**Project Overview:**

The aim of the project is to extract text from image by using Deep Learning models. The important application of OCRing images is, we can learn more about the contents/words in the image by extracting the text inside that and also becomes easier to search.

**Statement:**

There are lot of instances where we might need to understand the contents of an image. There are many specific use cases for this problem. I chose this problem, because this is very applicable for my current work. I work in a software company called SAP Concur Technologies. Our company has products/tools to do expense reporting for companies, without having to do submit paper receipts. So, we work a lot with receipt data and the text in the receipt.

One of our major problem areas has been OCRing images. Currently we use a third-party licensed software to extract the text from receipts. We need to process millions of images every day and we are paying huge money for the OCR company for licensing costs.

So, I was planning to come up with an in-house OCR extractor to reduce the dependency on a third-party API and also save finances for my company. This capstone project is a starting point for building that.

**Reference:**

<https://github.com/da03/Attention-OCR>

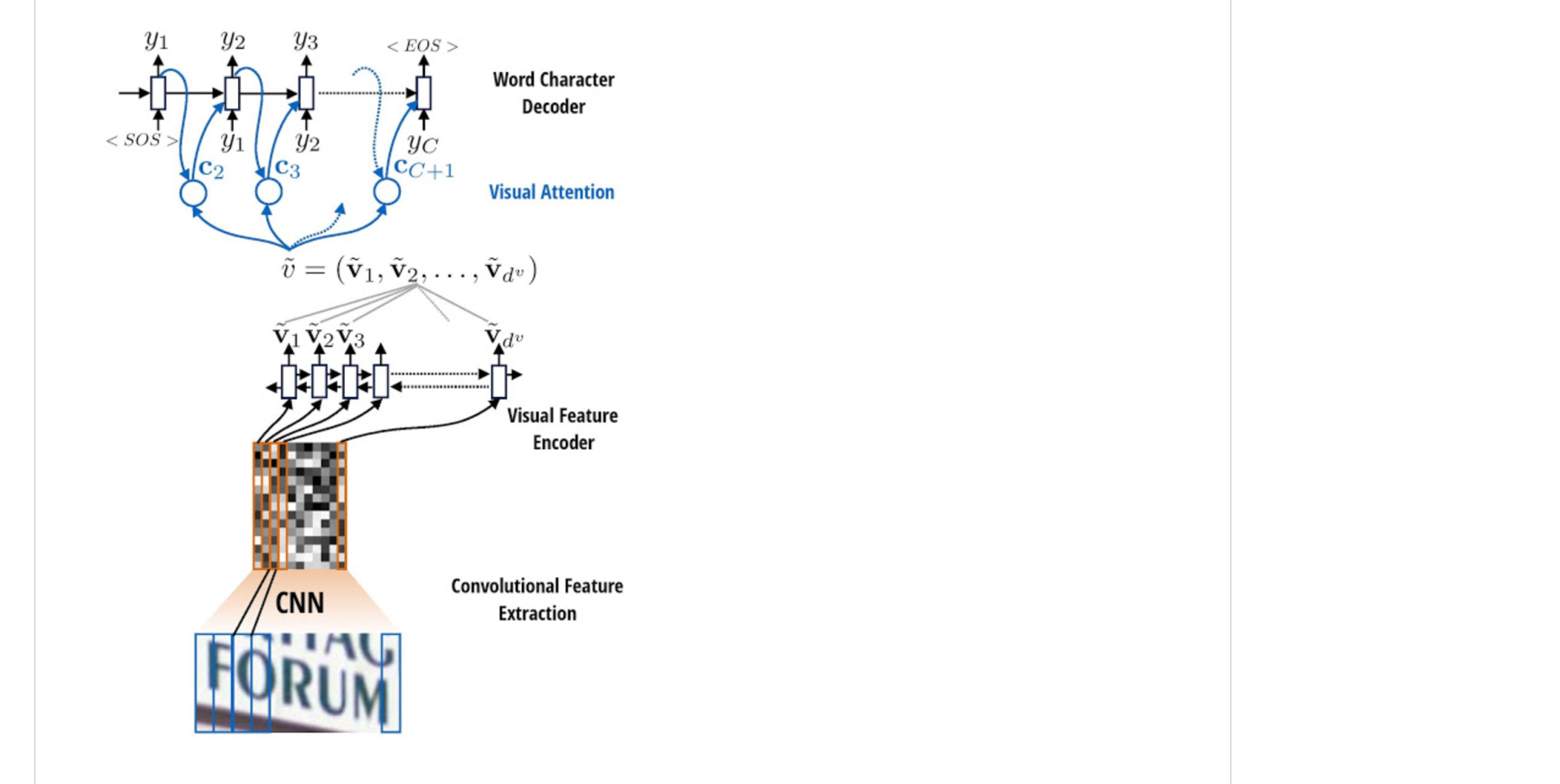
The codebase was solving a similar problem in TensorFlow. My code tries to replicate this in Keras (which is a wrapper around TensorFlow).

**Codebase:**

<https://github.com/adithya1111/ocr_keras_attention/blob/master/README.md>

**Model Overview:**

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



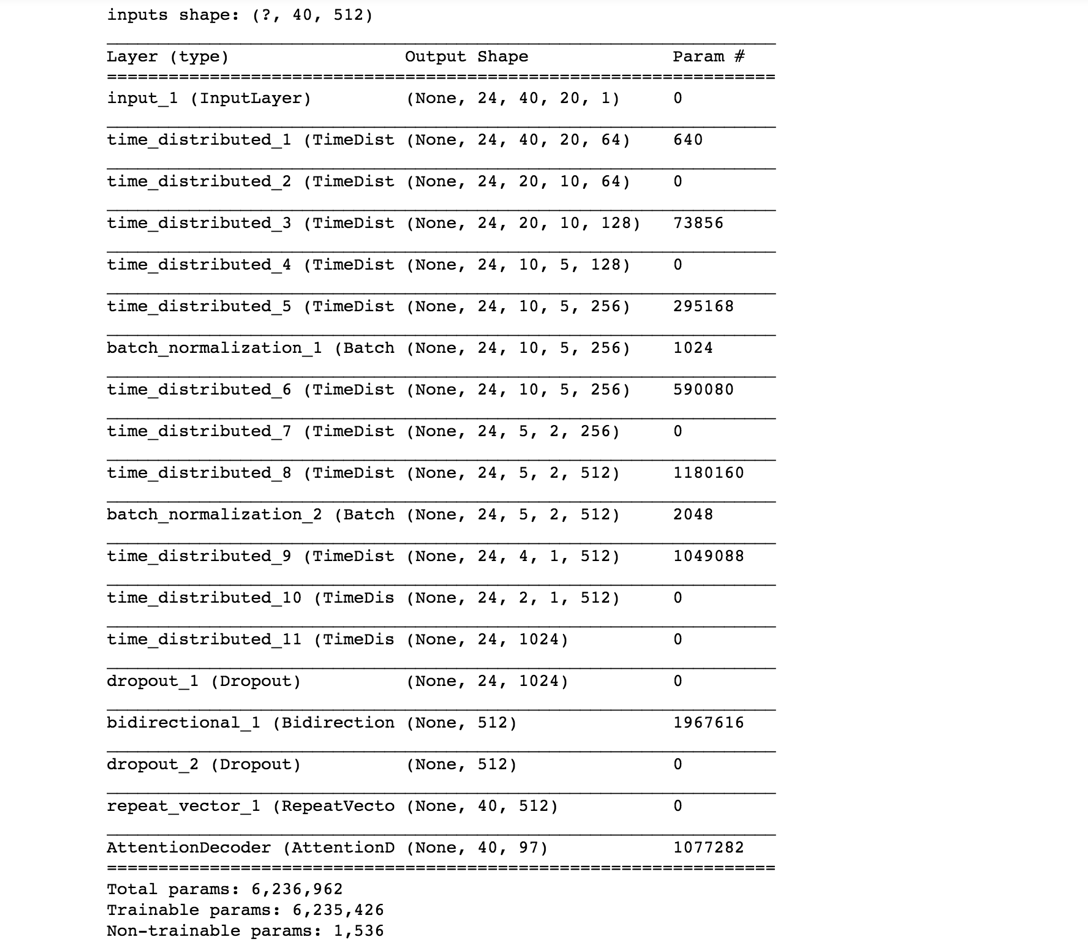
The current model is trained on extracting text from images containing one word

**Training Data:**

For training, I used images and text from this a dataset developed by University of Oxford. This is a synthetically generated dataset of over 1 million images along with the text. These images consist of only one word per image. The data is rendered for different fonts, distortions, borders, shadow and colors.

**Model Explained:**

The model was trained using Keras. PFB the model summary



**Model Architechture in code :**

x = TimeDistributed(Conv2D(64, (3, 3), padding='same', activation='relu'))(inp)

x = TimeDistributed(MaxPooling2D(pool\_size=(2, 2)))(x)

x = TimeDistributed(Conv2D(128, (3, 3), padding='same', activation='relu'))(x)

x = TimeDistributed(MaxPooling2D(pool\_size=(2, 2)))(x)

x = TimeDistributed(Conv2D(256, (3, 3), padding='same', activation='relu'))(x)

x = BatchNormalization(axis=4)(x)

x = TimeDistributed(Conv2D(256, (3, 3), padding='same', activation='relu'))(x)

x = TimeDistributed(MaxPooling2D(pool\_size=(2, 2)))(x)

x = TimeDistributed(Conv2D(512, (3, 3), padding='same', activation='relu'))(x)

x = BatchNormalization(axis=4)(x)

x = TimeDistributed(Conv2D(512, (2, 2), padding='valid', activation='relu'))(x)

x = TimeDistributed(MaxPooling2D(pool\_size=(2, 1)))(x)

x = TimeDistributed(Flatten())(x)

x = Dropout(DROP\_OUT)(x)

x = Bidirectional(GRU(attn\_num\_hidden, init='glorot\_uniform', inner\_init='orthogonal', return\_sequences=False,

kernel\_regularizer=l2(W\_REG), bias\_regularizer=l2(l=B\_REG)),

merge\_mode='concat')(x)

x = Dropout(DROP\_OUT)(x)

x = RepeatVector(y\_max\_len)(x)

x = AttentionDecoder(attn\_num\_hidden, char\_num, activation='softmax', kernel\_regularizer=l2(W\_REG),

bias\_regularizer=l2(l=B\_REG))(x)

As mentioned above the model uses couple of convolutional layers and pooling layers , followed by a sequence to sequence model consisting of a bidirectional GRU followed by an Attention layer for decoding.

**Receipt Vocabulary:**

The code leverages the vocabulary from this file rcptAlphabet.json and currently the vocabulary is restricted to these characters

[" ", "!", "\"", "#", "$", "%", "&", "'", "(", ")", "\*", "+", ",", "-", ".", "/", "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", "{", "}", "~"]

Also, there is code to generate the most frequently used characters in the training data as receipt vocabulary (make\_vocab.py), which can be leveraged.

**Training Data Generator:**

There is a data generator written to convert the images into features and labels. This generator accepts a text file containing the image path and the corresponding text inside the image as arguments. This text file is read in chunks. For each chunk, and for each batch (since generally neural networks are trained in batches before updating the model parameters)

The image is read as slices (of width: 20 pixels and height: 40 pixels) and the maximum number of slices is restricted to 40. All these are variable arguments and they can be changed as and when required. Every image is pre-processed, split into the required number of slices and stored in the training array.

Every image is pre-processed by following these steps

* Random augmentation – The images are augmented randomly (to introduce some noise in the training data, such as rotation, horizontal flip or adding gaussian noise) - (Please refer augment.py)
* The image is re-scaled to maximum height (currently 40 pixels)
* The RGB image is converted to a grey scale

The text for every image needs to be encoded, so that the model can use them. There is a receipt map variable, which reads the rcptVocabulary.json and converts to a dictionary. In addition to the characters in the receipt vocabulary, 3 characters are added

* UNK – If the receipt encounters a character not present in the receipt vocabulary
* STOP – the end character for every text. Every text is terminated by a STOP character
* PAD – if the actual string length is less than the maximum length of the output
* Create a numpy boolean array(FALSE) of the maximum length of allowable output
* Iterate through the text and for every character in the string
  + If the character is available in receipt vocabulary , mark that index as True
  + If the character is not available in the receipt vocabulary, mark the index of ‘UNK’ as TRUE
  + After iterating through every character, if there is still space left, mark the index of ‘PAD’ as TRUE. This is needed because every text in the training data is encoded to the same length. Some strings may be shorter, and some could be longer. We just make sure that maximum allowable length is larger.
  + If the end of string is reached, the string needs to be terminated with a STOP character. This is required because the model needs to know the end of the string.
* Thus, the text is encoded as a TRUE/FALSE Boolean array of the characters in the receipt vocabulary + the 3 special characters(‘UNK’,’PAD’,’STOP’)

**Training Steps**

The training script 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.

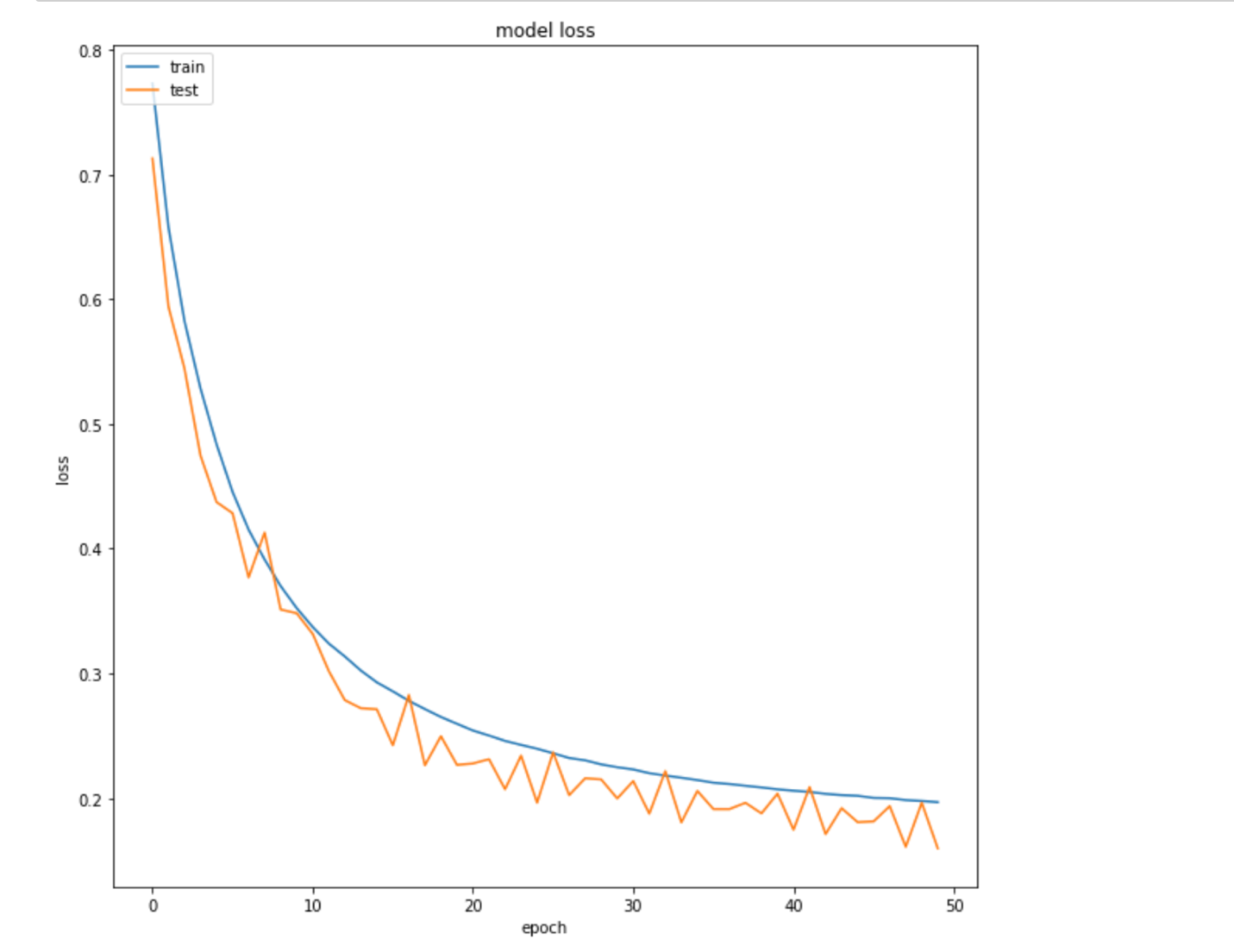
**Results**

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 acutal 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



**Conclusion and Future improvements:**

* To improve accuracy of current model
* To try out with different datasets like handwritten images
* **To do OCR for whole receipts:** The current model is designed to handle only images containing a single word. Next steps would be to to OCR larger images. The plan is to break the images into smaller pieces using OpenCV contours. Then for each of the smaller parts, run the model and piece the results back together
* To handle non-English receipts