**Capstone Project Adithya Kumar**

Machine Learning Engineering Nanodegree Feb 27, 2019

***Definition***

**Project Overview**

“Suppose you wanted to digitize a magazine article or a printed contract. You could spend hours retyping and then correcting misprints. Or you could convert all the required materials into digital format in several minutes using a scanner (or a digital camera) and Optical Character Recognition software.”

If we have a paper document, a scanner or a mobile phone camera would help us digitize the image. But if we need to get the contents of the document, that’s where OCR is very helpful. There are many licensed/paid softwares available, which would do the OCR. But the licensing costs for these softwares are high. The aim of this project is to build an in-house OCR engine, using Deep Learning technologies, which would extract text from images.

**Problem Statement**

The goal is to create a deep learning model which would extract the text from images. Currently the model is trained on small machine generated images and images containing only one word. The steps involved in this project are

* Download the dataset from the publicly available dataset

<http://www.robots.ox.ac.uk/~vgg/data/text/> . This is a synthetically generated dataset which is generally used for training deep learning models for real world images.

* Train a deep learning model to determine the text contained in an image
* For predicting the text on a new image, convert the images to the features that would be accepted the model and let the model predict the text

The project is an extended implementation of this GitHub repository where they have solved the same problem using TensorFlow. (<https://github.com/da03/Attention-OCR>)

**Metrics**

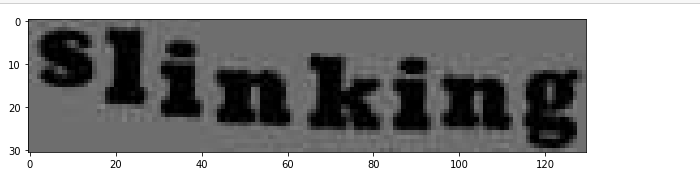
Accuracy is a common metric to evaluate sequence to sequence classifiers. If the predicted output is an exact match with the actual output, it is considered as a PASS else it is considered as a FAIL

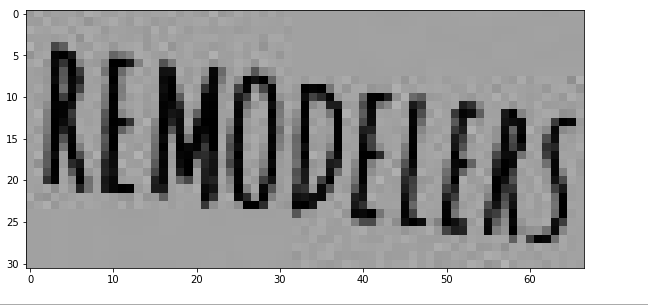
Accuracy = Total number of PASS rows / dataset size

**Analysis**

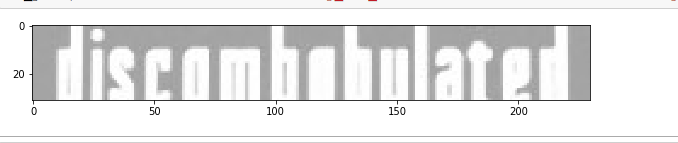
**Data Exploration and Exploratory Visualization**

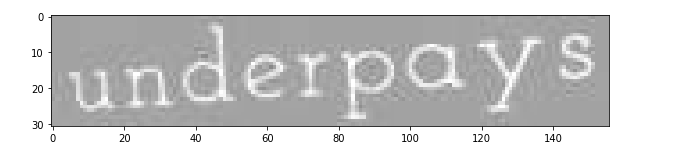
We can parse through the training dataset of images, to see how the images look like. This would give an idea of the complexity of the dataset.

****

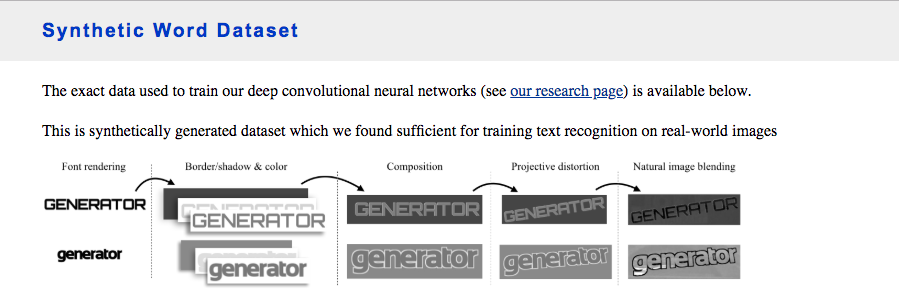
****







As we can see, the training data consists of images such as above. These are all images consisting of a single word generated on different fonts with varied borders, shadows, color compositions. This is the process by which the images are generated according to the dataset homepage.

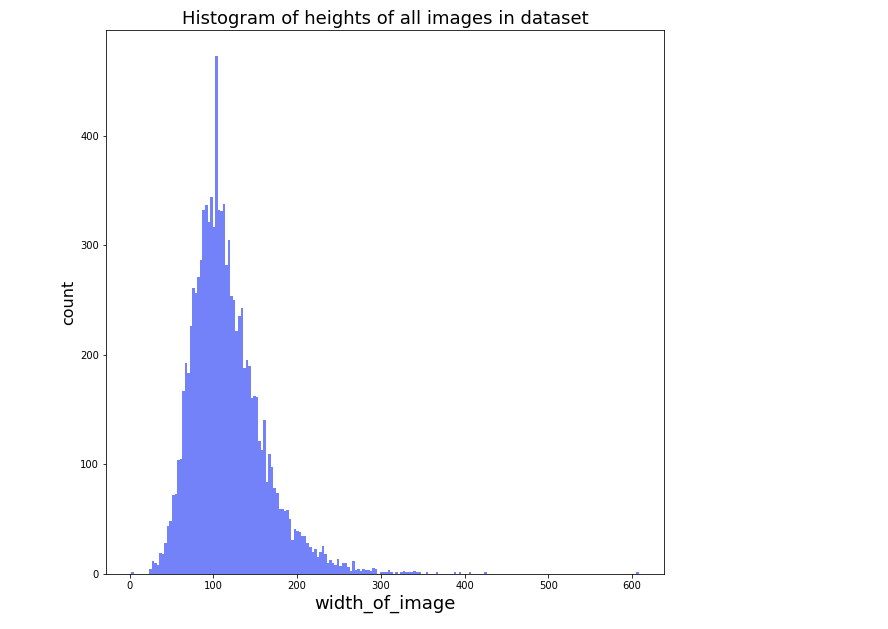


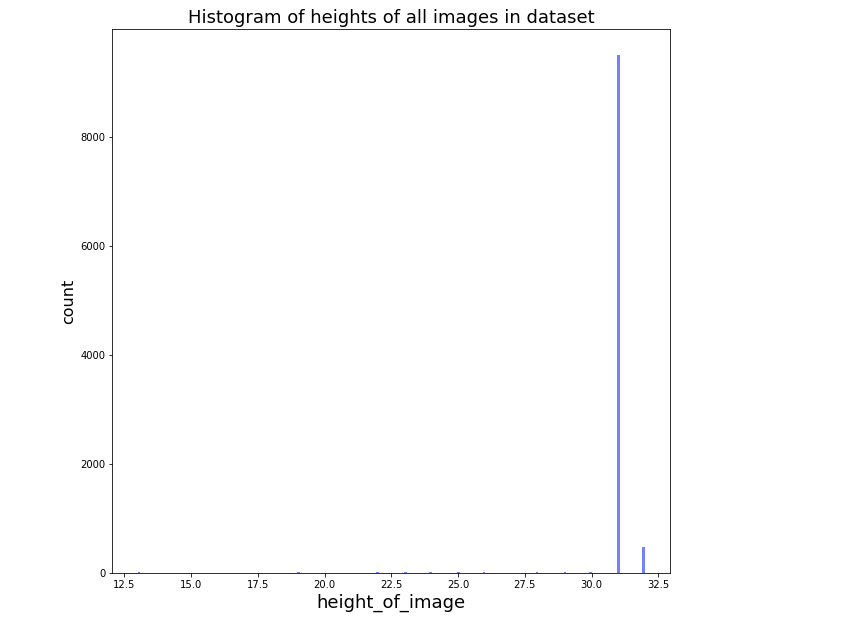
The initial scope of this model is to learn from small images. In subsequent versions of the model, the following methodology would be implemented to handle large images

* Use OpenCV python package to break down into smaller pieces by identifying the contours in the image
* For each of the contours identified, break down the images into smaller boxes, predict the text for each of the smaller box
* Piece them all together into a single string, which would be the predicted output

On analyzing the images, I find that all images have the same height of 31 pixels, whereas the width can vary depending on the length of the word.

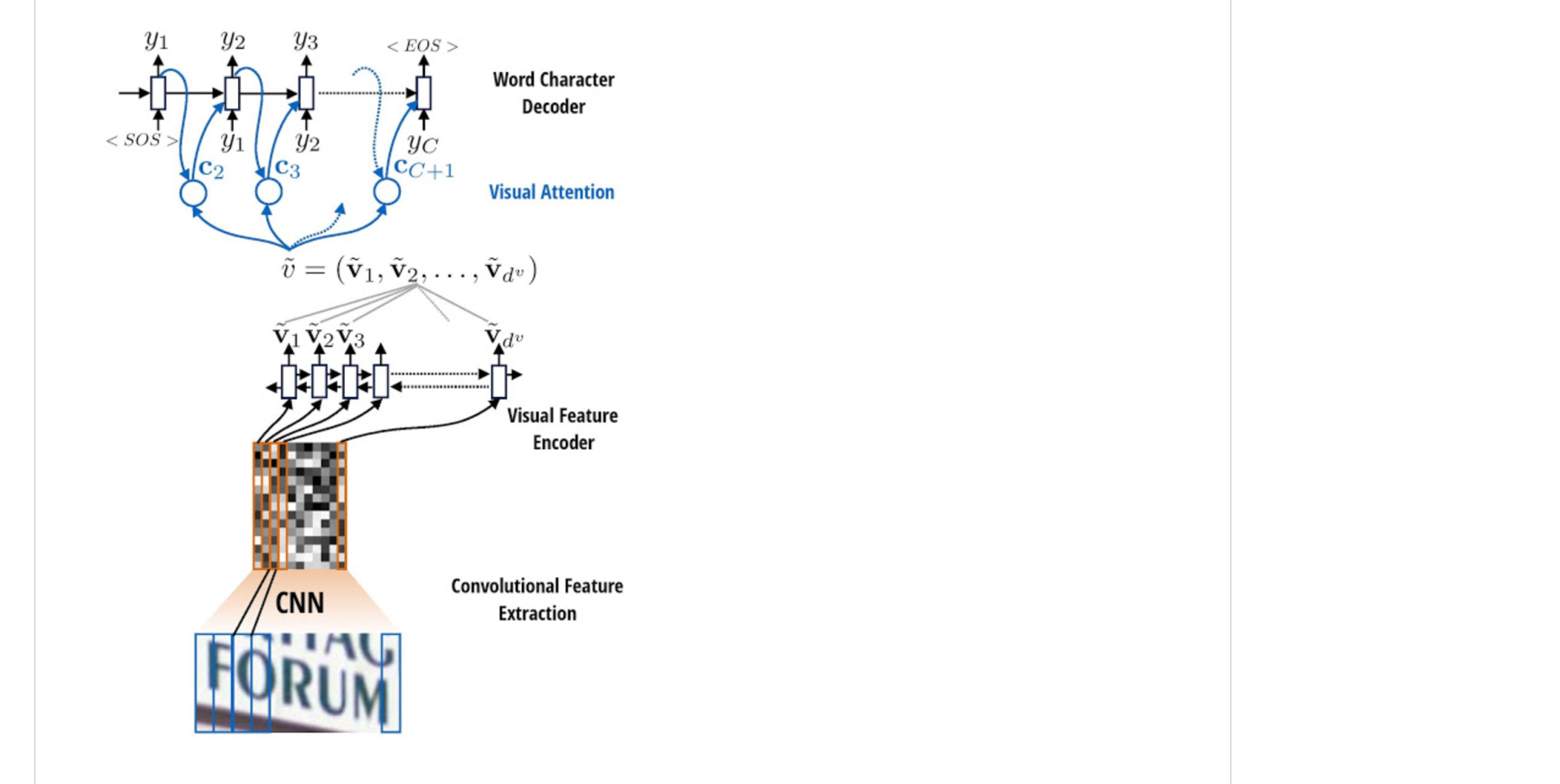
The reason for plotting these graphs was to check how varied the images were in terms of sizing and whether they required resizing. From my results, I saw that there wasn’t huge variation. The first plot shows a histogram of the height of all images in my dataset and the second plot shows a histogram of the width of all images in my dataset.





**Algorithms and Techniques**

The model is designed in such a way that it uses multiple convolutional layers to extract features from images and then a bidirectional GRU encoder with a visual attention layer as a decoder for producing the final outputs



The above diagram is a pictorial representation of the model architecture. The following steps would be implemented for training the model. More details about the implementation would be mentioned in the later sections

* Download the dataset into the AWS GPU
* Create a vocabulary file, which would be the characters used in the training data. The input label string would be encoded as characters from the vocabulary file
* For every image in the training dataset
  + Apply random augmentation on random images
  + Pre-process the image
  + Split image into slices and store the different slices as separate dimensions of a NumPy array
  + Encode the label text based on keys of the vocabulary file and if there are characters which are not present in the vocabulary file, encode with ‘UNK’. Every string is terminated with a ‘STOP’ character and padded with a ‘PAD’ character
* The CNN’s would extract visual features from the images and those features would be encoded into a single vector by a bi-directional LSTM layer with visual attention encoder. The results of the encoder would be passed to a visual attention encoder which would return

**Benchmark**

Unfortunately, for this project, I didn’t have a benchmark to test my results upon.

***Methodology***

**Data Pre-processing**

***Image – pre-processing***

The pre-processing of image consists of the following steps

* Image is read from the image path
* The height and width of the image are extracted
* For every image, the scale factor is calculated and if it is less than 1, scale factor is set to 1
* Image is resized to new width and height
  + New width = scale factor \* 40
  + New height = max height (which is set to 40 pixels)
* The resized image is converted to RGB channels array and using a function it is converted to a grey scale image

The max height of 40 is chosen based on the exploratory analysis of the image which revealed the heights of the images were either 31 or 32 pixels.

**Implementation**

***Conversion of image to feature arrays to be used by the model***

* There is a python file(~/src/make\_vocab.py) which would help in identifying the most frequent characters in the training dataset labels and save it as a JSON file. The JSON file rcptAlphabet.json was generated like that and contains the list of all the characters that the model can accept and return

[" ", "!", "\"", "#", "$", "%", "&", "'", "(", ")", "\*", "+", ",", "-", ".", "/", "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", ":", ";", "<", "=", ">", "?", "@", "A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z", "[", "\\", "]", "^", "\_", "`", "a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l", "m", "n", "o", "p", "q", "r", "s", "t", "u", "v", "w", "x", "y", "z", "{", "}", "~"]

* A data generator is created (~/src/data\_generator.py) : gen\_data(), which would accept the following arguments as input and return the feature array and the label array which can be passed to the model for training
  + Input\_folder : The folder path containing the images
  + Annotation\_file : The annotation file which would contain the path of the image and the image\_id. In this dataset the path of the image contains the text contained in the image too
    - For eg: this is one of the inputs in the annotation file ./3000/6/467\_strenuously\_75159.jpg
    - The above string is parsed and the word *strenuously* is stored as the label and the image\_path is saved as the concatenated string of Input\_folder + Image\_path
    - The above operations are being performed by the get\_image\_path\_text function
  + Rcpt\_map : Receipt map is a dictionary containing the list of acceptable characters read from the rcptAlphabet.json along with numeric keys. Three special characters are added to this list (‘PAD’,’STOP’,’UNK’)
  + Batch\_size : The batch\_size defines the number of samples that will be propagated through the network and the data generator returns features and labels based on the batch size. If the batch size is 32, the data generator would return 32 feature set and label sets from the input dataset.
  + Augment\_ratio: This is the percentage of images to be augmented. This sets a threshold on the number of images to be augmented.
    - Augmentation: The augmentation operations are defined in the file (~/src/augment.py). Currently three types of augmentation are supported by the model (Random rotation, random noise, and horizontal flip). These augmentations are introduced ,so that the models learns better spatial patterns from the images and would perform better on real world images which would be noisy
    - augment\_flag would randomly assign a boolean value and if it is TRUE and if the augment\_image\_count is lesser than the augment\_images\_total (which would be the fraction of images to be augmented as mentioned by the argument)
    - continuous = True : This is a flag to pass to the data generator to either keep generating data or just generate one row of data. During training, this would be set to True
* The image is split into slices and saved into a numpy array. The argument max\_slice\_num mentions the number of slices the image needs to be split. In this case that is set to 24. The slice width is set to 20 pixels. The get\_slices() function in (~/src/data\_generator.py) splits the image into slices and saves into a numpy array
* A larger numpy array containing the number of rows in the batch size is created. Every row of the input dataset is parsed, and every image is read, pre-processed, augmented, split into slices, saved into a numpy array and finally saved into the index of the larger numpy array. This would be the feature set which is used by the model, containing the number of rows of the batch size and every image split into slices.

***Encode labels to a format which can be used by the model***

* The encode\_text() function gets the text and the rcpt\_map as inputs. It created an output boolean array of the maximum allowable length (in this case 40 characters), The label string is iterated and for every character in the string, the receipt map is iterated and the corresponding key is returned. The index of the key in the output array is set to TRUE.
* If we encounter a character, not present in the receipt vocabulary the index of the UNK character is set to TRUE
* If end of string is reached, the index of the STOP character is set to TRUE in the output array. This indicates the end of the string to the model.
* If the string is lesser than the maximum allowable character (40 in this case), the rest of the output string is set to TRUE for the character PAD

The same process is repeated for every row and a batch of features and label arrays are returned by the data generator

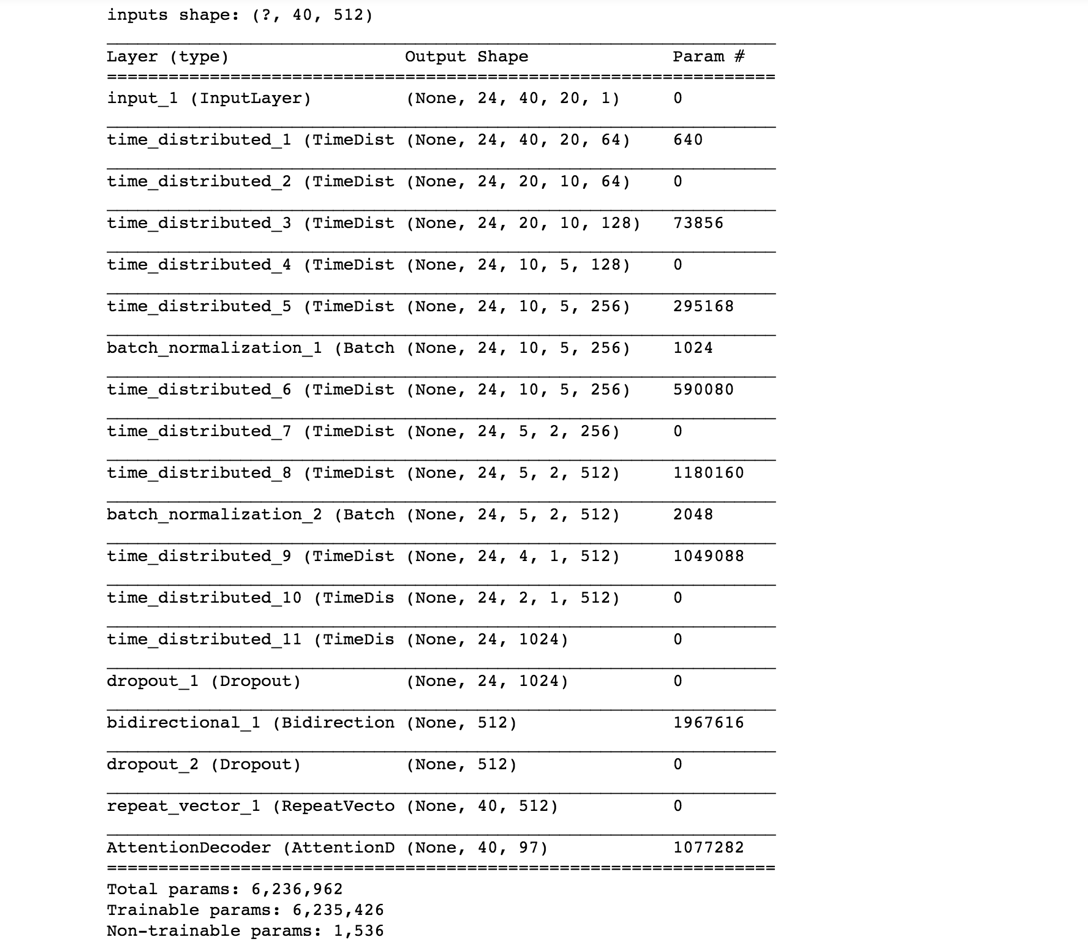
***Training Steps***

The training script(train\_model.py) takes as arguments the following

* Training folder name – This would be the folder containing the training images
* Train file – This would be the input file, which contains the image name(to be searched for in the training image folder) and the corresponding text in the image
* Validation folder name - This would be the folder containing the validation images
* Val file – This would be the input file, which contains the image name(to be searched for in the validation image folder) and the corresponding text in the image. This would be for the test set
* Vocab\_file – This would the rcptAlphabet.json file discussion above which contains the list of allowable characters the model is supposed to train and return

The model is trained in batches of 32 images for 2500 steps and for 30 epochs.

***Model architecture :***



**Refinement**

* The initial model returned an accuracy of 65%.
* The image augmentation piece was added later and that improved the accuracy to 70 %

**Results**

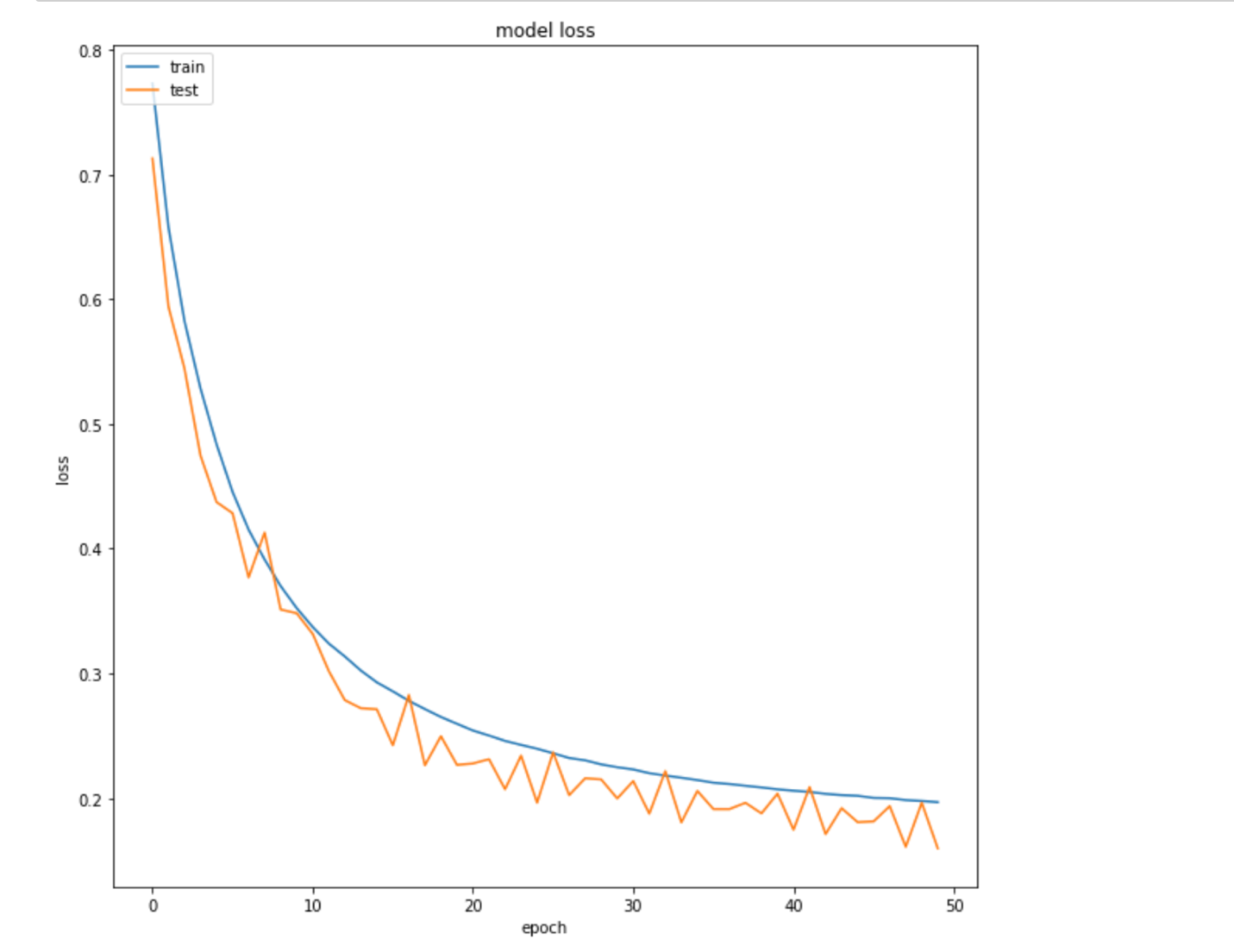
**Model Evaluation and Validation**

The model was trained on an AWS GPU.

The results were tested on the test set. I have created this test\_model.py script which would iterate through the test set, extract features, call the model and return the text output predicted by the model.

The metric I am considering the evaluate my model is EXACT MATCH. I am checking the output of my model along with the true output, only if I get an exact match (if all characters in the predicted output match with the actual output), I consider that to be correct. The number of correct images are validated with the total number of images in the test set.

The result was 70.16 % accuracy. Also shown below is the loss plot for both the training and validation sets after training for every epoch. We can see that the loss is decreasing for every epoch



**Justification**

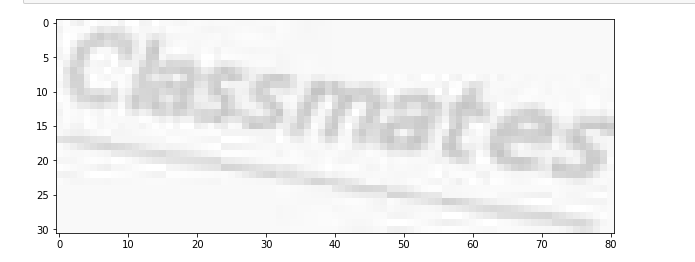
Since I didn’t have any prior benchmark for this model, I am leaving this section blank

**Conclusion**

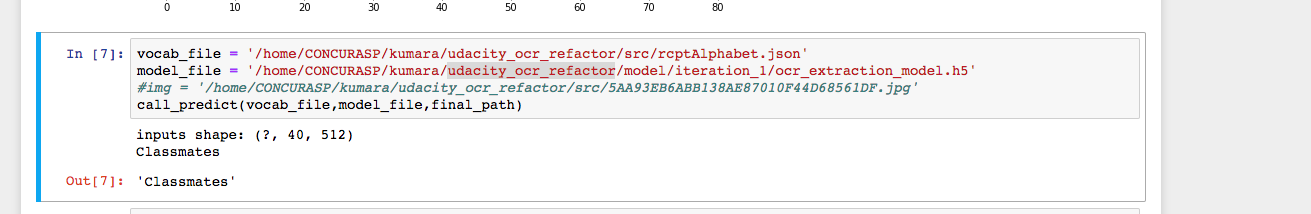
**Free-Form Visualization**

Theaim of theproject is to extract text from an image. Since the model was trained on images containing a single word, lets test it out to see if it works.

Image to test:

****

If the call\_predict function is called with the image file, the model returns the right result as classmates



**Reflection**

The aim of the project is to build a deep learning sequence to sequence model which would return the text contained in the image. The training data for this project was obtained from a labelled dataset by Oxford University, London. The dataset contained synthetically generated images, with varied background, color, distortion, borders and shadows. The labels of the dataset would be the text used to generate the image. The model was trained on close to 1 million images like these.

The model architecture was based out of a similar implementation using TensorFlow. One of the USP’s of this project is to use a bi-directional LSTM encoder with a visual attention decoder to identify the text in the image. This was a great learning experience for me in training large datasets for sequence to sequence models and also the use of attention layers in Keras. PFB couple of challenges I faced during the course of the project

* Creating the data generator was hard and it took lot of refinement to get it to the stage where it currently is
* The model architecture along with the hyper-parameters too was changed iteratively to improve the performance. That was challenging as well.

**Improvement**

* The model is currently trained on small images. Use this model to extract text from larger images by breaking them into smaller pieces and using Python’s OpenCV package
  + Use OpenCV python package to break down into smaller pieces by identifying the contours in the image
  + For each of the contours identified, break down the images into smaller boxes, predict the text for each of the smaller box
  + Piece them all together into a single string, which would be the predicted output
* Add data for other languages

**References**

* <https://www.abbyy.com/en-us/finereader/what-is-ocr/>
* <http://www.robots.ox.ac.uk/~vgg/data/text/#sec-synth>
* <https://github.com/da03/Attention-OCR>