Convolutional Language Model for Image Captioning: Deep Learning for Vision & Language Translation Models

Can Kocagil
Bilkent University EEE student
21602218
Ankara, Turkey
can.kocagil@ug.bilkent.edu.tr

Eksal Uras Kargı
Bilkent University EEE student
21600848
Ankara, Turkey
uras.kargi@ug.bilkent.edu.tr

Arman Vural Budunoğlu
Bilkent University EEE student
21602635
Ankara, Turkey
vural.budunoglu@ug.bilkent.edu.tr

***Abstract*—**This paper is mainly focused on image captioning task using the state of the art techniques in the context of deep learning. Image captioning is the process of generating textual description of an image by using both natural language processing and computer vision applications. Network consists of Convolutional Neural Network (CNN) to encode images into latent space representations followed by Recurrent Neural Network (RNN) to decode feature and word representations and build language models. Specifically, Long-Short Term Memory(LSTM) and Gated Recurrent Unit (GRU) are used as a RNN model with attention mechanism and teacher forcer algorithm. To realize that, transfer learning applications such as AlexNet, VGG-Net, ResNet, DenseNet and SquezeeNet are used as a convolutional encoder and Global Vector for Word Representation(GloVe) is used for word embedding. Flickr dataset is used for both training and testing as proposed. Various data augmentation techniques are implemented to boost the model performance. The model is compiled by Adam optimizer with scheduled learning rates. Masked Cross Entropy loss is used for criterion for the models. Finally, beam and greedy search algorithms are implemented to get the best image-to-caption translation.

Keywords— Transfer learning, Language model, Computer Vision, Natural Language Processing CNN, LSTM, GRU, Data Augmentation, Beam search, Parallel Distributed Processing, Translation model

I. Introduction

With the advent of new technologies related to artificial intelligence, image captioning has become one of the most attractive field for researchers. Image caption, automatically generating natural language descriptions according to the content observed in an image, is an important part of scene understanding, which combines the knowledge of computer vision and natural language processing [1]. The applications of image captioning are extensive such as connecting human-computer interaction and also it may help the visually impaired people "see" the world in the future [1]. In earlier stages of image captioning, statistical language models are used to come up with a solution for generating captions for images. Li et al. propose a n-gram method based on network scale, collecting candidate phrases and merging them to form sentences describing images from zero [2]. Yang et al. propose a language model trained from the English Gigaword corpus to obtain the estimation of motion in the image and the probability of collocated nouns, scenes, and prepositions and use these estimates as parameters of the hidden Markov model [2]. According to the literature review, we come up with essential stages of state of art image captioning in the context of deep learning. Here is the overview of the image captioning pipeline and widely used techniques:

A. Feature Extraction

Image captioning task starts with feature extraction from the images to reduce dimensionality of the possibly 3 (RGB) channel high dimensional data into latent space representation. In the literature, there are already pre-trained models that are trained by using the dataset called 'ImageNet' that consist of more than 1.2 million natural images, such as AlexNet, VGG-Net, ResNet, GoogleNet, DenseNet, SquezeeNet and so on. We implemented all mentioned models except for GoogleNet for our case that we will see later in this paper.

B. Language Model

As a second stage of image captioning, captions and latent space feature vectors are given to the language model to generate captions. To realize this, there are various models that are widely used in the literature such as LSTM's, bi-directional LSTM's, RNN's, CNN's, GRU's and TPGN. We have used both LSTM and GRU's recurrent networks for our implementations that we will discuss in the methods part.

C. Techniques

To generate image captioning model, the following image-to-sequence techniques are used in the literature:

- Encoder-Decoder
- Attention Mechanism
- Novel Objects
- Semantics

Note that there are various algorithms for implementing image-to-sequence networks, these are widely used techniques. In this project, we have used both encoder-to-decoder model and teacher forcer with attention mechanism.

D. Datasets

MSCOCO and Flickr datasets are widely used in image captioning tasks. In our case, Flickr web service dataset is used for both training/validation and testing that has more than 80 000 natural images. The URL's of the images are given to us to be used. But, since approximately 10% of the URLs are broken, we have nearly 70 000 to 73 000 images for training and validation. The whole set is split into 15% of validation and 85% of training set. Furthermore, in this dataset, each image is paired with 4 to 5 associated captions that describes the content of that particular image. The corpus size of this dataset was a 1004 including <x_START_>, <x_END_>, <x_NULL_> and <x_UNK_> phrases that represents start signal, stop signal, pad and unknown word, respectively. Finally, note that there were some irregular words such as 'xFor' and 'xWhile', they converted to regular format as a part of preprocessing of captions.

E. Performance Metrics

To evaluate the image-to-sequence models performance, the following evaluation metrics are used:

- BLEU-1
- BLEU-2
- BLEU-3
- BLEU-4
- CIDEr
- METEOR

For our implementations, we used BLUE-1, BLUE-2, BLUE-3, BLUE-4 and METEOR metrics to evaluate the language model's outputs at the end of the training. Then, various data augmentation techniques are implemented to boost the model's performance. For feature extraction, we have used CNN encoder structure with pre-trained models, ResNet152, AlexNet, VGG19-Net, DenseNet, SquezeeNet, to experiment with the performance on feature extractions. After such experiments, ResNet152 performed slightly better among others that we will see in the following pages. (ResNet stands for Deep Residual Learning for Image Recognition.) Our encoder CNN models are pre-trained in the ImageNet dataset and consists of multiple convolutional layers. For ResNet152, residual neural networks are utilizing skip connections, or shortcuts to jump over some layers to reduce the adaptivity of layers to a given image data. Typical *ResNet* models are implemented with double- or triple- layer skips that contain nonlinearities (ReLU) and batch normalization in between [3]. One motivation for skipping over layers is to avoid the problem of vanishing gradients, by reusing activations from a previous layer until the adjacent layer learns its weights [3]. Hence, the following figure represents the skipping connections in a visual way.

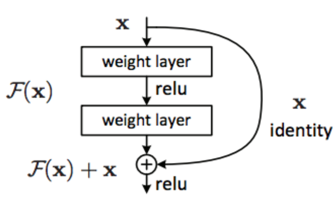


Fig. 1. Residual building block

Our residual networks consist of 152 layers. Since it is hard to visualize, the following architecture represents sample residual block structure.

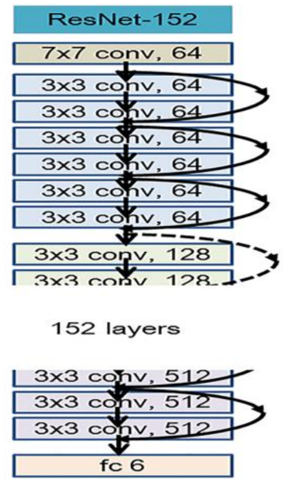


Fig. 2. 152-layer residual building block

Note that all ResNet152, AlexNet, VGG19-Net, DenseNet, SqueezeNet are used for end-to-end image captioning process, but it is worth to mentioned ResNet152 as our primary encoder. Then, for the language model, LSTM and GRU recurrent neural networks are utilized by attention mechanism and teacher forcing technique. As a first step, word embedding layer is used to word representation for sentence length captions. To realize this, we have used Global Vector for Word Representation (GloVe) that is an unsupervised learning algorithm for obtaining vector representations for words [4]. 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 [4]. We have used the GloVe model that is trained with 6 billion tokens with 400 000 corpus size. To use GloVe, necessary processing is done to retrieve corresponding weights for our word embedding. Also, it is worth to mention that more advanced GloVe vector that is trained with 840 billion tokes are also tried, but we face with CUDA memory issues related to processing word vector. After that, to minimize the cost of the model masked cross entropy loss is used and both encoder and decoder model are optimized by Adam optimizer. Hopefully, the end of the training phase, we expect our vision & language model to generate human-level captions for natural images.

II. Methods

A. Train/Validation/Testing Split

As proposed, we split the given training dataset into 85% training and 15% validation to keep track of the history of the model while training and also we applied cross-validation techniques to stop the training when the model starts overfitting. Therefore, we have 340 114 unique training images and 60 021 unique validation images at the end of dataset splitting. Furthermore, a separate testing dataset is given us to measure the performance of a model with a wide variety of natural images.

B. Preprocessing

Since images have different sizes, we firstly convert the images with acceptable sizes, in our case 224x224, to meet the vision models requirements. All mentioned CNN models are accepting the input with the (N x C x H x W) convention that is PyTorch RGB image batch convention. In our case, all CNN models accept the C,H,W = (3,224,224) format. Lastly, all images are normalized to get zero mean and unit standard deviation distributions to accelerate training. We applied normalization for each color channel separately by

- Mean μ = [0.485, 0.456, 0.406]
- Standard deviation σ = [0.229, 0.224, 0.225]

with the formula:

$$Z = \frac{x - \mu}{\sigma} \quad (1)$$

Where Z is the standard value, x is observed value. These are ImageNet's RGB channel means and standard deviations which may represent the generalized mean and standard deviation of natural images.

C. Data Augmentations

Data augmentations in data analysis are techniques used to increase the amount of data by adding slightly modified copies of already existing data or newly created synthetic data from existing data [5]. It acts as a regularizer and helps reduce overfitting when training a deep learning model [5]. To realize this, we applied following data augmentation techniques:

- **Random Horizontal Flip**
  We flip horizontally the image data with the probability of 0.5.
- **Random Vertical Flip**
  We flip vertically the image data with the probability of 0.5.
- **Random Cropping**
  We randomly crop the images with 224x224 dimensions randomly.
- **Random Resized Cropping**
  We resize the images 256x256 then crop 224x224 parts of it randomly.
- **Center Crop**
  We resize the images 256x256 then crop 224x224 parts of it at the center.

D. Transfer Learning: Encoder CNN

As discussed, we have used the ResNet152, AlexNet, VGG19-Net, DenseNet AND SqueezeeNet separately to convert high dimensional images into latent space representation. Hence, we applied feature extraction, i.e., we freeze the learnable/trainable parameters of the layers of the CNN model while training, i.e., we do not update the network parameters of the encoder model. Another possible approach for transfer learning is to fine-tune the network but we prefer to only extract features due to computational concerns. Then, we pass our feature vector into the embedding layer to get an embedded image feature vector. This embedding layer converts latent space representation into embedding space with a learnable way. Now, our images are ready to feed the language model. The following figure visualizes the explained architecture.

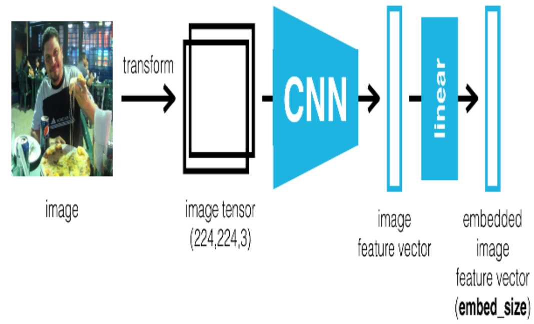


Fig. 3. Encoder CNN with embedding

Therefore, embedded image feature vectors are feed as an initial input of the decoder network with the <x_START_> caption in the teacher forcer algorithm.

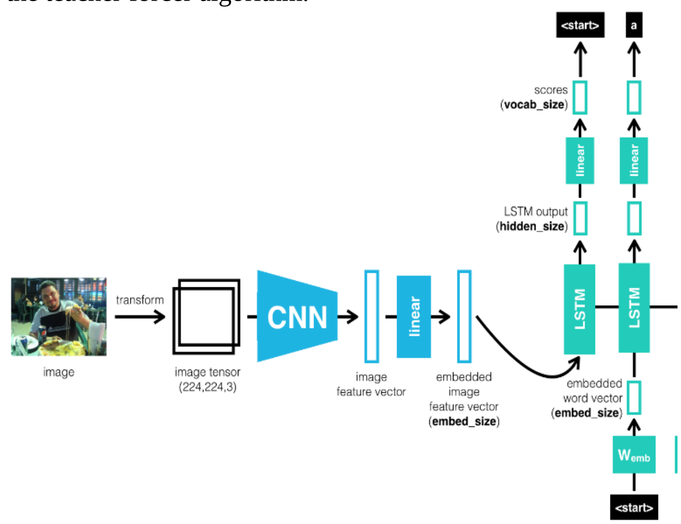


Fig. 4. Encoder CNN to language model

E. Transfer Learning: Word Embedding (GloVe)

As discussed, we bought a pre-trained GloVe word vector for our word embedding layer. This accelerated our training phase since these vectors are already trained with 6 billion English tokens. Hence, we successfully convert captions into advanced word representation so that they are ready to be input to the decoder model. Here is the word embedding process visualization.

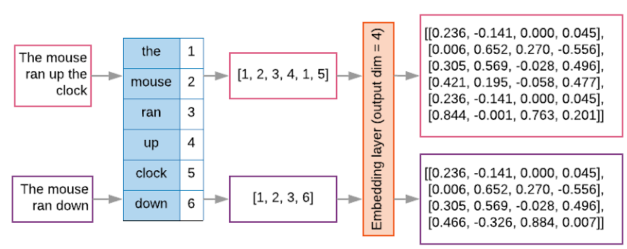


Fig. 5. Word embedding explanation

This layer converts our sentence length sequence captions into embedded word feature vectors so that the captions are ready to pass to the language model.

F. Decoder with Teacher Forcer

A lot of Recurrent Neural Networks in Natural Language Processing (e.g. in image captioning, machine translation) use Teacher Forcing in the training process [6]. Teacher forcing is a method for quickly and efficiently training recurrent neural network models that use the ground truth from a prior time step [7]. Let's give an example of a teacher forcer algorithm, let ground truth caption for arbitrary image is "Two people reading a book". Our model makes a mistake in predicting the 2^{nd} word and we have "Two" and "birds" for the $1^{st}$and 2^{nd}prediction respectively [8]

- Without *Teacher Forcing*, we would feed "birds" back to our RNN to predict the 3rd word. Let's say the 3rd prediction is "flying". Even though it makes sense for our model to predict "flying" given the input is "birds", it is different from the ground truth. [8]
- On the other hand, if we use *Teacher Forcing*, we would feed "people" to our RNN for the 3^{rd} prediction, after computing and recording the loss for the 2^{nd} prediction [8]

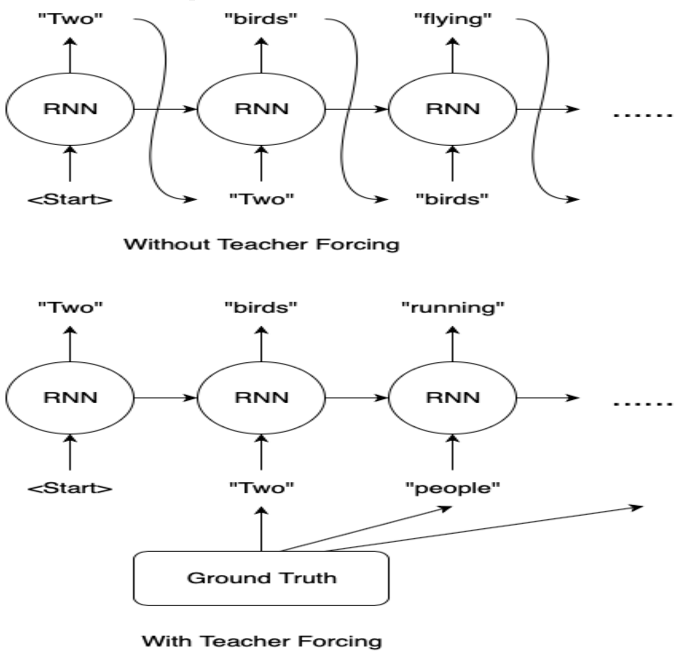


Fig. 6. W/o teacher forcer

Then, let's discuss the pros & cons of teacher forcer.

- Pros:
  Training with *Teacher Forcing* converges faster. At the early stages of training, the predictions of the model are very bad. If we do not use *Teacher Forcing*, the hidden states of the model will be updated by a sequence of wrong predictions, errors will accumulate, and it is difficult for the model to learn from that. [10]
- Cons:
  During inference, since there is usually no ground truth available, the RNN model will need to feed its own previous prediction back to itself for the next prediction. Therefore, there is a discrepancy between training and inference, and this might lead to poor model performance and instability. This is known as *Exposure Bias* in literature [11].

Therefore, we have used teacher forcer algorithms in our language models.

G. Decoder with Teacher Forcer and Attention Mechanism

As a second stage of model, we implemented a decoder with attention mechanism by LSTM and GRU networks. The job of the RNN is to decode the feature and word vector and turn it into a sequence of words [6]. In the decoder, we first pass the embedded feature vectors to the decoder at time t = 0 as a part of the teacher forcer algorithm. Then, we pass the captions word by word using the actual teacher forcer algorithm. Therefore, we implemented a language model to map the latent space vectors to the word space. [7]. The key idea here is to feed the latent space vector that represents the image as the input to the recurrent unit cell at time t=0 [7] Beginning at time t=1 we can start feeding our embedded target sentence into the recurrent cell as a part of the teacher forcer algorithm [8]. Then, the output of an LSTM cell is the hidden state vector. Hence, we will need some kind of mapping from the hidden state space to the vocabulary (dictionary) space [9]. We can achieve this by using a fully connected layer between the hidden state space and the vocabulary space [9]. The following architecture describes the LSTM mechanism:

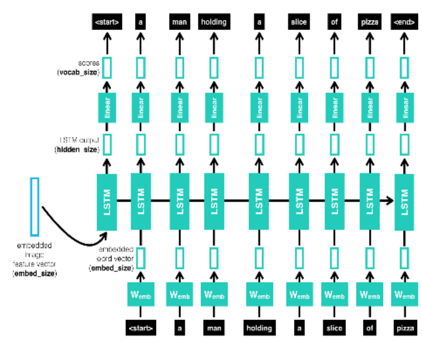


Fig. 7. Decoder mechanism

Here is the figure of overall image-to-sequence model

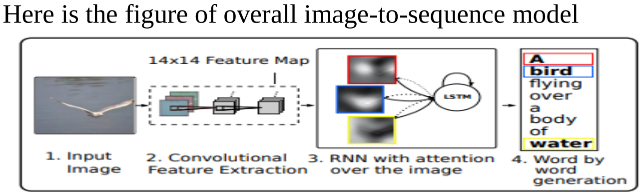


Fig. 8. Visual attention in image captioning

Then, in a setting with Attention, we want the decoder to be able to look at different parts of the image at different points in the sequence [17]. Instead of the simple average, we use the *weighted* average across all pixels, with the weights of the important pixels being greater [17]. This weighted representation of the image can be concatenated with the previously generated word at each step to generate the next word [17]. The attention mechanism computes these weights to estimate the important parts of images. We have used the stochastic soft attention mechanism, where the weights of the pixels add up to 1 as proposed in the Show, Attend and Tell paper [12]. If there are P pixels in our encoded image, then at each time step t

$$\sum_p^T \alpha_{p,t} = 1 \qquad (2)$$

One could interpret this entire process as computing the probability that a pixel is *the* place to look to generate the next word [17].

Fig. 9. Attention mechanism over time

The data flow starts with the convolutional vision model to create latent space representation of the images then followed by the recurrent model to create initial hidden and cell states for the LSTM, and hidden state for GRU decoder. At each time step of the decoding, the latent space representation and previously computed hidden states of the recurrent unit is used to generate weights for the image pixels as a part of the attention mechanism. Then, ground truth captions and weighted average of the encodings are fed to the decoder language model to generate the next caption as a combination of teacher forcer and attention algorithms. The following figure represents the information flow in the soft stochastic attention with teacher forcer.

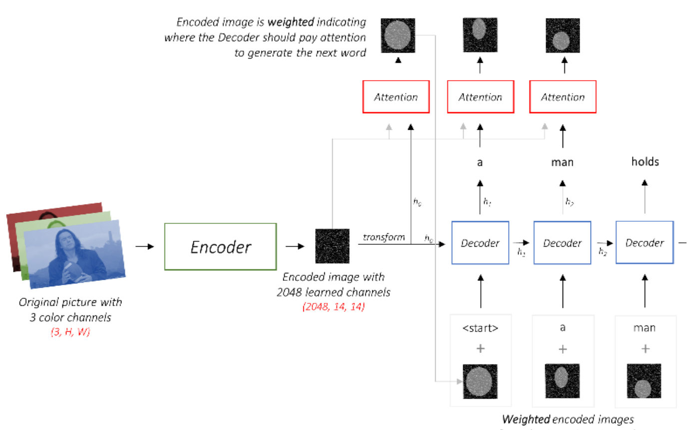


Fig. 10. Encoder to attention pipeline

H. Norm Gradient Clipping

Gradient clipping is a technique to prevent exploding gradients in very deep networks, usually in recurrent neural networks [17]. There are many ways to compute gradient clipping, but a common one is to rescale gradients so that their norm is at most a particular value [17]. With gradient clipping, pre-determined gradient threshold be introduced, and then gradients norms that exceed this threshold are scaled down to match the norm [17]. This prevents any gradient to have norm greater than the threshold and thus the gradients are clipped [17].

$$\|g\| \geq \delta_{thres} \xrightarrow{yields} g := \delta_{thres} * \frac{g}{\|g\|} \qquad (3)$$

Where g is the gradient to be clipped, δ_{thres} is the threshold value that is a hyperparamater and $\|g\|$ is the norm of g.

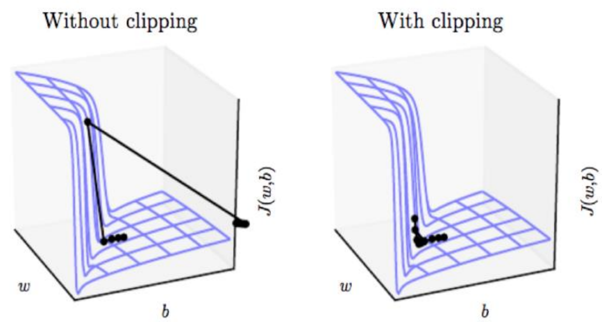


Fig. 11. Norm gradient clipping visualization

Therefore, we implemented norm gradient clipping to keep the gradients within certain range that is characterized by the δ_{thres} value and determined as a 10 after such experiments.

I. GPU acceleration and Parallel-Distributed Processing

Then, since we are using GPU acceleration and distributed computing, we need to convert our data types into tensors that have the ability of running image processing tasks in GPU. Specifically, NVIDIA Tesla K80 and GeForce RTX 2080 TI are used as a GPU to accelerate the training. Then, we have used Distributed Data Parallel (DDP) that implements data parallelism at the module level which can run across multiple GPU's. [10] Applications using DDP spawn multiple processes and create a single DDP instance per process [10]. Hence, we utilized the CUDA data parallelism to accelerate training. To realize this, deep learning framework PyTorch's DataParellel package is utilized.

J. Masked Cross Entropy

Masked cross entropy is actually a categorical cross entropy with applying masks to some inputs determined by sequence lengths. The reason behind this is that we have padded sequences, i.e., every caption in the dataset has different lengths so to construct a vector of captions, we need to pad the gaps by adding <Pad> to the end of the captions. The visual representation is:

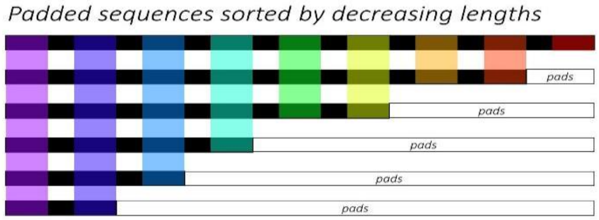


Fig. 12. Padding the captions

Therefore, to not compute the loss and gradients for padded regions, we implement masked cross entropy that takes predicted captions, actual captions and sentence lengths and apply cross entropy with non-padded regions. This accelerates the training procedures. Here is the formula for our criterion for the model.

Cross-Entropy Loss

$$Ł(O_{output_i}, Y_i) = \sum_{i=1}^{N} Y_i * \log(O_{output_i}) \quad (3)$$

Where O_{output_i} is the decoder models predictions and Y_i is the actual captions.

K. Adam Optimizer

Both encoder and decoder model is optimized with Adam optimizer with the following mathematical expressions:

Adam Optimizer

$$\delta_{M_i} = \beta_1 * \delta_{M_i} + (1 - \beta_1) * \nabla\theta_i ,$$

$$\delta_{V_i} = \beta_2 * \delta_{V_i} + (1 - \beta_2) * \nabla^2\theta_i$$

$$\widetilde{\delta_{M_i}} = \frac{\delta_{M_i}}{1-\beta_1} , \quad \widetilde{\delta_{V_i}} = \frac{\delta_{V_i}}{1-\beta_2} ,$$

$$\theta_i := \theta_{i-1} - \frac{\eta}{\sqrt{\widetilde{\delta_{V_i}} + \varepsilon}} * \widetilde{\delta_{M_i}} \quad (4)$$

Where δ_{M_i} *and* δ_{V_i} are the accumulated sum of gradients in first and second moment respectively and θ is the parameters to be updated. Also note that basic stochastic gradient descent based learning rules, RMSprop and AdaGrad are also utilized but the most compromising performance caught on Adam optimizer so we keep going with the Adam optimizer.

L. Adaptive Learning Rate Scheduler

We applied a dynamic learning rate scheduler based on the validation cross entropy and the argmax search accuracy with the help of the PyTorch optimizer package. The algorithm is to reduce the learning rate when

1) *Cross Entropy loss is not decreasing*
2) *Accuracy of argmax predictions is not increasing*

Hence, we reduced the learning rate when a metric has stopped improving. It enables a dynamic learning rate scheduler and may increase the performance of the model before early stopping. Furthermore, we also applied a straight forward learning rate scheduler based on the batch improvements. Hence, three different learning late schedulers are used to boost the performance of the model. Note that as a further implementation, the adaptive learning rate scheduler can be performed by using a BLEU scores that may give additional adaptivity within the network.

M. Early Stopping with Cross Entropy

Based on the cross entropy loss, the history of the model is tracked in batch and epoch wise to avoid overfitting. If the gap between the training and validation losses starts increasing, we stop training. Note that since we applied adaptive learning rate scheduling, we let the model adapt to the new learning rate, if the model failed to improve itself, we stopped training. Luckily, our model performs similarly in training and testing so that early stopping is not used for the entire process of training for all models.

N. Beam Search

The Show and Tell paper [12] presents Beam Search as the final step to generate a sentence with the highest likelihood of occurrence given the input image. [12] The algorithm is a best-first search algorithm which iteratively considers the set of the k best sentences up to time t as candidates to generate sentences of size t + 1, and keep only the resulting best k of them, because this better approximates the probability of getting the global maximum as mentioned in the paper [12].

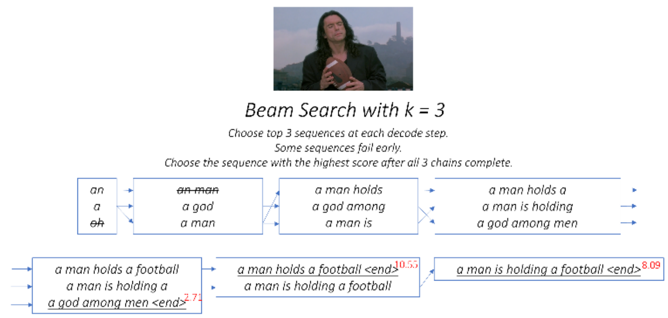


Fig. 13. Visual explanation of Beam Search [13]

We have implemented beam search with the beam size from 1 to 9 to see the changes with the parameter of beam size. The results are represented and discussed in the following part.

O. A Framework to Accelerate the Deep Encoder-Decoder

Let's recall and gather what we did to accelerate the model's performance. We mainly used the multiprocessing and multithreading concept in general. For loading and transforming the data, we have used the multithreading concept, with 4 threads. Also, we convert the loading and transforming application into GPU format to accelerate. Then, we have used the generator concept that is implemented internally in the PyTorch that will boost the RAM efficiency and allow such operations in the training phase. Most importantly, we utilized the distributed parallel computing applications given by the PyTorch on the GeForce RTX 2080 TI and Tesla K80. Then, for neural network model applications, we freeze the convolutional models and GloVe word embedding models learnable parameters, i.e., we did not calculate gradients for these layers that will accelerate the training phase obviously. For the language model, we implemented the teacher forcer algorithm that also improved the time efficiency for generating the captions. Furthermore, various hyperparamaters are tuned with the concern of model performance and time efficiency.

Finally, the optimization time is measured with different mentioned optimizers to choose the best optimizer with the criteria of batch improvements and update time.

P. Evaluation Metrics for Translation Models

As mentioned, we also evaluate the model using BLEU scores that is the Bilingual Evaluation Understudy, is a score for comparing a candidate translation of text to one or more reference translations [15]. It evaluates how good a model translates from one language to another [15]. It assigns a score for machine translation based on the unigrams, bigrams or trigrams present in the generated output and comparing it with the ground truth [15]. However, it doesn't consider the meaning, sentence structure, and morphologically rich part of the languages. Moreover, we used the metric METEOR (Metric for Evaluation of Translation with Explicit Ordering) that is based on the harmonic mean of unigram precision and recall, with recall weighted higher than precision. [14] Furthermore, the argmax accuracy is also calculated to see batch improvements while training the model. Note that this is not an appropriate metric for image-to-sequence translation models and it is implemented to see just for the sake of seeing the learning curve.

III. Results

We tried lots of algorithms and models for both encoder and decoder with different hyperparamaters. We have implemented ResNet152, AlexNet, VGG19-Net, DenseNet and SquzeeNet for convolutional encoders. Then, we built a trainable embedding layer with embedding size 256 (the value of embedding size was a hyperparamater, it is found after such experiments with embedding size). After that, we convert the captions into word embedding layers with the embedding size 256. Then, both embedded feature vectors and word embedding are passed into the language model with teacher forcer and attention mechanisms to generate captions. The mentioned algorithms are our final findings regarding our research in the context of image caption generation. We tried different batch sizes such as 32,64,128 and 256 as batch tuning. Our final model is trained with batch size 64. When the batch size is 256, even both models are performed slightly better, we meet a memory issue related the CUDA memory so we prefer 64 as an ideal batch size. Even if the dynamic learning scheduler is applied, we start with lower learning rates for both encoder and decoder model. For encoder 4x 10^{-2} is selected as an initial learning rate, and 1x 10^{-2} is selected for initial decoders learning rate. Then, let's talk about the model performance. The model is trained on both NVIDIA Tesla K80 GPU and GeForce RTX 2080 TI with 2-3 epochs. Each epoch takes 1-1.30 GPU hour NVIDIA Tesla K80 and 45-60 minutes in GeForce RTX 2080 TI. Hence, overall training time for each model approximately takes 2-4 hours depending on the performance. Our training time per epoch is super-fast since we have implemented various runtime efficient algorithms as mentioned. (see A Framework to Accelerate the Deep Encoder-Decoder part) Furthermore, the cross entropy loss starts at approximately 7 for all models and at the end of the training, we have reached 1.42-1.05 for our models that are presented in the below. We also calculate greedy accuracy, even accuracy is not the rights metric for our image-to-sequence translation model, just to see learning curves. The argmax accuracy started at 1% and reached around 65-78% at the end of the trainings. The BLUE and METEOR scores are calculated for all models and given below. According to the Table I (in the following page), the by far winner among whole vision & language models is ResNet152 for encoder and GRU for decoder with the attention mechanism by the criteria of BLEU score and METEOR that represents linguistic performance of generated captions so that they are best evaluation metric for image-to-sequence translation models. We see that at the end of the 3 epoch, the model reached 1.05 entropy loss with 78.2 argmax accuracy (even the accuracy is not the right metric for translation models). More importantly, we have reached a score of 67.5 for BLUE-1 and 22.59 for METEOR that are nearly perfect for translation models. With the slight difference, 2^{nd} winner is the model consisting of ResNet152 and LSTM. Hence, we may conclude that ResNet152 performed well for our feature extraction task. However, when we compare the all vision & language models, we see that there is not a big difference for our implementations with the Flickr dataset. Therefore, we can say that all vision & language models are performed very well in the testing case. Then, let's start with generated captions for testing images. But before that, it is worth discussing the structure of language to evaluate the generated captions linguistically. Generally, the language is consisting of several components such as phonetics, phonology, morphology, syntax, semantics and pragmatics. Here is the visualization of the language structure.

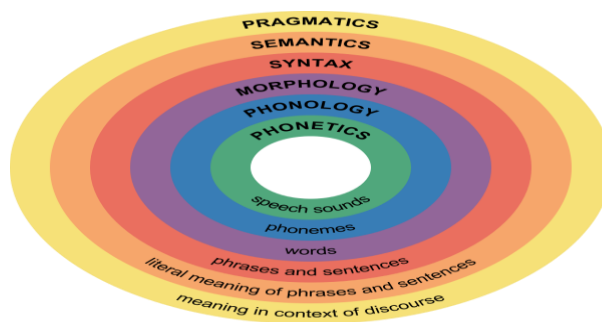


Fig. 14. Language structure

Here are the randomly selected samples from testing test and generated captions (in the below of the images) for that.

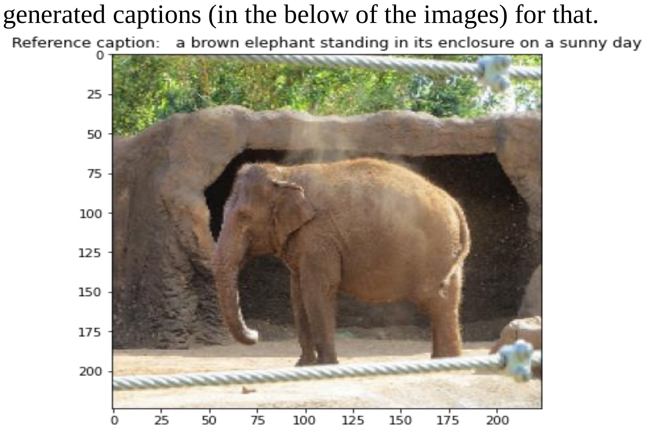


Fig. 15. Reference and generated caption by ResNet152-to-GRU for randomly selected image from test data

Encoder	*Decoder*	*Cross Entropy*	*Argmax Accuracy*	*METEOR*	*BLEU-1*	*BLEU-2*	*BLEU-3*	*BLEU-4*
ResNet152	LSTM	1.12	77.8	22.12	65	45.2	29.4	19.99
AlexNet	LSTM	1.21	75	21.68	61.1	43.32	—	—
VGG19-Net	LSTM	1.15	75.5	19.75	63.4	44.04	28.77	18.76
DenseNet	LSTM	1.32	69.8	16.3	59.57	40.39	—	—
ResNet152	GRU	1.05	78	22.59	67.5	47.8	29.7	20.1
VGG19-Net	GRU	1.42	65.3	17.55	58.85	—	25.55	—
SquezeeNet	GRU	1.38	66.7	18.98	59.02	—	25.67	17.78

TABLE I. CROSS ENTROPY, ARGMAX ACCURACY, BLEU-1,2,3,4 AND METEOR METRICS COMPARED AMONG ALL MODELS

It is interesting that our model catches that the elephant is a baby and the weather is sunny. It is good for us to see a perfect caption for that. Furthermore, it is also interesting that our model catches a pen in the figure. Hence, even if the language criteria are satisfied, it is good to see that the model can catch the details. Also, it can be inferenced that BLEU-1 score more than 60 percentage corresponds high quality captions often better than humans.

Fig. 16. Reference and generated by ResNet152-to-GRU caption for randomly selected image from test data

Fig. 17. Reference and generated caption by AlexNet-to-GRU for randomly selected image from test data

Generated caption 'a woman who is riding a horse' is a perfect caption for that image so that our model actually meets the language criteria perfectly since it has a proper syntax, understandable semantic, and proper context meaning. Let's continue examining examples.

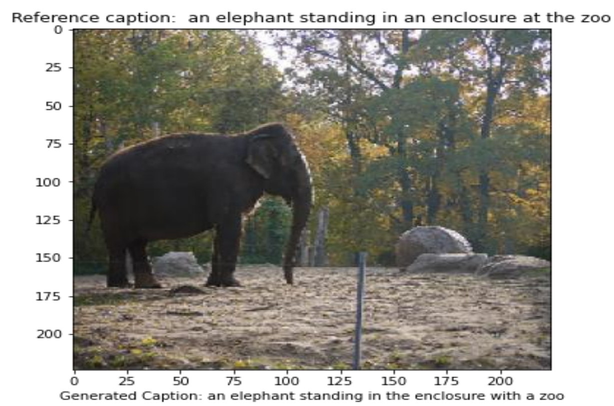


Fig. 18. Reference and generated by ResNet152-to-LSTM caption for randomly selected image from test data

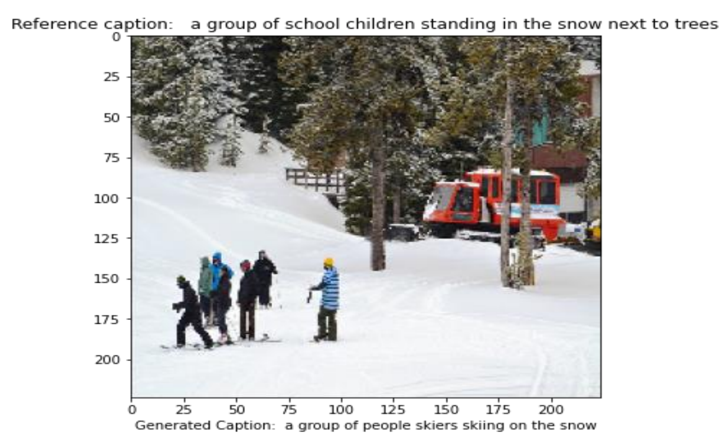


Fig. 19. Reference and generated by ResNet152-to-LSTM caption for randomly selected image from test data

We see that generated caption quality are human-level or more, i.e., they meet all corresponding language structure components. It has powerful semantics, syntax and pragmatics. Let's see another captions that are created by beam search. (see Appendix I for all captions)

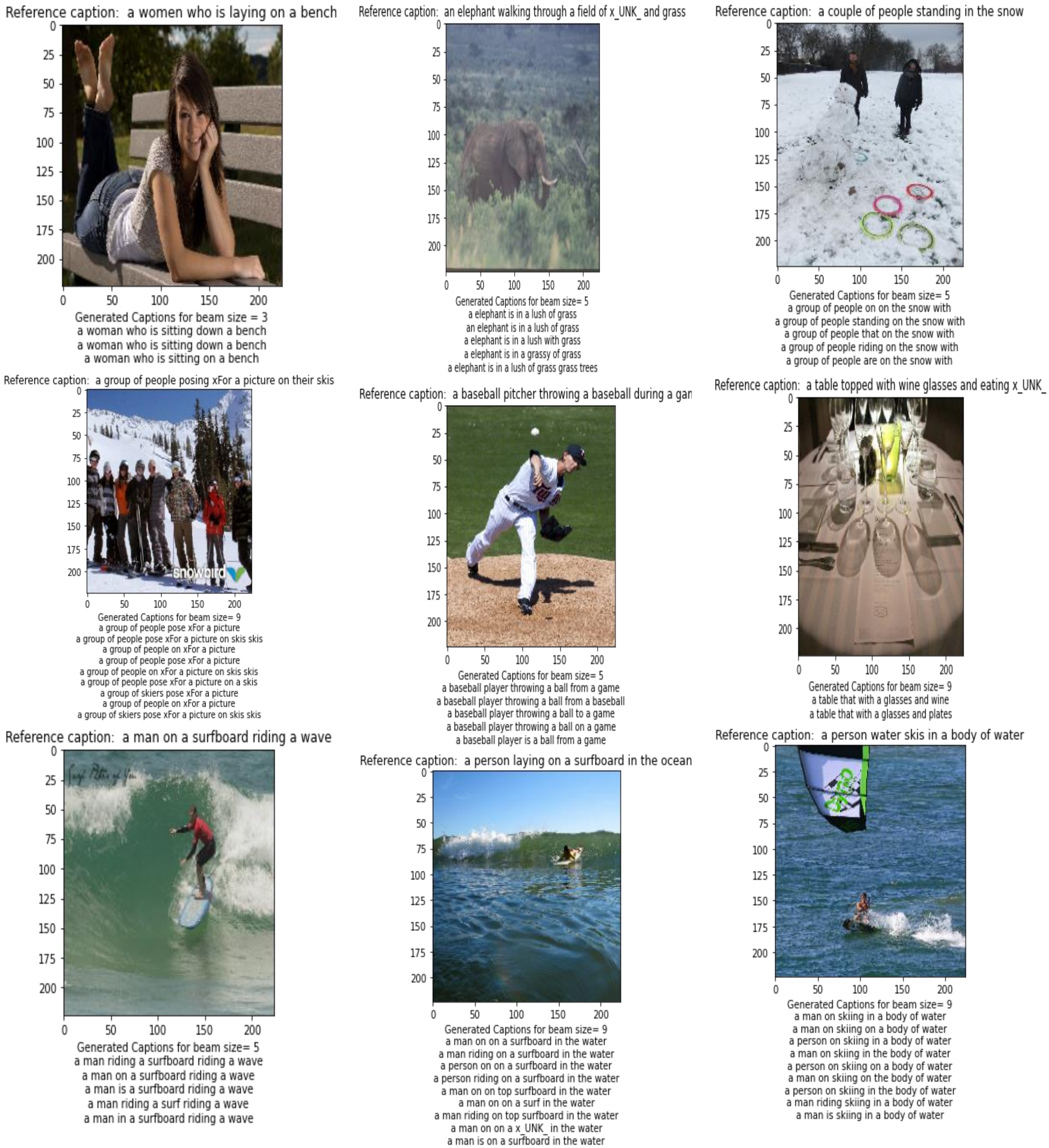


Fig. 20. References and generated captions by our vision & language models by seam search

We see that the captions generated by our vision & language models are very powerful with additional beam search algorithms. The captions satisfy the all language criteria, even in some of the cases, we have more meaningful captions than our reference captions. Here are more captions generated by ResNet152-to-GRU model with beam search and a beam width is 7.

Fig. 21. Reference and generated caption by ResNet152-to-GRU for randomly selected image from test data

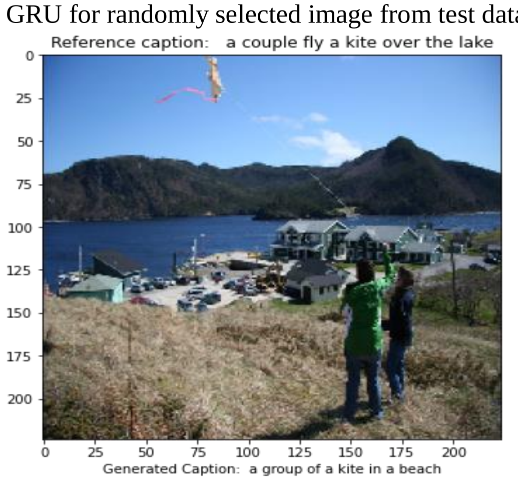


Fig. 22. Reference and generated caption by ResNet152-to-GRU for randomly selected image from test data

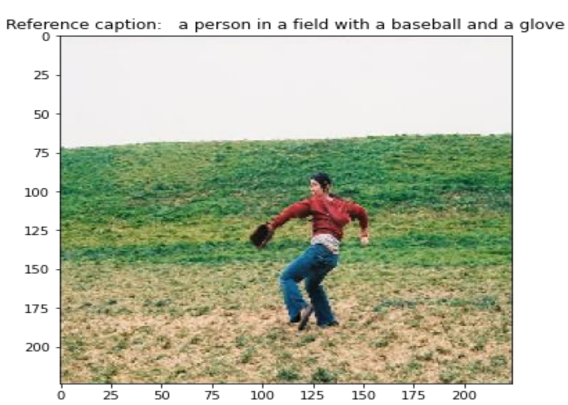


Fig. 23. Reference and generated caption by ResNet152-to-GRU for randomly selected image from test data

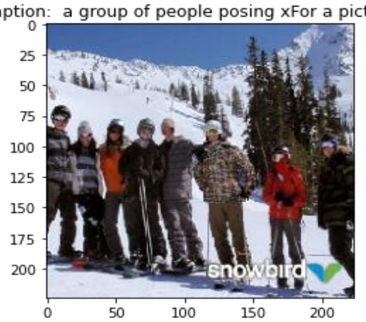
IV. Discussion

Automatically image captioning is far from mature and there are a lot of ongoing research projects aiming for more accurate image feature extraction and semantically better sentence generation [19]. In our case, we successfully generated captions which satisfy the rule of the language criteria such as syntax, semantics, pragmatics and morphological meanings. As a further improvement, there are various ways to increase vision & language models performance in the context of the image captioning. As mentioned, we did not perform fine tuning the pre-trained parameters for encoder models and GloVe word vector due to computational concerns, one can get extra performance by fine tuning these layers. Furthermore, since we have limited CUDA memory, we cannot train the models with bigger batch sizes that was a drawback for our implementations. Even, we have used multiple GPU's for training models, since we implemented 7 different models that are a combination of encoder and decoders, we could not train the models more than 2-3 epochs again due to computational restrictions and they all proved their performance within 1-3 epochs. Moreover, scheduled learning rate can be implemented via BLEU scores which gives more accurate adaptivity of learning rates to the models. Also, one can try the early stopping with BLEU score, of course in the case of longer trainings. Additionally, training can be boosted by beam search, it is not a common technique used in the literature, but one can try to train model by calculating the cross entropy based on the predictions generated by the beam search to see differences. Also, more hyperparamater tuning can be implemented with more RAM and GPU hardware components, one can try different learning rates, number of layers, number of neurons (hidden and embedding size), dropout rates, batch normalizations etc. As a further, bigger datasets can be used to reach more realistic captions, one can try merging FLİCKR and MSCOCO datasets. In the context of deep learning, different vision & language model architectures can be tried as a further tuning. Moreover, there are more than one attention mechanisms such as soft attention (what we use), hard attention, log bilinear attention, stochastically doubled attention and so on. Finally, we successfully implemented the image-to-sequence translation model using different convolutional encoders followed by different recurrent decoders with the teacher forcer and attention mechanisms with the help of distributed parallel computing in multiple GPUs.

APPENDIX

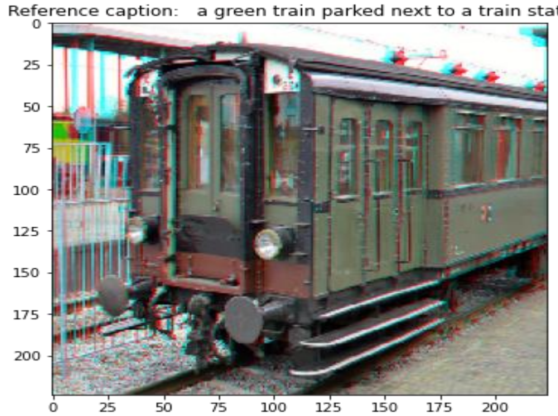
Appendix I – Generated Captions

Reference caption: a group of people posing xFor a picture on their skis



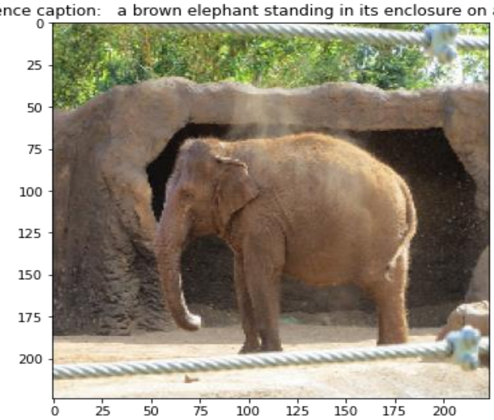
Generated Captions for beam size= 9
a group of people pose xFor a picture
a group of people pose xFor a picture on skis skis
a group of people on xFor a picture
a group of people pose xFor a picture
a group of people on xFor a picture on skis skis
a group of people pose xFor a picture on a skis
a group of skiers pose xFor a picture
a group of people on xFor a picture
a group of skiers pose xFor a picture on skis skis

Reference caption: a green train parked next to a train station



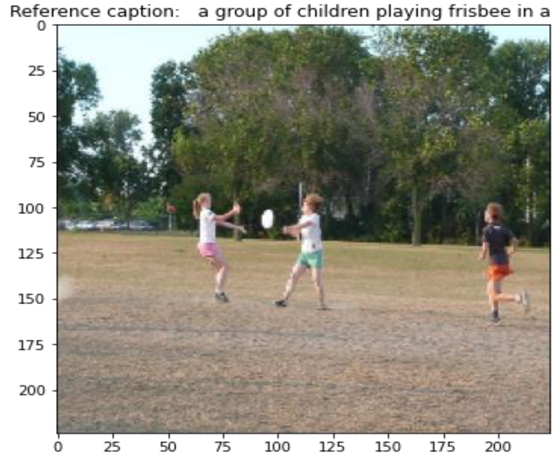
Generated Caption: a train and is on to a train station

Reference caption: a brown elephant standing in its enclosure on a sunny day



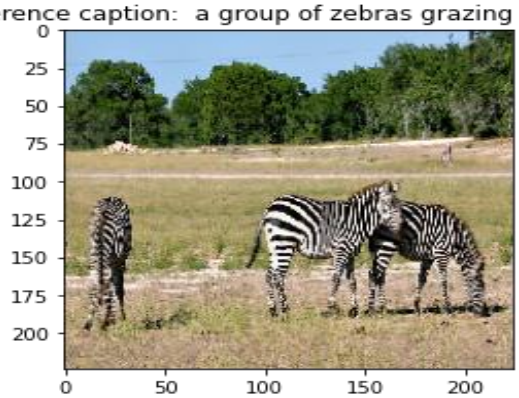
Generated Caption: a baby elephant standing next a pen in a sunny day

Reference caption: a group of children playing frisbee in a field



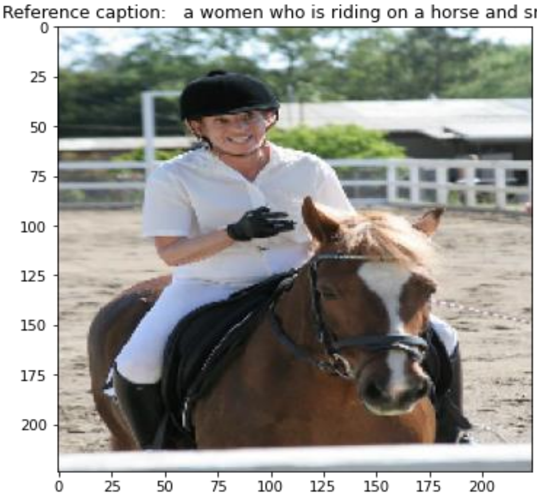
Generated Caption: a group of people playing soccer in a field

Reference caption: a group of zebras grazing in a field



Generated Captions for beam size= 5
a zebra of zebras standing in a field
a zebra of zebras standing on a field
a group of zebras standing in a field
a couple of zebras standing in a field
a zebra of zebras are in a field

Reference caption: a women who is riding on a horse and smiling

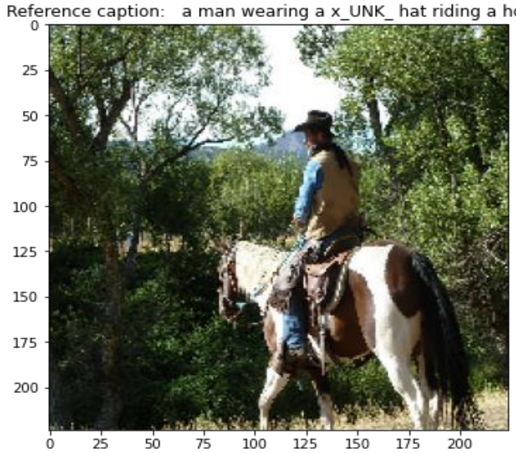


Generated Caption: a woman who is riding a a horse

Reference caption: a person riding a motorcycle down a road

Generated Caption: a man riding a motorcycle on a street

Reference caption: a man wearing a x_UNK_ hat riding a horse



Generated Caption: a man riding a x_UNK_ hat on a horse

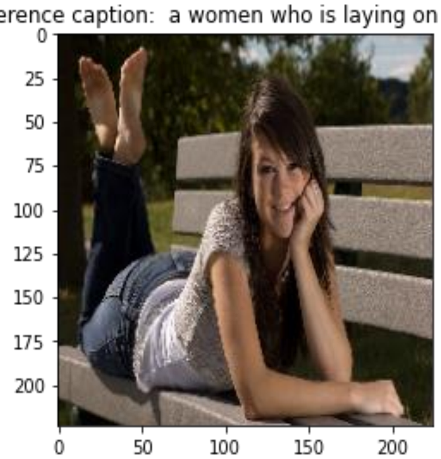
Reference caption: a couple fly a kite over the lake

Generated Caption: a group of a kite in a beach

Reference caption: a dog sitting on a bench looking into the distance

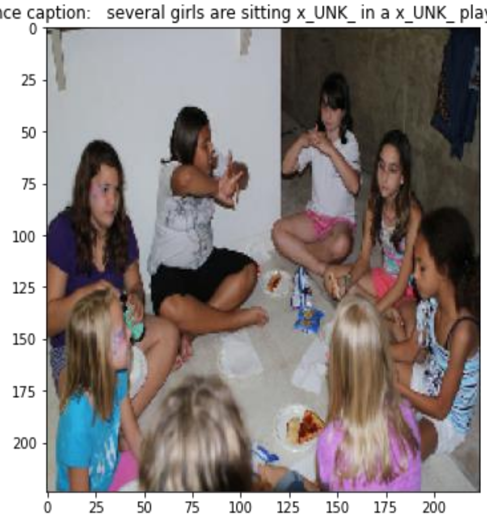
Generated Caption: a dog is on a couch in at a camera

Reference caption: a women who is laying on a bench



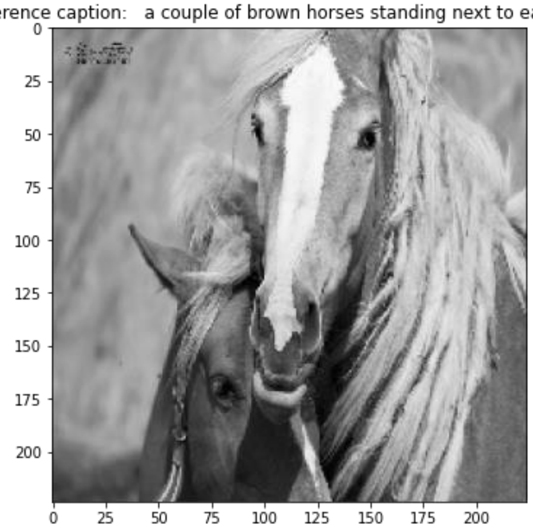
Generated Captions for beam size = 3
a woman who is sitting down a bench
a woman who is sitting down a bench
a woman who is sitting on a bench

Reference caption: several girls are sitting x_UNK_ in a x_UNK_ playing a game



Generated Caption: a people are sitting on on a x_UNK_

Reference caption: a couple of brown horses standing next to each other



Generated Caption: a brown of horses horses standing next to each other

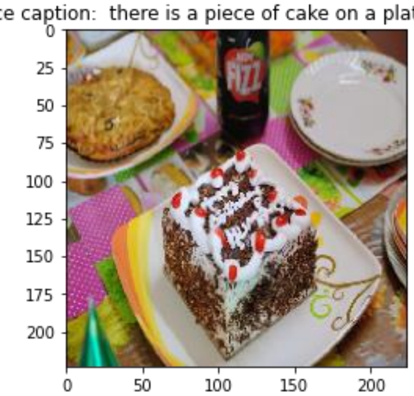
Reference caption: a table topped with wine glasses and eating x_UNK_

Generated Captions for beam size= 9
a table that with a glasses and wine
a table that with a glasses and plates

Reference caption: a couple of people standing in the snow

Generated Captions for beam size= 5
a group of people on on the snow with
a group of people standing on the snow with
a group of people that on the snow with
a group of people riding on the snow with
a group of people are on the snow with

Reference caption: there is a piece of cake on a plate on the table

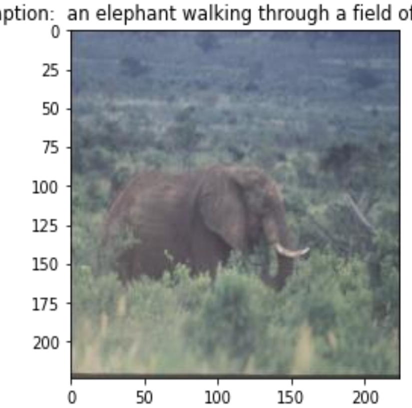


Generated Captions for beam size = 3
a is a plate of cake on a plate
a is a plate of cake on the plate
a are a plate of cake on a plate

Reference caption: a wooden table topped with lots of stuffed animals

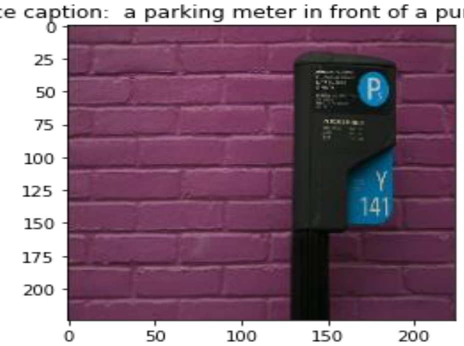
Generated Caption: a teddy table topped with a of stuffed animals

Reference caption: an elephant walking through a field of x_UNK_ and grass



Generated Captions for beam size= 5
a elephant is in a lush of grass
an elephant is in a lush of grass
a elephant is in a lush with grass
a elephant is in a grassy of grass
a elephant is in a lush of grass grass trees

Reference caption: a parking meter in front of a purple brick wall



Generated Captions for beam size= 9
a red meter sitting front of a building house building
a red meter with front of a building house building
a red meter sitting front of a building parking building
a red meter sitting front of a building brick building
a red meter with front of a building parking building
a red meter with front of a building brick building
a red meter sitting front of a building and building
a red meter is front of a building house building
a red meter with front of a building and building

Reference caption: a group of people sitting around each other on a couch

Generated Caption: a group of people sitting at a other on a couch

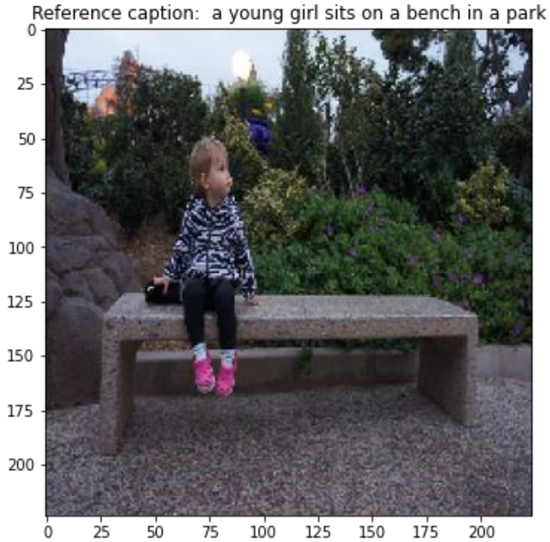
Reference caption: a man who is performing a jump on a snowboard

Generated Captions for beam size= 5
a man on is skiing a trick in a snowy
a man on is skiing a trick in a snowboard
a snowboarder on is skiing a trick in a snowy
a person on is skiing a trick in a snowy
a man riding is skiing a trick in a snowy

Reference caption: a group of people flying kites in the park

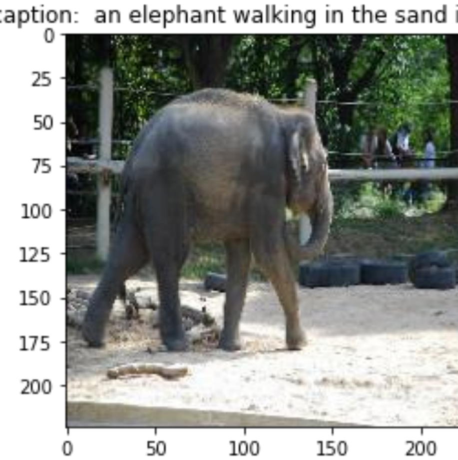
Generated Caption: a man of people standing kites in a sky
BLEU Score:0.6104735835807844

Reference caption: a young girl sits on a bench in a park



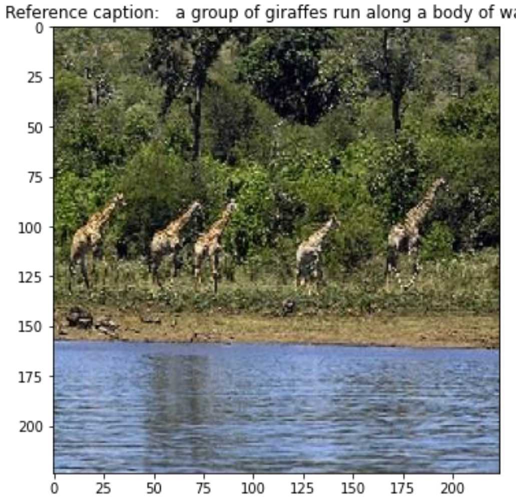
Generated Caption: a little boy sitting on a bench in front x_UNK_
BLEU Score:0.2777619034011791

Reference caption: an elephant walking in the sand in an x_UNK_ area



Generated Captions for beam size = 7
a elephant standing down the middle near the enclosure
a elephant standing on the middle near the enclosure
a elephant is down the middle near the enclosure
a elephant is on the middle near the enclosure
a elephant standing down the middle near a enclosure
a elephant standing on the middle near a enclosure
a elephant standing down the middle by the enclosure

Reference caption: a group of giraffes run along a body of water

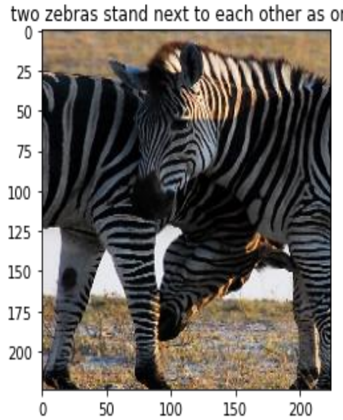


Generated Caption: a group of people are in a river of water

Reference caption: a bathroom with a vanity mirror toilet and bathtub

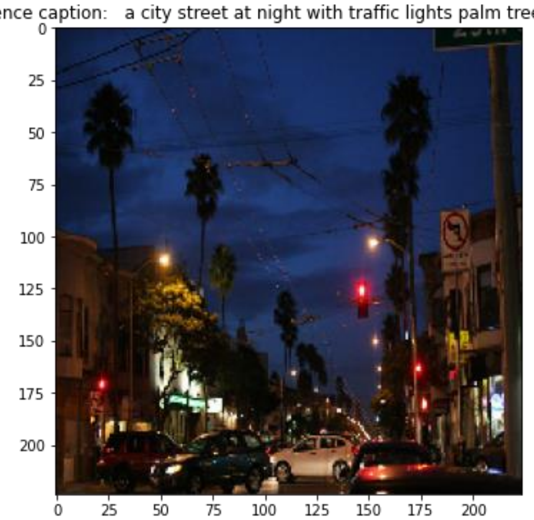
Generated Caption: a bathroom with a toilet sink and and sink

Reference caption: two zebras stand next to each other as one looks at the ground



Generated Captions for beam size= 5
a zebras are in to each other in others zebra x_UNK_ the camera
two zebras are in to each other in others zebra x_UNK_ the camera
a zebras standing in to each other in others zebra x_UNK_ the camera
a zebras are in to each other on others zebra x_UNK_ the camera
two zebras standing in to each other in others zebra x_UNK_ the camera

Reference caption: a city street at night with traffic lights palm trees and cars



Generated Caption: a street street with night with a lights

Reference caption: a black and white picture of a family lying on a bed

Generated Caption: a group and white photo of a man and on a bed

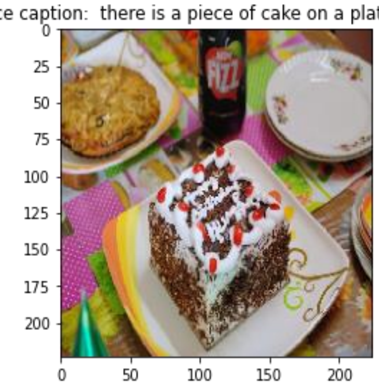
Reference caption: a cat sitting on a counter in a room

Generated Caption: a cat sitting on a x_UNK_ next a bathroom
BLEU Score:0.4671379777282001

Reference caption: a very old image of men leading a carriage

Generated Caption: a group large photo of a in horses cow
BLEU Score:0.7598356856515925

Reference caption: there is a piece of cake on a plate on the table

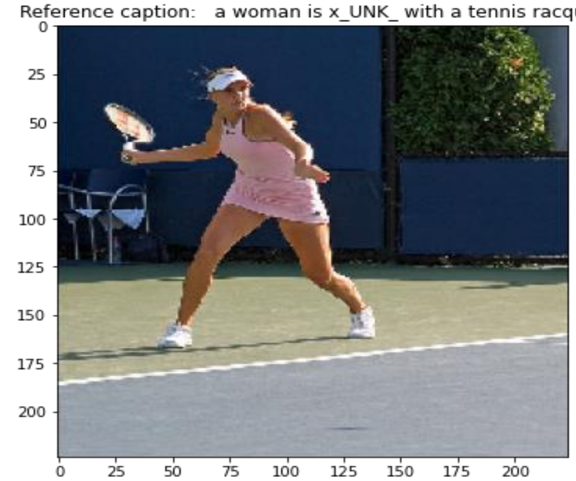


Generated Captions for beam size = 3
a is a plate of cake on a plate
a is a plate of cake on the plate
a are a plate of cake on a plate

Reference caption: a man on a surfboard riding a wave

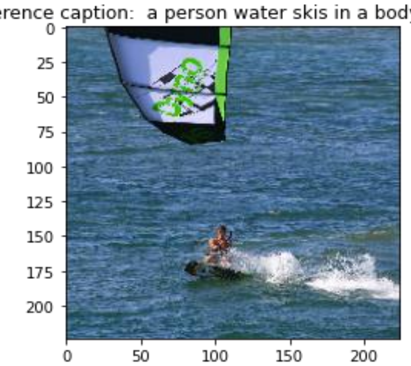
Generated Caption: a man riding a surfboard riding a wave
BLEU Score:0.5946035575013605

Reference caption: a woman is x_UNK_ with a tennis racquet



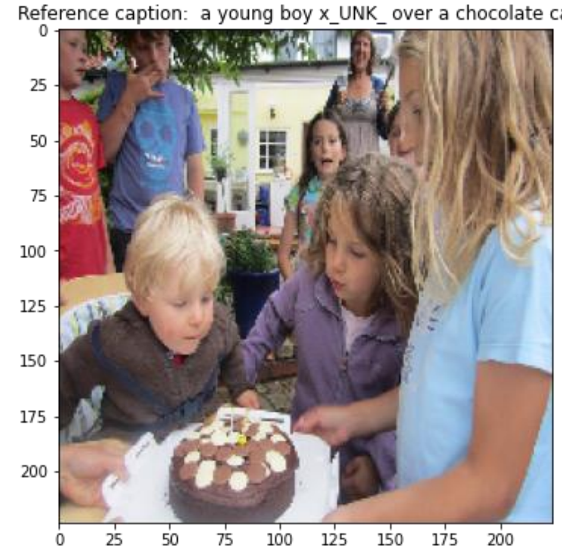
Generated Caption: a woman is playing a her tennis racket

Reference caption: a person water skis in a body of water



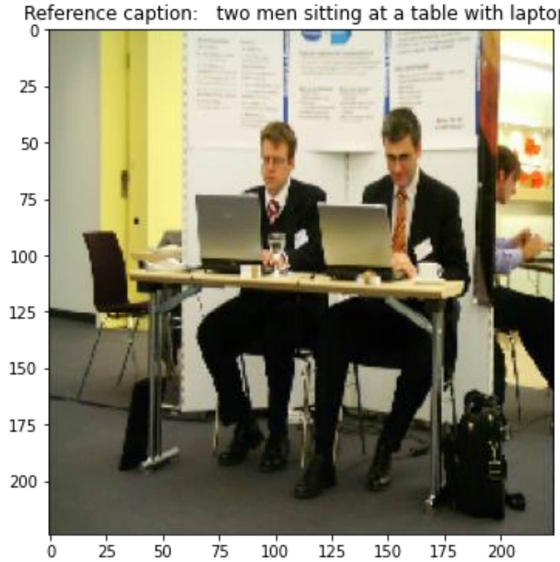
Generated Captions for beam size= 9
a man on skiing in a body of water
a man on skiing on a body of water
a person on skiing in a body of water
a man on skiing in the body of water
a person on skiing on a body of water
a man on skiing on the body of water
a person on skiing in the body of water
a man riding skiing in a body of water
a man is skiing in a body of water

Reference caption: a young boy x_UNK_ over a chocolate cake



Generated Caption: a couple child eating a eating cake donut
BLEU Score:0.7825422900366437

Reference caption: two men sitting at a table with laptops

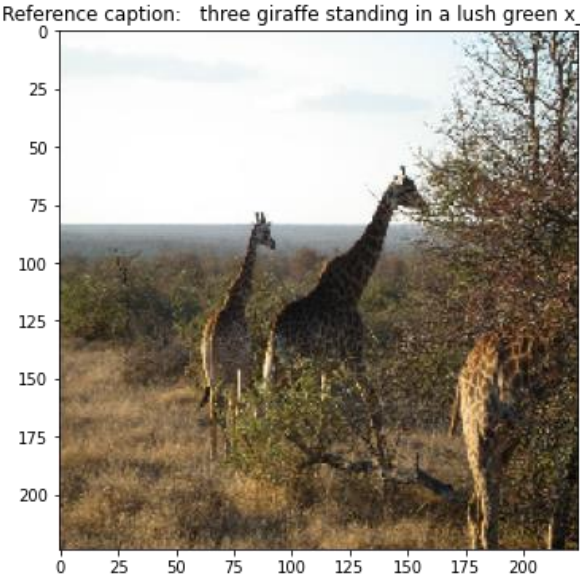


Generated Caption: a men are at a table with laptops

Reference caption: a group of people that are standing in the snow

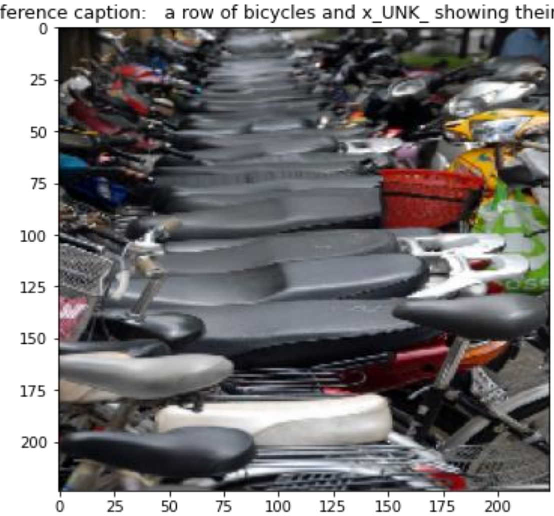
Generated Caption: a group of people on are skiing in the snow

Reference caption: three giraffe standing in a lush green x_UNK_



Generated Caption: a giraffes standing next a field green field

Reference caption: a row of bicycles and x_UNK_ showing their x_UNK_

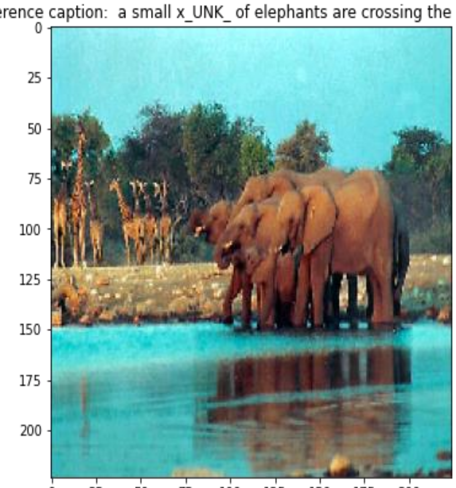


Generated Caption: a row of motorcycles parked a on a x_UNK_

Reference caption: two trains are parked in a train x_UNK_

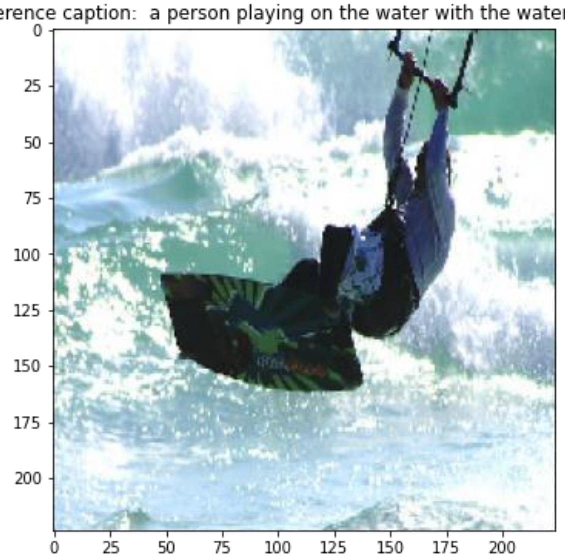
Generated Caption: a trains are parked on a train station

Reference caption: a small x_UNK_ of elephants are crossing the small lake



Generated Caption: a herd herd of elephants walking walking the water river
BLEU Score:0.4591497693322865

Reference caption: a person playing on the water with the water x_UNK_

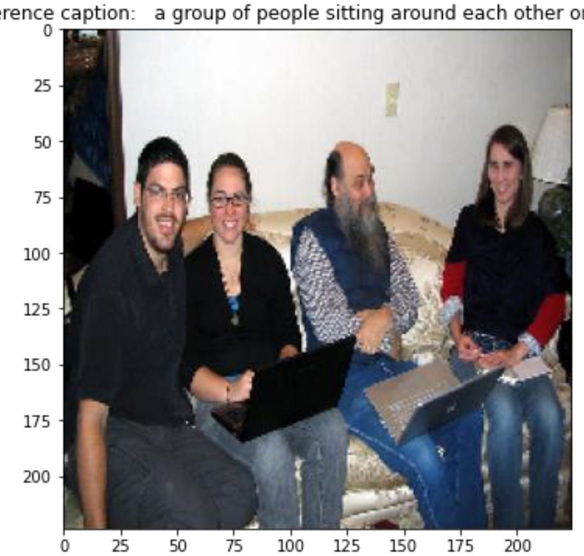


Generated Caption: a man on in a water catching a ocean
BLEU Score:0.6799308458396492

Reference caption: a city street at night with traffic lights palm trees and cars

Generated Caption: a street street with night with a lights

Reference caption: a group of people sitting around each other on a couch



Generated Caption: a group of people sitting at a other on a couch

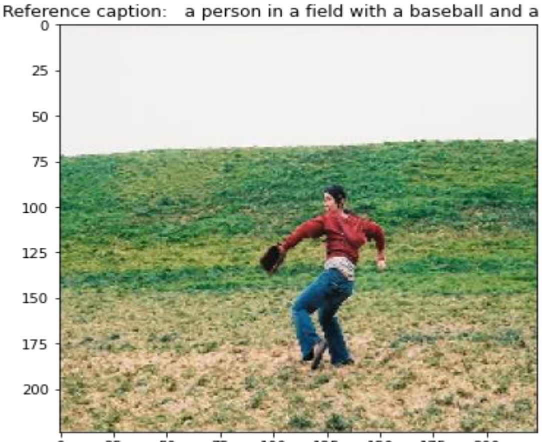
Reference caption: a group of people flying kites in the park

Generated Captions for beam size= 5
a man of people standing kites in a sky
a group of people standing kites in a sky
a man of people standing kites on a sky
a group of people standing kites on a sky
a man of people standing kites in a sky

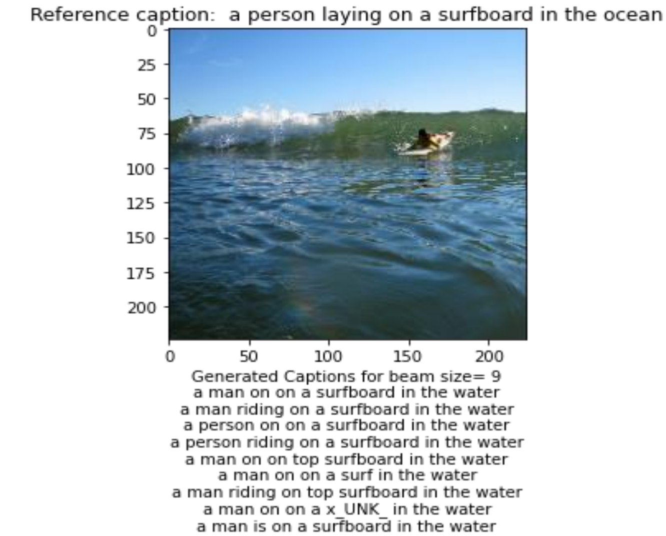
Reference caption: a black and white photo of a couple of baseball players

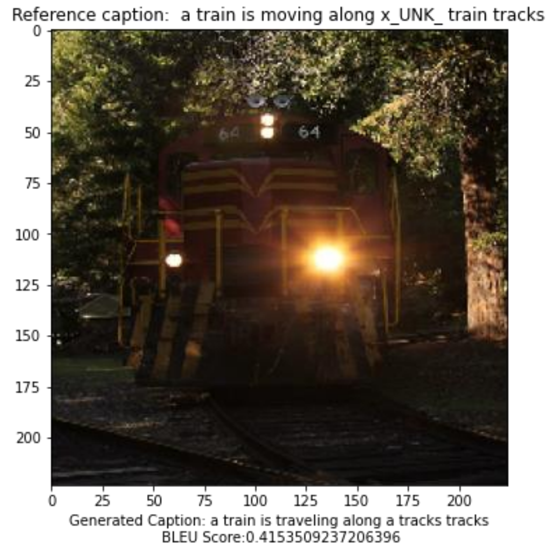
Generated Caption: a baseball and white photo of a baseball of baseball players

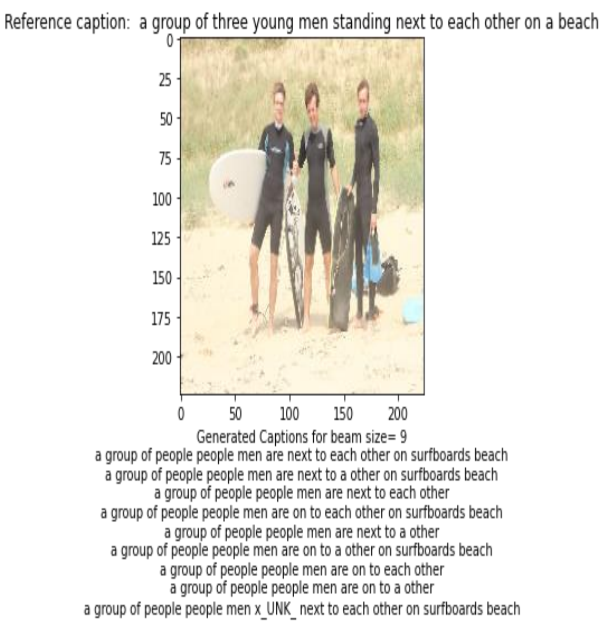
Reference caption: a person in a field with a baseball and a glove

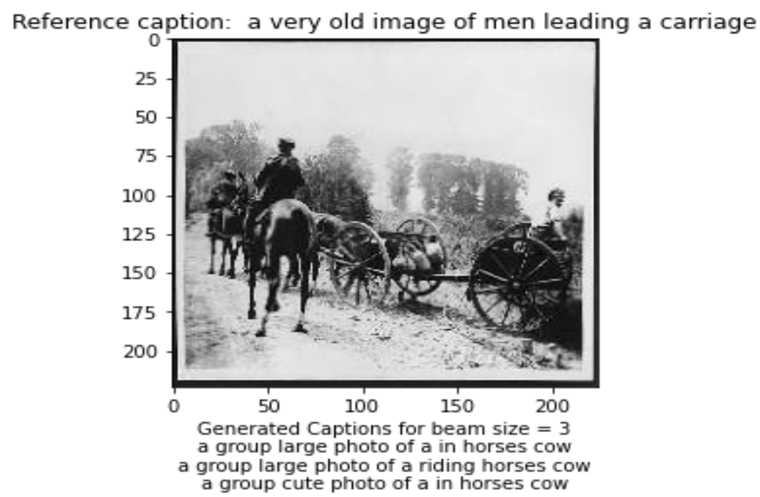


Generated Caption: a man on a field flying a frisbee bat a glove

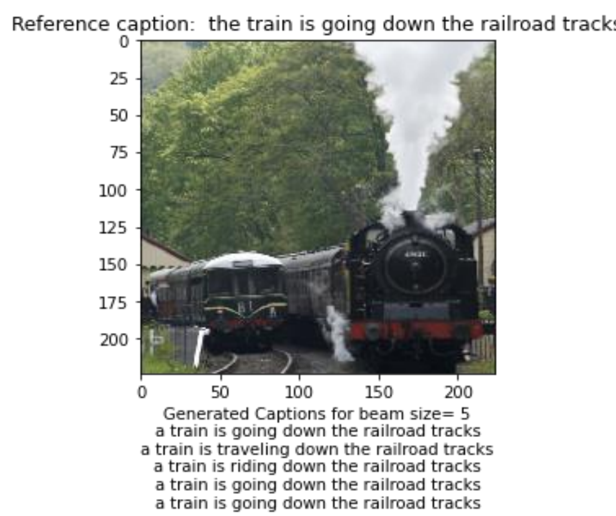

Reference caption: a person laying on a surfboard in the ocean
Generated Captions for beam size= 9
a man on on a surfboard in the water
a man riding on a surfboard in the water
a person on on a surfboard in the water
a person riding on a surfboard in the water
a man on on top surfboard in the water
a man on on a surf in the water
a man riding on top surfboard in the water
a man on on a x_UNK_ in the water
a man is on a surfboard in the water


Reference caption: a train is moving along x_UNK_ train tracks
Generated Caption: a train is traveling along a tracks tracks
BLEU Score:0.4153509237206396


Reference caption: a group of three young men standing next to each other on a beach
Generated Captions for beam size= 9
a group of people people men are next to each other on surfboards beach
a group of people people men are next to a other on surfboards beach
a group of people people men are next to each other
a group of people people men are on to each other on surfboards beach
a group of people people men are next to a other
a group of people people men are on to a other on surfboards beach
a group of people people men are on to each other
a group of people people men are on to a other
a group of people people men x_UNK_ next to each other on surfboards beach


Reference caption: a very old image of men leading a carriage
Generated Captions for beam size = 3
a group large photo of a in horses cow
a group large photo of a riding horses cow
a group cute photo of a in horses cow

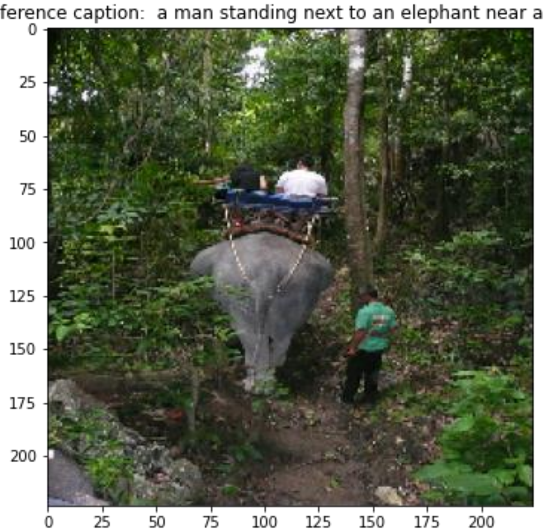

Reference caption: a laptop and computer sitting on a desk
Generated Captions for beam size= 5
a laptop computer a on on a desk
a laptop computer a on on a table
a laptop computer a on on a desk
a laptop computer a on on a table
a desk computer a on on a desk


Reference caption: the train is going down the railroad tracks
Generated Captions for beam size= 5
a train is going down the railroad tracks
a train is traveling down the railroad tracks
a train is riding down the railroad tracks
a train is going down the railroad tracks
a train is going down the railroad tracks

Reference caption: an orange and brown bus is in the city

Generated Captions for beam size= 5
a x_UNK_ and white bus on driving the street
a x_UNK_ and white bus on driving the street
a x_UNK_ and white bus on driving a street
a old and white bus on driving the street
a orange and white bus on driving the street

Reference caption: a man standing next to an elephant near a forest



Generated Caption: a man is on to a umbrella in a tree
BLEU Score:0.4591497693322865

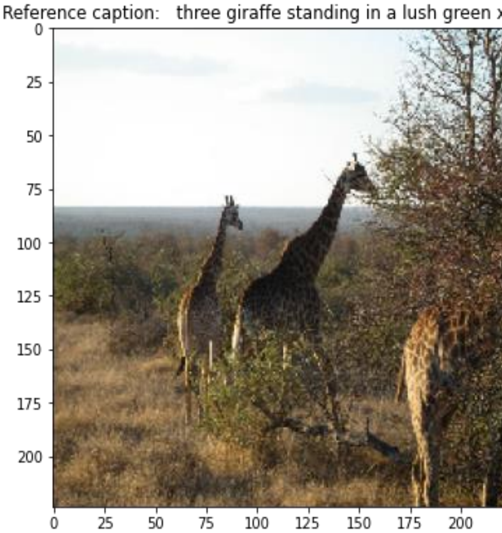
Reference caption: a young boy sitting in front of a pizza in a box

Generated Caption: a young boy eating at front of a slice
BLEU Score:0.39805020134303426

Reference caption: a man on a surfboard riding a wave

Generated Captions for beam size= 5
a man riding a surfboard riding a wave
a man on a surfboard riding a wave
a man is a surfboard riding a wave
a man riding a surf riding a wave
a man in a surfboard riding a wave

Reference caption: three giraffe standing in a lush green x_UNK_

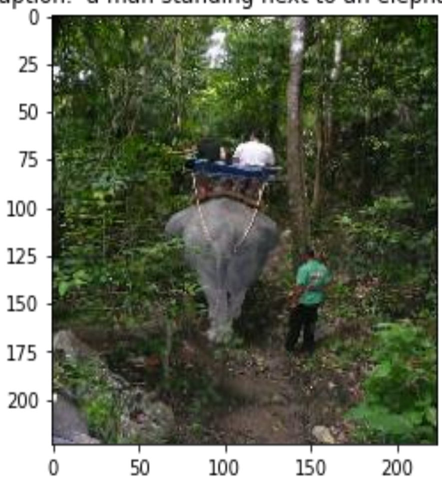


Generated Caption: a giraffes standing next a field green field

Reference caption: a group of elephants are walking around a zoo

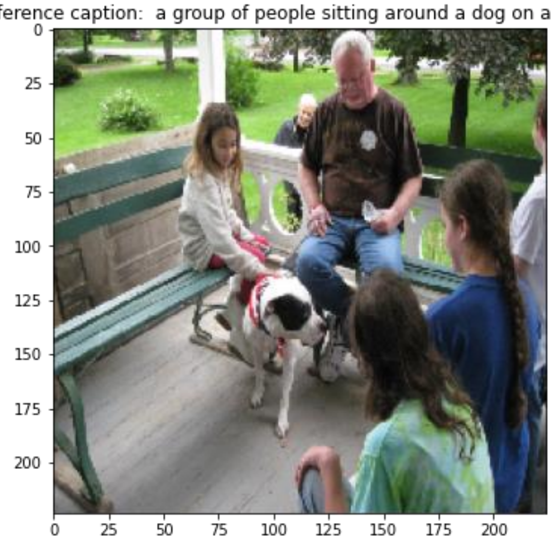
Generated Caption: a group of elephants x_UNK_ standing in a dirt
BLEU Score:0.3155984539112945

Reference caption: a man standing next to an elephant near a forest



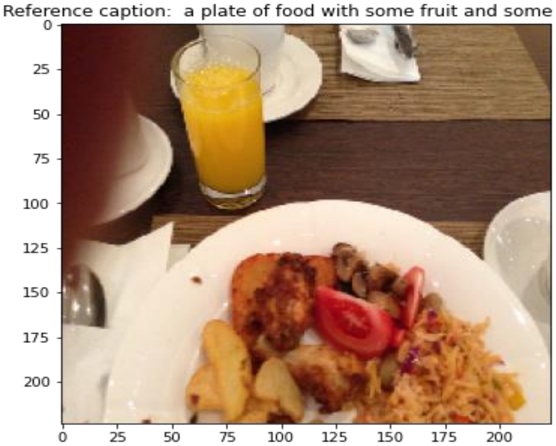
Generated Captions for beam size = 3
a man is on to a umbrella in a tree
a man is on to a x_UNK_ in a tree
a man is next to a umbrella in a tree

Reference caption: a group of people sitting around a dog on a x_UNK_



Generated Caption: a group of people sitting on a table
BLEU Score:0.40866465020165643

Reference caption: a plate of food with some fruit and some pasta

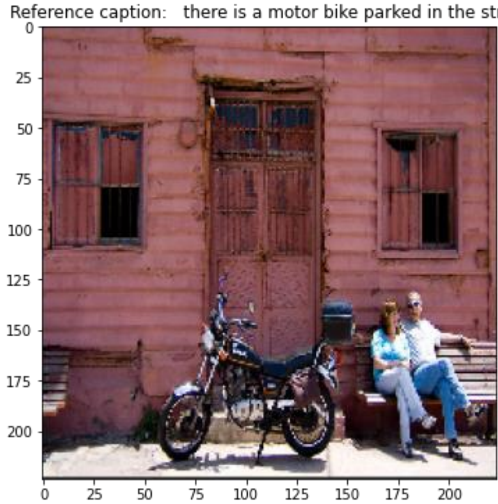


Generated Caption: a plate of food with a x_UNK_ and a x_UNK_
BLEU Score:0.4111336169005197

Reference caption: the desk has two laptops a monitor and keyboard

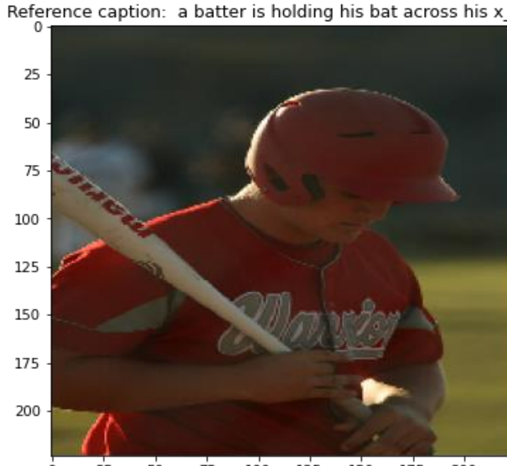
Generated Captions for beam size = 3
a computer has a computers and mouse and a
a computer has a x_UNK_ and mouse and a
a computer has a monitors and mouse and a

Reference caption: there is a motor bike parked in the street



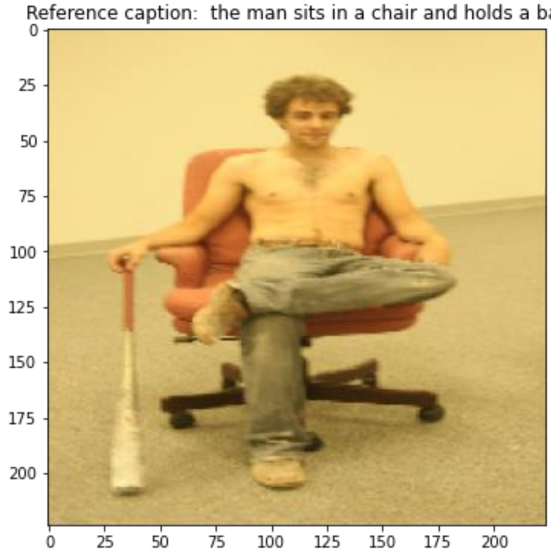
Generated Caption: a is a motorcycle bike parked on front street

Reference caption: a batter is holding his bat across his x_UNK_



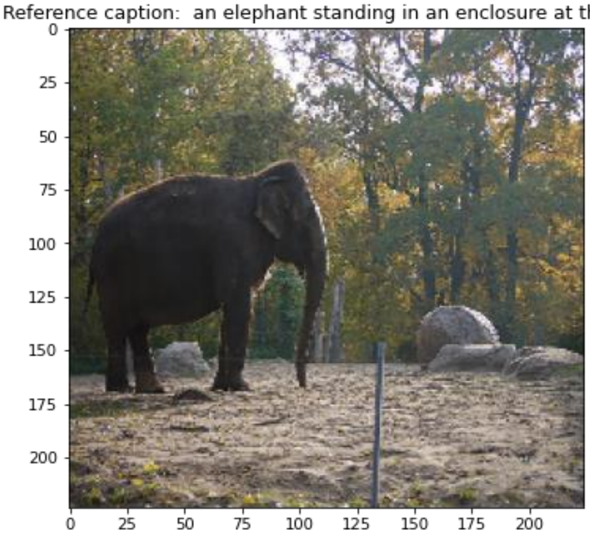
Generated Caption: a baseball is swinging a bat over the x_UNK_ x_UNK_
BLEU Score:0.7952707287670506

Reference caption: the man sits in a chair and holds a bat



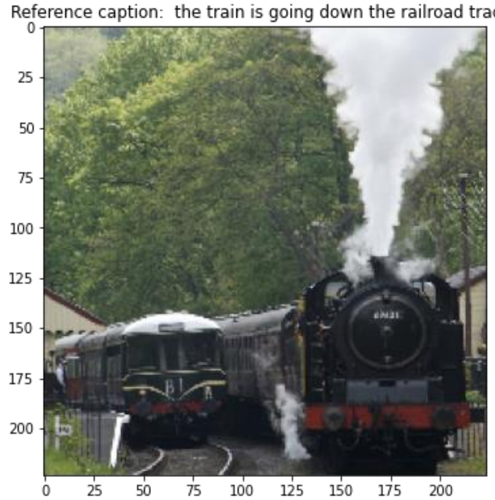
Generated Caption: a man is on the chair with holding a x_UNK_
BLEU Score:0.8408964152537145

Reference caption: an elephant standing in an enclosure at the zoo



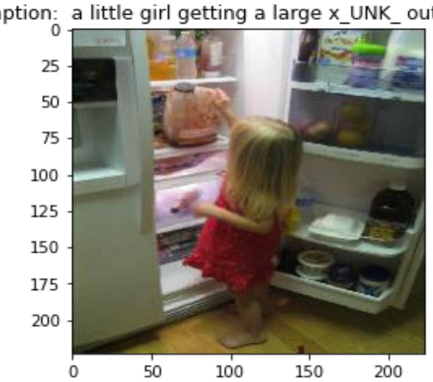
Generated Caption: an elephant standing in the enclosure with a zoo
BLEU Score:0.34329452398451965

Reference caption: the train is going down the railroad tracks



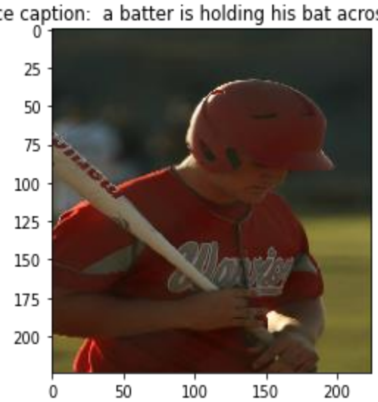
Generated Caption: a train is going down the railroad tracks
BLEU Score:0.8408964152537145

Reference caption: a little girl getting a large x_UNK_ out of the refrigerator



Generated Captions for beam size= 9
a little girl standing food refrigerator donut x_UNK_ of the refrigerator
a little girl standing food refrigerator donut x_UNK_ of a refrigerator
a little girl standing food refrigerator x_UNK_ x_UNK_ of the refrigerator
a little girl standing food refrigerator donut donut of the refrigerator
a little girl standing food refrigerator donut x_UNK_ of the refrigerator
a little girl standing food refrigerator x_UNK_ x_UNK_ of a refrigerator
a little girl standing food refrigerator donut donut of a refrigerator
a little girl is food refrigerator donut x_UNK_ of the refrigerator
a little girl standing food refrigerator donut x_UNK_ of a refrigerator

Reference caption: a batter is holding his bat across his x_UNK_

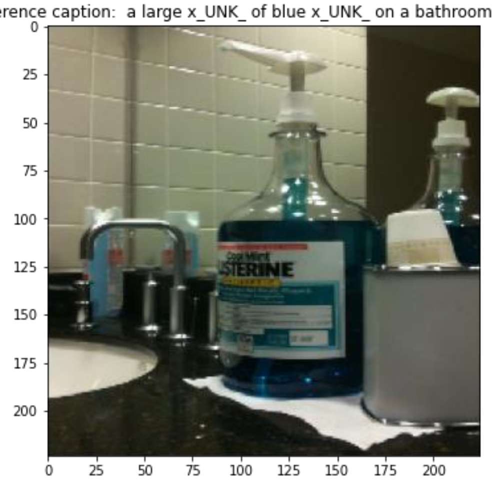


Generated Captions for beam size= 5
a baseball is swinging a bat over the x_UNK_ x_UNK_
a baseball in swinging a bat over the x_UNK_ x_UNK_
a baseball is swinging a bat over the x_UNK_
a baseball in swinging a bat over the x_UNK_
a baseball is swinging the bat over the x_UNK_ x_UNK_

Reference caption: a couple of people standing in the snow

Generated Caption: a group of people on on the snow with
BLEU Score:0.6104735835807844

Reference caption: a large x_UNK_ of blue x_UNK_ on a bathroom counter

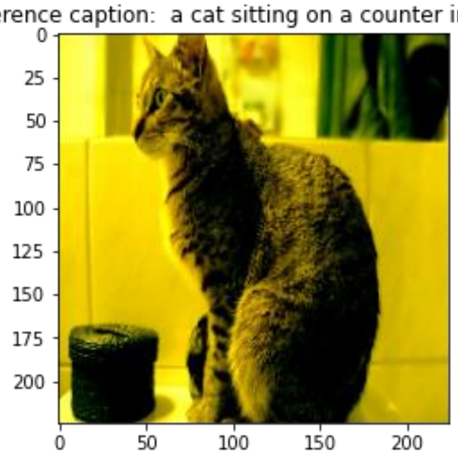


Generated Caption: a sink bathroom x_UNK_ x_UNK_ x_UNK_ sitting a counter
BLEU Score:0.8085785995823291

Reference caption: a desk with a chair monitors and a keyboard

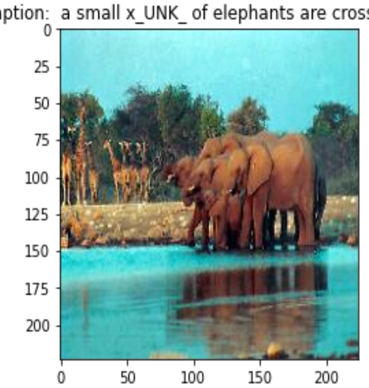
Generated Caption: a desk with a laptop and and a laptop

Reference caption: a cat sitting on a counter in a room



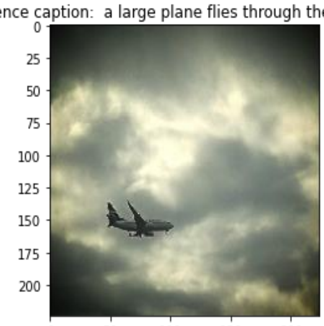
Generated Captions for beam size= 5
a cat sitting on a x_UNK_ next a bathroom
a cat sitting on top x_UNK_ next a bathroom
a cat is on a x_UNK_ next a bathroom
a cat sitting on a x_UNK_ with a bathroom
a cat sitting on a toilet next a bathroom

Reference caption: a small x_UNK_ of elephants are crossing the small lake



Generated Captions for beam size= 5
a herd herd of elephants walking walking the water river
a herd herd elephant elephants walking walking the water river
a herd herd of elephants walking x_UNK_ the water river
a herd herd of elephants walking in the water river
a herd herd elephant elephants walking x_UNK_ the water river

Reference caption: a large plane flies through the cloudy sky



Generated Captions for beam size= 5
a plane x_UNK_ flying through the air sky
a plane x_UNK_ flying through the air sky
a x_UNK_ x_UNK_ flying through the air sky
a large x_UNK_ flying through the air sky
a plane x_UNK_ flying through the air sky

Reference caption: a person sitting on a couch with a remote

Generated Caption: a man sitting on a couch with a x_UNK_ control
BLEU Score:0.537284965911771

Appendix II – Source Code

```
# -*- coding: utf-8 -*-
"""Final_project.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1AhHBTYkqcNnzBOfoS4jwM_tLvBZ41u7m
"""

from __future__ import print_function, division

# Importing basics:
import numpy as np
import h5py
import matplotlib.pyplot as plt
import pandas as pd

# For image pre/processing:
from PIL import Image
#from skimage.io import imread
from skimage.transform import resize
from skimage import io, transform

# Retrieve and manipulate paths:
import urllib
import requests
import os
import shutil
import pickle


# Performance metrics and train test split:
from sklearn.model_selection import train_test_split

# Useful
from tqdm import tqdm
import time
import copy

# PyTorch's:
import torch
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms, utils, datasets, models
import torch.nn as nn
```

```
import torch.optim as optim
from torch.optim import lr_scheduler
import torchvision
import torch.nn.functional as F
from torch.autograd import Variable

# To access Google Drive:
from google.colab import drive


# interactive mode
plt.ion()

# For translation model evaluation

from nltk.translate.bleu_score import sentence_bleu
#from nltk.translate.meteor_score import meteor_score


# Ignore warnings
import warnings
warnings.filterwarnings("ignore")

# PyTorch's versions:
print("PyTorch Version: ",torch.__version__)
print("Torchvision Version: ",torchvision.__version__)
print("NumPy Version: ",np.__version__)
drive.mount("/content/gdrive")

colap_path_train = '/content/gdrive/My Drive/Data/eee443_project_dataset_train.h5'
colap_path_test = '/content/gdrive/My Drive/Data/eee443_project_dataset_test.h5'

# Getting the data:
with h5py.File(colap_path_train,'r') as f:
# Names variable contains the names of training and testing file
    names = list(f.keys())
    train_cap = f[names[0]][()]
    train_imid = np.array(f[names[1]][()])
    train_imid -=1
    train_url = np.array(f[names[3]][()])

data_dir = '/content/gdrive/My Drive/Data/train'
os.chdir(path = data_dir)

print(train_cap.shape)
print(train_imid.shape)
```

```
print(train_imid.max()),
print(train_imid.min())
print(train_url.shape)

X = train_imid
y = train_cap

train_inds, val_inds, train_caps, val_caps = train_test_split(
     X, y, test_size = 0.15, random_state = 42)

train_cap_Str = {}
for i in range(len(train_caps)):
  train_cap_Str[i] = [element for element in train_caps[i] if element != 0]


val_cap_Str = {}
for i in range(len(val_caps)):
  val_cap_Str[i] = [element for element in val_caps[i] if element != 0]

sentence_lens_train = [len(ele) for ele in train_cap_Str.values()]
sentence_lens_val = [len(ele) for ele in val_cap_Str.values()]

print(train_inds.shape)
print(train_caps.shape)
print(val_inds.shape)
print(val_caps.shape)
print(np.isclose(0.15, val_inds.shape[0] /(train_inds.shape[0] + val_inds.shape[0]),5))

class ImageCaptionData(Dataset):
  """ Image Caption dataset
    Args:
      image_urls (np.ndarray)       : Image URL's.
      img_inds (np.ndarray)         : Indices of the images
      captions (np.ndarray)         : Captions indices for images
      sentence_lens (np.ndarray)    : Each captions length without pads
      transform (callable,optinal) : Transformation to be applied on a sample images

    """

  def __init__(self,image_urls,img_inds,captions,sentence_lens,transform = None):


    #self.data_dir = data_dir
    #self.image_list = os.listdir(self.data_dir)
    self.image_urls = image_urls
    self.img_inds = img_inds
```

```
    self.captions = captions
    self.transform = transform
    self.sentence_lens = sentence_lens


  def __len__(self):
    return self.img_inds.shape[0]

  def __getitem__(self,index):
    """ Get input,label in dict format """

    # Check the indices is in the correct format:
    if torch.is_tensor(index):
      index = index.tolist()

    connected = False

    while not connected:

      try:
        # Since we may use the same img more than 1 time, we slice the needed index:
        img_index = self.img_inds[index]

        img_url = self.image_urls[img_index].decode('UTF-8')
        img_name = img_url.split("/")[-1].strip()

        # Feeding needed index to get the path of the image in Google Colab:
        #img_path = self.image_list[img_index]
        #urllib.request.urlretrieve(img_url,img_name)
        # Reading images in PIL format since PyTorch works with PIL or Tensor types:
        img = Image.open(img_name).convert('RGB')

        # Reading the corresponding caption:
        caption = self.captions[index]

        sen_len = self.sentence_lens[index]

        connected = True

      except:
        index = np.random.randint(self.img_inds.shape[0])


    # If not works:
    #img = io.imread(img_path)
```

```
    # Converting array and normalizing:
    #img = np.array(img) / 255

    # Image + Caption:
    sample = {'image' : img, 'caption' : caption}


    # Apply transformation to the images:
    # Resize + Normalize + Convert to tensor format
    if self.transform is not None:
      sample = self.transform(sample)

    return sample,sen_len

class Resize(object):
  """ Rescale the images to the expected format.
      It depends on the CNN model's expected input size

      Args:

        out_size (tuple or int) : It is the desired output size. If the output size
        is given as a tuple format, we matched to dimensions, e.g. if we give (484x676)
        this class returns the image with dimensions with (484x676). After that, if we give
        int value to the Resize class, we output the square of the given integer, e.g. let
        outsize = 224, this class returns 224x224 resized format.
  """

  def __init__(self,out_size):
    assert isinstance(out_size, (int,tuple))
    self.out_size = out_size

  def __call__(self,sample):
    img, caption =  sample['image'],sample['caption']

    # Get height and width dimensions of the img
    #H,W = img.shape[:2]


    if isinstance(self.out_size, int):
      new_H,new_W = self.out_size, self.out_size

    else:
      new_H,new_W = self.out_size

    #new_H,new_W = int(new_H),int(new_W)
```

```
    #resized_img = transform.resize(img,(new_H,new_W))
    #tfsm_resize = transforms.Resize((new_H,new_W),interpolation = Image.NEAREST)
    #resized_img = tfsm_resize(Image.fromarray(img.astype('uint8'), 'RGB'))

    tfsm = transforms.Resize((new_H,new_W),interpolation=Image.NEAREST)


    return {'image' : tfsm(img), 'caption' : caption}

class RandomCrop(object):
    """Crop randomly the image in a sample.

    Args:
        output_size (tuple or int): Desired output size. If int, square crop
            is made.
    """

    def __init__(self, output_size):
        assert isinstance(output_size, (int, tuple))
        if isinstance(output_size, int):
            self.output_size = (output_size, output_size)
        else:
            assert len(output_size) == 2
            self.output_size = output_size

    def __call__(self, sample):
        image, caption = sample['image'], sample['caption']

        #h, w = image.shape[:2]
        #new_h, new_w = self.output_size

        #top = np.random.randint(0, h - new_h)
        #left = np.random.randint(0, w - new_w)

        #image = image[top: top + new_h,
                      #left: left + new_w]

        tsfm = transforms.RandomCrop(self.output_size)

        return {'image': tsfm(image), 'caption': caption}


class RandomHorizontalFlip(object):
  def __call__(self,sample):
    image, caption = sample['image'], sample['caption']
    in_tsfm = transforms.RandomHorizontalFlip(p=0.5)
```

```
    return {'image' : in_tsfm(image),'caption' : caption}

class CenterCrop(object):
  def __init__(self,out_size):
    self.out_size = out_size

  def __call__(self,sample):
    image, caption = sample['image'], sample['caption']
    in_tsfm = transforms.CenterCrop(self.out_size)
    return {'image' : in_tsfm(image),'caption' : caption}

class RandomVerticalFlip(object):
  def __call__(self,sample):
    image, caption = sample['image'], sample['caption']
    in_tsfm = transforms.RandomVerticalFlip(p=0.5)
    return {'image' : in_tsfm(image),'caption' : caption}
    #{'image' : in_tsfm(Image.fromarray(np.uint8(image))).convert('RGB'),'caption' : caption}

class RandomResizedCrop(object):
  def __init__(self,out_size):
    self.out_size = out_size

  def __call__(self,sample):
    image, caption = sample['image'], sample['caption']
    in_tsfm = transforms.RandomResizedCrop(self.out_size)
    return {'image' : in_tsfm(image),'caption' : caption}

class ToTensor(object):
  """ From numpy ndarray to PyTorch tensor format

      Args:
        sample (tuple) : samples contains both image and caption, this class
        apply transformation on images to normalize the ImageNet format since
        we utilizes pre-trained state-of-the-art models for transfer learning
        and all models are trained in Imagenet dataset. Then, convert both images
        and captions to PyTorch's tensor format.
  """

  def __call__(self,sample):
    img, caption =  sample['image'], sample['caption']

    # PyTorch's expected image size C x H x W
    #formatted_img = np.transpose(img,(2, 0, 1))
    #img = img.tranpose(2, 0, 1)
    #img = np.swapaxes(img,-1,0)
    #img = np.swapaxes(img,1,2)
```

```
    tf = transforms.ToTensor()

    #tensor_img = tf(img).float()
    # Convert from numpy ndarray to tensor format
    #tensor_img, tensor_caption = torch.from_numpy(formatted_img).float(), torch.from_numpy(caption)

    tensor_caption = torch.from_numpy(caption)
    #tensor_img = torch.from_numpy(img).float()

    return {'image' : tf(img), 'caption' : tensor_caption}



class Normalize(object):
  """ Custom normalization to images with given mean and
      standart deviation.
  """

  def __call__(self,sample):
    tensor_img, tensor_caption =  sample['image'], sample['caption']

    # Image normalization on samples, mean's and std's are specifically selected to
    # obey ImageNet means and standart deviations:
    in_transform = transforms.Normalize(mean = [0.485, 0.456, 0.406],
                                        std = [0.229, 0.224, 0.225])

    #in_transform = transforms.Compose([transforms.Normalize([0.5],[0.5])])

    # Applying normalization:
    tensor_img = in_transform(tensor_img)

    #tensor_img = torch.clip(tensor_img,0,1)


    return {'image' : tensor_img, 'caption' : tensor_caption}

class PreTrainedModels(object):
  """ In this class, state of the art CNN models are placed.
      All models are trained on ImageNet dataset and 1000 classes.
      However, all necessary changes are done in terms of shapes of layer.
      Models : [resnet, alexnet, vgg, squeezenet, densenet, inception]
      Args:
        num_class (int): Number of classes in the dataset.

  """
```

```
def __init__(self,num_classes):

  self.num_classes = num_classes
  self.model = None
  self.input_size = None

def set_parameter_requires_grad(self,model,feature_extracting):
  """ İf feature extracting, we set pre-trained parameters's
      requires_grad = False, since no need to calculate the gra
      dients of the non-updatable parameters.

      Args :
        model (callable)              : PyTorch's torchvision's CNN models
        feature_extracting (Boolean) : True if feature extracking and False if fine-tuning
  """

  if feature_extracting:
    for param in model.parameters():
      param.requires_grad = False


def ResNet(self,feature_extract = True):
  """ ResNet 18
  Args:
    feature_extracting (Boolean) : True if feature extracking and False if fine-tuning

  Returns a tuple of :
    ResNet 18 pretained model and it's expected input size

  """

  self.model = torchvision.models.resnet152(pretrained=True)
  self.set_parameter_requires_grad(self.model,feature_extract)
  in_ftrs = self.model.fc.in_features
  modules = list(self.model.children())[:-1]      # delete the last fc layer.
  self.model = nn.Sequential(*modules)
  #self.model.fc = nn.Linear(in_ftrs,self.num_classes)
  self.input_size = 224

  return self.model,in_ftrs, self.input_size

def AlexNet(self,feature_extract = True):
  """ AlexNet
      Args:
        feature_extracting (Boolean) : True if feature extracking and False if fine-tuning
```

```
        Returns a tuple of :
          AlexNet pretained model and it's expected input size

    """


    self.model = torchvision.models.alexnet(pretrained=True)
    self.set_parameter_requires_grad(self.model,feature_extract)
    in_ftrs = self.model.classifier[6].in_features
    self.model = nn.Sequential(*list(self.model.children())[:-1])
    self.input_size = 224

    return self.model,in_ftrs, self.input_size

  def VGG(self,feature_extract = True):
    """ VGG11_bn
        Args:
          feature_extracting (Boolean) : True if feature extracking and False if fine-tuning

        Returns a tuple of :
          VGG11_bn pretained model and it's expected input size
    """

    self.model = torchvision.models.vgg19_bn(pretrained=True)
    self.set_parameter_requires_grad(self.model,feature_extract)
    in_ftrs = self.model.classifier[6].in_features
    self.model = nn.Sequential(*list(self.model.children())[:-1])
    self.input_size = 224

    return self.model, self.input_size


  def SqueezeNet(self,feature_extract = True):
    """ Squeezenet
        Args:
          feature_extracting (Boolean) : True if feature extracking and False if fine-tuning

        Returns a tuple of :
          Squeezenet  pretained model and it's expected input size

    """

    self.model = torchvision.models.squeezenet1_0(pretrained=True)
    self.set_parameter_requires_grad(self.model,feature_extract)
    self.model.classifier[1] = nn.Conv2d(512, self.num_classes, kernel_size=(1,1), stride=(1,1))
    self.model.num_classes = self.num_classes
```

```
    self.input_size = 224

    return self.model, self.input_size


  def DenseNet(self,feature_extract = True):
    """ DenseNet
        Args:
          feature_extracting (Boolean) : True if feature extracking and False if fine-tuning

        Returns a tuple of :
          DenseNet  pretained model and it's expected input size

    """
    self.model = models.densenet201(pretrained=True)
    self.set_parameter_requires_grad(self.model, feature_extract)
    num_ftrs = self.model.classifier.in_features
    self.model = nn.Sequential(*list(self.model.children())[:-1])
    self.input_size = 224

    return self.model,num_ftrs,self.input_size


  def Inception_v3(self,feature_extract = True):
    """ Inception v3.
        Be careful, expects (299,299) sized images and has auxiliary output
        Args:
          feature_extracting (Boolean) : True if feature extracking and False if fine-tuning

        Returns a tuple of :
          Inception_v3  pretained model and it's expected input size
    """

    self.model = torchvision.models.inception_v3(pretrained=True)
    self.set_parameter_requires_grad(self.model,feature_extract)
    # Handle the auxilary net
    num_ftrs = self.model.AuxLogits.fc.in_features
    self.model.AuxLogits.fc = nn.Linear(num_ftrs, self.num_classes)
    # Handle the primary net
    num_ftrs = self.model.fc.in_features
    self.model.fc = nn.Linear(num_ftrs,self.num_classes)
    self.input_size = 299

    return self.model, self.input_size

class EncoderCNN(nn.Module):
```

```
    def __init__(self,in_ftrs,model = None,embed_size = 300):
        super(EncoderCNN, self).__init__()

        if model is not None:
          self.model = model

        else:
          resnet = torchvision.models.resnet18(pretrained=True)
          modules = list(resnet.children())[:-1]      # delete the last fc layer.
          self.model = nn.Sequential(*modules)
          for param in self.model.parameters():
            param.requires_grad = False

        self.linear = nn.Linear(in_ftrs, embed_size)
        self.batchNorm = nn.BatchNorm1d(embed_size, momentum=0.01)

        # add another fully connected layer
        #self.embed = nn.Linear(in_features=524, out_features=embed_size)

        # dropout layer
        #self.dropout = nn.Dropout(0.5)


        # activation layers
        #self.prelu = nn.PReLU()

    def forward(self, images):

        features = self.model(images)
        features = features.reshape(features.size(0), -1)
        features = self.batchNorm(self.linear(features))

        return features

class DecoderRNN(nn.Module):
    def __init__(self, embed_size, hidden_size, vocab_size, torch_embedding = False,num_layers=1):
        super(DecoderRNN, self).__init__()

        # define the properties
        self.embed_size = embed_size
        self.hidden_size = hidden_size
        self.vocab_size = vocab_size

        self.embed = nn.Embedding(num_embeddings=self.vocab_size, embedding_dim=self.embed_size)

        if torch_embedding:
```

```
            self.embedding_glove6B = torch.load('embedding_glove6B')
            self.embed.weight.data.copy_(self.embedding_glove6B.vectors)
            self.embed.weight.requires_grad = False

        else:

            pretrainedEmbeds = np.loadtxt('embeds300.txt', delimiter=',')
            self.embed.weight.data.copy_(torch.from_numpy(pretrainedEmbeds))
            self.embed.weight.requires_grad = False


        # lstm cell
        self.lstm_cell = nn.LSTMCell(input_size = embed_size, hidden_size = hidden_size)

        #if num_layers==2:
        self.lstm_cell_layer_2 = nn.LSTMCell(input_size = hidden_size, hidden_size = hidden_size)

        # output fully connected layer
        self.fully_connected = nn.Linear(in_features = hidden_size, out_features= vocab_size)

        # embedding layer
        #self.embed = nn.Embedding(num_embeddings=self.vocab_size, embedding_dim=self.embed_size)

        # activations
        #self.softmax = nn.Softmax(dim=-1)

    def forward(self, features, captions):

        # batch size
        batch_size = features.size(0)

        # init the hidden and cell states to zeros
        hidden_state = torch.zeros((batch_size, self.hidden_size)).to(device)
        cell_state = torch.zeros((batch_size, self.hidden_size)).to(device)
        #hidden_state_layer_2 = torch.zeros((batch_size, self.hidden_size)).to(device)
        #cell_state_layer_2 = torch.zeros((batch_size, self.hidden_size)).to(device)

        # define the output tensor placeholder
        outputs = torch.empty((batch_size, captions.size(1), self.vocab_size)).to(device)

        # embed the captions
        captions_embed = self.embed(captions)

        # pass the caption word by word
        for t in range(captions.size(1)):
```

```
            # for the first time step the input is the feature vector
            if t == 0:
                hidden_state, cell_state = self.lstm_cell(features, (hidden_state, cell_state))
                #hidden_state__layer_2, cell_state_layer_2 = self.lstm_cell_layer_2(hidden_state, (hidden_
state_layer_2, cell_state_layer_2))


            # for the 2nd+ time step, using teacher forcer
            else:
                hidden_state, cell_state = self.lstm_cell(captions_embed[:, t, :], (hidden_state, cell_sta
te))
                #hidden_state__layer_2, cell_state_layer_2 = self.lstm_cell_layer_2(hidden_state, (hidden_
state_layer_2, cell_state_layer_2))


            # output of the attention mechanism
            out = self.fully_connected(hidden_state)

            # build the output tensor
            outputs[:, t, :] = out


        return F.log_softmax(outputs, dim = -1)

class DecoderGRU(nn.Module):
    def __init__(self, embed_size, hidden_size, vocab_size, torch_embedding = False,num_layers=1):
        super(DecoderGRU, self).__init__()

        # define the properties
        self.embed_size = embed_size
        self.hidden_size = hidden_size
        self.vocab_size = vocab_size

        self.embed = nn.Embedding(num_embeddings=self.vocab_size, embedding_dim=self.embed_size)

        if torch_embedding:
          self.embedding_glove6B = torch.load('embedding_glove6B')
          self.embed.weight.data.copy_(self.embedding_glove6B.vectors)
          self.embed.weight.requires_grad = False

        else:

          pretrainedEmbeds = np.loadtxt('embeds300.txt', delimiter=',')
          self.embed.weight.data.copy_(torch.from_numpy(pretrainedEmbeds))
          self.embed.weight.requires_grad = False
```

```
# lstm cell
#self.lstm_cell = nn.LSTMCell(input_size = embed_size, hidden_size = hidden_size)
self.GRU_cell = nn.GRUCell(input_size = embed_size, hidden_size = hidden_size)

#if num_layers==2:
#self.lstm_cell_layer_2 = nn.LSTMCell(input_size = hidden_size, hidden_size = hidden_size)

# output fully connected layer
self.fully_connected = nn.Linear(in_features = hidden_size, out_features= vocab_size)

def forward(self, features, captions):

# batch size
batch_size = features.size(0)

# init the hidden and cell states to zeros
hidden_state = torch.zeros((batch_size, self.hidden_size)).to(device)
cell_state = torch.zeros((batch_size, self.hidden_size)).to(device)
#hidden_state_layer_2 = torch.zeros((batch_size, self.hidden_size)).to(device)
#cell_state_layer_2 = torch.zeros((batch_size, self.hidden_size)).to(device)

# define the output tensor placeholder
outputs = torch.empty((batch_size, captions.size(1), self.vocab_size)).to(device)

# embed the captions
captions_embed = self.embed(captions)

# pass the caption word by word
for t in range(captions.size(1)):

    # for the first time step the input is the feature vector
    if t == 0:
        hidden_state = self.GRU_cell(features, hidden_state)



    # for the 2nd+ time step, using teacher forcer
    else:
        hidden_state = self.GRU_cell(captions_embed[:, t, :], hidden_state)



    # output of the attention mechanism
    out = self.fully_connected(hidden_state)
```

```
            # build the output tensor
            outputs[:, t, :] = out


        return F.log_softmax(outputs, dim = -1)

class Attention(nn.Module):
    def __init__(self, embed_size, hidden_size, vocab_size, torch_embedding = False,num_layers=1):
        super(Attention, self).__init__()

        # define the properties
        self.embed_size = embed_size
        self.hidden_size = hidden_size
        self.vocab_size = vocab_size

        self.embed = nn.Embedding(num_embeddings=self.vocab_size, embedding_dim=self.embed_size)

        if torch_embedding:
          self.embedding_glove6B = torch.load('embedding_glove6B')
          self.embed.weight.data.copy_(self.embedding_glove6B.vectors)
          self.embed.weight.requires_grad = False

        else:

          pretrainedEmbeds = np.loadtxt('embeds300.txt', delimiter=',')
          self.embed.weight.data.copy_(torch.from_numpy(pretrainedEmbeds))
          self.embed.weight.requires_grad = False

        self.lstm = nn.LSTM(input_size = embed_size,hidden_size = hidden_size,
                            num_layers = num_layers, batch_first = True)

        # output fully connected layer
        self.fully_connected = nn.Linear(in_features = hidden_size, out_features= vocab_size)

        self.relu = nn.ReLU()

        self.softmax = nn.Softmax(dim=1)

    def forward(self, features, captions):


        embed = self.embed(captions)
        embed = torch.cat((features, embed), dim = 1)
        lstm_outputs, hidden = self.lstm(embed)
        out = self.fully_connected(lstm_outputs)
        alpha = self.softmax(att)
```

```
        attention_weighted_encoding = (features * alpha.unsqueeze(2)).sum(dim=1)  # (batch_size, encoder_d
im)

        return attention_weighted_encoding,F.log_softmax(output, dim = -1)

LATENT_SPACE = 300
feature_extract = True
ResNet,in_ftrs,input_size = PreTrainedModels(LATENT_SPACE).ResNet()
encoder_model = EncoderCNN(in_ftrs,ResNet)

# We will be working with GPU:
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

print(device)

preprocess_train = transforms.Compose([RandomHorizontalFlip(),
                                 RandomVerticalFlip(),
                                 Resize(256),
                                 RandomResizedCrop(224),
                                 ToTensor(),
                                 Normalize()
                               ])

tsfm_train = ImageCaptionData(image_urls = train_url,
                            img_inds = train_inds,
                            captions = train_caps,
                            sentence_lens = sentence_lens_train,
                            transform = preprocess_train
                            )



preprocess_val = transforms.Compose([Resize(224),
                                    ToTensor(),
                                    Normalize()
                               ])

tsfm_val = ImageCaptionData(image_urls = train_url,
                          img_inds = val_inds,
                          captions = val_caps,
                          sentence_lens = sentence_lens_val,
                          transform = preprocess_val)

num_GPU = torch.cuda.device_count()

print(num_GPU)
```

```
BATCH_SIZE = 64

data_transformed = {'train' : tsfm_train, 'val' : tsfm_val}

dataloader = {x : DataLoader(data_transformed[x], batch_size = BATCH_SIZE,
                        shuffle = True, num_workers = 4 * num_GPU , pin_memory = True)
              for x in ['train','val']}

dataset_sizes = {x: len(data_transformed[x]) for x in ['train', 'val']}

# Running on multiple GPU's and distributed processing:
if torch.cuda.device_count() > 1:
  print("Let's use", torch.cuda.device_count(), "GPUs!")
  encoder_model = nn.DataParallel(encoder_model)

encoder_model = encoder_model.to(device)

params_to_update = encoder_model.parameters()

# Just check the uptadable parameters:
if feature_extract:
  params_to_update = []
  for name,param in encoder_model.named_parameters():
    if param.requires_grad == True:
      params_to_update.append(param)
      print("\t",name)
else:
  for name,param in encoder_model.named_parameters():
    if param.requires_grad == True:
      print("\t",name)

# Observe that all parameters are being optimized
encoder_optimizer = optim.Adam(params_to_update,lr = 4e-3)

# Setup the loss function: (may not me used directly)
criterion = nn.CrossEntropyLoss().to(device)



wordC = pd.read_hdf("/content/gdrive/My Drive/Data/eee443_project_dataset_train.h5", 'word_code')
wordC = wordC.to_dict('split')
wordDict = dict(zip(wordC['data'][0], wordC['columns']))

VOCAB_SIZE = len(wordC['data'][0])
VOCAB_SIZE = 1004
EMBED_DIM = 300
```

```
HIDDEN_DIM = 256

decoder_model = DecoderGRU(EMBED_DIM,HIDDEN_DIM,VOCAB_SIZE)

decoder_optimizer = optim.Adam(decoder_model.parameters(),lr = 1e-3)

# Running on multiple GPU's and distributed processing:
if torch.cuda.device_count() > 1:
  print("Let's use", torch.cuda.device_count(), "GPUs!")
  decoder_model = nn.DataParallel(decoder_model)

decoder_model = decoder_model.to(device)

# Vanilla schedulers:
decoder_scheduler_LambdaLR = torch.optim.lr_scheduler.LambdaLR(decoder_optimizer, lr_lambda = lambda epoch
: 0.9 ** epoch)
encoder_scheduler_LambdaLR = torch.optim.lr_scheduler.LambdaLR(encoder_optimizer, lr_lambda= lambda epoch:
 0.9 ** epoch)


# Monitoring val loss:
encoder_scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(encoder_optimizer, 'max')
decoder_scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(decoder_optimizer, 'max')

decoder = decoder_model
encoder = encoder_model
dataloaders = dataloader
optimizers = [decoder_optimizer,encoder_optimizer]

# The next two functions are part of some other deep learning frameworks, but PyTorch
# has not yet implemented them. We can find some commonly-used open source worked arounds
# after searching around a bit: https://gist.github.com/jihunchoi/f1434a77df9db1bb337417854b398df1.
def _sequence_mask(sequence_length, max_len=None):
    if max_len is None:
        max_len = sequence_length.data.max()
    batch_size = sequence_length.size(0)
    seq_range = torch.arange(0, max_len).long()
    seq_range_expand = seq_range.unsqueeze(0).expand(batch_size, max_len)
    seq_range_expand = Variable(seq_range_expand)
    if sequence_length.is_cuda:
        seq_range_expand = seq_range_expand.cuda()
    seq_length_expand = (sequence_length.unsqueeze(1)
                         .expand_as(seq_range_expand))
    return seq_range_expand < seq_length_expand
```

```
def compute_loss(logits, target, length):
    """
    Args:
        logits: A Variable containing a FloatTensor of size
            (batch, max_len, num_classes) which contains the
            unnormalized probability for each class.
        target: A Variable containing a LongTensor of size
            (batch, max_len) which contains the index of the true
            class for each corresponding step.
        length: A Variable containing a LongTensor of size (batch,)
            which contains the length of each data in a batch.

    Returns:
        loss: An average loss value masked by the length.
    """
    # logits_flat: (batch * max_len, num_classes)
    logits_flat = logits.view(-1, logits.size(-1))
    # log_probs_flat: (batch * max_len, num_classes)
    log_probs_flat = logits_flat
    # target_flat: (batch * max_len, 1)
    target_flat = target.view(-1, 1)
    # losses_flat: (batch * max_len, 1)
    losses_flat = -torch.gather(log_probs_flat, dim=1, index=target_flat)
    # losses: (batch, max_len)
    losses = losses_flat.view(*target.size())
    # mask: (batch, max_len)
    mask = _sequence_mask(sequence_length=length, max_len=target.size(1))
    losses = losses * mask.float()
    loss = losses.sum() / length.float().sum()
    return loss

state_lr = 1e-3
decay_lr = 0.99

since = time.time()
PATH = 'models'

PRINT_EVERY = 50

# Number of epochs:
num_epochs = 3

# To keep track history:
val_loss_history = []
train_loss_history = []
val_acc_history = []
```

```
train_acc_history = []

best_decoder_wts = copy.deepcopy(decoder.state_dict())
best_encoder_wts = copy.deepcopy(encoder.state_dict())
best_acc = 0.0

if os.path.exists(os.path.join(data_dir,PATH,'SON_GRU') + '.pth'):
  print('Woring weapons ...')
  checkpoint = torch.load(os.path.join(data_dir,PATH,'1.43_GRU') + '.pth')
  decoder.load_state_dict(checkpoint['Decoder_state_dict'])
  encoder.load_state_dict(checkpoint['Encoder_state_dict'])
  decoder_optimizer.load_state_dict(checkpoint['Decoder_optim_state_dict'])
  encoder_optimizer.load_state_dict(checkpoint['Encoder_optim_state_dict'])
  decoder.to(device)
  encoder.to(device)

for epoch in range(num_epochs):
    print('Epoch {}/{}'.format(epoch + 1, num_epochs))
    print('_' * 10)

    if epoch != 0:

      if not os.path.exists(os.path.join(data_dir,PATH,str(epoch)) + ".pth"):

        checkpoint = {'Decoder_state_dict' : decoder.state_dict(),
                      'Encoder_state_dict' : encoder.state_dict(),
                      'Decoder_optim_state_dict' : decoder_optimizer.state_dict(),
                      'Encoder_optim_state_dict' : encoder_optimizer.state_dict(),
                      }

        torch.save(checkpoint, os.path.join(data_dir,PATH,str(epoch)) + ".pth")

        print(f'Model : {epoch} is succesfully saved')


    # Each epoch has a training and validation phase
    for phase in ['train','val']:
        if phase == 'train':
            decoder.train()  # Set model to training mode
            encoder.train()
        else:
            decoder.eval()  # Set model to training mode
            encoder.eval()

        running_loss = 0.0
        running_corrects = 0
```

```
        # Iterate over data.
        for i,(sample,caption_len) in enumerate(dataloaders[phase]):

            images,captions = sample['image'],sample['caption']


            captions_target = captions[:, 1:].long().to(device)
            captions_train = captions[:, :captions.shape[1]-1].long().to(device)

            # Move batch of images and captions to GPU if CUDA is available.
            images = images.to(device)

            if phase == 'train':

              # Safer approach of zero gradients in our case
              decoder.zero_grad()
              encoder.zero_grad()

              #for optim in optimizers:
                # zero the parameter gradients
                #optim.zero_grad()

              # Pass the inputs through the CNN-RNN model.
              features = encoder(images)
              outputs = decoder(features, captions_train)


              loss = compute_loss(outputs.contiguous(),
                                  captions_target.contiguous(),
                                  Variable(caption_len.long()).to(device))

              #loss = criterion(outputs.view(-1, VOCAB_SIZE), captions_target.contiguous().view(-1))

              if i % PRINT_EVERY == 0:
                print('train', loss.item())
                print('train acc : ' , (outputs.argmax(2) == captions_target.data).sum().item()/ (captions
_target.size(0) * captions_target.size(1)) * 100)


              # backward + optimize only if in training phase
              loss.backward()

              torch.nn.utils.clip_grad_norm(decoder.parameters(),10.0)
              torch.nn.utils.clip_grad_norm(encoder.parameters(),10.0)
```

```
            for optim in optimizers:
              # zero the parameter gradients
              optim.step()

          if phase == 'val':

            with torch.no_grad():
              # Pass the inputs through the CNN-RNN model.
              features = encoder(images)
              outputs = decoder(features, captions_train)

              # Calculate the batch loss
              loss = compute_loss(outputs.contiguous(),
                                  captions_target.contiguous(),
                                  Variable(caption_len.long()).to(device))

              if i % PRINT_EVERY == 0:
                print('Val',loss.item())
                print('Val correct' , (outputs.argmax(2) == captions_target.data).sum().item()/ (caption
s_target.size(0) * captions_target.size(1)) * 100)



          # statistics
          running_loss += loss.item() * images.size(0)
          running_corrects += (outputs.argmax(2) == captions_target.data).sum().item()/ (captions_target
.size(0) * captions_target.size(1))
          #print('running_corrects',running_corrects)

          if i % 1000 == 0:

            for optim in optimizers:
              for param_group in optim.param_groups:
                param_group['lr'] = state_lr

            state_lr *= decay_lr
            print('state_lr ',state_lr)


      epoch_loss = running_loss / len(dataloaders[phase].dataset)
      epoch_acc = (running_corrects / len(dataloaders[phase].dataset)) * 100

      decoder_scheduler_LambdaLR.step()
      encoder_scheduler_LambdaLR.step()
      decoder_scheduler.step(epoch_acc)
      encoder_scheduler.step(epoch_acc)
```

```
        if phase == 'val':
          val_loss_history.append(epoch_loss)
          val_acc_history.append(epoch_acc)

        if phase == 'train':
          train_loss_history.append(epoch_loss)
          train_acc_history.append(epoch_acc)


        print('{} Loss: {:.4f} '.format(phase, epoch_loss))
        print('{} Acc: {:.4f} '.format(phase, epoch_acc))

        # deep copy the model
        if phase == 'val' and epoch_acc > best_acc:
            best_acc = epoch_acc
            best_decoder_wts = copy.deepcopy(decoder.state_dict())
            best_encoder_wts = copy.deepcopy(encoder.state_dict())



    print()

time_elapsed = time.time() - since
print('Training complete in {:.0f}m {:.0f}s'.format(time_elapsed // 60, time_elapsed % 60))


# load best model weights
#decoder.load_state_dict(best_decoder_wts)
#encoder.load_state_dict(best_encoder_wts)


torch.save({'Decoder_state_dict' : decoder.state_dict(),
            'Encoder_state_dict' : encoder.state_dict(),
            'Decoder_optim_state_dict' : decoder_optimizer.state_dict(),
            'Encoder_optim_state_dict' : encoder_optimizer.state_dict(),
              },
            os.path.join(data_dir,PATH,'final_model_GRU') + ".pth")

LSTM_path = 'best_model_sensible'
GRU_path = '2GRU'
Best_GRU_path = 'SON_GRU'

batch,_ = next(iter(dataloader['val']))
if os.path.exists(os.path.join(data_dir,'models',GRU_path) + '.pth'):
  print('Working...')
```

```
  checkpoint = torch.load(os.path.join(data_dir,'models',GRU_path) + '.pth', map_location=torch.device('cp
u'))
  decoder.load_state_dict(checkpoint['Decoder_state_dict'])
  encoder.load_state_dict(checkpoint['Encoder_state_dict'])
  decoder.to(device)
  encoder.to(device)

with torch.no_grad():
  decoder.eval()
  encoder.eval()
  images,captions = batch['image'],batch['caption']


  captions_target = captions[:, 1:].long().to(device)
  captions_train = captions[:, :-1].long().to(device)

  # Move batch of images and captions to GPU if CUDA is available.
  images = images.to(device)

  features = encoder(images)
  outputs = decoder(features, captions_train)

  greedy_outputs = outputs.argmax(2)

words = pd.read_hdf("/content/gdrive/My Drive/Data/eee443_project_dataset_train.h5", 'word_code')
words = words.to_dict('split')
wordDict = dict(zip(words['data'][0], words['columns']))

def generate_caption_argmax_search(images,greedy_outputs,captions_target, index = None, METEOR = False, n_
gram_bleus = False,cumulative_bleus = False):
  if index is None:
    index = np.random.randint(BATCH_SIZE)


  caption = [wordDict[i] for i in captions_target[index].cpu().detach().numpy()]

  captionOut = [wordDict[i] for i in greedy_outputs.cpu().detach().numpy()[index]]

  try:
    hypothesis = captionOut[:captionOut.index('x_END_')]
  except ValueError:
    hypothesis = captionOut[:captionOut.index('x_NULL_')]

  reference = caption[:caption.index('x_END_')]

  BLEUscore = sentence_bleu([reference], hypothesis)
```

```
  if METEOR:
    METEOR = meteor_score([reference], hypothesis)
    print(METEOR)

  plt.figure(figsize=(8,6))

  plt.imshow(images[index].permute(1,2,0).cpu())
  plt.title('Reference caption:  ' + ' '.join(reference))

  plt.xlabel('Generated Caption: ' + ' '.join(hypothesis) + '\n BLEU Score:' + str(BLEUscore))

  if n_gram_bleus:
    print('BLEU1 Score: %f' % sentence_bleu(reference, hypothesis, weights=(1, 0, 0, 0)))
    print('BLEU2 Score: %f' % sentence_bleu(reference, hypothesis, weights=(0, 1, 0, 0)))
    print('BLEU3 Score: %f' % sentence_bleu(reference, hypothesis, weights=(0, 0, 1, 0)))
    print('BLEU4 Score: %f' % sentence_bleu(reference, hypothesis, weights=(0, 0, 0, 1)))


  if cumulative_bleus:
    print('Cumulative BLEU : %f' % sentence_bleu(reference, hypothesis, weights=(0.25, 0.25, 0.25, 0.25)))

generate_caption_argmax_search(images,greedy_outputs,captions_target, index = 32)

# beam search
def beam_search_decoder(data, k):
  sequences = [[list(), 0.0]]
  # walk over each step in sequence
  for row in data:
    all_candidates = list()
    # expand each current candidate
    for i in range(len(sequences)):
      seq, score = sequences[i]
      for j in range(len(row)):
        candidate = [seq + [j], score - np.log(row[j] + 50)]
        all_candidates.append(candidate)
    # order all candidates by score
    ordered = sorted(all_candidates, key=lambda tup:tup[1])
    # select k best
    sequences = ordered[:k]
  return sequences

def beam_search_decoder_alternative(predictions, top_k = 3):

    #start with an empty sequence with zero score
    output_sequences = [([], 0)]
```

```
    #looping through all the predictions
    for token_probs in predictions:
        new_sequences = []

        #append new tokens to old sequences and re-score
        for old_seq, old_score in output_sequences:
            for char_index in range(len(token_probs)):
                new_seq = old_seq + [char_index]
                #considering log-likelihood for scoring
                new_score = old_score + np.log(token_probs[char_index] + 50)
                new_sequences.append((new_seq, new_score))

        #sort all new sequences in the de-creasing order of their score
        output_sequences = sorted(new_sequences, key = lambda val: val[1], reverse = True)

        #select top-k based on score
        # *Note- best sequence is with the highest score
        output_sequences = output_sequences[:top_k]

    return output_sequences

def generate_caption_beam_search(images,raw_outputs,captions_target,
                                 index = None, beam_size = 3,
                                 METEOR = False, n_gram_bleus = False, cumulative_bleus = False,
                                 print_bleu = False):

  if index is None:
    index = np.random.randint(BATCH_SIZE)

  plt.figure(figsize=(8,6))

  caption = [wordDict[i] for i in captions_target[index].cpu().detach().numpy()]




  beam_decoded = beam_search_decoder(raw_outputs[index].cpu().detach().numpy(),k = beam_size)

  captionOut = [seq[0] for seq in beam_decoded]
  captionOutList = []
  for cap_inds in captionOut:
    captionOutList.append([wordDict[i] for i in cap_inds])

  hypothesisList = []
```

```
for captions in captionOutList:
  try:
    hypothesis = captions[:captions.index('x_END_')]
    hypothesisList.append(hypothesis)
  except ValueError:
    hypothesis = captions[:captions.index('x_NULL_')]
    hypothesisList.append(hypothesis)

reference = caption[:caption.index('x_END_')]




if print_bleu:
  BLEUscore = sentence_bleu([reference], hypothesis)
  print('BLEUscore :' + str(BLEUscore))



if METEOR:
  METEOR = meteor_score([reference], hypothesis)
  print(METEOR)

plt.figure()

plt.imshow(images[index].permute(1,2,0).cpu())
plt.title('Reference caption:  ' + ' '.join(reference))

if beam_size == 3:
  plt.xlabel('Generated Captions for beam size = '  + str(beam_size) + '\n' +
          ' '.join(hypothesisList[0]) + '\n' +
          ' '.join(hypothesisList[1]) + '\n' +
          ' '.join(hypothesisList[2]) + '\n'
          )

if beam_size == 5:
  plt.xlabel('Generated Captions for beam size= '  + str(beam_size) + '\n' +
          ' '.join(hypothesisList[0]) + '\n' +
          ' '.join(hypothesisList[1]) + '\n' +
          ' '.join(hypothesisList[2]) + '\n' +
          ' '.join(hypothesisList[3]) + '\n' +
          ' '.join(hypothesisList[4]) + '\n'
          )




if beam_size == 7:
  plt.xlabel('Generated Captions for beam size = '  + str(beam_size) + '\n' +
```

```
            ' '.join(hypothesisList[0]) + '\n' +
            ' '.join(hypothesisList[1]) + '\n' +
            ' '.join(hypothesisList[2]) + '\n' +
            ' '.join(hypothesisList[3]) + '\n' +
            ' '.join(hypothesisList[4]) + '\n' +
            ' '.join(hypothesisList[5]) + '\n' +
            ' '.join(hypothesisList[6]) + '\n'
            )
  if beam_size == 9:
    plt.xlabel('Generated Captions for beam size= '  + str(beam_size) + '\n' +
            ' '.join(hypothesisList[0]) + '\n' +
            ' '.join(hypothesisList[1]) + '\n' +
            ' '.join(hypothesisList[2]) + '\n' +
            ' '.join(hypothesisList[3]) + '\n' +
            ' '.join(hypothesisList[4]) + '\n' +
            ' '.join(hypothesisList[5]) + '\n' +
            ' '.join(hypothesisList[6]) + '\n' +
            ' '.join(hypothesisList[7]) + '\n' +
            ' '.join(hypothesisList[8]) + '\n'
            )

  if n_gram_bleus:
    print('BLEU1 Score: %f' % sentence_bleu(reference, hypothesis, weights=(1, 0, 0, 0)))
    print('BLEU2 Score: %f' % sentence_bleu(reference, hypothesis, weights=(0, 1, 0, 0)))
    print('BLEU3 Score: %f' % sentence_bleu(reference, hypothesis, weights=(0, 0, 1, 0)))
    print('BLEU4 Score: %f' % sentence_bleu(reference, hypothesis, weights=(0, 0, 0, 1)))



  if cumulative_bleus:
    print('Cumulative BLEU : %f' % sentence_bleu(reference, hypothesis, weights=(0.25, 0.25, 0.25, 0.25)))

generate_caption_beam_search(images,outputs,captions_target,index = 5,beam_size=3)
generate_caption_argmax_search(images,greedy_outputs,captions_target, index = 5)

# Getting the data:
with h5py.File(colap_path_test,'r') as f:
# Names variable contains the names of training and testing file
    names = list(f.keys())
    test_caps = f[names[0]][()]
    test_imid = np.array(f[names[1]][()])
    train_imid -=1
    test_url = np.array(f[names[3]][()])

test_cap_Str = {}
for i in range(len(test_caps)):
  test_cap_Str[i] = [element for element in test_caps[i] if element != 0]
```

```
sentence_lens_test = [len(ele) for ele in test_cap_Str.values()]

preprocess_test = transforms.Compose([Resize(224),
                                    ToTensor(),
                                    #Normalize()

                               ])

tsfm_test = ImageCaptionData(image_urls = test_url,
                          img_inds = test_imid,
                          captions = test_caps,
                          sentence_lens = sentence_lens_test,
                          transform = preprocess_test)

dataloader_test = DataLoader(tsfm_test, batch_size = BATCH_SIZE,
                        shuffle = True, num_workers = 4 * num_GPU , pin_memory = True)

batch,_ = next(iter(dataloader_test)
if os.path.exists(os.path.join(data_dir,'models',GRU_path) + '.pth'):
  print('Working...')
  checkpoint = torch.load(os.path.join(data_dir,'models',GRU_path) + '.pth', map_location=torch.device('cp
u'))
  decoder.load_state_dict(checkpoint['Decoder_state_dict'])
  encoder.load_state_dict(checkpoint['Encoder_state_dict'])
  decoder.to(device)
  encoder.to(device)

with torch.no_grad():
  decoder.eval()
  encoder.eval()
  images,captions = batch['image'],batch['caption']


  captions_target = captions[:, 1:].long().to(device)
  captions_train = captions[:, :-1].long().to(device)

  # Move batch of images and captions to GPU if CUDA is available.
  images = images.to(device)

  features = encoder(images)
  outputs = decoder(features, captions_train)

  greedy_outputs = outputs.argmax(2)

generate_caption_beam_search(images,outputs,captions_target,index = 5,beam_size=3)
generate_caption_argmax_search(images,greedy_outputs,captions_target, index = 5)
```

REFERENCES

[1] X. Rong, word2vec Parameter Learning Explained.

[2] "Transfer Learning," *CS231n Convolutional Neural Networks for Visual Recognition*. [Online]. Available: https://cs231n.github.io/transfer-learning/. [Accessed: 08-Jan-2021].

[3] "Finetuning Torchvision Models¶," *Finetuning Torchvision Models - PyTorch Tutorials 1.2.0 documentation*. [Online]. Available: https://pytorch.org/tutorials/beginner/finetuning_torchvision_models_tutorial.html. [Accessed: 08-Jan-2021].

[4] Pulkit SharmaMy research interests lies in the field of Machine Learning and Deep Learning. Possess an enthusiasm for learning new skills and technologies., "Transfer Learning: Transfer Learning in Pytorch," *Analytics Vidhya*, 08-May-2020. [Online]. Available: https://www.analyticsvidhya.com/blog/2019/10/how-to-master-transfer-learning-using-pytorch/?utm_source=blog&utm_medium=building-image-classification-models-cnn-pytorch. [Accessed: 08-Jan-2021].

[5] P. Radhakrishnan, "Image Captioning in Deep Learning," *Medium*, 10-Oct-2017. [Online]. Available: https://towardsdatascience.com/image-captioning-in-deep-learning-9cd23fb4d8d2. [Accessed: 08-Jan-2021].

[6] "Tutorial on Attention-based Models (Part 1)," *Karan Taneja*, 02-Jun-2018. [Online]. Available: https://krntneja.github.io/posts/2018/attention-based-models-1#:~:text=Attention%2Dbased%20models%20belong%20to,in%20general%2C%20of%20different%20lengths. [Accessed: 08-Jan-2021].

[7] S. Ulyanin, "Captioning Images with PyTorch," *Medium*, 23-Feb-2019. [Online]. Available: https://medium.com/@stepanulyanin/captioning-images-with-pytorch-bc592e5fd1a3. [Accessed: 08-Jan-2021].

[8] K. Kshirsagar, "Automatic Image Captioning with CNN & RNN," *Medium*, 21-Jan-2020. [Online]. Available: https://towardsdatascience.com/automatic-image-captioning-with-cnn-rnn-aae3cd442d83. [Accessed: 08-Jan-2021].

[9] "Sequence-to-Sequence Modeling with nn.Transformer and TorchText¶," *to*. [Online]. Available: https://pytorch.org/tutorials/beginner/transformer_tutorial.html. [Accessed: 08-Jan-2021].

[10] "NLP From Scratch: Translation with a Sequence to Sequence Network and Attention¶," *NLP From Scratch: Translation with a Sequence to Sequence Network and Attention - PyTorch Tutorials 1.7.1 documentation*. [Online]. Available: https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html. [Accessed: 08-Jan-2021].

[11] "Sequence Models and Long-Short Term Memory Networks¶," *Sequence Models and Long-Short Term Memory Networks - PyTorch Tutorials 1.7.1 documentation*. [Online]. Available: https://pytorch.org/tutorials/beginner/nlp/sequence_models_tutorial.html. [Accessed: 08-Jan-2021].

[12] Kelvin Xu, Jimmy Lei Ba, Ryan Kiros, Kyunghyun Cho, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio, Show, Attend and Tell: Neural Image Caption Generation with Visual Attention, Apr. 2016.

[13] "Word Embeddings: Encoding Lexical Semantics¶," *Word Embeddings: Encoding Lexical Semantics - PyTorch Tutorials 1.7.1 documentation*. [Online]. Available: https://pytorch.org/tutorials/beginner/nlp/word_embeddings_tutorial.html. [Accessed: 08-Jan-2021].

[14] Sgrvinod, "sgrvinod/a-PyTorch-Tutorial-to-Image-Captioning," *GitHub*. [Online]. Available: https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Image-Captioning. [Accessed: 08-Jan-2021].

[15] J. Brownlee, "How to Implement a Beam Search Decoder for Natural Language Processing," *Machine Learning Mastery*, 03-Jun-2020. [Online]. Available: https://machinelearningmastery.com/beam-search-decoder-natural-language-processing/. [Accessed: 08-Jan-2021].

[16] R. Khandelwal, "An intuitive explanation of Beam search," *Medium*, 03-Feb-2020. [Online]. Available: https://towardsdatascience.com/an-intuitive-explanation-of-beam-search-9b1d744e7a0f. [Accessed: 08-Jan-2021].

[17] S. Sarkar, "Image Captioning using Attention Mechanism," *Medium*, 07-Mar-2020. [Online]. Available: https://medium.com/swlh/image-captioning-using-attention-mechanism-f3d7fc96eb0e. [Accessed: 08-Jan-2021].

[18] Aayush Bajaj, Avantari et al., Machine Learning Engineer,"Understanding Gradient Clipping", 2020