

CSE 4/510: Applied Deep learning
Summer 2020

Final Project Report

Digit Recognizing Application

Team Members:

Rohith Kumar Poshala – 50320370

Sreekar Guggilam - 50318495

Topic: Deploying deep learning models as a Web Application.

Abstract:

In this project we have deployed an application on AWS server which lets user to draw a digit/number (handwritten digit) in the front end and gets the number recognized by the application correctly.

Introduction:

In the real world, humans handwriting is not as simple as we have in any of the available datasets (like mnist dataset). But as the main motivation of neural networks is imitate a human brain. The models which we build should be able to perform as perfectly as a human does. As we know that human visual system is one of the wonders of the world, we expect our convolutional neural network to be able to perform nearly as perfect as human. In this project we are going to fine tune the model to perform good on the real handwritten digits and we have tuned the model by testing it on the worst possible handwritten digits we could give.

Implementation:

1. Back-end:

Preprocessing of training and test (input) image data:

For this digit recognizing project, we had picking mnist dataset from Keras. Initially to train the deep learning model, the default train and test split of the mnist dataset has been taken and the training model was also giving us around 99.5% training and validation accuracies. But when these models are tested on manually handwritten digits, this model was performing very poor.

Dataset size: 70,000 images of size 28 X 28

Challenge 1: Deep learning models which were performing extremely well on training and testing data (default split) are poorly performing on manually given images of digits.

Action: Both training and testing data were combined and split again. When we fit same model used previously, validation accuracy was not that great as it had been previously.

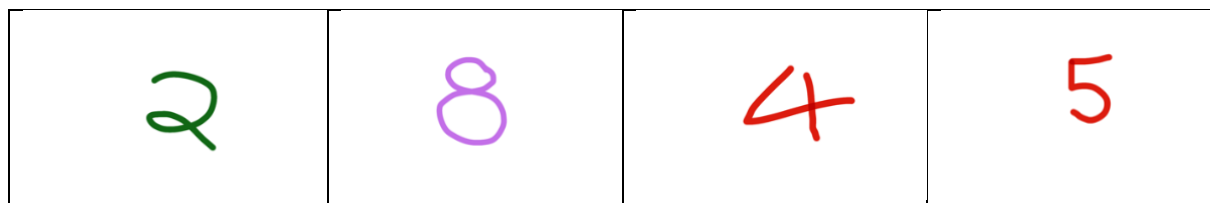
So, we had included extra layers and introduced dropouts to the model expecting better results. Doing this we could increase the validation accuracy but the performance on the manually fed digit images was not up to the mark.

Challenge 2: With this new model fit on the new split, model was performing very well on the half of the digits and extremely poor on the remaining half.

Action: Training and testing data has been split in such a way that they had equal proportions of every class.

Along with the above to increase the variance to the model, we had also implemented data augmentation on the training data and increased the training dataset size to 90,000. Data augmentation included shifting of the images and rotation till 30° (horizontal and vertical flips were not used as it doesn't benefit the current dataset).

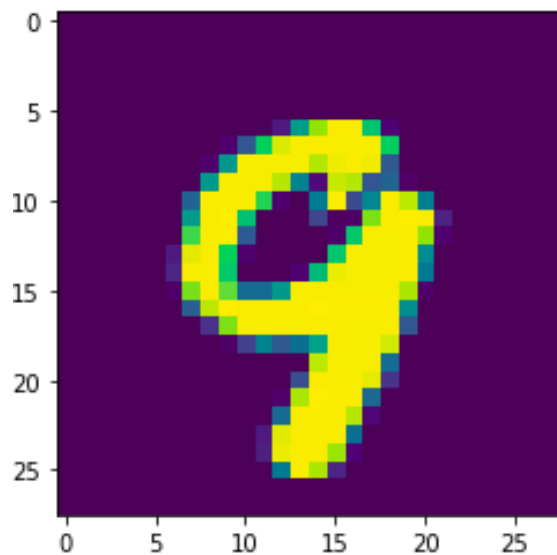
Challenge 3: Even though the model fit on this balanced data with additional training data generated by data augmentation was generalizing well. It was not performing well manually fed images like below. And here in this case, data augmentation and training on the increased training dataset was taking more computational time than earlier.



Action: After visualizing the samples with the help of matplotlib/cv2, we had observed that most of the images are already slightly rotated. So, rotating the sample image further was causing the digit to look like another digit (ex: 6 as 9, 2 as 5). So, it was clear that data augmentation was not helping us the model to generalize well rather it is causing an increase in computational time.

So instead of increasing the training dataset to make the model generalize well, we had converted the manually given image similar to the training and validation data set images.

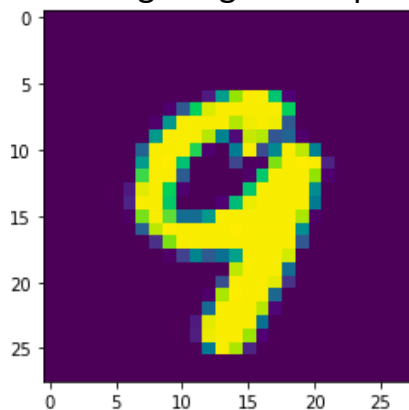
Training image : (shape 28X28)



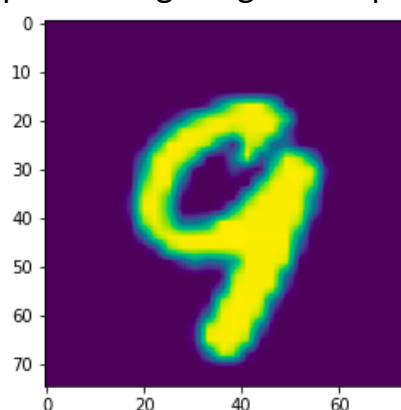
So, a function was defined to crop the test image and the cropped image was resized to a desired shape before implementing the trained model.

Resizing of the test image had been done to 75x75 shape rather than 28x28 in order to get a little better compressed image and 75x75 has picked as it doesn't overshoot the RAM allocated in the google colab. So, in order to make the model compatible to the testing data, we have also resized the image to higher dimensions from 28 to 75.

Sample training image of shape 28x28

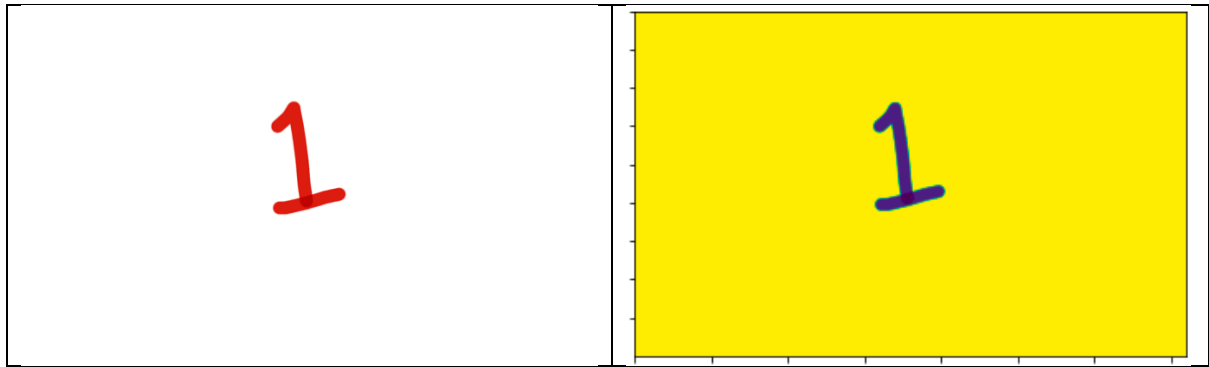


Sample training image of shape 75x75



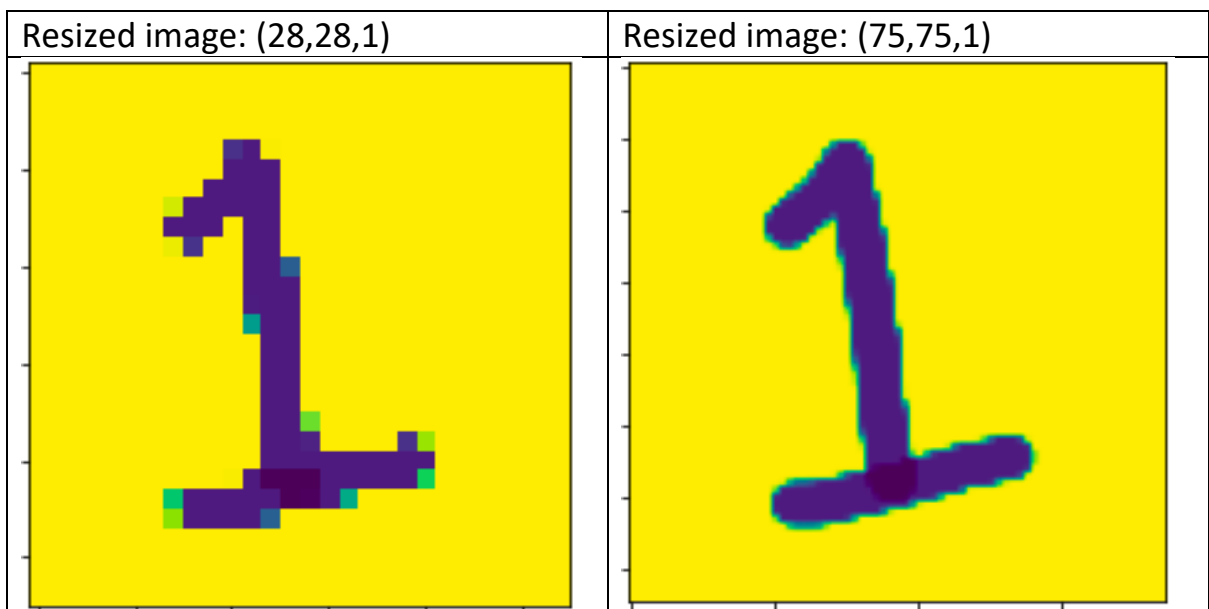
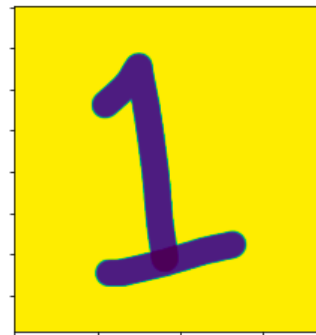
Test image:

Original image: (900,1400)	Gray scale image: (900,1400)
----------------------------	------------------------------



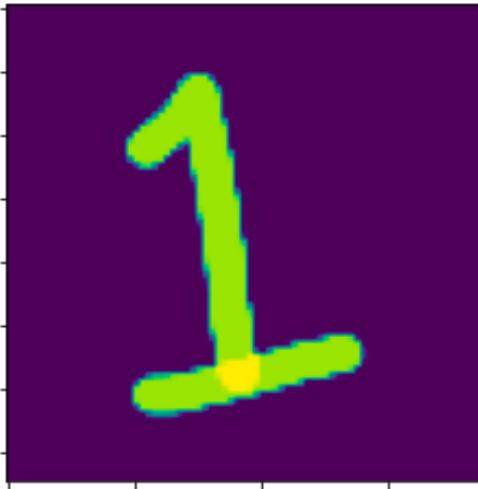
Later the image is cropped. Cropping function is defined in such a way that it first crops the image exactly to its boundaries. The row or column with first and last pixel of the digit is observed by taking the difference between each row/column and the row/column which is next to it. With this we can remove the extra plain background region. Later the image was getting with 50% of its width on right and left and with 20% of its height on top and bottom of the image.

Cropped image: (394,373,1)



From the above resized images, we can observe that the image resized to 75x75 was clearer than 25x25. This implies that the additional information stored in this 75x75 image may get us good performance of the model on testing images.

Final image of shape 1x75x75x1 sent to the model. (This has been done as this input image has a white background and our training data images have black color. But we need not do it for the images we get from front-end)



Building and training a Deep learning model:

A Convolutional network was built and tuning of hyper parameters has been done. Below is the finalized deep learning model used for this project.

ConvNet used for this project:

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3,3), activation='relu', input_shape=input_shape))
model.add(Conv2D(128, (3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.2))

model.add(Conv2D(64, kernel_size=(3,3), activation='relu', input_shape=input_shape))
model.add(Conv2D(32, (3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.2))

model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
```

ReduceLROnPlateau was used as callback monitoring validation loss with patience 5 and factor 0.85. And while compiling the model, Adam optimizer with categorical cross entropy loss has been used.

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 73, 73, 32)	320
conv2d_1 (Conv2D)	(None, 71, 71, 128)	36992
max_pooling2d (MaxPooling2D)	(None, 35, 35, 128)	0
dropout (Dropout)	(None, 35, 35, 128)	0
conv2d_2 (Conv2D)	(None, 33, 33, 64)	73792
conv2d_3 (Conv2D)	(None, 31, 31, 32)	18464
max_pooling2d_1 (MaxPooling2D)	(None, 15, 15, 32)	0
dropout_1 (Dropout)	(None, 15, 15, 32)	0
flatten (Flatten)	(None, 7200)	0
dense (Dense)	(None, 128)	921728
dropout_2 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 10)	1290
Total params: 1,052,586		
Trainable params: 1,052,586		
Non-trainable params: 0		

The above model was fit on training data of batch size of 128 and 32 epochs were used in total. The test data was used as validation set.

This model had performed well on training and validation set with 99.87% and 99.06% accuracies respectively.

```
loss: 0.0034 - accuracy: 0.9987 - val_loss: 0.0417 - val_accuracy: 0.9906
```

Saving the model:

Earlier we had saved entire model with the help of below command:

```
model.save('saved_model/my_model')
```

 where the input argument was the path.

Challenge 4: Though loading this saved model worked well on google colab, this was caused a multi-threading issue while deployed in our local machine.

Action: So, we had dropped the idea of saving the entire model and instead we saved the weights of the model in .h5 format.

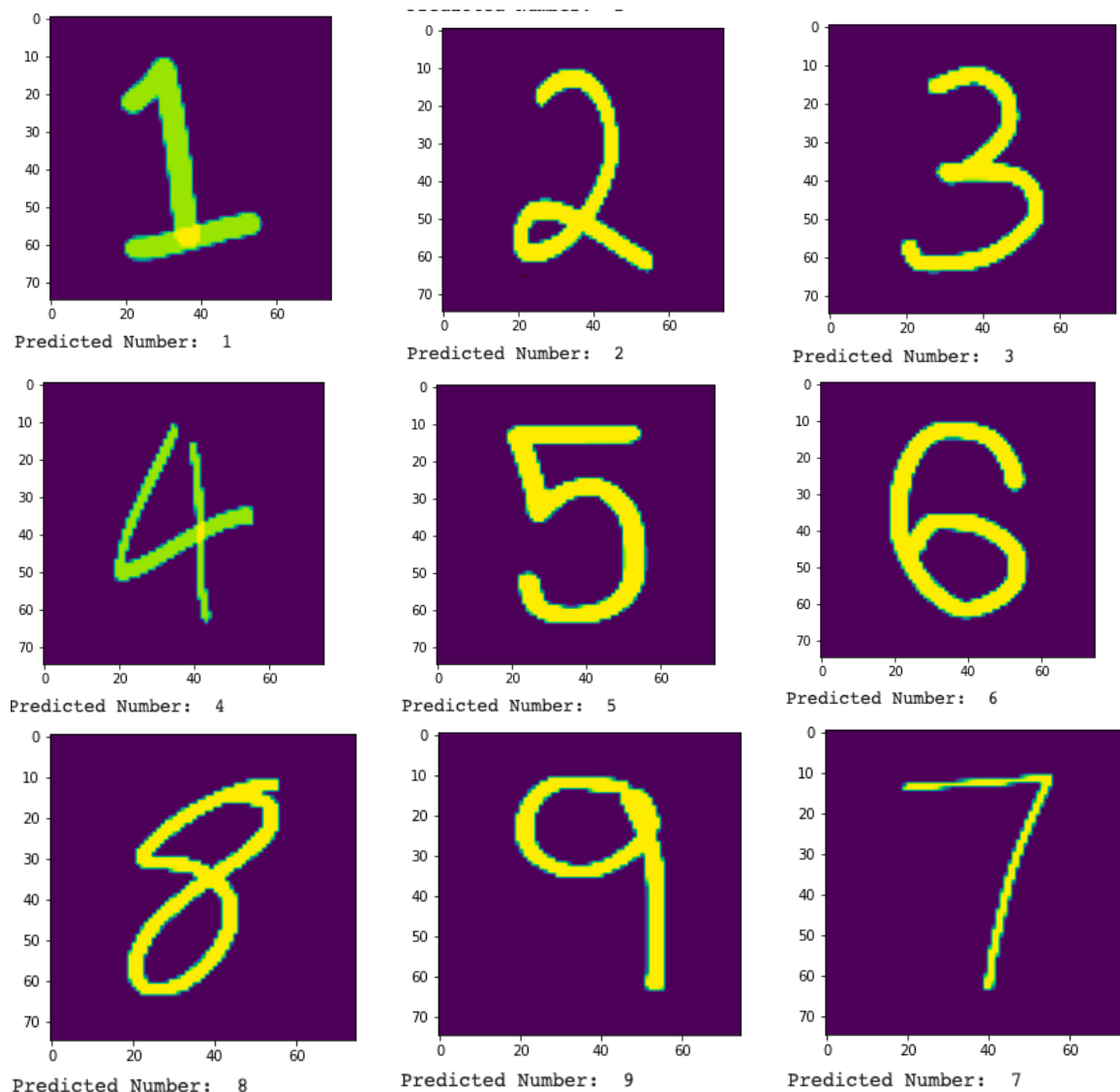
```
model.save_weights('saved_weights.h5')
```

And before loading the weights, we have to create the model with same architecture used for learning the saved weights.

```
new_model = create_model()
```

```
new_model.load_weights('saved_weights.h5')
```

Testing the deep learning model with hand written images:



Challenge 5: Even though the saved weights were successfully loaded. We could not perform prediction due to a multi-threading issue on our local system.

Error: OMP: Error #15: Initializing libiomp5.dylib, but found libiomp5.dylib already initialized.

Action: Below command has been used to resolve the above issue.

```
import os
os.environ['KMP_DUPLICATE_LIB_OK']='True'
```


Flask framework:

Flask framework has been used to integrate the deep learning model and python code with the front-end.

For this application, we have to get the image drawn by the user from the front-end and send the number predicted by the model to the front-end to display.

So, for this project, we have used only one function to post and get the data from and to front-end.

Challenge 6: The image which is received from the front-end was in base 64 format (here we were expecting URL or png or matrix)

Action: We have trimmed the base 64 string to take the encoded data and it was decoded with the help of base64.

Below is the code to convert the base64 data into a matrix:

```
imageData = request.form.get('image')
imageData = imageData.replace('data:image/png;base64,', '')
imageData = imageData.replace(' ', '+')
imageBytes = base64.b64decode(imageData)
img = Image.open(BytesIO(imageBytes))
```

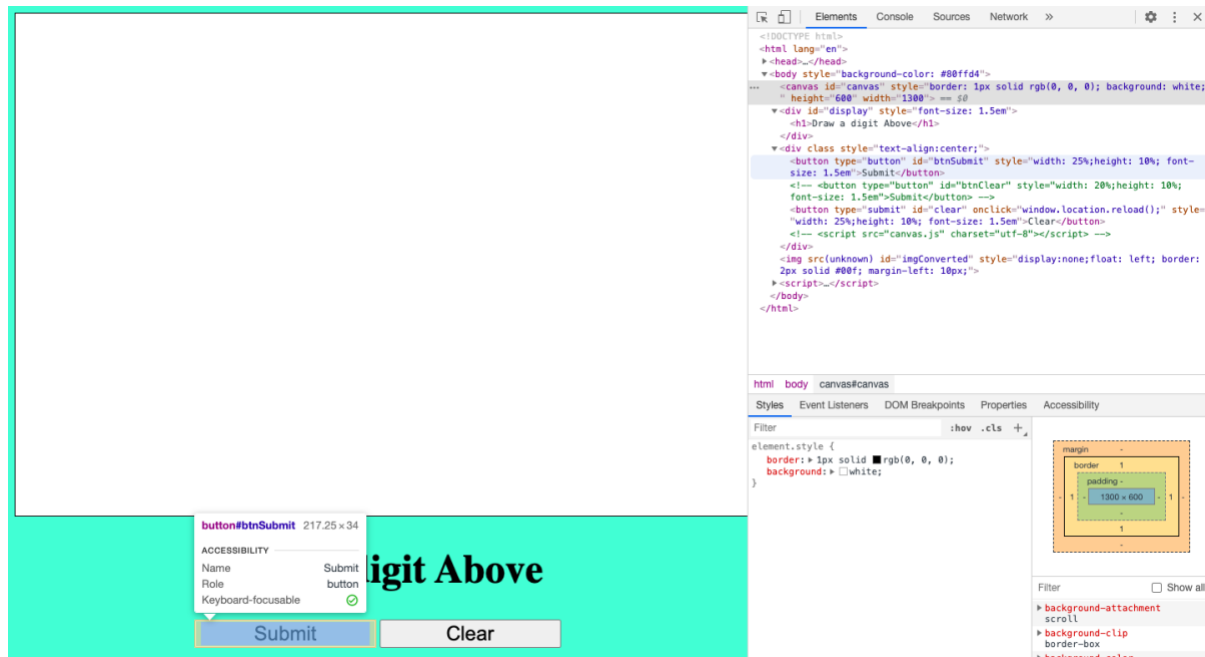
The decode matrix had 4 channels (RGBA: Red Green Blue Alpha where Alpha consists of pixel information regarding the background of image). For this project, color of the pen has been set to 'Red' and only first channel of the entire input image matrix was considered for rest of the operations.

2. Front-end:

HTML was used for the front end where we had defined background color, a canvas java script to enable a window for users to draw and two buttons (submit and clear). Styling has been used to assign the text and buttons in the center.

Canvas Java Script: As part of the script, we had defined the window dimensions and a context which lets user draw on the defined window.

This functionality to let user draw a line or a point was defined in the script part of the code.



Sending request to server:

JQuery, Ajax was used to send user input data to the server and retrieve the results from the server.

On clicking Submit button, we send the input image data on the window is sent to the server and the predicted number from the server is retrieved. This number is converted to words and displayed to the user.

On clicking Clear button, page gets refreshed, clearing the window and allowing the user to draw again.

3. Application Deployment:

- a) **Local Deployment:** HTML file was placed in a folder named templates. Templates folder and python file along with saved weights .h5 file are placed in a folder. From terminal (in mac), go to the above folder and execute the below command.

Command to deploy application locally: python server.py (if server is the python file name)

It deploys and gives us a local IP address. In most of the cases, it would be 'http://127.0.0.1:5000/'.

```
2020-08-14 21:21:19.410414: I tensorflow/compiler/xla/service/s
* Serving Flask app "server" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a pr
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

b) Deployment on AWS server:

For this project, we have initially tried to deploy on Heroku server but as the final slug size of this application was 770 MB which was more than the limit 500 MB which Heroku server can support, we couldn't deploy it there.

Later we deployed the on AWS server. In process of deploying the application on AWS server, we had created an account on AWS and initialized an EC2 instance and defined the security group setting to allow multiple ports from anywhere. And in this process, we will have to download a key to ensure the connection is safe.

Later from our local terminal, we will have to create a secured connection using the above downloaded key with the help of below command.

```
"ssh -i "sankhya.pem" ubuntu@ec2-3-15-233-56.us-east-2.compute.amazonaws.com"
```

Here ubuntu is the username, sankhya.pem is the key and 'ec2-3-15-233-56.us-east-2.compute.amazonaws.com' was the IP address given by AWS for this instance.

```

AC(base) Kaushiks-Air:herokudeploy kaushik$ ssh -i "sankhya.pem" ubuntu@ec2-3-15-233-56.us-east-2.compute.amazonaws.com
-----
      _ _|_ _|_ _
      | |< _/ _/ Deep Learning AMI (Ubuntu 18.04) Version 32.0
      |_|_|_|_|_|_|
-----

Welcome to Ubuntu 18.04.5 LTS (GNU/Linux 5.3.0-1032-aws x86_64v)

Please use one of the following commands to start the required environment with the framework of your choice:
for MXNet(+Keras2) with Python3 (CUDA 10.1 and Intel MKL-DNN) ----- source activate mxnet_p36
for MXNet(+Keras2) with Python2 (CUDA 10.1 and Intel MKL-DNN) ----- source activate mxnet_p27
for MXNet(+AWS Neuron) with Python3 ----- source activate aws_neuron_mxnet_p36
for TensorFlow(+Keras2) with Python3 (CUDA 10.0 and Intel MKL-DNN) ----- source activate tensorflow_p36
for TensorFlow(+Keras2) with Python2 (CUDA 10.0 and Intel MKL-DNN) ----- source activate tensorflow_p27
for TensorFlow(+AWS Neuron) with Python3 ----- source activate aws_neuron_tensorflow_p36
for TensorFlow 2(+Keras2) with Python3 (CUDA 10.1 and Intel MKL-DNN) ----- source activate tensorflow2_p36
for TensorFlow 2(+Keras2) with Python2 (CUDA 10.1 and Intel MKL-DNN) ----- source activate tensorflow2_p27
for TensorFlow 2.3 with Python3.7 (CUDA 10.2 and Intel MKL-DNN) ----- source activate tensorflow2_latest_p37
for PyTorch 1.4 with Python3 (CUDA 10.1 and Intel MKL) ----- source activate pytorch_p36
for PyTorch 1.4 with Python2 (CUDA 10.1 and Intel MKL) ----- source activate pytorch_p27

```

Later we had created a tensorflow environment and checked the version of tensorflow. If the version is not updated, uninstall and install latest version of the tensorflow.

Activate the created tensorflow environment and add the files and folders similar to the local folder.

```

Last login: Fri Aug 14 22:54:54 2020 from 98.5.144.224
[ubuntu@ip-172-31-17-0:~$ conda activate tf
(tf) ubuntu@ip-172-31-17-0:~$

```

```

(tf) ubuntu@ip-172-31-17-0:~$ ls
LICENSE  Nvidia_Cloud_EULA.pdf  README  anaconda3  app.py  examples
homebrew  requirements.txt  saved_weights.h5  src  templates  tools  tutorials

```

We have to execute the python file in this EC2 instance in the similar way to the local deployment by executing the below command.

```

(tf) ubuntu@ip-172-31-17-0:~$ python app.py
2020-08-15 01:58:30.178006: I tensorflow/core/platform/cpu_feature_guard.cc:143] Your
o use: SSE4.1 SSE4.2 AVX AVX2 FMA
2020-08-15 01:58:30.183004: I tensorflow/core/platform/profile_utils/cpu_utils.cc:102
2020-08-15 01:58:30.183220: I tensorflow/compiler/xla/service/service.cc:168] XLA ser
e that XLA will be used). Devices:
2020-08-15 01:58:30.183334: I tensorflow/compiler/xla/service/service.cc:176] Strea
2020-08-15 01:58:30.183514: I tensorflow/core/common_runtime/process_util.cc:147] Cre
op_parallelism_threads for best performance.
* Serving Flask app "app" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)

```

In order to get the link for the deployed application, we have to sign in to the AWS and go the action tab of our EC2 instance.

We have to add ':8080/' to the existing Public DNS.

With the help of the below link we can connect to our instance:

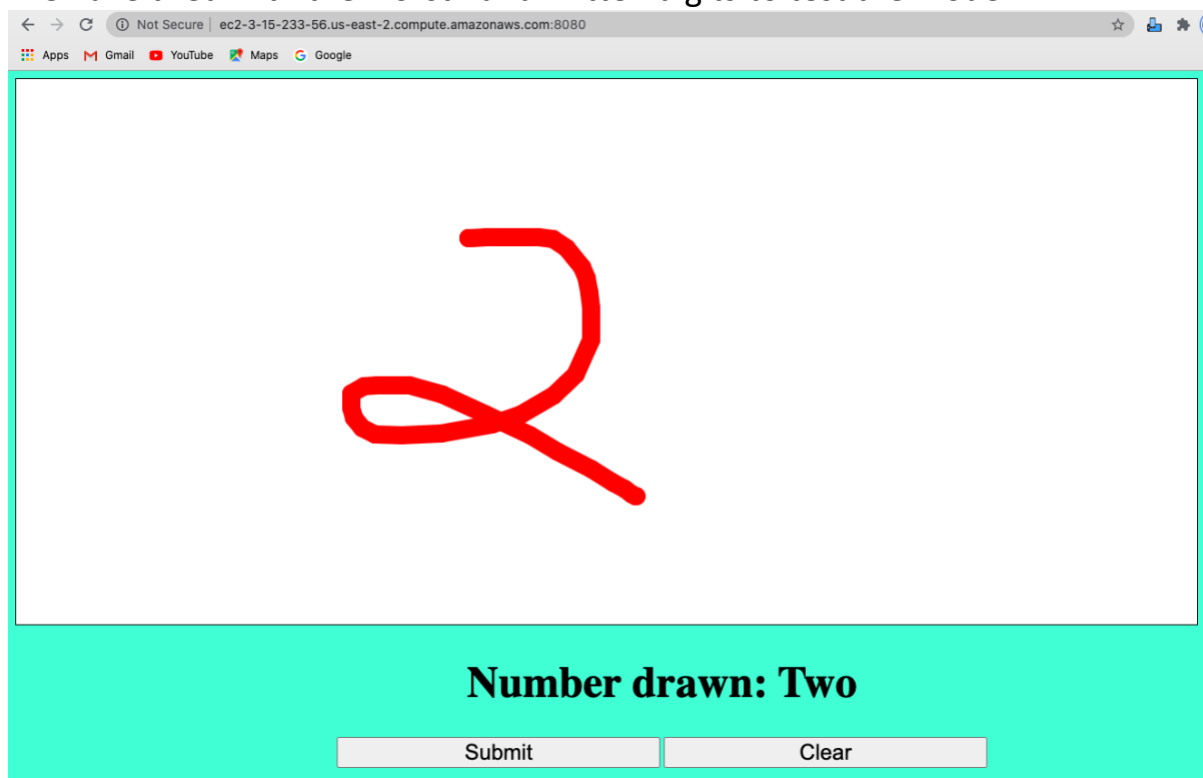
<http://ec2-3-15-233-56.us-east-2.compute.amazonaws.com:8080/>

Conclusion & Results:

Application has been deployed successfully and we have got some good results for most of the digits which are written in very ugly way.

Below are the few examples:

We have tried with the worst handwritten digits to test the model.





Number drawn: Seven

Submit

Clear



Number drawn: Nine

Submit

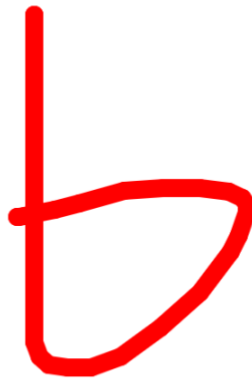
Clear

A hand-drawn number '3' in red ink, consisting of two curved strokes.

Number drawn: Three

Submit

Clear

A hand-drawn number '6' in red ink, consisting of a vertical stroke and a curved bottom.

Number drawn: Six

Submit

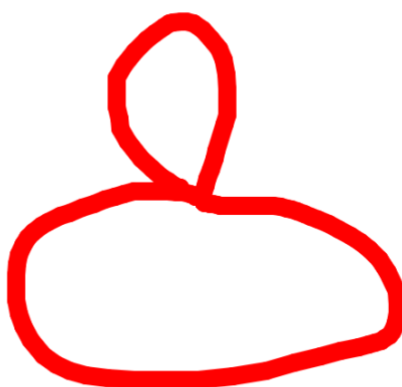
Clear



Number drawn: Five

Submit

Clear



Number drawn: Eight

Submit

Clear



Number drawn: Five

Submit

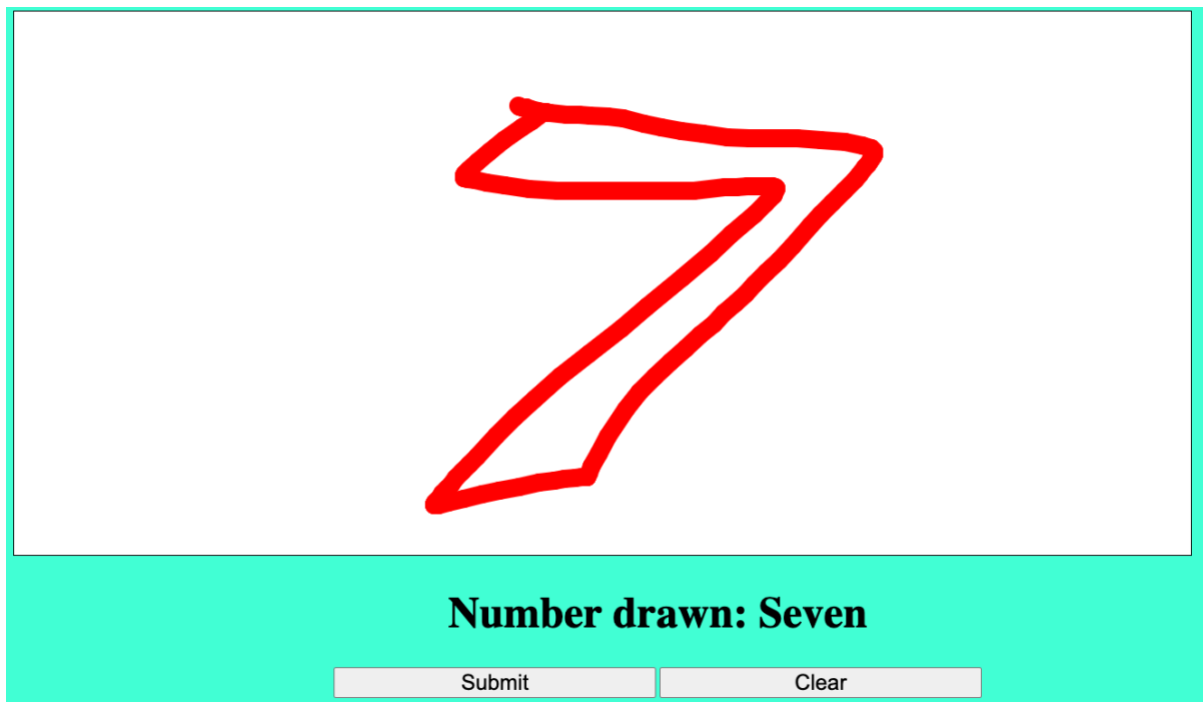
Clear



Number drawn: Four

Submit

Clear



As you have seen above, our model was doing pretty well.

Possible improvements:

1. A major improvement would be to include padding to the cropped image. Doing this the model can be made more efficient even for the digits draw in the corner.
2. Including a slider to decide on the size of the pen used and a check box or dropdown to pick the color of our choice.

Reference:

<https://keras.io/api/datasets/mnist/>

<https://www.youtube.com/watch?v=3GqUM4mEYKA>

<https://www.youtube.com/watch?v=YoVJWZrS2WU>

https://www.youtube.com/watch?v=Ozc5Yu_IcaI

<https://stackoverflow.com/questions/55549164/how-to-connect-backend-python-flask-with-frontend-html-css-javascript>

<https://github.com/nathanmargaglio/Deployable-Model>