Image Caption Generation using Deep Learning

Aikansh Priyam
AXP190019
Axp190019@utdallas.edu
Computer Science Department

Shashi Shekhar
SXS180301
sxs180301@utdallas.edu
Computer Science Department

Abstract

In our project, we have analyzed how to generate a caption of the given images using deep learning. Given an image, the model should generate a sentence in the English language and the sentence should describe the image. There were three parts in this model: 1.) Extracting features from images and use that feature to give input to the deep learning model. 2.) Cleaning the text description and applying word embedding on the description and giving the output of this word embedding as input to LSTM. 3) Merging image features and LSTM output and using a decoder to generate the sentence. We defined different models and used 2 different feature extraction techniques for images. We got the best result for the merge model when the image feature extraction technique used was VGG16. Our results show that our model can generate captions for the image which is almost as good enough as humans.

1. Introduction

With the improvement in cameras and better camera quality in a smartphone, it has become really easy for people to click more and more pics. With the rise of social media and other photo-sharing websites, the number of photos available on the internet has increased exponentially. Many companies like Facebook have been working on this task of generating captions of the photos. Automatic caption generation from an image using natural languages like English is a challenging task, but it can have a great impact. For example, it can be very helpful to visually impaired people, as the auto-generation of captions can help them understand the background even better. This model can also be very helpful in getting accurate information from images or surveillance videos.

In our project, we have built a neural network model that will take an image and generate the sentence that describes the image. Deep learning has been used in this project to create a model and generate the sentences. The Deep Learning model consists of Convolution Neural Network, Long Term Short Memory(LSTM), and a decoder used for decoding the final sentence[5][4][6]. After learning from the features of images and all possible captions of the images, the deep learning model will generate an English sentence which will be grammatically correct and semantically descriptive. While human beings use a scene in the image to describe the image. There have been various methods designed by researchers that can take image as well as text as an input. Many such different models have been described by Marc Tanti et al. in his research paper "Where to put Image in an Image Caption Generator".

Bernardi et al., paper "Image captioning" has been a benchmark for other researchers working on this topic. Most captioning systems focus on what Hodosh et al. (2013) refer to as concrete conceptual descriptions, i.e, captions that describe what is strictly within the image, but, there has been growing interest in moving beyond this, with research on visual question-answering (Antol et al., 2015) and image grounded narrative generation (Huang et al., 2016) among others. The below image explains the block diagram of a basic model.

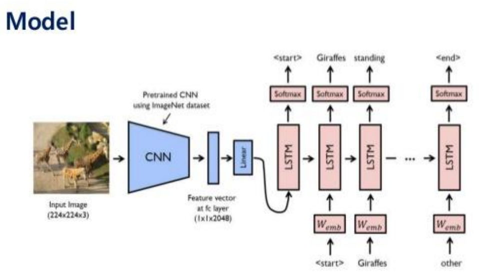


Figure 1. Architecture of the model for Caption Generation. CNN will be used to extract features from the image. LSTM will be used for input text and a decoder will be used to finally generate captions.

There are many approaches that have been used for image captioning. Few of them have been listed before:

1. One of the approach that has been used is using computer vision techniques to extract object detections and features from the image source. These features are then given as input to the NLG stage.
2. Other approach is that a caption is identified by computing the relevance of the text provided in training data of an individual image.
3. The other approach is where researchers have used neural network models where these models don't perform partial caption retrieval, instead they generate novel captions using Long Term Short Memory(LSTM), a type of Recurrent Neural Network(RNN).

In our project we have used 3rd approach where we have used Convolution Neural Network for feature extraction from the images and Long Term Short Memory for text processing. Depending on the way we use the feature extracted from Convolution Neural Network and Long Term Short Memory, we can build different neural network models that can help us get better output.

Our project aims to understand how we can use the combination of CNN and LSTM to generate the best model that can generate a sequence which closely describes the image. We have used two different existing CNN models to understand how features extracted from these models has impact on the models.

Contributions:

We started working on the project together and we started by doing literature survey by going through different research papers to understand the different approaches used by the researchers to build a model that can automatically generate captions. The papers published used 3 different datasets, Flickr8k, Flickr30k, and MOSCO but because of our limited computational power of our laptop and the limited time we had we decided to go with Flickr8k dataset. For the coding part, Shashi used existing CNN models VGG16 and Xception to extract features from the images. He then processed the text features corresponding to an images. The next part of the project was creating a deep learning model. Aikansh used the data generated in the first par(Image features and text) to create the deep learning model. Two different models(merge and par inject models) were defined using both feature extraction CNN models VGG16 and Xception and the models were tuned by changing the hyperparameters like Dropout, optimizer, learning rate. He(Aikansh) then created the sequence of the sentence for the model and then defined the model and fit the model on training and validation dataset. We used hyperparamters to tune the model with different optimisers, learning rate . Finally, we used our best model to run it on our test dataset to learn about our model efficiency by calculating the bleu score.

2. Related Work:

Many papers have been published in this area and different methods have been proposed by various authors in the last few years. Before that approaches were used on the basis of generation of simple sentences, which were generated by using feature extracted from the images[1][2]. Object detection was used to predict the sentences. With the growth of Deep Learning, there has been new techniques developed using Recurrent Neural Networks and Convolution Neural Networks[5][6][7].

Marc Tanti et. al.[7] proposed a few methods that can be used to define the models. In some papers like Bernardi et. al. [9] where he used RNNs only as generators. In his work RNN was trained to generate the next word in the sentence. A similar approach has also been discussed by LeCun et. al. [10] and Sutskever et. al. [11].

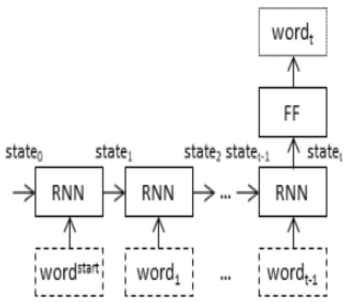


Figure 2. Depicts how RNN-based neural models work. RNN-> Recurrent Neural Network FF->Feed Forward layer, $word_i$ -> ith generated word in the caption

On the training side, there were a few papers which were based on a time-wise cross entropy training, few papers have also been published that have used Reinforcement Learning for this task and made a significant achievement and the use of Reinforcement learning has enabled the use of non-differentiable caption metrics as optimization objectives[12][13]. Steven et. al. has used self-critical sequence training (SCST) in their paper to generate captions. Self-critical sequence training or SCST is a form of the popular REINFORCE algorithm that, rather than estimating a "baseline" to normalize the rewards and reduce variance, utilizes the output of its own test-time inference algorithm to normalize the rewards it experiences. They used this approach that helped them in estimating the reward signal and the normalisation estimation was avoided, while harmonizing the model with respect to its test-time inference procedure.

As discussed in Ranzato et. al. [14] deep generative models for text are typically trained to maximize the likelihood of the next ground-truth word given the previous ground-truth word using back-propagation. This approach has been called "Teacher-Forcing" which has been discussed in the paper published by Bengio et. al. [15]. Teacher forcing is a method for quickly and efficiently training recurrent neural network models that use the ground truth from a prior time step as input. It is a network training method critical to the development of deep learning language models used in machine translation, text summarization, and image captioning, among many other applications. But, there is an issue with the approach as this approach creates a mismatch between training and testing, since at test-time the model uses the previously generated words from the model distribution to predict the next word. The exposure bias which has been discussed in Ranzato et. al. [14], lead to error accumulation during generation at test time, since the model has never been exposed to its own predictions.

Many approaches have been proposed by different researchers to overcome the issue of exposure bias problem that has been described above. In the paper published by Bengio et. al. [15] they have showed that feeding back the model's own predictions and slowly increasing the feedback probability during training leads to significantly better test-time performance. Another paper by Lamb et. al. proposed a method called Professor-Forcing [16]. Professor Technique is a technique which uses adversarial domain adaptation to encourage the dynamics of the recurrent network to be the same when training the network and when sampling from the network over multiple time steps.

Marc Tanti, in his work gave an alternative view stating that RNNs can be used more than generators. It can be used to primarily to encode sequences, but not using it directly to generate them. There have been many discussions as to how RNNs can be used for image captioning. The question of where the image feature should be given to model stills exists with a broader set of questions concerning how language can be grounded in perceptual information, questions which have been addressed by cognitive scientists (Harnad) and AI practitioners (Roy).

Two models that have been used by Tanti et. al. in his paper are:

1. Conditioning by injecting the image: Conditioning by injecting the image means injecting the image into the same RNN that processes the words. In inject architectures, the image vector which is derived from the activation values of a hidden layer in a convolutional neural network is injected into the RNN by treating it as a word and including it as a part of the caption. The RNN is then trained by encoding the features extracted by the convolution neural network model and the word embedding created after cleaning the input text into a single vector in such a way that this merged vector can be used to predict the words in the sentence correctly.

2. Conditioning by merging the image: Conditioning by merging the image means that the final state of the RNN is merged with the image features that were extracted. In merge architecture, the image vector which is derived from the activation values of a hidden layer in a convolutional neural network is merge with the output the Recurrent Neural Network by adding both the vectors and including it as a part of the caption. The words are then predicted using the encoded features merged by adding the features extracted from the convolution neural network model and the word embedding created after cleaning the input text into a single vector in such a way that this merged vector can be used to predict the words in the sentence correctly.

3. Proposed Work:

In our project, we have used deep learning to generate the captions of the image. There are three parts of the model that we developed.

1. The first part is extracting features from the image using a convolution Neural Network.
2. The second part is using the text related to the images by cleaning and processing the text and using word embeddings to feed it to the Long Term Short memory(LSTM).
3. The third part of the model was generating the caption using a decoder.

3.1 Convolution Neural Network(CNN):

A convolution neural network is a deep learning algorithm that takes an input as an image, gives weights and bias to different aspects of the objects in the image and be able to differentiate one from the other. The pre-processing that we normally do in other Machine Learning models is more than what we do in convolution neural network.

The architecture of the convolution neural network has been inspired by the human brain, Visual cortex to be precise. The advantage of using convolution neural network for extracting features from the images are that, is able to successfully capture the Spatial and Temporal dependencies in an image through the application of relevant filters. The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and reusability of weights.

A convolution neural network can be used to reduce the images into a form that is comparatively easier to process, without losing features which is important to get a good prediction. The different operations that can be performed on the images are to get a convolution layer, pool layer, full connected layer. The Convolution layer helps to extract high-level features from the image. Similarly, pool layer helps to reduce the spatial size of the Convolved Feature. It helps to decrease the computational power required to process the data through dimensionality reduction. Using Pool layer also help in extracting dominant features which are rotational and positional invariant, thus maintaining the process of effectively training of the model. After the image goes through the convolution layer and the pool layers the model gets enabled to understand the features. Finally, we add a Fully Connected layer that is a way to learn non-linear combination of learning of the high level features as represented by the output of the convolutional layer. The model is trained over a certain number of epochs and the model can learn to differentiate between low-level and high-level features which are classified using a softmax function.

Many convolution neural network architectures have been trained on millions of images which we can directly use to extract features. We have used 2 such architectures in our models to extract features. The two models used in this project are:

3.1.1 VGG16:

VGG16 is a convolutional neural network model proposed by K. Simonyan and A. Zisserman in their paper Simonyan et. al. [17]. This model proposed by Simonyan et. al. have a test accuracy of 92.7% in ImageNet[18]. ImageNet is a dataset of over 14 million images that belongs to almost 1000 classes. VGG16 is still one of the famous model that researchers use for their feature extraction in convolution neural network. VGG16 makes the improvement over AlexNet by replacing large kernel-sized filters with multiple 3×3 kernel-sized filters one after another. VGG16 was trained for weeks and was using NVIDIA Titan Black GPU’s.

VGG16 as the name suggests has 16 hidden layers and significantly outperforms previous CNN models.

3.2.2 Xception:

Xception is a convolutional neural network model proposed by Francois Chollet [19] while he was working at Google**.** Xception is a convolutional neural network that is 71 layers deep. In this paper the Inception modules have been replaced with depthwise separable convolutions.

3.2 Long Term Short Memory (LSTM):

Long Short-Term Memory (LSTM) networks are a type of recurrent neural network capable of learning order dependence in sequence prediction problems. This is a behavior required in complex problem domains like machine translation, speech recognition, and more.

$$\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
\tilde{c}_t &= \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c}_t \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}$$

Figure 3: LSTM with forget gate

In out project we have used LSTM at different places in the model. In the two models that we have created I have mentioned how we have used LSTM in these models.

3.3. Deep Learning Models

We have created two different models in our project. Both the models have been described below:

1. Par Inject model: Par Inject model in which the image is injected into the same RNN that processes the words. In inject architectures, the image vector which is derived from the activation values of a hidden layer in a convolutional neural network is injected into the RNN by treating it as a word and including it as a part of the caption. The RNN is then trained by encoding the features extracted by the convolution neural network model and the word embedding created after cleaning the input text into a single vector in such a way that this merged vector can be used to predict the words in the sentence correctly.

2. Merge model: In the merge model the final state of the RNN is merged with the image features that were extracted. In merge architecture, the RNN is not fed with the image vector at any point. Instead, the image vector which is derived using the Xception/VGG models are merged with the output the Recurrent Neural Network by adding both the vectors. The words are then predicted using the encoded features merged by adding the features extracted from the convolution neural network model and the word embedding created after cleaning the input text into a single vector in such a way that this merged vector can be used to predict the words in the sentence correctly.

The architecture of both the models is present below:

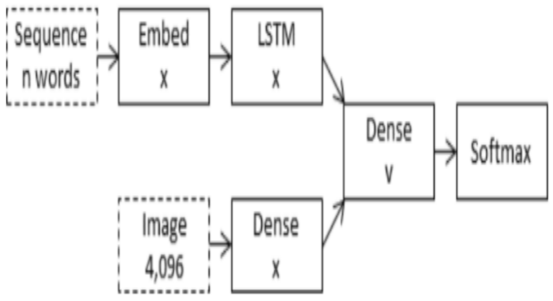


Figure 4.a) Merge architecture

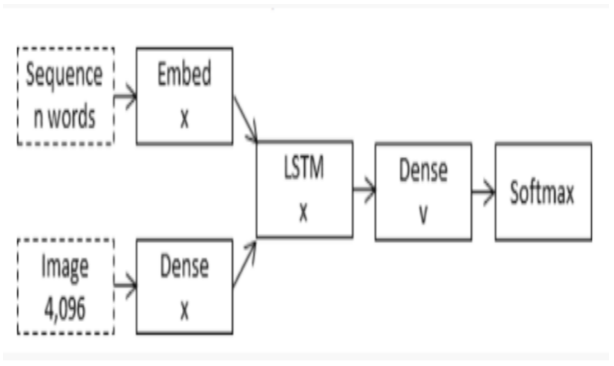


Figure 4.b) Par-inject architecture

4. Experiment:

4.1. Datasets:

There are mainly three popular datasets that have been used in different models previously.

1. The smallest dataset is Flickr8k[20] which has 8000 images, all of which are paired with 5 captions describing about the images. The images in this datasets were chosen from six different Flickr groups, and tend not to contain any well-known people or locations, but were manually selected to depict a variety of scenes and situations.

2. The second dataset is Flickr30k [21] which has 30000 images, which augments the 158k captions from Flickr30k with 244k coreference chains linking mentions of the same entities in images, as well as 276k manually annotated bounding boxes corresponding to each entity.

3. The third and the largest dataset is MSCOCO dataset[22]. It has more than 330k images out of which more than 220k are labelled. All of the images have 5 captions associated with it. The images belongs to 80 object categories.

We have used Flickr8k dataset in our project because of the limitation of our computational power. Out of the 8000 images present in the dataset, 6000 images have been used for training, 1000 images have been used for testing and 1000 images have been used for testing. Even though the size of the dataset is small it has given us pretty good results.

The Datasets were taken from: https://www.kaggle.com/adityajn105/flickr8k/activity

4.2 Training Methods:

For training our model, we have created multiple models and used different metrics to tune our models with hyperparameters.

The image features were extracted from predefined convolution neural network models Xception and VGG16. We did not rescaled or cropped the image. The entire image was taken and all the features related to the data were extracted. We had a vocabulary of more than 7000 words which helped us generate sentences that were grammatically and semantically correct.

As discussed in the above sections, we have used LSTM in different ways to create 2 different models to see how the result varied. We trained our models using Stochastic gradient descent and Adam optimiser for our models. We varied learning rate between 0.01 to 0.001 and observed how the output varied with different learning rate. We also used the hyperparameter “Dropouts” and varied it between 0.4 to 0.6 to observe our results.

We used bilingual evaluation understudy, i.e BLEU score to determine the efficiency of our model. BLEU score is an algorithm for evaluating the quality of text which has been machine translated from one language to another. Quality is considered to be the correspondence between a machine's output and that of a human. Bleu score ranges between 0 and 1. A bleu score of 1 means that the translation was perfect, i.e. as good as a human and bleu score of 0 means the translation was very poor.

A huge problem while training our model was that of overfitting. That is the reason we applied dropouts of 0.5 so that our model don’t overfit but still after 5-6 epochs our model started overfitting.

We ran our model ranging from 2 to 20 epochs and most of the time after 5-6 epochs the model stopped improving its validation loss and started to overfit.

4.4 Results:

The results for different models have been mentioned in the below table.

Model Name	BLEU-1	BLEU-2	BLEU-3	BLEU-4
VGG Merge Model	0.5500920810	0.2872521751	0.186646455	0.07890677
Xception Merge Model	0.5466526093832367	0.3169599470644139	0.22686134178017556	0.11226832275762506
VGG Inject model	0.4028189374774124	0.2177655795698062	0.1495423428 8161417	0.0629744035 8052132
Xception Inject Model	0.32935934489402696	0.18299946929667998	0.13270695161429769	0.05907220551002299

The epochs of the best 2 models have also been mentioned below. The below epochs shows how early the model starts to overfit.

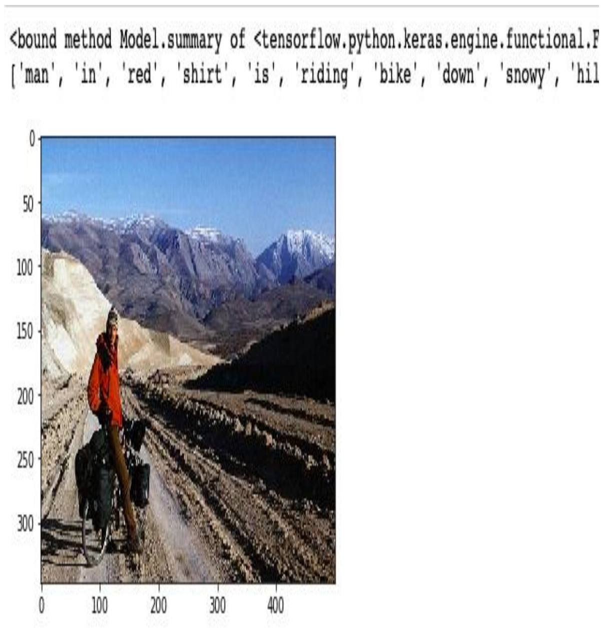
Training result for Xception Merge Model:

```
Epoch 1/4

Epoch 00001: val_loss improved from inf to 3.9052
6, saving model to model-ep001-loss4.355-val_loss
3.905.h5

9576/9576 - 975s - loss: 4.3550 - val_loss: 3.9053
Epoch 2/4

Epoch 00002: val_loss improved from 3.90526 to 3
.76824, saving model to model-ep002-loss3.652-va
l_loss3.768.h5

9576/9576 - 1028s - loss: 3.6523 - val_loss: 3.7682

Epoch 3/4

Epoch 00003: val_loss improved from 3.76824 to 3
.72397, saving model to model-ep003-loss3.420-va
l_loss3.724.h5
9576/9576 - 849s - loss: 3.4204 - val_loss: 3.7240

Epoch 4/4

Epoch 00004: val_loss did not improve from 3.723
97
9576/9576 - 867s - loss: 3.2996 - val_loss: 3.7499
```

Training result for VGG Merge Model:

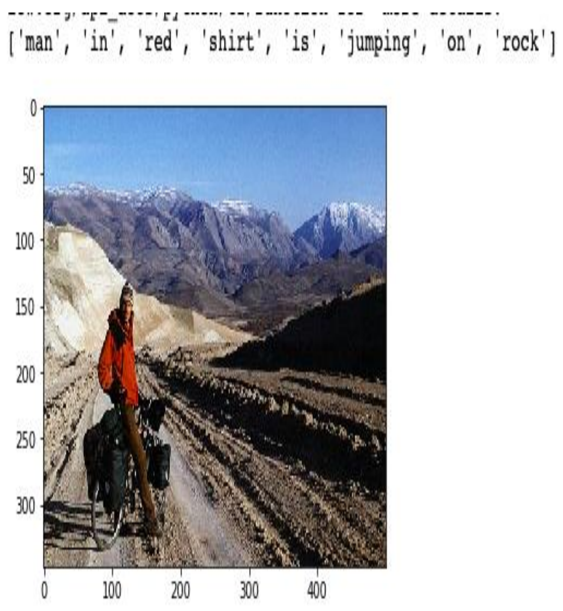
```
Epoch 1/4

Epoch 00001: val_loss improved from inf to 4.0643
3, saving model to model-ep001-loss4.526-val_loss
4.064.h5

9576/9576 - 2437s - loss: 4.5256 - val_loss: 4.0643

Epoch 2/4

Epoch 00002: val_loss improved from 4.06433 to 3
.89721, saving model to model-ep002-loss3.847-va
l_loss3.897.h5

9576/9576 - 860s - loss: 3.8468 - val_loss: 3.8972

Epoch 3/4

Epoch 00003: val_loss improved from 3.89721 to 3
.87423, saving model to model-ep003-loss3.634-va
l_loss3.874.h5

9576/9576 - 915s - loss: 3.6339 - val_loss: 3.8742

Epoch 4/4

Epoch 00004: val_loss did not improve from 3.874
23
9576/9576 - 973s - loss: 3.5321 - val_loss: 3.8937
```

4.4 Comparison of outputs of different models



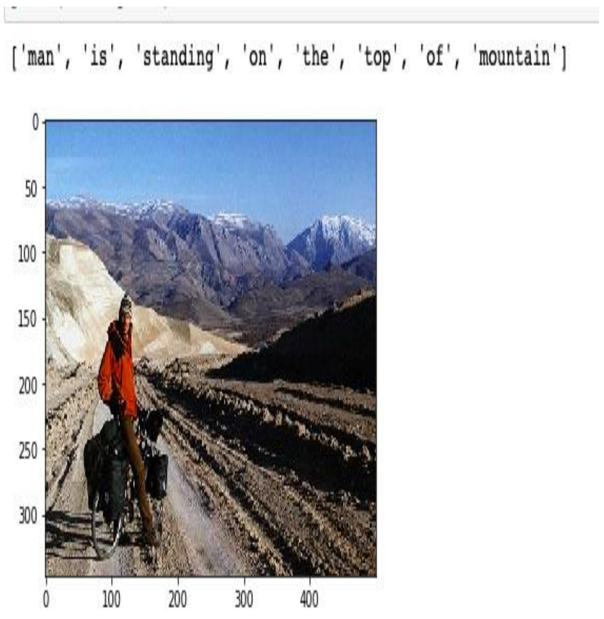
```
<bound method Model.summary of <tensorflow.python.keras.engine.functional.F
['man', 'in', 'red', 'shirt', 'is', 'riding', 'bike', 'down', 'snowy', 'hil
```

Figure 5.a) Output for VGG merge model



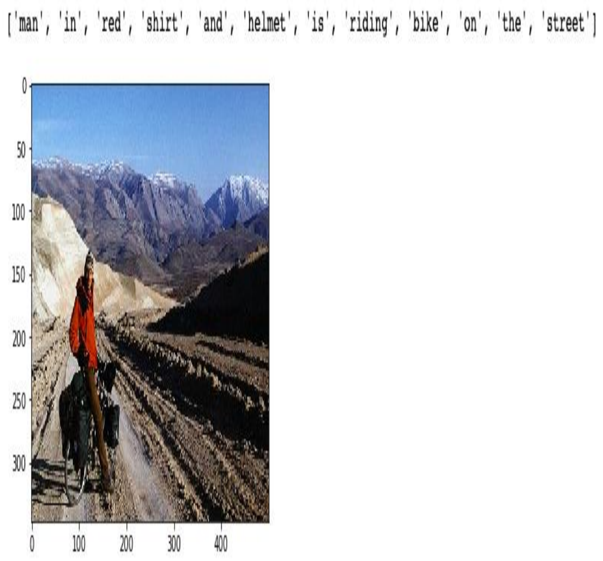
```
['man', 'in', 'red', 'shirt', 'is', 'jumping', 'on', 'rock']
```

Figure 5c) Output for VGG inject model



```
['man', 'is', 'standing', 'on', 'the', 'top', 'of', 'mountain']
```

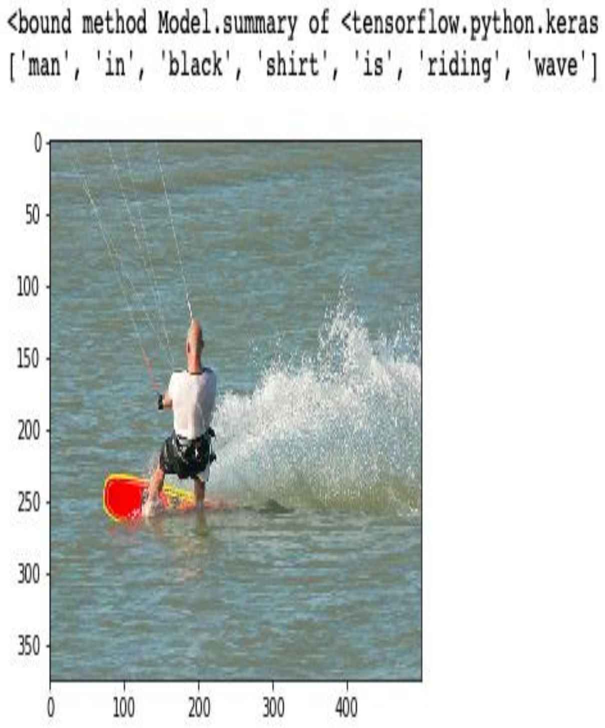
Figure 5b) Output for Xception Merge model



```
['man', 'in', 'red', 'shirt', 'and', 'helmet', 'is', 'riding', 'bike', 'on', 'the', 'street']
```

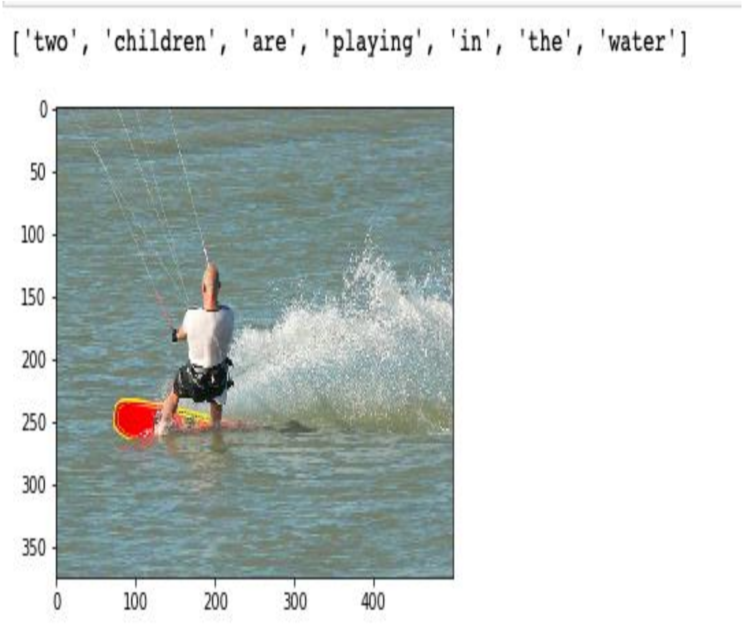
Figure 5.d) Output for Xception inject

Comparison of 2nd set of image:



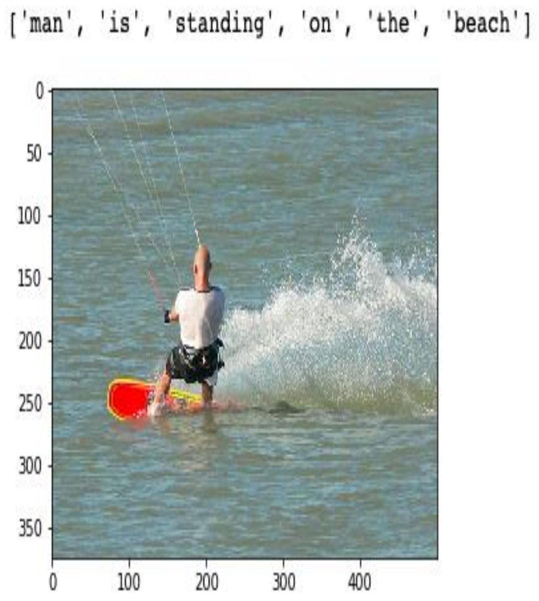
```
<bound method Model.summary of <tensorflow.python.keras
['man', 'in', 'black', 'shirt', 'is', 'riding', 'wave']
```

Figure 6.a) Output for VGG merge model



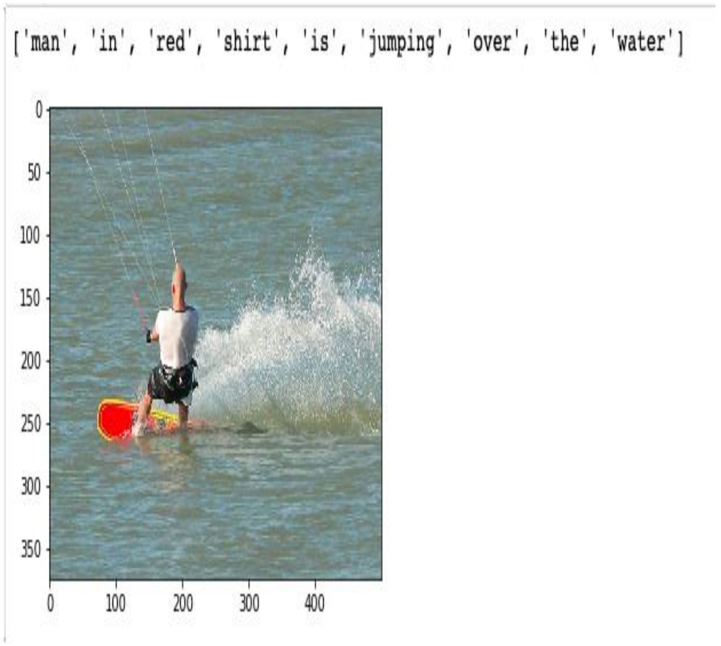
```
['two', 'children', 'are', 'playing', 'in', 'the', 'water']
```

Figure 6.c) Output for VGG inject model



```
['man', 'is', 'standing', 'on', 'the', 'beach']
```

Figure 6.b) Output for Xception Merge model



```
['man', 'in', 'red', 'shirt', 'is', 'jumping', 'over', 'the', 'water']
```

Figure 6.d) Output for Xception inject

5. Conclusion and Future Work:

In the results section we compared bleu scores of all the models that we developed. While belu-1 score of both VGG_Merge and Xception_Merge were almost same but bleu-2, bleu-3 and bleu-4 scores of Xception_merge was slightly better than VGG_Merge.

We also gave results of how our validation loss changed with increasing number of epochs. We observed that validation loss decreased for 3-4 epochs and after that it stopped decreasing as the model started overfitting.

Finally we compared our models by testing them with individual pictures. We tested our models with a few sample pictures and as can be seen above the output for the inject models were not up to the mark but the results of merge models were pretty good. While bleu score of merge models for VGG and Xception were almost same but we could see in the above images that VGG model is slightly better.

We can conclude from our results that the best model that we have for now is VGG Merge model.

The limitation of our work is that though we were able to generate captions that were good enough but the captions sometimes missed an important object in the picture. One of the reason of our limitation could be that the dataset we used was the smallest and our model was not trained enough to give a perfect caption.

In the future, we would like to explore more methods that we could use to build our models such that we could get better score even with Flick8k dataset. We also plan to implement our existing models with bigger datasets Flickr30k and MSCOCO datasets to see how our models perform with those datasets.

6. References:

1. Richard Socher and Li Fei-Fei. Connecting modalities: Semi-supervised segmentation and annotation of images using unaligned text corpora. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2010.
2. Benjamin Z Yao, Xiong Yang, Liang Lin, Mun Wai Lee, and Song-Chun Zhu. I2t: Image parsing to text description. Proceedings of the IEEE, 98(8):1485–1508, 2010
3. Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015
4. Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and Tell: Lessons Learned from the 2015 MSCOCO Image Captioning Challenge. IEEE Transactions on Pattern Analysis and Machine Intelligence, 39(4):652– 663, 2017
5. Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard S Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In Proceedings of the International Conference on Machine Learning, 2015.
6. Marcella Cornia, Lorenzo Baraldi, and Rita Cucchiara. Show, Control and Tell: A Framework for Generating Controllable and Grounded Captions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019.

7. Marc Tanti, Albert Gatt, Kenneth P. Camilleri, Where to put the Image in an Image Caption Generator, Natural Language Engineering, 24(03)

8. Marc Tanti, Albert Gatt, Kenneth P. Camilleri, What is the Role of Recurrent Neural Networks (RNNs) in an Image Caption Generator?, Proc. 10th International Conference on Natural Language Generation (INLG'17)

9. Raffaella Bernardi, Ruket Cakici, Desmond Elliott, Aykut Erdem, Erkut Erdem, Nazli Ikizler-Cinbis, Frank Keller, Adrian Muscat, and Barbara Plank. 2016. Automatic Description Generation from Images: A Survey of Models, Datasets, and Evaluation Measures. Journal of Artificial Intelligence Research, 55:409–442.
10. Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. Nature, 521(7553):436–444.

11. Ilya Sutskever, James Martens, and Geoffrey Hinton. 2011. Generating Text with Recurrent Neural Networks. In Procededings of the 28th International Conference on Machine Learning (ICML’11), pages 1017– 1024, Bellevue, WA. ACM

12. Steven J Rennie, Etienne Marcheret, Youssef Mroueh, Jarret Ross, and Vaibhava Goel. Self-critical sequence training for image captioning. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017
13. Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence level training with recurrent neural networks. In Proceedings of the International Conference on Learning Representations, 2015
14. Marc' Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence level training with recurrent neural networks. ICLR, 2015.
15. Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In NIPS, 2015
16. Alex Lamb, Anirudh Goyal, Ying Zhang, Saizheng Zhang, Aaron Courville, and Yoshua Bengio. Professor forcing: A new algorithm for training recurrent networks. Neural Information Processing Systems (NIPS) 2016, 2016
17. K. Simonyan and A. Zisserman University of Oxford "Very Deep Convolutional Networks for Large-Scale Image Recognition".
18. Deng, J. et al., 2009. Imagenet: A large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition. pp. 248–255.
19. Chollet, Francois. (2017). Xception: Deep Learning with Depthwise Separable Convolutions. 1800-1807. 10.1109/CVPR.2017.195.
20. Hodosh, Micah, Peter Young, and Julia Hockenmaier. "Framing image description as a ranking task: Data, models and evaluation metrics." Journal of Artificial Intelligence Research 47 (2013): 853-899.
21. B. A. Plummer, L. Wang, C. M. Cervantes, J. C. Caicedo, J. Hockenmaier and S. Lazebnik, "Flickr30k Entities: Collecting Region-to-Phrase Correspondences for Richer Image-to-Sentence Models," *2015 IEEE International Conference on Computer Vision (ICCV)*, Santiago, 2015, pp. 2641-2649, doi: 10.1109/ICCV.2015.303.
22. Lin, Tsung-Yi & Maire, Michael & Belongie, Serge & Hays, James & Perona, Pietro & Ramanan, Deva & Dollár, Piotr & Zitnick, C.. (2014). Microsoft COCO: Common Objects in Context.