JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY

Image Captioning With Deep Learning

Submitted by:
Vanshika Mittal : 16103012
Shubham Mishra : 16103220
Abhishek Verma : 16103237
Aman Goyal : 16103279

Submitted to:
Dr. Anuja Arora

Abstract:

The problem of image caption generation involves outputting a readable and concise description of the contents of a photograph. Image captioning is a challenging artificial intelligence problem as it requires both techniques from computer vision to interpret the contents of the photograph and techniques from natural language processing to generate the textual description.

We trained our 4 models InceptionResnet with Fasttext, InceptionResnet without Fasttext, VGG 16 with Fasttext, and VGG 16 on dataset of flicker 8k. Then the generated caption is compared with references for each model using BLUE and ROUGE metrics.

Clear division of project work among group members:

All of the work in project is equally done by each group member and coordination was done using google colaboratory and gmail.

Problem Statement / Research Objective:

Caption generation is a challenging artificial intelligence problem where a textual description must be generated for a given photograph.It requires both methods from computer vision to understand the content of the image and a language model from the field of natural language processing to turn the understanding of the image into words in the right order. Recently, deep learning methods have achieved state-of-the-art results on examples of this problem.
Deep learning methods have demonstrated state-of-the-art results on caption generation problems. What is most impressive about these methods is a single end-to-end model can be defined to predict a caption, given a photo, instead of requiring sophisticated data preparation or a pipeline of specifically designed models.

Research Paper studied:

Title: Show and Tell: A Neural Image Caption Generator
Authors: Oriol Vinyals, Alexander Toshev, Samy Bengio, Dumitru Erhan
Datasets: For evaluation we use a number of datasets which consist of images and sentences in English describing these images.
The statistics of the datasets are as follows:

Dataset name	size		
	train	valid.	test
Pascal VOC 2008 [6]	-	-	1000
Flickr8k [26]	6000	1000	1000
Flickr30k [33]	28000	1000	1000
MSCOCO [20]	82783	40504	40775
SBU [24]	1M	-	-

With the exception of SBU, each image has been annotated by labellers with 5 sentences that are relatively visual and unbiased. SBU consists of descriptions given by image owners when they uploaded them to Flickr. As such they are not guaranteed to be visual or unbiased and thus this dataset has more noise.
The Pascal dataset is customary used for testing only after a system has been trained on different data such as any of the other four dataset. In the case of SBU, we hold out 1000 images for testing and train on the rest. Similarly, we reserve 4K random images from the MSCOCO validation set as test, called COCO-4k, and use it to report results in the following section.
Methodology: NIC (Neural Image Captioner) is based on a convolution neural network that encodes an image into a compact representation, followed by a recurrent neural net- work that generates a corresponding sentence. The model is trained to maximize the likelihood of the sentence given the image. Experiments on several datasets show the robust- ness of NIC in terms of qualitative results (the generated sentences are very reasonable) and quantitative evaluations, using either ranking metrics or BLEU, a metric used in ma- chine translation to evaluate the quality of generated sentences.
Results:
Since our model is data driven and trained end-to-end, and given the abundance of datasets, we wanted to answer questions such as “how dataset size affects generalization”, “what kinds of transfer learning it would be able to achieve”, and “how it

would deal with weakly labeled examples". As a result, we performed experiments on five different datasets, which enabled us to understand our model in depth.

Title: Where to put the Image in an Image Caption Generator
Authors: Marc Tanti, Albert Gatt, Kenneth P. Camilleri
Datasets: The datasets used for all experiments were the version of Flickr8K (Hodosh et al., 2013), Flickr30K (Young et al., 2014), and MSCOCO (Lin et al., 2014) distributed by Karpathy and Fei-Fei (2015).6 All three datasets consist of images taken from Flickr combined with between five and seven manually written captions per image. The provided datasets are split into a training, validation, and test set using the following number of images respectively: Flickr8K - 6000, 1000, 1000; Flickr30K - 29000, 1014, 1000; MSCOCO - 82783, 5000, 5000. The images are already vectorised into 4096-element vectors via the activations of layer 'fc7' (the penultimate layer) of the VGG OxfordNet 19-layer convolutional neural network (Simonyan and Zisserman, 2014), which was trained for object recognition on the ImageNet dataset (Deng et al., 2009).
The known vocabulary consists of all the words in the captions of the training set that occur at least 5 times. This amounts to 2539 tokens for Flickr8K, 7415 tokens for Flickr30K, and 8792 tokens for MSCOCO. These words are used both as inputs, which are embedded and fed to the RNN, and as outputs, which are assigned probabilities by the softmax function. Any other word which is not part of the vocabulary is replaced with an UNKNOWN token.
Methodology: In this paper author empirically showed that it is not especially detrimental to performance whether one architecture is used or another. The merge architecture does have practical advantages, as conditioning by merging allows the RNN's hidden state vector to shrink in size by up to four times.
Results: Three runs of each experiment, on each of the three datasets, were performed. For the various evaluation measures, we report the mean together with the standard deviation (reported in parentheses) over the three runs. For each run, the initial model weights, minibatch selections, and dropout selections are different since these are randomly determined. Everything else is identical across runs.

Title: Show and Tell: Lessons Learned from the 2015 MSCOCO Image Captioning Challenge .
Author: Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan
Datasets: For evaluation we use a number of datasets which consist of images and sentences in English describing these images. The statistics of the datasets are as follows:

With the exception of SBU, each image has been annotated by labelers with 5 sentences that are relatively visual and unbiased. SBU consists of descriptions given by image owners when they uploaded them to Flickr. As such they are not guaranteed to be visual or unbiased and thus this dataset has more noise. The Pascal dataset is customary used for testing only after a system has been trained on different data such as any of the other four dataset. In the case of SBU, we hold out 1,000 images for testing and train on the rest as used by [14]. Similarly,
we reserve 4 K random images from the MSCOCO validation set as test, called COCO-4k, and use it to report results in the following section.

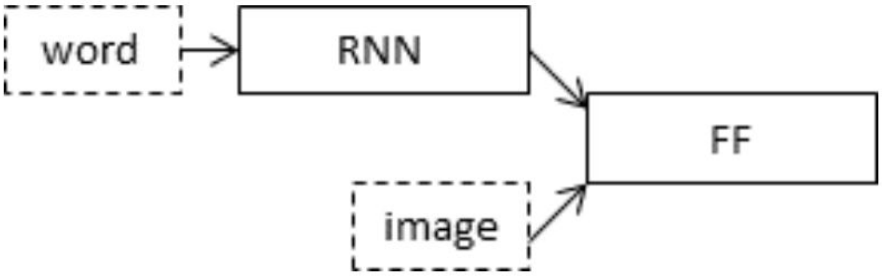
Dataset name	size		
	train	valid.	test
Pascal VOC 2008 [2]	-	-	1,000
Flickr8k [42]	6,000	1,000	1,000
Flickr30k [43]	28,000	1,000	1,000
MSCOCO [44]	82,783	40,504	40,775
SBU [18]	1M	-	-

Methodology: When we first submitted our image captioning paper to CVPR 2015, we used the best convolutional neural network at the time, known as GoogleLeNet [48], which had 22 layers, and was the winner of the 2014 ImageNet competition. Later on, an even better approach was proposed in [24] and included a new method, called Batch Normalization, to better normalize each layer of a neural network with respect to the current batch of examples, so asto be more robust to nonlinearities.
The new approach got significant improvement on the ImageNet task (going from 6.67 percent down to 4.8 percent top-5 error) and the MSCOCO image captioning task, improving BLEU-4 by 2 points absolute.
Image Model Fine Tuning In the original set of experiments, to avoid overfitting we initialized the image convolutional network with a pretrained model (we first used GoogleLeNet, then switched to the better Batch Normalization model), but then fixed its parameters and only trained the LSTM part of the model on the MS COCO training set.
Result: Since our model is data driven and trained end-to-end, and given the abundance of datasets, we wanted to answer questions such as “how dataset size affects generalization”, “what kinds of transfer learning it would be able to achieve”, and “how it would deal with weakly labelled examples”. As a result, we performed experiments on five different datasets, explained in Section 4.2, which enabled us to understand our model in depth.

Research Paper Integration:

We defined a deep learning based on the "*merge-model*" described by Marc Tanti, et al. in their 2017 papers:

- Where to put the Image in an Image Caption Generator, 2017.
- What is the Role of Recurrent Neural Networks (RNNs) in an Image Caption Generator?, 2017.



Schematic of the Merge Model For Image Captioning

The model is described in three parts:

- **Photo Feature Extractor**: This is a 16-layer VGG model pre-trained on the ImageNet dataset. We have pre-processed the photos with the VGG model (without the output layer) and will use the extracted features predicted by this model as input.
- **Sequence Processor**: This is a word embedding layer for handling the text input, followed by a Long Short-Term Memory (LSTM) recurrent neural network layer.
- **Decoder** (for lack of a better name): Both the feature extractor and sequence processor output a fixed-length vector. These are merged together and processed by a Dense layer to make a final prediction.

The Photo Feature Extractor model expects input photo features to be a vector of 4,096 (VGG16) / 1,536 (Inception ResNet V2) elements. These are processed by a Dense layer to produce a 256 element representation of the photo.

The Sequence Processor model expects input sequences with a pre-defined length (34 words) which are fed into an Embedding layer that uses a mask to ignore padded values. This is followed by an LSTM layer with 256 memory units.

Both the input models produce a 256 element vector. Further, both input models use regularization in the form of 50% dropout. This is to reduce overfitting the training dataset, as this model configuration learns very fast.

The Decoder model merges the vectors from both input models using an addition operation. This is then fed to a Dense 256 neuron layer and then to a final output Dense layer that makes a softmax prediction over the entire output vocabulary for the next word in the sequence.

Algorithms/ Approach used and Implemented:

Data Gathering Phase

Flickr8k_Dataset.zip (1 Gigabyte) An archive of all photographs.
Flickr8k_text.zip (2.2 Megabytes) An archive of all text descriptions for photographs.
Flicker8k_Dataset: Contains 8092 photographs in JPEG format.
Flickr8k_text: Contains a number of files containing different sources of descriptions for the photographs. The dataset has a pre-defined training dataset (6,000 images), development dataset (1,000 images), and test dataset (1,000 images).

Preparation of Photo Data

Pre-trained models are used to interpret the content of the photos which are Inception ResNet V2 and VGG 16.

The “photo features” are pre-computed using the pre-trained model and saved to a file. These features are loaded later and fed into our model as the interpretation of a given photo in the dataset. It is no different than running the photo through the full VGG model, it is just we will have done it once in advance.

This is an optimization that will make training our models faster and consume less memory.

We pop (removed) the last layer from the loaded model, as this is the model used to predict a classification for a photo. We are not interested in classifying images, but we were interested in the internal representation of the photo right before a classification was made. These are the “features” that the model has extracted from the photo.

Preparation of Text Data

The dataset contains multiple descriptions for each photograph and the text of the descriptions requires cleaning.

Each photo has a unique identifier. This identifier is used on the photo filename and in the text file of descriptions.

Each photo identifier maps to a list of one or more textual descriptions.

We clean the text in the following ways in order to reduce the size of the vocabulary of words we will need to work with:

- Convert all words to lowercase.
- Remove all punctuation.

- Remove all words that are one character or less in length (e.g. ‘a’).
- Remove all words with numbers in them.

Developing Deep Learning Model

Loading Data

We load the prepared photo and text data so that we can use it to fit the model.

We train the data on all of the photos and captions in the training dataset. While training, we monitor the performance of the model on the development dataset and use that performance to decide when to save models to file.

The train and development dataset have been predefined in the *Flickr_8k.trainImages.txt* and *Flickr_8k.devImages.txt* files respectively, that both contain lists of photo file names. From these file names, we extract the photo identifiers and use these identifiers to filter photos and descriptions for each set.

The model we develop generates a caption given a photo, and the caption generates one word at a time. The sequence of previously generated words will be provided as input. Therefore, we need a ‘*first word*’ to kick-off the generation process and a ‘*last word*‘ to signal the end of the caption.

We will use the strings ‘*startseq*‘ and ‘*endseq*‘ for this purpose. These tokens are added to the loaded descriptions as they are loaded. It is important to do this before encoding the text so that the tokens are also encoded correctly.

The description text will need to be encoded to numbers before it can be presented to the model as in input or compared to the model’s predictions.

Encoding the data requires a consistent mapping from words to unique integer values. Keras provides the *Tokenizer* class that can learn this mapping from the loaded description data.

Each description is split into words. The model is provided one word and the photo and generates the next word. Then the first two words of the description are provided to the model as input with the image to generate the next word. This is how the model is trained.

For example, the input sequence "*little girl running in field*" would be split into 6 input-output pairs to train the model:

X1	**X2 (text sequence),**	**y (word)**
photo	startseq,	little
photo	startseq, little,	girl
photo	startseq, little, girl,	running
photo	startseq, little, girl, running,	in
photo	startseq, little, girl, running, in,	field
photo	startseq, little, girl, running, in, field,	endseq

When the model is used to generate descriptions, the generated words are concatenated and recursively provided as input to generate a caption for an image.

There are two input arrays to the model: one for photo features and one for the encoded text. There is one output for the model which is the encoded next word in the text sequence.

The input text is encoded as integers, which will be fed to a word embedding layer. The photo features will be fed directly to another part of the model. The model will output a prediction, which will be a probability distribution over all words in the vocabulary.

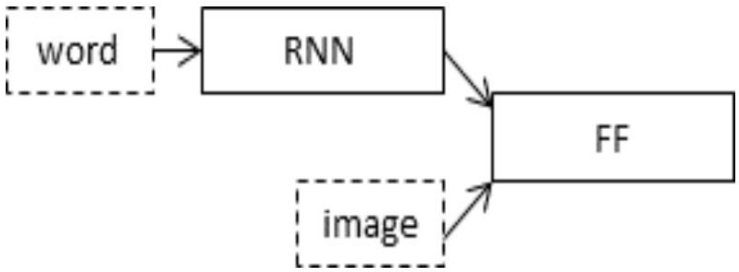
The output data will therefore be a one-hot encoded version of each word, representing an idealized probability distribution with 0 values at all word positions except the actual word position, which has a value of 1.

We calculated the maximum number of words in the longest description i.e. 34.

Defining the Model

We will define a deep learning based on the "*merge-model*" described by Marc Tanti, et al. in their 2017 papers:

- Where to put the Image in an Image Caption Generator, 2017.
- What is the Role of Recurrent Neural Networks (RNNs) in an Image Caption Generator?, 2017.



Schematic of the Merge Model For Image Captioning

The model is described in three parts:

- **Photo Feature Extractor**. This is a 16-layer VGG model pre-trained on the ImageNet dataset. We have pre-processed the photos with the VGG model (without the output layer) and will use the extracted features predicted by this model as input.
- **Sequence Processor**. This is a word embedding layer for handling the text input, followed by a Long Short-Term Memory (LSTM) recurrent neural network layer.

- **Decoder** (for lack of a better name). Both the feature extractor and sequence processor output a fixed-length vector. These are merged together and processed by a Dense layer to make a final prediction.

The Photo Feature Extractor model expects input photo features to be a vector of 4,096 (VGG16) / 1,536 (Inception ResNet V2) elements. These are processed by a Dense layer to produce a 256 element representation of the photo.

The Sequence Processor model expects input sequences with a pre-defined length (34 words) which are fed into an Embedding layer that uses a mask to ignore padded values. This is followed by an LSTM layer with 256 memory units.

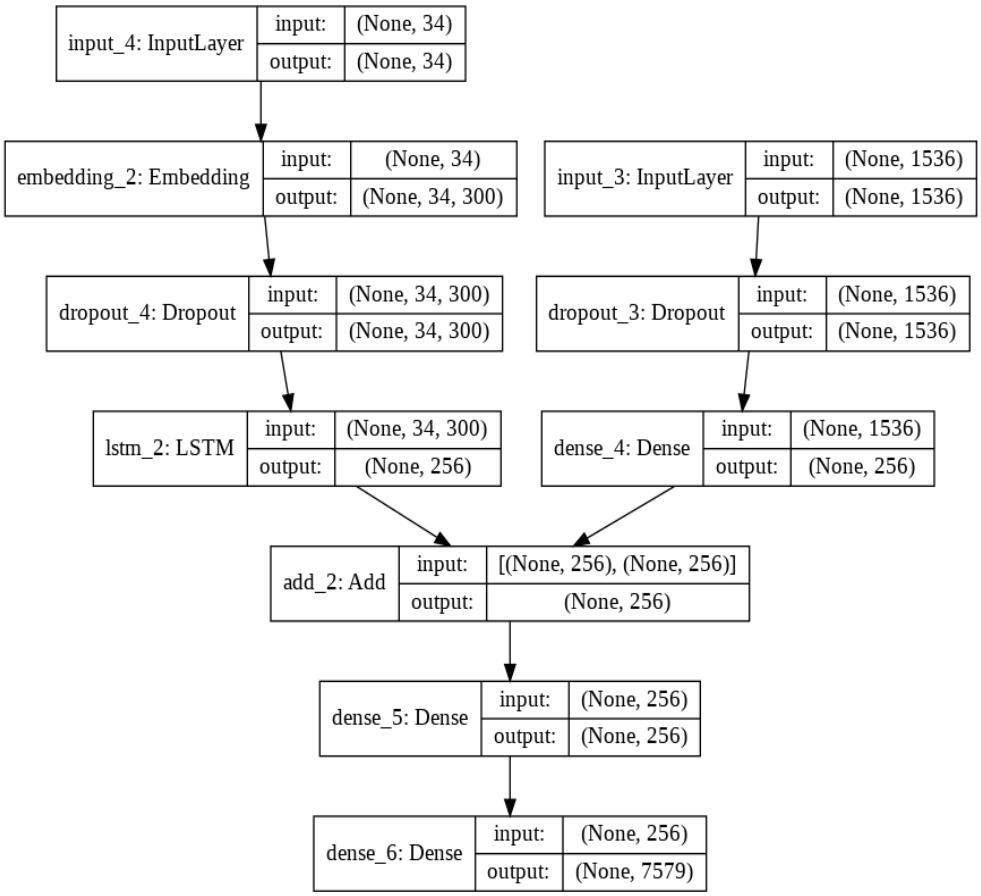
Both the input models produce a 256 element vector. Further, both input models use regularization in the form of 50% dropout. This is to reduce overfitting the training dataset, as this model configuration learns very fast.

The Decoder model merges the vectors from both input models using an addition operation. This is then fed to a Dense 256 neuron layer and then to a final output Dense layer that makes a softmax prediction over the entire output vocabulary for the next word in the sequence.

Model with Inception ResNet V2 and FastText

Layer (type)	Output Shape	Param #	Connected to
input_2 (InputLayer)	(None, 34)	0	
input_1 (InputLayer)	(None, 1536)	0	
embedding_1 (Embedding)	(None, 34, 300)	2273700	input_2[0][0]
dropout_1 (Dropout)	(None, 1536)	0	input_1[0][0]
dropout_2 (Dropout)	(None, 34, 300)	0	embedding_1[0][0]
dense_1 (Dense)	(None, 256)	393472	dropout_1[0][0]
lstm_1 (LSTM)	(None, 256)	570368	dropout_2[0][0]
add_1 (Add)	(None, 256)	0	dense_1[0][0] lstm_1[0][0]
dense_2 (Dense)	(None, 256)	65792	add_1[0][0]
dense_3 (Dense)	(None, 7579)	1947803	dense_2[0][0]

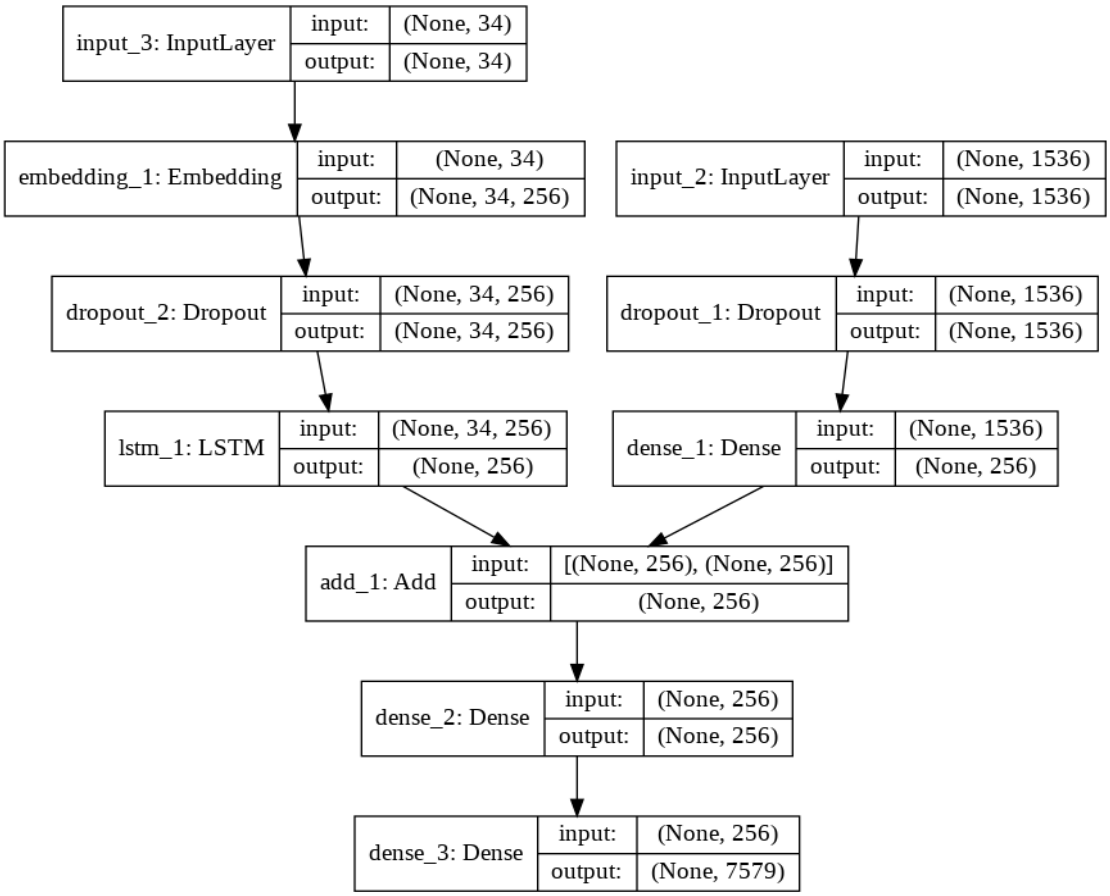
Total params: 5,251,135
Trainable params: 5,251,135
Non-trainable params: 0



Model with Inception ResNet

Layer (type)	Output Shape	Param #	Connected to
input_3 (InputLayer)	(None, 34)	0	
input_2 (InputLayer)	(None, 1536)	0	
embedding_1 (Embedding)	(None, 34, 256)	1940224	input_3[0][0]
dropout_1 (Dropout)	(None, 1536)	0	input_2[0][0]
dropout_2 (Dropout)	(None, 34, 256)	0	embedding_1[0][0]
dense_1 (Dense)	(None, 256)	393472	dropout_1[0][0]
lstm_1 (LSTM)	(None, 256)	525312	dropout_2[0][0]
add_1 (Add)	(None, 256)	0	dense_1[0][0] lstm_1[0][0]
dense_2 (Dense)	(None, 256)	65792	add_1[0][0]
dense_3 (Dense)	(None, 7579)	1947803	dense_2[0][0]

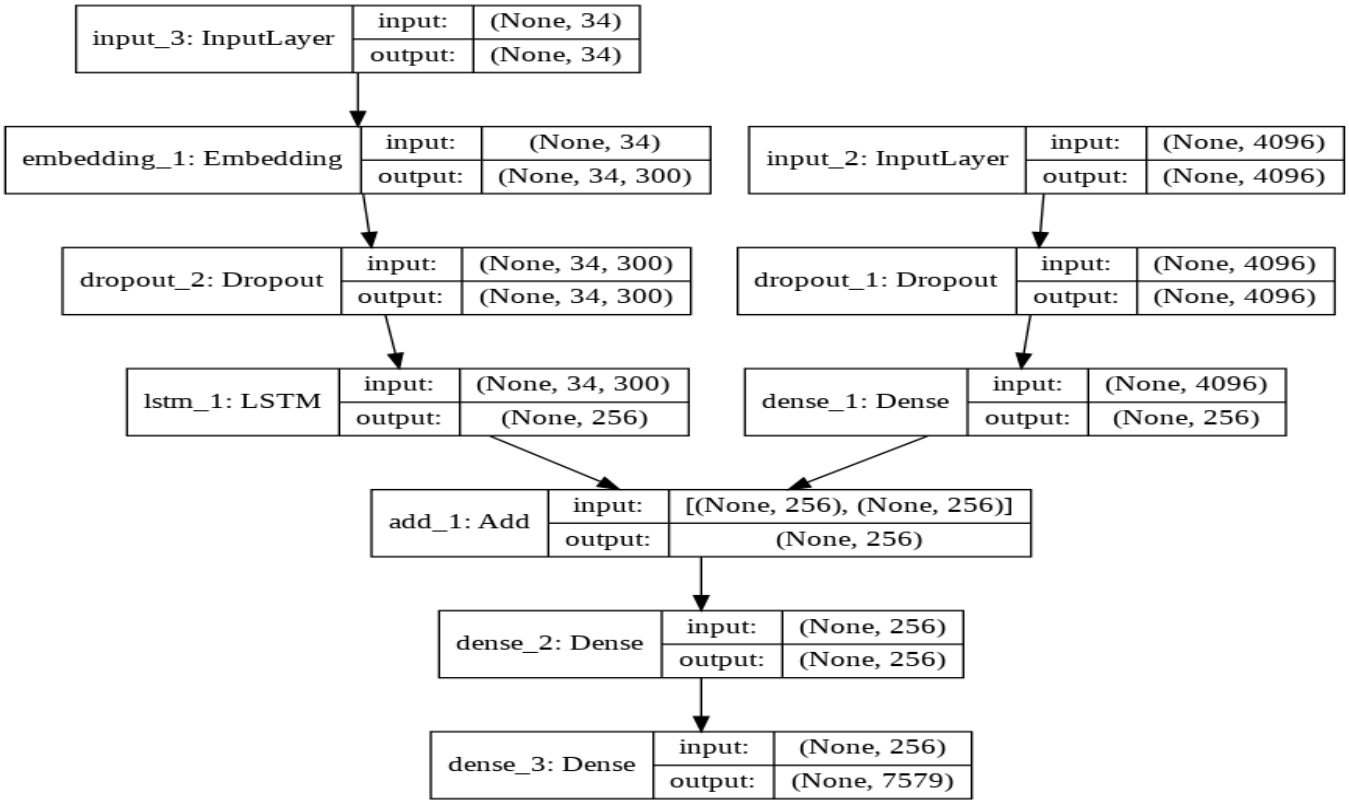
Total params: 4,872,603
Trainable params: 4,872,603
Non-trainable params: 0



Model with VGG 16 and FastText

Layer (type)	Output Shape	Param #	Connected to
input_3 (InputLayer)	(None, 34)	0	
input_2 (InputLayer)	(None, 4096)	0	
embedding_1 (Embedding)	(None, 34, 300)	2273700	input_3[0][0]
dropout_1 (Dropout)	(None, 4096)	0	input_2[0][0]
dropout_2 (Dropout)	(None, 34, 300)	0	embedding_1[0][0]
dense_1 (Dense)	(None, 256)	1048832	dropout_1[0][0]
lstm_1 (LSTM)	(None, 256)	570368	dropout_2[0][0]
add_1 (Add)	(None, 256)	0	dense_1[0][0] lstm_1[0][0]
dense_2 (Dense)	(None, 256)	65792	add_1[0][0]
dense_3 (Dense)	(None, 7579)	1947803	dense_2[0][0]

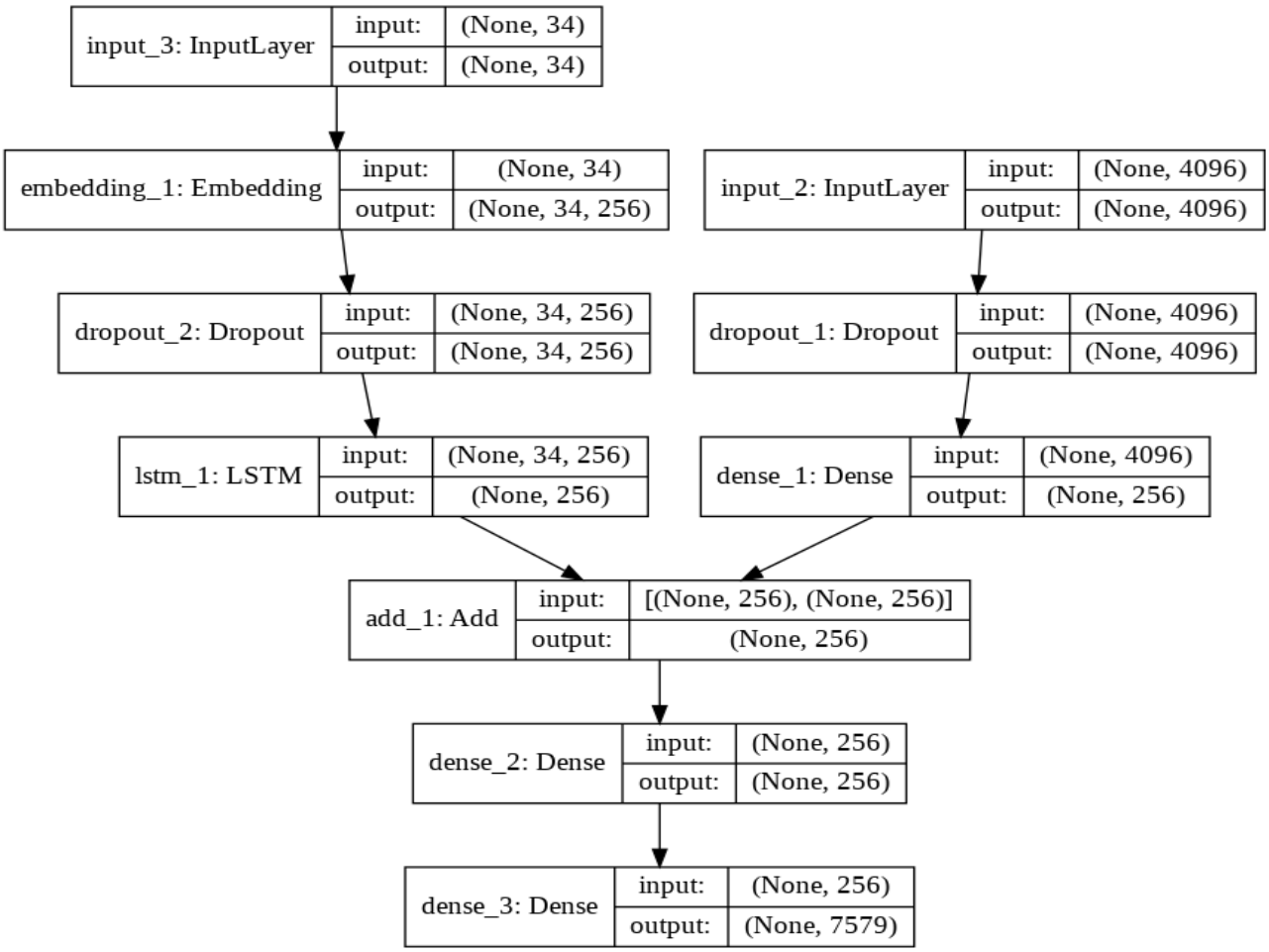
Total params: 5,906,495
Trainable params: 5,906,495
Non-trainable params: 0



Model with VGG 16

Layer (type)	Output Shape	Param #	Connected to
input_3 (InputLayer)	(None, 34)	0	
input_2 (InputLayer)	(None, 4096)	0	
embedding_1 (Embedding)	(None, 34, 256)	1940224	input_3[0][0]
dropout_1 (Dropout)	(None, 4096)	0	input_2[0][0]
dropout_2 (Dropout)	(None, 34, 256)	0	embedding_1[0][0]
dense_1 (Dense)	(None, 256)	1048832	dropout_1[0][0]
lstm_1 (LSTM)	(None, 256)	525312	dropout_2[0][0]
add_1 (Add)	(None, 256)	0	dense_1[0][0] lstm_1[0][0]
dense_2 (Dense)	(None, 256)	65792	add_1[0][0]
dense_3 (Dense)	(None, 7579)	1947803	dense_2[0][0]

Total params: 5,527,963
Trainable params: 5,527,963
Non-trainable params: 0



To improvise we have used pre-trained word vectors. The model learned the word vectors as part of fitting the model. Better performance is achieved by using word vectors pre-trained on a much larger corpus of text, such as news articles or Wikipedia.

We used FastText 300-d to make word embeddings of our dataset. FastText was proposed by Facebook in 2016. FastText breaks words into several n-grams (sub-words) rather than feeding each individual work into Neural Network. Example: the tri-grams for the word “delhi” is “del”,”elh” and “lhi”. The sum of all these n-grams will be the word embedding vector for “delhi”. Given the training dataset, we can get word embeddings for all the n-grams after training

the Neural Network. This gives rare words the chance to be properly represented since it is highly probable that we find their n-grams in other words.

We make use of keras's pre-processing text's tokenizer. The dataset was tokenized (Text corpus is vectorized by turning each text into either a vector where the coefficient for each token could be binary, based on word count or tf-idf, or into a sequence of integers (where each integer is the index of a token in dictionary)) and its word index (dictionary mappings of words (str) to their rank/index (int) which is only set after tokenizer was trained on the text) was used to make a matrix of embedded words (integer matrix) which is to be used as the default weights for the neural network.

Fitting the Model

The model learns fast and quickly overfits the training dataset. For this reason, we monitor the skill of the trained model on the holdout development dataset. When the skill of the model on the development dataset improves at the end of an epoch, we save the whole model to file.

At the end of the run, we can then use the saved model with the best skill on the training dataset as our final model.

We do this by defining a *ModelCheckpoint* in Keras and specifying it to monitor the minimum loss on the validation dataset and save the model to a file that has both the training and validation loss in the filename.

We specify the checkpoint in the call to *fit()* via the *callbacks* argument. We also specify the development dataset in *fit()* via the *validation_data* argument.

We fit the model for 20 epochs.We have the data generator yield one photo's worth of data per batch. This will be all of the sequences generated for a photo and its set of descriptions. This saves RAM as colab resources were exhausted without using generator.

Evaluate Model

Once the model is fit, we evaluate the skill of its predictions on test,train and validation datasets.

We evaluate a model by generating descriptions for all photos in the respective dataset and evaluating those predictions with a standard cost function.

We generate a description for a photo using a trained model. This involves passing in the start description token '*startseq*', generating one word, then calling the model recursively with generated words as input until the end of sequence token is reached '*endseq*' or the maximum description length is reached.

With the encoding of text, we create tokenizers and save them so that we can load it quickly whenever we need it without needing the entire Flickr8K dataset.

The actual and predicted descriptions are collected and evaluated collectively using the corpus BLEU score and ROUGE score that summarizes how close the generated text is to the expected text.

BLEU and ROUGE scores are used in text translation for evaluating translated text against one or more reference translations.

Here, we compare each generated description against all of the reference descriptions for the photograph. We then calculate BLEU scores for 1, 2, 3 and 4 cumulative n-grams. In ROUGE, rouge-1, rouge-2,rouge-l gives the scores for 1 gram, 2 gram and Longest common subsequence. With each of these we get precision, recall and f-score values. Since, rouge metrics compare predicted caption to only one actual caption out of 5 so we compared the predicted caption with all of our actual captions and took max, min and average scores. This resulted in one resultant max, min and average metric for a particular image which we added for all images in sub-dataset (train / test / validation) and divided by the number of images.

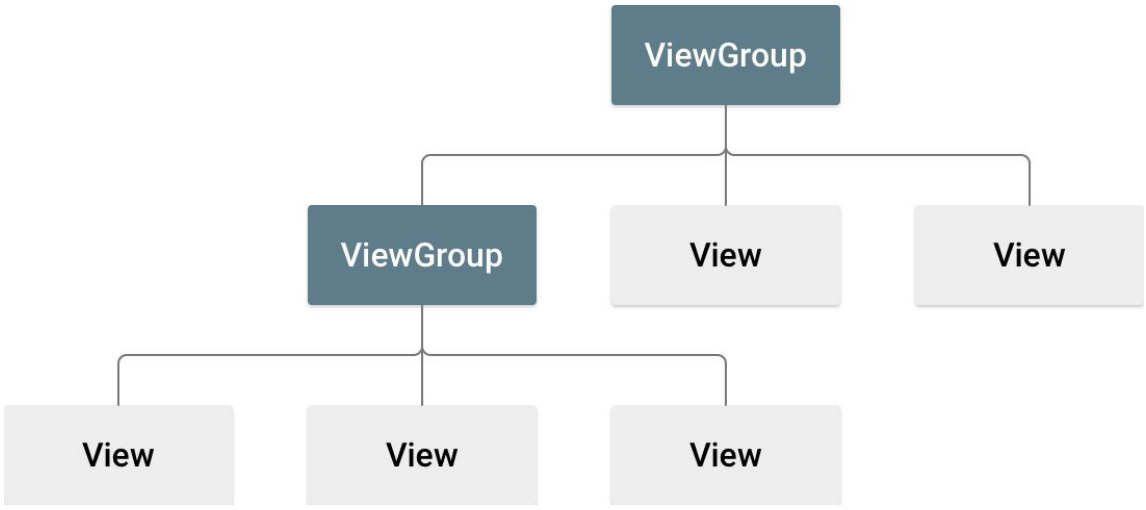
App Formation:

After working on the machine learning part, we implement the whole model on an android app and make is more user friendly.

We build an android app using Android Studio 3.2.1 where we implement the code using XML and Kotlin. We also use Tensor flow and Flask to merge the python model with the app and get the output of the model directly on the app.

Firstly, we create an android app with the desired layout. To make the layout we use material design and other features of the android studio to make it more attractive.

A layout defines the structure for a user interface in our app, such as in an activity. All elements in the layout are built using a hierarchy of View and ViewGroup objects. A View usually draws something the user can see and interact with. Whereas a ViewGroup is an invisible container that defines the layout structure for View and other ViewGroup objects, as shown in figure.



The View objects are usually called "widgets" and can be one of many subclasses, such as Button or Text View. The View Group are usually called "layouts" can be one of many types that provide a different layout structure, such as Linear Layout and Constraint Layout. We used Linear Layout to give the basic structure and make a placeholder to place the photo. We make a spinner to select the model that a user wants to run and generate caption of the image.

Secondly, after preparing the layout we make two choices for the user to select an image either by directly capturing it from the camera or select an image from the gallery of the mobile.

Algorithm to Capture the Image:

With the commence of Android Marshmallow, runtime permissions need to be implemented forefront.

Add the following permissions in the Android Manifest.xml file, above the application tag.

```
<uses-feature
        android:name="android.hardware.camera"
        android:required="false" />
    <uses-feature
        android:name="android.hardware.camera.autofocus"
        android:required="false" />
    <uses-feature
        android:name="android.hardware.camera.flash"
        android:required="false" />

    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="ANDROID.PERMISSION.READ_EXTERNAL_STORAGE"/>
```

By adding android.hardware.camera, Play Store detects and prevents installing the application on devices with no camera.

Intent is the standard way to delegate actions to another application.
To start the native camera the Intent
requires android.provider.MediaStore.ACTION_IMAGE_CAPTURE.

To choose an image from gallery, the Intent requires the following
argument: **Intent.ACTION_GET_CONTENT**.

There are a lot of inferences to be drawn from the code above.

- We need to ask for the Camera runtime permissions when the user starts the activity.
- As we are starting the intent to get some result back, we need to call startActivityForResult with the relevant arguments
- Instead of using a dialog to separately call the Intents for Camera and Gallery, we've used a method **getPickImageChooserIntent()** that creates a single chooser intent for all the camera and gallery intents(note the documents intent). **Intent.EXTRA_INITIAL_INTENTS** is used to add the multiple application intents at one place
- For the camera **intent, MediaStore.EXTRA_OUTPUT** is passed as an extra to specify the image storage path. Without this you'll be returned only a small resolution image.
- The URI path for the image returned by camera is fetched inside the method **getCaptureImageOutputUri()**.
- The onActivityResult essentially returns a URI to the image. Some devices do return the bitmap as data.getExtras().get("data");.
- When an image is clicked, the camera screen while returning restarts the activity thereby causing the URI stored from the method **getCaptureImageOutputUri()** to become null. Hence it's essential that we store and restore that URI using **onSaveInstanceState()**and **onRestoreInstanceState()**.
- The bitmap is retrieved from the URI in the following line of code. myBitmap = MediaStore.Images.Media.getBitmap(this.getContentResolver(), picUri);
- Devices like Samsung galaxy are known to capture the image in landscape orientation. Retrieving the image and displaying as it is can cause it to be displayed in the wrong orientation. Hence we've called the method **rotateImageIfRequired(myBitmap, picUri);**
- ExifInterface is a class for reading and writing Exif tags in a JPEG file or a RAW image file.
- In the end we call the method **getResizedBitmap()** to scale the bitmap by width or height(whichever is larger) and set the image to the image view using setImageBitmap.

After these steps we can now either capture image from the camera or directly upload an image from the gallery.

Algorithm For Text to Speech:

We have to create an object of TextToSpeech class and this is the Android class that does the job of converting the text to speech and also provides parameters to set the language, speed and pitch.Method textToSpeech.setLanguage() is used to set the language of the speech. Many languages are supported by Android TTS engine. In the current example Android app we will use

english(United States).

After setting up the speech parameters, we have to invoke OnClickListener when button is clicked. Inside the listener, we will convert user input into string. textToSpeech.speak() method does the main job of the text. It is good practice to shutdown the Android TTS engine once it's task is over. To accomplish this onDestroy() method is used.

```
private void speakOut() {

    String text = txtText.getText().toString();

    tts.speak(text, TextToSpeech.QUEUE_FLUSH, null);
}
```

Result:

Model with Inception ResNet V2 and FastText

Epoch 1/20

6000/6000 [==============================] - 787s 131ms/step - loss: 4.1096 - val_loss: 3.7981

Epoch 00001: val_loss improved from inf to 3.79809, saving model to
model-ep001-loss4.130-val_loss3.798.h5

Epoch 2/20

6000/6000 [==============================] - 785s 131ms/step - loss: 3.5695 - val_loss: 3.6499

Epoch 00002: val_loss improved from 3.79809 to 3.64990, saving model to
model-ep002-loss3.591-val_loss3.650.h5

Epoch 3/20

6000/6000 [==============================] - 792s 132ms/step - loss: 3.3379 - val_loss: 3.6310

Epoch 00003: val_loss improved from 3.64990 to 3.63100, saving model to
model-ep003-loss3.360-val_loss3.631.h5

Epoch 4/20

6000/6000 [==============================] - 782s 130ms/step - loss: 3.1950 - val_loss: 3.6452

Epoch 00004: val_loss did not improve from 3.63100

Epoch 5/20

6000/6000 [==============================] - 796s 133ms/step - loss: 3.0949 - val_loss: 3.6888

Epoch 00005: val_loss did not improve from 3.63100

Epoch 6/20

6000/6000 [==============================] - 795s 133ms/step - loss: 3.0214 - val_loss: 3.7159

Epoch 00006: val_loss did not improve from 3.63100

Epoch 7/20

6000/6000 [==============================] - 790s 132ms/step - loss: 2.9623 - val_loss: 3.7623

Epoch 00007: val_loss did not improve from 3.63100

Epoch 8/20

6000/6000 [==============================] - 784s 131ms/step - loss: 2.9207 - val_loss: 3.7796

Epoch 00008: val_loss did not improve from 3.63100

Epoch 9/20

6000/6000 [==============================] - 776s 129ms/step - loss: 2.8833 - val_loss: 3.8126

Epoch 00009: val_loss did not improve from 3.63100

Epoch 10/20

6000/6000 [==============================] - 777s 130ms/step - loss: 2.8526 - val_loss: 3.7904

Epoch 00010: val_loss did not improve from 3.63100

Epoch 11/20

6000/6000 [==============================] - 775s 129ms/step - loss: 2.8267 - val_loss: 3.8225

Epoch 00011: val_loss did not improve from 3.63100

Epoch 12/20

6000/6000 [==============================] - 775s 129ms/step - loss: 2.8079 - val_loss: 3.8293

Epoch 00012: val_loss did not improve from 3.63100

Epoch 13/20

6000/6000 [==============================] - 772s 129ms/step - loss: 2.7886 - val_loss: 3.8901

Epoch 00013: val_loss did not improve from 3.63100

Epoch 14/20

6000/6000 [==============================] - 789s 132ms/step - loss: 2.7789 - val_loss: 3.9223

Epoch 00014: val_loss did not improve from 3.63100

Epoch 15/20

6000/6000 [==============================] - 813s 136ms/step - loss: 2.7659 - val_loss: 3.9117

Epoch 00015: val_loss did not improve from 3.63100

Epoch 16/20

6000/6000 [==============================] - 807s 135ms/step - loss: 2.7557 - val_loss: 3.9356

Epoch 00016: val_loss did not improve from 3.63100

Epoch 17/20

6000/6000 [==============================] - 812s 135ms/step - loss: 2.7478 - val_loss: 3.9597

Epoch 00017: val_loss did not improve from 3.63100

Epoch 18/20

6000/6000 [==============================] - 818s 136ms/step - loss: 2.7392 - val_loss: 3.9812

Epoch 00018: val_loss did not improve from 3.63100

Epoch 19/20

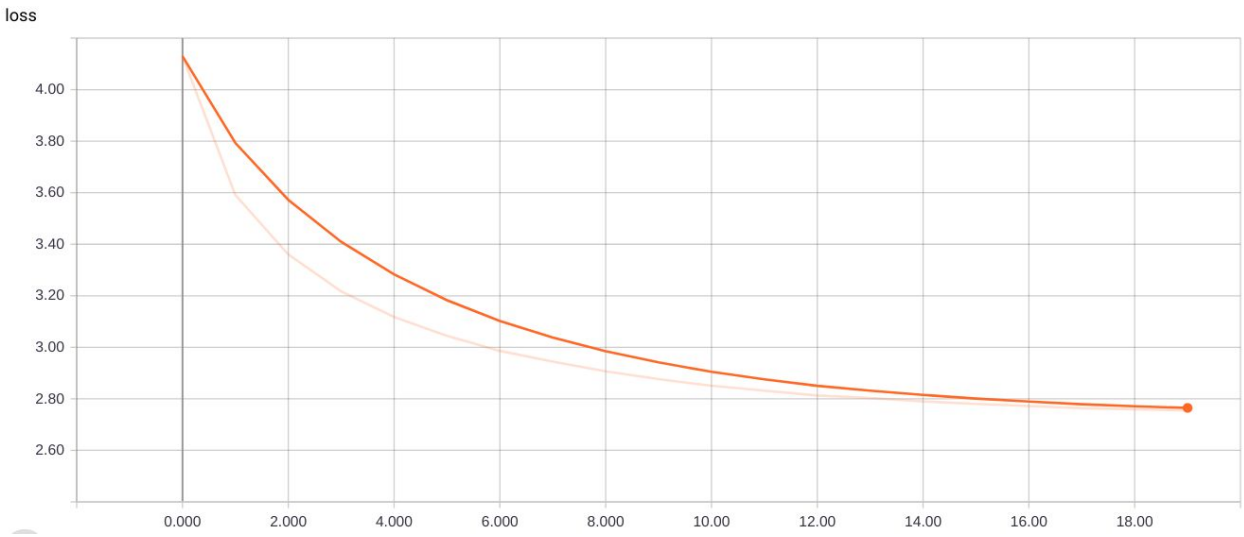
6000/6000 [==============================] - 825s 138ms/step - loss: 2.7364 - val_loss: 3.9889

Epoch 00019: val_loss did not improve from 3.63100

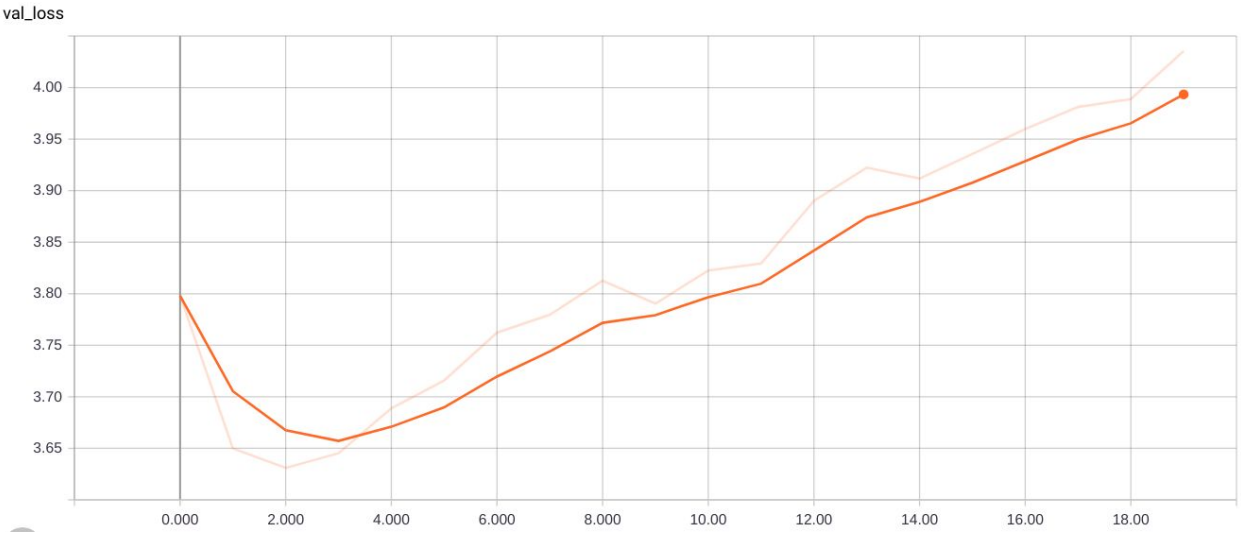
Epoch 20/20

6000/6000 [==============================] - 819s 137ms/step - loss: 2.7308 - val_loss: 4.0355

Epoch 00020: val_loss did not improve from 3.63100



Loss on training dataset



Loss on validation dataset

Model with Inception ResNet V2

Epoch 1/20

6000/6000 [==============================] - 535s 89ms/step - loss: 4.4907 - val_loss: 3.9083

Epoch 00001: val_loss improved from inf to 3.90825, saving model to model-ep001-loss4.511-val_loss3.908.h5

Epoch 2/20

6000/6000 [==============================] - 534s 89ms/step - loss: 3.6803 - val_loss: 3.7235

Epoch 00002: val_loss improved from 3.90825 to 3.72346, saving model to model-ep002-loss3.701-val_loss3.723.h5

Epoch 3/20

6000/6000 [==============================] - 533s 89ms/step - loss: 3.4121 - val_loss: 3.6833

Epoch 00003: val_loss improved from 3.72346 to 3.68325, saving model to model-ep003-loss3.433-val_loss3.683.h5

Epoch 4/20

6000/6000 [==============================] - 537s 89ms/step - loss: 3.2504 - val_loss: 3.6963

Epoch 00004: val_loss did not improve from 3.68325

Epoch 5/20

6000/6000 [==============================] - 535s 89ms/step - loss: 3.1381 - val_loss: 3.7217

Epoch 00005: val_loss did not improve from 3.68325

Epoch 6/20

6000/6000 [==============================] - 537s 90ms/step - loss: 3.0562 - val_loss: 3.7385

Epoch 00006: val_loss did not improve from 3.68325

Epoch 7/20

6000/6000 [==============================] - 535s 89ms/step - loss: 2.9936 - val_loss: 3.7721

Epoch 00007: val_loss did not improve from 3.68325

Epoch 8/20

6000/6000 [==============================] - 536s 89ms/step - loss: 2.9405 - val_loss: 3.8061

Epoch 00008: val_loss did not improve from 3.68325

Epoch 9/20

6000/6000 [==============================] - 537s 89ms/step - loss: 2.8997 - val_loss: 3.8207

Epoch 00009: val_loss did not improve from 3.68325

Epoch 10/20

6000/6000 [==============================] - 538s 90ms/step - loss: 2.8664 - val_loss: 3.8656

Epoch 00010: val_loss did not improve from 3.68325

Epoch 11/20

6000/6000 [==============================] - 533s 89ms/step - loss: 2.8392 - val_loss: 3.9066

Epoch 00011: val_loss did not improve from 3.68325

Epoch 12/20

6000/6000 [==============================] - 536s 89ms/step - loss: 2.8162 - val_loss: 3.9228

Epoch 00012: val_loss did not improve from 3.68325

Epoch 13/20

6000/6000 [==============================] - 534s 89ms/step - loss: 2.8031 - val_loss: 3.9522

Epoch 00013: val_loss did not improve from 3.68325

Epoch 14/20

6000/6000 [==============================] - 530s 88ms/step - loss: 2.7846 - val_loss: 3.9770

Epoch 00014: val_loss did not improve from 3.68325

Epoch 15/20

6000/6000 [==============================] - 532s 89ms/step - loss: 2.7668 - val_loss: 4.0082

Epoch 00015: val_loss did not improve from 3.68325

Epoch 16/20

6000/6000 [==============================] - 534s 89ms/step - loss: 2.7542 - val_loss: 4.0131

Epoch 00016: val_loss did not improve from 3.68325

Epoch 17/20

6000/6000 [==============================] - 535s 89ms/step - loss: 2.7462 - val_loss: 4.0174

Epoch 00017: val_loss did not improve from 3.68325

Epoch 18/20

6000/6000 [==============================] - 536s 89ms/step - loss: 2.7365 - val_loss: 4.0312

Epoch 00018: val_loss did not improve from 3.68325

Epoch 19/20

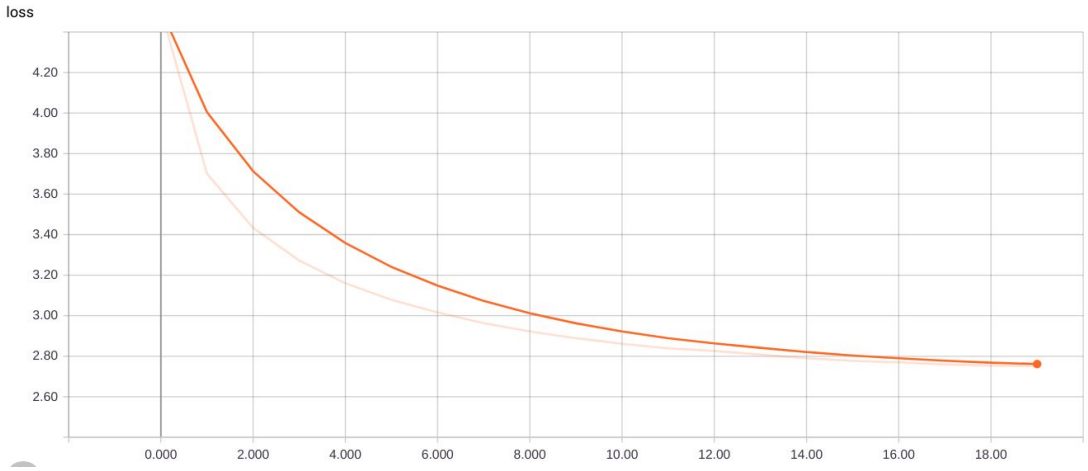
6000/6000 [==============================] - 536s 89ms/step - loss: 2.7305 - val_loss: 4.0749

Epoch 00019: val_loss did not improve from 3.68325

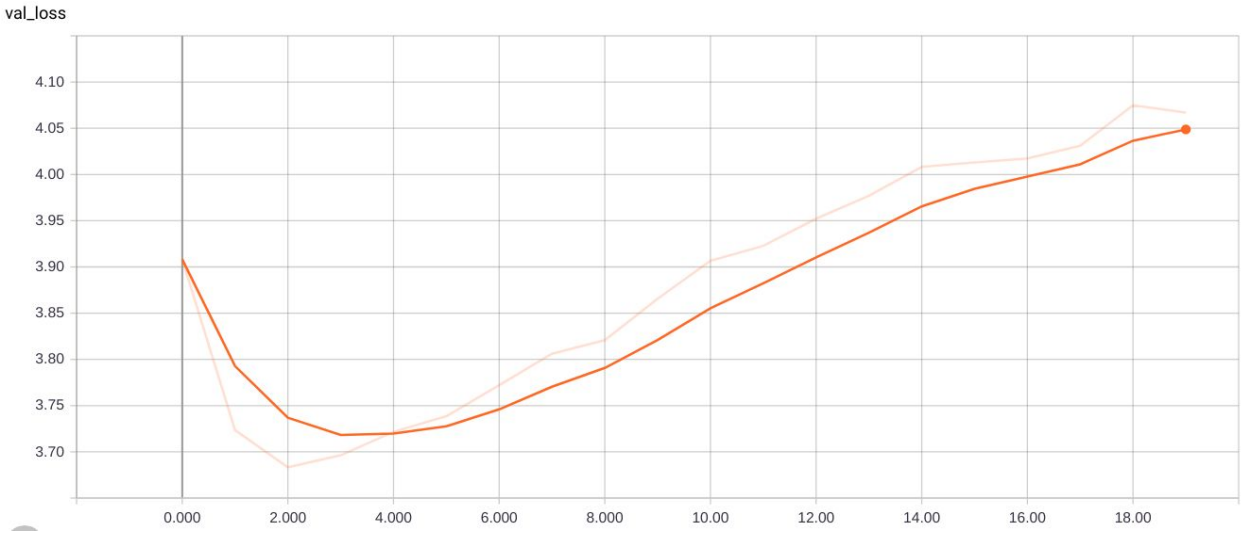
Epoch 20/20

6000/6000 [==============================] - 537s 89ms/step - loss: 2.7281 - val_loss: 4.0670

Epoch 00020: val_loss did not improve from 3.68325



Loss on training dataset



Loss on validation dataset

Model with VGG 16 and FastText

Epoch 1/20

6000/6000 [==============================] - 466s 78ms/step - loss: 4.5692 - val_loss: 4.0156

Epoch 00001: val_loss improved from inf to 4.01561, saving model to model-ep001-loss4.590-val_loss4.016.h5

Epoch 2/20

6000/6000 [==============================] - 463s 77ms/step - loss: 3.8083 - val_loss: 3.8181

Epoch 00002: val_loss improved from 4.01561 to 3.81814, saving model to model-ep002-loss3.828-val_loss3.818.h5

Epoch 3/20

6000/6000 [==============================] - 462s 77ms/step - loss: 3.5631 - val_loss: 3.7721

Epoch 00003: val_loss improved from 3.81814 to 3.77214, saving model to model-ep003-loss3.583-val_loss3.772.h5

Epoch 4/20

6000/6000 [==============================] - 457s 76ms/step - loss: 3.4209 - val_loss: 3.7708

Epoch 00004: val_loss improved from 3.77214 to 3.77078, saving model to model-ep004-loss3.440-val_loss3.771.h5

Epoch 5/20

6000/6000 [==============================] - 458s 76ms/step - loss: 3.3218 - val_loss: 3.7824

Epoch 00005: val_loss did not improve from 3.77078

Epoch 6/20

6000/6000 [==============================] - 456s 76ms/step - loss: 3.2567 - val_loss: 3.7951

Epoch 00006: val_loss did not improve from 3.77078

Epoch 7/20

6000/6000 [==============================] - 454s 76ms/step - loss: 3.2110 - val_loss: 3.7988

Epoch 00007: val_loss did not improve from 3.77078

Epoch 8/20

6000/6000 [==============================] - 453s 76ms/step - loss: 3.1745 - val_loss: 3.8171

Epoch 00008: val_loss did not improve from 3.77078

Epoch 9/20

6000/6000 [==============================] - 455s 76ms/step - loss: 3.1477 - val_loss: 3.8303

Epoch 00009: val_loss did not improve from 3.77078

Epoch 10/20

6000/6000 [==============================] - 453s 76ms/step - loss: 3.1241 - val_loss: 3.8739

Epoch 00010: val_loss did not improve from 3.77078

Epoch 11/20

6000/6000 [==============================] - 455s 76ms/step - loss: 3.1076 - val_loss: 3.8728

Epoch 00011: val_loss did not improve from 3.77078

Epoch 12/20

6000/6000 [==============================] - 452s 75ms/step - loss: 3.0918 - val_loss: 3.8766

Epoch 00012: val_loss did not improve from 3.77078

Epoch 13/20

6000/6000 [==============================] - 454s 76ms/step - loss: 3.0778 - val_loss: 3.9067

Epoch 00013: val_loss did not improve from 3.77078

Epoch 14/20

6000/6000 [==============================] - 462s 77ms/step - loss: 3.0734 - val_loss: 3.9381

Epoch 00014: val_loss did not improve from 3.77078

Epoch 15/20

6000/6000 [==============================] - 463s 77ms/step - loss: 3.0642 - val_loss: 3.9081

Epoch 00015: val_loss did not improve from 3.77078

Epoch 16/20

6000/6000 [==============================] - 463s 77ms/step - loss: 3.0595 - val_loss: 3.9424

Epoch 00016: val_loss did not improve from 3.77078

Epoch 17/20

6000/6000 [==============================] - 464s 77ms/step - loss: 3.0577 - val_loss: 3.9576

Epoch 00017: val_loss did not improve from 3.77078

Epoch 18/20

6000/6000 [==============================] - 470s 78ms/step - loss: 3.0555 - val_loss: 3.9531

Epoch 00018: val_loss did not improve from 3.77078

Epoch 19/20

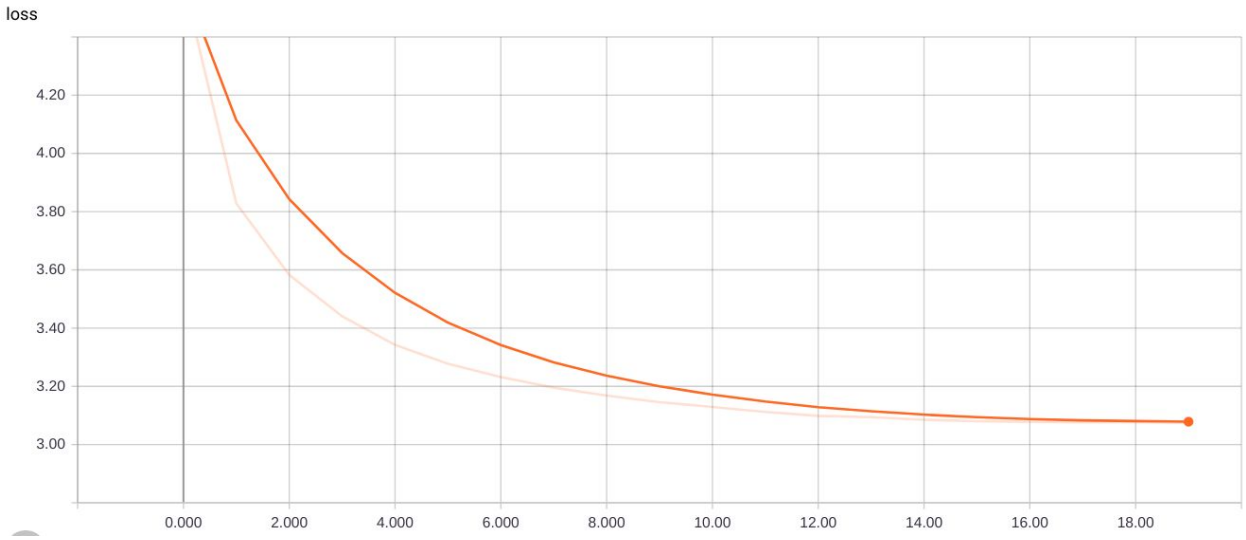
6000/6000 [==============================] - 467s 78ms/step - loss: 3.0562 - val_loss: 3.9463

Epoch 00019: val_loss did not improve from 3.77078

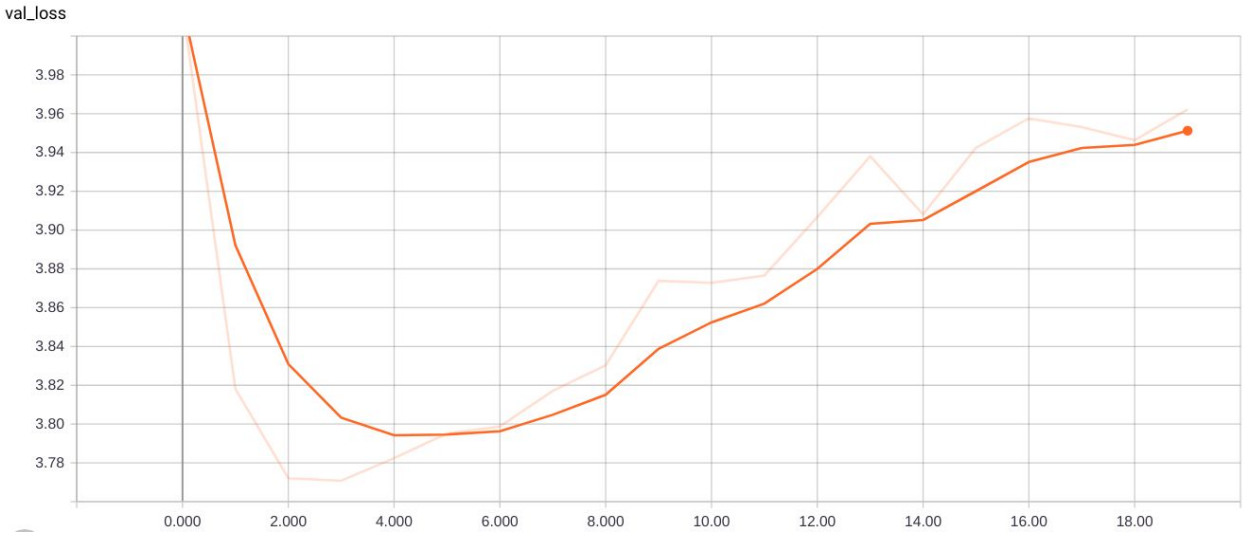
Epoch 20/20

6000/6000 [==============================] - 473s 79ms/step - loss: 3.0543 - val_loss: 3.9622

Epoch 00020: val_loss did not improve from 3.77078



Loss on training dataset



Loss on validation dataset

Model with VGG 16

Epoch 1/20

6000/6000 [==============================] - 527s 88ms/step - loss: 4.6468 - val_loss: 4.1110

Epoch 00001: val_loss improved from inf to 4.11097, saving model to
model-ep001-loss4.667-val_loss4.111.h5

Epoch 2/20

6000/6000 [==============================] - 517s 86ms/step - loss: 3.8972 - val_loss: 3.8974

Epoch 00002: val_loss improved from 4.11097 to 3.89738, saving model to
model-ep002-loss3.917-val_loss3.897.h5

Epoch 3/20

6000/6000 [==============================] - 513s 86ms/step - loss: 3.6494 - val_loss: 3.8377

Epoch 00003: val_loss improved from 3.89738 to 3.83766, saving model to
model-ep003-loss3.669-val_loss3.838.h5

Epoch 4/20

6000/6000 [==============================] - 512s 85ms/step - loss: 3.5079 - val_loss: 3.8230

Epoch 00004: val_loss improved from 3.83766 to 3.82300, saving model to
model-ep004-loss3.528-val_loss3.823.h5

Epoch 5/20

6000/6000 [==============================] - 512s 85ms/step - loss: 3.4139 - val_loss: 3.8069

Epoch 00005: val_loss improved from 3.82300 to 3.80694, saving model to
model-ep005-loss3.434-val_loss3.807.h5

Epoch 6/20

6000/6000 [==============================] - 511s 85ms/step - loss: 3.3470 - val_loss: 3.8188

Epoch 00006: val_loss did not improve from 3.80694

Epoch 7/20

6000/6000 [==============================] - 506s 84ms/step - loss: 3.2992 - val_loss: 3.8311

Epoch 00007: val_loss did not improve from 3.80694

Epoch 8/20

6000/6000 [==============================] - 510s 85ms/step - loss: 3.2607 - val_loss: 3.8433

Epoch 00008: val_loss did not improve from 3.80694

Epoch 9/20

6000/6000 [==============================] - 509s 85ms/step - loss: 3.2261 - val_loss: 3.8651

Epoch 00009: val_loss did not improve from 3.80694

Epoch 10/20

6000/6000 [==============================] - 510s 85ms/step - loss: 3.1998 - val_loss: 3.8504

Epoch 00010: val_loss did not improve from 3.80694

Epoch 11/20

6000/6000 [==============================] - 513s 86ms/step - loss: 3.1790 - val_loss: 3.8783

Epoch 00011: val_loss did not improve from 3.80694

Epoch 12/20

6000/6000 [==============================] - 517s 86ms/step - loss: 3.1597 - val_loss: 3.8938

Epoch 00012: val_loss did not improve from 3.80694

Epoch 13/20

6000/6000 [==============================] - 514s 86ms/step - loss: 3.1441 - val_loss: 3.8947

Epoch 00013: val_loss did not improve from 3.80694

Epoch 14/20

6000/6000 [==============================] - 515s 86ms/step - loss: 3.1305 - val_loss: 3.9083

Epoch 00014: val_loss did not improve from 3.80694

Epoch 15/20

6000/6000 [==============================] - 512s 85ms/step - loss: 3.1230 - val_loss: 3.9260

Epoch 00015: val_loss did not improve from 3.80694

Epoch 16/20

6000/6000 [==============================] - 514s 86ms/step - loss: 3.1122 - val_loss: 3.9317

Epoch 00016: val_loss did not improve from 3.80694

Epoch 17/20

6000/6000 [==============================] - 511s 85ms/step - loss: 3.1054 - val_loss: 3.9361

Epoch 00017: val_loss did not improve from 3.80694

Epoch 18/20

6000/6000 [==============================] - 517s 86ms/step - loss: 3.0989 - val_loss: 3.9453

Epoch 00018: val_loss did not improve from 3.80694

Epoch 19/20

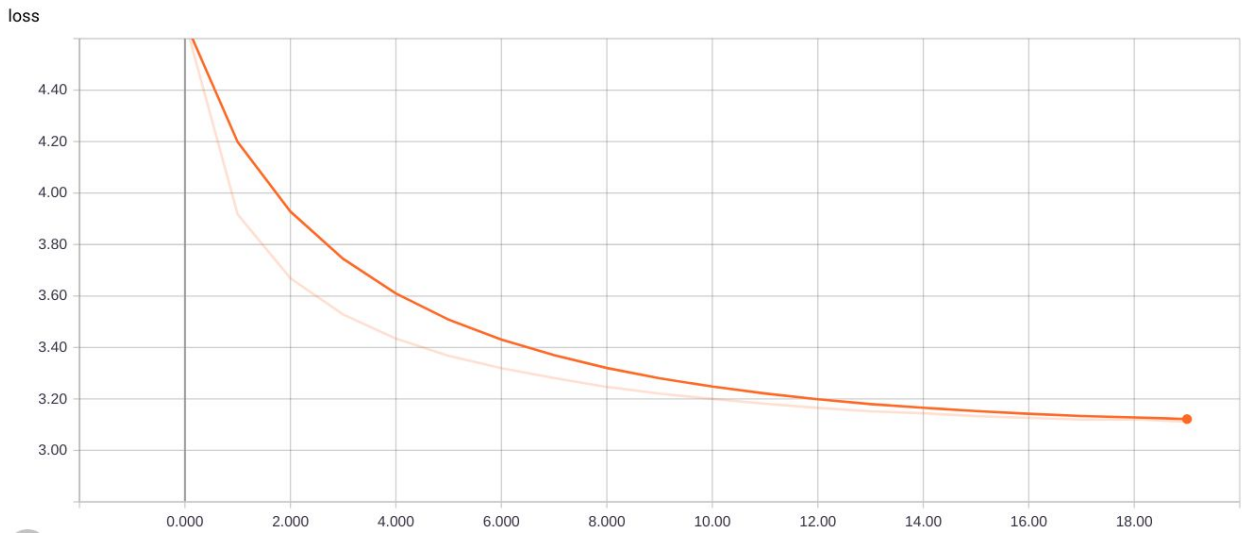
6000/6000 [==============================] - 517s 86ms/step - loss: 3.0995 - val_loss: 3.9580

Epoch 00019: val_loss did not improve from 3.80694

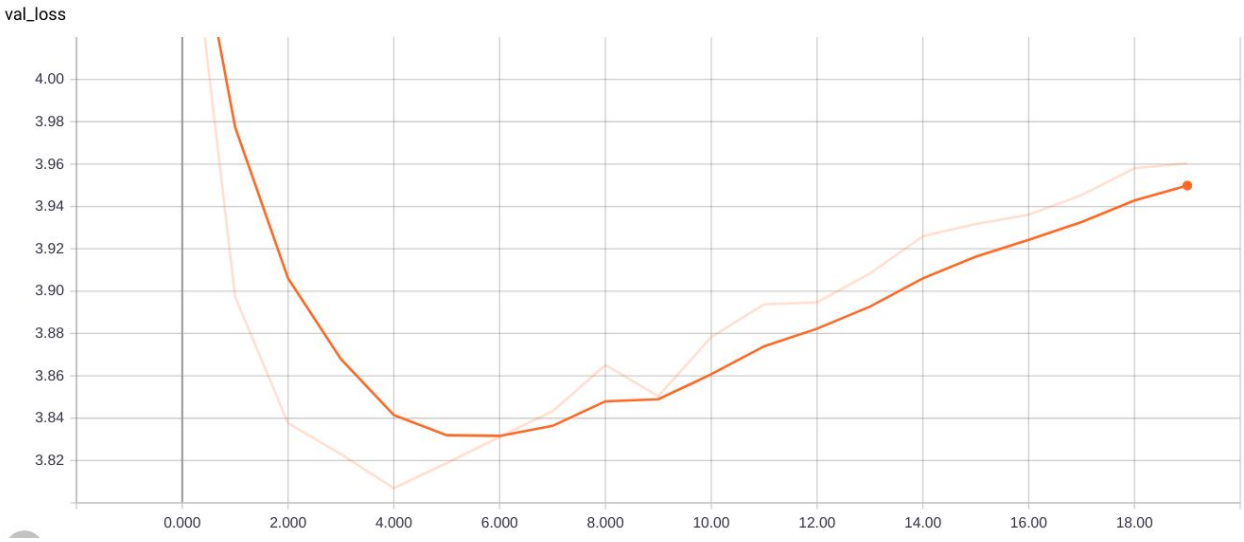
Epoch 20/20

6000/6000 [==============================] - 512s 85ms/step - loss: 3.0905 - val_loss: 3.9604

Epoch 00020: val_loss did not improve from 3.80694



Loss on training dataset



Loss on validation dataset

Results Validation/Testing reports:

InceptionResNetV2 with Fasttext:

Blue scores for tested data:

```
BLEU - 1 : 0.548384
BLEU - 2 : 0.306833
BLEU - 3 : 0.213625
BLEU - 4 : 0.103650
```

Blue scores for validation data:

```
BLEU - 1 : 0.539671
BLEU - 2 : 0.293452
BLEU - 3 : 0.203698
BLEU - 4 : 0.096276
```

Rouge scores for tested data:

Max score:

```
rouge-1 f  -->  0.4624803486039034
rouge-1 p  -->  0.48812131699999306
rouge-1 r  -->  0.4860838679852995
rouge-2 f  -->  0.14525472241680532
```

```
rouge-2 p  -->  0.1512867924516145
rouge-2 r  -->  0.14984904297915155
rouge-l f  -->  0.43118846871737243
rouge-l p  -->  0.4646600222489923
rouge-l r  -->  0.4724997299011614
```

Min score:

```
rouge-1 f  -->  0.25518982933884565
rouge-1 p  -->  0.26721383698001505
rouge-1 r  -->  0.24672515610035056
rouge-2 f  -->  0.009320750473997289
rouge-2 p  -->  0.01086065780624604
rouge-2 r  -->  0.008425656593148846
rouge-l f  -->  0.23491819231621058
rouge-l p  -->  0.2643025787937569
rouge-l r  -->  0.23922741761019706
```

Average score:

```
rouge-1 f  -->  0.3531856663364575
rouge-1 p  -->  0.3692813179794065
rouge-1 r  -->  0.3573926099388518
rouge-2 f  -->  0.06199432651474956
rouge-2 p  -->  0.06627795275673773
rouge-2 r  -->  0.06183210630412235
rouge-l f  -->  0.3264658434446139
rouge-l p  -->  0.3568438974913978
rouge-l r  -->  0.34589343429366254
```

Rouge scores for validation data:

Max score:

```
rouge-1 f  -->  0.4525239386952361
rouge-1 p  -->  0.47777440859793757
rouge-1 r  -->  0.4751623340024262
rouge-2 f  -->  0.1345950355753031
rouge-2 p  -->  0.14215377318330116
rouge-2 r  -->  0.13762309286561564
rouge-l f  -->  0.4191775106474262
rouge-l p  -->  0.4541641548810666
rouge-l r  -->  0.4606900377271501
```

Min score:

```
rouge-1 f  -->  0.25012420477380287
rouge-1 p  -->  0.26256574634515967
rouge-1 r  -->  0.24169372036827672
```

```
rouge-2 f  -->  0.008999827112244823
rouge-2 p  -->  0.010498795648795644
rouge-2 r  -->  0.008212813379431019
rouge-l f  -->  0.23044218246640658
rouge-l p  -->  0.25961426865618203
rouge-l r  -->  0.23447792523003572
```

Average score:

```
rouge-1 f  -->  0.34661380933471575
rouge-1 p  -->  0.36272648622619263
rouge-1 r  -->  0.3506524933963398
rouge-2 f  -->  0.05770147264007184
rouge-2 p  -->  0.06196898243397488
rouge-2 r  -->  0.057478130135159344
rouge-l f  -->  0.3201540393262862
rouge-l p  -->  0.3503322122975068
rouge-l r  -->  0.33941558403435806
```

InceptionResNetV2

Blue scores for tested data:

```
BLEU - 1 : 0.422588
BLEU - 2 : 0.235717
BLEU - 3 : 0.163521
BLEU - 4 : 0.072335
```

Blue scores for validation data:

```
BLEU - 1 : 0.411673
BLEU - 2 : 0.223612
BLEU - 3 : 0.153555
BLEU - 4 : 0.068918
```

Rouge scores for tested data:

Max score:

```
rouge-1 f  -->  0.45747949429176243
rouge-1 p  -->  0.502441552891553
rouge-1 r  -->  0.46277664987661127
rouge-2 f  -->  0.14083103642757872
rouge-2 p  -->  0.151733426704015
rouge-2 r  -->  0.14067245569474698
rouge-l f  -->  0.4284166619519872
rouge-l p  -->  0.4824505411255407
```

```
rouge-l r  -->  0.4533763270784944
```

Min score:

```
rouge-1 f  -->  0.2539295772047749
rouge-1 p  -->  0.27618183205683305
rouge-1 r  -->  0.23733214676190784
rouge-2 f  -->  0.009213488185270723
rouge-2 p  -->  0.0107762099012099
rouge-2 r  -->  0.008377517335605567
rouge-l f  -->  0.2342523216849833
rouge-l p  -->  0.2740855616605626
rouge-l r  -->  0.23079599718736796
```

Average score:

```
rouge-1 f  -->  0.34999258816208384
rouge-1 p  -->  0.3814604390054395
rouge-1 r  -->  0.34061260551133254
rouge-2 f  -->  0.05944953299019661
rouge-2 p  -->  0.06550353654515431
rouge-2 r  -->  0.05773433717640173
rouge-l f  -->  0.3240548844009533
rouge-l p  -->  0.37031046176046195
rouge-l r  -->  0.33137660264566315
```

Rouge scores for validation data:

Max score:

```
rouge-1 f  -->  0.4507953383443884
rouge-1 p  -->  0.4993066544566543 4
rouge-1 r  -->  0.45273262374020845
rouge-2 f  -->  0.13173527420722472
rouge-2 p  -->  0.14511459651459668
rouge-2 r  -->  0.12959921084591397
rouge-l f  -->  0.41737407390287024
rouge-l p  -->  0.4766755994005992
rouge-l r  -->  0.4398182340578233
```

Min score:

```
rouge-1 f  -->  0.24685129083910654
rouge-1 p  -->  0.2712100732600741
rouge-1 r  -->  0.22997491238265336
rouge-2 f  -->  0.008589291104826022
rouge-2 p  -->  0.010078843378843376
```

```
rouge-2 r  -->  0.007697336734734427
rouge-l f  -->  0.22767304535339938
rouge-l p  -->  0.26961912254412335
rouge-l r  -->  0.2246585914916421
```

Average score:

```
rouge-1 f  -->  0.34430631641581233
rouge-1 p  -->  0.37648937950937983
rouge-1 r  -->  0.33455698520041094
rouge-2 f  -->  0.05649370779358377
rouge-2 p  -->  0.06305845987345998
rouge-2 r  -->  0.05422343813967629
rouge-l f  -->  0.31814060216207124
rouge-l p  -->  0.36516824120324104
rouge-l r  -->  0.32513230811966587
```

VGG16 with Fasttext:

Blue scores for tested data:

```
BLEU - 1 : 0.527234
BLEU - 2 : 0.276132
BLEU - 3 : 0.186770
BLEU - 4 : 0.082055
```

Blue scores for validation data:

```
BLEU - 1 : 0.527834
BLEU - 2 : 0.277858
BLEU - 3 : 0.193462
BLEU - 4 : 0.091846
```

Rouge scores for tested data:

Max Scores:

```
rouge-1 f  -->  0.4426602528084306
rouge-1 p  -->  0.46779059829059805
rouge-1 r  -->  0.4634395481969009
rouge-2 f  -->  0.1206922535786308
rouge-2 p  -->  0.12610330919080964
rouge-2 r  -->  0.1239340382184038
rouge-l f  -->  0.4132719534206428
rouge-l p  -->  0.4453715950715948
rouge-l r  -->  0.4519066139742611
```

Min scores:

```
rouge-1 f  -->  0.24971020151598589
rouge-1 p  -->  0.2603996003996016
rouge-1 r  -->  0.241354437650414
rouge-2 f  -->  0.007089532801356564
rouge-2 p  -->  0.008019516594516591
rouge-2 r  -->  0.00659908645276292
rouge-l f  -->  0.2329040922907665
rouge-l p  -->  0.2587826479076491
rouge-l r  -->  0.23504234657671294
```

Average scores:

```
rouge-1 f  -->  0.34186128957310297
rouge-1 p  -->  0.35723604978354967
rouge-1 r  -->  0.3442156001526276
rouge-2 f  -->  0.05001138269574386
rouge-2 p  -->  0.05377725219225224
rouge-2 r  -->  0.04971431600874254
rouge-l f  -->  0.3179600539904673
rouge-l p  -->  0.3462107484182484
rouge-l r  -->  0.3341856721529246
```

Rouge scores for validation data:

Max scores:

```
rouge-1 f  -->  0.4446849065594658
rouge-1 p  -->  0.46906860084359986
rouge-1 r  -->  0.46618459341226337
rouge-2 f  -->  0.12385793497163533
rouge-2 p  -->  0.1299280921692689
rouge-2 r  -->  0.1272835167268757
rouge-l f  -->  0.4141315760723 27
rouge-l p  -->  0.4471779151404147
rouge-l r  -->  0.45331800895612795
```

Min scores:

```
rouge-1 f  -->  0.2505637342668432
rouge-1 p  -->  0.2589467282717295
rouge-1 r  -->  0.24350730343553637
rouge-2 f  -->  0.007349858208002882
rouge-2 p  -->  0.008445787545787547
rouge-2 r  -->  0.006793431024599754
```

```
rouge-l f  -->  0.23310905062796825
rouge-l p  -->  0.25713376900877016
rouge-l r  -->  0.2364687502139315
```

Average scores:

```
rouge-1 f  -->  0.3416217865077551
rouge-1 p  -->  0.3551571520146523
rouge-1 r  -->  0.34563156384605154
rouge-2 f  -->  0.051845529088174164
rouge-2 p  -->  0.055391025134995824
rouge-2 r  -->  0.051879556258090785
rouge-l f  -->  0.3177027941636258
rouge-l p  -->  0.34403887751137724
rouge-l r  -->  0.3353722509060513
```

VGG16:

Blue scores for tested data:

```
BLEU - 1 : 0.527234
BLEU - 2 : 0.276132
BLEU - 3 : 0.186770
BLEU - 4 : 0.082055
```

Blue scores for validation data:

```
BLEU - 1 : 0.483710
BLEU - 2 : 0.244437
BLEU - 3 : 0.164638
BLEU - 4 : 0.073413
```

Rouge scores for tested data:

Max scores:

```
rouge-1 f  -->  0.4439370756176144
rouge-1 p  -->  0.4615903774330249
rouge-1 r  -->  0.4787691352275177
rouge-2 f  -->  0.12503257795502282
rouge-2 p  -->  0.12980489828798672
rouge-2 r  -->  0.13104469692244505
rouge-l f  -->  0.4095245274634008
rouge-l p  -->  0.4372951713319362
rouge-l r  -->  0.46472994268803114
```

Min scores:

```
rouge-1 f  -->  0.2481328108332503
```

```
rouge-1 p  -->  0.2539601354527836
rouge-1 r  -->  0.24883256260853148
rouge-2 f  -->  0.007111116280866235
rouge-2 p  -->  0.008272118241235886
rouge-2 r  -->  0.006580954584631051
rouge-l f  -->  0.2282690334742405
rouge-l p  -->  0.2523844670035858
rouge-l r  -->  0.2416371181407184
```

Average scores:

```
rouge-1 f  -->  0.34060673336420466
rouge-1 p  -->  0.34846123908770993
rouge-1 r  -->  0.3552434513442063
rouge-2 f  -->  0.05251160329159693
rouge-2 p  -->  0.05584605123634561
rouge-2 r  -->  0.053380248259057336
rouge-l f  -->  0.31305487384593816
rouge-l p  -->  0.3370844273373691
rouge-l r  -->  0.34373448545308527
```

Rouge scores for validation data:

Max scores:

```
rouge-1 f  -->  0.43200569606111705
rouge-1 p  -->  0.4486153718830186
rouge-1 r  -->  0.46583581156545506
rouge-2 f  -->  0.1166146784301684
rouge-2 p  -->  0.12182633053221298
rouge-2 r  -->  0.12203029086628633
rouge-l f  -->  0.39824697356553385
rouge-l p  -->  0.425989745385333
rouge-l r  -->  0.4515220364559759
```

Min scores:

```
rouge-1 f  -->  0.2434037527881582
rouge-1 p  -->  0.24640775580628654
rouge-1 r  -->  0.24588343679080935
rouge-2 f  -->  0.007154386901722444
rouge-2 p  -->  0.008200511906394264
rouge-2 r  -->  0.0067608955331397765
rouge-l f  -->  0.2235753714969846
rouge-l p  -->  0.2444421968554335
rouge-l r  -->  0.2382473776386898
```

Average scores:

```
rouge-1 f  -->  0.33361885642784045
rouge-1 p  -->  0.33957737115825387
rouge-1 r  -->  0.34978893447038145
rouge-2 f  -->  0.04873904478543653
rouge-2 p  -->  0.05172667541608738
rouge-2 r  -->  0.04970483488560209
rouge-l f  -->  0.3063116741922885
rouge-l p  -->  0.32810452524599615
rouge-l r  -->  0.33815837542468047
```

Future Scope:

Sharper attention: From the result we notice that the attention coefficient are evenly distributed, which means that the model takes the whole picture information to generate the next time step hidden layer via LSTM. But we expect that we can highlight specific part of the picture related to the certain word. To achieve this goal we can use hard attention, which restricts information extraction from image as whole. We can also use a harper activation function instead of softmax to produce a suitable attention distribution. Moreover, we can label more detailed captions to force the mode to attend smaller parts.

Language model: Since the model will produce a probability distribution of the vocabulary on every time step, we can use language model to generate natural sentences based on these vocabulary probability distributions. Furthermore, we can build a markov random field like hidden markov model on the top of the the softmax output layer. We can try different architecture and especially layer of CNN encoder to get a better feature map level.

Video Captioning: We can extend our model to do multiple task such as video captioning. As video data increases, there has been a recent surge of interest in automatic video content analysis. Furthermore, technological advancement in computer vision, natural language processing, and machine learning has resulted in an increase of interest in complex intelligence problems relating to the simultaneous understanding of natural language and video clips. Video-based complex intelligence problems typically include video captioning and video question answering. Video captioning refers to the task of generating a natural language sentence that explains the content of the input video clip.

Visual Question Answering: Visual Question Answering is a research area about building a computer system to answer questions presented in an image and a natural language.

Using Bigger Dataset: We can use Flickr 30k dataset and other various datasets on better hardware. This project was done on Google Colab which was insufficient for bigger datasets.

References:

1. Show and Tell: A Neural Image Caption Generator, 2015.
2. Show, Attend and Tell: Neural Image Caption Generation with Visual Attention, 2015.
3. Where to put the Image in an Image Caption Generator, 2017.
4. What is the Role of Recurrent Neural Networks (RNNs) in an Image Caption Generator?, 2017.
5. Automatic Description Generation from Images: A Survey of Models, Datasets, and Evaluation Measures, 2016.