Neural Network Minor Project Guideline- Dr. Anuja Arora, Associate Professor Neural Network Minor Project Guideline Phase-IIII Final Evaluation Phase Report Format and Submission Guidelines Evaluation Date: 6 th May 2019 Marks: 50 You are required to submit your Final Project Report for Neural Network Workshop Minor Project to concerned teacher in your respective labs. Please adhere to the deadlines mentioned. Following Content has to be submitted in final submission of minor project viva. A. Demo of project and implementation code in CD [35 Marks] B. Neatly typed A4 size hard copy of project report which includes [15 Marks]

1. Abstract 2. Clear division of project work among group members 3. Problem Statement/ Research Objective 4. Research Paper studied 5. Research Paper Integration

6. Algorithms/ Approach used and Implemented

7. Results 8. Results Validation/ Testing reports 9. Future Scope 10. References: All research papers detail in IEEE format

=========================================================================

Work distribution - 4 models hain, sab kuch 4 hai, app mei button bhi 4 hain, and 4 log hain, sabne ek ek ki bakchodi kari hai.

Algorithms/ Approach used and Implemented

## **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**

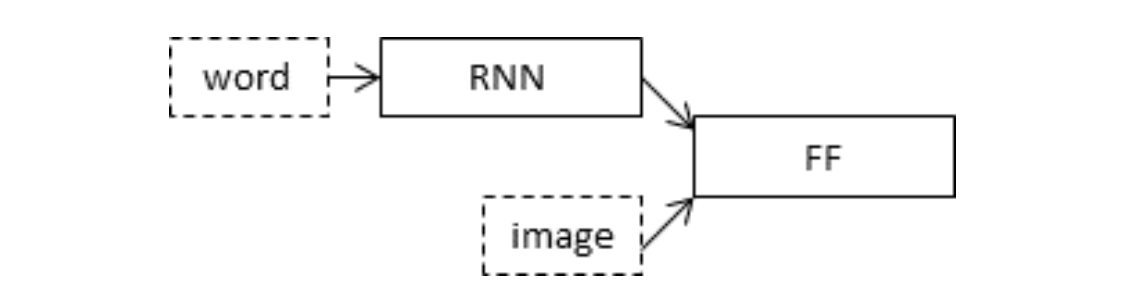
We will define a deep learning based on the “*merge-model*” described by Marc Tanti, et al. in their 2017 papers:

* [Where to put the Image in an Image Caption Generator](https://arxiv.org/abs/1703.09137), 2017.
* [What is the Role of Recurrent Neural Networks (RNNs) in an Image Caption Generator?](https://arxiv.org/abs/1708.02043), 2017.

For a gentle introduction to this architecture, see the post:

* [Caption Generation with the Inject and Merge Architectures for the Encoder-Decoder Model](https://machinelearningmastery.com/caption-generation-inject-merge-architectures-encoder-decoder-model/)

The authors provide a nice schematic of the model, reproduced below.



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.

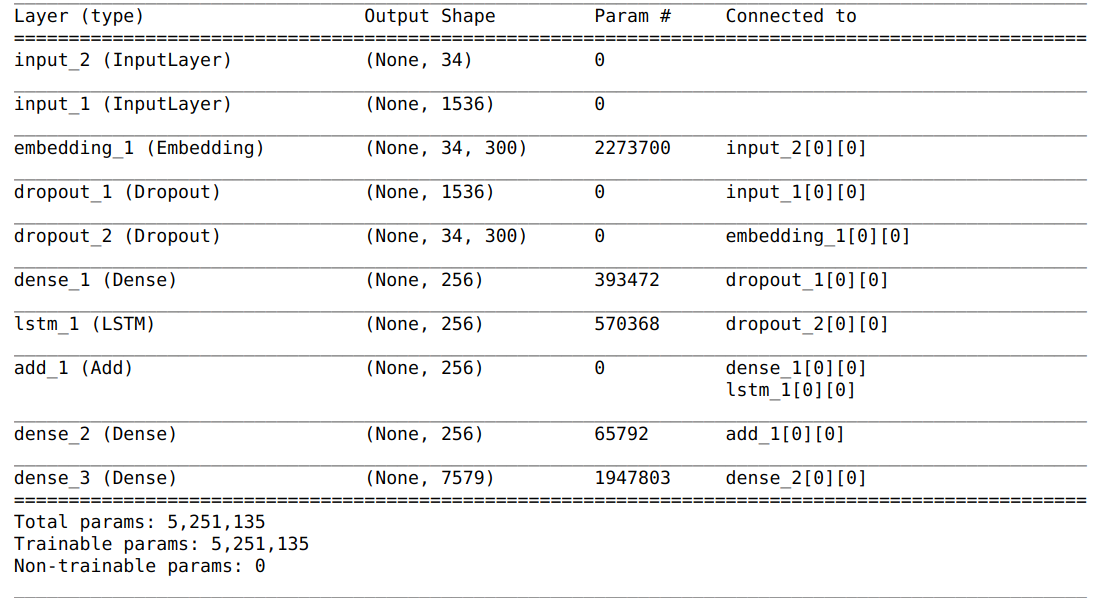
The Sequence Processor model expects input sequences with a pre-defined length (34 words) which are fed into an Embedding layer that uses a mask to ignore padded values. This is followed by an LSTM layer with 256 memory units.

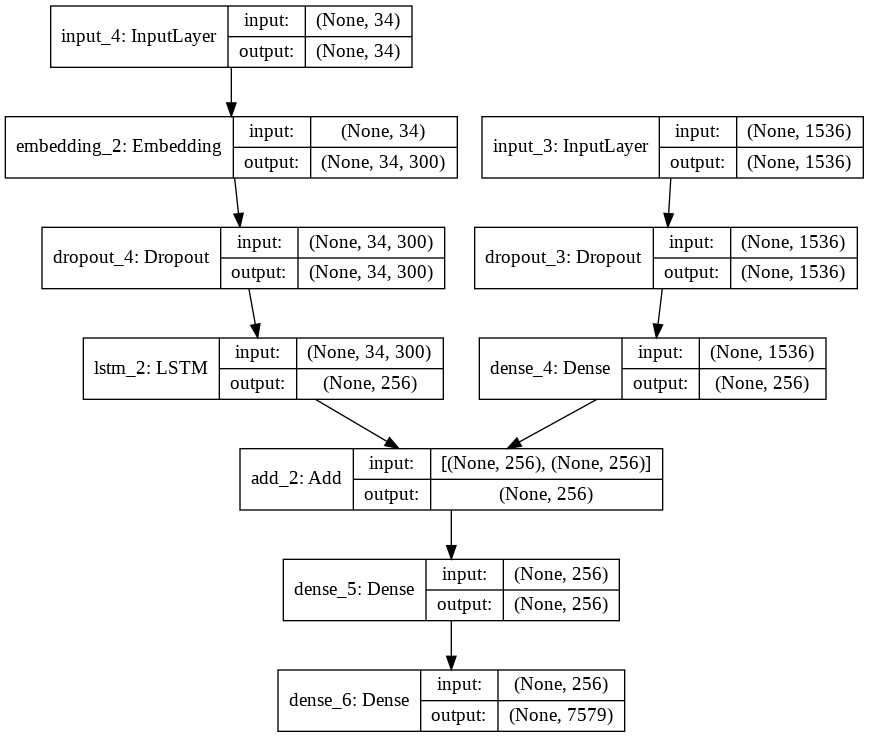
Both the input models produce a 256 element vector. Further, both input models use regularization in the form of 50% dropout. This is to reduce overfitting the training dataset, as this model configuration learns very fast.

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.

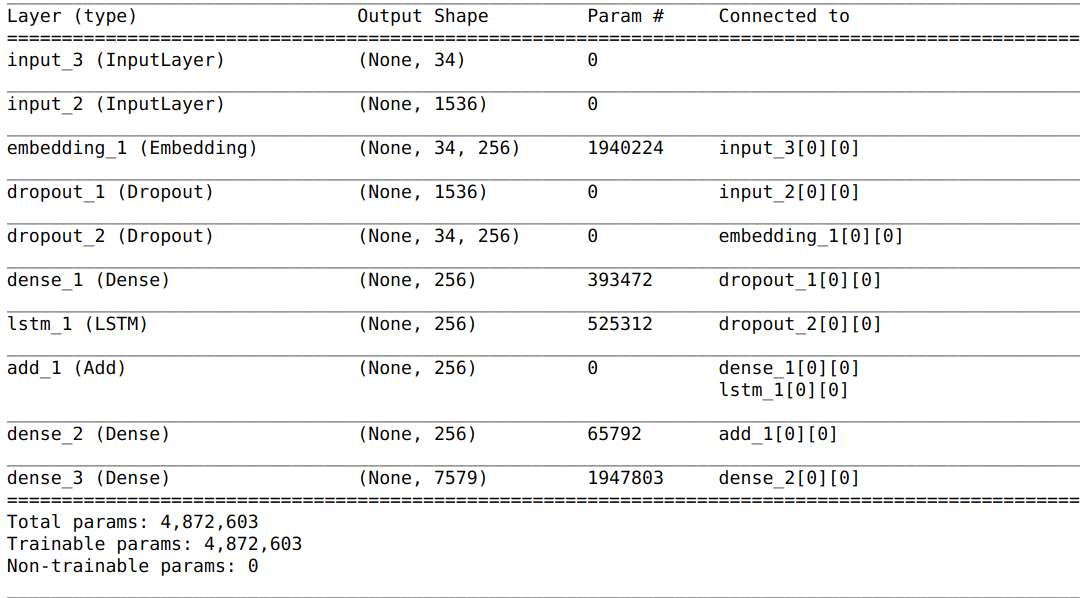
Summary of 4 models and pics of 4 models\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

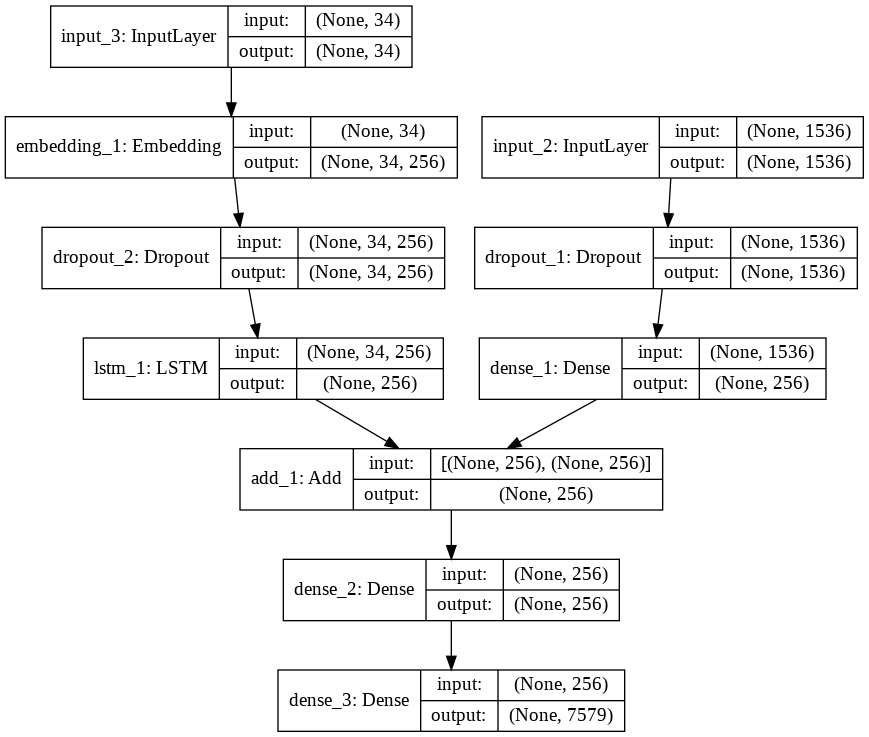
**Model with Inception ResNet V2 and FastText**



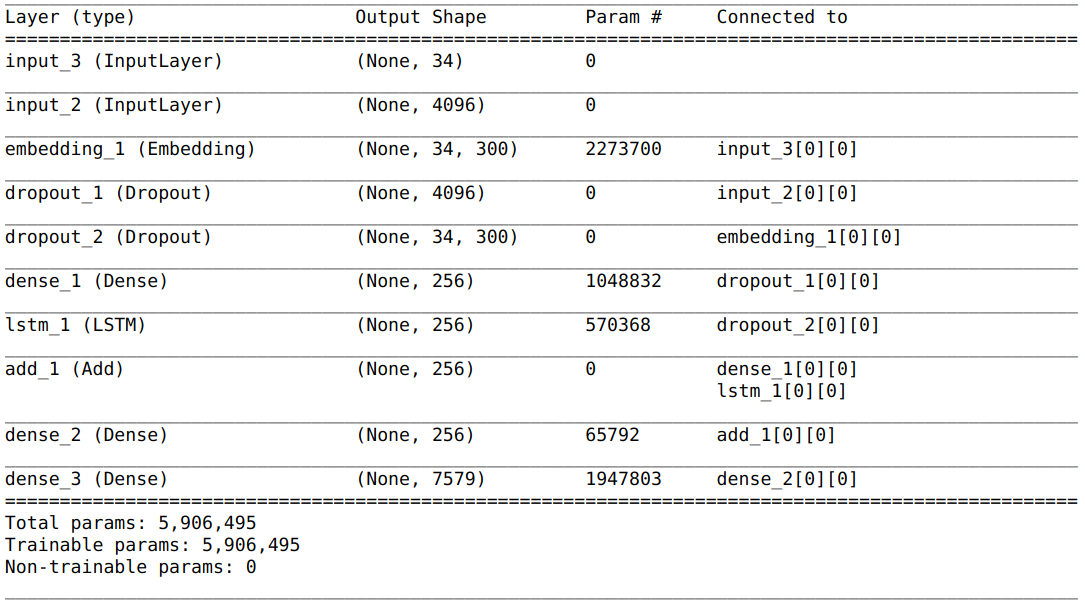


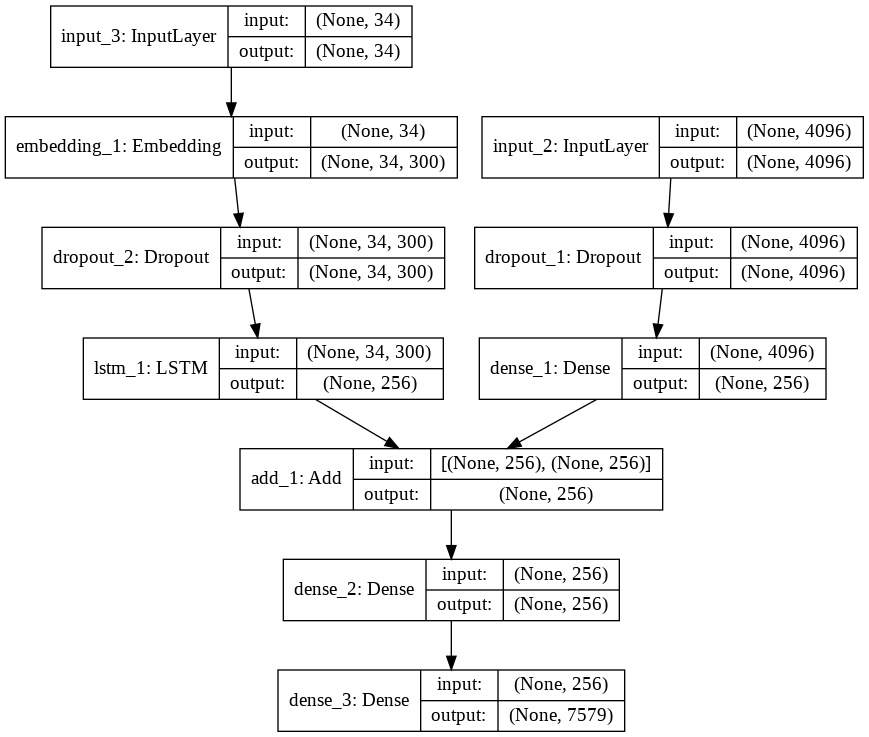
**Model with Inception ResNet**

****

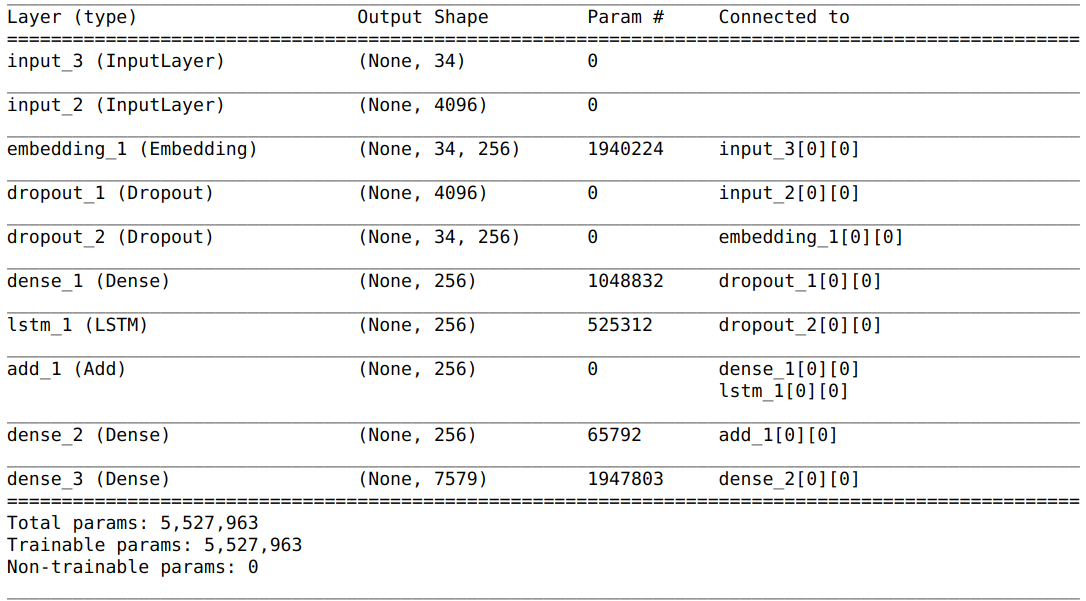
****

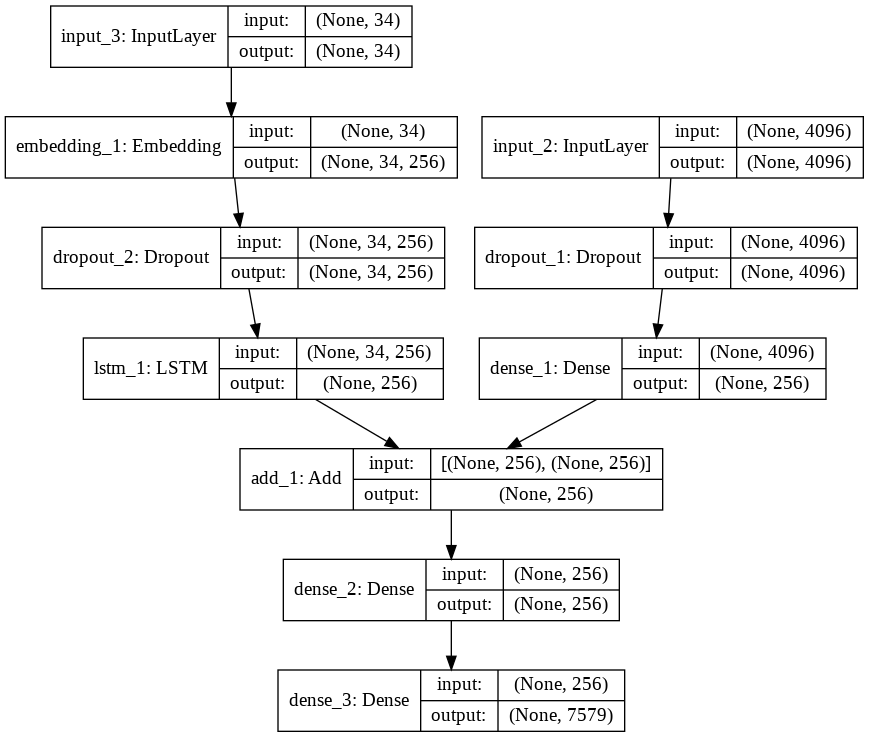
**Model with VGG 16 and FastText**

****

****

**Model with VGG 16**

****

****

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 to only one reference so we compared the predicted caption with all of our references

EPOCHS

**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

**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

**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

**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

Results

Bleu

Gleu

Rouge

Tensorboard graphs - epochs...initial and final loss….

Future

Train on flickr 30k