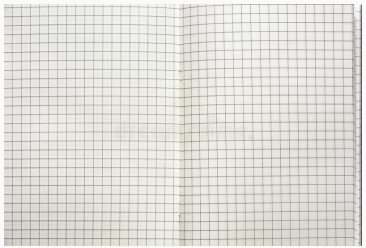
Challenges Faced

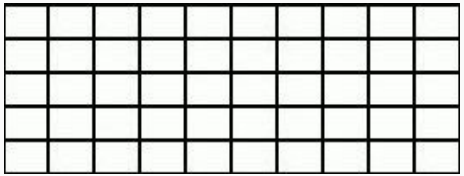
1. Here the main problem I faced was generating images that had boxes around the characters as the generator had no such option. There was an option to generate boxes later after generating the images by identifying the characters but it was not much viable and would have been difficult to implement and would consume more time.

2. After exploring all of the text generator's options and some time on the internet, I got the idea of using a background on which the generated text would look similar to text on the forms.

3. So I downloaded pictures of empty grids which would simulate the behaviour of boxes.
   I also downloaded some dark base for those images which were dark.







5. Attention OCR Model

- I used the given attention ocr model by emedvedev(iv) which uses a convolutional neural network (cnn) for the extraction of features or text from the images then it feeds into the recurrent neural network (lstm) for modelling the dependencies between the characters, predicting the future sequence etc and then finally the attention mechanism to use all of the states of the rnn in order to learn instead of just using the final output of cells.

- I looked at different files and connections between different modules to understand what was happening and checked out all the available options. I changed some options such as the maximum width and height of image to 650 and 45 respectively and maximum prediction length to 15.

- I started the training with the default settings only to realize later that the number of epochs had been set to 1000 and the speed with which it was training would take a number of days at least to complete so I had to cancel this training.

- So then I changed the number of epochs to 50. The learning rate was to 1 because it was using the AdaDelta optimizer for this so it didn't matter. I was using a batch size of 60 on a dataset of 60k and saved the checkpoint after every 1000 steps.

- I stopped this training at 37 epochs as I felt that the loss had decreased enough as it was around 0.2 but when I checked it on my test set it was not performing very good so I decided to train it further and then stopped training at 46 epochs because the loss had started to become constant. It was giving an accuracy of 76% on my testset.

- I had to discard this one as well because when the public test set was provided I realized that the images I had generated were having way more variation and distortion compared to those in the test set and also the model was performing poorly on that set giving an accuracy of around 38%.

- I generated a new dataset more similar to the public dataset and also kept the previous one so as to train my model first on the previous dataset and then training it further on the new dataset to make it more accurate as well as more robust.

- This time I set the learning rate to 0.07 because even though the model was using optimizer but if I would later train it on the new dataset then it will learn very quickly because then I would not be able to change the learning rate of the existing model and as a result it will change the learned weights very quickly.

Parameters

- Initial Learning Rate = 0.07
- Epochs = 28
- Batch Size = 60
- Max Dimensions = 650 * 45
- Max label length = 15
- Steps per Checkpoint = 1000
- Channels = 1(Grayscale)
- Dropout = 0.5
- Optimizer = Adadelta

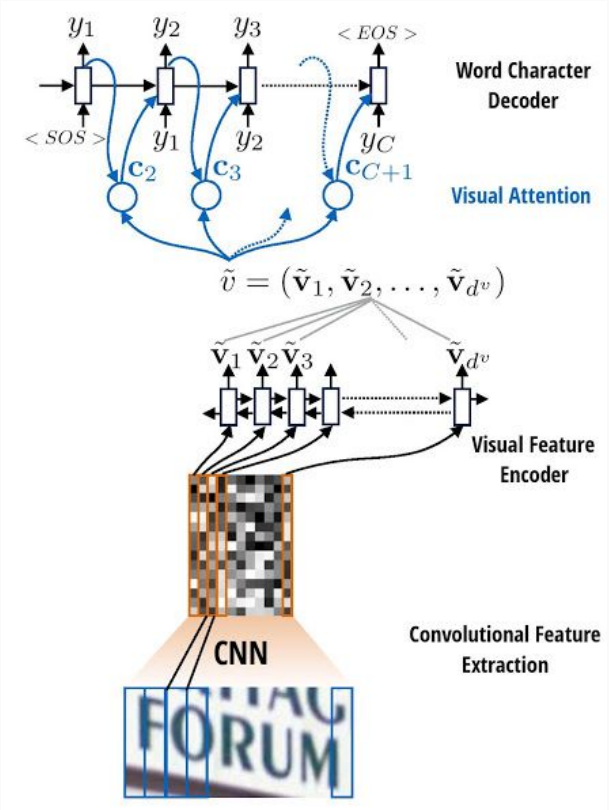


Fig. Model Structure

Performance

- The model was trained on a mix of custom data set of 60k images and another dataset of 60k which included the images from the public test set.
- The model was test on the custom test set(10k) as well as the public test set of around 1700 images.
- After 28 epochs the model was just learning things because if I trained on custom set the performance was better on the custom test set but decreased on the public test set and vice versa.

Total Time Taken for Training	9 hrs
Average Loss at the end of training	1.473
Avg. Accuracy on custom test set	65.99%

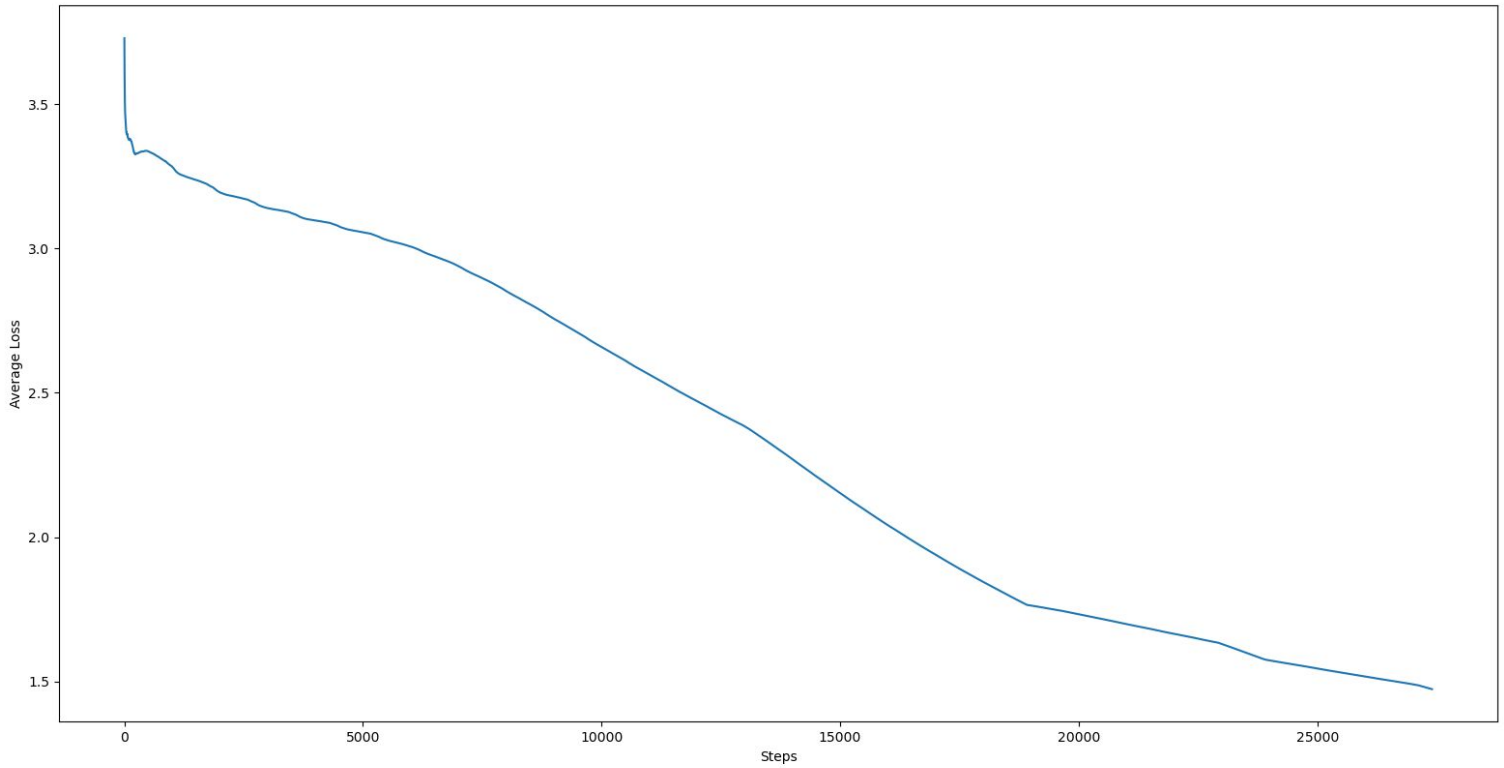


Fig. Average loss vs Steps plot for original model

Challenges Faced

- The training speed was the major problem in this. Since the model was training both the cnn and attention simultaneously it was taking a very long time. It was taking around 15 to 20 minutes for 1 epoch and therefore it was causing problems to experiment by tweaking different settings and finding the best one.

- Also I had to change some code for resolving the problem of maximum label length allowed because the decoder input units were set according to this and if the label of the image was greater than this then it was raising an exception. For this I changed the part of code which was taking input from the tfrecords file and set that it would store the image details and label only if the length was less than or equal to the maximum length or reject this image.

6. Modified Attention OCR Model

- First thing before modifying was to understand what was actually happening under the model, how was the data being passed from one module to another, what was the function of different modules, what can be a possible modification and what will be its effect.

- In the end I decided to replace the CNN module by a ResNet50 pretrained model because the function of the CNN was to just extract features from the image, it was a good idea if we could replace this with a model which has been trained on millions of images and thousands of categories. So the idea was to use what the model had already learnt and then use this for our task.

- The plan was to use the model as it is and if the performance was low then maybe fine tune it i.e. unfreeze few of its top layers and then using a low learning rate train the model but due to an error in the attention module when using the ResNet I could not use this. After a lot of experimenting and searching on the internet I decided to modify the existing CNN by adding more layers and making a network which will be similar to the ResNet50.

- So I added a new layer at the bottom with 32 filters, a 3x3 filter size and ReLu activation and two more layers of 64 filters, 3x3 as filter's size and ReLu activation, after the second layer. This was to simulate the effect of ResNet where we forget some part of information from the current layer and learn back the information of some previous layers. This makes the training more easier and more effective as we don't completely neglect the things learned by previous layers. For adding the outputs of the layers they had to be of the same shape.

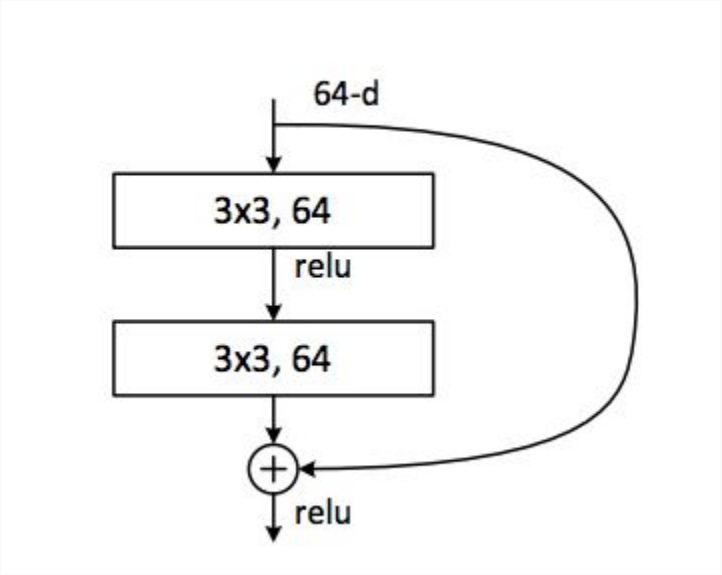


Fig. Skip Connection in a ResNet

Parameters and Performance

The parameters for this model were exactly the same as for the original model.

- Initial Learning Rate = 0.07
- Epochs = 26
- Batch Size = 60
- Max Dimensions = 650 * 45
- Max label length = 15
- Steps per Checkpoint = 1000
- Channels = 1(Grayscale)
- Dropout = 0.5
- Optimizer = Adadelta

- The model was trained on a mix of custom data set of 60k images and another dataset of 60k which included the images from the public test set.
- The model was test on the custom test set(10k) as well as the public test set of around 1700 images.

Time Taken	9 hrs
Average Loss at the end of training	1.359
Avg. Accuracy on custom test set	68.33%

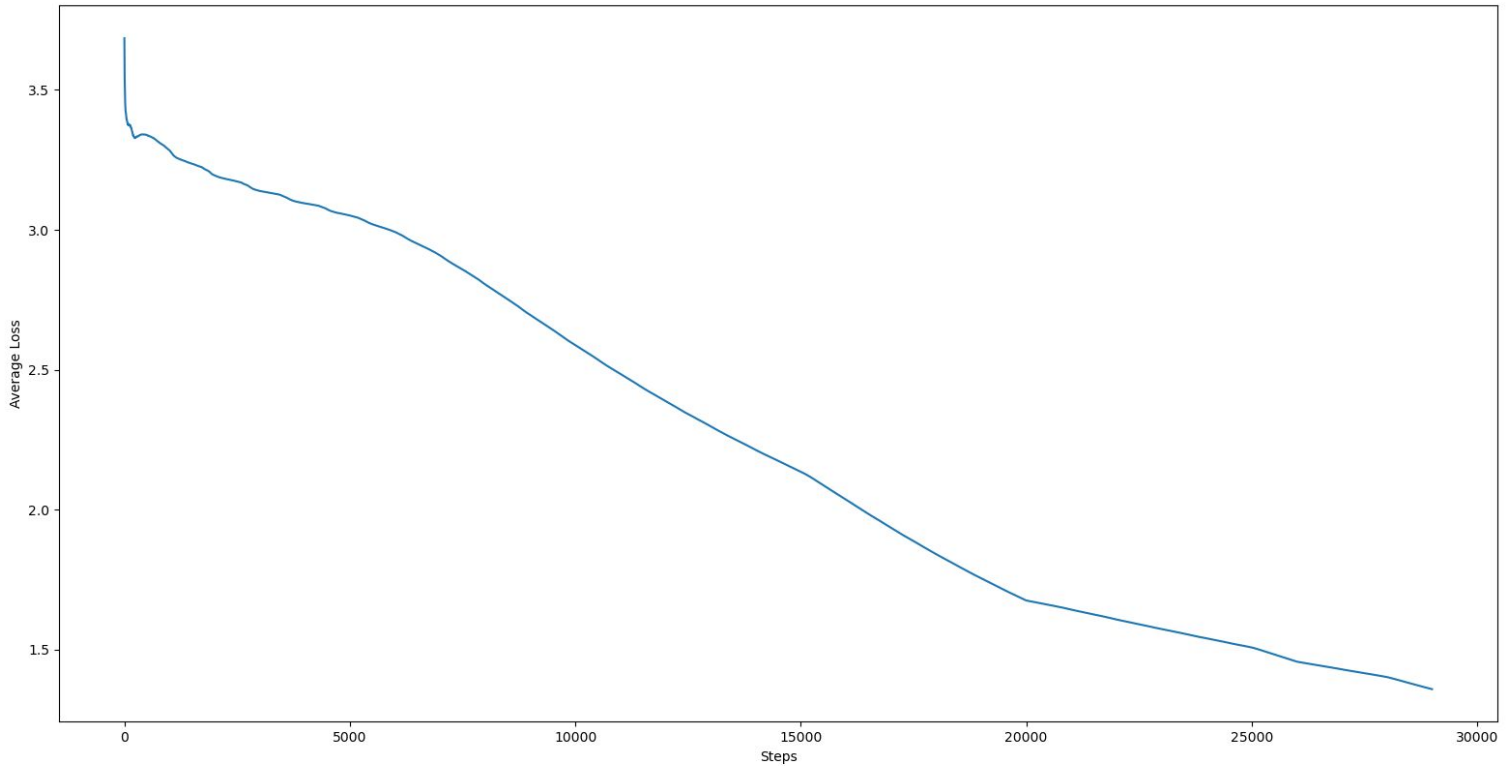


Fig. Average loss vs Steps plot for the modified model

Challenges Faced

- First problem was that I was taking the output of the last layer of the pretrained ResNet50 model whose dimensions were not matching with the dimensions which were required as input of the next layers. The maxpooling layers in this pretrained model were continuously decreasing the size of the input and in the end the length of one of the dimension required was 116 but it was giving 15 as its output. So for this, instead of taking output from its last layer I searched for the layer where the length of the dimension was correct and took output from that layer.

- After this there was a problem that I could not add any more convolution layers because it was using the value of some uninitialized variable. For this, instead of using the custom defined conv2d functions in the model I directly used the conv2d layer from the tf.nn module.

- Now the problem was in the attention module. Here also there was the same problem of using an uninitialized value. I compared the outputs of both original and this resnet50 model and could not find the difference and therefore had to ultimately drop the idea of using pretrained Resnet50 model.

- Adding more layers was also a problem. Whenever I was adding more than 3 layers of 64 filters each then it was demanding more gpu memory than it could allocate and hence this was not working. Also I had to add a certain number of layers in order to simulate a ResNet like effect. I had to add either 3 layers or 5 or 7 to add the residue in the layers above but since it was not possible to train the model with more of these layers I ended up adding 3 at the bottom part and then the old layers were on top of these.

7. Comparison Between The 2 Models

Original Model's CNN

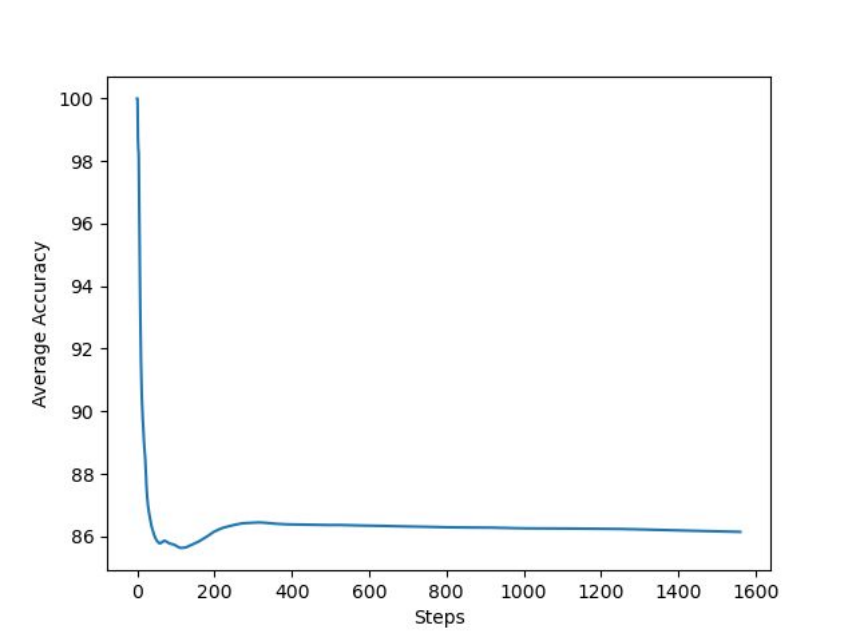
```
net = ConvRelu(net, 64, (3, 3), 'conv_conv1')
net = max_2x2pool(net, 'conv_pool1')
net = ConvRelu(net, 128, (3, 3), 'conv_conv2')
net = max_2x2pool(net, 'conv_pool2')
net = ConvReluBN(net, 256, (3, 3), 'conv_conv3', is_training)
net = ConvRelu(net, 256, (3, 3), 'conv_conv4')
net = max_2x1pool(net, 'conv_pool3')
net = ConvReluBN(net, 512, (3, 3), 'conv_conv5', is_training)
net = ConvRelu(net, 512, (3, 3), 'conv_conv6')
net = max_2x1pool(net, 'conv_pool4')
net = ConvReluBN(net, 512, (2, 2), 'conv_conv7', is_training)
net = max_2x1pool(net, 'conv_pool5')
```

Modified Model's CNN

```
net = ConvRelu(net, 32, (3, 3), 'conv_conv8')
net = ConvRelu(net, 64, (3, 3), 'conv_conv1')
x = max_2x2pool(net, 'conv_pool1')
net = ConvRelu(x, 64, (3, 3), 'conv_conv9')
net = ConvRelu(net, 64, (3, 3), 'conv_conv10')
net = tf.keras.layers.add([x, net])
net = ConvRelu(net, 128, (3, 3), 'conv_conv2')
net = max_2x2pool(net, 'conv_pool2')
net = ConvReluBN(net, 256, (3, 3), 'conv_conv3', is_training)
net = ConvRelu(net, 256, (3, 3), 'conv_conv4')
net = max_2x1pool(net, 'conv_pool3')
net = ConvReluBN(net, 512, (3, 3), 'conv_conv5', is_training)
net = ConvRelu(net, 512, (3, 3), 'conv_conv6')
net = max_2x1pool(net, 'conv_pool4')
net = ConvReluBN(net, 512, (2, 2), 'conv_conv7', is_training)
net = max_2x1pool(net, 'conv_pool5')
```

- The difference between the two models was that in the second model the CNN had 3 more layers all added at the bottom. These layers were added to simulate the effect of a ResNet but since the number of layers were not much the increase in performance was not so much pronounced.

	Average Accuracy	
	Custom Test Set(10k)	Public Test Set(1.8k)
Original Model	65.99%	85.67%
Modified Model	68.33%	87.60%



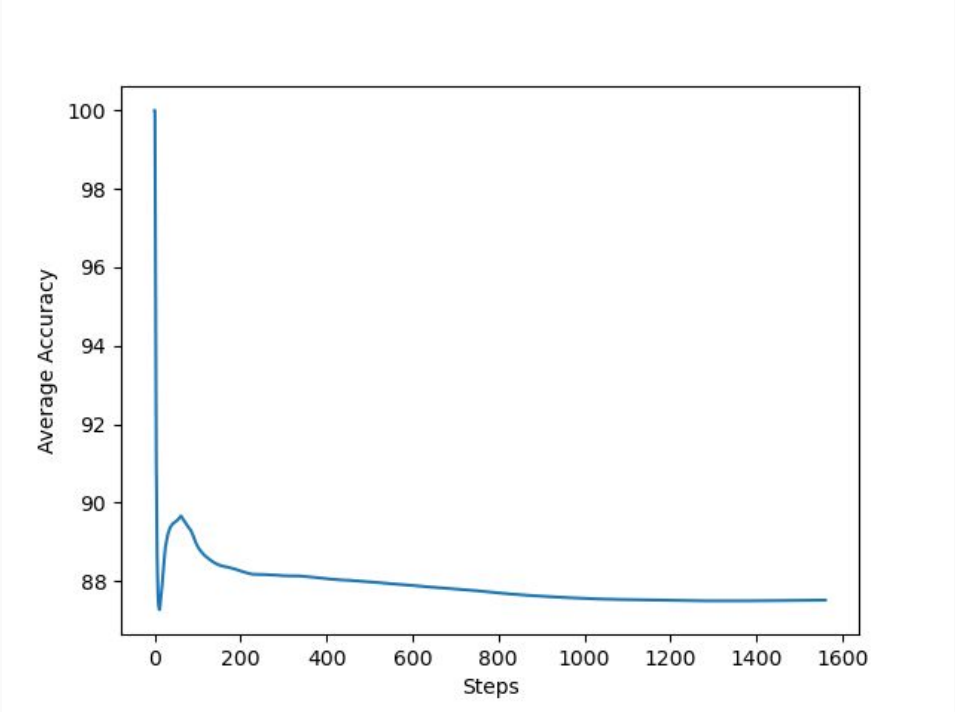


Fig. Average Accuracy vs Steps
Left (Original Model), Right(Modified Model)

- There was an increase in performance by ~3% which is not so big.
- Looking at the difference I feel that this can even be due to more number of neurons/layers in modified model and therefore more features were learned.
- I don’t feel that the addition of residue caused much of a difference because for that to happen we need this process to keep happening at least three or more times for its affect to be seen.
- The original model when trained beyond 28 epochs started memorizing and therefore if I trained on dataset 1 the performance on the dataset 2 decreased and if I trained on dataset 2, performance on the first one would decrease. On the other hand the modified model was not showing such affect and could have been trained for a few more epochs.

8. Web App

- The web app has been made using Flask in python and is hosted publically using the ngrok service.
- The app has a very basic front end part which consists of a simple form like format which takes in the URL of an image and then shows the prediction of the text written on that image and its probability.
- The app I made earlier required an upload of image from the local machine where it was being executed but then I changed it to take a URL of any image from the internet, then save the image and read the image and then finally predict using the exported model.
- The ngrok service is free and quite easy to use and we just have to run the app locally and then give that port number where the app is running to ngrok and it becomes publicly available.

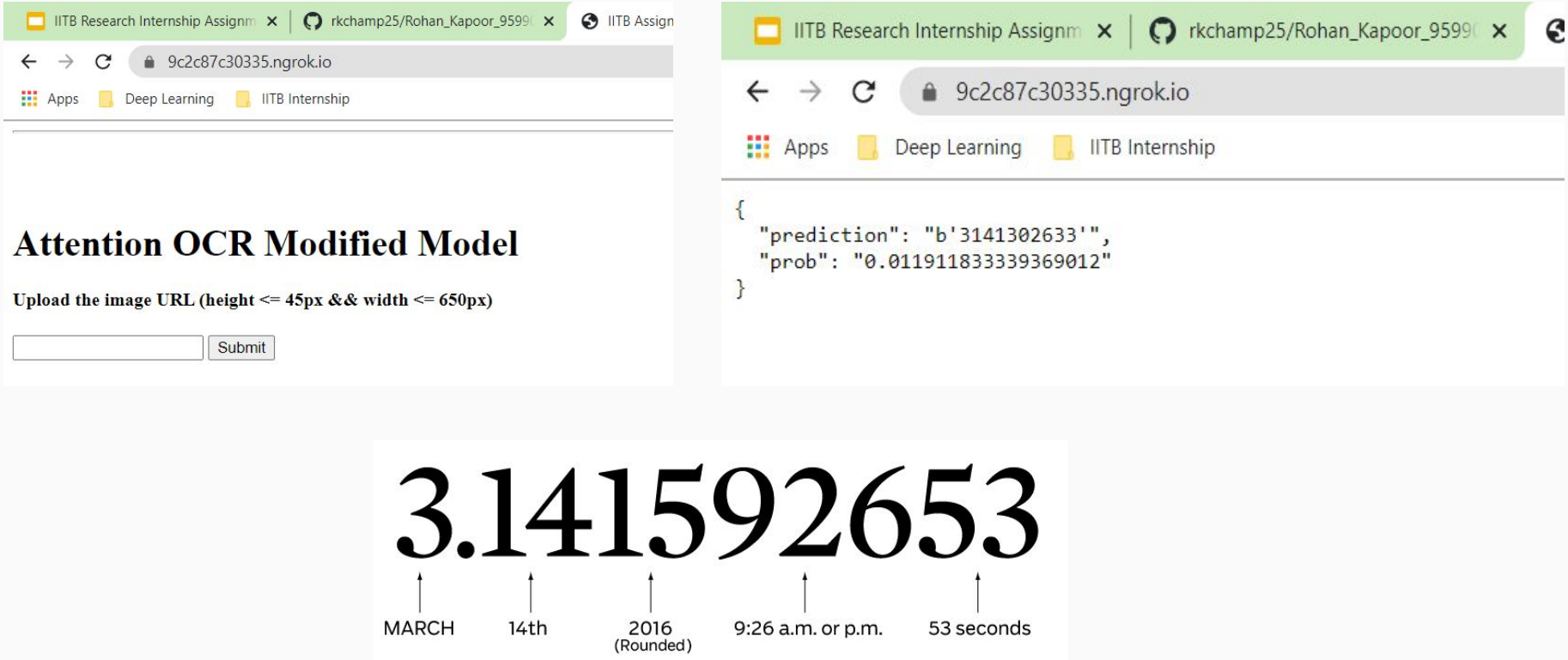


Fig. Home Page, Prediction Page of the application deployed using ngrok and image whose prediction is shown

Challenges Faced

- There were small errors here and there as it was first time I tried flask but there was nothing whose solution I could not find on the internet.
- I was stuck at the 404 error for some time but that was the issue of the html file not being found. So instead I moved the html code in the app.py file itself as it was much easier to handle. I returned that file whenever the request was GET and returned the predicted text when the request was POST.
- Earlier I tried to deploy the app using Heroku but there the allowed size was 500MB and my app was taking around 800MB maybe because of tensorflow gpu and the checkpoints that I saved in my repository and so I switched to ngrok.

9. References

- https://github.com/ankush-me/SynthText
- https://github.com/Sanster/text_renderer
- https://github.com/Belval/TextRecognitionDataGenerator
- https://github.com/emedvedev/attention-ocr
- https://blog.miguelgrinberg.com/post/designing-a-restful-api-with-python-and-flask)