BILKENT UNIVERSITY

EEE443 Neural Network – Fall 2021-2022

A Comparative Study of Image Captioning Methods

Atakan Topcu - 21803095

Ata Yavuzyılmaz - 21802466

Dora Tütüncü - 21801687

Ata Korkusuz - 21803262

Models: https://drive.google.com/drive/folders/1tZ5A3hUF0nWLll4WMuVfzZZ2ENBeCqx2?usp=sharing

Table of Contents

Abstract **2**

Introduction **2**

Methods **3**

Dataset Description 3

Preprocessing of the Dataset 3

Feature Extraction 4

Generator Model Training 4

LSTM 5

Bidirectional LSTM 6

GRU 6

Bidirectional GRU 7

Model Evaluation 7

Results **7**

LSTM with non-trainable embedding matrix 7

GRU with non-trainable embedding matrix 8

Bidirectional LSTM with non-trainable embedding matrix 8

Bidirectional GRU with non-trainable embedding matrix 9

Bidirectional LSTM with trainable embedding matrix 9

Bidirectional GRU with trainable embedding matrix 10

BLEU Scores 10

Comparison of Our Models with Results in Literature 10

Discussion **11**

References **12**

Appendix **13**

Captioned Images 13

Code 32

1. Abstract

The purpose of this project is to caption textual descriptions of images by generating sentences that include the states of objects and the behaviors of people and animals in the corresponding images. The dataset provided by the instructor contains training and test images along with their captions. The image features are extracted by using well-known pretrained network which is ResNetV2. Neural network architectures containing Convolutional Neural Networks (CNNs) and variations of LSTM and GRU recurrent neural networks are utilized to create captions that capture information about the contents of an image. Furthermore, trainable GloVe 300d and non-trainable GloVe 300d embedding vectors have been utilized to understand the effects of embedding vectors. The training and validation loss graphs are obtained for each model and their BLEU scores are calculated on the test dataset in order to compare the performances of the models. Overall, we were able to train and test six networks and compare their performances in order to decide on the model with the best performance.

2. Introduction

Image captioning has been and is continuing to be an emergent area for machine learning due to its major dependence on cutting edge neural network models for image feature extraction and sequence generation. In essence image captioning is a sequence to sequence problem where images are inputted as a sequence of pixels and a sequence of words are outputted. The model is also preprocessed using the GloVe word embeddings in preparation for inputs to the proposed Neural Network models. Existing works in literature follow a similar pattern of image sequence extraction usually through a CNN and a word sequence generation mostly using an RNN. In image caption generation tasks, it is difficult to evaluate a model performance through a generic measure such as accuracy or precision as there are multiple ways of describing an image which might be correct. Therefore, the bilingual evaluation understudy (BLEU) metric is used for evaluating how well the generated sentence compares to the reference sentence. The Flickr dataset will be referred to as the test data for the following B-4 scores. In this context, *Deep Visual-Semantic Alignments for Generating Image Descriptions* article explores a par-inject model where the image is encoded with a CNN and the sentences are encoded with a bidirectional LSTM aiming to align words with relevant parts of the image before feeding it to the generative model [1]. This combination of two models that sets the fundamentals of image captioning achieves a B-4 score of 15.7. *Paying More Attention to Saliency: Image Captioning with Saliency and Context Attention* article additionally explores the idea of saliency, where the context is extracted using a CNN and the salient regions are extracted using a convolutional LSTM [2]. Other than the two maps generated from the images, the rest of the model follows a similar structure feeding the two maps to each LSTM node, which achieves a B-4 score of 21.3. *Unified Vision-Language Pre-Training for Image Captioning and VQA* article differs almost entirely from the two previous methods by implementing a transformer architecture [3]. The model uses a unified encoder-decoder to go in between image-text pairs and generated captions. The model still includes a pre-training for the objective with a dropout layer like structure masking the words and it includes a pretraining for the images that implements a simple image detector, finally achieving a 30.1 B-4 score.

The purpose of our work is to investigate comparatively the model variations on the image captioning task. The sudden shift to the transformer model leaves unexplored areas on the bidirectional LSTM model. The rising number of models on the image captioning task also brings into question whether the older models are outperformed only due to the usage of increased computation power in the new models. By rebuilding and testing models similar to the existing literature using a new dataset, we aim to level the playground for the performance evaluation of these models and their variations. The comparison of these model variations on the caption generation side is expected to show which methodology in image captioning is more promising to pursue as a human-like image descriptor. The CNN model Inception-ResNet v2 is used to extract features from the images. To generate captions of the images, we have

tried different models that are used in Natural Language Processing: LSTM, Bidirectional LSTM, GRU, and Bidirectional GRU.

3. Methods

3.1. Dataset Description

The dataset that is employed in this project is a custom dataset that includes 82783 Flickr image URLs and 400135 sample captions corresponding to the images for the train dataset. For the test dataset, there are 40504 images along with 195954 captions. The training and test image id maps are also given, which maps the captions to their relevant images as images have multiple captions. The images have varying sizes and dimensions and are RGB. Furthermore, some of the URLs (links) in the given datasets were broken and some of the proper ones were not in the format of JPEG. Thus, after filtering, we were able to use 71676 training and 22481 test images along with their captions.

The captions of training/test images are stored in the 'train_cap/test cap' variable where the maximum length of the captions is 17. The captions are not stored as string literals but rather stored as integers. These integers can then be used in coalition with the word_count array to get a string representation of the descriptions. 'word_count' array contains words and their corresponding indices and it has 1004 words in total. The indices of training/test images are stored in the 'train_imid/test_imid' variable. With the indices of the images, we are able to match the images with 'train_url/test_url' and 'train_cap/test_cap'. Though there were pretrained images in the train_ims and test_ims, they were not used in this project.

3.2. Preprocessing of the Dataset

To evaluate sequences of words, the implementation of word embeddings were also necessary for the project. Word embeddings are types of word representations that allow words with similar meaning to have similar representations. They are expressed as classes of techniques where individual words are represented as real-valued vectors in a predefined vector space. Each word is mapped to one vector and the vector values are learned in a way that resembles a neural network, and hence the technique is often lumped into the field of deep learning. The distributed representation is learned based on the usage of words. This allows words that are used in similar ways to result in having similar representations, essentially capturing their meaning.

These embeddings serve 3 primary purposes in neural networks:

- Finding nearest neighbors in the embedding space. These can be used to make recommendations based on user interests or cluster categories.
- As input to a machine learning model for a supervised task.
- For visualization of concepts and relations between categories.

The Global Vectors for Word Representation, or GloVe, algorithm is an extension to the word2vec method for efficiently learning word vectors, developed by Pennington, et al. at Stanford [4]. GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space. Rather than using a window to define local context, GloVe constructs an explicit word-context or word co-occurrence matrix using statistics across the whole text corpus [5].

The basic idea behind the GloVe word embedding is to derive the relationship between the words from statistics. Unlike the occurrence matrix, the co-occurrence matrix tells you how often a particular word pair occurs together [6]. The advantage of GloVe is that, unlike the other common embedding, the Word2vec, GloVe does not rely just on local statistics (local context information of words), but incorporates global statistics (word co-occurrence) to obtain word vectors. Due to the synergy between the GloVe and Word2vec, we believed that GloVe would be a better option for our task at hand [7].

To obtain the actual images, we have extracted the images from a URL provided to us. A connection to the URL was established, and images from Flickr were pulled using this link. However, not all images obtained from this link were in the jpeg format and some links were corrupted which led to decrease in both train and test images. For filtering, the jpeg format images were extracted among groups of images that were different image formats, such as png, jfif, etc.

Another crucial process in the preprocessing part was to map the captions according to the indices of the images. In order to achieve this, we have assigned 5 captions to each image that was provided to us. The captions provided to us were complicated and scattered, however we have adjusted the dataset to suit the indexed images. Then, in order to establish equal caption and image sizes, the 5 captions that were found were converted to the list class, providing easily manageable data. The data provided to us were classified as test images initially, however in order to accurately evaluate the results, evaluation data, we need a validation dataset as well. We have decided to establish this by separating the dataset as 85% and 15%, with them being test and validation data, respectively. Then, we trained over the test data, and left the validation dataset for further evaluation. For further processing, an encoding process has been established.

3.3. Feature Extraction

A pre-trained CNN model is used in order to achieve cutting edge object detection and classification from the images. The pretrained model of choice for transfer learning is the Inception ResNet v2. This model uses residual connections rather than using a filter concatenation stage. These connections speed up the training by a significant margin while not compromising on accuracy. Inception-ResNet-v2 is a variation of the Inception V3 model which is another popular model for image analysis and computer vision, and Inception-ResNet-v2 is considerably deeper than the previous Inception V3 [8]. The last layer of resnet is removed as we do not require the softmax classification output but the features themselves. To use the Inception model, some preprocessing is required. Inception-ResNet-v2 takes 3-channel (RGB) (299, 299) images as inputs. Thus, since some images change in size, reshaping needs to be done on the train and test images which is done using tensorflow's library. After the reshaping process, the dataset has been used as an input to the resnet encoder model. Output of the model yields a 2D vector containing the extracted features for each image. These feature arrays are then stacked together to form a single numpy feature array. Finally, the extracted feature encodings are saved in pkl format.

3.4. Generator Model Training

The whole model consists of the following layers where both columns are input the extracted features. The decrease in column number indicates the addition of the two outputs of the two columns at the preceding row.

Embedding	Dropout
Dropout	Dense
Generator	Relu Activation
Dense	

Relu Activation
Dense
Softmax Activation

The dropout rate at each dropout layer is set as 0.5. The dropout layers prevent overfitting and majorly helps with the generalizability of the model especially in the cases of unknown word tokens. Relu activations introduce non linearity without contributing too much to the computational burden at gradient calculations in back propagation. Embedding layer is set as non-trainable as the image embeddings are pre trained beforehand. Since the subsequent layers support masking and 0 is used as special padding at the embedding layer. The embedding layer has been tried with the bidirectional RNN models only as they have outperformed their non-bidirectional counterparts. The initial dense layer takes in the image as a sequence of pixels after some pixels are dropped. The second dense layer acts as a combining operation for the generator sentence sequence output and the image sequence output. The final dense layer in combination with the softmax activation makes a prediction for the next word in the sequence, where the output is the confidence for each tokenized word.

Four different generator RNN models have been tested with namely: LSTM, bidirectional LSTM, GRU, bidirectional GRU. The generative layers are explored in the following subheadings.

3.4.1. LSTM

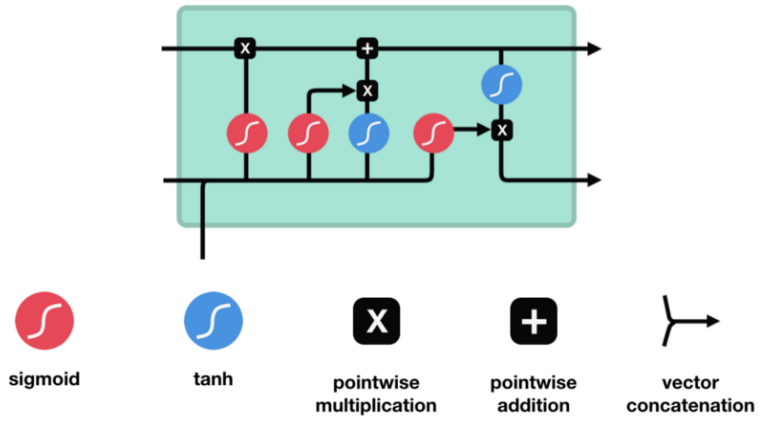


Fig. 1: LSTM Architecture [9]

The core concept of LSTM is the cell state, and it's various gates. The cell state acts as a transport highway that transfers relative information all the way down the sequence chain. The cell state, in theory, can carry relevant information throughout the processing of the sequence. As the cell state goes on its journey, information gets added or removed to the cell state via gates. The gates are different neural networks that decide which information is allowed on the cell state. The gates can learn what information is relevant to keep or forget during training. Gates contains sigmoid activations. which squishes values between 0 and 1, aiding in the update or forget data processes due to a number getting multiplied by 0, essentially causing values to disappear or be "forgotten." Any number multiplied by 1 is the same value therefore that value stays the same or is "kept." The network can learn which data is not important therefore can be forgotten or which data is important to keep.

There are three different gates that regulate information flow in an LSTM cell. A forget gate, input gate, and output gate. The forget gate decides what information should be thrown away or kept. Information from the previous hidden state and information from the current input is passed through the sigmoid function. Values come out between 0 and 1. The closer to 0 means to forget, and the closer to 1 means to keep. To update the cell state, we have the input gate.

First, we pass the previous hidden state and current input into a sigmoid function. That decides which values will be updated by transforming the values to be between 0 and 1. 0 means not important, and 1 means important. You also pass the hidden state and current input into the tanh function to squish values between -1 and 1 to help regulate the network. Then you multiply the tanh output with the sigmoid output. The sigmoid output will decide which information is important to keep from the tanh output. Finally, the output gate decides what the next hidden state should be. Remember that the hidden state contains information on previous inputs. The hidden state is also used for predictions. First, we pass the previous hidden state and the current input into a sigmoid function. Then we pass the newly modified cell state to the tanh function. We multiply the tanh output with the sigmoid output to decide what information the hidden state should carry. The output is the hidden state. The new cell state and the new hidden is then carried over to the next time step.

3.4.2. Bidirectional LSTM

Bidirectional LSTM is an extension of the LSTM model. Two LSTM models are used at the same time where one of the models learns the reverse of the input sequence. Bidirectional LSTM preserves information from both past and future and Unidirectional LSTM preserves information only from past. Our algorithm uses bidirectional LSTM at the beginning in order to understand the context better because it performs more suitably on complicated tasks, leading to a better understanding in semantics. As two models are used, the output of the two models are then combined together at the merge step with summing, averaging, multiplication or concatenation. Our merging step of choice was concatenation as it is more representative of the bidirectional LSTM implementations in literature.

3.4.3. GRU

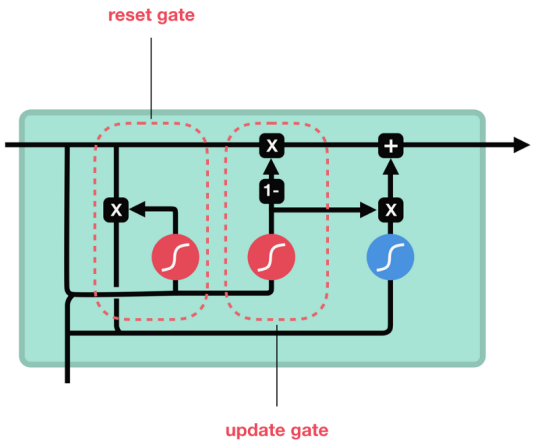


Fig. 2: GRU Architecture [9]

The GRU is the newer generation of Recurrent Neural networks and is pretty similar to an LSTM. GRU's got rid of the cell state and used the hidden state to transfer information. It also only has two gates, a reset gate and update gate. The update gate acts similar to the forget and input gate of an LSTM. It decides what information to throw away and what new information to add. The reset gate is another gate used to decide how much past information to forget. GRU's have fewer tensor operations; therefore, they are a little faster to train than LSTM.
We decided to use both LSTM and GRU together because these models are usually used in conjunction, because in the ML community, there isn't a clear answer to which is the better model. We will try both models to determine which one works better for our case.

3.4.4. Bidirectional GRU

Similar to the bidirectional LSTM, bidirectional GRU combines two GRU models running at the same time where one of the models takes as input the reverse sequence of what the other model takes. This allows for future and past information preservation compared to the only past data information of the traditional unidirectional version. As two models are used, their respective outputs are combined once again with concatenation as the merging step choice. The reasoning for this choice is yet again due to concatenate being more widespread and therefore more representative of the literature.

3.5. Model Evaluation

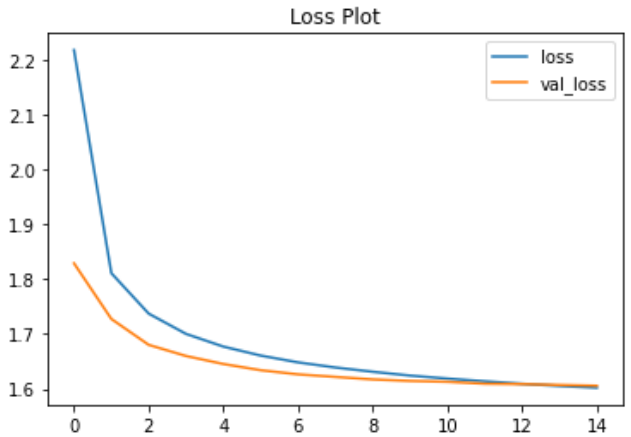
When training the models, the change in the training loss and the validation loss is recorded to plot the loss change. The plots can be seen in the Results section. For this project, we have decided to use the BLEU algorithm to evaluate our models. BLEU (bilingual evaluation understudy) is an algorithm for evaluating the quality of text which has been machine-translated from one natural language to another. This was chosen due to its high popularity and inexpensive metrics for evaluation [10]. Scores are calculated for individual translated segments , which for our case were sentences, by comparing them with a set of good quality reference translations. Those scores are then averaged over the whole corpus to reach an estimate of the translation's overall quality. Intelligibility or grammatical correctness are not taken into account. To calculate the BLEU scores, we need to caption all the test images. To do this in an effective way, we have created a captioning dictionary for each model and stored the captions with their respective image ID. Afterwards, we have also created another dictionary which contains the original list of captions of each image, and stored them with the image IDs. Using these dictionaries, we were able to compare the prediction captions and original captions to calculate four different BLEU scores.

4. Results

The loss function used for all the models is categorical cross-entropy. After training the models for 15 epochs, all of the images in the test dataset are captioned and the results are compared with the original captions to calculate BLEU scores.

4.1. LSTM with non-trainable embedding matrix

The results for LSTM with non-trainable embedding matrix is as follows.

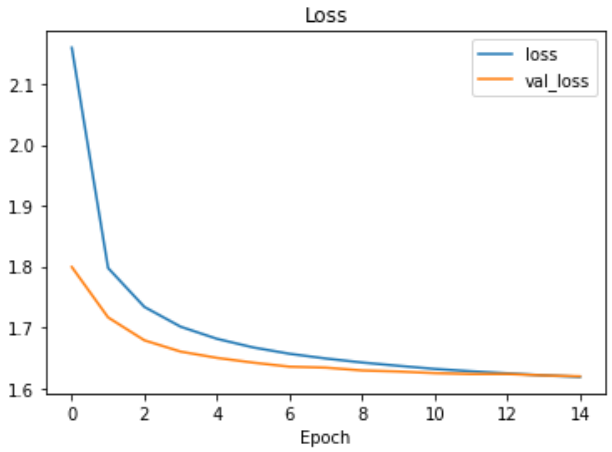



```
2299/2299 [==============================] - 539s
2299/2299 [==============================] - 517s
2299/2299 [==============================] - 495s
2299/2299 [==============================] - 505s
2299/2299 [==============================] - 509s
2299/2299 [==============================] - 522s
2299/2299 [==============================] - 526s
2299/2299 [==============================] - 523s
2299/2299 [==============================] - 517s
2299/2299 [==============================] - 516s
2299/2299 [==============================] - 513s
2299/2299 [==============================] - 516s
```

Fig. 3: LSTM training and validation loss with respect to epoch (left), training time for 12 epochs (right)

4.2. GRU with non-trainable embedding matrix

The results for GRU with non-trainable embedding matrix is as follows.



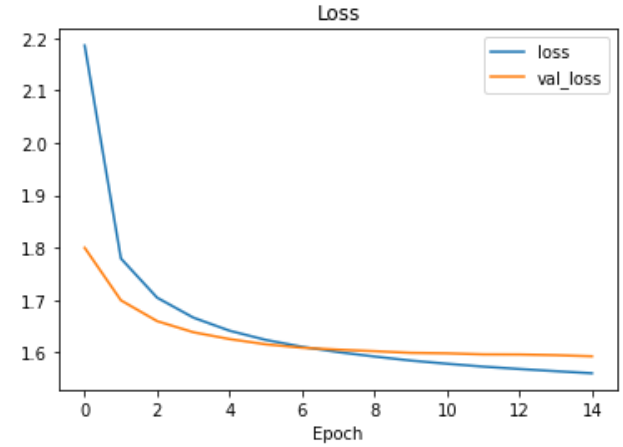

```
2299/2299 [==============================] - 497s
2299/2299 [==============================] - 477s
2299/2299 [==============================] - 473s
2299/2299 [==============================] - 475s
2299/2299 [==============================] - 474s
2299/2299 [==============================] - 472s
2299/2299 [==============================] - 470s
2299/2299 [==============================] - 471s
2299/2299 [==============================] - 472s
2299/2299 [==============================] - 472s
2299/2299 [==============================] - 477s
2299/2299 [==============================] - 479s
```

Fig. 4: GRU training and validation loss with respect to epoch (left), training time for 12 epochs (right)

For GRU with non-trainable embedding matrices, we see that the epochs take shorter time, and the losses are similar to the LSTM with non-trainable embedding matrices. Yet, we can still expect LSTM to perform slightly better convergence starting from a higher loss.

4.3. Bidirectional LSTM with non-trainable embedding matrix

The results for Bidirectional LSTM with non-trainable embedding matrix is as follows.



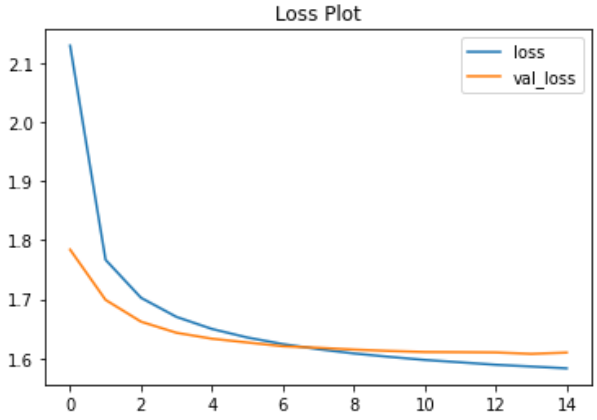

```
2299/2299 [==============================] - 893s
2299/2299 [==============================] - 887s
2299/2299 [==============================] - 882s
2299/2299 [==============================] - 891s
2299/2299 [==============================] - 891s
2299/2299 [==============================] - 885s
2299/2299 [==============================] - 879s
2299/2299 [==============================] - 877s
2299/2299 [==============================] - 889s
2299/2299 [==============================] - 895s
2299/2299 [==============================] - 889s
2299/2299 [==============================] - 877s
```

Fig. 5: Bidirectional LSTM training and validation loss with respect to epoch (left), training time for 12 epochs (right)

The training time for Bidirectional LSTM is higher than the LSTM model as expected. However, the expected performance is higher than LSTM.

4.4. Bidirectional GRU with non-trainable embedding matrix

The results for Bidirectional GRU with non-trainable embedding matrix is as follows.



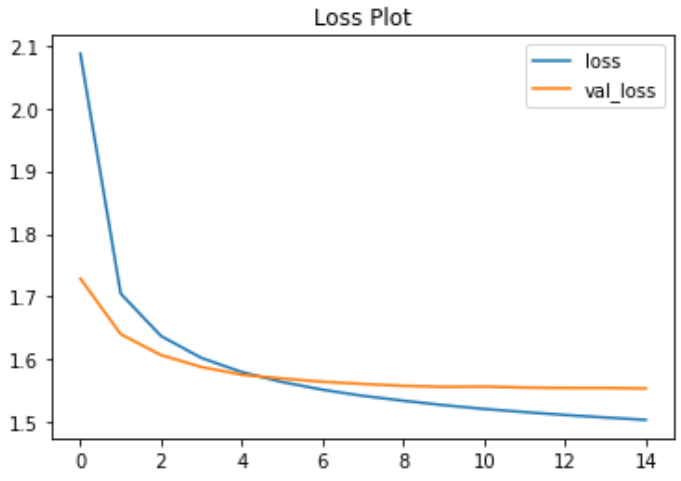

```
2299/2299 [==============================] - 798s
2299/2299 [==============================] - 781s
2299/2299 [==============================] - 771s
2299/2299 [==============================] - 770s
2299/2299 [==============================] - 756s
2299/2299 [==============================] - 761s
2299/2299 [==============================] - 778s
2299/2299 [==============================] - 809s
2299/2299 [==============================] - 789s
2299/2299 [==============================] - 782s
2299/2299 [==============================] - 780s
2299/2299 [==============================] - 761s
```

Fig. 6: Bidirectional GRU training and validation loss with respect to epoch (left), training time for 12 epochs (right)

Just like Bidirectional LSTM, the Bidirectional GRU model takes longer time to train compared to GRU.

4.5. Bidirectional LSTM with trainable embedding matrix

The results for Bidirectional LSTM with trainable embedding matrix is as follows.



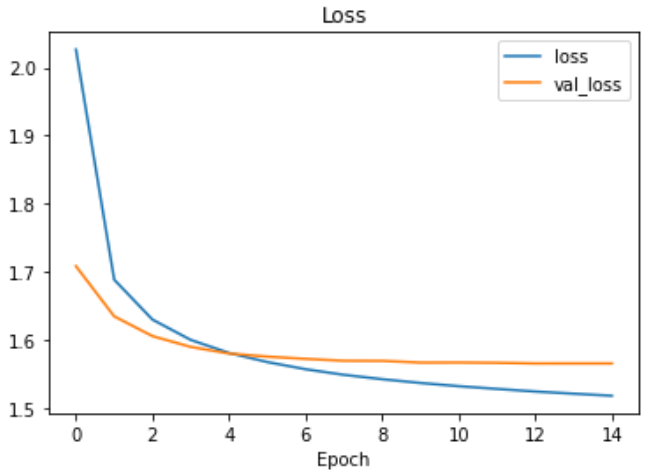

```
Epoch 1/15
2299/2299 [==============================] - 1019s
cy: 0.6112
Epoch 2/15
2299/2299 [==============================] - 999s
y: 0.6230
Epoch 3/15
2299/2299 [==============================] - 1025s
cy: 0.6288
Epoch 4/15
2299/2299 [==============================] - 1040s
cy: 0.6317
Epoch 5/15
2299/2299 [==============================] - 1005s
cy: 0.6333
Epoch 6/15
2299/2299 [==============================] - 1004s
```

Fig. 7: Bidirectional LSTM with trainable embedding matrix training and validation loss with respect to epoch (left), training time for 6 epochs (right)

Since the weights are trainable, this model takes longer time to train compared to other models. However, the loss is lowest for this model. Due to this fact, we expect the highest evaluation scores for this model. Also, we can see that as we implement trainable embedding matrix to the system, the model was able to learn more which leads to decrease in loss.

4.6. Bidirectional GRU with trainable embedding matrix

The results for Bidirectional GRU with trainable embedding matrix is as follows.




```
Epoch 1/15
2299/2299 [==============================] - 1001s
cy: 0.6230
Epoch 2/15
2299/2299 [==============================] - 1022s
cy: 0.6314
Epoch 3/15
2299/2299 [==============================] - 1023s
cy: 0.6359
Epoch 4/15
2299/2299 [==============================] - 1023s
cy: 0.6386
Epoch 5/15
2299/2299 [==============================] - 1029s
cy: 0.6402
Epoch 6/15
2299/2299 [==============================] - 1013s
```

Fig. 8: Bidirectional GRU with trainable embedding matrix training and validation loss with respect to epoch (left), training time for 6 epochs (right)

4.7. BLEU Scores

The BLEU scores of all the implemented models are as follows.

	LSTM	GRU	BLSTM	BGRU	trainable BLSTM	trainable BGRU
BLEU-1	54.79	53.71	54.97	54.42	**56.75**	54.58
BLEU-2	35.43	34.29	35.57	34.99	**37.15**	35.05
BLEU-3	21.82	21.20	22.46	21.98	**23.23**	21.97
BLEU-4	13.23	12.93	14.06	13.83	**14.47**	13.74

4.8. Comparison of Our Models with Results in Literature

The BLEU scores of the models in the literature trained with the Flickr8k dataset like our model, are shown below.

	BRNN [1]	m-RNN [12]	Soft-Attention [13]
BLEU-1	57.9	56.5	67
BLEU-2	38.3	38.6	44.8
BLEU-3	24.5	25.6	29.9
BLEU-4	16.0	17.0	19.5

5. Discussion

In this project we have successfully implemented various neural network models, which were trainable bidirectional GRU, trainable LSTM, standart LSTM, standart GRU, bidirectional GRU and bidirectional LSTM. The models were run on the Flickr dataset, and the captions were embedded through GloVe.

Looking at all the loss vs epoch graphs it can be seen that the validation loss starts changing in a negligible way showing that the model has reached its most generalizable state. The models could have been trained longer to see the point where the validation loss starts increasing and therefore the model truly starts overfitting. The more rapid decrease in the training loss can be observed when the embedding matrix is made trainable, indicating that the model fits the given data faster in such variations.

In terms of computational effectiveness, we have seen that, unsurprisingly, the GRU speed is faster. The Figures obtained in results show that GRU is generally faster compared to LSTM, which is expected due to the low computational cost, with the exception of trainable models, which used different hardware (and hence yielded different results). The training time gain with the GRU compared with the LSTM was around 100 seconds each epoch, which might be worth the performance loss switching between the models in computation wise restrained cases.

We selected the model parameters based on the minimum validation loss which is calculated by using cross-entropy and used BLEU scores to compare the model performance on the test set. The BLEU metric scores indicate similar performances by the generative model variations, parallel to how the graphics show similar patterns. LSTM dominates all corresponding GRU metrics when used in a similar model structure whether it is the bidirectional case or the trainable embedding layer case. Bidirectional RNN implementation shows an improvement over the unidirectional cases. Meanwhile the trainable embedding layer shows an improvement over the non-trainable cases in LSTM but shows no improvement in GRU. Overall, the non-trainable embedded layered bidirectional LSTM model outperforms the rest of the model variations.

The obtained BLEU-4 score of 14.47 compared to the ones in literature is relatively low [1],[12],[13]. However, keeping in mind that the models in literature are trained for higher epoch numbers and implement region of interest identifiers rather than the pretrained ResNet in our test model makes the difference in performance expected. Also, in some cases, our models were able to catch up with the available models in the literature. For example, our bidirectional LSTM with a trainable embedded vector is fairly similar to BRNN on Flickr8k dataset. All in all, the rest of the model being kept constant while changing the generative layer allowed for a reliable benchmarking environment, which was our goal from the start.

When we look at the captioned images at the Appendix, in Figure A.2, we can see that the models aren’t able to caption the flowers in the image. When the embedding matrix is trained, we can see that the models can create reasonable and correct captions to the image. This shows that due to increase in trainable variables, models with trainable embedding matrices were able to learn more which causes some previously uncaptionable images to be captioned. In conclusion, we have achieved results that are fairly accurate considering the dataset at hand, and have obtained fairly similar scores compared to other works done in the Neural Networks literature.

6. References

[1] A. Karpathy and L. Fei-Fei, “Deep visual-semantic alignments for generating image descriptions,” 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Apr. 2015.

[2] L. Fei-Fei, “Deep visual-semantic alignments for generating image descriptions,” 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Apr. 2015.

[3] M. Cornia, L. Baraldi, G. Serra, and R. Cucchiara, “Paying more attention to saliency,” ACM Transactions on Multimedia Computing, Communications, and Applications, vol. 14, no. 2, pp. 1–21, May 2018.

[4] L. Zhou, H. Palangi, L. Zhang, H. Hu, J. Corso, and J. Gao, “Unified vision-language pre-training for image captioning and VQA,” Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, no. 07, pp. 13041–13049, Dec. 2019.

[5] J. Pennington, R. Socher, and C. Manning, “Glove: Global vectors for word representation,” Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2014.

[6] E. Muñoz, “Getting started with NLP: Word embeddings, glove and text classification,” Eduardo Muñoz NLP Blog, 15-Aug-2020. [Online]. Available: https://edumunozsala.github.io/BlogEms/jupyter/nlp/classification/embeddings/python/2020/08/15/Intro_NLP_WordEmbeddings_Classification.html. [Accessed: 14-Jan-2022].

[7] A. Choudhary, M. Ramnani, and A. Nandi, “Hands-on guide to word embeddings using glove,” Analytics India Magazine, 07-Oct-2021. [Online]. Available: https://analyticsindiamag.com/hands-on-guide-to-word-embeddings-using-glove/. [Accessed: 14-Jan-2022].

[8] T. Ganegedara, “Light on math ML: Intuitive Guide to Understanding Glove embeddings,” Medium, 15-Nov-2021. [Online]. Available: https://towardsdatascience.com/light-on-math-ml-intuitive-guide-to-understanding-glove-embeddings-b13b4f19c010. [Accessed: 13-Jan-2022].

[9] B. Arun, J. Bockenek, A. Chauhan, and J. A. Khan, “Overview of models - jabocken.github.io,” Image Classification for cdiscount.com, 2017. [Online]. Available: https://jabocken.github.io/ML2017Fall/docs/models/. [Accessed: 13-Jan-2022].

[10] M. Phi, “Illustrated guide to LSTM's and GRU's: A step by step explanation,” Medium, 28-Jun-2020. [Online]. Available: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf2. [Accessed: 14-Jan-2022].

[11] “Bleu,” Wikipedia, 02-Dec-2021. [Online]. Available: https://en.wikipedia.org/wiki/BLEU. [Accessed: 14-Jan-2022]

[12] J. Mao, W. Xu, Y. Yang, J. Wang, Z. Huang, and A. Yuille, “Deep Captioning with Multimodal Recurrent Neural Networks (m-RNN),” Dec. 2014.

[13] K. Xu, J. Lei Ba, R. Kiros , K. Cho, A. Courville, R. Salakhutdinov, R. S. Zemel , and Y. Bengio, “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention,” Apr. 2016.

7. Appendix

A. Captioned Images

```
Prediction:
LSTM: a UNKNOWN UNKNOWN plane flying through the air
GRU: a UNKNOWN UNKNOWN plane flying over a UNKNOWN
BLSTM: a large UNKNOWN flying through the air over a lake
BGRU: a UNKNOWN UNKNOWN plane is flying in the air
Trainable BLSTM: a large UNKNOWN flying through a blue sky
Trainable BGRU: a UNKNOWN UNKNOWN plane is flying in the sky
Actual:
a UNKNOWN UNKNOWN airplane flying in a blue sky
a small air plane flying UNKNOWN the air
a small airplane flying through a light blue sky
there is an old plane flying in the sky
a yellow and black fighter plane flying in blue sky
```

Figure A.1

Prediction:
LSTM: a UNKNOWN UNKNOWN UNKNOWN UNKNOWN UNKNOWN UNKNOWN UNKNOWN
GRU: a UNKNOWN UNKNOWN UNKNOWN UNKNOWN UNKNOWN UNKNOWN UNKNOWN
BLSTM: a UNKNOWN of a UNKNOWN UNKNOWN UNKNOWN UNKNOWN
BGRU: a UNKNOWN UNKNOWN UNKNOWN UNKNOWN UNKNOWN UNKNOWN UNKNOWN
Trainable BLSTM: a vase of flowers sitting on a table
Trainable BGRU: a vase with a flower in it sitting on a table
Actual:
some orange flowers in a tall silver vase
orange red and white flowers in vases on tables
a small silver vase with some orange flowers in it
orange flowers UNKNOWN vases in the flower shop
a metal vase of flowers on top of a counter

Figure A.2

Prediction:
LSTM: a man in a suit and tie standing next to a woman
GRU: a man in a suit and tie standing next to a woman
BLSTM: a man in a suit and tie is standing in front of a UNKNOWN
BGRU: a man in a suit and tie standing next to a woman
Trainable BLSTM: a man in a suit and tie is smiling
Trainable BGRU: a man in a suit and tie standing next to a UNKNOWN
Actual:
a group of people sitting side by side to take a photograph
a black and white photo of a group of people in UNKNOWN UNKNOWN
a big group people that are posing xFor a picture
a UNKNOWN of men and woman wearing UNKNOWN clothing
a black and white picture of a group of people

Figure A.3

Prediction:
LSTM: a man in a UNKNOWN UNKNOWN UNKNOWN a frisbee
GRU: a man UNKNOWN a frisbee in a grassy field
BLSTM: a zebra standing next to a UNKNOWN tree
BGRU: a zebra standing in a field next to a tree
Trainable BLSTM: a zebra standing in a field next to a tree
Trainable BGRU: a zebra standing next to a UNKNOWN zebra
Actual:
a field with various UNKNOWN and trees in the background
a group of zebras in a field by some trees
the group of zebras are in the field
zebras and UNKNOWN UNKNOWN living together on the UNKNOWN
many animals in a field with trees and bushes in the background

Figure A.4

Prediction:
LSTM: a man is UNKNOWN a UNKNOWN UNKNOWN UNKNOWN
GRU: a man is playing tennis on a tennis court
BLSTM: a desk with a computer and a laptop on it
BGRU: a desk with a laptop and a desktop computer
Trainable BLSTM: a desk with a laptop and a UNKNOWN
Trainable BGRU: a computer desk with a laptop and a desktop computer
Actual:
a desktop computer sitting on top of a wooden desk
a computer is UNKNOWN on on top of a desk
a desktop computer sits UNKNOWN a cluttered desk
a desk containing a computer monitor and keyboard
computer station with one computer and many UNKNOWN

Figure A.5

Prediction:
LSTM: a man in a suit and tie standing next to a woman
GRU: a man in a suit and tie standing next to a woman
BLSTM: a man in a suit and tie is standing in front of a UNKNOWN
BGRU: a man in a suit and tie standing next to a woman
Trainable BLSTM: a man in a suit and tie is smiling
Trainable BGRU: a man in a suit and tie standing next to a UNKNOWN
Actual:
an old black and white photo of two men in suits
two men are UNKNOWN wearing suits in this UNKNOWN photo
a black and white photo of two young UNKNOWN next to each other
a man standing leaning on a man sitting
an old black and white photo of two men next to UNKNOWN

Figure A.6

Prediction:
LSTM: a woman swinging a tennis racquet at a tennis ball
GRU: a woman swinging a tennis racquet at a tennis ball
BLSTM: a woman playing tennis on a tennis court
BGRU: a man swinging a tennis racquet at a tennis ball
Trainable BLSTM: a man is playing tennis on a tennis court
Trainable BGRU: a woman UNKNOWN a tennis ball with her racket
Actual:
a man is serving in a game of tennis
a person standing on a tennis court hitting a tennis ball with a racquet
one tennis player jumps to hit the tennis ball on a court
four playing a UNKNOWN games of tennis on a tennis court

Figure A.7

Prediction:
LSTM: a man riding a bike down a street next to a UNKNOWN
GRU: a man riding a bike down a street next to a woman
BLSTM: a man riding a bike down a street next to a tall building
BGRU: a man is riding a bike with a dog
Trainable BLSTM: a bathroom with a toilet and a sink
Trainable BGRU: a man riding a bike with a surfboard on top of it
Actual:
a man in white shirt on bicycle with a dog riding in the back
a man on a bicycle with a dog sitting in the back of the bike
an old photo of a person on a bike in a parking lot
a man and his dog riding on a bike
there is a man riding a bike with a dog on the back

Figure A.8

Prediction:
LSTM: a train traveling down train tracks next to a building
GRU: a train traveling down tracks next to a train station
BLSTM: a train traveling down tracks next to a forest
BGRU: a train is traveling down the tracks near a building
Trainable BLSTM: a train is traveling down the tracks near a UNKNOWN
Trainable BGRU: a train traveling down train tracks next to a building
Actual:
a train moving along a track during the day
a train UNKNOWN by a road on a railroad track
yellow train engine UNKNOWN UNKNOWN cars past a UNKNOWN crossing
a large long train on a steel track
a yellow train engine pulling train cars over road

Figure A.9

Prediction:
LSTM: a man in a UNKNOWN UNKNOWN UNKNOWN a frisbee
GRU: a zebra standing in a field with a UNKNOWN
BLSTM: a zebra standing next to a UNKNOWN tree
BGRU: a zebra standing in a field next to a tree
Trainable BLSTM: a zebra standing in a field next to a tree
Trainable BGRU: a zebra standing next to a wooden fence
Actual:
a herd of zebras UNKNOWN around a UNKNOWN area
three zebras in an UNKNOWN feeding on a UNKNOWN ground
zebras that are hanging around and eating something off the ground
a couple of zebras that are out in a dirt field
two zebra stand on dirt in a fenced off area

Figure A.10

Prediction:
LSTM: a man riding a horse in a UNKNOWN
GRU: a man riding a horse drawn carriage on a beach
BLSTM: a man riding a horse on a beach
BGRU: a man riding a horse drawn carriage down a street
Trainable BLSTM: a horse drawn carriage riding down a street
Trainable BGRU: a man riding a horse drawn carriage with a horse
Actual:
a person riding a UNKNOWN and horse down the street
a small horse pulling a two UNKNOWN UNKNOWN
a house pulling a cart with UNKNOWN on it
a horse is pulling a carriage in the street
a horse UNKNOWN an UNKNOWN carriage on a UNKNOWN road

Figure A.11

Prediction:
LSTM: a man is standing in front of a television
GRU: a man is playing tennis on a tennis court
BLSTM: a man is playing a video game xWhile
BGRU: a man is standing next to a woman
Trainable BLSTM: a man in a UNKNOWN shirt UNKNOWN a UNKNOWN
Trainable BGRU: a man is UNKNOWN a UNKNOWN UNKNOWN UNKNOWN
Actual:
a man UNKNOWN over a laptop computer next to UNKNOWN equipment
a man in a black shirt and black pants on a laptop in a UNKNOWN
two men UNKNOWN looking at the screen on a laptop
man drinking a beer on a white computer
two men working on a computer on a wooden table

Figure A.12

Prediction:
LSTM: a man riding a surfboard on a wave in the ocean
GRU: a man riding a wave on top of a surfboard
BLSTM: a man is riding a wave on a surfboard
BGRU: a man is UNKNOWN a soccer ball on a soccer field
Trainable BLSTM: a man riding a wave on top of a surfboard
Trainable BGRU: a man riding a wave on top of a surfboard
Actual:
two surfers on their boards on a UNKNOWN day
UNKNOWN boarder in middle of UNKNOWN water xWhile on board
a man riding on top of a UNKNOWN board in the ocean
a UNKNOWN is UNKNOWN xWhile another person lies on their board nearby

Figure A.13

Prediction:
LSTM: a man is UNKNOWN a UNKNOWN UNKNOWN UNKNOWN
GRU: a laptop computer sitting on top of a desk
BLSTM: a laptop computer sitting on top of a desk
BGRU: a laptop computer sitting on top of a desk
Trainable BLSTM: a laptop computer sitting on top of a wooden desk
Trainable BGRU: a laptop computer sitting on top of a wooden desk
Actual:
a woman using a laptop computer on top of a wooden desk
two female hands and a black laptop and a camera and controller
a table with a camera a remote and a person using a laptop computer

Figure A.14

Prediction:
LSTM: a man is playing frisbee on a beach
GRU: a man is UNKNOWN a frisbee in a park
BLSTM: a man is jumping a skateboard over a UNKNOWN
BGRU: a man is jumping a skateboard over a ramp
Trainable BLSTM: a man is UNKNOWN a frisbee in a field
Trainable BGRU: a man is riding a skateboard down a street
Actual:
a man on the side of the road holding a frisbee
a man standing on a park holding a white frisbee
a man with a ball of some UNKNOWN
man catching a frisbee xWhile outside his home
a man UNKNOWN UNKNOWN to xCatch a frisbee

Figure A.15

Prediction:
LSTM: a man in a UNKNOWN UNKNOWN UNKNOWN a UNKNOWN
GRU: a man is UNKNOWN a UNKNOWN UNKNOWN UNKNOWN
BLSTM: a man is UNKNOWN a UNKNOWN UNKNOWN UNKNOWN
BGRU: a man is standing next to a woman
Trainable BLSTM: a man in a suit and tie is standing in front of a UNKNOWN
Trainable BGRU: a man is standing in front of a UNKNOWN
Actual:
a man and a woman standing next to each other in front of a crowd
a UNKNOWN couple UNKNOWN their wedding with their UNKNOWN
a large group of people UNKNOWN at a UNKNOWN
a man and woman UNKNOWN their glass xFor a UNKNOWN
a young couple in UNKNOWN UNKNOWN UNKNOWN UNKNOWN the crowd

Figure A.16

Prediction:
LSTM: a man is playing a video game with a remote
GRU: a man is playing a video game with a remote controller
BLSTM: a man playing a video game xWhile others watch
BGRU: a man standing in a living room holding a wii remote
Trainable BLSTM: a man standing in a living room holding a game controller
Trainable BGRU: a man is playing a video game with a remote controller
Actual:
a man with UNKNOWN glasses who is holding a wii remote
a young man wearing glasses and a red shirt holding a small white device
a young man wearing a red UNKNOWN is holding his wii UNKNOWN
boy posing in a UNKNOWN UNKNOWN with his wii controller
boy with glasses and red shirt holding UNKNOWN

Figure A.17

Prediction:
LSTM: a woman swinging a tennis racquet at a tennis ball
GRU: a woman swinging a tennis racquet at a tennis ball
BLSTM: a tennis player UNKNOWN to hit a tennis ball
BGRU: a woman hitting a tennis ball with a tennis racket
Trainable BLSTM: a woman playing tennis on a tennis court
Trainable BGRU: a woman UNKNOWN a tennis ball with her racket
Actual:
a kid is standing on a tennis court with a racket
a young girl playing tennis at a tennis court
a young girl playing tennis on an UNKNOWN court
a young girl standing on top of a tennis court holding a racquet
a kid holding a racket ready to UNKNOWN the ball

Figure A.18

Prediction:
LSTM: a man in a wet suit is playing frisbee
GRU: a man UNKNOWN a soccer ball with his UNKNOWN
BLSTM: a woman UNKNOWN a soccer ball with a tennis racquet
BGRU: a man is playing a game of soccer
Trainable BLSTM: a man UNKNOWN a soccer ball on a soccer field
Trainable BGRU: a man is playing a game of soccer
Actual:
a man standing next to another man xWhile UNKNOWN a soccer ball
two men in a soccer field UNKNOWN a ball
two soccer players UNKNOWN each other xFor the ball
two soccer players UNKNOWN to be UNKNOWN each other
two soccer players UNKNOWN against each other as they UNKNOWN to get to the ball

Figure A.19

B. Code

```
#!/usr/bin/env python
# coding: utf-8

# # EEE 443 - Final Project - Image Captioning
#
# ## Group 12:
#
# Atakan Topcu, Ata Korkusuz, Ata Yavuzyılmaz, Dora Tütüncü
#
#
#-----------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------
#

# In[283]:


import random
import shutil
import numpy as np
import matplotlib.pyplot as plt
import h5py
import seaborn as sn
import requests
from PIL import Image
import os
import requests
import pandas as pd
file_train="eee443_project_dataset_train.h5"
file_test="eee443_project_dataset_test.h5"


# ### Next Step:
# We first create function to read from the given h5 file.
#
#
#-----------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------

# In[275]:


def Dataset_Read(file_train):
    '''
    Funtction to read given h5 train file
    '''
```

```
    f = h5py.File(file_train, 'r') #Load Data
    print("Keys: %s" % list(f.keys()))
    train_cap = f['train_cap'][()] #Captions for training images (contains 17 indices from the vocabulary for
each image
    train_imid = f['train_imid'][()] #The indices of training images (since a single image can be captioned
several times
    train_url = f['train_url'][()] #Flickr URLs for training images
    word_code = f['word_code'][()] #Dictionary for converting words to vocabulary indices
    train_ims = None  #Do not use train_ims. Otherwise, score is halved!!
    print("Size of captions: ",train_cap.shape)
    print("Size of indices of training images: ",train_imid.shape[0])
    print("Size of URL: ",train_url.shape[0])
    print("Size of Dictionary: ",word_code.shape[0])
    return train_imid, train_cap, train_url, word_code

def Test_Read(file_test):
    '''
    Funtction to read given h5 test file
    '''
    f = h5py.File(file_test, 'r') #Load Data
    print("Keys: %s" % list(f.keys()))
    test_caps = f['test_caps'][()] #Captions for test images (contains 17 indices from the vocabulary for each
image
    test_imid = f['test_imid'][()] #The indices of test images (since a single image can be captioned several
times
    test_url = f['test_url'][()] #Flickr URLs for test images
    test_ims = None  #Do not use test_ims. Otherwise, score is halved!!
    print("Size of captions: ", test_caps.shape)
    print("Size of indices of test images: ", test_imid.shape[0])
    print("Size of URL: ", test_url.shape[0])
    return test_imid, test_caps, test_url


# ### Next Step:
# We create a new directory for downloading the images from the Flickr URL addresses. Also verification
method is implemented. We look at two factors for images that will be used in the project:
#
# 1. HTTP Status Code (Needs to be 200)
# 2. JPEG Format
#
#
#
-----------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------

# In[297]:
```

```
#root_dir = os.getcwd()
direc='C:\\Users\\ataka\\Desktop\\EEE443_Proje' #Change accordingly
imgs_dir = direc + '\\Downloaded_Ims'
exports_dir = direc + '\\Exported_Ims'
testNEW=direc + '\\testNEW'
from tqdm.notebook import tqdm #For progress bar representation. Purely decoration purposes!
os.chdir('C:\\Users\\ataka\\Desktop\\EEE443_Proje')#Change accordingly
_, _, test_url=Test_Read(file_test)
def Download(directory,URLa):
    """
    Function to save images from given urls
    """
    os.chdir('C:\\Users\\ataka\\Desktop\\EEE443_Proje')#Change accordingly
    train_imid,train_cap,URL,word_code=Dataset_Read(file_train)
    if not os.path.exists(directory): #Create the directory if it does not exist in the first place.
        os.makedirs(directory)
    os.chdir(directory)
    print('Current working directory: ', os.getcwd()) #Check

    headers = {'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:95.0) Gecko/20100101
Firefox/95.0'}#Change accordingly
    for URL in tqdm(range(6686,len(URLa))):
        url = URLa[URL].decode('utf-8')
        response = requests.get(url, allow_redirects=True, stream=True, headers=headers)
        #If HTTP 200 is achieved, write file
        if response.status_code == 200:
            with open(directory+ '/' + str(URL), 'wb') as outfile:
                outfile.write(response.content)

    os.chdir('..')
    print('Current working directory: ', os.getcwd())#Check

    return


# In[299]:


#Download(testNEW,test_url)


# In[277]:


def Image_Verification(directory,train_url):
    os.chdir('C:\\Users\\ataka\\Desktop\\EEE443_Proje')#Change accordingly
    os.chdir(directory)
    corrupt_index = []
```

```
    d_index=[]
    print('Current working directory: ', os.getcwd()) #Check
    for URL in range(len(train_url)):
        try:
            img = Image.open(directory + '/' + str(URL))
            if img.format == 'JPEG':
                continue
            else:
                print('Invalid image type, not JPEG')
                corrupt_index.append(directory + '\\' + str(URL))
                d_index.append(URL)
        except:
            continue
    os.chdir('..')
    print('Current working directory: ', os.getcwd())#Check
    # Take note of the problematic image files
    with open('D_Ims.txt', 'w') as txt_file:
        for items in corrupt_index:
            txt_file.write('%s\n' % items)
    return d_index


# In[278]:


os.getcwd()


# In[279]:


#os.chdir('/content/drive/MyDrive/EE443 Project')
os.chdir(direc)
train_imid,train_cap,train_url,word_code=Dataset_Read(file_train)
#Del_index=Image_Verification(imgs_dir,train_url)


# ### Next Step:
#
# We start implementing the transfer model. In our case, we will be using Inception Resnet V2. Other models can also be used such as Inception v3 or VGGA16. In order to extract features using such models, consider two factors:
#
# 1. Input Size: Input size must be (255,255,3) for the Inception Resnet V2 model. Thus, you have to preprocess your images.
# 2. Get rid of the final layer in your model. We want the pre-Trained Model as Feature Extractor Preprocessor. Thus, you have to get rid of the last layer as it is used for the original classification task.
#
```

```
#
-----------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------

# In[280]:


import tensorflow as tf
from tensorflow.keras.applications.inception_resnet_v2 import InceptionResNetV2, preprocess_input
from tensorflow.keras.applications.inception_v3 import InceptionV3, preprocess_input
from tensorflow.keras.preprocessing import image


"""
Target Model:

        InceptionResNetV2(
            include_top=True, weights='imagenet', input_tensor=None,
            input_shape=None, pooling=None, classes=1000,
            classifier_activation='softmax', **kwargs
        )
"""

#If time allows different methods can also be utilized such as VGG16.
def Feature_Extraction_Model():
    tf.keras.backend.clear_session() # clears previous session if this code is run multiple time
    #Pre-Trained Model as Feature Extractor Preprocessor
    ResNet = InceptionResNetV2(include_top=True, weights='imagenet', input_shape=(299,299,3))
    ResNet.trainable = False
    ResNet = tf.keras.Model(ResNet.input, ResNet.layers[-2].output)# remove the output layer
    return ResNet

def ImageEncoding(directory, imageIndex,model):
    #Process images so that they can be inputted in the model correctly.
    img = tf.io.read_file(directory+imageIndex)
    img = tf.image.decode_jpeg(img, channels=3)
    img = tf.image.resize(img, (299, 299))
    img = tf.expand_dims(img, axis=0)
    #print(img.shape)
    img = preprocess_input(img)
    encoding = model.predict(img) #Extracts features into single 2D vector.
    #print(encoding.shape)
    return encoding    #Should be of size (1,1536)


# ### Next Step:
#
```

```
# In this step, we define the final function for creating the encoded features using our transfer model. There are three factors to consider:
#
# 1. Problematic Files: Previously, we have defined problematic non-JPEG files in a .txt file of our main directory. Before you start encoding, you have to make sure these files are not used in the process for the sake of performance and reliability.
#
# 2. Iteratively, you will go through each file in the train directory and start encoding. You will return a list called Encoded_Features which will be used in future.
#
# 3. ***Never stack model:*** This is extremely important. You should define the model outside of the Create_Features function. Otherwise, it will stack up in the for loop and it will take significant time (days) to compile the images.
#
#
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

# In[281]:


from tqdm import tqdm
def Create_Features(directory,model):
    #os.chdir('C:\\Users\\ataka\\Desktop\\EEE443_Proje')#Change accordingly
    txt_file=open('D_Ims.txt', 'r') #If you named the .txt file differently, change accordingly.
    Problem_files=txt_file.readlines()#Read the problematic file names.

    os.chdir(directory) #Change to training directory!
    print(os.getcwd())#Check
    Encoded_Features = []
    for img in tqdm(os.listdir()): #Scans the directory. img is a string!
        check=os.getcwd()+'\\'+img+'\n'
        if not Problem_files.count(check):
            tmp = ImageEncoding(os.getcwd()+ '\\',img, model)
            Encoded_Features.append(tmp)
        else:
            print("Warning:non_JPEG format detected. It will not be used!")

    Encoded_Features=np.array(Encoded_Features)

Encoded_Features=Encoded_Features.reshape(Encoded_Features.shape[0],Encoded_Features.shape[2])
    return Encoded_Features #Should output (x,1536) where x is the number of samples.


# In[8]:


os.getcwd()
```

```
# ### Next Step:
#
# We don't want to start this encoding process over and over as it will take time. Thus, save it as pickle file
and load it later when needed.
#
#
-----------------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------

# In[187]:


import pickle #Call the function


# In[ ]:


os.chdir(imgs_dir)
Index=[]
for img in tqdm(os.listdir()): #Scans the directory. img is a string!
        check=os.getcwd()+'\\'+img+'\n'
        if not Problem_files.count(check):
            Index.append(img)


# In[ ]:


print('Encoding finished. Pickling.')
if not os.path.exists(exports_dir): #Create the directory if it does not exist in the first place.
        os.makedirs(exports_dir)
os.chdir(exports_dir)

# save to file
print(Encoded_Features.shape)
pickle.dump(Encoded_Features, open('Feature_Encodes.pkl', 'wb'))
pickle.dump(Index, open('Feature_Index.pkl', 'wb'))


os.chdir(direc) # return back to the main directory


# ### Check GPU Availability:
#
```

Tensorflow might cause problem in enabling GPU usage since it requires many packages to be downloaded (CUDA, cuDNN, etc.). Thus, I recommend using Google Collab as it eases this operation. If you are not using Google Collab, you should check the compability of cuDNN and CUDA first as the newest vesions are not backwards compatible
#
Collab --->Edit ---> Notebook Settings ---> Hardware Accelerator ---> GPU ---> Save
#
For Local Jupyter Application, follow the link for setting up CUDA and cuDNN on your environment path.
#
#
https://towardsdatascience.com/installing-tensorflow-with-cuda-cudnn-and-gpu-support-on-windows-10-60693e46e781
#
#
#

In[188]:

```
print("GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
if len(tf.config.list_physical_devices('GPU'))==0:
  print("Warning: Need to Enable GPU")
else:
  print((tf.config.list_physical_devices('GPU')))

#Should enable GPU. Otherwise, project will take time.
```

Load Pickle File Again
#

In[189]:

```
import pickle
print(os.getcwd())
os.chdir("Exported_Ims")
print(os.getcwd())
print(os.listdir())
```

In[ ]:

```
Feature_Encodes = pickle.load(open(exports_dir + "\\Feature_Encodes.pkl",'rb'), encoding='utf8')
Feature_Index = pickle.load(open(exports_dir + "\\Feature_Index.pkl",'rb'), encoding='utf8')
print(Feature_Encodes.shape)
Feature_Index=np.array(Feature_Index)
print(Feature_Index.shape)


# ### Extract Word Dictionary Along with Creating Functions for Captioning
#
----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------

# In[190]:


#Export Word Dictionary. Size should be 1004!
os.chdir(direc)
_,_,_,word_code=Dataset_Read(file_train)
Words = pd.DataFrame(word_code)
Words = Words.sort_values(0, axis=1)
word_columns = np.asarray(Words.columns)
wordDict = {}
for i in range(len(word_columns)):
  word = word_columns[i]
  wordDict[i] = word
print('Size of Extracted Word Dictionary: ', len(wordDict.keys()))


def Caption_string(caption_array):
    """
    Format caption numpy array to a list of string(s).
    """

    Captions_list = []
    caption = ""
    for caps in caption_array:
        for word in caps:
            if (word == 'x_NULL_') or (word == 'x_START_') or (word == 'x_END_'):
                continue #Don't count x_NULL, x_Start and x_END
            elif (word == 'x_UNK_'):
                word = 'UNKNOWN'
            caption += word + " "

        Captions_list.append(caption.strip()) #Remove spaces and append
        caption = ""
    return Captions_list
```

```
def get_key(val,dictionary): # Function to return key for any value
    for key, value in dictionary.items():
        if val == key:
            return value



# In[191]:


wordDict


# In[ ]:


import pickle

pickle.dump(wordDict, open("ixtoword.pkl", 'wb'))


# In[192]:


os.getcwd()


# In[193]:


def Fetch_Caption(image_id, imid, cap, word_dict):
    """
    A function which returns the caption of an image wrt. its ID
    """
    # Query this on train_imid to extract which indices hold the captions for this image:
    indices = np.where(imid == int(image_id))[0] #[0] removes tuple's second element which is dtype=int64
    caps = [] #Extract the caps of the specified image
    for idx in indices:
        cap1 = cap[idx][:]
        caps.append(cap1)
    #Conversion to string
    text_cap = []
    for item in caps:
        temp = []
        for word in item:
            key=get_key(word,word_dict) # Function to return key for any value
            temp.append(key)
        text_cap.append(temp)
```

```
    caps = [list(c) for c in caps] # list comprehension to make arrays list
    '''
    Should return something like:
    [[1, 7, 78, 240, 20, 244, 5, 7, 213, 35, 2, 0, 0, 0, 0, 0, 0], [1, 78, 240, 6, 193, 927, 244, 5, 63, 2, 0, 0, 0, 0,
0, 0, 0], [1, 30, 240, 20, 244, 5, 3, 10, 536, 3, 542, 2, 0, 0, 0, 0, 0], [1, 4, 423, 240, 244, 10, 109, 63, 8, 4,
35, 2, 0, 0, 0, 0, 0], [1, 78, 193, 3, 240, 244, 5, 7, 63, 2, 0, 0, 0, 0, 0, 0, 0]]
    '''
    return caps

# Print captions of an image
def Caption_Print(caps, word_dict):
    i = 0
    text_cap = []

    for item in caps:
        i += 1
        temp = []
        for word in item:
            key=get_key(word,word_dict)
            temp.append(key)
        text_cap.append(temp)
        # x_NULL_ strings are only ignored, not erased from the captions
    Captions_list = Caption_string(text_cap)
    for caption in range(i):
        print('Caption ', caption+1,': ',Captions_list[caption])
    return Captions_list


# # !Important!
#
# **For training data, train index yields the train index-1 image**. So, if the index is 314, the train image
should be 313!!.
#
# **For Test data, train index yields the train index image**. So, if the index is 314, the test image should be
314!!.
#
#
-----------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------

# In[194]:


#Example
cap=Fetch_Caption(314, train_imid, train_cap, wordDict)
Captions_list = Caption_Print(cap,wordDict)
Image.open(imgs_dir+'\\313') #Careful: We decrement the index given in Fetch_Caption!
```

Sub-Divide the Training Data:
Divide your training data as 85% Training and 15% Validation data. Function will output two new dictionaries with sorted IDs.
#
#

In[195]:

```
import random
import math
Feature_Dictionary= dict(zip(Feature_Index, Feature_Encodes)) #We create dictionary from the encoded files (index,features)
def SubDivision(percentage, list_to_divide):
    indices = random.sample(list(list_to_divide.keys()), len(list_to_divide)-math.ceil((1-percentage)*len(list_to_divide)))
    Validation = {}
    Train = {}
    results = list(map(int, indices)) #Convert From String to Int for sorting
    OG=list(map(int, list_to_divide.keys()))
    results.sort(reverse=False) #Ascending Order
    rest=list(set(OG).difference(set(results)))
    rest.sort(reverse=False)
    for idx in results:
        Train[str(idx)]=(list_to_divide[str(idx)])

    for idv in rest:
        Validation[str(idv)]=(list_to_divide[str(idv)])

    return Validation, Train
```

In[ ]:

```
TRAIN_PERCENTAGE = 0.85
Validation, Train=SubDivision(TRAIN_PERCENTAGE, Feature_Dictionary)
print('Length of the Train Data:',len(Train))
print('Length of the Validation Data:',len(Validation))
```

Prepare the input data:
#

Each key with train, validation data contains keys for encoding dictionary. Each key in the encoding corresponds to a single encoded image, and each encoded image corresponds to approximately 4-5 captions.
#
#
#

In[196]:

```
from tqdm.notebook import tqdm #For progress bar representation. Purely decoration purposes!
#from tqdm import tqdm #For progress bar representation

#With this dataset, we will call our previous functions for captioning.
def DataPreparation(imid, cap, KEY_Lists,  encoding_dict , word_dict):
    ENCODES = []
    CAPTIONS = []
    for key in tqdm(list(KEY_Lists.keys())): # Iteration for each key in train or validation (Could also be test dataset)
        key_captions = Fetch_Caption(key, imid, cap, word_dict)
        # Replicate encoding number of caption times.
        for i in range(len(key_captions)):
            ENCODES.append(encoding_dict[key]) #Iterate for number of captions

        # Process the captions:
        for c in key_captions:
            CAPTIONS.append(c)

    return ENCODES,CAPTIONS
```

In[ ]:

```
Train_Encodes, Train_Captions = DataPreparation(train_imid, train_cap, Train, Feature_Dictionary, wordDict)
```

In[ ]:

```
Validation_Encodes, Validation_Captions = DataPreparation(train_imid, train_cap, Validation, Feature_Dictionary, wordDict)
```

Saving the Outputs:

Now that we have processed the train and validation data, we can proceed to save them in pickle file format so that they can be called in the future without any compilation.
#
#

In[ ]:

```
import pickle #Call the function
os.chdir(exports_dir)
print(os.getcwd())

# save to file
pickle.dump(Train_Encodes, open('Train_Encodes.pkl', 'wb'))
pickle.dump(Train_Captions, open('Train_Captions.pkl', 'wb'))

pickle.dump(Validation_Encodes, open('Validation_Encodes.pkl', 'wb'))
pickle.dump(Validation_Captions, open('Validation_Captions.pkl', 'wb'))


os.chdir(direc) # return back to the main directory
```

In[ ]:

```
print('Train Encode Shape:',np.array(Train_Encodes).shape)
print('Train Caption Shape:',np.array(Train_Captions).shape)

print('Validation Encode Shape:',np.array(Validation_Encodes).shape)
print('Validation Caption Shape:',np.array(Validation_Captions).shape)
```

Load the Data Back Again:
This way, we create checkpoints where a problem occurs we can just load the file instead of compiling.
#
#

In[ ]:

```
from google.colab import drive

drive.mount("/content/drive")
```

In[197]:

```
import pickle #Call the function
import os
#exports_dir = "/content/drive/MyDrive/EE443 Project/Exported_Ims"
os.chdir(exports_dir)
print(os.getcwd())

Train_Encodes = pickle.load(open(exports_dir + "/Train_Encodes.pkl",'rb'), encoding='utf8')
Train_Captions = pickle.load(open(exports_dir + "/Train_Captions.pkl",'rb'), encoding='utf8')
Validation_Encodes = pickle.load(open(exports_dir + "/Validation_Encodes.pkl",'rb'), encoding='utf8')
Validation_Captions = pickle.load(open(exports_dir + "/Validation_Captions.pkl",'rb'), encoding='utf8')
```

In[198]:

```
Validation_Captions[0]
```

In[199]:

```
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical

def data_generator(descriptions, photos, wordtoix, max_length, num_photos_per_batch):
    X1, X2, y = [], [], []
    n=0
    vocab_size = len(wordtoix)

    while 1:
        for i in range(len(descriptions)):
            n += 1

            photo = photos[i]
            seq = descriptions[i]
            for j in range(len(seq)):
                # split into input and output pair
                in_seq, out_seq = seq[:j], seq[j]
                # 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 = [], [], []
                n=0
    return X1, X2, y


# In[200]:


os.getcwd()


# In[201]:


os.chdir(direc)
Embedding_Matrix = pickle.load(open("embedding_matrix.pkl",'rb'), encoding='utf8')


# In[202]:


from keras.layers import LSTM, Embedding, TimeDistributed, Dense, RepeatVector,
Activation, Flatten, Reshape, concatenate, Dropout, BatchNormalization, GRU, Bidirectional
from keras.layers.merge import add
from keras.models import Model
from keras import Input, layers
from keras import optimizers

def model_lstm(embedding_matrix):
  vocab_size = embedding_matrix.shape[0]
  embed_dim = embedding_matrix.shape[1]
  max_length = 17

  inputs1 = Input(shape=(1536,))
  drop1 = Dropout(0.5)(inputs1)
  drop2 = Dense(256, activation='relu')(drop1)
  inputs2 = Input(shape=(max_length,))
  se1 = Embedding(vocab_size, embed_dim, mask_zero=True, weights=[embedding_matrix], trainable =
False)(inputs2)
  se2 = Dropout(0.5)(se1)
  se3 = LSTM(256)(se2)
  decoder1 = add([drop2, se3])
```

```
  decoder2 = Dense(256, activation='relu')(decoder1)
  output = Dense(vocab_size, activation='softmax')(decoder2)
  model = Model(inputs=[inputs1, inputs2], outputs=output)
  model.summary()

  return model
model_lstm(Embedding_Matrix)


# In[203]:


def model_bilstm(embedding_matrix):
  vocab_size = embedding_matrix.shape[0]
  embed_dim = embedding_matrix.shape[1]
  max_length = 17

  inputs1 = Input(shape=(1536,))
  drop1 = Dropout(0.5)(inputs1)
  drop2 = Dense(512, activation='relu')(drop1)
  inputs2 = Input(shape=(max_length,))
  se1 = Embedding(vocab_size, embed_dim, mask_zero=True, weights=[embedding_matrix], trainable =
False)(inputs2)
  se2 = Dropout(0.5)(se1)
  # The Bidirectional outputs two sequences: forward and backward. The merge_mode parameter selects
what to do with these sequences.
  # currently, the model concatenates the sequences. (concat)
  se3 = Bidirectional(LSTM(256), merge_mode = 'concat')(se2)
  decoder1 = add([drop2, se3])
  decoder2 = Dense(256, activation='relu')(decoder1)
  output = Dense(vocab_size, activation='softmax')(decoder2)
  model = Model(inputs=[inputs1, inputs2], outputs=output)
  model.summary()

  return model
model_bilstm(Embedding_Matrix)


# In[204]:


def model_gru(embedding_matrix):
  vocab_size = embedding_matrix.shape[0]
  embed_dim = embedding_matrix.shape[1]
  max_length = 17

  inputs1 = Input(shape=(1536,))
  drop1 = Dropout(0.5)(inputs1)
```

```
  drop2 = Dense(256, activation='relu')(drop1)
  inputs2 = Input(shape=(max_length,))
  se1 = Embedding(vocab_size, embed_dim, mask_zero=True, weights=[embedding_matrix], trainable =
False)(inputs2)
  se2 = Dropout(0.5)(se1)
  se3 = GRU(256)(se2)
  decoder1 = add([drop2, se3])
  decoder2 = Dense(256, activation='relu')(decoder1)
  output = Dense(vocab_size, activation='softmax')(decoder2)
  model = Model(inputs=[inputs1, inputs2], outputs=output)
  model.summary()

  return model
model_gru(Embedding_Matrix)


# In[205]:


def model_biGru(embedding_matrix):
  vocab_size = embedding_matrix.shape[0]
  embed_dim = embedding_matrix.shape[1]
  max_length = 17

  inputs1 = Input(shape=(1536,))
  drop1 = Dropout(0.5)(inputs1)
  drop2 = Dense(512, activation='relu')(drop1)
  inputs2 = Input(shape=(max_length,))
  se1 = Embedding(vocab_size, embed_dim, mask_zero=True, weights=[embedding_matrix], trainable =
False)(inputs2)
  se2 = Dropout(0.5)(se1)
  # The Bidirectional outputs two sequences: forward and backward. The merge_mode parameter selects
what to do with these sequences.
  # currently, the model concatenates the sequences. (concat)
  se3 = Bidirectional(GRU(256), merge_mode= 'concat')(se2)
  decoder1 = add([drop2, se3])
  decoder2 = Dense(256, activation='relu')(decoder1)
  output = Dense(vocab_size, activation='softmax')(decoder2)
  model = Model(inputs=[inputs1, inputs2], outputs=output)
  model.summary()

  return model
model_biGru(Embedding_Matrix)


# In[208]:
```

```
def trainable_bilstm(embedding_matrix):
  vocab_size = embedding_matrix.shape[0]
  embed_dim = embedding_matrix.shape[1]
  max_length = 17

  inputs1 = Input(shape=(1536,))
  drop1 = Dropout(0.5)(inputs1)
  drop2 = Dense(512, activation='relu')(drop1)
  inputs2 = Input(shape=(max_length,))
  se1 = Embedding(vocab_size, embed_dim, mask_zero=True, weights=[embedding_matrix])(inputs2)
  se2 = Dropout(0.5)(se1)
  # The Bidirectional outputs two sequences: forward and backward. The merge_mode parameter selects
what to do with these sequences.
  # currently, the model concatenates the sequences. (concat)
  se3 = Bidirectional(LSTM(256), merge_mode = 'concat')(se2)
  decoder1 = add([drop2, se3])
  decoder2 = Dense(256, activation='relu')(decoder1)
  output = Dense(vocab_size, activation='softmax')(decoder2)
  model = Model(inputs=[inputs1, inputs2], outputs=output)
  model.summary()

  return model
trainable_bilstm(Embedding_Matrix)


# In[ ]:


def trainable_biGru(embedding_matrix):
  vocab_size = embedding_matrix.shape[0]
  embed_dim = embedding_matrix.shape[1]
  max_length = 17

  inputs1 = Input(shape=(1536,))
  drop1 = Dropout(0.5)(inputs1)
  drop2 = Dense(512, activation='relu')(drop1)
  inputs2 = Input(shape=(max_length,))
  se1 = Embedding(vocab_size, embed_dim, mask_zero=True, weights=[embedding_matrix])(inputs2)
  se2 = Dropout(0.5)(se1)
  # The Bidirectional outputs two sequences: forward and backward. The merge_mode parameter selects
what to do with these sequences.
  # currently, the model concatenates the sequences. (concat)
  se3 = Bidirectional(GRU(256), merge_mode= 'concat')(se2)
  decoder1 = add([drop2, se3])
  decoder2 = Dense(256, activation='relu')(decoder1)
  output = Dense(vocab_size, activation='softmax')(decoder2)
  model = Model(inputs=[inputs1, inputs2], outputs=output)
  model.summary()
```

```
  return model
trainable_biGru(Embedding_Matrix)


# # Train the models

# In[214]:



epochs = 15
number_pics_per_batch = 128
max_length = 17
steps = len(Train_Captions)//number_pics_per_batch
val_steps = len(Validation_Captions)//number_pics_per_batch


# In[ ]:


#import os
#os.chdir("/content/drive/MyDrive/EE443 Project")


# In[210]:


wordtoix = pickle.load(open("wordtoix.pkl",'rb'), encoding='utf8')


# In[212]:


import numpy as np


# In[213]:


os.chdir(direc)
os.getcwd()


# ## Train LSTM with Non-Trainable Embedding Matrix

# In[34]:
```

```
from tensorflow.keras.optimizers import Adam

lstm_model = model_lstm(Embedding_Matrix)
lstm_model.compile(loss='categorical_crossentropy', optimizer=Adam(learning_rate=0.001),
metrics=['accuracy'])

generator = data_generator(Train_Captions, Train_Encodes, wordtoix, max_length,
number_pics_per_batch)
val_generator = data_generator(Validation_Captions, Validation_Encodes, wordtoix, max_length,
number_pics_per_batch)
hist = lstm_model.fit(generator, epochs=epochs, steps_per_epoch=steps, verbose=1,
validation_data=val_generator, validation_steps=val_steps)
lstm_model.save('v2_lstmweights.h5')
```

```
# In[35]:
```

```
import matplotlib.pyplot as plt

loss = hist.history['loss']
val_loss = hist.history['val_loss']
plt.plot(loss)
plt.plot(val_loss)
plt.legend(['loss', 'val_loss'])
plt.title('Loss Plot')
plt.show()
```

```
# ## Train Bidirectional LSTM with Trainable Embedding Matrix

# In[215]:
```

```
from tensorflow.keras.optimizers import Adam

trainable_bilstm_model = trainable_bilstm(Embedding_Matrix)
trainable_bilstm_model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

generator = data_generator(Train_Captions, Train_Encodes, wordtoix, max_length,
number_pics_per_batch)
```

```
val_generator = data_generator(Validation_Captions, Validation_Encodes, wordtoix, max_length,
number_pics_per_batch)
hist_trainable_bilstm = trainable_bilstm_model.fit(generator, epochs=epochs, steps_per_epoch=steps,
verbose=1, validation_data=val_generator, validation_steps=val_steps)
trainable_bilstm_model.save('v3_trainable_bilstm.h5')
```

In[38]:

```
import matplotlib.pyplot as plt

loss = hist_trainable_bilstm.history['loss']
val_loss = hist_trainable_bilstm.history['val_loss']
plt.plot(loss)
plt.plot(val_loss)
plt.legend(['loss', 'val_loss'])
plt.title('Loss Plot')
plt.show()
```

Train Bidirectional GRU with Non-Trainable Embedding Matrix

In[40]:

```
from tensorflow.keras.optimizers import Adam

biGRU_model = model_biGru(Embedding_Matrix)
biGRU_model.compile(loss='categorical_crossentropy', optimizer=Adam(learning_rate=0.001),
metrics=['accuracy'])

generator = data_generator(Train_Captions, Train_Encodes, wordtoix, max_length,
number_pics_per_batch)
val_generator = data_generator(Validation_Captions, Validation_Encodes, wordtoix, max_length,
number_pics_per_batch)
hist_biGRU_model = biGRU_model.fit(generator, epochs=epochs, steps_per_epoch=steps, verbose=1,
validation_data=val_generator, validation_steps=val_steps)
biGRU_model.save('v2_biGRU.h5')
```

In[41]:

```
import matplotlib.pyplot as plt

loss = hist_biGRU_model.history['loss']
val_loss = hist_biGRU_model.history['val_loss']
plt.plot(loss)
plt.plot(val_loss)
plt.legend(['loss', 'val_loss'])
plt.title('Loss Plot')
plt.show()
```

In[ ]:

```
import numpy as np
from tensorflow.keras.optimizers import Adam

bilstm_model = model_bilstm(Embedding_Matrix)

bilstm_model.compile(loss='categorical_crossentropy', optimizer=Adam(learning_rate=0.001),
metrics=['accuracy'])

generator = data_generator(Train_Captions, Train_Encodes, wordtoix, max_length,
number_pics_per_batch)
val_generator = data_generator(Validation_Captions, Validation_Encodes, wordtoix, max_length,
number_pics_per_batch)
hist_bilstm_model = bilstm_model.fit(generator, epochs=epochs, steps_per_epoch=steps, verbose=1,
validation_data=val_generator, validation_steps=val_steps)
bilstm_model.save('v2_bilstm.h5')

loss = hist_bilstm_model.history['loss']
val_loss = hist_bilstm_model.history['val_loss']
plt.plot(loss)
plt.plot(val_loss)
plt.legend(['loss', 'val_loss'])
plt.title('Loss Plot')
plt.show()
```

In[ ]:

```
gru_model = model_gru(Embedding_Matrix)
```

```
gru_model.compile(loss='categorical_crossentropy', optimizer=Adam(learning_rate=0.001),
metrics=['accuracy'])

generator = data_generator(Train_Captions, Train_Encodes, wordtoix, max_length,
number_pics_per_batch)
val_generator = data_generator(Validation_Captions, Validation_Encodes, wordtoix, max_length,
number_pics_per_batch)
hist_gru_model = gru_model.fit(generator, epochs=epochs, steps_per_epoch=steps, verbose=1,
validation_data=val_generator, validation_steps=val_steps)
gru_model.save('v2_gru.h5')

loss = hist_gru_model.history['loss']
val_loss = hist_gru_model.history['val_loss']
plt.plot(loss)
plt.plot(val_loss)
plt.legend(['loss', 'val_loss'])
plt.title('Loss Plot')
plt.show()


# In[ ]:


import numpy as np
from tensorflow.keras.optimizers import Adam

bilstm_model = model_bilstm(Embedding_Matrix)

bilstm_model.compile(loss='categorical_crossentropy', optimizer=Adam(learning_rate=0.001),
metrics=['accuracy'])

generator = data_generator(Train_Captions, Train_Encodes, wordtoix, max_length,
number_pics_per_batch)
val_generator = data_generator(Validation_Captions, Validation_Encodes, wordtoix, max_length,
number_pics_per_batch)
hist_bilstm_model = bilstm_model.fit(generator, epochs=epochs, steps_per_epoch=steps, verbose=1,
validation_data=val_generator, validation_steps=val_steps)
bilstm_model.save('v2_bilstm.h5')

loss = hist_bilstm_model.history['loss']
val_loss = hist_bilstm_model.history['val_loss']
plt.plot(loss)
plt.plot(val_loss)
plt.legend(['loss', 'val_loss'])
plt.title('Loss Plot')
plt.show()
```

```
# In[ ]:


from tensorflow.keras.optimizers import Adam

trainablebiGru = trainable_biGru(Embedding_Matrix)

trainablebiGru.compile(loss='categorical_crossentropy', optimizer=Adam(learning_rate=0.001),
metrics=['accuracy'])

generator = data_generator(Train_Captions, Train_Encodes, wordtoix, max_length,
number_pics_per_batch)
val_generator = data_generator(Validation_Captions, Validation_Encodes, wordtoix, max_length,
number_pics_per_batch)
hist = trainablebiGru.fit(generator, epochs=15, steps_per_epoch=steps, verbose=1,
validation_data=val_generator, validation_steps=val_steps)
trainablebiGru.save('v2_trainablebiGru.h5')

import matplotlib.pyplot as plt

loss = hist.history['loss']
val_loss = hist.history['val_loss']
plt.plot(loss)
plt.plot(val_loss)
plt.legend(['loss', 'val_loss'])
plt.show()


# ## Test Image Encoding
#
# In this section, we start encoding the test images just as we did the same for train dataset.

# In[ ]:


model_test=Feature_Extraction_Model() #ResNetv2 Model for Test Image


# In[17]:


os.chdir('C:\\Users\\ataka\\Desktop\\EEE443_Proje') #Change Accordingly

def Test_Dataset_Read(file_train):
    '''
    Funtction to read given h5 train file
    '''
```

```
    f = h5py.File(file_train, 'r') #Load Data
    print("Keys: %s" % list(f.keys()))
    test_cap = f['test_caps'][()] #Captions for training images (contains 17 indices from the vocabulary for
each image
    test_imid = f['test_imid'][()] #The indices of training images (since a single image can be captioned
several times
    test_url = f['test_url'][()] #Flickr URLs for training images

    test_ims = None  #Do not use train_ims. Otherwise, score is halved!!
    print("Size of captions: ",test_cap.shape)
    print("Size of indices of test images: ",test_imid.shape[0])
    print("Size of URL: ",test_url.shape[0])

    return test_imid, test_cap, test_url

test_imid,test_cap,test_url=Test_Dataset_Read(file_test)


# In[ ]:


Test_dir = direc + '\\Test_Ims'
model_test.summary()


# ## New Functions for Test Dataset Encoding:
#
# Below, you will find modified versions of the encoding functions. Normally, when you download the files
from URL, your file type should not be specificly stated. If it is already stated as JPEG. You might want to
use the modified functions below. Otherwise, original functions might create some problems. Since this was
a direct JPEG file, I have deleted the PNG files manually. Thus, modified functions were needed. Sorry for
the lousy coding practice.

# In[18]:


from tqdm.notebook import tqdm
import tensorflow as tf
from tensorflow.keras.applications.inception_resnet_v2 import InceptionResNetV2, preprocess_input
from tensorflow.keras.preprocessing import image
#os.environ['CUDA_VISIBLE_DEVICES'] = '-1'

def TESTImageEncoding(directory, imageIndex,model):
    #Process images so that they can be inputted in the model correctly.
    img = tf.io.read_file(directory+imageIndex)
    img = tf.image.decode_jpeg(img, channels=3)
    img = tf.image.resize(img, (299, 299))
    img = tf.expand_dims(img, axis=0)
```

```
    #print(img.shape)
    img = preprocess_input(img)
    with tf.device("/cpu:0"): #BE CAREFUL, IF YOU USE GPU, THERE WILL BE MEMORY LEAK!!
        encoding = model.predict(img) #Extracts features into single 2D vector.
    return encoding    #Should be of size (1,1536)


def Create_TEST_Features(directory,model):
    os.chdir('C:\\Users\\ataka\\Desktop\\EEE443_Proje')#Change accordingly
    os.chdir(directory) #Change to Test directory!
    print(os.getcwd())#Check
    Encoded_Features = []
    for img in tqdm(os.listdir()): #Scans the directory. img is a string!
        tmp = TESTImageEncoding(os.getcwd()+ '\\',img, model)
        Encoded_Features.append(tmp)

    Encoded_Features=np.array(Encoded_Features)

Encoded_Features=Encoded_Features.reshape(Encoded_Features.shape[0],Encoded_Features.shape[2])
    return Encoded_Features #Should output (x,1536) where x is the number of samples.


# In[ ]:


Encoded_TEST_Features=Create_TEST_Features(Test_dir,model_test) #Will take some time! Progress
bar gives a better estimate of the current state of the fucntion.


# In[19]:


import pickle #Call the function in case you have not!


# In[ ]:


os.chdir(Test_dir)
Index_TEST=[]
for img in tqdm(os.listdir()): #Scans the directory. img is a string!
        Index_TEST.append(img) #Index is used for storing the current index order of the encoded dataset!


# In[ ]:


print('Encoding finished for Test Dataset. Pickling.')
```

```
if not os.path.exists(exports_dir): #Create the directory if it does not exist in the first place.
        os.makedirs(exports_dir)
os.chdir(exports_dir)
print(os.getcwd())#Check

# save to file
print(Encoded_TEST_Features.shape)#Checking the shape
pickle.dump(Encoded_TEST_Features, open('TEST_Feature_Encodes.pkl', 'wb'))
pickle.dump(Index_TEST, open('TEST_Feature_Index.pkl', 'wb'))


os.chdir(direc) # return back to the main directory
print(os.getcwd())#Check


# ## Check whether the file is saved correctly by loading it again.

# In[18]:


print(os.getcwd())
os.chdir(exports_dir)
print(os.getcwd())


# In[21]:


TEST_Encodes = pickle.load(open(exports_dir + "\\TEST_Feature_Encodes.pkl",'rb'), encoding='utf8')
TEST_Index = pickle.load(open(exports_dir + "\\TEST_Feature_Index.pkl",'rb'), encoding='utf8')
print(TEST_Encodes.shape)
TEST_Index=np.array(TEST_Index)
print(TEST_Index.shape)


# # Test Encodes and Captions files

# In[ ]:


TEST_Feature_Dictionary= dict(zip(TEST_Index, TEST_Encodes)) #We create dictionary from the
encoded files (index,features)
import random
import math
def Test_Prep(percentage, list_to_divide):
    indices = random.sample(list(list_to_divide.keys()), len(list_to_divide)-
math.ceil((1-percentage)*len(list_to_divide)))
    Test = {}
```

```
    indices = [x[:-4] for x in indices]
    results = list(map(int, indices)) #Convert From String to Int for sorting
    results.sort(reverse=False) #Ascending Order
    for idx in results:
            Test[str(idx)]=(list_to_divide[str(idx)+'.jpg'])
    return Test
Test=Test_Prep(1.0, TEST_Feature_Dictionary)


from tqdm.notebook import tqdm #For progress bar representation. Purely decoration purposes!
#from tqdm import tqdm #For progress bar representation

#With this dataset, we will call our previous functions for captioning.
def TestDataPreparation(imid, cap, KEY_Lists,  encoding_dict , word_dict):
    ENCODES = []
    CAPTIONS = []
    for key in tqdm(list(KEY_Lists.keys())):
        key_captions = Fetch_Caption(key, imid, cap, word_dict)
        # Replicate encoding number of caption times.
        for i in range(len(key_captions)):
            ENCODES.append(encoding_dict[key+'.jpg']) #Iterate for number of captions

        # Process the captions:
        for c in key_captions:
            CAPTIONS.append(c)

    return ENCODES,CAPTIONS


# In[ ]:


Test_Encode, Test_Caption = TestDataPreparation(test_imid, test_cap, Test, TEST_Feature_Dictionary,
wordDict)


# ### Saving the Outputs:
# Now that we have processed the test data, we can proceed to save them in pickle file format so that they
can be called in the future without any compilation.
#
#
----------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------

# In[ ]:


import pickle #Call the function
```

```
os.chdir(exports_dir)
print(os.getcwd())

# save to file
pickle.dump(Test_Encode, open('Test_Encodes.pkl', 'wb'))
pickle.dump(Test_Caption, open('Test_Captions.pkl', 'wb'))


os.chdir(direc) # return back to the main directory


# In[ ]:


print('Test Encode Shape:',np.array(Test_Encode).shape)
print('Test Caption Shape:',np.array(Test_Caption).shape)


# In[236]:


import pickle #Call the function
os.chdir(exports_dir)
Test_Encodes = pickle.load(open('TEST_Feature_Encodes.pkl', 'rb'))
Test_Index = pickle.load(open('TEST_Feature_Index.pkl', 'rb'))
Test_Index=np.array([Test_Index]).T
print(Test_Index.shape)
print(Test_Encodes.shape)


# In[244]:


def Caption_GET(caps, word_dict):
    i = 0
    text_cap = []

    for item in caps:
        temp = []
        for word in item:
            key=get_key(word,word_dict)
            temp.append(key)
        text_cap.append(temp)
        # x_NULL_ strings are only ignored, not erased from the captions
    Captions_list = Caption_string(text_cap)

    return Captions_list
```

```
def Caption_GENERATE(caps, word_dict):
    temp = []
    Captions_list=[]
    for word in caps:
            key=get_key(word,word_dict)
            temp.append(key)
        # x_NULL_ strings are only ignored, not erased from the captions
    caption = ""
    for word in temp:
            if (word == 'x_NULL_') or (word == 'x_START_') or (word == 'x_END_'):
                continue #Don't count x_NULL, x_Start and x_END
            elif (word == 'x_UNK_'):
                word = 'UNKNOWN'
            caption += word + " "

    Captions_list.append(caption.strip()) #Remove spaces and append
    return Captions_list
```

BLEU Score Implementation

In[245]:

```
def GenerateBatch(Test_Index, Test_Encodes,wordtoix, model,batch_size=281):
    '''
        Create Batches and Evaluate BLEU score
    '''
    actual_container={}
    pred_container={}
    TestDataset=tf.data.Dataset.from_tensor_slices((Test_Encodes,Test_Index))
    for batch in TestDataset.batch(255):
            feats = batch[0].numpy() #Indexes
            iteration = feats.shape[0]
            seq = np.tile(np.array([1] + [0]*16), (iteration, 1))
            for i in range(16):
                with tf.device("/cpu:0"): #BE CAREFUL, IF YOU USE GPU, THERE WILL BE MEMORY LEAK!!
                    pred = model.predict([feats, seq])
                seq[:, i+1] = np.argmax(pred, axis=1)
            for i in range(iteration):
                name = batch[1].numpy()[i][0].decode()
                #!!! If your keys are '.jpg' then do this. Otherwise, comment
```

```
            #nameID=int(name[:-4])
            pred_container[name] = Caption_GENERATE(seq[i],wordtoix)

    for d in TestDataset.take(Test_Encodes.shape[0]):
        name = d[1].numpy()[0].decode()
        nameID=int(name[:-4])
        cap=Fetch_Caption(nameID, test_imid, test_cap, wordtoix)
        actual_container[name] = Caption_GET(cap,wordtoix)
    return actual_container, pred_container



# In[318]:


from keras.models import load_model
os.chdir(direc)
model = load_model("v2_bilstm.h5")
actual_container, pred_container= GenerateBatch(Test_Index, Test_Encodes,wordDict,
model,batch_size=281)


# In[319]:


import pickle #Call the function
os.chdir(exports_dir)
print(os.getcwd())

# save to file
#pickle.dump(actual_container, open('actual_container.pkl', 'wb'))
#pickle.dump(pred_container, open('pred_container_v2_trainable_bilstm.pkl', 'wb'))
# save to file
#pickle.dump(actual_container, open('actual_container.pkl', 'wb'))
#pickle.dump(pred_container, open('pred_container_v2_lstmweights.pkl', 'wb'))
#pickle.dump(pred_container, open('pred_container_v2_biGRU.pkl', 'wb'))
#pickle.dump(pred_container, open('pred_container_v2_gru.pkl', 'wb'))
#pickle.dump(pred_container, open('pred_container_v2_trainablebiGru.pkl', 'wb'))
pickle.dump(pred_container, open('pred_container_v2_bilstm.pkl', 'wb'))


# In[320]:



from nltk.translate.bleu_score import corpus_bleu

# evaluate the skill of the model
```

```
def eval_model(act, pred):
  # get descriptions
  predicted = [i[0].split() for i in pred.values()]

  # get real captions
  actual = [ [ j.split() for j in m ] for m in actual_container.values() ]

  print('BLEU-1: %f' % corpus_bleu(actual, predicted, weights=(1.0, 0, 0, 0)))
  print('BLEU-2: %f' % corpus_bleu(actual, predicted, weights=(0.5, 0.5, 0, 0)))
  print('BLEU-3: %f' % corpus_bleu(actual, predicted, weights=(0.33, 0.33, 0.33, 0)))
  print('BLEU-4: %f' % corpus_bleu(actual, predicted, weights=(0.25, 0.25, 0.25, 0.25)))

import os
import pickle
os.chdir("/content/drive/MyDrive/EE443 Project")
actual_container = pickle.load(open('actual_container.pkl', 'rb'))
predicted_container = pickle.load(open('pred_container_v2_trainable_bilstm.pkl', 'rb'))

import numpy as np
print("Bidirectional LSTM with trainable embedding")
eval_model(actual_container, predicted_container)

trainablebigru_predicted = pickle.load(open('pred_container_v2_trainablebiGru.pkl', 'rb'))
print("Bidirectional GRU with trainable embedding")
eval_model(actual_container, trainablebigru_predicted)

print("LSTM")
lstm_predicted = pickle.load(open('pred_container_v2_lstmweights.pkl', 'rb'))
eval_model(actual_container, lstm_predicted)

print("Bidirectional LSTM")

gru_predicted = pickle.load(open('pred_container_v2_gru.pkl', 'rb'))
print("GRU")
eval_model(actual_container, gru_predicted)

bigru_predicted = pickle.load(open('pred_container_v2_biGRU.pkl', 'rb'))
print("Bidirectional GRU")
eval_model(actual_container, bigru_predicted)

bilstm_predicted = pickle.load(open('pred_container_v2_bilstm.pkl', 'rb'))
print("Bidirectional LSTM")
eval_model(actual_container, bilstm_predicted)



# # Test Image Captioning
```

```
# In[ ]:


from google.colab import drive

drive.mount("/content/drive")


# In[ ]:


os.chdir("/content/drive/MyDrive/EE443 Project/Exported_Ims")
Test_Encodes = pickle.load(open('TEST_Feature_Encodes.pkl', 'rb'))
Test_Index = pickle.load(open('TEST_Feature_Index.pkl', 'rb'))


# In[49]:


os.chdir(direc)

wordtoix=pickle.load(open('wordtoix.pkl', 'rb'))
#(wordtoix)

ixtoword=pickle.load(open('ixtoword.pkl', 'rb'))
#(ixtoword)


# In[68]:


from tensorflow.keras.preprocessing.sequence import pad_sequences
import numpy as np

def generateCaption(photo, wordtoix, ixtoword, model):
    input = 'x_START_'
    photo = photo.reshape(1, 1536)
    sequence = np.zeros((1, 17))
    sequence[:,0] = 1
    for i in range(16):
        yhat = model.predict([photo,sequence], verbose = 0)
        yhat = np.argmax(yhat)
        sequence[:, i+1] = yhat
        word = ixtoword[yhat]
        input += ' ' + word
        if word == 'x_END_':
            break
    final = input.split()
```

```
    final = final[1:-1]
    final = ' '.join(final)
    return final

import pickle
import matplotlib.pyplot as plt
from keras.models import load_model

os.chdir("/content/drive/MyDrive/EE443 Project")
actual_container = pickle.load(open('actual_container.pkl', 'rb'))
lstm_predicted = pickle.load(open('pred_container_v2_lstmweights.pkl', 'rb'))
gru_predicted = pickle.load(open('pred_container_v2_gru.pkl', 'rb'))
bilstm_predicted = pickle.load(open('pred_container_v2_bilstm.pkl', 'rb'))
bigru_predicted = pickle.load(open('pred_container_v2_biGRU.pkl', 'rb'))
trainablebilstm_predicted = pickle.load(open('pred_container_v2_trainable_bilstm.pkl', 'rb'))
trainablebigru_predicted = pickle.load(open('pred_container_v2_trainablebiGru.pkl', 'rb'))

"""# Predictions"""

import pickle
import matplotlib.pyplot as plt
from keras.models import load_model
os.chdir("/content/drive/MyDrive/EE443 Project")
test_img_dir = "/content/drive/MyDrive/EE443 Project/Test_Ims"

def caption_testImg(id):
  img_dir = Test_Index[id].split(".")[0]
  im = Image.open(test_img_dir+"/"+img_dir, mode = 'r')
  plt.figure(figsize=(8,8))
  plt.imshow(im)
  plt.axis('off')

  idx = Test_Index[id]
  print("Prediction:")
  print("LSTM:", lstm_predicted[idx][0])
  print("GRU:", gru_predicted[idx][0])
  print("BLSTM:", bilstm_predicted[idx][0])
  print("BGRU:", bigru_predicted[idx][0])
  print("Trainable BLSTM:", trainablebilstm_predicted[idx][0])
  print("Trainable BGRU:", trainablebigru_predicted[idx][0])
  print("Actual:")
  actuals = actual_container[idx]
  for caps in actuals:
    print(caps)

caption_testImg(139)

caption_testImg(8)
```

caption_testImg(159)

caption_testImg(179)

caption_testImg(880)

caption_testImg(339)

caption_testImg(20000)

caption_testImg(69)

caption_testImg(259)

caption_testImg(1071)

caption_testImg(1075)

caption_testImg(3155)

caption_testImg(21420)

caption_testImg(859)

caption_testImg(559)

caption_testImg(447)

caption_testImg(875)

caption_testImg(3069)

caption_testImg(731)