# Capstone-1 Final Report

# Image caption generator

1. **Introduction**
2. **Dataset Description**
3. **Exploratory Data Analysis**
4. **Data Preprocessing**
5. **Model Building and Training**
6. **Inference**
7. **Future Work**
8. **Applications**
9. **Conclusion**

## Introduction:

As humans, generating descriptions or suitable captions for an image comes almost naturally to us. However, when dig deep and retrospect this 'second nature' of ours, we can almost attribute this ease to the past experiences and learnings in our lives. Then comes the interesting question of how we can make a computer or a machine replicate this behavior of ours. Obviously, the poor little machine doesn't share our experiences or education or even consciousness to do the same.

This exact problem was deemed to be one of the greatest challenges that computer researchers were facing. But with the recent widescale adaptation of neural networks and deep learning in general, this problem is tackled to such an extent where the computer-generated captions/descriptions turn out to be more accurate than the ones generated by their human counter parts. This very shift in paradigm interested me the most and intrigued me to get my hands dirty to develop a image caption generator of my own. Because of the obvious facts such as limited access to large datasets and huge computational power, I used a small dataset and a pre-trained model (transfer learning) and fine-tuned enough for some satisfactory and interesting results.

## Dataset Description

For this project, we use the famous Flicker 8k dataset. This dataset contains 8000 images (hence the name 8k) and each image has 5 captions telling us what is happening in the image. These 8000 images are split as follows:

1. 6000 Training images and their descriptions.
2. 1000 Development/Validation images and their descriptions.
3. 1000 Test images and their descriptions

## EDA

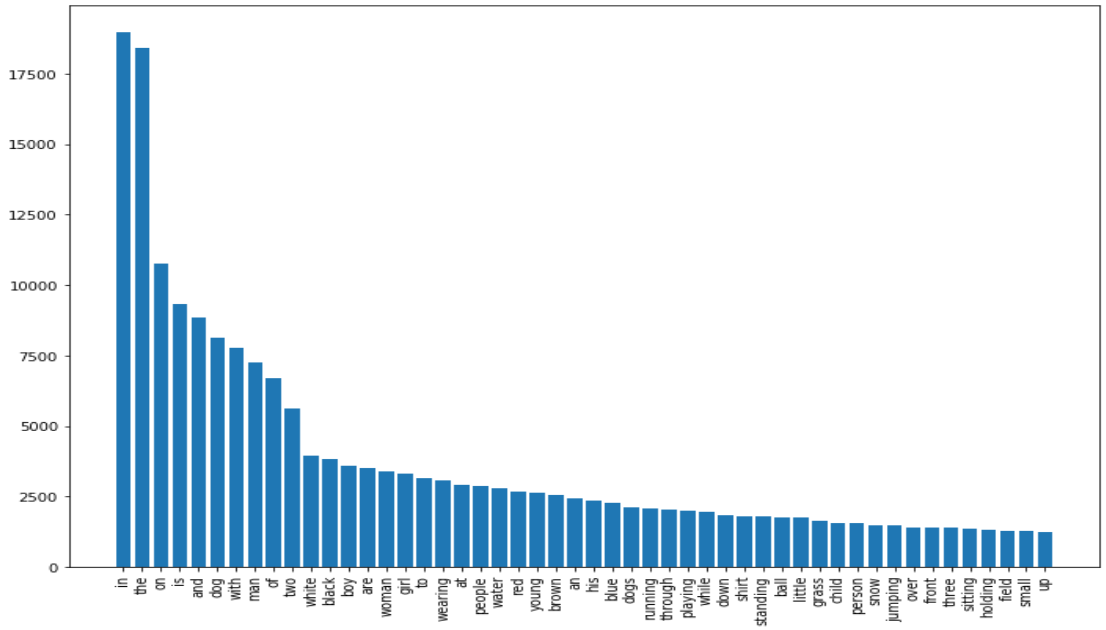
Since the dataset is only consists of images and text, there is not much scope of exploratory data analysis like the one done on numerical data.
To get an idea of how images and their captions are set up in the dataset, below are few examples:

['helmeted man jumping off rock on mountain bike', 'man jumping on his bmx with another bmx er watching', 'mountain biker is jumping his bike over rock as another cyclist stands on the trail watching', 'person taking jump off rock on dirt bike', 'the bike rider jumps off rock']

['girl with pink headband and blue shirt is drumming on blue plastic and metal bowls with chops ticks', 'little asian girl banging on pans', 'young chinese girl playing the drums with two chopstick s pot and blue plastic bowl', 'young girl with pink headband hits dish and pan with chopsticks', 't he child is using chopsticks on overturned kitchen items']

['man is carrying surfboard along stone wall that leads to the ocean', 'man in green swim trunks carrying surfboard along rock wall to the ocean', 'surfers walking along seawall as the ocean chu rns around them', 'two surfers carry their boards to the end of walkway to join the others in the water', 'two surfers walk along rock wall to reach the waves']

We can see that the dataset has wide variety of images like animals, sports, humans, adventures etc. Next, let us see how the distribution of words is across the corpus. To do that, we are going to plot the histogram of the top 50 frequently repeating words across the corpus.



We can see that most of the frequent words are common English language words and it is logical that they are the ones with highest count since they ought to appear in almost any properly constructed English sentence.

## Data Preprocessing:

This section is divided into three main parts:

1. Image data preprocessing
2. Text data preprocessing
3. Creating input-output pairs

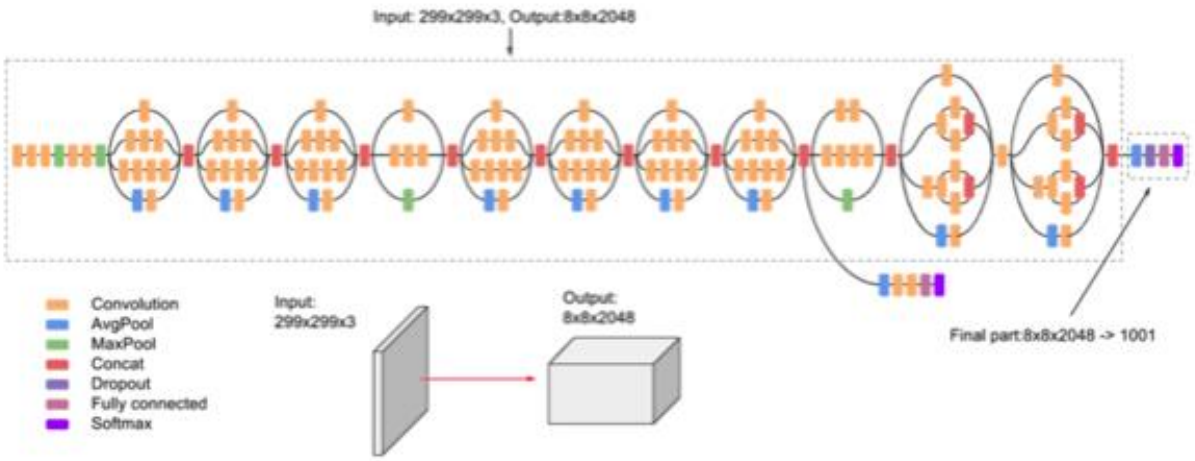
### 1. Image data preprocessing:

In the world of deep learning, CNNs (convolutional neural networks) have almost dominated in various tasks such as classification, object detection, localization, semantic segmentation (where the pixels corresponding to a particular subject are colored or highlighted). They've proven to yield results with very high precision.

In our project, we implement similar technique of using a CNN to extract high level features from the images. At this point, there are two options in front of us: a) to develop a custom CNN trained on all the images that we have and then use an intermediate layer to extract high level features. b) to use pre-trained models readily available out in the internet, which have been trained on much larger and diverse image datasets.

I chose to go with second option because pre trained models can produce embeddings that are better than the embeddings produced by a custom CNN. This is because the pre trained model has seen wide variety of images. Imagine the embeddings produced by CNN to be points in a space. This space is defined by the training data. For a custom CNN this space is limited to only the training data that we have. Pre trained model on the other hand has been trained on millions of images, this space encodes the information of all those images and hence the embedding produced by a pre trained model is more generalized.

Out of the wide variety of state-of-the-art CNN models, I chose to go with InceptionV3 (version 3) network. Below is the architecture of the model:




This model is developed by google and is the first runner up of image net competition in 2015. It has 42 layers. This model has been trained on large dataset of 1.2 million images belonging to 1000 categories. More about the inceptionV3 network can be found here.

As shown in the figure above, if we remove the last layer (which a 1000 unit dense layer with softmax activation to predict the class to which an image belongs) and pass in an image, we get a 2048 dimensional image (this is the no. of units in the Dense layer). This 2048-dimensional vector can be considered as a very high-level representation of an image which might not make sense to us. This vector (same as embedding) is point in the space as discussed above.

We pass all our images through the network and save these high level representations for further use so that this process is not repeated every time.

## 2. Text data preprocessing:

There are several preprocessing steps involved with the text data:

a) Firstly, we read the text file with all that has all the image names and their corresponding list of descriptions, clean all the descriptions by removing punctuations, add special tokens 'startseq' and 'endseq' at the start and end of each sentence (this is done to ensure that models knows when a sentence starts and when it ends) and store them on a single text file with each image name and its corresponding descriptions.
b) Then we read the file containing image names for train, dev (validation) and test sets. We then make three different dictionaries of image id to list of descriptions from the cleaned descriptions for train, dev and test sets respectively. We also do the similar process on images but on the extracted high-level representations.
c) Next, we replace all the tokens (words) in each sentence with a unique integer id using a mapping dictionary that maps each word to unique integer. This dictionary is formed using only the training data. All the words in dev and test set that are not present in training sentences are replaced with a <UNK> (unknown) token.

d) We take maximum sentence length sentence out of all the sentences present in corpus (from train, dev and test) and pad rest of the sequences (sentences converted into sequence of numbers) with zeros. This is done to ensure that every time, our model sees sequences of equal length.

3. **Creating input-output pairs:**

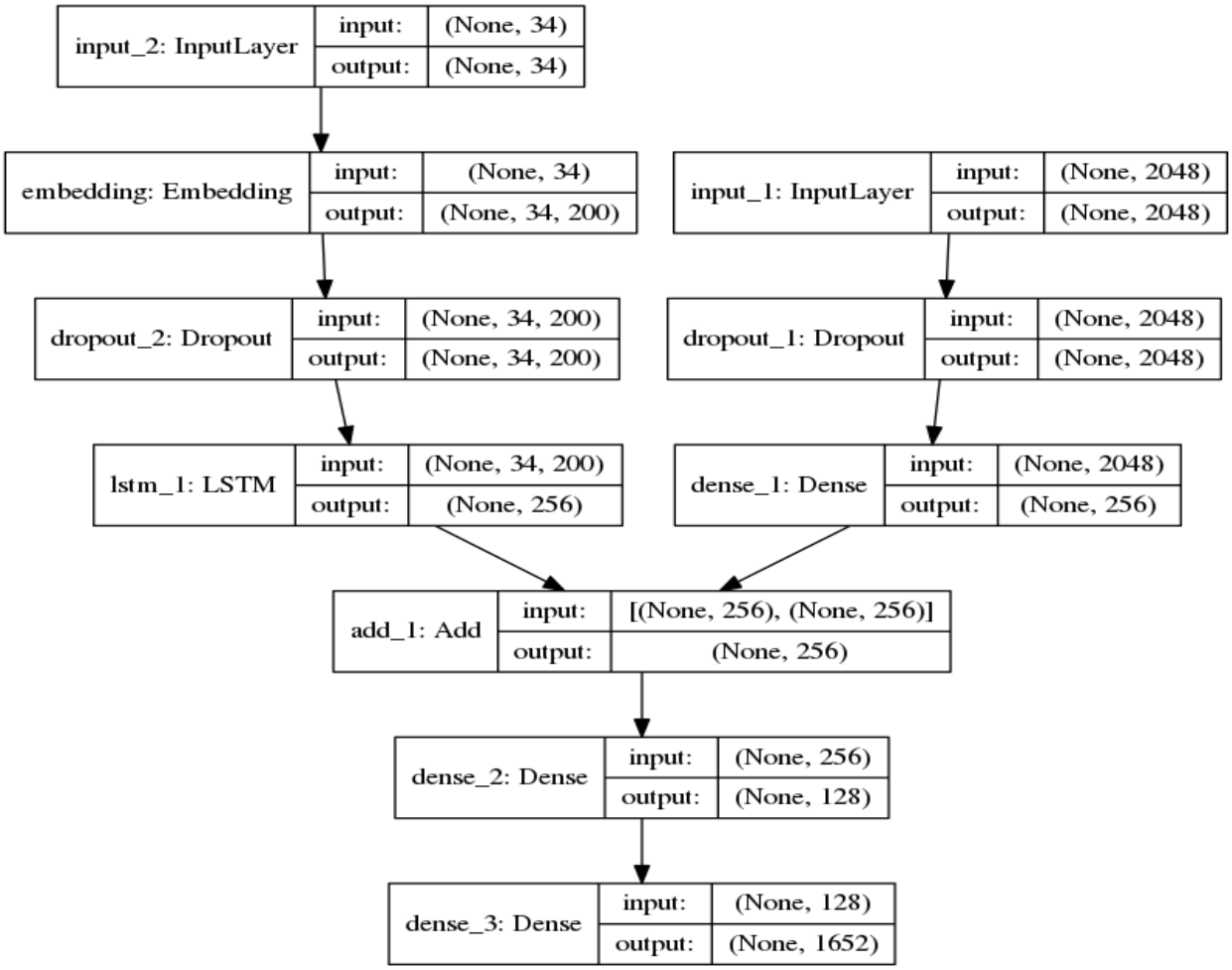
The basic idea behind creating these pairs is that the model will be provided one word and the photo and generate the next word. Then the first two words of the description will be provided to the model as input with the image to generate the next word. This is how the model will be trained. For example, a description 'little girl running on the field' is would be split into the following input output pairs:

| X1, | X2 (text sequence), | y (word) |
|---|---|---|
| photo | startseq, | little |
| photo | startseq, little, | girl |
| photo | startseq, little, girl, | running |
| photo | startseq, little, girl, running, | on |
| photo | startseq, little, girl, running, on, | the |
| photo | startseq, little, girl, running, on, the, | field |
| ploto | startseq, little, girl, running, on, the, field | endseq |

## Building and Training the Deep Learning model

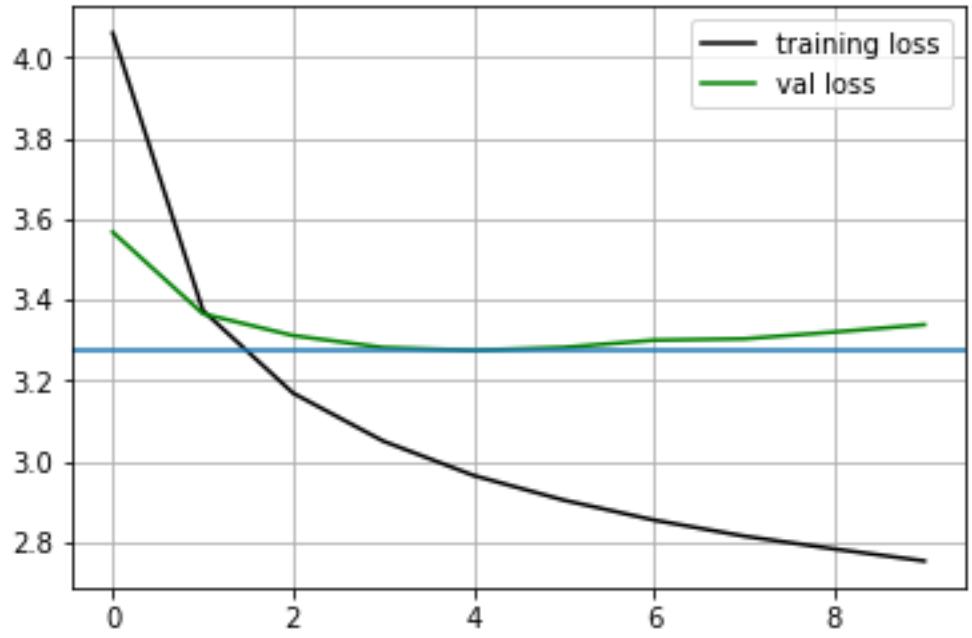
After detailed and time-consuming data preprocessing steps, we are finally ready to build the neural network which is the magical component in this project. Our network is a multimodal neural network where the two modes are two types of input (image and text) sent into the network at the same time.

before we train our network let's step back a little bit and think about our text data. Word embeddings have always proven to perform better in the tasks of natural language processing. Here we are left through options like above, training a custom embedding layer or using pre trained word embeddings. We chose to go with the second option for the same obvious reasons, pre trained word embeddings have been trained with millions of words that are currently not present in our text corpus. So using this embeddings will represent the words in a better and generalized manner. In this project, we use GloVe word embeddings. Each word is represented by 200-dimensional vector. We extract the embeddings only for those words that are present in our text corpus and make them do with the weights for our embedding layer, also we make the weights of the embedding layer non trainable so that they don't get updated during backpropagation. Below is the architecture of the model:




Here, we can see that there are two inputs into the network i.e. text sequence and image feature vector extracted using pre trained model. An LSTM (long short-term memory) layer is used to take care of the text sequences. LSTM is a special kind of RNN (recurrent neural network) that is used to handle sequential data. Here sequences are our text sequences converted into word embeddings i.e. each unique integer in the text sequence is replaced by a 200-dimensional word embedding. LSTM takes in each word embedding at a time for all the words present in a sentence and finally gives out a 256-dimensional vector representing all those words in the sentence. Parallelly, we also down-size the 2048 vector to 256 dimensions using a Dense layer. We fuse both 256 vectors (from the two input modes) by adding them resulting in a single 256-dimensional vector which is further used to predict the next word in the sentence using a Dense layer with number of units equal to the vocabulary length (no. of unique words in the corpus) and the activation function of this layer is SoftMax. SoftMax ensures that the sum of probabilities of all the words given by the output layer sum up to 1.

After 10 epochs and a train time of 2.5 hours we get our Keras callbacks to save the best model at $5^{th}$ epoch. Below is the training and validation loss plotted against number of epochs. We can see that after $5^{th}$ epoch, even though the training loss goes down, the validation loss starts increasing which might indicate that the model could possibly start overfitting the data.




The best model has training and validation losses to be 2.9659 and 3.2751 respectively.

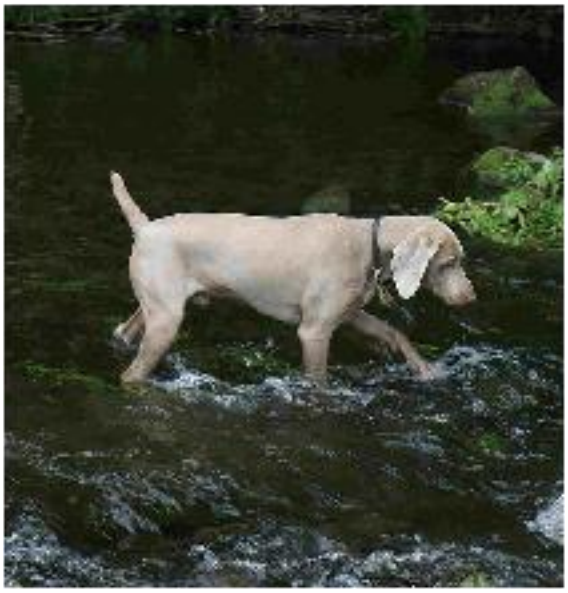
## Inference

Now that our deep learning model is trained and ready, we can use it for inference. But the inference is not so straight forward as calling “model.predict()” like we normally do in Keras. Below are the steps performed to get meaningful predictions from the network.

1. We first start with the initial seed which is ‘startseq’ indicating the start of the sentence and make it the input sentence.
2. Next, we convert this input sentence which is in text to input sequence (by replacing each word in input sentence with unique integer by using the same mapping dictionary used in training to maintain consistency).
3. Then we pad this input sequence with zeros to match the length of longest sequence in our training data.
4. Now, we send in this input sequence padded with zeros along with the feature vector of image and predict the next number in the sequence.
5. The predicted number is converted to word using a reverse mapping dictionary that maps unique integers to their respective words. This word is then appended to the input sentence.
6. Steps 2 to 5 are repeated until the network predicts the ‘endseq’ token or length of the predicted sequence is equal to the length of longest training sequence.

Here are some of the interesting captions generated by the models for test set images that it hasn’t seen before:

startseq surfer is jumping over wave endseq



startseq dog is running through water endseq

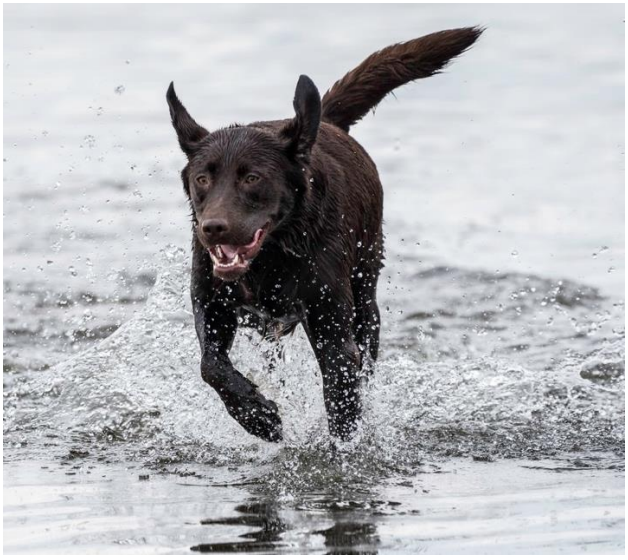

startseq basketball player in red is running endseq

startseq skier is skiing down snowy hill endseq

startseq woman in black shirt and white shirt is standing in front of whit
e building endseq

startseq man in black jacket is standing on sidewalk endseq

We can see that there some discrepancies where the model wasn't able to accurately describe what's happening in the image and by no means this is a perfect model as there is a lot of scope for model tuning for better results. Below is an entirely new image downloaded from the internet and we can see that the model was able to describe somewhat accurately of what's happening in the image (although the dog seems to be running rather than swimming).



```
startseq black dog is swimming in water endseq
```

## Future Work

Some of the ideas to improve the performance of the model are:

1. Using a larger dataset like the MS-COCO data set : http://cocodataset.org/#download. This ensures that our model is exposed to a wide variety of images and descriptions and hence perform better during inference.
2. Doing more hyper parameter tuning (learning rate, batch size, number of layers, number of units, dropout rate, batch normalization etc.) like a grid search using dev set.
3. Using Beam Search instead of Greedy Search during Inference.
4. Try reducing the size of vocabulary. Many of the words might appear less than 5 or 10 times. Eliminating these can improve the performance of the model.

## Applications

1. This cool application can open doors to some of the amazing possibilities that were only seen in sci-fi movies until now. Some of them are:
2. Self/Automatic driving is one of the biggest challenges and if we can properly caption the scene around the car, it can give a boost to the self-driving system.
3. We can create a product for the blind which will guide them to travel on roads or a product that automatically translates what is happening in the photo/video into speech. We can do this by first converting the scene into text and then the text to voice. Both are now some of the bleeding edge applications of Deep Learning.

4. CCTV cameras are everywhere today, but along with viewing the world, if we can also generate relevant captions, then we can raise alarms as soon as there is some malicious activity going on somewhere. This could probably help reduce some crime and/or accidents.
5. Automatic Captioning can make Image Search as good as web browsing, as then every image could be first converted into a caption and then search can be performed based on the caption.

## Conclusion

We have seen that with a right approach for data preprocessing and knowledge in neural networks i.e. how to build and train a neural network for particular task, training it, tweaking by playing with hyper parameters, we can develop a pretty decent image caption generator even though we have a fairly limited dataset. Finally, the some of the elements in sci-fi movies coming into reality. Also, this application can be used to solve numerous problems like the ones mentioned in the above section. It's like living in the future guys!