Image Captioning

Neural Networks - EEE443 (Final Project Report)

Usama Faisal, Muhammad Abdullah Shafat Mulkana, Mohammad Usayrim Rehman, Hasaan Ijaz

Abstract

In this paper, we explore models designed for image captioning. We use multiple encoder-decoder models in order to determine the better architecture. We also implement an attention model, which is a technique employed to enhance the performance of the models. Due to limited computational resources, we could not explore the full capability of the attention model. The best model from our results is the 2 LSTM layer injection model since it also uses the image encodings in the recurrent layer.

Introduction

Image captioning is a fundamental deep learning problem that has been the focus of research for many years. It combines the fields of Computer Vision (CV) and Natural Language Processing (NLP) in order to first capture information in the images and, based on this, generate a sequence of words accurately describing the image. Applications of systems that can use image captioning include visual aid systems for the visually impaired, social media, image indexing, and personal assistants [1]. The difficulty in this task is to create models that can accurately capture the relationship between the different objects in an image and then generate accurate, comprehensible captions [2]. *Mao et al.* used an encoder-decoder approach consisting of a Convolutional Neural Network and a Recurrent Neural Network, respectively, for image captioning [3][2]. The encoder is used to get dense representations of the images, usually using a Convolutional Neural Network (CNN). The decoder consists of a Recurrent Neural Network (RNN), and the two models are combined using a multimodal layer [3]. Common practice is to use popular CNN models such as ResNet and Inception pre-trained on large natural image datasets such as 'ImageNet' to extract the encodings. Furthermore, the captions are tokenized, and word embeddings are generated, for which it is common practice to use pre-trained word embeddings such as Global Vectors (GloVe) [4]. GloVe 6B are word embeddings of a corpus of 400,000 uncased words and give embeddings in 50, 100, 200, and 300 dimensions [5]. Image captioning models differ with respect to the encodings used, embeddings used, and the decoder structure. Another popular decoder structure uses the concept of attention. Attention models are used as input models which focus on a specific part of the images [6]. In attention models, attention weights are first generated, usually through a Neural Network, which is then multiplied with the input

to the model in order to get a context vector, i.e., the part of the input under focus for that instance [7].

Evaluating the performance of an image captioning model is not as straightforward as evaluating the performance of an image classification model, where the accuracy metric can be used to judge the performance. For image captioning models, multiple captions can represent the same image, and therefore the Bi-Lingual Evaluation Understudy (BLEU) score will be used to evaluate the performance of the models. BLEU compares the predicted translation by the model (known as candidate translation) with the original caption (or captions) (known as reference translations). [8]

Methods

This section will describe the dataset we are using, the model architectures employed, and the techniques used to get the results.

Dataset

The dataset provided contains URLs of images, image ids, image captions, image encodings (not used), and a words dictionary. We have 82,782 links for the images. We use the 'urllib' library to access the URLs and save the images. During this process, we notice that some links are broken, giving either the '404 Not Found' or '500 Server error' messages, and thus only 70977 images could be downloaded from the training set. Inspecting the dataset, we also see that we have an average of about five captions per image, with a maximum of 7 and a minimum of 1 observed. The total number of captions then is 343,134. The images are natural images of varying dimensions. The word dictionary consists of 1004 words. Each caption starts with a 'x_START_' and ends with 'x_END_'. Since the captions are padded to make them all of length 17, the padded 0's represent 'x_NULL_'. Furthermore, there exists a code 'x_UNK_' for unknown words.

The testing dataset is similar in structure with 40,503 URLs, out of which only 35,099 images could be downloaded.

GloVe Embeddings

Word embeddings help in understanding what different words mean and map the words onto a dense vector space. The embeddings cluster together similar words, and words that are distinct stay far apart. Many state-of-the-art implementations extract word embeddings, like Word2vec, ELMo, and GloVe [9]. We use the 6B GloVe 300d pre-trained embeddings, which give us the word embeddings in 300 dimensions [5]. GloVe is a better choice for the image-captioning problem since it incorporates global statistics, as well as local statistics, to acquire the word vectors. Global statistics are essential in deriving the semantic relationships between different words [9]. The pre-trained embeddings are available as a .txt file. Using the script in the Keras tutorial [10], we import this file and extract only the embeddings we want for our 1004 words. During extraction of word embeddings, we see that there exists words 'xWhile', 'xCatch', 'xCase',

'xFor', 'xEnd'. Removing the x's and making all these words lowercase, we see that the embedding matrix can be extracted without issue. However, this is not an issue when training the embeddings. The 'x_START_', 'x_END_', 'x_NULL_', 'x_UNK_' are vectors with a 1 placed at a specific location in the embedding dimensions in order for the model to distinguish them.

ResNet encodings

In order to get the encodings, we use the ResNet50 CNN model, pre-trained on the ImageNet dataset. The ResNet model uses the concept of skip connections which tackles the problem of vanishing gradients by introducing alternative paths for the gradients to follow [11]. These paths greatly increase the learning capability and thus the performance of the ResNet [11]. The top 1% error of the model is 23.85% and the top 5% error is 7.13% [12]. These values are reasonable for us in order to get the encodings. In order to get the encodings, we remove the average pooling, fully connected, and the final softmax layers (these layers are used for classification). This means we have encodings of dimensions 1x1x2048, which we put in a dictionary and save using pickle, a Python library.

Merge Model (GloVe + LSTM)

Merge models have an RNN based decoder where the RNN only takes the caption sequence as the input [13]. In the case that the image features are also fed into the RNN, the model is referred to as the Injection model [13] which is discussed later. The model architecture implemented is shown in Fig. 1

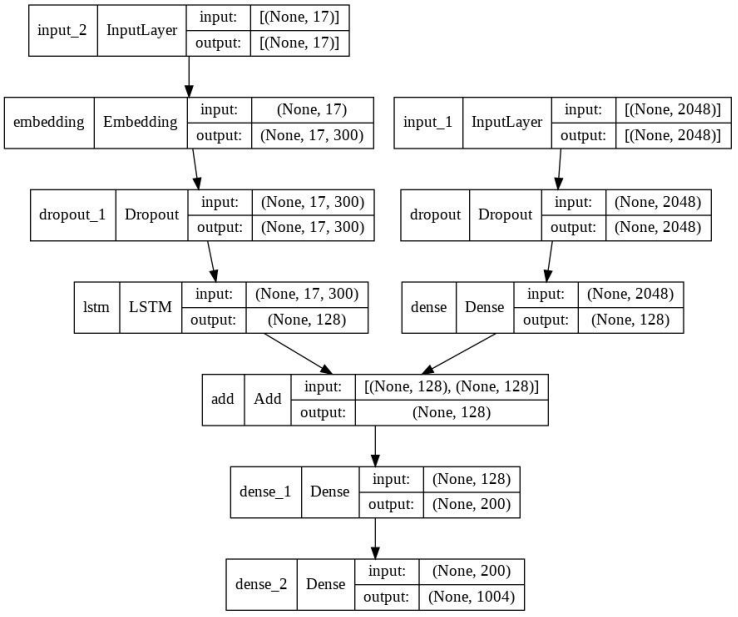


Figure 1: The model architecture of the first merge model implemented. Model taken from [1]. The model consists of 10 layers out of which 1 layer (embedding) is not trained since GloVe embeddings are used. Total parameters are 1,010724, out of which 709,524 are trainable, and 301,200 are non-trainable. The non-trainable parameters are of the embedding layer.

The model has two main branches. The first handles the caption information using the caption input. It determines the embeddings, passes through a dropout layer, and then to a Long-Short Term Memory (LSTM) layer with 128 units. LSTMs have 3 gates: input, output, and forget [14]. Using these gates, it controls its memory. The embedding layer in this model is set to 'trainable = False' with the weights initialized as the ones extracted using GloVe. The second branch uses the image encodings, passing them through a dropout layer and then a dense layer to decrease the feature space. As mentioned in [4], there are multiple ways to merge these

two outputs, where adding them works best. Therefore, we add the outputs from the two branches and pass them through a two-layer dense network which gives the output of dimension 1004, which corresponds to the words in the dictionary. The loss function for the model is the Sparse cross-categorical entropy, the optimizer is Adam (where the default learning rate is 1e-3). The activations for all layers except the output layer is ReLU, while the activation function for the final layer is softmax. The dropout probability is 0.5, which is done to keep the model from overfitting to the training dataset.

This implementation is taken from [1], where the model predicts the next word given an image encoding and previous words in the caption. Therefore, a data generator function is used, which creates, for each caption of each image, vectors that consist of previous n words, and the n+1 word is set as the target word. In this way, the model generates the captions on a word-to-word basis.

Merge Model (Self Embedding + GRU)

Here, we use the same model structure defined previously, but we use a Gated Recurrent Unit (GRU) instead of the LSTM. GRUs have only two gates, reset and update, and are sometimes more effective than LSTMS to determine the long-term dependencies [14]. The selection of GRU over LSTM or vice versa depends on the type of dataset used [14]. Therefore, we use the GRU to explore its performance on the given task. Furthermore, we randomly initialize the embedding weight matrix and train it together with the model to observe the effects of self-trained embeddings, which are expected to be more tailored to our specific data. The model architecture is given in Fig. 2

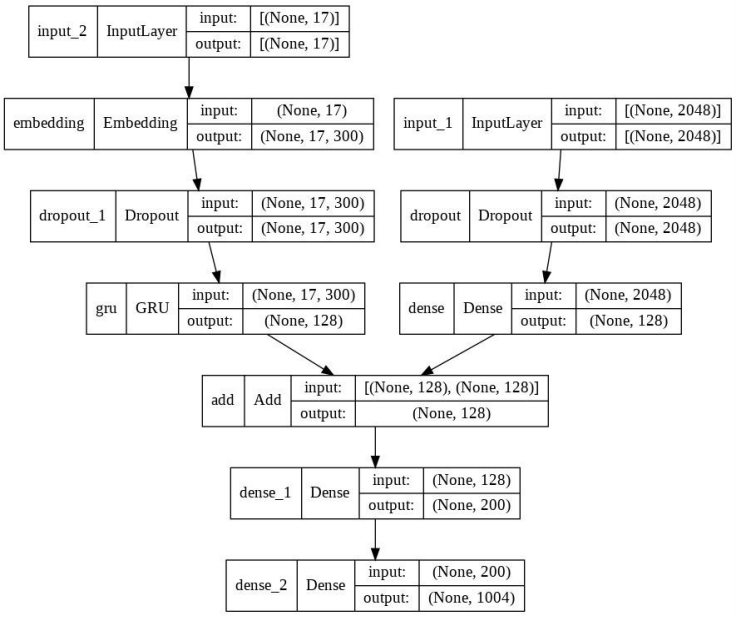


Figure 2: The model architecture of the second merge model used. Similar in structure to the previous model but uses a trainable embedding layer with GRU as RNN layer. The total parameters are 956,196, which are all trainable.

Injection Model (GloVe +LSTM)

As mentioned before, models where the recurrent layer handles both language and visual feature information are referred to as injection models [13]. The model architecture, shown in Fig. 3, is taken from [15], where the two inputs are passed through a dense layer, making the two equal in size. In the implementation in [15], the two inputs are concatenated. However, we choose to add them as research suggests that this is a better option [4]. Next, this sum is passed through a two-layer LSTM

network with 128 and 512 layers, respectively. Finally, a fully connected layer is used along with an activation layer using softmax to predict the captions. In this model, we again use the GloVe embeddings since it reduces the number of trainable parameters and training time.

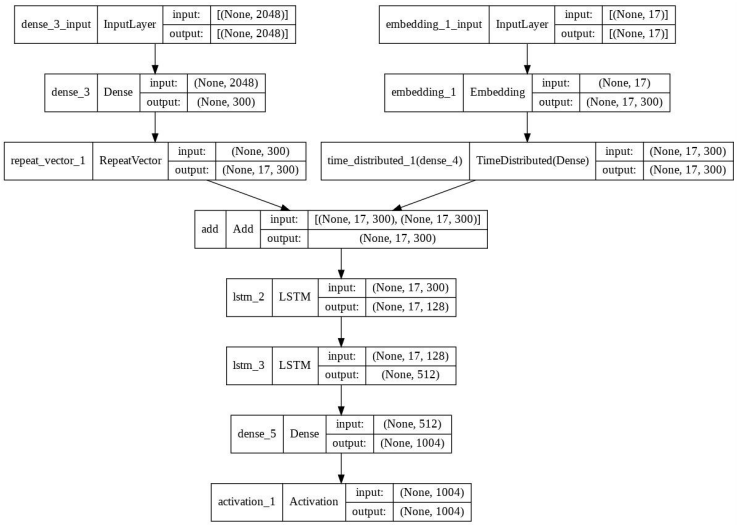


Figure 3: The model architecture of the injection model used. Model taken from [15]. Uses GloVe embedding layer and two-layer LSTM network. The total number of parameters is 3,053,668 where 2,752,468 are trainable and 301200 are non-trainable (embedding layer).

Injection Model (InceptionV3+GloVe +LSTM)

The architecture plot for the model is given below. The recurrent layer of this model comprises a Long Short Term Memory with 256 layers, and Glove Embeddings are used to minimize training time. The model architecture is explained in detail along with the flow chart.

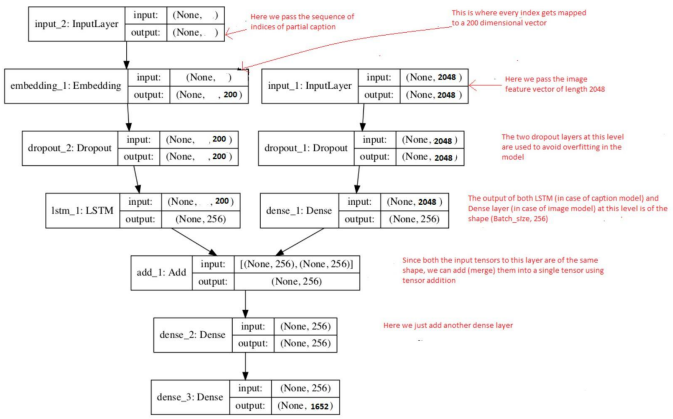


Figure 3: The model architecture of the Inception Injection model used. Model taken from [16]. The total number of parameters is 1,516,218 where 1,315,818 are trainable and 200,400 are non-trainable (embedding layer).

Attention Model

We implement the model in [17] which is based on the model proposed by the *"Show, Attend and Tell: Neural Image Caption Generation with Visual Attention"* paper. The model uses the Inceptionv3 model, pre-trained on ImageNet, to generate the encodings. This is done so that we can have image encodings of dimensions 8x8x2048 which are taken from the last convolutional layer of the CNN model [17]. The model consists of two blocks: a CNN encoder and an RNN decoder. The 8x8x2048 Inception v3 encodings are reshaped to 64x2048 and passed through the CNN encoder, extracting a vector of size 64x256 (the second dimension is kept the same as the embedding dimension). Next, the output from the decoder is given to the RNN decoder along with the previous hidden state of the RNN (initialized as zero) as well as the single word from the caption at that time step. The model then predicts the next word and in the

following iteration, the next actual word from the caption is given as the input. This method of giving the actual caption words to predict the next word is called teacher forcing and is a common technique employed to increase the performance of RNNs [17][18]. The RNN decoder uses an attention technique where the hidden state from the previous time step is used to compute a score for each feature of the image. Multiplying these scores with the features gives us a more focused representation of the image encodings which is then concatenated with the caption embeddings and fed to a GRU. Lastly, a two-layer dense network is used to get the final predictions over the 1004 words. When trained for the first time, we observed early overfitting so we used a dropout layer between the two output layers with a probability of dropout = 0.1.

Due to the large encoding size and model architecture, this model requires high computational power to train. Since this was not available the attention model was not trained on the fully downloaded training set. To train this model, we used only 30393 images and we used a validation set of 9118 images. The model was saved every 5 epochs and the best model was loaded for inference.

Analysis Methods

To analyze our model results, we chose a random set of 10,000 images from the test set. Keeping this set constant over the merge and injection models, BLEU-1, BLEU-2, BLEU-3, and BLEU-4 scores were calculated. BLEU-1 calculates the 1-gram overlap between the model's prediction and the original caption. For BLEU-2, BLEU-3, and BLEU-4, the n-gram changes to 2, 3, and 4, respectively. These scores are calculated using the NLTK library of Python, where equal weights are provided to calculate the specific n-gram BLEU score [19]. We then compare the difference in these scores across the three models. Furthermore, we chose 10 random images from the test set and generated captions using all the models, and then compared them with the given captions.

Results

The models were trained on GTX 1050 gpu, and their training times are shared in table 1 below.

Table 1. Training times for all the models implemented in this project, and the number of epochs until when they were trained. The max number of epochs for each model was 100.

Models	**Merge Model (GloVe + LSTM)**	**Merge Model (Self Embedding + GRU)**	**Injection Model (GloVe + LSTM)**	**Injection Model (InceptionV3+GloVe+LSTM)**	**Attention Model**
Training Time	9 hrs (60 epochs)	7.5 hrs (50 epochs)	7 hrs (22 epochs)	7 hrs (50 epochs)	3 hrs (10 epochs)

The training loss and validation loss vs epochs plots are shared in the figures below.

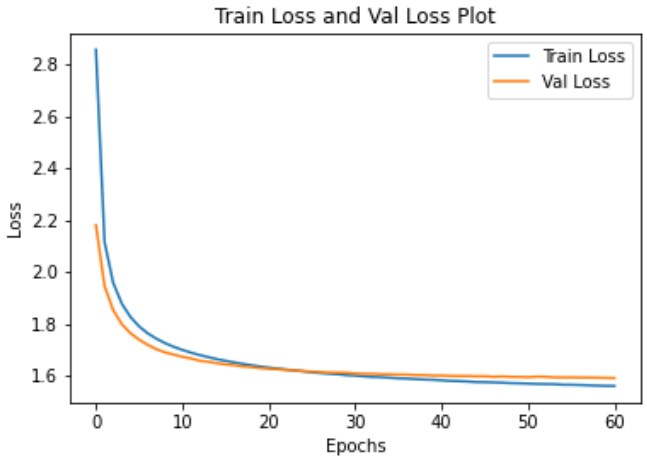


Figure 4: Training Loss and Validation Loss plots for the Merge Model, which uses ResNet50 for the image encodings, an LSTM network as the RNN decoder, and GloVe embeddings.

Fig. 4 shows the training loss and validation loss vs epochs plot for the Merge Model (GloVe + LSTM). The model does not overfit as the validation loss stays on the same path as the training loss over the course of around 60 epochs. The maximum number of epochs was set as 100, but the model was stopped after around 60 epochs as the validation loss did not seem to decrease.

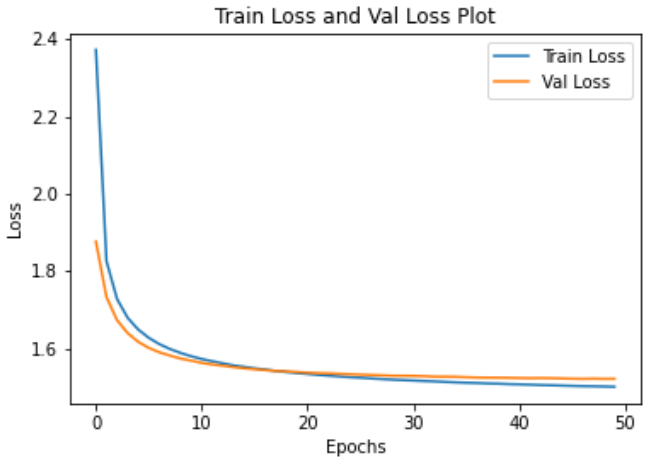


Figure 5: Training Loss and Validation Loss plots for the Merge Model, which uses ResNet50 for the image encodings, a GRU network as the RNN decoder, and word embeddings that are trained along with the rest of the model.

Fig. 5 shows the training loss and validation loss vs epochs plot for the Merge Model (Self Embedding + GRU). The plots for this model are similar to the plots for the previous model. This model learns just as well as the other model. Although, it reaches a lower validation loss score than the previous model, and it takes fewer epochs to achieve better results.

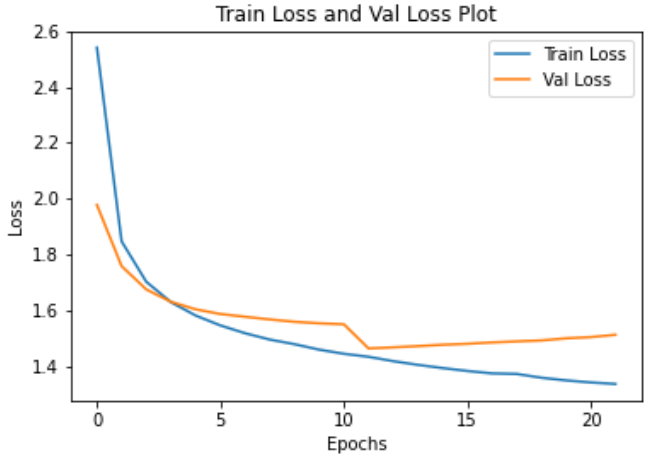


Figure 6: Training Loss and Validation Loss plots for the Injection Model, which uses ResNet50 for the image encodings, a 2-layer LSTM network as the RNN decoder, and GloVe embeddings to extract word vectors for the vocabulary words.

Fig. 6 shows the training loss and validation loss plots against the number of epochs for the Injection Model (GloVe + LSTM). Compared to the previous two models, this model learns the quickest and achieves much better performance after around 20 epochs only. The reason behind the improvement in model learning is the 2-layer LSTM network in the Injection Model. The training was stopped after 20 epochs as the validation loss seemed to increase, meaning our model was moving towards over-fitting.

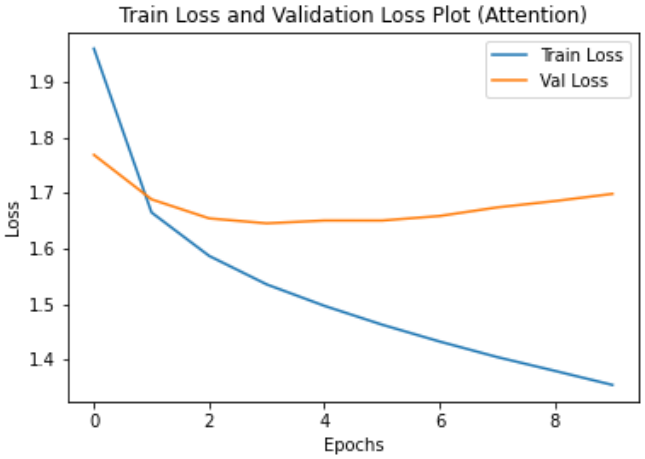


Figure 7: Training Loss and Validation Loss plots for the Attention Model, which uses InceptionV3 for the image encodings, a single-layer GRU as the RNN decoder, and word embeddings that are trained along with the rest of the model.

Fig. 7 shows the training validation loss plots for the Attention Model. The model starts to overfit fairly quickly after just 4 epochs, but the loss trajectory before epoch 4 seems to be good. The model was stopped after 10 epochs as there were no signs of the model improving on validation loss. For the inference on the test dataset, the saved model after the 5th epoch is used.

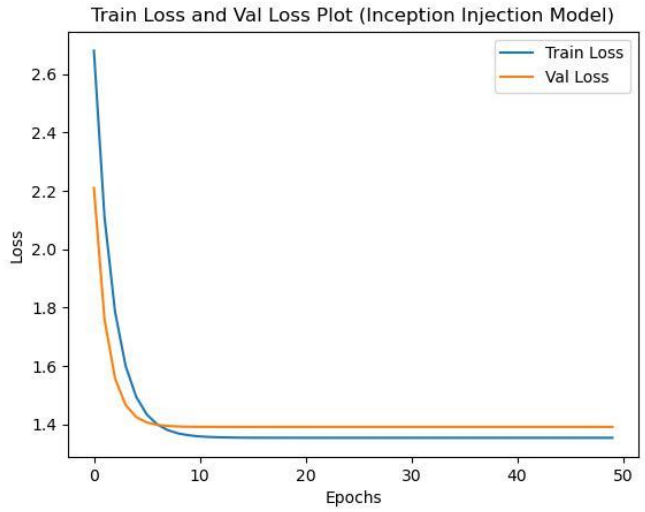


Figure 8: Training Loss and Validation Loss plots for the Injection Model, which uses InceptionV3 for the image encodings, a single-layer LSTM as the RNN decoder, and GloVe word embeddings

Fig. 8 shows the loss plots for the injection model using InceptionV3 encodings. We see the model converges within 10 epochs and performs well on both the training data and the validation data as there is little to no overfitting. Using the trained models, captions were predicted for the test dataset, and the results are shared. Fig. 9 below shows the histogram of the BLEU-4 scores of 10,000 random test images using the Merge Model (GloVe + LSTM)

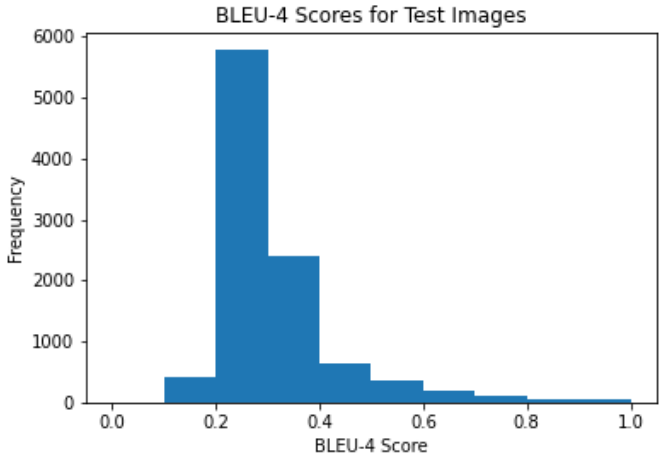


Figure 9: Histogram of BLEU-4 scores for 10,000 test images. The model used for inference is the Merge Model (GloVe + LSTM). 10,000 images were used as computations resources were limited to do inference on the complete test dataset.

The histogram shared above reveals that the most frequently occurring BLEU-4 scores are between 0.2 and 0.4 for the model, meaning the model performs well.

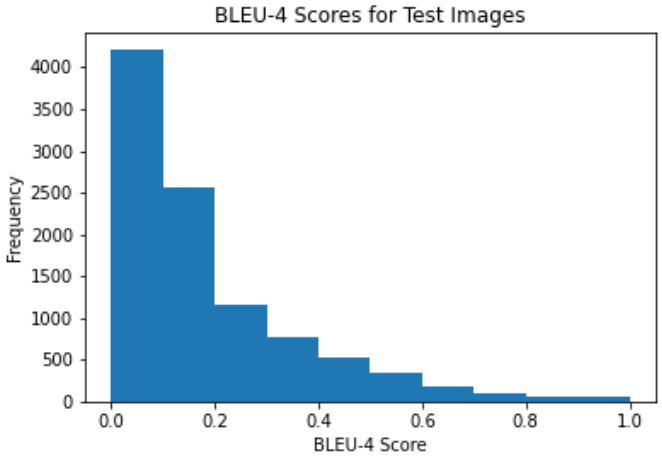


Figure 10: Histogram of BLEU-4 scores for 10,000 test images. The model used for inference is the Merge Model (Self Embedding + GRU). 10,000 images were used as computations resources were limited to do inference on the complete test dataset.

The histogram plot above shows that the most frequently occurring BLEU-4 scores for the test images are between 0.0 and 0.2, which is not ideal.

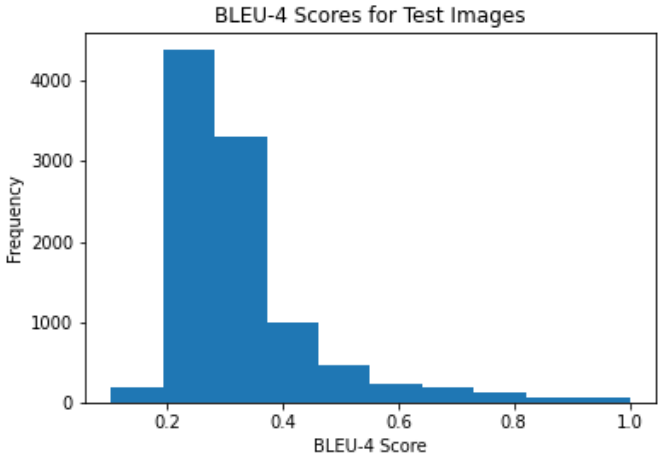


Figure 11: Histogram of BLEU-4 scores for 10,000 test images. The model used for inference is the Injection Model (GloVe + LSTM). 10,000 images were used as computations resources were limited to do inference on the complete test dataset.

The histogram plot shared above shows that the most frequently occurring BLEU-4 scores for the test images using the specific model mentioned in the caption are between 0.2 and 0.4, which is really good.

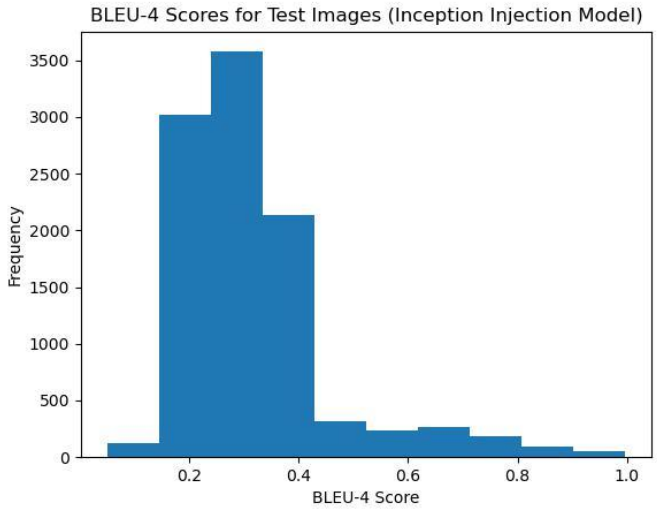


Figure 12: Histogram of BLEU-4 scores for 10,000 test images. The model used for inference is the Injection Model (InceptionV3+ GloVe + LSTM). 10,000 images were used as computations resources were limited to do inference on the complete test dataset.

The histogram plot above shows that the most frequently occurring BLEU-4 scores for the test images are between 0.2 and 0.4, which is good. In addition to the BLEU-4 score, BLEU-3, BLEU-2, and BLEU-1 scores were also calculated for the test images, and an average of all these scores for each of the trained models was determined. Using these averages, these different models can be compared in terms of their performance. Table 2 below shows the BLEU scores for the implemented models and a few state-of-the-art models implemented recently.

Table 2. Comparison of BLEU-4, BLEU-3, BLEU-2 and BLEU-1 scores between the different implemented models and some state-of-the-art models implemented recently.

BLEU Scores				
	BLEU-1	**BLEU-2**	**BLEU-3**	**BLEU-4**
Merge Model (GloVe + LSTM)	0.628	0.456	0.370	0.315
Merge Model (Self Embedding + GRU)	0.641	0.439	0.290	0.196
Injection Model (InceptionV3+GloVe + LSTM)	0.658	0.472	0.379	0.323
Injection Model (GloVe + LSTM)	**0.661**	**0.482**	**0.388**	**0.328**
Oscar (state-of-the-art) [20]	-	-	-	0.417
RDN (state-of-the-art) [21]	0.775	0.618	0.479	0.368

From the BLEU scores, it is evident that three of the implemented models have performed very well, and are comparable to some state-of-the-art models.

Table 3 below shows three random images selected from the test dataset, their corresponding generated captions by each of the four models, and one of the original captions.

Table 3. Three random images were selected from the test dataset. Captions generated by each of the four models for these images are shown. One of the original captions of the images is also shown. Showing all the original captions would make the table cluttered.

Images			
Merge Model (GloVe + LSTM)	a group of people playing soccer in a field	a man riding a wave on top of a surfboard	a zebra standing in a field with a x_UNK_ x_UNK_
Merge Model (Self Embedding + GRU)	a group of young men playing soccer in a field	a man riding a wave on top of a surfboard	a zebra standing in a field of grass
Injection Model (InceptionV3 +GloVe + LSTM)	a group of boys playing soccer in a field	a surfer riding a wave on a surfboard	a group of zebras standing in a field
Injection Model (GloVe + LSTM)	a group of young boys playing soccer in a field	a man riding a wave on a surfboard	a group of zebras standing in a field
Attention Model	a group of young girls playing a game	a surfer in the air while sitting on a surfboard	a group of an adult zebra are standing in a field
Original Captions	a group of young children on a field playing soccer	a man riding a wave on a surfboard	a group of zebras grazing in a field

Discussion

From our results, we can see that all models have a decent performance when compared to more complex state-of-the-art models such as Oscar and RDN which

are trained on millions of images. The results for the merge models show that they do not overfit, and could be trained further, although there would be slow learning. These models were not trained further as computation resources were limited. The merge model with GRU and trainable embeddings does not perform as well as the merge model with LSTM and GloVe embeddings, and that is to be expected. GloVe captures meaningful relationships between the vocabulary words, and maps them onto a vector space in the best way possible, so it’s performance is better than training the word embeddings from scratch.

The attention model, expected to be the best, was surprising with the early overfitting. Looking at the results in Table 3 as well as in Appendix A. we see that the attention model recognizes the individual objects with ease but fails to establish the correct relation between them. If we had the resources, we would have trained the model for longer on the full dataset in order to see its full capability. The injection models perform exceptionally well and give the highest average BELU scores. Currently, the best model is the injection model with two recurrent layers. This is because the LSTM layers use the image encodings as well as the previous captions to predict the next word in the caption. One of the places where this model fails to provide good results is when the image has multiple objects (usually more than three). The model can’t identify the objects properly, and predict a caption that is semantically meaningful.

From our efforts during this project, we experience why the task of image captioning is a difficult one. Research into the field has produced a structure for models for image captioning (encoder-decoder) but the variation of models used for these two vary a lot. Recent advancements in more complex attention models such as multi-headed attention may give better results but they require computational power that wasn't available to us.

Links to models

Injection Model (GloVe + LSTM) (best model):
https://drive.google.com/file/d/1w7DZPQY8yHTqjZbZQ4k5AMsbyF4xLxxY/view?usp=sharing

Injection Model(InceptionV3+GloVe+LSTM):
https://drive.google.com/file/d/1xz_jrm_2bM4WndieeJsZgl0WcXkvCz_B/view?usp=sharing

Merge Model (GloVe + LSTM):
https://drive.google.com/file/d/1yxX9sRe_7v2dx58_hBwhizDd58LMQqsi/view?usp=sharing

Merge Model (Self Embeddings + GRU):
https://drive.google.com/file/d/1FHLyjzqtP0A-HJtsWWPQDvMBzql3f3tg/view?usp=sharing

Attention Model:
https://drive.google.com/file/d/12xPlqOnYhPxFVP0QjBOZLsOcJB-eb4kF/view?usp=sharing

References

[1] Bharath K, “Image Captioning With TensorFlow And Keras”, *PaperspaceBlog*, Published: Oct 2021, https://blog.paperspace.com/image-captioning-with-tensorflow/#building-the-final-model-architecture

[2] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhutdinov, R. Zemel, Y. Bengio, “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”, *Proceedings of the 32nd International Conference on Machine Learning*, in *Proceedings of Machine Learning Research,* 2015, arXiv:1502.03

[3] J. Mao, W. Xu, Y. Yang, J. Wang, Z. Huang, A. Yuille, “Deep Captioning with Multimodal Recurrent Neural Networks (m-RNN)”, *International Conference on Learning Representations*, 2015, arXiv:1412.66

[4] J. Brownlee, “Caption Generation with the Inject and Merge Encoder-Decoder Models”, *Machine Learning Mastery*, Last Updated: 7th Aug 2019, https://machinelearningmastery.com/caption-generation-inject-merge-architectures-encoder-decoder-model/

[5] J. Pennington, R. Socher, and C. D. Manning. 2014. “GloVe: Global Vectors for Word Representation”, https://nlp.stanford.edu/projects/glove/

[6] “Attention Models”, DeepAI, https://deepai.org/machine-learning-glossary-and-terms/attention-models

[7] A. Singh, “Brief Introduction to Attention Models”, *towards data science*, Published: 6th Sept 2019, https://towardsdatascience.com/attention-networks-c735befb5e9f

[8] B. Shmueli, “NLP Metrics Made Simple: The BLEU Score”, *towards data science*, Published: 8th Feb 2021, https://towardsdatascience.com/nlp-metrics-made-simple-the-bleu-score-b06b14fbdbc1

[9] T. Ganegedara, “Intuitive Guide to Understanding GloVe Embeddings”, *towards data science*, Published: 5th May 2019, https://towardsdatascience.com/light-on-math-ml-intuitive-guide-to-understanding-glove-embeddings-b13b4f19c010

[10] F. Chollet, “Using pre-trained word embeddings”, *Keras*, Last Modified: 5th May 2020, https://keras.io/examples/nlp/pretrained_word_embeddings/

[11] P .Dwivedi, “Understanding and Coding a ResNet in Keras”, *towards data science*, Published: 4th Jan 2019, https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33

[12] PyTorch Team, “RESNET”, pyTorch, https://pytorch.org/hub/pytorch_vision_resnet/

[13] M. Tanti, A. Gatt, K. P. Camilleri, “Where to put the Image in an Image Caption Generator”, *University of Malta*, 2018, https://arxiv.org/pdf/1703.09137.pdf

[14] T. Shen, “GRUs vs. LSTMs”, *Medium,* 11th Aug 2017,

https://medium.com/paper-club/grus-vs-lstms-e9d8e2484848

[15] S. Hussain, "Automated Image Captioning (Flickr8)", *Kaggle, https://www.kaggle.com/shadabhussain/automated-image-captioning-flickr8*

[16] H.Lamba, "Image Captioning with Keras", towards data science, Published: 4th Nov 2018, https://towardsdatascience.com/image-captioning-with-keras-teaching-computers-to-describe-pictures-c88a46a311b8

[17] "Image captioning with visual attention", *Tensorflow Core: tutorials,* https://www.tensorflow.org/tutorials/text/image_captioning#model

[18] J. Brownlee, "What is Teacher Forcing for Recurrent Neural Networks?", *Machine Learning Mastery*, Last Updated: 6th Dec 2017, https://machinelearningmastery.com/teacher-forcing-for-recurrent-neural-networks/

[19] J. Brownlee, "A Gentle Introduction to Calculating the BLEU Score for Text in Python", *Machine Learning Mastery*, Last Updated: 19th Dec 2019, https://machinelearningmastery.com/calculate-bleu-score-for-text-python/

[20] X. Li , X. Yin, C. Li, P. Zhang, X. Hu, L. Zhang, L. Wang, H. Hu, L. Dong, F. Wei, Y. Choi , and J. Gao, "Oscar: Object-Semantics Aligned Pre-training for Vision-Language Tasks", *Microsoft Corporation, University of Washington,* April 2015. arXiv:2004.06165

[21] L. Ke, W. Pei, R. Li, X. Shen, and Y. Tai, "Reflective Decoding Network for Image Captioning," *in Proceedings of International Conference of Computer Vision,* 2019. arXiv:1908.11824

Appendix

Appendix A: Supplementary results

Merge Model (GloVe + LSTM)

```
Predicted Caption:
 a group of people playing baseball on a field
Original Captions:
a group of school children posing xFor a picture
the large group of children are outside posing xFor a photo
a group of children pose xFor a x_UNK_ picture
a picture of a lot of girls and boys in front of a building
a group of young children sitting next to each other
Bleu score: 0.2342054179517238
```

```
Predicted Caption:
 a plate of food with a x_UNK_ and a x_UNK_
Original Captions:
a close up of food with broccoli and peppers
some broccoli and meat in a white bowl and sauce
the meat is covered with x_UNK_ and vegetables
a plate topped with broccoli chicken covered in sauce
some meat that is x_UNK_ in with some vegetables
Bleu score: 0.29447798363548333
```

```
Predicted Caption:
 a plane flying through the sky with a blue sky
Original Captions:
a bird flies in front of a x_UNK_ engine x_UNK_
a large airplane flying through a blue sky
an airplane x_UNK_ x_UNK_ in very blue skies
the airplane is flying in the clear blue sky
an airplane flying in the sky near a bird
Bleu score: 0.29447798363548333
```

```
Predicted Caption:
 a bathroom with a toilet and a sink
Original Captions:
its a bathroom with a toilet sink and a shower
a white toilet sitting next to a white sink
a bath room with a toilet a shower and a sink
a toilet with a sink and a shower area
a bathroom with a toilet sink and shower
Bleu score: 0.691441569283882
```

Predicted Caption:
 a man and woman are standing in a x_UNK_
Original Captions:
two young kids x_UNK_ in the x_UNK_ store produce x_UNK_
people and a dog in a x_UNK_ in the fruit x_UNK_
two kids play in the fresh produce x_UNK_ of a busy x_UNK_ store
a family x_UNK_ at a x_UNK_ x_UNK_ with their dog
several people and a dog stand in front of a fruit stand
Bleu score: 0.2215998685042953

Predicted Caption:
 a man is swinging a tennis racket at a ball
Original Captions:
a man in a white tennis outfit hitting a tennis ball
a tennis player x_UNK_ the ball using a x_UNK_
a man holding a tennis racquet on a tennis court
a person is getting ready to hit a tennis ball
Bleu score: 0.2914039837679991

Predicted Caption:
 a man riding skis down a snow covered slope
Original Captions:
a man with one foot in the x_UNK_ on a snow board
a man smiling xWhile standing in the snow with one foot on his snowboard
a snowboarder on a mountain posing xFor a picture
a man standing on top of a ski slope
a man is standing on a snowboard in the snow
Bleu score: 0.280123621768958O5

Merge Model (Self Embedding + GRU)

Predicted Caption:
a group of people standing around a x_UNK_
Original Captions:
a group of school children posing xFor a picture
the large group of children are outside posing xFor a photo
a group of children pose xFor a x_UNK_ picture
a picture of a lot of girls and boys in front of a building
a group of young children sitting next to each other
Bleu score: 0.26604951461486437

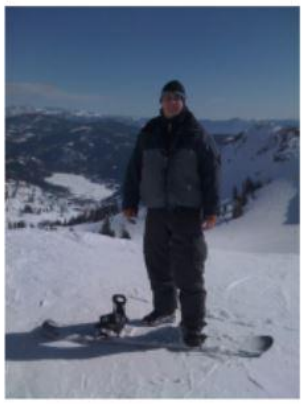
Predicted Caption:
a plate of food with x_UNK_ and broccoli
Original Captions:
a close up of food with broccoli and peppers
some broccoli and meat in a white bowl and sauce
the meat is covered with x_UNK_ and vegetables
a plate topped with broccoli chicken covered in sauce
some meat that is x_UNK_ in with some vegetables
Bleu score: 0.4581407929310677

Predicted Caption:
a plane flying through the air with a sky background
Original Captions:
a bird flies in front of a x_UNK_ engine x_UNK_
a large airplane flying through a blue sky
an airplane x_UNK_ x_UNK_ in very blue skies
the airplane is flying in the clear blue sky
an airplane flying in the sky near a bird
Bleu score: 0.23577691904498785

Predicted Caption:
a bathroom with a toilet and a shower
Original Captions:
its a bathroom with a toilet sink and a shower
a white toilet sitting next to a white sink
a bath room with a toilet a shower and a sink
a toilet with a sink and a shower area
a bathroom with a toilet sink and shower
Bleu score: 0.691441569283882

Predicted Caption:
 a man and a woman are petting a cow
Original Captions:
two young kids x_UNK_ in the x_UNK_ store produce x_UNK_
people and a dog in a x_UNK_ in the fruit x_UNK_
two kids play in the fresh produce x_UNK_ of a busy x_UNK_ store
a family x_UNK_ at a x_UNK_ x_UNK_ with their dog
several people and a dog stand in front of a fruit stand
Bleu score: 0.18551332268245557

Predicted Caption:
 a man is playing tennis on a court
Original Captions:
a man in a white tennis outfit hitting a tennis ball
a tennis player x_UNK_ the ball using a x_UNK_
a man holding a tennis racquet on a tennis court
a person is getting ready to hit a tennis ball
Bleu score: 0.2825111076148885



Predicted Caption:
 a man on a snowboard in the snow
Original Captions:
a man with one foot in the x_UNK_ on a snow board
a man smiling xWhile standing in the snow with one foot on his snowboard
a snowboarder on a mountain posing xFor a picture
a man standing on top of a ski slope
a man is standing on a snowboard in the snow
Bleu score: 0.6752918218126556

Injection Model (GloVe + LSTM)

```
Predicted Caption:
 a group of men in uniform posing xFor a picture
Original Captions:
a group of school children posing xFor a picture
the large group of children are outside posing xFor a photo
a group of children pose xFor a x_UNK_ picture
a picture of a lot of girls and boys in front of a building
a group of young children sitting next to each other
Bleu score: 0.392814650900513
```

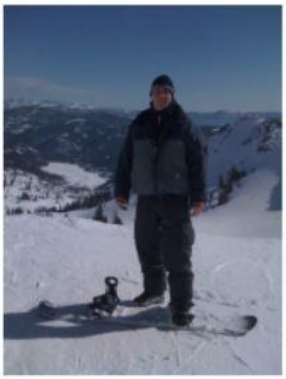
```
Predicted Caption:
 a plate of food with broccoli and x_UNK_
Original Captions:
a close up of food with broccoli and peppers
some broccoli and meat in a white bowl and sauce
the meat is covered with x_UNK_ and vegetables
a plate topped with broccoli chicken covered in sauce
some meat that is x_UNK_ in with some vegetables
Bleu score: 0.6147881529512643
```

```
Predicted Caption:
 a large airplane flying through the air with a x_UNK_ x_UNK_
Original Captions:
a bird flies in front of a x_UNK_ engine x_UNK_
a large airplane flying through a blue sky
an airplane x_UNK_ x_UNK_ in very blue skies
the airplane is flying in the clear blue sky
an airplane flying in the sky near a bird
Bleu score: 0.4497332084013506
```

```
Predicted Caption:
 a bathroom with a toilet and a sink
Original Captions:
its a bathroom with a toilet sink and a shower
a white toilet sitting next to a white sink
a bath room with a toilet a shower and a sink
a toilet with a sink and a shower area
a bathroom with a toilet sink and shower
Bleu score: 0.691441569283882
```

Predicted Caption:
a man and a woman x_UNK_ a cow
Original Captions:
two young kids x_UNK_ in the x_UNK_ store produce x_UNK_
people and a dog in a x_UNK_ in the fruit x_UNK_
two kids play in the fresh produce x_UNK_ of a busy x_UNK_ store
a family x_UNK_ at a x_UNK_ x_UNK_ with their dog
several people and a dog stand in front of a fruit stand
Bleu score: 0.18227693048255034

Predicted Caption:
a man holding a tennis racquet on a tennis court
Original Captions:
a man in a white tennis outfit hitting a tennis ball
a tennis player x_UNK_ the ball using a x_UNK_
a man holding a tennis racquet on a tennis court
a person is getting ready to hit a tennis ball
Bleu score: 1.0



Predicted Caption:
a man on a snowboard in the snow
Original Captions:
a man with one foot in the x_UNK_ on a snow board
a man smiling xWhile standing in the snow with one foot on his snowboard
a snowboarder on a mountain posing xFor a picture
a man standing on top of a ski slope
a man is standing on a snowboard in the snow
Bleu score: 0.6752918218126556

Injection Model (Inception V3 + GloVe Embeddings+LSTM)

Predicted Caption:
a group of children in uniform posing xFor a picture
Original Captions:
a group of school children posing xFor a picture
the large group of children are outside posing xFor a photo
a group of children pose xFor a x_UNK_ picture
a picture of a lot of girls and boys in front of a building
a group of young children sitting next to each other

Predicted Caption:
a plate of food with chicken and broccoli
Original Captions:
a close up of food with broccoli and peppers
some broccoli and meat in a white bowl and sauce
the meat is covered with x_UNK_ and vegetables
a plate topped with broccoli chicken covered in sauce
some meat that is x_UNK_ in with some vegetables

Predicted Caption:
a plane flying in the sky with a x_UNK
Original Captions:
a bird flies in front of a x_UNK_ engine x_UNK_
a large airplane flying through a blue sky
an airplane x_UNK_ x_UNK_ in very blue skies
the airplane is flying in the clear blue sky
an airplane flying in the sky near a bird

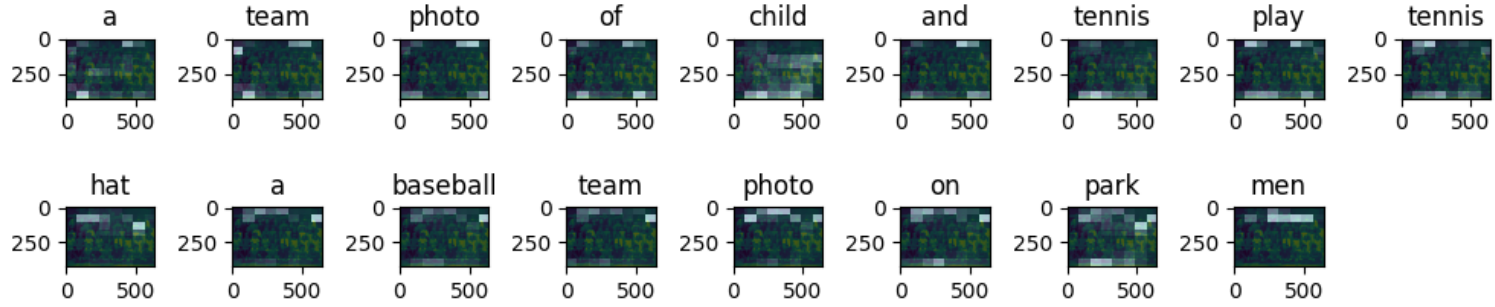
Predicted Caption:
a bathroom with a shower and sink
Original Captions:
its a bathroom with a toilet sink and a shower
a white toilet sitting next to a white sink
a bath room with a toilet a shower and a sink
a toilet with a sink and a shower area
a bathroom with a toilet sink and shower

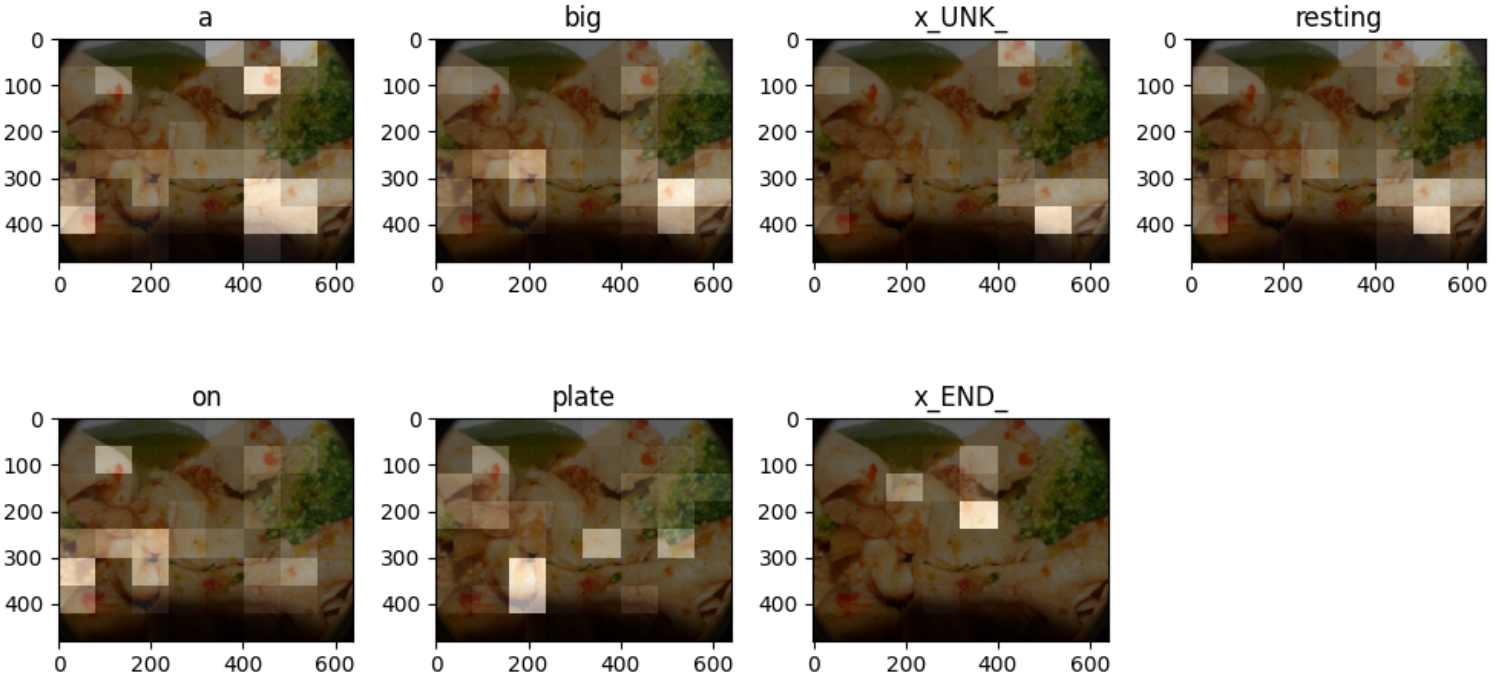
Predicted Caption:
two young kids play with their dog
Original Captions:
two young kids x_UNK_ in the x_UNK_ store produce x_UNK_
people and a dog in a x_UNK_ in the fruit x_UNK_
two kids play in the fresh produce x_UNK_ of a busy x_UNK_ store
a family x_UNK_ at a x_UNK_ x_UNK_ with their dog
several people and a dog stand in front of a fruit stand

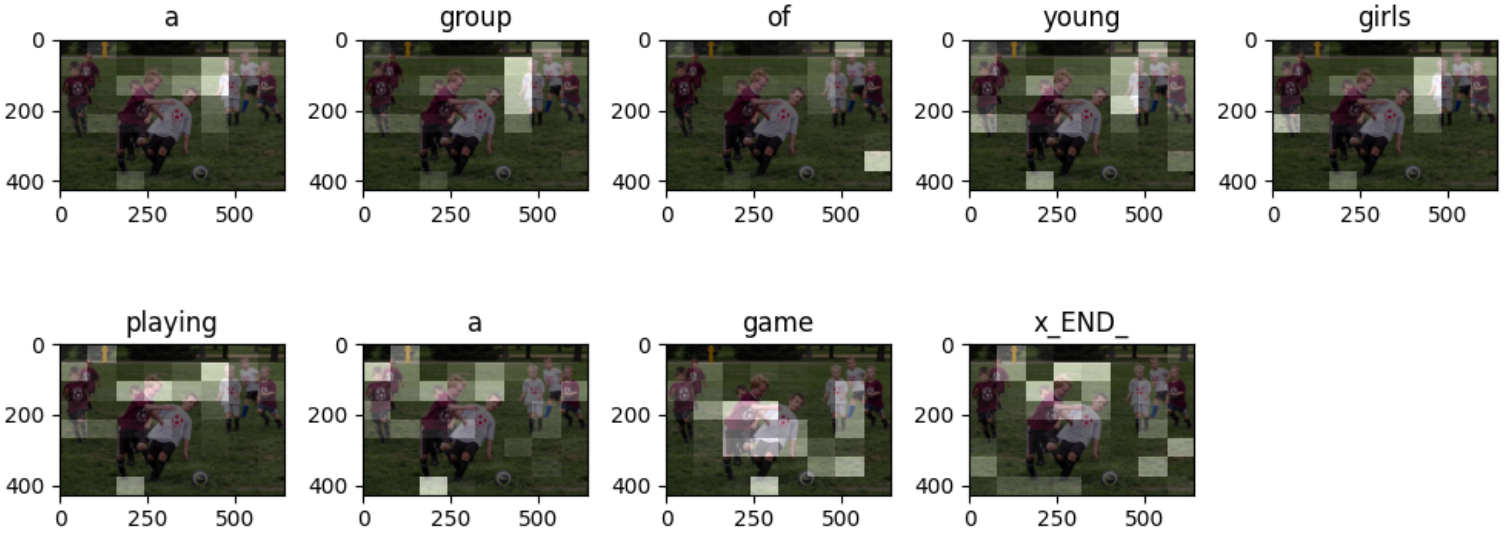
Predicted Caption:
a man holding a tennis racquet hitting a tennis ball
Original Captions:
a man in a white tennis outfit hitting a tennis ball
a tennis player x_UNK_ the ball using a x_UNK_
a man holding a tennis racquet on a tennis court
a person is getting ready to hit a tennis ball

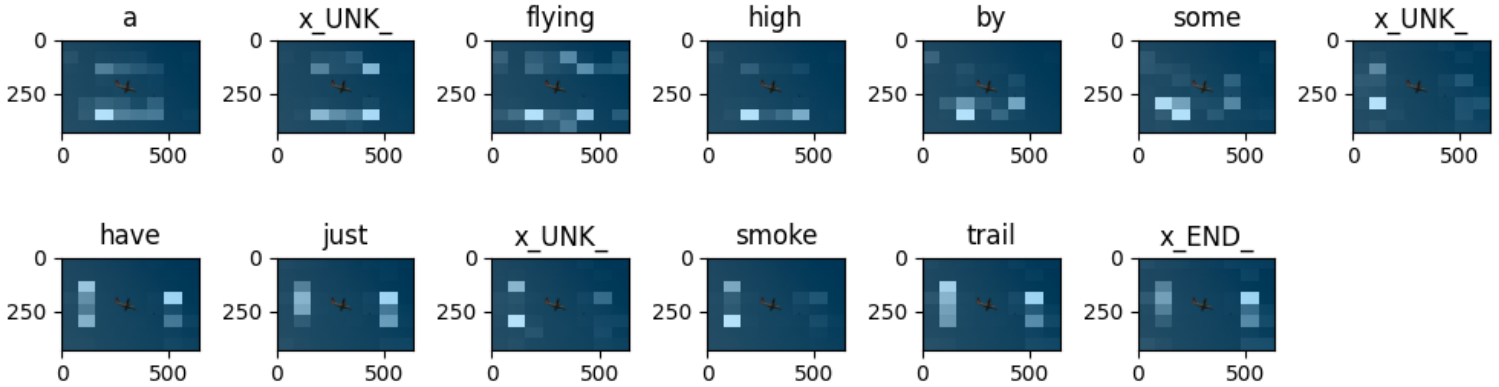
Predicted Caption:
a snowboarder standing in the snow
Original Captions:
a man with one foot in the x_UNK_ on a snow board
a man smiling xWhile standing in the snow with one foot on his snowboard
a snowboarder on a mountain posing xFor a picture
a man standing on top of a ski slope
a man is standing on a snowboard in the snow

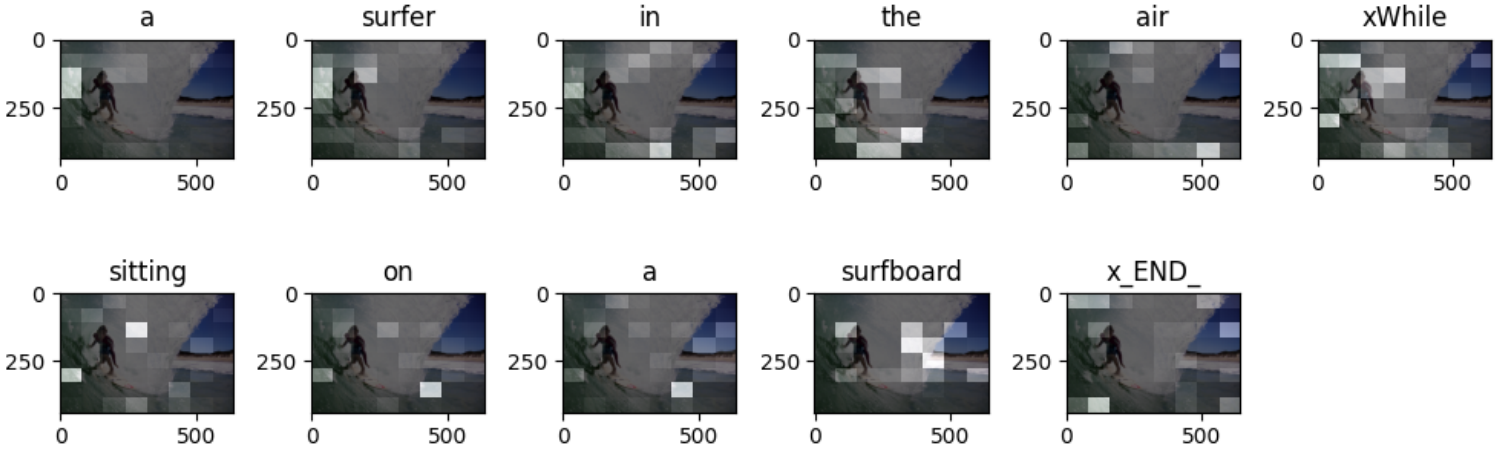
Attention Model

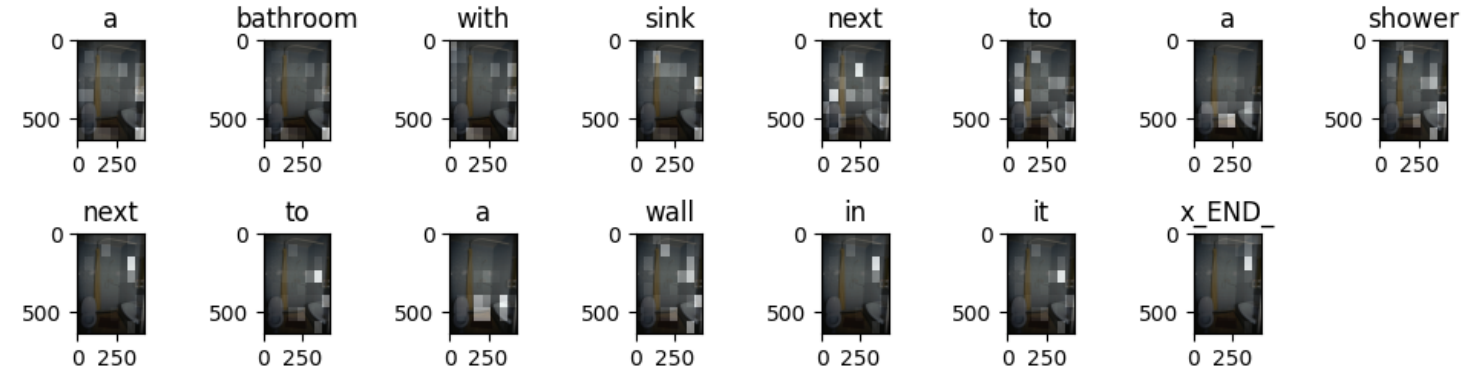

a
team
photo
of
child
and
tennis
play
tennis
hat
a
baseball
team
photo
on
park
men

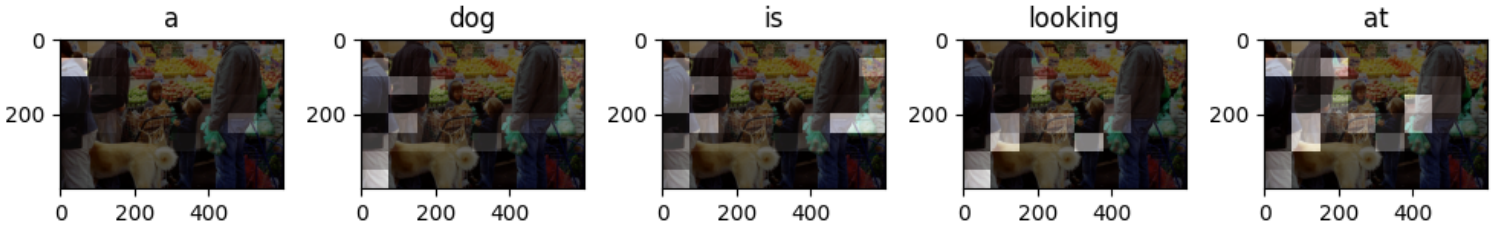

a
big
x_UNK_
resting
on
plate
x_END_

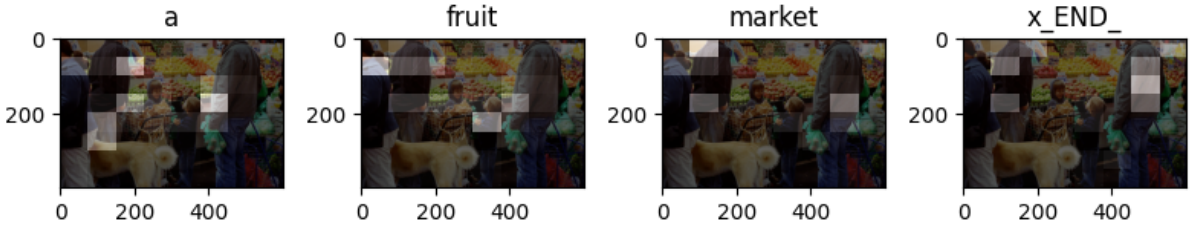

a
group
of
young
girls
playing
a
game
x_END_

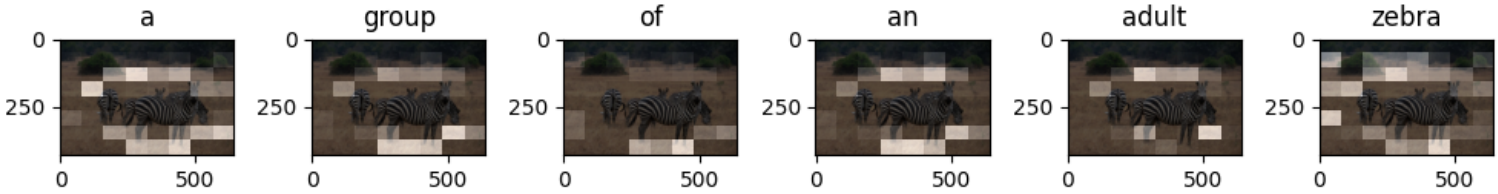

a
x_UNK_
flying
high
by
some
x_UNK_
have
just
x_UNK_
smoke
trail
x_END_

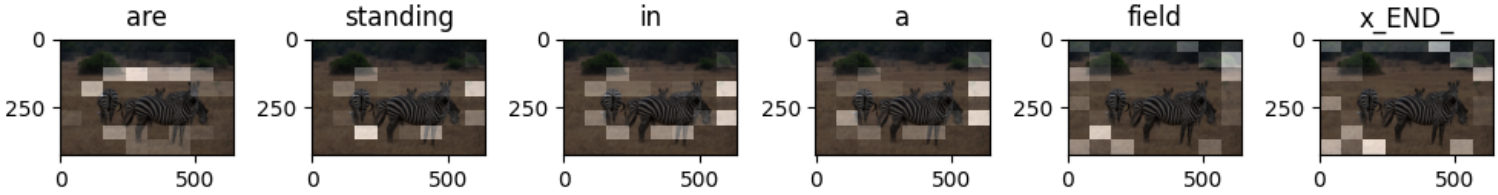

a
surfer
in
the
air
xWhile
sitting
on
a
surfboard
x_END_

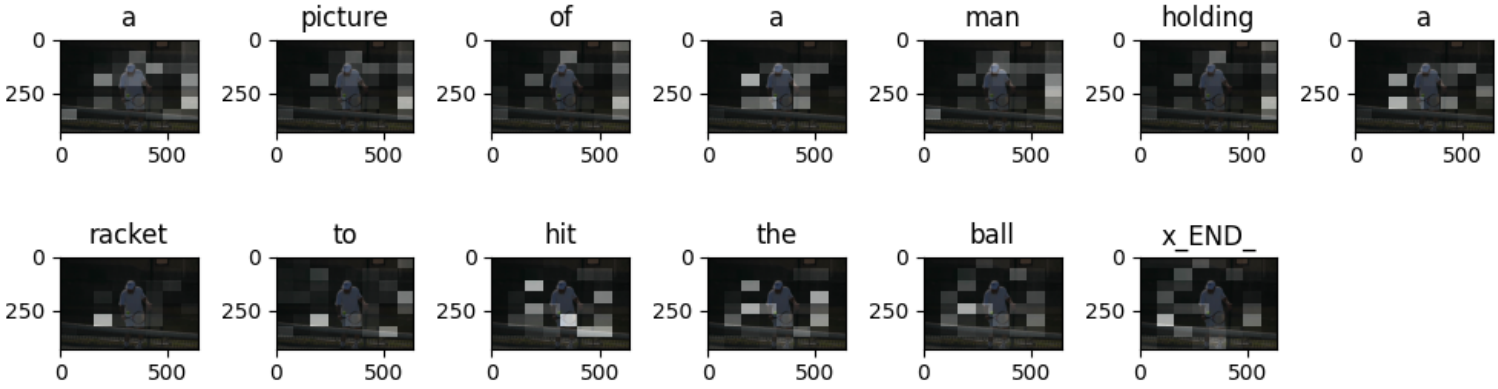

a
bathroom
with
sink
next
to
a
shower
next
to
a
wall
in
it
x_END_

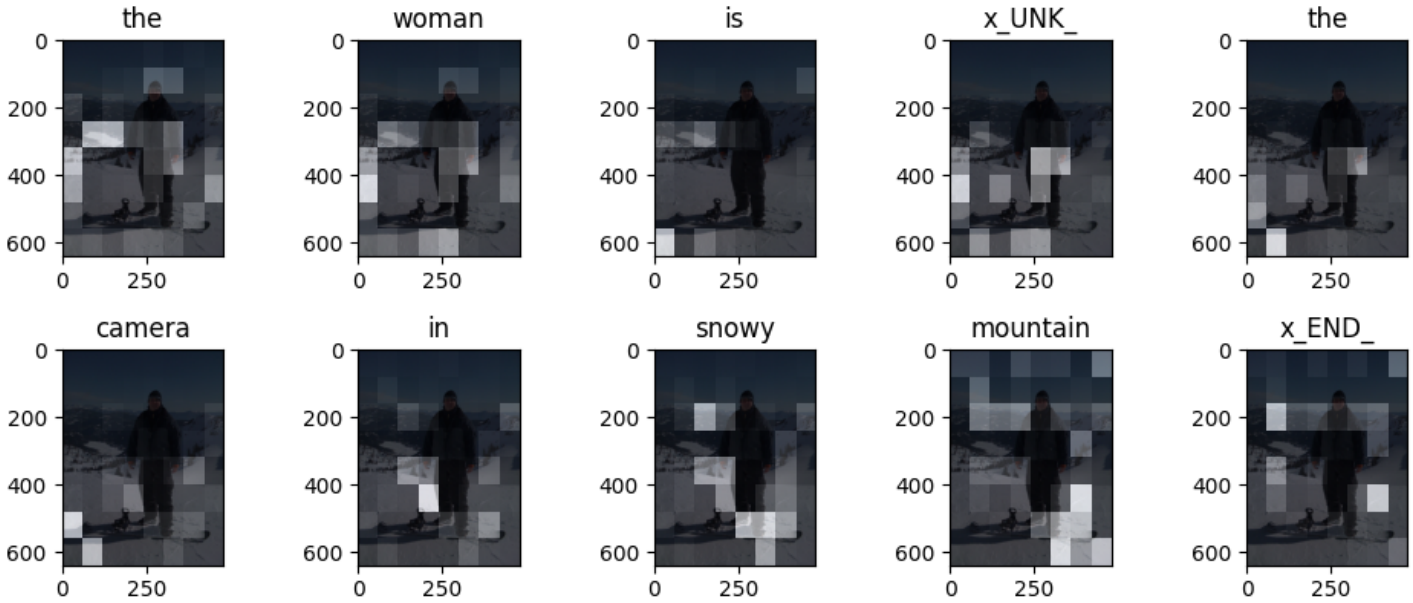

a
dog
is
looking
at


a
fruit
market
x_END_


a
group
of
an
adult
zebra


are
standing
in
a
field
x_END_


a
picture
of
a
man
holding
a
racket
to
hit
the
ball
x_END_


the
woman
is
x_UNK_
the
camera
in
snowy
mountain
x_END_

Appendix B: Code

Preprocessing

```
import matplotlib.pyplot as plt
import pandas as pd
import pickle
import numpy as np
import os
import tensorflow as tf
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.preprocessing import image
import random
import h5py
import urllib.request

# loading the data provided on Moodle
train_dataset = h5py.File('eee443_project_dataset_train.h5', 'r')
test_dataset = h5py.File('eee443_project_dataset_test.h5', 'r')

# seperating training captions, train ids, train URLs and word codes to generate
dictionary for converting index-to-word
# test captions, test ids, and test urls are also seperated
train_cap = np.array(train_dataset['train_cap'])
train_imid = np.array(train_dataset['train_imid'])
train_url = np.array(train_dataset['train_url'])
word_code = train_dataset['word_code']

test_cap = np.array(test_dataset['test_caps'])
test_imid = np.array(test_dataset['test_imid'])
test_url = np.array(test_dataset['train_url'])

# images are downloaded using the script below. urllib.request gets the image URL, and
downloads and saves the image locally
# a not_found list is initiliazed as all URLs don't work, and there are some URLs which
give an error
not_found = []
for i in range(len(train_url)):
    url = train_url[i].decode('UTF-8')
    try:
        image = urllib.request.urlopen(url).read()
```

```
        with open(f'train_images/im_{i}' + '.' + url.split('.')[-1], 'wb') as image_file:
            image_file.write(image)
            image_file.close()
    except Exception as e:
        not_found.append(i)

# saving the not_found list for later use
textfile = open("not_found.txt", "w")
for element in not_found:
    textfile.write(str(element) +"\n")
textfile.close()

# train images are downloaded, and now the test images are downloaded in the same manner
# the not_found_test list is initialized to store ids of images which couldn't be
downloaded
not_found_test = []
for i in range(len(test_url)):
    url = test_url[i].decode('UTF-8')
    try:
        image = urllib.request.urlopen(url).read()
        with open(f'test_images/im_{i}' + '.' + url.split('.')[-1], 'wb') as image_file:
            image_file.write(image)
            image_file.close()
    except Exception as e:
        not_found_test.append(i)

# saving the not_found_test list for later use
textfile = open("not_found_test.txt", "w")
for element in not_found_test:
    textfile.write(str(element) +"\n")
textfile.close()

# loading the ResNet50 model for extracting the image encodings
img_model =
ResNet50(include_top=False,weights='imagenet',input_shape=(224,224,3),pooling='avg')

#function for pre-processing the image before feeding into ResNet model
def preprocess_image(path):
  img = image.load_img(path, target_size=(224,224,3))
  img = image.img_to_array(img)
```

```
  img = np.expand_dims(img, axis=0)

  return img

# the image encodings for ResNet are saved in a dictionary so that they can be accessed
directly through the image id
train_data = {}
for i in range(len(train_url)):
  if i in not_found_test:
    continue
  else:
    img_path = f'train_images/im_{i}.jpg'
    img = preprocess_image(img_path)
    img_enc = img_model.predict(img).reshape(2048)
    train_data[i+1] = img_enc

# after the encodings are saved in the dictionary, they are saved locally for later use
with open( "train_encoded_images_ResNet.p", "wb" ) as pickle_f:
    pickle.dump(train_data, pickle_f )

# the same process is followed for the test data
test_data = {}
for i in range(len(test_url)):
  if i in not_found_test:
    continue
  else:
    img_path = f'test_images/im_{i}.jpg'
    img = preprocess_image(img_path)
    img_enc = img_model.predict(img).reshape(2048)
    test_data[i+1] = img_enc
```

Merge Model (GloVe + LSTM) [1]

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dropout, Dense, Embedding, LSTM, add, Input
from tensorflow.keras.initializers import Constant

def LSTM_GLoVe_model(emb_matrix):
    input_1 = Input(shape=(2048,))
    img_enc_1 = Dropout(0.5)(input_1)
    img_enc_2 = Dense(128, activation='relu')(img_enc_1)
```

```
    input_2 = Input(shape=(17,))
    cap_enc_1 = Embedding(1004, 300, mask_zero=True,
embeddings_initializer=Constant(emb_matrix), trainable=False)(input_2)
    cap_enc_2 = Dropout(0.5)(cap_enc_1)
    cap_enc_3 = LSTM(128)(cap_enc_2)

    decoder_1 = add([img_enc_2, cap_enc_3])
    decoder_2 = Dense(200, activation='relu')(decoder_1)
    outputs = Dense(1004, activation='softmax')(decoder_2)

    model = Model(inputs=[input_1, input_2], outputs=outputs)

    return model
```

Merge Model (Self Embedding + GRU) [1]

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dropout, Dense, Embedding, GRU, add, Input

def GRU_Trained_model():
    input_1 = Input(shape=(2048,))
    img_enc_1 = Dropout(0.5)(input_1)
    img_enc_2 = Dense(128, activation='relu')(img_enc_1)

    input_2 = Input(shape=(17,))
    cap_enc_1 = Embedding(1004, 300, mask_zero=True)(input_2)
    cap_enc_2 = Dropout(0.5)(cap_enc_1)
    cap_enc_3 = GRU(128)(cap_enc_2)

    decoder_1 = add([img_enc_2, cap_enc_3])
    decoder_2 = Dense(200, activation='relu')(decoder_1)
    outputs = Dense(1004, activation='softmax')(decoder_2)

    model = Model(inputs=[input_1, input_2], outputs=outputs)

    return model
```

Injection Model (Glove + LSTM) [15]

```
from tensorflow.keras.models import Sequential, Model
```

```
from tensorflow.keras.layers import Activation, Dense, Embedding, LSTM, TimeDistributed,
RepeatVector, add, Input
from tensorflow.keras.initializers import Constant

def Inject_LSTM_model(emb_matrix):
    img_model = Sequential(name='Image Model')
    img_model.add(Dense(300, input_shape=(2048,), activation='relu'))
    img_model.add(RepeatVector(17))

    lng_model = Sequential(name='Language Model')
    lng_model.add(Embedding(1004, 300, 17, embeddings_initializer=Constant(emb_matrix),
trainable=False))
    lng_model.add(TimeDistributed(Dense(300)))

    concat = add([img_model.output, lng_model.output])
    x = LSTM(128, return_sequences=True)(concat)
    x = LSTM(512, return_sequences=False)(x)
    x = Dense(1004)(x)
    out = Activation('softmax')(x)
    model = Model(inputs=[img_model.input, lng_model.input], outputs=out,
name='ImageCaptioningModel')

    return model
```

Training

```
import matplotlib.pyplot as plt
import pandas as pd
import pickle
import numpy as np
import tensorflow as tf
from keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.preprocessing import image
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.utils import to_categorical
from LSTMGloVe import LSTM_GLoVe_model
from GRUTrained import GRU_Trained_model
from InjectLSTM import Inject_LSTM_model
import h5py
import os
```

```
# checkiing if some essential paths exists. if they don't exist, make the folders.
model_path = 'LSTM_GloVe_models/'
if os.path.exists(model_path):
    pass
else:
    os.mkdir('LSTM_GloVe_models/')

model_path = 'GRU_Trained_models/'
if os.path.exists(model_path):
    pass
else:
    os.mkdir('GRU_Trained_models/')

model_path = 'Inject_LSTM_models/'
if os.path.exists(model_path):
    pass
else:
    os.mkdir('Inject_LSTM_models/')

# Dataloader function for feeding the data to train the model, the dataloader inputs
sequences of captions for each image [1]
# to the model, and the model will predict the next word
def data_generator(img_ids, cap_dict, enc_imgs, vocab_size, max_length,
num_photos_per_batch):
    X1, X2, y = list(), list(), list()
    n=0
    # loop for ever over images
    while 1:
        for id in img_ids:
            n+=1
            # retrieve the photo feature
            desc_list = cap_dict[id]
            photo = enc_imgs[id]
            for desc in desc_list:
                # encode the sequence
                seq = desc
                # split one sequence into multiple X, y pairs
                for i in range(1, len(seq)):
                    # split into input and output pair
                    in_seq, out_seq = seq[:i], seq[i]
```

```
                    # pad input sequence
                    in_seq = pad_sequences([in_seq], maxlen=max_length)[0]
                    # encode output sequence
                    out_seq = to_categorical([out_seq], num_classes=vocab_size)[0]
                    # store
                    X1.append(photo)
                    X2.append(in_seq)
                    y.append(out_seq)
            # yield the batch data
            if n==num_photos_per_batch:
                yield ([np.array(X1), np.array(X2)], np.array(y))
                X1, X2, y = list(), list(), list()
                n=0

# loading the data provided on Moodle
train_dataset = h5py.File('eee443_project_dataset_train.h5', 'r')
test_dataset = h5py.File('eee443_project_dataset_test.h5', 'r')

# seperating training captions, train ids, train URLs and word codes to generate
dictionary for converting index-to-word
# test captions, test ids, and test urls are also seperated
train_cap = np.array(train_dataset['train_cap'])
train_imid = np.array(train_dataset['train_imid'])
train_url = np.array(train_dataset['train_url'])
word_code = train_dataset['word_code']

test_cap = np.array(test_dataset['test_caps'])
test_imid = np.array(test_dataset['test_imid'])
test_url = np.array(test_dataset['train_url'])

# loading the not_found file for discarding ids from train imids that don't exist
# get values for images that were not found
with open('not_found.txt', 'r') as f:
    lines = f.readlines()

not_found = []
for line in lines:
    not_found.append(int(line.strip())+1)

# deleting the captions and ids where images don't exist
```

```
indexes = np.array([])
for number in not_found:
  indices = np.where(train_imid == number)[0]
  indexes = np.concatenate((indexes, indices)).astype(int)

train_imid_found = np.delete(train_imid, indexes)
train_caps_found = np.delete(train_cap, indexes, axis=0)

# converting the data to a DataFrame so that it can be handled better
image_captions_df = pd.DataFrame()
image_captions_df['image_ids'] = train_imid_found
image_captions_df['captions'] = train_caps_found.tolist()
image_captions_df =
image_captions_df.sort_values('image_ids').reset_index().drop(columns=['index'])

# get dictionary of vocabulary words for index-to-word and word-to-index
word_2_ind = {}
ind_2_word = {}
indexes = np.asarray(word_code[0].tolist()).astype(int)
words = np.asarray(word_code.dtype.names)
for i in range(len(indexes)):
  ind_2_word[int(indexes[i])] = words[i]
  word_2_ind[words[i]] = int(indexes[i])

with open('ind2word.pkl', 'wb') as f:
    pickle.dump(f, ind_2_word)

with open('word2ind.pkl', 'wb') as f:
    pickle.dump(f, word_2_ind)

# load the images and their corresponding caption indexes in a dictionary
tokens_dict_id = {}
for i in range(len(image_captions_df)):
  if image_captions_df['image_ids'].loc[i] in tokens_dict_id:

tokens_dict_id[image_captions_df['image_ids'].loc[i]].append(image_captions_df['captions']
.loc[i])
  else:
    tokens_dict_id[image_captions_df['image_ids'].loc[i]] =
[image_captions_df['captions'].loc[i]]
```

```
# randomly shuffle the image ids for validation and training set, the split is 85:15
img_ids = image_captions_df.image_ids.unique()
np.random.shuffle(img_ids)
train_length = int(len(img_ids)*0.85)
train_img_ids, val_img_ids = img_ids[:train_length], img_ids[train_length:]
len_sentence = len(image_captions_df['captions'][1])

# loading the embedding dictionary for the GloVe embeddings so that the vocabulary words
provided
# can be represented in the vector space. 300 is the embedding dimension.
# the GloVe embeddings used are trained on 400000 vocabulary words
embedding_dict = dict()
glove_file = open('glove.6B.300d.txt', encoding='utf8')
for line in glove_file:
  word = line.split()[0]
  coefs = np.asarray(line.split()[1:], dtype='float32')
  embedding_dict[word] = coefs
glove_file.close()

new_indices = np.argsort(indexes)
new_words = words[new_indices]
new_words_ind = indexes[new_indices]

emb_dim = len(embedding_dict[words[0]])
# the no_go list is mentioned below. this list contains words that are not in the GloVe
embeddings
no_go_list = ['xWhile', 'xEnd', 'x_UNK_', 'xCatch', 'xCase', 'xFor']
vocab_size = len(ind_2_word.keys())

# creating the embedding matrix for the 1004 vocabulary words in our dataset
emb_matrix = np.zeros((vocab_size, emb_dim))
i = 0
for word in new_words:
  if word == 'x_NULL_':
    coef = np.zeros(300)
    coef[100] = 1
    emb_matrix[i] = coef
  elif word == 'x_START_':
    coef = np.zeros(300)
```

```
    coef[0] = 1
    emb_matrix[i] = coef
  elif word == 'x_END_':
    coef = np.zeros(300)
    coef[-1] = 1
    emb_matrix[i] = coef
  elif word == 'x_UNK_':
    coef = np.zeros(300)
    coef[200] = 1
    emb_matrix[i] = coef
  elif word == 'xWhile':
    coef = embedding_dict.get('while')
    emb_matrix[i] = coef
  elif word == 'xEnd':
    coef = embedding_dict.get('end')
    emb_matrix[i] = coef
  elif word == 'xCatch':
    coef = embedding_dict.get('catch')
    emb_matrix[i] = coef
  elif word == 'xCase':
    coef = embedding_dict.get('case')
    emb_matrix[i] = coef
  elif word == 'xFor':
    coef = embedding_dict.get('for')
    emb_matrix[i] = coef
  else:
    coef = embedding_dict.get(word)
    emb_matrix[i] = coef
  i += 1

np.save('emb_matrix.npy', emb_matrix)

# loading the image encodings from ResNet for training the merge and injection models
with open('train_encoded_images_ResNet.p', 'rb') as f:
    enc_imgs = pickle.load(f, encoding="bytes")

# importing our first model for training, this model takes ResNet encodings, and is a
merge model
# the word embeddings for this model are taken from GloVe, a pre-trained word embedding
vector
```

```
LSTM_GLoVe_model = LSTM_GLoVe_model(emb_matrix)

# compiling the model
LSTM_GLoVe_model.compile(loss='categorical_crossentropy', optimizer='adam')

# setting the hyper-parameters for our model
epochs = 100
number_pics_per_batch = 64
train_steps = len(train_img_ids)//number_pics_per_batch
val_steps = len(val_img_ids)//number_pics_per_batch
tf.config.run_functions_eagerly(True)

# callbacks for monitoring the progress of our model
callbacks = [EarlyStopping(monitor='val_loss', patience=4, verbose=1),
             ReduceLROnPlateau(monitor='val_loss', patience=3, factor=0.1, min_lr=0.00001,
verbose=1)]

# training the model
tr_loss = []
val_loss = []
for i in range(epochs):
  train_generator = data_generator(train_img_ids, tokens_dict_id, enc_imgs, vocab_size,
len_sentence, number_pics_per_batch)
  val_generator = data_generator(val_img_ids, tokens_dict_id, enc_imgs, vocab_size,
len_sentence, number_pics_per_batch)
  hist = LSTM_GLoVe_model.fit(train_generator, epochs=1, steps_per_epoch=train_steps,
                  validation_data=val_generator, validation_steps=val_steps,
                  callbacks = [callbacks], shuffle=True, verbose=1)
  tr_loss.append(hist.history['loss'])
  val_loss.append(hist.history['val_loss'])
  np.save('LSTM_GloVe_models/tr_loss.npy', tr_loss)
  np.save('LSTM_GloVe_models/val_loss.npy', val_loss)
  if i % 5 == 0:
    LSTM_GLoVe_model.save('LSTM_GloVe_models/model'+str(i)+'.h5')

LSTM_GLoVe_model.save('LSTM_GloVe_models/model'+str(i)+'.h5')

# importing our second model for training, this model takes ResNet encodings, and is
another merge model
```

```
# the word embeddings for this model are trained from scratch, and a GRU layer is used
instead of a LSTM layer
GRU_Trained_model = GRU_Trained_model()

# compiling the model
GRU_Trained_model.compile(loss='categorical_crossentropy', optimizer='adam')

# setting the hyper-parameters for our model
epochs = 100
number_pics_per_batch = 64
train_steps = len(train_img_ids)//number_pics_per_batch
val_steps = len(val_img_ids)//number_pics_per_batch
tf.config.run_functions_eagerly(True)

# training the model
tr_loss = []
val_loss = []
for i in range(epochs):
  train_generator = data_generator(train_img_ids, tokens_dict_id, enc_imgs, vocab_size,
len_sentence, number_pics_per_batch)
  val_generator = data_generator(val_img_ids, tokens_dict_id, enc_imgs, vocab_size,
len_sentence, number_pics_per_batch)
  hist = GRU_Trained_model.fit(train_generator, epochs=1, steps_per_epoch=train_steps,
                  validation_data=val_generator, validation_steps=val_steps,
                  callbacks = [callbacks], shuffle=True, verbose=1)
  tr_loss.append(hist.history['loss'])
  val_loss.append(hist.history['val_loss'])
  np.save('GRU_Trained_models/tr_loss.npy', tr_loss)
  np.save('GRU_Trained_models/val_loss.npy', val_loss)
  if i % 5 == 0:
    GRU_Trained_model.save('GRU_Trained_models/model'+str(i)+'.h5')

GRU_Trained_model.save('GRU_Trained_models/model'+str(i)+'.h5')

# importing our third model for training, this model also takes ResNet encodings, and is
an injection model
# the word embeddings for this model are taken from GloVe, and two LSTM layers are used
Inject_LSTM_model = Inject_LSTM_model(emb_matrix)

# compiling the model
```

```
Inject_LSTM_model.compile(loss='categorical_crossentropy', optimizer='adam')

# setting the hyper-parameters for our model
epochs = 100
number_pics_per_batch = 64
train_steps = len(train_img_ids)//number_pics_per_batch
val_steps = len(val_img_ids)//number_pics_per_batch
tf.config.run_functions_eagerly(True)

# training the model
tr_loss = []
val_loss = []
for i in range(epochs):
  train_generator = data_generator(train_img_ids, tokens_dict_id, enc_imgs, vocab_size,
len_sentence, number_pics_per_batch)
  val_generator = data_generator(val_img_ids, tokens_dict_id, enc_imgs, vocab_size,
len_sentence, number_pics_per_batch)
  hist = Inject_LSTM_model.fit(train_generator, epochs=1, steps_per_epoch=train_steps,
                  validation_data=val_generator, validation_steps=val_steps,
                  callbacks = [callbacks], shuffle=True, verbose=1)
  tr_loss.append(hist.history['loss'])
  val_loss.append(hist.history['val_loss'])
  np.save('Inject_LSTM_models/tr_loss.npy', tr_loss)
  np.save('Inject_LSTM_models/val_loss.npy', val_loss)
  if i % 5 == 0:
    Inject_LSTM_model.save('Inject_LSTM_models/model'+str(i)+'.h5')

Inject_LSTM_model.save('Inject_LSTM_models/model'+str(i)+'.h5')
```

Testing

```
from tensorflow.keras.models import load_model
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing.sequence import pad_sequences
import pandas as pd
from nltk.translate.bleu_score import corpus_bleu
from nltk.translate.bleu_score import SmoothingFunction
import numpy as np
import matplotlib.pyplot as plt
import pickle
```

```
import os
import h5py

# function to predict a caption of a given image using a certain model
def Image_Caption(img_data, len_sentence, ind_2_word, word_2_ind, model):
    in_txt = 'x_START_'
    for i in range(len_sentence):
        seq = [word_2_ind[w] for w in in_txt.split() if w in word_2_ind]
        seq = pad_sequences([seq], maxlen=len_sentence)
        yhat = model.predict([img_data, seq], verbose=0)
        yhat = np.argmax(yhat)
        word = ind_2_word[yhat]
        in_txt += ' ' + word
        if word == 'x_END_':
            break
    final = in_txt.split()
    final = final[1:-1]
    final_arr = final.copy()
    final = ' '.join(final)

    return final, final_arr

# function to return all the original captions of a given image
def original_caps(df, img_id, ind_2_word):
  caps = df.loc[df['image_ids'] == img_id]
  caps = caps['captions'].values
  all_caps = []
  for cap in caps:
    cap_df = pd.Series(cap)
    cap_df = cap_df.map(ind_2_word)
    cap_arr = np.array(cap_df)
    first_ind, last_ind = 0, np.where(cap_arr == 'x_END_')
    cap_arr = cap_arr[first_ind+1:int(last_ind[0])]
    all_caps.append(list(cap_arr.reshape(-1)))

  return all_caps

# function to plot the image, and the corresponding predicted and original captions
def plot_image_captions(img_no, test_enc_imgs, len_sentence, ind_2_word, word_2_ind, df,
model):
```

```
    test_cap, test_cap_arr = Image_Caption(test_enc_imgs[img_no+1].reshape(1,2048),
len_sentence, ind_2_word, word_2_ind, model)
    all_caps = original_caps(df, img_no, ind_2_word)
    smoothie = SmoothingFunction().method4

    img = plt.imread(f'test_images/im_{img_no}.jpg')
    plt.imshow(img)
    plt.axis('off')
    plt.show()
    score = corpus_bleu([all_caps], [test_cap_arr], smoothing_function=smoothie)
    print("Predicted Caption: \n", test_cap)
    print("Original Captions:")
    for cap in all_caps:
        print(' '.join(cap))
        print(f"Bleu score: {score}")

dataset = h5py.File('eee443_project_dataset_test.h5', 'r')

# load the data
test_cap = np.array(dataset['test_caps'])
test_imid = np.array(dataset['test_imid'])
#train_url = np.array(dataset['train_url'])
#word_code = dataset['word_code']

# get values for images that were not found
with open('not_found_test.txt', 'r') as f:
    lines = f.readlines()

not_found = []
for line in lines:
    not_found.append(int(line.strip()))

with open('test_encoded_images_ResNet.p', 'rb') as f:
    test_enc_imgs = pickle.load(f, encoding="bytes")

indexes = np.array([])
for number in not_found:
  indices = np.where(test_imid == number)[0]
  indexes = np.concatenate((indexes, indices)).astype(int)
```

```
test_imid_found = np.delete(test_imid, indexes)
test_caps_found = np.delete(test_cap, indexes, axis=0)

image_captions_df = pd.DataFrame()
image_captions_df['image_ids'] = test_imid_found
image_captions_df['captions'] = test_caps_found.tolist()
image_captions_df =
image_captions_df.sort_values('image_ids').reset_index().drop(columns=['index'])

test_img_ids = image_captions_df.image_ids.unique()
len(test_img_ids)
np.random.shuffle(test_img_ids)
for_test_ids = test_img_ids[:10000]

ind_2_word = pickle.load(open('ind2word.pkl', 'rb'))
word_2_ind = pickle.load(open('word2ind.pkl', 'rb'))

# load the images and their corresponding caption indexes in a dictionary
test_tokens_dict_id = {}
for i in range(len(image_captions_df)):
  if image_captions_df['image_ids'].loc[i] in test_tokens_dict_id:

test_tokens_dict_id[image_captions_df['image_ids'].loc[i]].append(image_captions_df['capti
ons'].loc[i])
  else:
    test_tokens_dict_id[image_captions_df['image_ids'].loc[i]] =
[image_captions_df['captions'].loc[i]]

len_sentence = 17

# a smoothing function is created so that the bleu score calculated will not over-perform
# which is the case if the smoothing function is not applied
smoothie = SmoothingFunction().method4

# calculating the bleu scores for 1-ngram, 2-ngram, 3-ngram, and 4 n-gram
# and captions for all 10000 random test images with the GRU_trainable emb model
model = load_model('model_GRU_train.h5')
test_scores, predicted_caps = [], {}
Bleu_3, Bleu_2, Bleu_1 = [], [], []
j = 0
```

```
for i in for_test_ids:
  test_cap, test_cap_arr = Image_Caption(test_enc_imgs[i+1].reshape(1,2048), len_sentence,
ind_2_word, word_2_ind, model)
  predicted_caps[i] = test_cap
  all_caps = original_caps(image_captions_df, i, ind_2_word)
  score = corpus_bleu([all_caps], [test_cap_arr], smoothing_function=smoothie)
  test_scores.append(score)
  score_1 = corpus_bleu([all_caps], [test_cap_arr], smoothing_function=smoothie,
weights=(0.33, 0.33, 0.33, 0))
  Bleu_3.append(score_1)
  score_2 = corpus_bleu([all_caps], [test_cap_arr], smoothing_function=smoothie,
weights=(0.5, 0.5, 0, 0))
  Bleu_2.append(score_2)
  score_3 = corpus_bleu([all_caps], [test_cap_arr], smoothing_function=smoothie,
weights=(1.0, 0, 0, 0))
  Bleu_1.append(score_3)
  if j % 1000 == 0:
    print(j)
  j += 1

tr_loss = np.load('LSTM_GloVe_models/tr_loss.npy')
val_loss = np.load('LSTM_GloVe_models/val_loss.npy')

plt.plot(tr_loss)
plt.plot(val_loss)
plt.legend(['Train Loss', 'Val Loss'])
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Train Loss and Val Loss Plot (GRU Trainable Emb)')
plt.show()

plt.hist(test_scores)
plt.xlabel('BLEU-4 Score')
plt.ylabel('Frequency')
plt.title('BLEU-4 Scores for Test Images (GRU Trainable Emb)')
plt.show()

# calculating the bleu scores for 1-ngram, 2-ngram, 3-ngram, and 4 n-gram
# and captions for all 10000 random test images with the LSTM_glove model
model = load_model('model_resnet_lstm_glove.h5')
```

```
test_scores, predicted_caps = [], {}
Bleu_3, Bleu_2, Bleu_1 = [], [], []
j = 0
for i in for_test_ids:
  test_cap, test_cap_arr = Image_Caption(test_enc_imgs[i+1].reshape(1,2048), len_sentence,
ind_2_word, word_2_ind, model)
  predicted_caps[i] = test_cap
  all_caps = original_caps(image_captions_df, i, ind_2_word)
  score = corpus_bleu([all_caps], [test_cap_arr], smoothing_function=smoothie)
  test_scores.append(score)
  score_1 = corpus_bleu([all_caps], [test_cap_arr], smoothing_function=smoothie,
weights=(0.33, 0.33, 0.33, 0))
  Bleu_3.append(score_1)
  score_2 = corpus_bleu([all_caps], [test_cap_arr], smoothing_function=smoothie,
weights=(0.5, 0.5, 0, 0))
  Bleu_2.append(score_2)
  score_3 = corpus_bleu([all_caps], [test_cap_arr], smoothing_function=smoothie,
weights=(1.0, 0, 0, 0))
  Bleu_1.append(score_3)
  if j % 1000 == 0:
    print(j)
  j += 1

tr_loss = np.load('GRU_Trained_models/tr_loss.npy')
val_loss = np.load('GRU_Trained_models/val_loss.npy')

plt.plot(tr_loss)
plt.plot(val_loss)
plt.legend(['Train Loss', 'Val Loss'])
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Train Loss and Val Loss Plot (LSTM GloVe Emb)')
plt.show()

plt.hist(test_scores)
plt.xlabel('BLEU-4 Score')
plt.ylabel('Frequency')
plt.title('BLEU-4 Scores for Test Images (LSTM GloVe Emb)')
plt.show()
```

```
# calculating the bleu scores for 1-ngram, 2-ngram, 3-ngram, and 4 n-gram
# and captions for all 10000 random test images with the inject LSTM glove model
model = load_model('model_inject_LSTM')
test_scores, predicted_caps = [], {}
Bleu_3, Bleu_2, Bleu_1 = [], [], []
j = 0
for i in for_test_ids:
  test_cap, test_cap_arr = Image_Caption(test_enc_imgs[i+1].reshape(1,2048), len_sentence,
ind_2_word, word_2_ind, model)
  predicted_caps[i] = test_cap
  all_caps = original_caps(image_captions_df, i, ind_2_word)
  score = corpus_bleu([all_caps], [test_cap_arr], smoothing_function=smoothie)
  test_scores.append(score)
  score_1 = corpus_bleu([all_caps], [test_cap_arr], smoothing_function=smoothie,
weights=(0.33, 0.33, 0.33, 0))
  Bleu_3.append(score_1)
  score_2 = corpus_bleu([all_caps], [test_cap_arr], smoothing_function=smoothie,
weights=(0.5, 0.5, 0, 0))
  Bleu_2.append(score_2)
  score_3 = corpus_bleu([all_caps], [test_cap_arr], smoothing_function=smoothie,
weights=(1.0, 0, 0, 0))
  Bleu_1.append(score_3)
  if j % 1000 == 0:
    print(j)
  j += 1

tr_loss = np.load('Inject_LSTM_models/tr_loss.npy')
val_loss = np.load('Inject_LSTM_models/tr_loss.npy')

plt.plot(tr_loss)
plt.plot(val_loss)
plt.legend(['Train Loss', 'Val Loss'])
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Train Loss and Val Loss Plot  (Inject LSTM GloVe Emb)')
plt.show()

plt.hist(test_scores)
plt.xlabel('BLEU-4 Score')
plt.ylabel('Frequency')
```

```
plt.title('BLEU-4 Scores for Test Images (Inject LSTM GloVe Emb)')
plt.show()
```

Attention

```
import os
import tensorflow as tf
import pickle
import numpy as np
import time

# Train script for Attention model from:
https://www.tensorflow.org/tutorials/text/image_captioning#caption


# defining Bahdanau Attention submodel
class BahdanauAttention(tf.keras.Model):
  def __init__(self, units):
    super(BahdanauAttention, self).__init__()
    self.W1 = tf.keras.layers.Dense(units)
    self.W2 = tf.keras.layers.Dense(units)
    self.V = tf.keras.layers.Dense(1)

  def call(self, features, hidden):



    hidden_with_time_axis = tf.expand_dims(hidden, 1)


    attention_hidden_layer = (tf.nn.tanh(self.W1(features) +
                                         self.W2(hidden_with_time_axis)))


    score = self.V(attention_hidden_layer)

    attention_weights = tf.nn.softmax(score, axis=1)

    context_vector = attention_weights * features
    context_vector = tf.reduce_sum(context_vector, axis=1)
```

```
    return context_vector, attention_weights


class CNN_Encoder(tf.keras.Model):
    # Since you have already extracted the features and dumped it
    # This encoder passes those features through a Fully connected layer
    def __init__(self, embedding_dim):
        super(CNN_Encoder, self).__init__()
        # shape after fc == (batch_size, 64, embedding_dim)
        self.fc = tf.keras.layers.Dense(embedding_dim)

    def call(self, x):
        x = self.fc(x)
        x = tf.nn.relu(x)
        return x

class RNN_Decoder(tf.keras.Model):
  def __init__(self, embedding_dim, units, vocab_size):
    super(RNN_Decoder, self).__init__()
    self.units = units

    self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
    self.gru = tf.keras.layers.GRU(self.units,
                                   return_sequences=True,
                                   return_state=True,
                                   recurrent_initializer='glorot_uniform')
    self.fc1 = tf.keras.layers.Dense(self.units)
    self.dropout = tf.keras.layers.Dropout(0.2)
    self.fc2 = tf.keras.layers.Dense(vocab_size)

    self.attention = BahdanauAttention(self.units)

  def call(self, x, features, hidden):
    # defining attention as a separate model
    context_vector, attention_weights = self.attention(features, hidden)

    # x shape after passing through embedding == (batch_size, 1, embedding_dim)
    x = self.embedding(x)
    # x shape after concatenation == (batch_size, 1, embedding_dim + hidden_size)
```

```
    x = tf.concat([tf.expand_dims(context_vector, 1), x], axis=-1)

    # passing the concatenated vector to the GRU
    output, state = self.gru(x)

    # shape == (batch_size, max_length, hidden_size)
    x = self.fc1(output)
    x = self.dropout(x)

    # x shape == (batch_size * max_length, hidden_size)
    x = tf.reshape(x, (-1, x.shape[2]))

    # output shape == (batch_size * max_length, vocab)
    x = self.fc2(x)

    return x, state, attention_weights

  def reset_state(self, batch_size):
    return tf.zeros((batch_size, self.units))


# Import training dataset.
import h5py
filename = "eee443_project_dataset_train.h5"
with h5py.File(filename, "r") as f:
    train_cap = f['train_cap'][:]
    train_imid = f['train_imid'][:]
    word_code = (f['word_code'][:])

# Get the words from the dtypes. sort them so that they are accoring to index
word_dict = word_code.dtype.names
w = np.argsort(word_code.item())
words = [word_dict[i] for i in w]

# change the troubling words
words[55] = 'while'
words[80] = 'for'
words[492] = 'catch'
words[561] = 'case'
words[621] = 'end'
```

```
# make a dictionary of all captions with the training imid as keys
captions_all = {}
for i in range(1,len(train_imid)+1):
  try:
    x = np.where(train_imid == int(i))
    cap1 = (train_cap[x])
    captions_all[i] = cap1
  except:
    print(i)


# Get the list of inception encodings
inception_encs = os.listdir('inception_encodings')
# print(inception_encs)
# inception_encs1 = inception_encs[]

# parameters set in tutorial
BATCH_SIZE = 100
BUFFER_SIZE = 1000
embedding_dim = 256
units = 512
features_shape = 2048
attention_features_shape = 64
vocab_size = 1004

# get the ids of the images for which we have the inception encodings
im_name = []

for i in range(85000):
    im_str = 'im_' + str(i) + '.npy'
    if im_str in inception_encs:
        im_name.append(i)


# Splitting the data into training and validation sets. Not using all the training set
here as not enough resources
# to run the model. Therefore, only 50% of the images for which we have the inception
encodings are used for
# training and 15% are used for validation. training (30393) validation (9118)
```

```
unique_ids = (im_name)
train_ids = unique_ids[:int(np.round(len(unique_ids)*0.50))]
val_ids =
unique_ids[int(np.round(len(unique_ids)*0.50)):int(np.round(len(unique_ids)*0.65))]
print(len(train_ids))
print(len(val_ids))


# get the training im ids and im captions
train_cap = []
train_im_name = []
for i in train_ids:
    for j in range(len(captions_all[i])):
        train_im_name.append(i)
        train_cap.append(captions_all[i][j])



# get the validation im ids and im captions
val_cap = []
val_im_name = []
for i in val_ids:
    for j in range(len(captions_all[i])):
        val_im_name.append(i)
        val_cap.append(captions_all[i][j])


train_num_steps = (len(train_cap)) // BATCH_SIZE
val_num_steps = (len(val_cap)) // BATCH_SIZE



# Load the numpy files
def map_func(img_name, cap):

  img_tensor = np.load('inception_encodings/im_'+str(img_name)+'.npy')
#   print(img_tensor.dtype)
#   print(cap.dtype)
  return img_tensor, cap
```

```
# Create datasets
train_dataset = tf.data.Dataset.from_tensor_slices((train_im_name, train_cap))

train_dataset = train_dataset.map(lambda item1, item2: tf.numpy_function(
          map_func, [item1, item2], [tf.float32, tf.int32]),
          num_parallel_calls=tf.data.AUTOTUNE)

train_dataset = train_dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
train_dataset = train_dataset.prefetch(buffer_size=tf.data.AUTOTUNE)

val_dataset = tf.data.Dataset.from_tensor_slices((val_im_name, val_cap))

val_dataset = val_dataset.map(lambda item1, item2: tf.numpy_function(
          map_func, [item1, item2], [tf.float32, tf.int32]),
          num_parallel_calls=tf.data.AUTOTUNE)

val_dataset = val_dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
val_dataset = val_dataset.prefetch(buffer_size=tf.data.AUTOTUNE)


# initialize models
encoder = CNN_Encoder(embedding_dim)
decoder = RNN_Decoder(embedding_dim, units, vocab_size)


# define optimizer and lpss functions
optimizer = tf.keras.optimizers.Adam()
loss_object = tf.keras.losses.SparseCategoricalCrossentropy(
    from_logits=True, reduction='none')


def loss_function(real, pred):
  mask = tf.math.logical_not(tf.math.equal(real, 0))
  loss_ = loss_object(real, pred)

  mask = tf.cast(mask, dtype=loss_.dtype)
  loss_ *= mask
```

```
  return tf.reduce_mean(loss_)


loss_plot = []

# training step taken from tutorial
@tf.function
def train_step(img_tensor, target):
  loss = 0


  hidden = decoder.reset_state(batch_size=target.shape[0])

  dec_input = tf.expand_dims(tf.ones(1 * target.shape[0], 1),1)

  with tf.GradientTape() as tape:
      features = encoder(img_tensor)

      for i in range(1, target.shape[1]):
          # passing the features through the decoder
          predictions, hidden, _ = decoder(dec_input, features, hidden)

          loss += loss_function(target[:, i], predictions)

          # using teacher forcing
          dec_input = tf.expand_dims(target[:, i], 1)

  total_loss = (loss / int(target.shape[1]))

  trainable_variables = encoder.trainable_variables + decoder.trainable_variables

  gradients = tape.gradient(loss, trainable_variables)

  optimizer.apply_gradients(zip(gradients, trainable_variables))

  return loss, total_loss
```

```
# validation code same as training step but no updates
def validate_step(img_tensor, target):
  loss = 0

  # initializing the hidden state for each batch
  # because the captions are not related from image to image
  hidden = decoder.reset_state(batch_size=target.shape[0])

  dec_input = tf.expand_dims(tf.ones(1 * target.shape[0], 1),1)

  with tf.GradientTape() as tape:
      features = encoder(img_tensor)

      for i in range(1, target.shape[1]):
          # passing the features through the decoder
          predictions, hidden, _ = decoder(dec_input, features, hidden)

          loss += loss_function(target[:, i], predictions)

          # using teacher forcing
          dec_input = tf.expand_dims(target[:, i], 1)

  total_loss = (loss / int(target.shape[1]))

  return loss, total_loss


# saving model checkpoints
checkpoint_path = "./checkpoints/train"
ckpt = tf.train.Checkpoint(encoder=encoder,
                           decoder=decoder,
                           optimizer=optimizer)
ckpt_manager = tf.train.CheckpointManager(ckpt, checkpoint_path, max_to_keep=5)


start_epoch = 0
if ckpt_manager.latest_checkpoint:
  start_epoch = int(ckpt_manager.latest_checkpoint.split('-')[-1])
  # restoring the latest checkpoint in checkpoint_path
  ckpt.restore(ckpt_manager.latest_checkpoint)
```

```
# Main loop from tutorial expanded to accomodate validation.
EPOCHS = 20
val_loss_plot = []
for epoch in range(start_epoch, EPOCHS):
    start = time.time()
    start1 = time.time()
    total_loss = 0
    val_total_loss = 0

    if epoch % 5 == 0:
      ckpt_manager.save()

    for (batch, (img_tensor, target)) in enumerate(train_dataset):

        batch_loss, t_loss = train_step(img_tensor, target)
        total_loss += t_loss

        if batch % 100 == 0:
            average_batch_loss = batch_loss.numpy()/int(target.shape[1])
            print(f'Epoch {epoch+1} Batch {batch} Loss {average_batch_loss:.4f} Time taken
{time.time()-start1:.2f} sec')
            start1 = time.time()
    # storing the epoch end loss value to plot later
    loss_plot.append(total_loss / train_num_steps)



    for (batch, (img_tensor, target)) in enumerate(val_dataset):

        val_batch_loss, val_t_loss = validate_step(img_tensor, target)
        val_total_loss += val_t_loss

        if batch % 100 == 0:
            average_batch_loss = val_batch_loss.numpy()/int(target.shape[1])
            print(f'Epoch {epoch+1} Batch {batch} Loss {average_batch_loss:.4f} Time taken
{time.time()-start1:.2f} sec')
            start1 = time.time()
```

```
    val_loss_plot.append(val_total_loss / val_num_steps)

    np.save('Attention/tr_loss.npy',loss_plot )
    np.save('Attention/val_loss.npy',val_loss_plot )
    print(f'Epoch {epoch+1} training Loss {total_loss/train_num_steps:.6f} validation Loss
{val_total_loss/val_num_steps:.6f}')
    print(f'Time taken for 1 epoch {time.time()-start:.2f} sec\n')
```

```
import os
import tensorflow as tf
import pickle
import numpy as np
import time

# Train script for Attention model from:
https://www.tensorflow.org/tutorials/text/image_captioning#caption


# defining Bahdanau Attention submodel
class BahdanauAttention(tf.keras.Model):
  def __init__(self, units):
    super(BahdanauAttention, self).__init__()
    self.W1 = tf.keras.layers.Dense(units)
    self.W2 = tf.keras.layers.Dense(units)
    self.V = tf.keras.layers.Dense(1)

  def call(self, features, hidden):



    hidden_with_time_axis = tf.expand_dims(hidden, 1)


    attention_hidden_layer = (tf.nn.tanh(self.W1(features) +
```

```
                                        self.W2(hidden_with_time_axis)))

    score = self.V(attention_hidden_layer)

    attention_weights = tf.nn.softmax(score, axis=1)

    context_vector = attention_weights * features
    context_vector = tf.reduce_sum(context_vector, axis=1)

    return context_vector, attention_weights


class CNN_Encoder(tf.keras.Model):
    # Since you have already extracted the features and dumped it
    # This encoder passes those features through a Fully connected layer
    def __init__(self, embedding_dim):
        super(CNN_Encoder, self).__init__()
        # shape after fc == (batch_size, 64, embedding_dim)
        self.fc = tf.keras.layers.Dense(embedding_dim)

    def call(self, x):
        x = self.fc(x)
        x = tf.nn.relu(x)
        return x

class RNN_Decoder(tf.keras.Model):
  def __init__(self, embedding_dim, units, vocab_size):
    super(RNN_Decoder, self).__init__()
    self.units = units

    self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
    self.gru = tf.keras.layers.GRU(self.units,
                                   return_sequences=True,
                                   return_state=True,
                                   recurrent_initializer='glorot_uniform')
    self.fc1 = tf.keras.layers.Dense(self.units)
    self.dropout = tf.keras.layers.Dropout(0.2)
    self.fc2 = tf.keras.layers.Dense(vocab_size)
```

```
    self.attention = BahdanauAttention(self.units)

  def call(self, x, features, hidden):
    # defining attention as a separate model
    context_vector, attention_weights = self.attention(features, hidden)

    # x shape after passing through embedding == (batch_size, 1, embedding_dim)
    x = self.embedding(x)
    # x shape after concatenation == (batch_size, 1, embedding_dim + hidden_size)
    x = tf.concat([tf.expand_dims(context_vector, 1), x], axis=-1)

    # passing the concatenated vector to the GRU
    output, state = self.gru(x)

    # shape == (batch_size, max_length, hidden_size)
    x = self.fc1(output)
    x = self.dropout(x)

    # x shape == (batch_size * max_length, hidden_size)
    x = tf.reshape(x, (-1, x.shape[2]))

    # output shape == (batch_size * max_length, vocab)
    x = self.fc2(x)

    return x, state, attention_weights

  def reset_state(self, batch_size):
    return tf.zeros((batch_size, self.units))



# Import training dataset.
import h5py
filename = "eee443_project_dataset_train.h5"
with h5py.File(filename, "r") as f:
    train_cap = f['train_cap'][:]
    train_imid = f['train_imid'][:]
    word_code = (f['word_code'][:])

# Get the words from the dtypes. sort them so that they are accoring to index
word_dict = word_code.dtype.names
```

```
w = np.argsort(word_code.item())
words = [word_dict[i] for i in w]

# change the troubling words
words[55] = 'while'
words[80] = 'for'
words[492] = 'catch'
words[561] = 'case'
words[621] = 'end'

# make a dictionary of all captions with the training imid as keys
captions_all = {}
for i in range(1,len(train_imid)+1):
  try:
    x = np.where(train_imid == int(i))
    cap1 = (train_cap[x])
    captions_all[i] = cap1
  except:
    print(i)



# Get the list of inception encodings
inception_encs = os.listdir('inception_encodings')
# print(inception_encs)
# inception_encs1 = inception_encs[]

# parameters set in tutorial
BATCH_SIZE = 100
BUFFER_SIZE = 1000
embedding_dim = 256
units = 512
features_shape = 2048
attention_features_shape = 64
vocab_size = 1004

# get the ids of the images for which we have the inception encodings
im_name = []

for i in range(85000):
    im_str = 'im_' + str(i) + '.npy'
```

```
    if im_str in inception_encs:
        im_name.append(i)


# Splitting the data into training and validation sets. Not using all the training set
here as not enough resources
# to run the model. Therefore, only 50% of the images for which we have the inception
encodings are used for
# training and 15% are used for validation. training (30393) validation (9118)

unique_ids = (im_name)
train_ids = unique_ids[:int(np.round(len(unique_ids)*0.50))]
val_ids =
unique_ids[int(np.round(len(unique_ids)*0.50)):int(np.round(len(unique_ids)*0.65))]
print(len(train_ids))
print(len(val_ids))


# get the training im ids and im captions
train_cap = []
train_im_name = []
for i in train_ids:
    for j in range(len(captions_all[i])):
        train_im_name.append(i)
        train_cap.append(captions_all[i][j])



# get the validation im ids and im captions
val_cap = []
val_im_name = []
for i in val_ids:
    for j in range(len(captions_all[i])):
        val_im_name.append(i)
        val_cap.append(captions_all[i][j])


train_num_steps = (len(train_cap)) // BATCH_SIZE
val_num_steps = (len(val_cap)) // BATCH_SIZE
```

```
# Load the numpy files
def map_func(img_name, cap):

  img_tensor = np.load('inception_encodings/im_'+str(img_name)+'.npy')
#   print(img_tensor.dtype)
#   print(cap.dtype)
  return img_tensor, cap



# Create datasets
train_dataset = tf.data.Dataset.from_tensor_slices((train_im_name, train_cap))

train_dataset = train_dataset.map(lambda item1, item2: tf.numpy_function(
          map_func, [item1, item2], [tf.float32, tf.int32]),
          num_parallel_calls=tf.data.AUTOTUNE)

train_dataset = train_dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
train_dataset = train_dataset.prefetch(buffer_size=tf.data.AUTOTUNE)

val_dataset = tf.data.Dataset.from_tensor_slices((val_im_name, val_cap))

val_dataset = val_dataset.map(lambda item1, item2: tf.numpy_function(
          map_func, [item1, item2], [tf.float32, tf.int32]),
          num_parallel_calls=tf.data.AUTOTUNE)

val_dataset = val_dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
val_dataset = val_dataset.prefetch(buffer_size=tf.data.AUTOTUNE)


# initialize models
encoder = CNN_Encoder(embedding_dim)
decoder = RNN_Decoder(embedding_dim, units, vocab_size)


# define optimizer and lpss functions
optimizer = tf.keras.optimizers.Adam()
loss_object = tf.keras.losses.SparseCategoricalCrossentropy(
```

```
    from_logits=True, reduction='none')


def loss_function(real, pred):
  mask = tf.math.logical_not(tf.math.equal(real, 0))
  loss_ = loss_object(real, pred)

  mask = tf.cast(mask, dtype=loss_.dtype)
  loss_ *= mask

  return tf.reduce_mean(loss_)


loss_plot = []

# training step taken from tutorial
@tf.function
def train_step(img_tensor, target):
  loss = 0


  hidden = decoder.reset_state(batch_size=target.shape[0])

  dec_input = tf.expand_dims(tf.ones(1 * target.shape[0], 1),1)

  with tf.GradientTape() as tape:
      features = encoder(img_tensor)

      for i in range(1, target.shape[1]):
          # passing the features through the decoder
          predictions, hidden, _ = decoder(dec_input, features, hidden)

          loss += loss_function(target[:, i], predictions)

          # using teacher forcing
          dec_input = tf.expand_dims(target[:, i], 1)

  total_loss = (loss / int(target.shape[1]))

  trainable_variables = encoder.trainable_variables + decoder.trainable_variables
```

```
  gradients = tape.gradient(loss, trainable_variables)

  optimizer.apply_gradients(zip(gradients, trainable_variables))

  return loss, total_loss



# validation code same as training step but no updates
def validate_step(img_tensor, target):
  loss = 0

  # initializing the hidden state for each batch
  # because the captions are not related from image to image
  hidden = decoder.reset_state(batch_size=target.shape[0])

  dec_input = tf.expand_dims(tf.ones(1 * target.shape[0], 1),1)

  with tf.GradientTape() as tape:
      features = encoder(img_tensor)

      for i in range(1, target.shape[1]):
          # passing the features through the decoder
          predictions, hidden, _ = decoder(dec_input, features, hidden)

          loss += loss_function(target[:, i], predictions)

          # using teacher forcing
          dec_input = tf.expand_dims(target[:, i], 1)

  total_loss = (loss / int(target.shape[1]))

  return loss, total_loss


# saving model checkpoints
checkpoint_path = "./checkpoints/train"
ckpt = tf.train.Checkpoint(encoder=encoder,
                           decoder=decoder,
```

```
                           optimizer=optimizer)
ckpt_manager = tf.train.CheckpointManager(ckpt, checkpoint_path, max_to_keep=5)


start_epoch = 0
if ckpt_manager.latest_checkpoint:
  start_epoch = int(ckpt_manager.latest_checkpoint.split('-')[-1])
  # restoring the latest checkpoint in checkpoint_path
  ckpt.restore(ckpt_manager.latest_checkpoint)



# Main loop from tutorial expanded to accomodate validation.
EPOCHS = 20
val_loss_plot = []
for epoch in range(start_epoch, EPOCHS):
    start = time.time()
    start1 = time.time()
    total_loss = 0
    val_total_loss = 0

    if epoch % 5 == 0:
      ckpt_manager.save()

    for (batch, (img_tensor, target)) in enumerate(train_dataset):

        batch_loss, t_loss = train_step(img_tensor, target)
        total_loss += t_loss

        if batch % 100 == 0:
            average_batch_loss = batch_loss.numpy()/int(target.shape[1])
            print(f'Epoch {epoch+1} Batch {batch} Loss {average_batch_loss:.4f} Time taken
{time.time()-start1:.2f} sec')
            start1 = time.time()
    # storing the epoch end loss value to plot later
    loss_plot.append(total_loss / train_num_steps)



    for (batch, (img_tensor, target)) in enumerate(val_dataset):
```

```
        val_batch_loss, val_t_loss = validate_step(img_tensor, target)
        val_total_loss += val_t_loss

        if batch % 100 == 0:
            average_batch_loss = val_batch_loss.numpy()/int(target.shape[1])
            print(f'Epoch {epoch+1} Batch {batch} Loss {average_batch_loss:.4f} Time taken
{time.time()-start1:.2f} sec')
            start1 = time.time()

    val_loss_plot.append(val_total_loss / val_num_steps)

    np.save('Attention/tr_loss.npy',loss_plot )
    np.save('Attention/val_loss.npy',val_loss_plot )
    print(f'Epoch {epoch+1} training Loss {total_loss/train_num_steps:.6f} validation Loss
{val_total_loss/val_num_steps:.6f}')
    print(f'Time taken for 1 epoch {time.time()-start:.2f} sec\n')
```

Attention Test

```
import tensorflow as tf
import numpy as np
import pickle
import matplotlib.pyplot as plt
from PIL import Image
import numpy as np

# Test script for Attention model from:
https://www.tensorflow.org/tutorials/text/image_captioning#caption

class BahdanauAttention(tf.keras.Model):
  def __init__(self, units):
    super(BahdanauAttention, self).__init__()
    self.W1 = tf.keras.layers.Dense(units)
    self.W2 = tf.keras.layers.Dense(units)
    self.V = tf.keras.layers.Dense(1)

  def call(self, features, hidden):
    # features(CNN_encoder output) shape == (batch_size, 64, embedding_dim)
```

```
    # hidden shape == (batch_size, hidden_size)
    # hidden_with_time_axis shape == (batch_size, 1, hidden_size)
    hidden_with_time_axis = tf.expand_dims(hidden, 1)

    # attention_hidden_layer shape == (batch_size, 64, units)
    attention_hidden_layer = (tf.nn.tanh(self.W1(features) +
                                         self.W2(hidden_with_time_axis)))

    # score shape == (batch_size, 64, 1)
    # This gives you an unnormalized score for each image feature.
    score = self.V(attention_hidden_layer)

    # attention_weights shape == (batch_size, 64, 1)
    attention_weights = tf.nn.softmax(score, axis=1)

    # context_vector shape after sum == (batch_size, hidden_size)
    context_vector = attention_weights * features
    context_vector = tf.reduce_sum(context_vector, axis=1)

    return context_vector, attention_weights


class CNN_Encoder(tf.keras.Model):
    # Since you have already extracted the features and dumped it
    # This encoder passes those features through a Fully connected layer
    def __init__(self, embedding_dim):
        super(CNN_Encoder, self).__init__()
        # shape after fc == (batch_size, 64, embedding_dim)
        self.fc = tf.keras.layers.Dense(embedding_dim)

    def call(self, x):
        x = self.fc(x)
        x = tf.nn.relu(x)
        return x

class RNN_Decoder(tf.keras.Model):
  def __init__(self, embedding_dim, units, vocab_size):
    super(RNN_Decoder, self).__init__()
    self.units = units
```

```
    self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
    self.gru = tf.keras.layers.GRU(self.units,
                                   return_sequences=True,
                                   return_state=True,
                                   recurrent_initializer='glorot_uniform')
    self.fc1 = tf.keras.layers.Dense(self.units)
    self.dropout = tf.keras.layers.Dropout(0.2)
    self.fc2 = tf.keras.layers.Dense(vocab_size)

    self.attention = BahdanauAttention(self.units)

  def call(self, x, features, hidden):
    # defining attention as a separate model
    # print(x.shape)
    context_vector, attention_weights = self.attention(features, hidden)

    # x shape after passing through embedding == (batch_size, 1, embedding_dim)
    x = self.embedding(x)
    # print(x.shape)
    # print(context_vector.shape)
    # x shape after concatenation == (batch_size, 1, embedding_dim + hidden_size)
    x = tf.concat([tf.expand_dims(context_vector, 1), x], axis=-1)

    # passing the concatenated vector to the GRU
    output, state = self.gru(x)

    # shape == (batch_size, max_length, hidden_size)
    x = self.fc1(output)
    x = self.dropout(x)

    # x shape == (batch_size * max_length, hidden_size)
    x = tf.reshape(x, (-1, x.shape[2]))

    # output shape == (batch_size * max_length, vocab)
    x = self.fc2(x)

    return x, state, attention_weights

  def reset_state(self, batch_size):
    return tf.zeros((batch_size, self.units))
```

```
def load_image(image_path):
    img = tf.io.read_file(image_path)
    img = tf.io.decode_jpeg(img, channels=3)
    img = tf.keras.layers.Resizing(299, 299)(img)
    img = tf.keras.applications.inception_v3.preprocess_input(img)
    return img, image_path

image_model = tf.keras.applications.InceptionV3(include_top=False,
                                                 weights='imagenet')
new_input = image_model.input
hidden_layer = image_model.layers[-1].output

image_features_extract_model = tf.keras.Model(new_input, hidden_layer)

def evaluate(image):
    attention_plot = np.zeros((50, attention_features_shape))

    hidden = decoder.reset_state(batch_size=1)

    temp_input = tf.expand_dims(load_image(image)[0], 0)
    img_tensor_val = image_features_extract_model(temp_input)
    img_tensor_val = tf.reshape(img_tensor_val, (img_tensor_val.shape[0],
                                                 -1,
                                                 img_tensor_val.shape[3]))

    features = encoder(img_tensor_val)


    dec_input = tf.ones(1,1)
    dec_input = tf.expand_dims(dec_input, 0)
    result = []

    for i in range(50):
        predictions, hidden, attention_weights = decoder(dec_input,
                                                         features,
                                                         hidden)

        attention_plot[i] = tf.reshape(attention_weights, (-1, )).numpy()
```

```
        predicted_id = tf.random.categorical(predictions, 1)[0][0].numpy()
        predicted_word = tf.compat.as_text(ind2word[predicted_id])
        result.append(predicted_word)

        if predicted_id == 2:
            return result, attention_plot

        dec_input = tf.expand_dims([predicted_id], 0)

    attention_plot = attention_plot[:len(result), :]
    return result, attention_plot


def plot_attention(image, result, attention_plot):
    temp_image = np.array(Image.open(image))

    fig = plt.figure(figsize=(10, 10))

    len_result = len(result)
    for i in range(len_result):
        temp_att = np.resize(attention_plot[i], (8, 8))
        grid_size = max(int(np.ceil(len_result/2)), 2)
        ax = fig.add_subplot(grid_size, grid_size, i+1)
        ax.set_title(result[i])
        img = ax.imshow(temp_image)
        ax.imshow(temp_att, cmap='gray', alpha=0.6, extent=img.get_extent())

    plt.tight_layout()
    plt.savefig('Attention/im_11898.png')
    plt.show()


BATCH_SIZE = 100
BUFFER_SIZE = 1000
embedding_dim = 256
units = 512
features_shape = 2048
attention_features_shape = 64
vocab_size = 1004
optimizer = tf.keras.optimizers.Adam()
```

```
encoder = CNN_Encoder(embedding_dim)
decoder = RNN_Decoder(embedding_dim, units, vocab_size)

checkpoint_path = "./checkpoints/train"
ckpt = tf.train.Checkpoint(encoder=encoder,
                           decoder=decoder,
                           optimizer=optimizer)
ckpt_manager = tf.train.CheckpointManager(ckpt, checkpoint_path, max_to_keep=5)


#restore second checkpoint as it is not that overfitted. best checkpoint
ckpt.restore(checkpoint_path+'\ckpt-2')



# load the ind2word stores earlier
with open('ind2word.pkl', 'rb') as f:
    ind2word = pickle.load(f)
image = 'testimages/im_11898.jpg'
result, attention_plot = evaluate(image)

print('Prediction Caption:', ' '.join(result))
plot_attention(image, result, attention_plot)
```

Injection Model (InceptionV3+GloVe+LSTM)

```
import h5py
import urllib.request
import numpy as np
from numpy import array
import pandas as pd
import matplotlib.pyplot as plt
#%matplotlib inline
import tensorflow
import string
import os
from PIL import Image
import glob
from pickle import dump, load
from time import time
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from keras.preprocessing import sequence
```

```
from keras.models import Sequential
from keras.layers import LSTM, Embedding, TimeDistributed, Dense, RepeatVector,\
                         Activation, Flatten, Reshape, concatenate, Dropout,
BatchNormalization
from tensorflow.keras.optimizers import Adam, RMSprop
from keras.layers.wrappers import Bidirectional
from keras.layers.merge import add
from keras.applications.inception_v3 import InceptionV3
from keras.preprocessing import image
from keras.models import Model
from keras import Input, layers
from keras import optimizers
from keras.applications.inception_v3 import preprocess_input
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical
#DOWNLOADING TRAIN IMAGES
#f = h5py.File('eee443_project_dataset_train.h5', 'r')
# trainurl=f['train_url']
# i=1
# for url in trainurl:
#    print(i)
#    name='image'+ str(i) +'.jpg'
#    url1=url.decode('utf-8')
#    img=urllib.request.urlretrieve(url1,name)
#    i+=1
# Creating Description Dictionary
f = h5py.File('eee443_project_dataset_train.h5', 'r')
train_imid = np.array(f['train_imid'])
train_cap = np.array(f['train_cap'])
words = np.array(f['word_code'].dtype.names)
word_inds = np.array(list(f['word_code'][0]))
word_inds = word_inds.astype(np.int32)

for i in range(len(words)):
    if words[i][0] == 'x' and words[i][1] != '_':
        words[i] = words[i][1:]
    if words[i] == 'x_START_':
        words[i] = 'startseq'
    if words[i] == 'x_END_':
```

```
        words[i] = 'endseq'
path = 'C:/Users/Usayrim/Desktop/NN-Final Project/train_images'
files = os.listdir(path)
img_cap = dict()
for name in files:
    key = name.split('.')[0]
    img_cap[key] = list()
    id = int(key.split('_')[1])
    inds = np.where(train_imid == id + 1)[0]
    for ind in inds:
        cap = train_cap[ind]
        cap_str = ''
        for word in cap:
            ind2 = np.where(word_inds == word)
            if word != 0:
                if word == 2:
                    cap_str += words[ind2[0]][0]
                else:
                    cap_str += words[ind2[0]][0] + ' '
        img_cap[key].append(cap_str)
train_descriptions=dict(list(img_cap.items())[:60330])
val_descriptions= dict(list(img_cap.items())[60330:])
# convert the loaded descriptions into a vocabulary of words
def to_vocabulary(descriptions):
    # build a list of all description strings
    all_desc = set()
    for key in descriptions.keys():
        [all_desc.update(d.split()) for d in descriptions[key]]
    return all_desc

# summarize vocabulary
vocabulary = to_vocabulary(img_cap)
vocabulary.remove('startseq')
vocabulary.remove('endseq')
print('Original Vocabulary Size: %d' % len(vocabulary))
def preprocess(image_path):
    # Convert all the images to size 299x299 as expected by the inception v3 model
    img = image.load_img(image_path, target_size=(299, 299))
    # Convert PIL image to numpy array of 3-dimensions
    x = image.img_to_array(img)
```

```
    # Add one more dimension
    x = np.expand_dims(x, axis=0)
    # preprocess the images using preprocess_input() from inception module
    x = preprocess_input(x)
    return x
# Create a list of all the training images with their full path names
trimages = 'C:/Users/Usayrim/Desktop/NN-Final Project/train_images'
# Create a list of all image names in the directory
train_img = glob.glob(timages + '*.jpg')

# Create a list of all the test images with their full path names
teimages = 'C:/Users/Usayrim/Desktop/NN-Final Project/test_images'
# Create a list of all image names in the directory
test_img = glob.glob(timages + '*.jpg')

# Load the inception v3 model
model = InceptionV3(weights='imagenet')
# Create a new model, by removing the last layer (output layer) from the inception v3
model_new = Model(model.input, model.layers[-2].output)
def encode(image):
    image = preprocess(image) # preprocess the image
    fea_vec = model_new.predict(image) # Get the encoding vector for the image
    fea_vec = np.reshape(fea_vec, fea_vec.shape[1]) # reshape from (1, 2048) to (2048, )
    return fea_vec
# Call the funtion to encode all the train images
# This will take a while on CPU - Execute this only once
start = time()
encoding_train = {}
for img in train_img:
    encoding_train[img[len(trimages):]] = encode(img)
print("Time taken in seconds =", time()-start)
# Save the bottleneck train features to disk
with open("C:/Users/Usayrim/Desktop/NN-Final Project/encoded_train_images.pkl", "wb") as
encoded_pickle:
    pickle.dump(encoding_train, encoded_pickle)
# Call the funtion to encode all the test images - Execute this only once
start = time()
encoding_test = {}
for img in test_img:
    encoding_test[img[len(teimages):]] = encode(img)
```

```
print("Time taken in seconds =", time()-start)
# Save the bottleneck test features to disk
with open("C:/Users/Usayrim/Desktop/NN-Final Project/encoded_test_images.pkl", "wb") as
encoded_pickle:
    pickle.dump(encoding_test, encoded_pickle)
features = load(open("C:/Users/Usayrim/Desktop/NN-Final Project/encoded_train_images.pkl",
"rb"))
print('Photos: train=%d' % len(train_features))
val_features = features[60330:,:]
train_features=features[:60330,:]
ixtoword = {}
wordtoix = {}

ix = 1
for w in vocabulary:
    wordtoix[w] = ix
    ixtoword[ix] = w
    ix += 1
vocab_size = len(ixtoword) + 1 # one for appended 0's
Vocab_size
def to_lines(descriptions):
    all_desc = list()
    for key in descriptions.keys():
        [all_desc.append(d) for d in descriptions[key]]
    return all_desc

# calculate the length of the description with the most words
def max_length(descriptions):
    lines = to_lines(descriptions)
    return max(len(d.split()) for d in lines)

# determine the maximum sequence length
max_length = max_length(train_descriptions)
print('Description Length: %d' % max_length)
# data generator, intended to be used in a call to model.fit_generator()
def data_generator(descriptions, photos, wordtoix, max_length, num_photos_per_batch):
    X1, X2, y = list(), list(), list()
    n=0
    # loop for ever over images
    while 1:
```

```
        for key, desc_list in descriptions.items():
            n+=1
            # retrieve the photo feature
            photo = photos[key+'.jpg']
            for desc in desc_list:
                # encode the sequence
                seq = [wordtoix[word] for word in desc.split(' ') if word in wordtoix]
                # split one sequence into multiple X, y pairs
                for i in range(1, len(seq)):
                    # split into input and output pair
                    in_seq, out_seq = seq[:i], seq[i]
                    # pad input sequence
                    in_seq = pad_sequences([in_seq], maxlen=max_length)[0]
                    # encode output sequence
                    out_seq = to_categorical([out_seq], num_classes=vocab_size)[0]
                    # store
                    X1.append(photo)
                    X2.append(in_seq)
                    y.append(out_seq)
            # yield the batch data
            if n==num_photos_per_batch:
                yield ([array(X1), array(X2)], array(y))
                X1, X2, y = list(), list(), list()
                n=0
# Load Glove vectors
glove_dir = 'C:/Users/Usayrim/Desktop/NN-Final Project'
embeddings_index = {} # empty dictionary
f = open(os.path.join(glove_dir, 'glove.6B.200d.txt'), encoding="utf-8")

for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()
print('Found %s word vectors.' % len(embeddings_index))
embedding_dim = 200

# Get 200-dim dense vector for each of the 10000 words in out vocabulary
embedding_matrix = np.zeros((vocab_size, embedding_dim))
```

```
for word, i in wordtoix.items():
    #if i < max_words:
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        # Words not found in the embedding index will be all zeros
        embedding_matrix[i] = embedding_vector
embedding_matrix.shape
#MODEL
inputs1 = Input(shape=(2048,))
fe1 = Dropout(0.5)(inputs1)
fe2 = Dense(256, activation='relu')(fe1)
inputs2 = Input(shape=(max_length,))
se1 = Embedding(vocab_size, embedding_dim, mask_zero=True)(inputs2)
se2 = Dropout(0.5)(se1)
se3 = LSTM(256)(se2)
decoder1 = add([fe2, se3])
decoder2 = Dense(256, activation='relu')(decoder1)
outputs = Dense(vocab_size, activation='softmax')(decoder2)
model = Model(inputs=[inputs1, inputs2], outputs=outputs)
model.summary()
model.layers[2]
model.layers[2].set_weights([embedding_matrix])
model.layers[2].trainable = False
model.compile(loss='categorical_crossentropy', optimizer='adam')
train_loss=[]
val_loss=[]
model.optimizer.lr = 0.0001
epochs = 50
number_pics_per_bath = 16
train_steps = len(train_descriptions)//number_pics_per_bath
val_steps = len(val_descriptions)//number_pics_per_bath

callbacks = [EarlyStopping(monitor='val_loss', patience=4,
verbose=1),ReduceLROnPlateau(monitor='val_loss', patience=3, factor=0.1, min_lr=0.00001,
verbose=1)]
for i in range(epochs):
    tgenerator = data_generator(train_descriptions, train_features, wordtoix, max_length,
number_pics_per_bath)
```

```
    vgenerator = data_generator(val_descriptions, val_features, wordtoix, max_length,
number_pics_per_bath)
    Incep=model.fit_generator(tgenerator, epochs=1,
steps_per_epoch=train_steps,validation_data=vgenerator, validation_steps=val_steps,
callbacks = [callbacks], shuffle=True, verbose=1)
    train_loss.append(hist.history['loss'])
    val_loss.append(hist.history['val_loss'])
model.save('model_inception_injection.h5')

f = h5py.File('eee443_project_dataset_test.h5', 'r')
test_imid = np.array(f['test_imid'])
test_cap = np.array(f['test_caps'])
words = np.array(f['word_code'].dtype.names)
word_inds = np.array(list(f['word_code'][0]))
word_inds = word_inds.astype(np.int32)

for i in range(len(words)):
    if words[i][0] == 'x' and words[i][1] != '_':
        words[i] = words[i][1:]
    if words[i] == 'x_START_':
        words[i] = 'startseq'
    if words[i] == 'x_END_':
        words[i] = 'endseq'
path = 'C:/Users/Usayrim/Desktop/NN-Final Project/test_images'
files = os.listdir(path)
img_cap = dict()
for name in files:
    key = name.split('.')[0]
    img_cap[key] = list()
    id = int(key.split('_')[1])
    inds = np.where(test_imid == id + 1)[0]
    for ind in inds:
        cap = test_cap[ind]
        cap_str = ''
        for word in cap:
            ind2 = np.where(word_inds == word)
            if word != 0:
                if word == 2:
                    cap_str += words[ind2[0]][0]
                else:
```

```
                    cap_str += words[ind2[0]][0] + ' '
        img_cap[key].append(cap_str)
test_descriptions=img_cap
with open("C:/Users/Usayrim/Desktop/NN-Final Project/encoded_test_images.pkl", "rb") as
encoded_pickle:
    encoding_test = load(encoded_pickle)

def greedySearch(photo):
    in_text = 'startseq'
    for i in range(max_length):
        sequence = [wordtoix[w] for w in in_text.split() if w in wordtoix]
        sequence = pad_sequences([sequence], maxlen=max_length)
        yhat = model.predict([photo,sequence], verbose=0)
        yhat = np.argmax(yhat)
        word = ixtoword[yhat]
        in_text += ' ' + word
        if word == 'endseq':
            break
    final = in_text.split()
    final = final[1:-1]
    final_arr = final.copy()
    final = ' '.join(final)
    return final,final_arr
```