OptimuZ Prime

Ahamed Kannanari
akannanari3@gatech.edu

Alexander Song
asong49@gatech.edu

Deepu Thomas
dthomas334@gatech.edu

Sachin Chandrasekhar
schandra63@gatech.edu

Abstract

The paper "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention" [22] introduced attention layers to improve the Image Captioning problem. In this project, we tried to improve the performance of the architecture proposed in the paper using various newer techniques such as transformers, newer embeddings, newer convolutional neural network architectures, and beam search during training.

1. Introduction/Background/Motivation

We've tried to build a model that generates a caption that describes an input image with the newest approaches. We've tried to solve the low precision problem of the model presented in the "Show, Attend and Tell" [22] paper. Although an impressive approach, some output captions can contain considerable amount of imprecise vocabulary that are important to understand the caption. As this paper was published in 2015, there has been numerous advancements in computer vision and natural language processing approaches that are more effective than those used in this paper. Therefore, we've decided to adopt and explore these new approaches to improve the precision of these captions.

Current proven approaches include leveraging the Convolutional Neural Network [9] and the Recurrent Neural Network with attention mechanism. Even though this is a remarkable approach used by the paper, current caption generation results still have limited precision.

If successfully improved, the image captioning task has immense potential in benefiting visually or speech impaired people or serving as a secondary assistive communication device for anyone. For example, this could help a distracted driver or be applied to autonomous driving.

We used the Flickr8k data set which has 8000 free unique images from Flickr and corresponding captions. We used 6:1:1 split for training, validation and test sets. Each image is mapped to 5 sentences that describe salient objects in the image. The baseline paper used both Coco data set as well as Flickr8k and Flick30k data sets. We initially tried to use Coco, but however required huge computing power and time to run the experiments and hence and decided to use Flickr8k data set which had reasonable size to run experiments which suited the scope and timelines of this particular project.

2. Approach

"Show, Attend and Tell" [22] paper introduces an attention based decoder model that automatically learns to describe the content of image. This is a very interesting topic and is heavily researched, our goal was to study each component in the architecture defined by the authors meticulously, explore areas of improvement and try to improve or change the existing components by using modern and better components. The baseline architecture uses a Sequence to Sequence [18] architecture with Encoder and Decoder [2] layers. The encoder layer uses a Convolutional Neural Network to extract image features and it is then fed to a decoder layer, which uses pre-trained Glove embedding [12] along with Attention [13] layer to generate the captions using Long Short Term Memory [5].

We created four work streams that explore different potential improvement areas. We attempted to improve the baseline performance by attempting the below changes:

1. First opportunity was to experiment replacing the simple CNN encoder in the original paper with the Transformer from the "Attention Is All You Need" [19] paper. We wanted to add more details to the encoded image in addition to the features extracted by CNN encoders. Hence we propose the usage of Transformer encoder. Transformer encoder consists of a pre-trained CNN layer, positional encoding layer, multi-head attention layers, feedforward layers and dropout layers. This way we are able to add positional information and multi-head attention information to the baseline decoder. In order to make the Transformer encoder attach to the baseline code, the embedding layer of the transformer encoder layer is replaced by a linear

layer. This is because we have image information as the input to transformer and not the vocabulary as in traditional transformers. The output of CNN layer is (batch size * 2048, encoded image size * encoded image size). Encoded image size is 14, which is the output of Adaptive pooling layer in the baseline code. We then feed this tensor to the embedding layer which is just a linear layer followed by ReLU activation. It is then forwarded to multi-head attention and then to the feedForward layer and to further droput layers. The output of the transformer encoder is then fed to the baseline decoder.

Modified network architecture we used is shown in Figure 1 and modified Transformer architecture we used is shown in Figure 2.

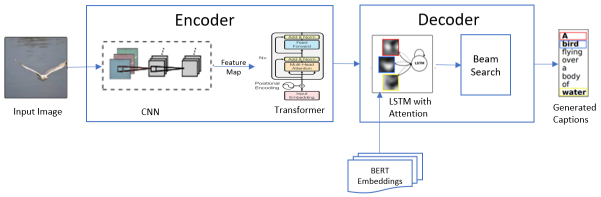

Figure 1. Our Modified Architecture

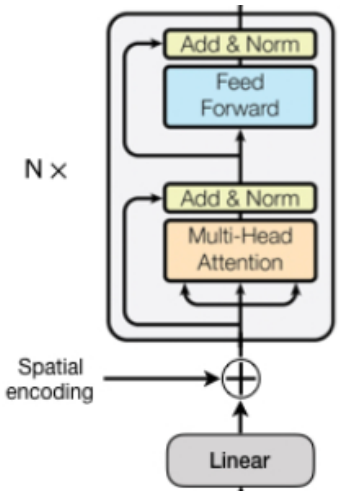

Figure 2. Our Modified Transformer Architecture

2. Second opportunity was to use a better word embedding for captions than what was used in the baseline. Hence we wanted to replace pre-trained word embedding in the decoder with various flavors of Google BERT [3] word embedding. BERT stands for Bidirectional Encoder Representations from Transformers and provides contextualized word embeddings. We wanted to experiment with different variations of BERT model such as 'bert-base' that has 12-layer, 768-hidden, 12-heads and 110M parameters, and 'bert-large', which has 24-layer, 1024-hidden, 16-heads and 336M parameters. In addition, we also wanted to experiment with reduced vocabulary which filters out the least common words in the data set while improving efficiency of the model.

3. Third opportunity we identified was using newer and larger CNN architectures. Because the output from CNN conditions the downstream networks, we experimented different pre-trained CNN architectures such as VGG [16], ResNet [7], and ResNeXt [21] to see if newer and larger architecture helps improve the accuracy of captions. Given the complex features that are present in Flickr images, the intuition is that deeper networks can capture and discriminate finer details and improve the conditioning of the recurrent neural network. Then, the quality of inference can improve as a result. Additionally applying transfer learning or fine tuning could further improve results. The original paper used VGG-16 and we tried out deeper architectures from PyTorch's torchvision module that are pre-trained on ImageNet. Specifically, we experimented with ResNet-101, ResNeXt-101-32x8d, which has 101 layers, 32 groups, and group widths C of 8. Then, the benefit of newer architectures are measured by comparing their accuracy against the baseline accuracy. The ResNext architecture clearly performed better than the baseline VGG-16 but not by a large margin but slightly better than the ResNet.

4. Finally, we wanted to use beam search technique during training to exploit the joint probabilities of the sequence to select the next word instead of the greedy approach in the baseline. [4] In the baseline implementation of the decoder, we could see that a linear layer was used to transform the output into a score for the corresponding word from the vocabulary. After the score is assigned, a greedy search algorithm is used which would select the word with the highest score in order to predict the next word. Even though this approach often appears to suffice, this is not an optimal approach as at the time of scoring for the first word, the model has no idea about the sequence or even the next word and thus might have scored a sub optimal word as the first word. Now as the next word depends on the first word, you might end up having a sub-optimal sequence. This is like scoring each word as soon as they reach the linear layer without being aware of the context. This is where beam search shines where the sequence with the highest overall score from a list of sequences is selected. Since beam search is trying to find the most optimal sequence, it is helpful for natural language problems. However, we have observed from various papers and publications that almost all of them seem to have implemented the greedy decoding method and have used beam search only for evaluation of model or during caption generation.

Even though the greedy approach seems to be working for language models, the metric used during the training phase does not align with the one used during evaluation

phase. Thus, we wanted the model to have the same metric sequence level score (BLEU [11] during training rather than per word level score based on cross entropy. Therefore, we wanted to try out and implement beam search during training and see how big of a change it can produce on the performance of the model. Now along with implementing beam search during training, we wanted to implement them using the fastai API. Also since implementing beam search during training was going to be computationally expensive, we explored multiple options within fastai like EarlyStoppingCallback which stops training when a given metric such as BLEU-4 score is not improving after a set number of epochs.

By performing the above 4 changes, we attempted to improve the baseline BLUE score of the model. By experimenting with different CNN architectures we hoped to get the best image features to feed to Encoder. Since the transformer encoder has positional encoding and Multi-head Attention, there are more details to the encoded image rather than just the CNN output. The BERT embedding would provide additional contextual information and improved word vectors to Decoder. And finally, we expected the Beam Search to improve the quality of the generated captions rather than the default greedy algorithm.

2.1. Challenges Faced

The goal of this project is to improve the architecture presented in "Show, Attend and Tell" [22] paper. As our goal was to replace the components in the paper, it was not an easy task. All of us took different parts of the architecture and tried to improve the specific pieces as stated above. For all of us it was pretty difficult to incorporate another module in the code base and make it working. Sometimes we saw that even if we are able to attach a new module without any issues, the BLEU score was noticeably lower than the baseline model. This was because of minor bugs. But after fine tuning and looking into the code more thoroughly we were able to tune the right parameters and improve the BLUE score from the baseline model.

One big challenge we faced was the dataset selection. We wanted to use COCO data set, but found that training and running experiments on it is computationally expensive and it was taking numerous days to complete even with GPUs. This was not ideal for a course project like ours which had few weeks to do the project.

Another challenge was establishing the baseline. Even though the authors of "Show, Attend and Tell" [22] paper claimed certain accuracy numbers and BLEU scores, we were not able to reproduce the same performance with baseline configuration outlined in the paper and reference code. So we had to establish a baseline first and then run experiments to improve the performance against our baseline.

Another important challenge was the unreliability of Google Cloud Platform (GCP). GCP was very unreliable for us and was unable to create instances with GPUs when we needed to run large experiments. GCP has been giving errors such as "GPU resources not available in the zone you are requesting" for every zone. This was disappointing and limited our ability to run complex experiments. We had to resort to Google Colab GPU instance to run most of our experiments but Colab terminated our GPU instances without warnings. This resulted in lot of wasted cycles.

Yet another challenge was the complexity of FastAI framework. Even though FastAI claimed to make it easier to experiment, we found that, in practice, it makes it more difficult to experiment and reason about the results. It took us a while to figure things out.

Another challenge was rebuilding the vocabulary. Since we change the word embedding to BERT, we found that BERT breaks up words differently than Glove. So we had to pre-process all the captions and regenerate the vocabulary so that word tokens are processed properly by different parts of the seq2seq pipeline.

One of the initial challenges while implementing beam search during training was making sure we have a working version of beam search, which can be tried out during evaluation and caption generation. Now with fastai having its own way of implementing functions using callbacks, we had to have a deeper understanding of how the custom callbacks are implemented as well as how they can be called during training. Since executing beam search is computationally expensive, we trained in stages where the encoder will be frozen in the first phase so that the model can focus on updating the decoder weights first. The main reason for training in phases was because we were already using a pre-trained encoder whereas the decoder was training from scratch. Thus we had to decide if we want to include it only during the stage 2 when encoder stage is unfrozen and both encoder and decoder are running. We experimented with both approach and saw that enabling beam search only during stage 2 did not have much impact on the performance of the model, whereas the approach saved considerable amount of training time as beam search was taking almost twice as much time than training with greedy search during both training phases.

Now in order to evaluate the models' performance on validation set during training phase, we used the BLEU (Bilingual evaluation understudy) metric which was one of the metrics that were used mainly in the paper. However, in the baseline implementation, only BLEU-4 score was calculated compared to BLEU (1-4) scores in our approach. Thus we decided to implement the calculation of all BLEU scores. The main challenge in implementing the BLEU (1-4) score calculation was integrating it with the fastai api. On further analysis of the code, we found that the baseline implementation of the decoder had teacher forcing enabled

based on a probability which is known as scheduled sampling. The reason for adopting teacher forcing is if we want to take ground truth caption as input in order to reach convergence quicker rather than one generated in the previous timestamp. Thus we modified the decoder portion of the code to avoid teacher forcing during training when running with beam search.

Sub-optimal feature extraction from CNN was anticipated if there was no fine tuning done so freezing gradient calculation of different layers of the CNN was experimented. Unfreezing all the layers or any of the first few layers did not improve the caption accuracy score because it is determined unnecessary to change the early weights that contain information about basic fine features. Nonetheless, unfreezing all layers faced RAM limitation. In the end, unfreezing only the last few layers gave best results when the last linear and pooling layers are replaced with 2D adaptive average pooling to resize encoded image to a fixed size. The linear layer is removed since the task of interest is not a classification task. The key is to retain the early pre-trained layers for their rudimentary knowledge of features and apply transfer learning to the last few layers for best results. Determining the optimal learning rate for the encoder became a problem because convergence was not reached but decaying of learning rate after a certain number of repeated unsuccessful improvement in accuracy score or selecting a low learning rate was helpful, which may be because pre-trained model is already so optimized that a high learning rate is not necessary.

One issue with adding transformer to the baseline network was selecting correct output shapes from CNN layer to transformer positional encoder, multi-head attention and the feed-forward layers. Another issue with the transformer encoder was that there is no need for the embedding layer in the encoder as we are using image output as input to the encoder. After some analysis, we found that it is best to use a simple linear layer in place of embedding layer which is then passed to the positional encoder before multi-head attention.

Baseline Transformer Encoder code was taken from the below paper that implements a machine translation. [10], [14] It is then modified to fit to the CNN encoder output. Transformer decoder is not used from this code. For understanding of the implementation of beam search, we have referred to the publicly available code[15] where beam search was used for evaluation and image captioning.

3. Experiments and Results

We conducted a number of experiments as explained in Section 2. We improved 4 different parts of the architecture and ran large number of experiments. Here are the different configurations for each part used.

1. CNN in Encoder: we used VGG-16, Resnet-101 and Resnext-101 pretrained networks.

2. Pre-trained Word Embedding in Decoder: we used Glove, BERT Base with embedding size of 768, BERT Large with embedding size 1024, BERT Base with word frequency of 2+ and BERT Large with word frequency of 2+.

3. Encoder: CNN and Transformer with CNN

4. Sequence generation: Greedy and Beam search with beam size of 2 3

3.1. Framework & Baseline

We have used fastai as our deep learning framework which is built on top of pytorch and which provides various APIs making it easier to create deep learning models.One of the main reason why we chose this implementation as the baseline is it was pretty recent implementation with a different deep learning framework and also had lot more scope of improvement compared to other regular implementations on pytorch and tensorflow which had been fine tuned over many years. Our code was based on the following code base [17]. Even though we can see many implementations online of this paper, the main reason why we got interested to this implementation is because of the use of fastai framework. The existing code provided us with the basic framework of CNN encoder, an attention layer and an LSTM decoder which was pretty much common for all the implementations that was present online.The existing code also provided us with the fasttext embedding along with the implementation of bleu4 score.Even-though the author had talked about implementing beam search during evaluation as well as caption generation, this was something which was not properly implemented in the existing code base.

3.2. Loss Function

Since this is a model where we are generating a sequence of words, we have used the cross-entropy loss.From the raw scores in the final layer of the decoder, the loss function will perform operations of soft-max and log.The total loss is equal to the sum of cross-entropy between the predicted and target captions and the attention alpha loss. The objective of alpha loss is to ensure that rather than paying too much attention on the same part, the encoder and decoder gives attention throughout the image.This will prevent the model from generating captions with repeated words.In order to avoid computing losses over the padded region, we have used the pack padded sequence function from pytorch which will flatten the tensor while also ensuring that the padded regions are ignored.

No	CNN	Embedding	Encoder	Search	Val Loss	Top-5 Acc	BLEU-3	BLEU-4
1	VGG-16	Fasttext	CNN	Greedy	3.98	68.061	0.253	0.152
2	Resnext-101	BERT Base	Transformer	Greedy	3.764	71.146	0.261	0.163
3	VGG-16	BERT Base Freq 2+	CNN	Greedy	3.743	69.753	0.248	0.152
4	Resnet-101	BERT Base	Transformer	Greedy	3.787	70.404	0.255	0.158
5	Resnet-101	BERT Base Freq 2+	CNN	Greedy	3.747	69.698	0.248	0.152
6	Resnet-101	BERT Large	CNN	Greedy	3.775	70.983	0.258	0.161
7*	Resnext-101	BERT	CNN	Beam-2	3.673	71.400	0.252	0.156
8	Resnet-101	BERT Base Freq 2+	Transformer	Beam-3	3.700	70.873	0.254	0.158

Table 1. Experiment Results (some experiments were unfinished due to GPU issues and we have provided the best results we got so-far)

3.3. Experiment Setup

We used Jupyter notebook for our code and ran the experiments in Google Colab Pro and Google Cloud Platform(GCP) Compute with P100 and A100 GPUs. Colab was used for medium sized experiments which takes less than a day to run. For experiments which takes larger compute, we used GPU-powered GCP instance as the runtime for Colab. We spun up GCP instances with GPU, then established an SSH tunnel from the laptop to GCP instance, and then chose 'local runtime' in Colab and provided the tunnel port. This way, when Colab tried to connect to local runtime in the laptop, it gets tunnelled to the GCP instance and notebook gets executed using the kernel in the GCP instance.

3.4. Pre-processing

Our model have used the following data as inputs at various stages of its architecture:

For **word embeddings**, we had to do pre-processing. The baseline used the FastText embedding with word frequency 2+. We downloaded the pre-trained word vectors for 157 languages from [6]. The vocabulary and caption tokens were based on the FastText embedding projected on the captions in entire Flickr8k data set. When we introduced the BERT embeddings, we had to pre-process the captions in the entire Flickr8k data set to run the BERT model on them to get contextualized embeddings. Since BERT uses different token breakup method than FastText, we then had to re-build the vocabulary based on the BERT tokens. We also updated the token information in the caption files accordingly. We also created embeddings by filtering out tokens which has a word frequency of at least 2 in the Flickr8k data set. This was done to evaluate whether forcing out less common words have any impact on the caption quality.

Vocabulary which are used to generate captions are the inputs to the decoder for which we have used Andrej Karpathy's training, validation, and test splits. [8] Also since the captions are padded, we had to keep track of the length of each caption and pass it as an input along with the caption.

Images are resized to 350 by 350 and normalized to ImageNet statistics.

3.5. Hyper-parameter Tuning & Model Evaluation

One of the advantages of using the fastai framework was the availability of in-built functions like lr finder which will go over a range of learning rates and provide a plot against the losses there by helping us in choosing the optimal value as shown in the below plot.

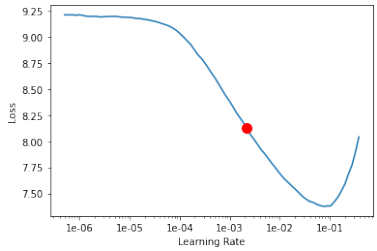

Figure 3. Our Modified Transformer Architecture

We have used a learning rate of 5e-4 for stage 1 and 1e-4 for stage 2 as our baseline setup. We had a batch size of 6 and we did the training both stage 1 and stage 2 for 10 epoch each.We have used Adam optimizer in our baseline code. All the above parameters were from our baseline code which were then used as is or slightly tuned further for each individual experiments based on the requirement.
For the Transformer an EncoderLayer consists of a Multi-head attention layer with 8 heads, feedforward layer and 2 dropout layers. We have 6 such Encoder layers. Dropout is set to 0.1

3.6. Measuring Results

To measure the success, we primarily used the BLEU scores and top-5 accuracy. We ran the experiments using the baseline code to reproduce the baseline results and then changed the architecture one by one and reran the experiments to measure if the change made any difference. We used one-change-at-a-time initially to measure the impact of each enhancement and then eventually ran experiments

using combinations of changes. The detailed results of various experiments are shown in Table 1.

3.7. Results

We initially ran baseline experiment and we got a BLEU-3 score of 0.253 and BLEU-4 score of 0.152. The top-5 accuracy was 68.061. Then we ran large number of experiments with different configurations. The best results we got was for for the architecture which used Resnext-101 as CNN, BERT Base as embedding, Transformer in the encoding layer and greedy search. We were able to get BLEU-3 score of 0.261 and BLEU-4 score of 0.163 with Top-5 accuracy of 71.146 and validation loss of 3.764.

Figure 4. Baseline Caption

The above figure demonstrates the caption generated after we trained the baseline model.Now after implementing the transformer as well as the beam search, we trained the model and the caption was generated as shown in the figure below.

Figure 5. Transformer with Beam Search Caption

With the aim of aligning the objective function with the bleu score during evaluation, we implemented beam search at the decoder step during training of the model.However after running multiple experiments with different beam sizes and tuning other hyper-parameters, we could see that training the model with beam search had a slightly sub optimal performance when compared to the greedy decoder.Even though the results for both the approaches were roughly comparable in the sense that beam search was not performing way bad than the other approach, the results were contrary to our belief of beam search performing exceedingly well.After further research, we were able to narrow down the reason as mentioned in the the paper "Sequence-to-Sequence Learning as Beam-Search Optimization" [20] that Loss-Evaluation mismatch was the main issue with our approach.The authors have clearly called out a Beam-search Training Scheme to address the same.

4. Future work

As this project belongs to an area where more and more advanced techniques and architectures keep coming up every now an then, we do have immense scope to improve the model by adapting to newer optimized techniques. Implementing Beam-search Training Scheme as mentioned in paper "Sequence-to-Sequence Learning as Beam-Search Optimization" [20] is definitely one of the areas where we can focus on to build from here. In this project, we were not able to replace the decoder to use a transformer. A full fledged transformer implementation is something is very promising and expected to produce better results. That is something we want to further experiment. Evaluation of model with other metrices like METEOR, ROUGE, CIDEr etc is something which we would like to get implemented in order to perform a more holistic evaluation of the model. In this project, we experimented with BERT embedding, but there are other promising embeddings such as GPT-3 [1] and XL-Net [23]. We would like to experiment with those to see if it can further improve the performance.

References

[1] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. 6

[2] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014. 1

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. 2

[4] Markus Freitag and Yaser Al-Onaizan. Beam search strategies for neural machine translation. *Proceedings of the First Workshop on Neural Machine Translation*, 2017. 2

[5] F.A. Gers. Learning to forget: continual prediction with lstm. *IET Conference Proceedings*, pages 850–855(5), January 1999. 1

[6] Edouard Grave, Piotr Bojanowski, Prakhar Gupta, Armand Joulin, and Tomas Mikolov. Learning word vectors for 157 languages. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*, 2018. 5

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. 2

[8] Andrej Karpathy, 2021. 5

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012. 1

[10] Samuel Lynn-Evans. How to code the transformer in pytorch, Oct 2018. 4

[11] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318, 2002. 3

[12] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014. 1

[13] Alexander M Rush, Sumit Chopra, and Jason Weston. A neural attention model for abstractive sentence summarization. *arXiv preprint arXiv:1509.00685*, 2015. 1

[14] SamLynnEvans. Samlynnevans/transformer. 4

[15] Sgrvinod. sgrvinod/a-pytorch-tutorial-to-image-captioning. 4

[16] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015. 2

[17] Santhosh Skumarr53. Skumarr53/image-caption-generation-using-fastai, 2020. 4

[18] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014. 1

[19] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017. 1

[20] Sam Wiseman and Alexander M. Rush. Sequence-to-sequence learning as beam-search optimization. *CoRR*, abs/1606.02960, 2016. 6

[21] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks, 2017. 2

[22] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057. PMLR, 2015. 1, 3

[23] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. Xlnet: Generalized autoregressive pretraining for language understanding, 2020. 6

Appendices

A. Division of Work among Team Members

Student Name	Contributed Aspects	Details
Ahamed Kannanari	BERT Embedding setup, GCP Backend runtime setup with Colab	Added BERT embedding to the architecture. Wrote the pre-processing code for tokenizing the Flickr8k captions using BERT Base and BERT Large and rebuilt the vocabulary based on the tokens. Setup the GCP instance to work as backend runtime for Google Colab. Discovered and fixed many bugs. Setup the the final fully automated notebook which uses a new folder everytime an experiment is run. Figured out using Google Drive shortcut to save time copying the large data set for each experiment.
Alex Song	Baseline setup and CNN architecture experimentation	Setting up various non-fastai based baseline code, Evaluated feasibility of training full COCO dataset, Experimented freezing of different layers for fine tuning and various CNN architectures, Experimented various pre-processing parameters. Experimented hyperparameter tuning.
Deepu Thomas	Transformer Encoder setup	Researched on various architectures that can give more contextual information to the CNN images. Converted a seq2seq translation transformer to fit in the current image captioning problem. Able to replaced the default CNN Encoder with Transformer, Transformer consists of existing CNN layer, Linear layer, Positional Encoding, Multi-head attention, feedforward layer and dropout layers.
Sachin Chandrasekhar	Baseline setup and Beam search implementation	Setup the baseline code of image caption using fastai. Got the beam search code working for the evaluation and caption generation stages as it had bugs in the baseline code. Implemented beam search during training of the model rather than the normal implementation during the evaluation or caption generation stage. Also implemented the calculation of Bleu scores(1-4) which was the metric used to evaluate the model.Tried setting up different metric like METEOR and ROUGE for evaluating the model.

Table 2. Division of Work among Team Members

B. Project Code repository

- Google Drive link: https://tinyurl.com/r8wm8e7x
- Github link: https://tinyurl.com/yrv58ftm