Department of Computer Engineering

CS464 - Introduction to Machine Learning

Spring 2021

Term Project Final Report

AutoCapNet: Deep Learning Based Approaches for Automatic Image Captioning

Group 14

Teaching Assistant: Furkan Özden

Section: 2

Group Members:

Melisa Taşpınar	21803668
Yiğit Gürses	21702746
Emin Adem Buran	21703279
Musa Ege Ünalan	21803617
Mustafa Göktan Güdükbay	21801740

Table of Contents

Introduction

In this project, we aim to perform automatic image captioning using deep learning-based methods. To be more specific, the project aims to describe given images with sentence-based captions with several deep learning methods. The procedure can be divided into two main stages: encoding with a convolutional neural network (CNN) and decoding with a recurrent neural network (RNN). While conducting this project, we have used different CNN and RNN architectures with different attention mechanisms. As the CNN architecture at the encoding stage, we have used *InceptionV3* [1], *VGG16* [2], and *ResNet50* [3]. Then, as the RNN architecture at the decoding stage, we have used LSTM [4] and GRU [4]. We also experimented with transformer[5] based decoders. The dataset used was the *Flickr8k* [6] dataset. In the end, we have provided our results and made comparisons based on performance. When we compare the CNN architectures, the best accuracy was obtained with *ResNet50*, though *InceptionV3* gave good results. In terms of RNN, GRU gave slightly better results compared to LSTM. In terms of the attention mechanism, Bahdanau performed well. The transformer performed relatively well, given the memory constraints we had.

Problem description

Image captioning is a process involving computer vision and natural language processing. After the dataset is cleaned and preprocessed for the job, the remaining procedure can be divided into two main stages: encoding, which is related to computer vision, and decoding, which is related to natural language processing.

The part related to computer vision is feature extraction, i.e., the encoding stage. First of all, we are supposed to extract features from input images. These features will then be flattened into a vector and then transformed/resized so that the feature vector is ready to be used in the decoding stage. Thus, we should use a system that can extract features from images and use these, form feature vectors. The attributes of the extracted features are essential as they affect the accuracy of the image captioning system. The size of the extracted features itself is an example of this, as it can directly affect the decoding stage.

The decoding stage involves natural language processing, as it is the stage at which the features are translated into words. In other words, we try to decode the image features we acquired in the first stage and with these form sequences of words that can describe the image at hand. In simple image captioning decoders, the next word depends on the previous one forming a sequence. This behavior can lead to long and meaningless captions. Therefore, the composition and generalization in the decoder part are crucial and affect the model's accuracy. The structure used here should possess some memory and not suffer from long-term dependencies to form meaningful sentences.

We are trying to address the question, "What is the best approach for automatic image captioning? What leads to the highest accuracy?". Or, to be more specific, we will be trying to answer the following questions: "Which CNN architecture is better for feature extraction?", "Which RNN architecture is better for encoding the extracted features?", "Do attention mechanisms give better performance and accuracy?", "If attention mechanisms give better performance, which one is the best?", "What is the best combination of CNN and RNN architecture?" and so on.

In this project, we try to find the optimal approach for extracting features and producing captions with composition and generalization.

Methods

For our project, we used the *Flickr8k* [6] dataset, as we stated in our proposal. *Flickr8k* is a benchmark dataset designed to be used for the task of sentence-based image description. The dataset contains 8,091 images, and each image has five different captions describing the contents of each image.

The first task to perform automatic image captioning is to clean and preprocess the dataset. Afterward, automatic image captioning is divided into two smaller processes: encoding images and decoding the acquired features to form strings of words.

For the first stage, i.e., encoding the images, we used a Convolutional Neural Network (CNN), as it will be able to extract features from the input images. We used a CNN that performs

classification and got rid of its last layer to output the relatively high-level features for classification. These output features were then flattened to acquire a vector of features. Then this vector of features was transformed and resized so that it had the same length as the word embedding.

To be more specific, we tried three different CNN architectures to extract the features from the images. These architectures are *InceptionV3* [1], *VGG16* [2] and *ResNet50* [3]. These were pretrained on the *ImageNet* [7] data set.

InceptionV3 architecture is built to obtain a computationally efficient convolutional neural network. It uses factorized convolutions with larger filter size, smaller convolutions, asymmetric convolutions auxiliary classifiers, and grid size reduction. Therefore, the network contains a smaller number of parameters and a smaller number of computations, which allows for faster training of the model. Figure 1 depicts the *InceptionV3* architecture.

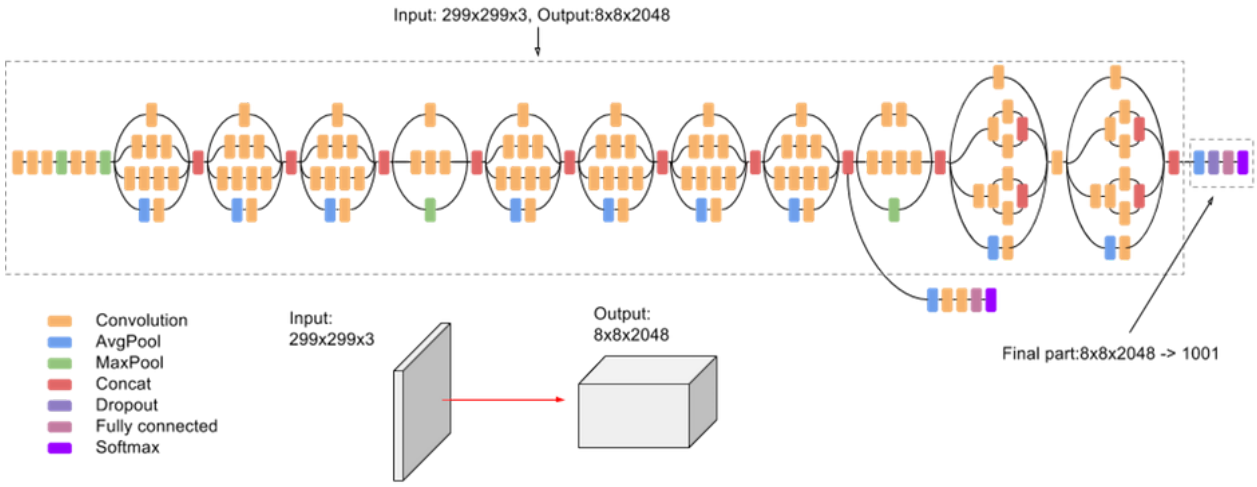


Figure 1: *InceptionV3* architecture [1].

VGG16 architecture has only 16 convolutional and pooling layers with some weights. It uses a 3x3 kernel size and a 2x2 pool size for all the layers and achieves an accuracy of 92.7%, noting that this accuracy can increase the performance of the image captioning task. Figure 2 shows the *VGG16* architecture.

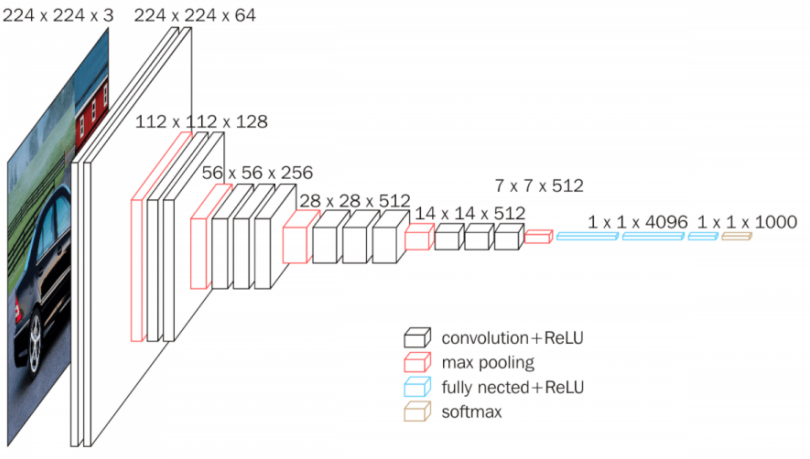


Figure 2: *VGG16* architecture [2].

ResNet50 is a CNN with 50 layers. It is pretrained on more than a million images from the *ImageNet* dataset and can categorize images into 1000 object categories. Figure 3 shows the *ResNet50* architecture.

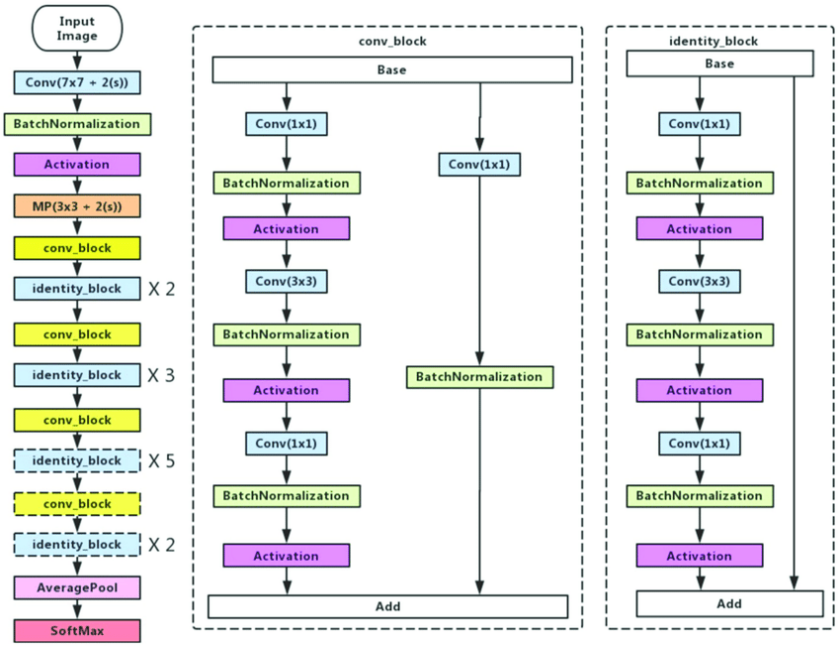


Figure 3: *ResNet50* architecture [3].

The second stage for image captioning is to decode the features acquired in the first stage and to use these features to compose sequences of words that fittingly describe the given images. For this task, we used an RNN. RNNs, unlike CNNs, accommodate loops that allow them to

maintain information and thus have some kind of memory. However, this memory tends to be limited, and as a result, RNNs suffer from so-called "long-term dependencies." [8]. As a remedy to this problem, a particular type of RNN, called Long Short Term Memory (LSTM), is proposed. LSTM are structured to tackle the long-term dependency problem. These networks have memory cells in their architectures that can preserve information for an extended amount of time. Therefore, they can be used for the decoding task during image captioning. [9]

An LSTM has four layers: the input gate, forget gate, cell state, and output gate. The forget gate passes the previous hidden state and current input features through a sigmoid layer to decide which information to discard. The input gate passes the previous hidden state and the current input through a sigmoid function to determine the importance of the inputs to the cell. An intermediate candidate cell state is produced by passing the previous hidden state and the current input through a *tanh* function. The current cell state is then updated using the output of the forget gate, previous cell state, the output of the input gate, and the candidate cell. The output gate decides on the next hidden state. It passes the previous hidden state together with the current input through a sigmoid function. The current cell state calculated is passed through the hyperbolic tangent function. Lastly, an element-wise multiplication is performed with the result of the output gate and the result of the *tanh* function to obtain the hidden layer's output [4], [8].

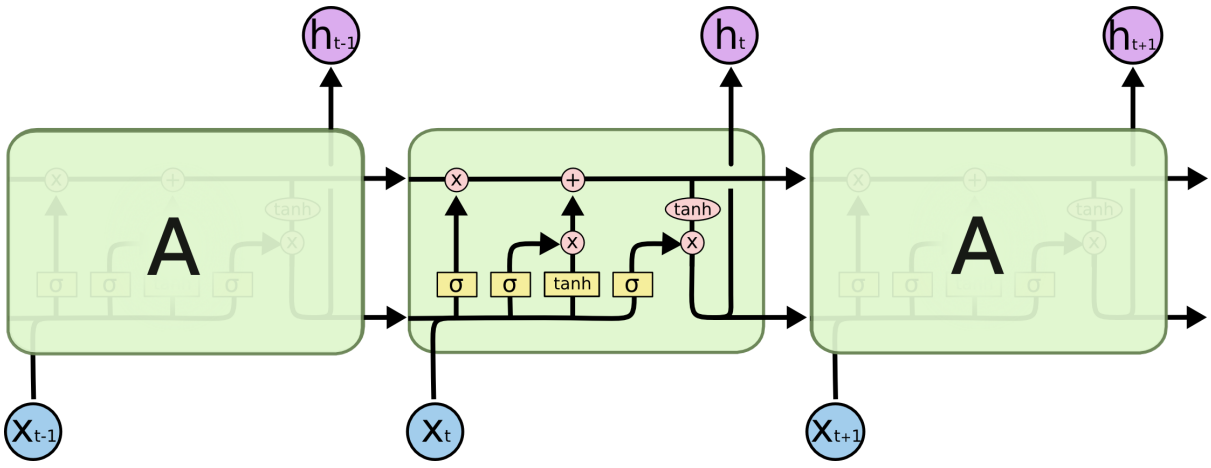


Figure 4: The repeating module in an LSTM [8]

Equations for the LSTM are as follows [4], [8].

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f)$$

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = tanh(W_C[h_{t-1}, x_t] + b_c)$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \quad o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * tanh(C_t)$$

In these equations,

- f_t is the forget gate vector,
- i_t is the input gate vector,
- $\tilde{C}_t$ is the intermediate cell state vector,
- C_t is the current cell state vector,
- o_t is the output gate vector, and
- h_t is the vector for the output of the hidden layer.

Another type of RNN is a Gated Recurrent Unit (GRU). GRU is similar to LSTM, but they are simpler and deal with the vanishing gradient problem. They have a reset and an update gate. The reset gate decides if the previous cell's output has an effect and the update gate decides if the current input affects the next state. The update and reset states are calculated by passing the input feature and the previous hidden state through a sigmoid function, each with different weights. An intermediate memory is produced by passing the input feature and the output of the reset gate through the hyperbolic tangent function. Finally, the output is calculated by adding two components: the element-wise multiplication of the output of the update gate and the previous hidden state. The element-wise multiplication of the output of the update gate is subtracted from 1 and the intermediate memory.

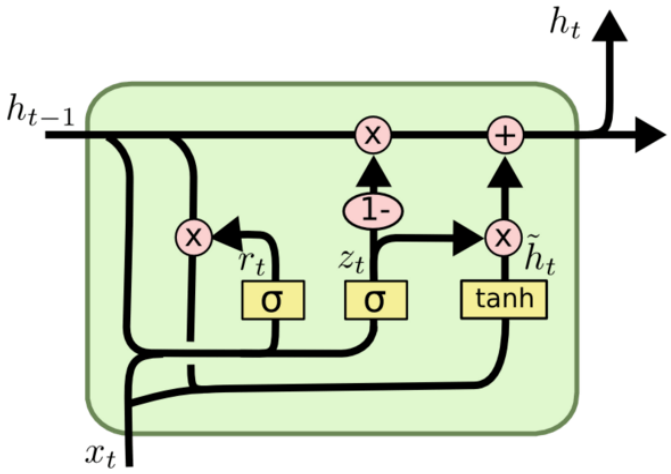


Figure 5: The structure of a GRU [8]

Equations for the GRU are as follows [10]:

$$z_t = \sigma(W_r x_t + U_z h_{t-1} + b_z)$$

$$r_t = \sigma(W_r x_t + U_z h_{t-1} + b_z)$$

$$\hat{h}_t = tanh(W_h x_t + r_t * U_h * h_{(t-1)} + b_z)$$

$$h_t = z_t * h_{(t-1)} + (1 - z_t) * \hat{h}_t$$

where

- z_t is the update gate vector,
- r_t is the reset gate vector,
- $\hat{h}_t$ is the candidate activation vector, and
- h_t is the output vector for the hidden layer.

While using GRU, we used two different attention mechanisms: Luong's (Global) attention and Bahdanau's (Local) Attention. These attention mechanisms help the RNN use relevant parts of the given image to generate words [11].

When Bahdanau's attention is used together with the RNN, there are six major steps. First, the encoder hidden states are produced, and the previous decoder hidden states are combined with each of the encoder hidden states to calculate the alignment scores.

$$score_{alignment} = W_{combined} . tanh(W_{decoder} . H_{decoder} + w_{encoder} H_{encoder})$$

In this equation, which gives the alignment score for Bahdanau attention,

- $W_{combined}$ is the combined weights,
- $W_{decoder}$ is the weights of the decoder,
- $H_{decoder}$ is the hidden state of the decoder,
- $W_{encoder}$ is the weights of the encoder, and
- $H_{encoder}$ is the hidden state of the encoder.

The alignment score is then softmaxed. Then the context vector is calculated by multiplying the encoder hidden states and alignment scores. Then the context vector is concatenated with the previous decoder output and fed into the RNN to produce the new output vector. This process repeats until a word is produced.

When Luong attention is used together with the RNN, there are seven major steps. Similarly, the first encoder hidden states are produced. However, after this, the previous decoder's hidden state and decoder output is passed through the Decoder RNN to generate a new hidden state. This new hidden state calculates the alignment score, and the alignment score is then softmaxed. The context vector is calculated by multiplying the encoder hidden states and the alignment scores. When the final output is produced the context vector is *concatenated* with the new decoder hidden to produce a new output. This process repeats until a word is produced.

There are three different ways to calculate the alignment score in Luong attention. These are *dot*, *general* and *concat*. We used the general type to calculate the alignment scores in our image captioning task [12].

$$score_{alignment} = W_{combined}(H_{encoder} . H_{decoder})$$

In this equation, which gives the general alignment score for Luong attention,

- $W_{combined}$ is the combined weights,
- $H_{decoder}$ is the hidden state of the decoder, and
- $H_{encoder}$ is the hidden state of the encoder.

Our project used the Bilingual Evaluation Understudy (BLEU) score to evaluate the captions our model generates by treating them like translations. BLEU is calculated using brevity penalty and a precision score calculated from the n-grams of the generated captions and the captions from the dataset. The calculations are as follows. [13]

$$BP = \begin{cases} 1 & if\ c > r \\ e^{(1-r/c)} & if\ c \leq r \end{cases}$$

$$BLEU = BP * exp(\sum_{n=1}^{N} w_n log p_n)$$

where

- BP is the Brevity penalty,
- r is the count of words in a reference translation (one of the original captions from the database R),
- c is the count of words in a candidate translation (a caption generated by our model),
- N is the number of n-grams, we usually use unigram, bigram, 3-gram, and 4-gram,
- w_n is the weight for each modified precision, by default N is 4, w_n is 1/4=0.25, and
- p_n is the modified precision.

Transformers are currently considered the state of the art for NLP tasks. This is because they are capable of processing and generating sequential data very well, on a large scale and in parallel. In an RNN architecture, to generate the output for time step *t*, the output of time step (*t* - 1) is required. This means the time steps have to be processed sequentially, and therefore, RNNs are problematic in the context of parallelization. Transformers can overcome this problem since they do not use the information propagated from previous time steps. Instead, they use a

combination of positional encoding (spatial encoding in the case of images) and attention [5]. This property allows transformer-based models to take advantage of parallel computing in high-powered compute clusters.

That being said, transformers do take up lots of memory, and it is not possible to take advantage of the parallelizability with a single GPU. Since we were limited to 4GB of ram and 1 GPU during training and evaluation of models, these factors heavily affected the success of our transformerased decoder.

The baseline architecture of a transformer is given in Figure 6.

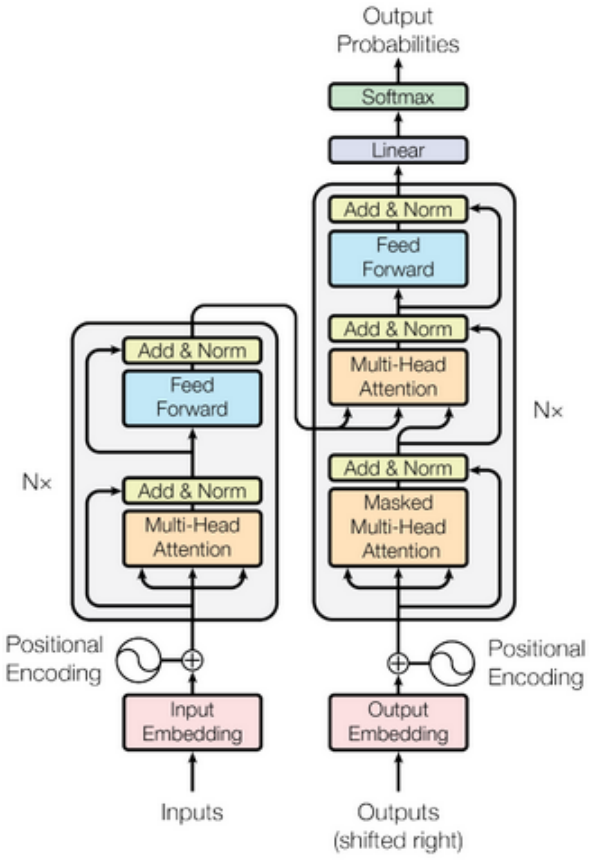


Figure 6: Baseline Transformer Architecture [14]

The left side of the image is the encoder and the right side is the decoder. These encoders and decoders can be stacked indefinitely. In our project, we used a single encoder, single decoder architecture due to memory and computation limitations. Inputs to the encoder and decoders first

pass through an embedding layer, and then the embeddings are positionally encoded. This positional encoding allows the model to take advantage of the index an entry occupies in a sequence. We can do positional encoding in various ways as long as it somewhat captures the positional information in the result. However, we used the following equations for our positional encoding ($d_{emb\text{-}dim}$=32 in our experiments, j=32 too, so we used the sin formula in this case):

$$p_{i,j} = \begin{cases} sin\left(\frac{i}{10000^{\frac{j}{d_{emb-dim}}}}\right) & \text{if j is even} \\ cos\left(\frac{i}{10000^{\frac{j-1}{d_{emb-dim}}}}\right) & \text{if j is odd} \end{cases}$$

Once the positionally encoded embeddings are passed to the network, self-attention layers are utilized to calculate the relevance of each index to the others. We used multi-headed attention to increase the accuracy of this process. This means that multiple self-attention layers independently calculate the relevance scores. These results are then concatenated and passed through a linear layer (feed-forward) to get a final result. This allows different heads to focus on different patterns and features, which boosts the overall performance. However, it comes with the cost of significantly increasing the model size and computational complexity.

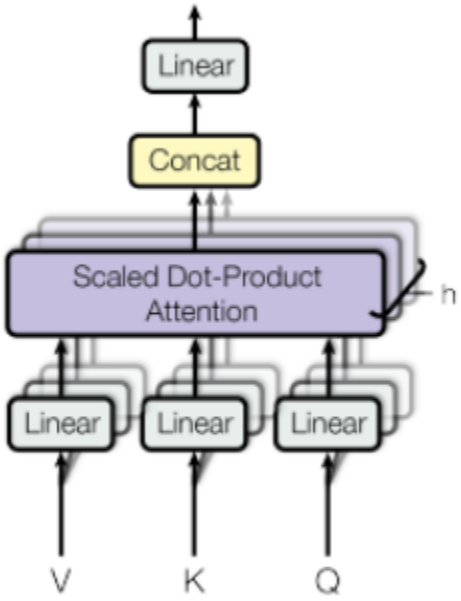


Figure 7: Multi-Head Attention Illustration [14].

The addition and layer normalization is used with residual skip connections. Layer normalization is similar to batch normalization. However, instead of normalizing the input values for each neuron in a mini-batch, the input values of all neurons for a given data sample are normalized.

The outputs of the decoder are passed through a linear layer + softmax layer to obtain the likelihood of the next entry in the sequence. The *argmax* of these likelihoods will give us the predicted outputs. These predicted outputs are shifted one time step backward and fed back to the decoder to get the next entry in the sequence and so forth. Once the whole output sequence is obtained, just like in the RNN models, we used cross-entropy loss and adam optimizer for training.

This is how a baseline transformer architecture works on a sequence to sequence basis. However, since our inputs are images, we either need to transform them into sequences using patches and spatial encoding or skip this step altogether. Instead, the image features obtained through a CNN can be directly fed to the decoder in the place the encoder's output normally would. An illustration of a similar concept can be seen in Figure 8.

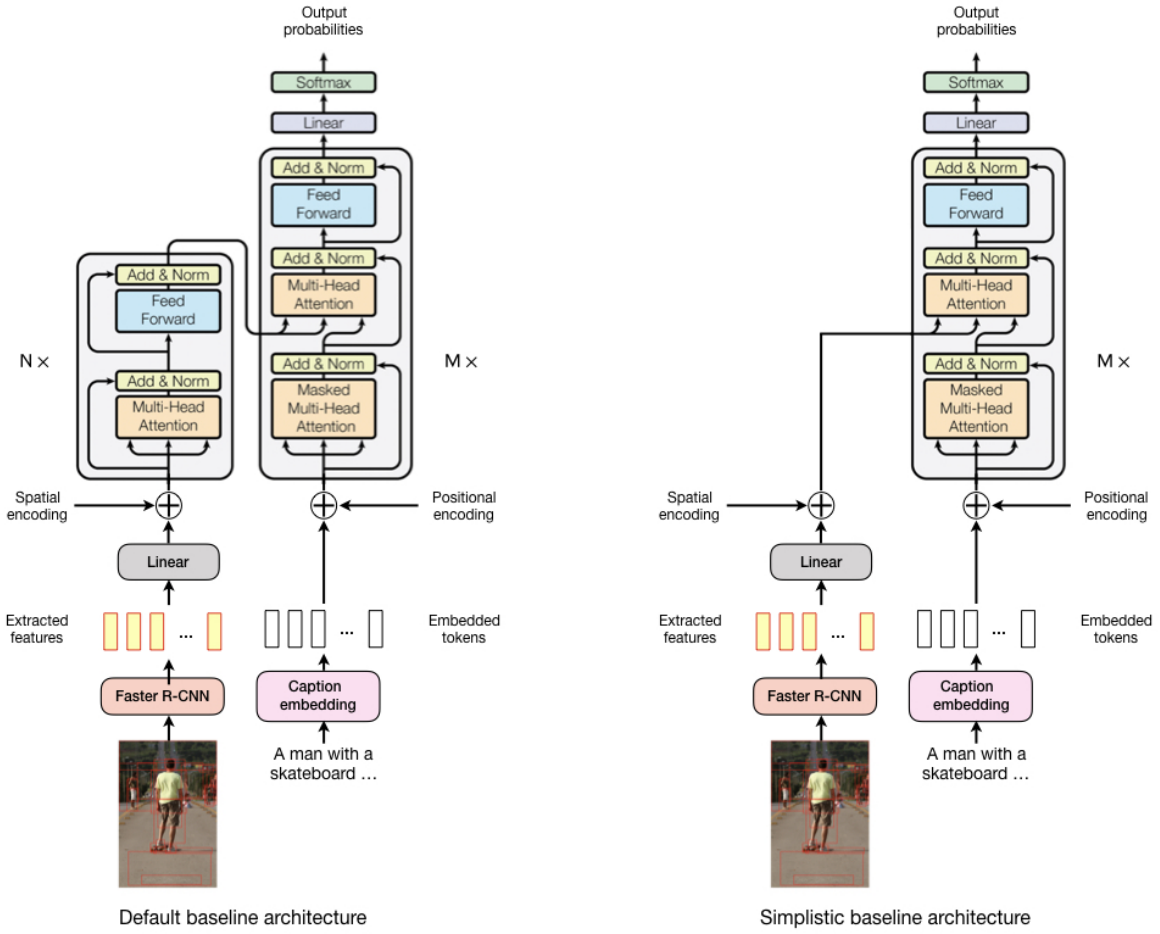


Figure 8: Transformer Architecture for Image Captioning [15]

Indeed, that was the approach we took with our model. As the RNN architecture was also only used as the decoder, this made sense in the context of this project. However, our approach was simpler than it is in figure 8, as we directly fed the image feature vector (after a linear transformation) to the multi-head attention layer in the decoder.

Results

While implementing the image captioning task, we used *InceptionV3, VGG16, ResNet50* architectures that were pretrained on the ImageNet data set at the encoding stage. Then, we used two different RNN implementations at the decoding stage. Mainly, we worked on two different RNN implementations. In the first implementation we used GRU, LSTM as RNN architecture with the help of the Tensorflow library. With the LSTM we only tried Bahdanau while with the GRU we tried both Bahdanau and Luong attention mechanisms.

In the second implementation, we used Pytorch library to implement LSTM's. We also tried *InceptionV3, VGG16, ResNet50* architectures but we did not use any attention mechanism in this part.

In both implementations, while training our models, we determined batch size to be 64 and epoch number to be 50.

When training the transformer, in order to get a batch size of 64, we had to significantly lower the model size. Transformers are much more computationally intensive and require much more memory compared to a regular RNN model. Even then, it was not feasible to get 50 epochs (each epoch was still taking around 15 minutes) so the training for the transformer was limited to 10 epochs.

The results can be seen on Table 1.

Encoder Architecture	**Decoder Architecture**	**Attention Mechanism**	**Bleu 1**	**Bleu 2**	**Bleu 3**	**Bleu 4**
InceptionV3	GRU	Bahdanau	0.5150	0.2935	0.1281	0.0483
InceptionV3	GRU	Luong	0.4274	0.2251	0.0690	0.0216
InceptionV3	LSTM	Luong	0.4951	0.2531	0.0946	0.0342
VGG16	GRU	Bahdanau	0.4981	0.2803	0.1125	0.0377
VGG16	GRU	Luong	0.3837	0.1721	0.0275	0.0050
VGG16	LSTM	Luong	0.4376	0.1973	0.0470	0.0140
ResNet50	GRU	Bahdanau	**0.5312**	**0.3157**	**0.1447**	**0.0568**
ResNet50	GRU	Luong	0.4136	0.2100	0.0511	0.0136
ResNet50	LSTM	Luong	0.4916	0.2517	0.0918	0.0347
InceptionV3	LSTM	None	0.5135	0.2712	0.1185	0.0498
VGG16	LSTM	None	0.5065	0.2650	0.1104	0.0436
ResNet50	LSTM	None	0.5109	0.2663	0.1091	0.0433
ResNet50	Transformer	Self Attention	0.5221	0.2122	0.0149	0.0018
VGG16	Transformer	Self Attention	0.3441	0.0821	0.0000	0.0000
InceptionV3	Transformer	Self Attention	0.5031	0.1639	0.0002	0.0000

Table 1: Bleu Scores Using Different CNN, RNN Architectures and Attention Mechanisms

The highest Bleu 1, Bleu 2, Bleu 3, and Bleu 4 scores we got are 0.5312, 0.3157, 0.1447, and 0.0568, respectively. They all belong to the model where ResNet50 was used together with GRU and Bahdanau Attention. When we used Luong's attention mechanism with GRU, the Bleu scores decreased significantly. Bleu 3 and Bleu scores were generally lower.

Questions to Answer

1. Which CNN architecture is better for feature extraction?
2. Which RNN architecture is better for encoding the extracted features?
3. Do attention mechanisms give better performance and accuracy?
4. If attention mechanisms give better performance, which one is the best?
5. What is the best combination of CNN and RNN architecture?

Discussion

In the GRU-based implementation, we had six different models. We used two different attention mechanisms and three different CNN architectures. When we compare results in terms of attention mechanisms, Bahdanau's Attention was better in Bleu scores. When we compare results in terms of the CNN architectures used for feature extraction, ResNet50 architecture gave the best performance in Bleu scores. The best performance with the GRU was the combination of ResNet50 architecture with the Bahdanau Attention Mechanism. In GRU-based implementation we got logical Bleu 1 and Bleu 2 scores, but Bleu 3 and Bleu 4 were low.

In LSTM-based implementation, we had six different models. We tried three different CNN architectures with Luong and without any attention mechanisms. LSTM without any attention mechanism gave slightly better results compared to LSTM with Luong attention. In LSTM-based implementation we got the best performance with InceptionV3 architecture for feature extraction and without any attention mechanism. However, the Bleu scores were similar compared to the cases where ResNet50 and VGG16 were used for feature extraction and none of the attention mechanisms were used.

As it was mentioned in the previous sections, we had technical difficulties with the transformer-based model due to mainly two factors. The first factor was the fact that we could train our models on only a single GPU at a time which meant that we could not utilize the parallelizability of the transformer which is where it gets its power from. The second factor was the fact that the 4GB RAM of our GPU was not big enough to contain a decently sized transformer model that could also be trained with a reasonable batch size. This meant that we

had to compromise and reduce the size of the model significantly. Overall, we believe that all these factors led to the sub-par results we obtained from the transformers, but considering the situation, we can say that it performed relatively well.

Conclusion

As a result of our implementations for different models, we have concluded that the Bleu score depends on the dimensions of the features extracted from the CNN. As the number of filters in the layer that we extracted features increased, we got better results. For example, InceptionV3 and ResNet50 had 2048 filters, while VGG16 had 512 filters, and in final stages, we observed that InceptionV3 and ResNet50 gave better performance than VGG16. Besides, we realized that the output of extracted features had an impact on the performance of different models. For example, in VGG16 and Resnet50, the output from the last layer was 7x7. However, in Inception V3, it was 8x8. When we obtained the Blue scores with different CNN architectures, the Bleu score obtained with CNN, which has smaller output of extracted features was higher than other Bleu scores obtained with CNN, which has a larger output of extracted features. Looking at our results, we think that Bahdanau Attention should be the attention mechanism of choice.

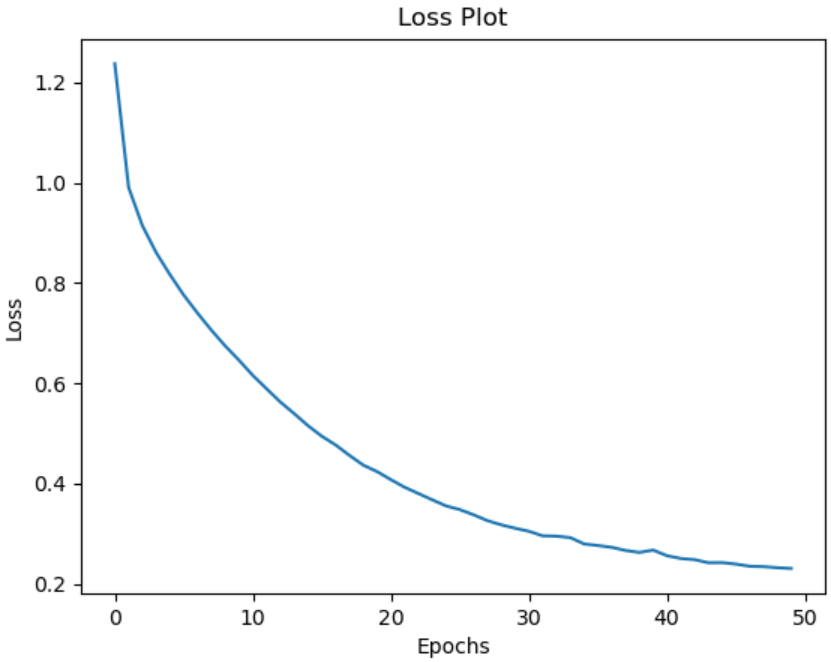
For future directions of interest, we consider trying different convolutional dimensions in feature extraction using our custom CNN and creating our own attention mechanism. We think that creating our own attention mechanism might give better performance than our existing models.

Appendix

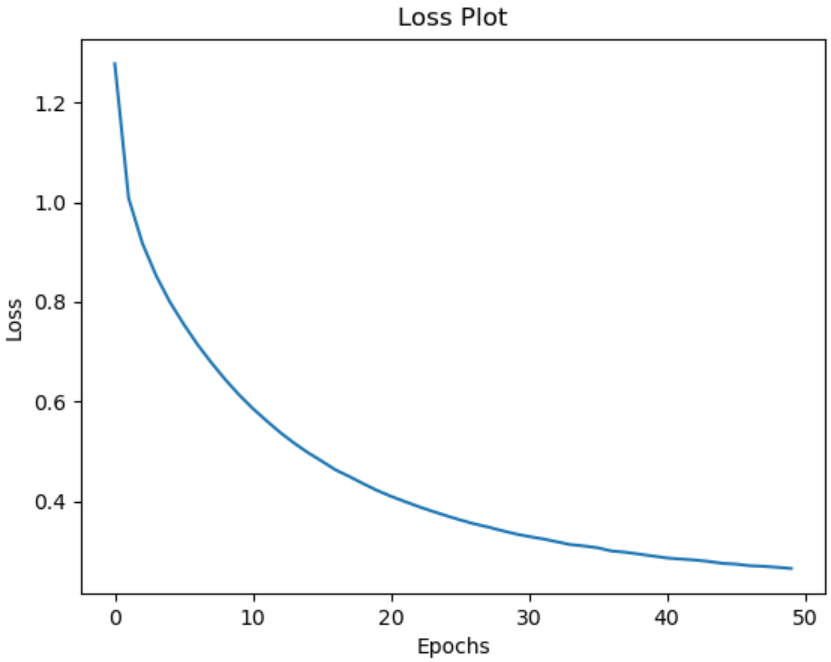
The contribution of each person is as follows.

- Melisa Taşpınar: Contributed to research, presentation and report; worked on the LSTM model.
- Yiğit Gürses: Worked on LSTM and Transformer models, contributed to research, presentation and report.
- Mustafa Göktan Güdükbay: Contributed to the report and worked on the TensorFlow models with GRU and LSTM.
- Musa Ege Ünalan: Worked on the TensorFlow model that used GRU - LSTM with different attention mechanisms and different CNNs. Contributed to the report.
- Emin Adem Buran: Contributed to final report and presentation, worked on some TensorFlow models with GRU and different CNN's.

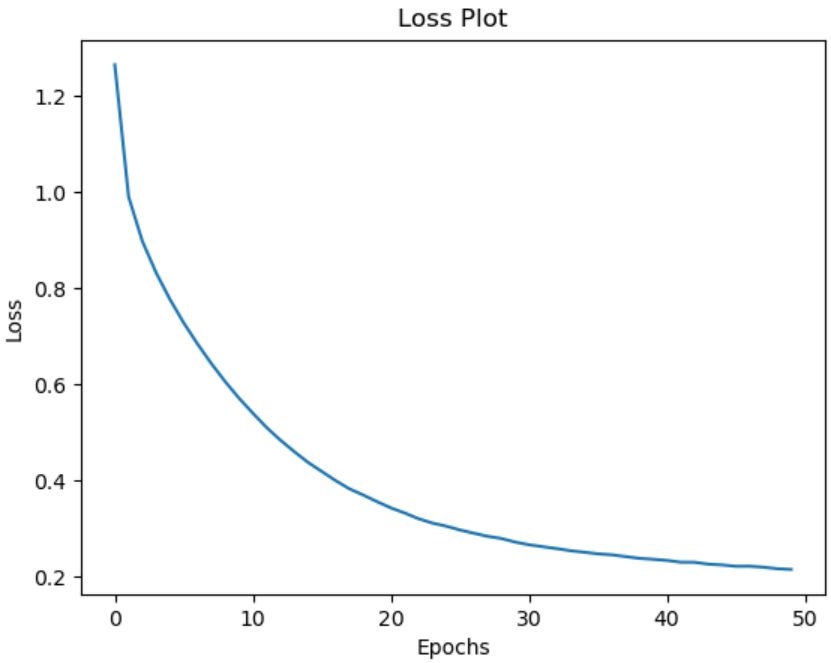
In addition, we have included our loss graphs during the training stage and some example captions below.



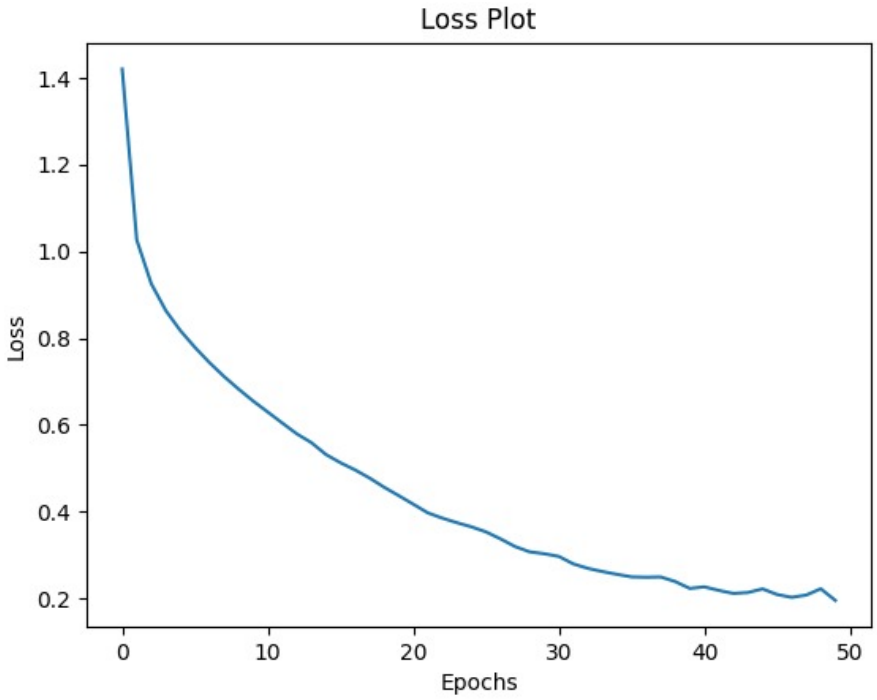
InceptionV3 - GRU With Bahdanau Attention Loss Graph



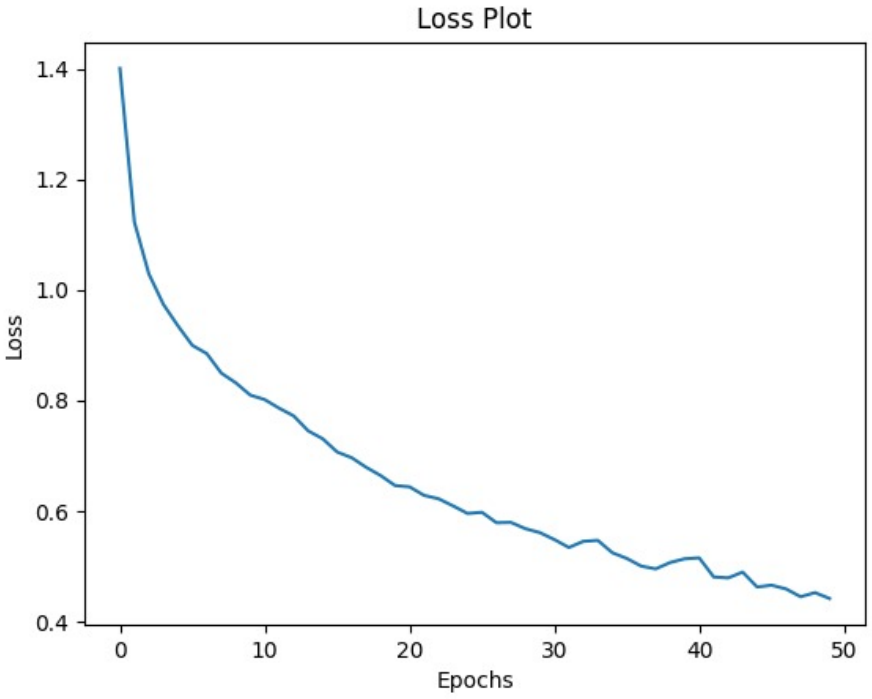
VGG16 - GRU With Bahdanau Attention Loss Graph



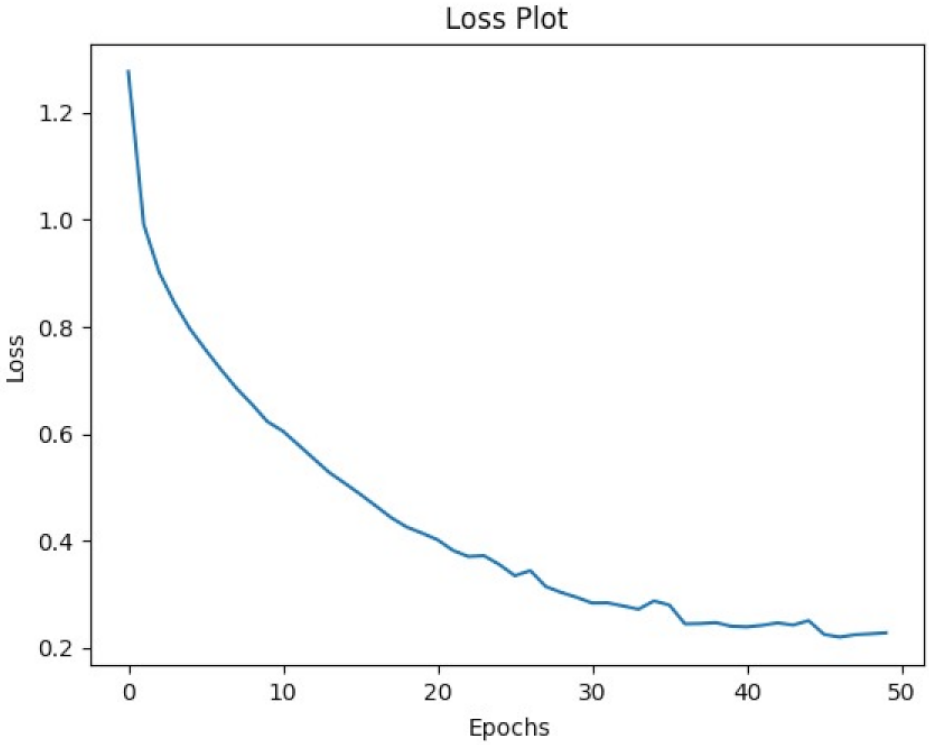
ResNet50 - GRU Bahdanau Attention Loss Graph



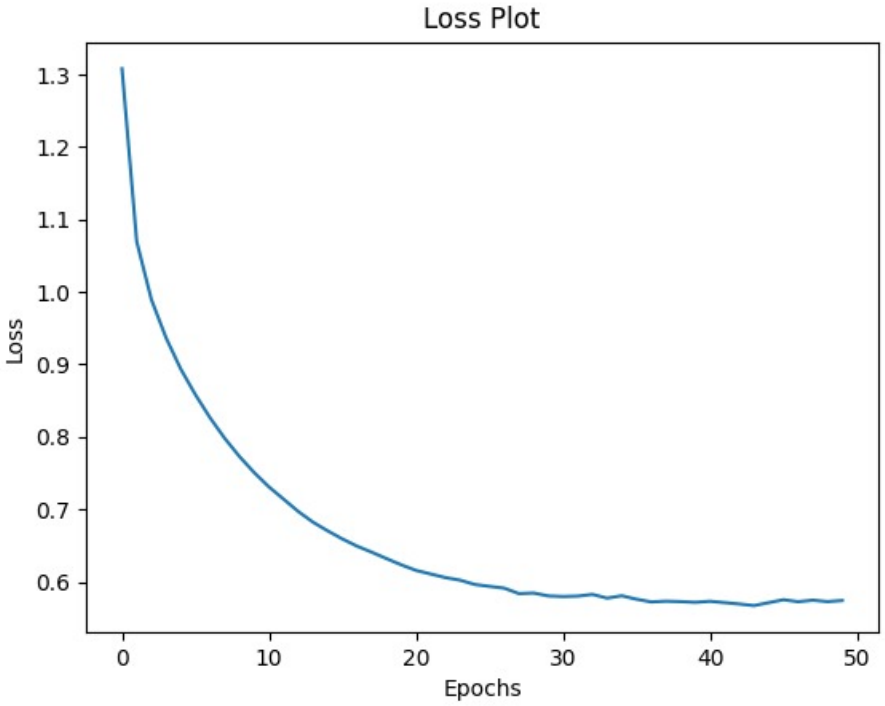
InceptionV3 - LSTM Luong Attention Loss Graph



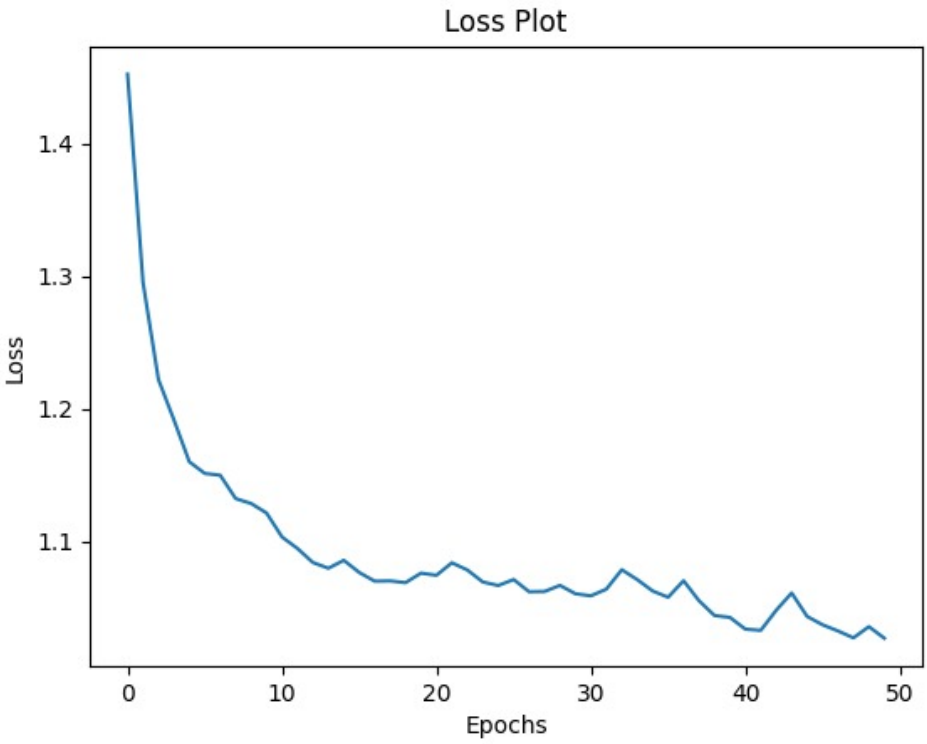
VGG16 - LSTM Luong Attention Loss Graph



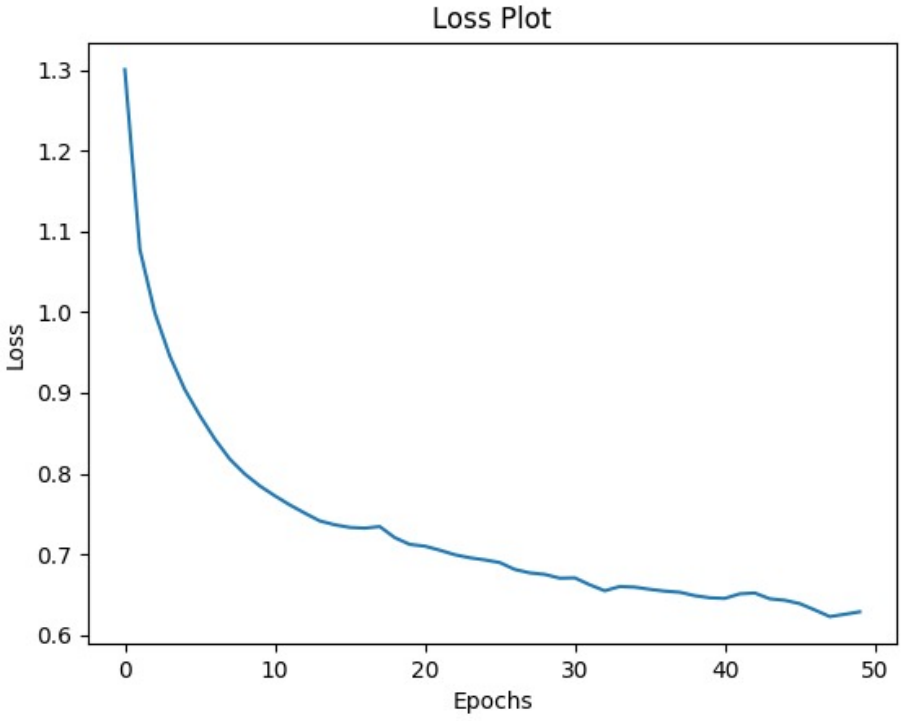
ResNet50 - LSTM Luong Attention Loss Graph



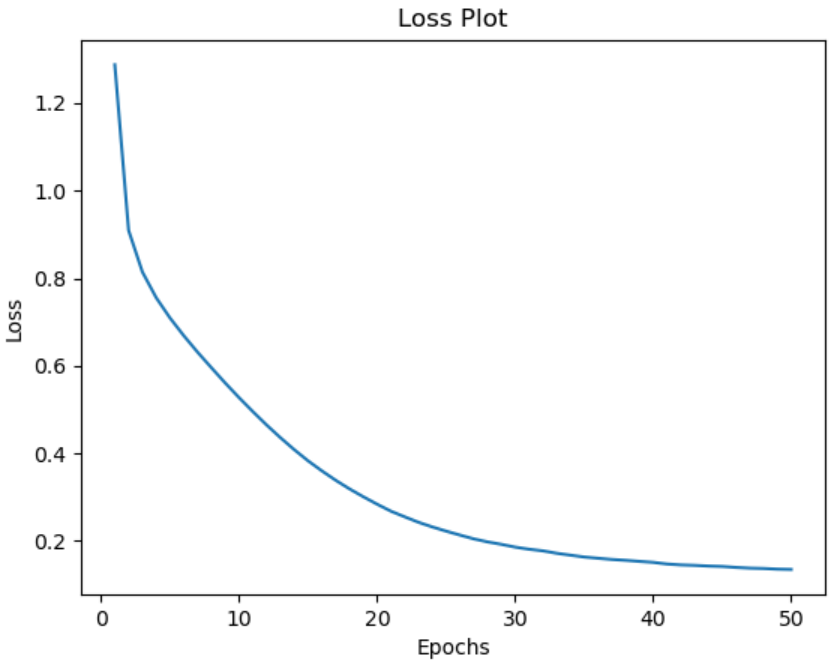
InceptionV3 - GRU Luong Attention Loss Graph



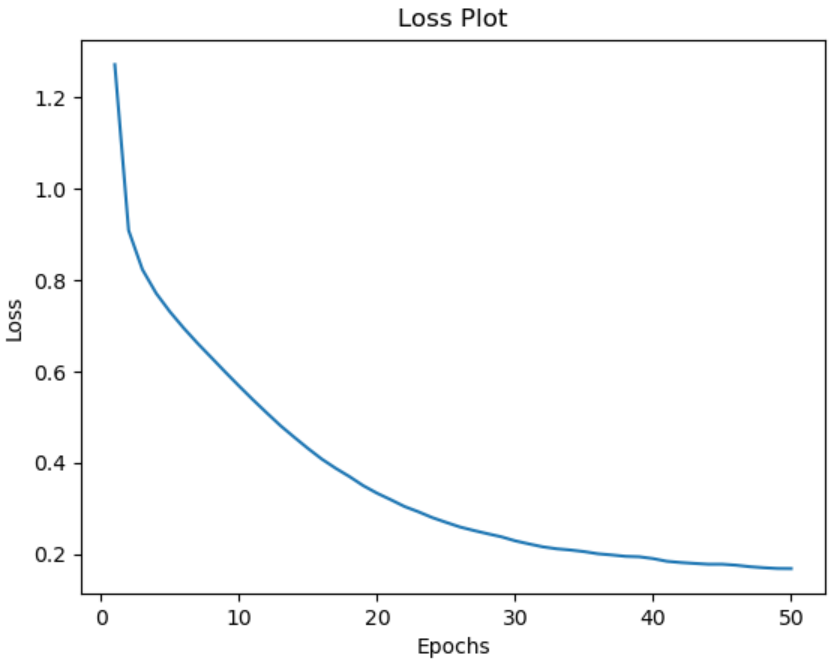
VGG16 - GRU Luong Attention Loss Graph



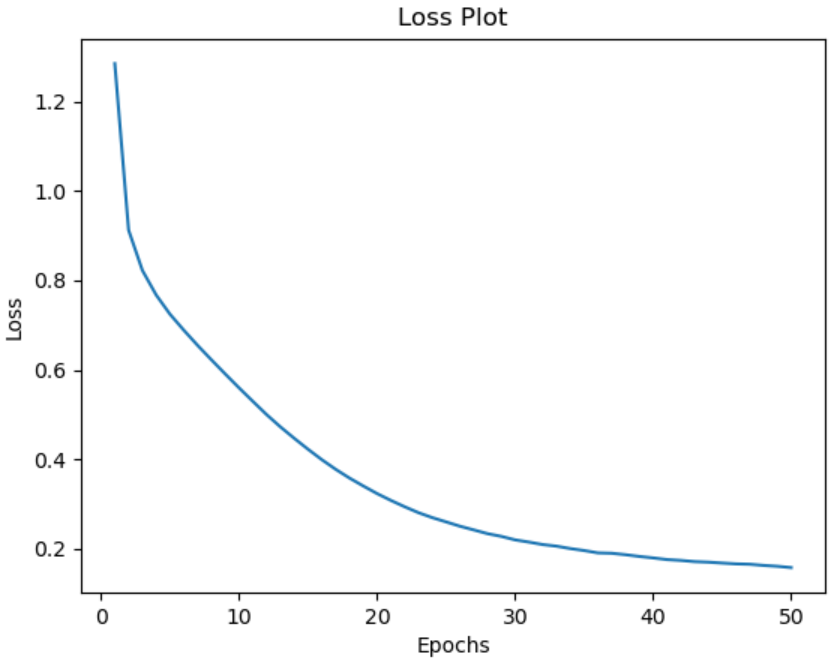
ResNet50 - GRU Luong Attention Loss Graph



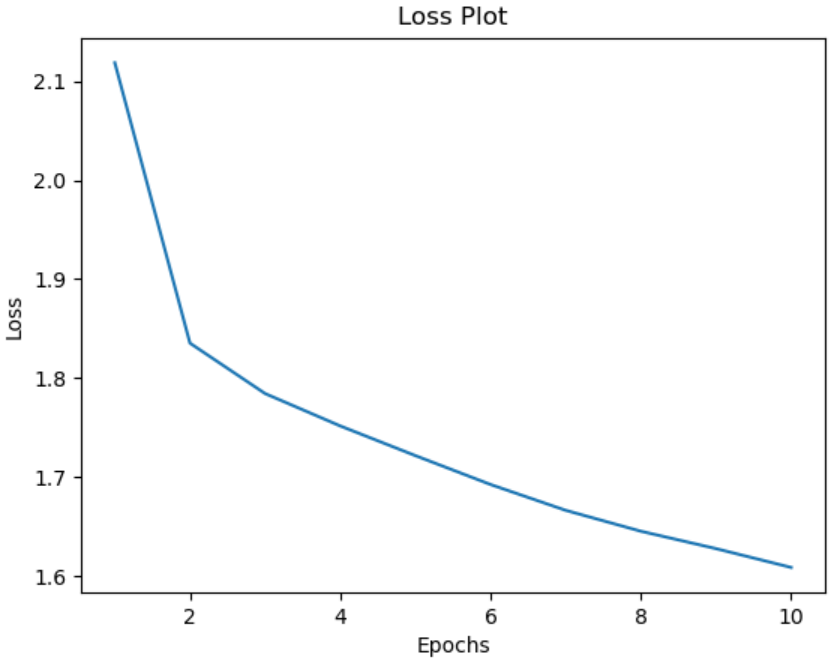
InceptionV3 - LSTM Loss Graph



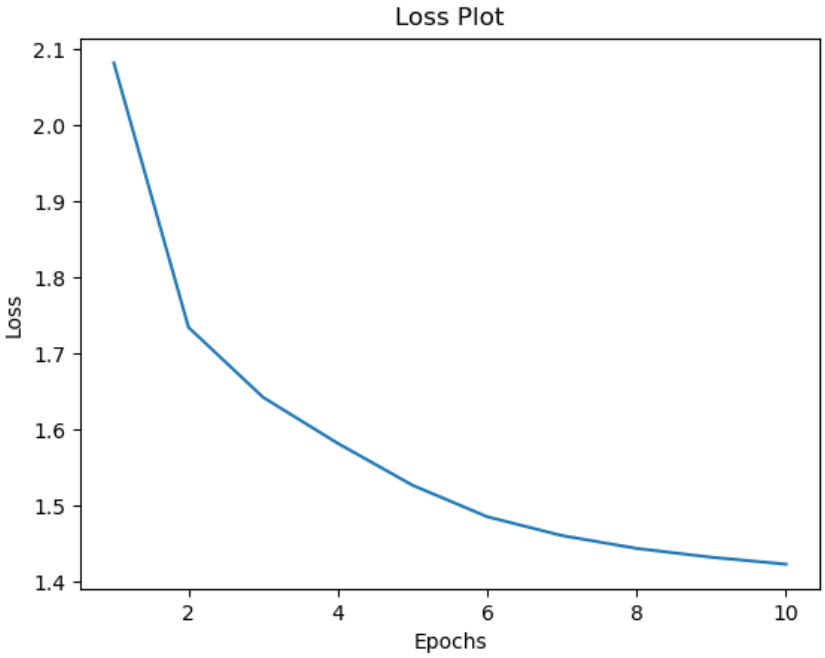
VGG16 - LSTM Loss Graph



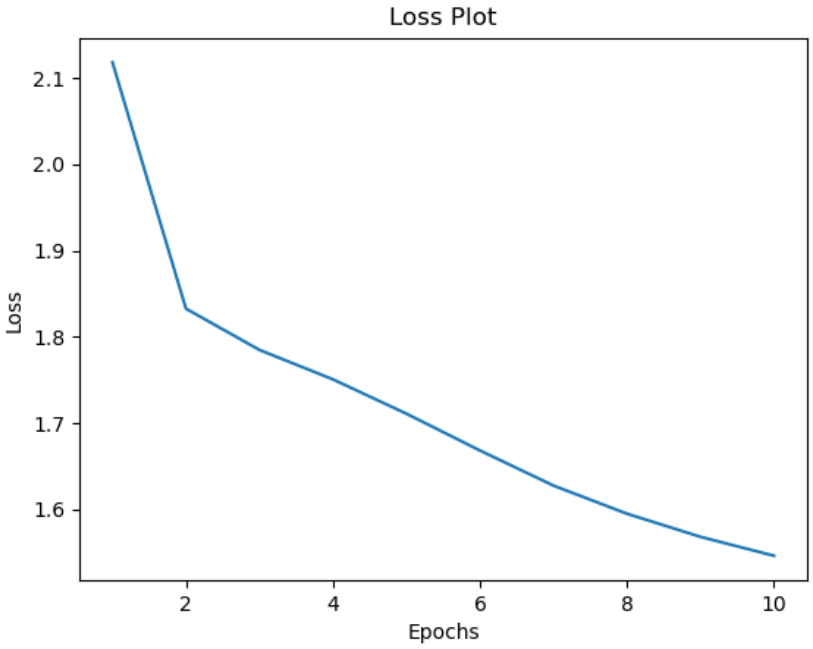
ResNet50 - LSTM Loss Graph



ResNet50 - Transformer Loss Graph



InceptionV3 - Transformer Loss Graph

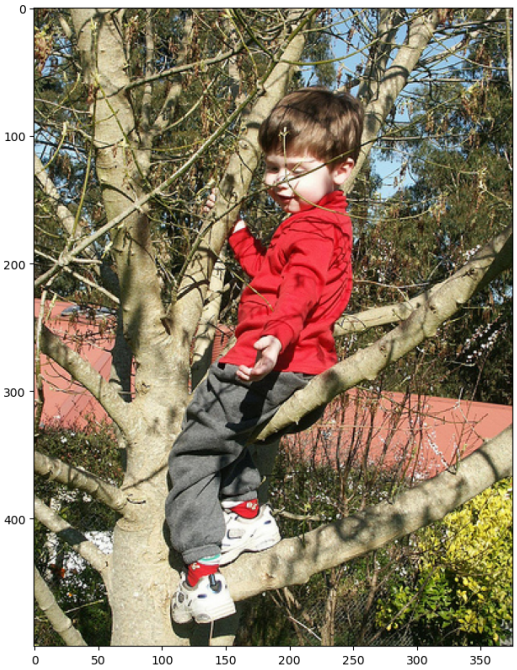


VGG16 - Transformer Loss Graph

Visual Results

Predicted Caption: “A person stands on a rock ledge overlooking the water”

(InceptionV3 Architecture Combined With LSTM and Luong Attention)



Predicted Caption: “a little boy is climbing on a red and white coat”

(ResNet50 Architecture Combined with GRU and Bahdanau Attention)

References

[1] "Inception V3 model, with weights pre-trained on ImageNet.," *TensorFlow*. [Online]. Available: https://tensorflow.rstudio.com/reference/keras/application_inception_v3/ . [Accessed: 03-Apr-2021].

[2] "VGG16 – Convolutional Network for Classification and Detection," *Neurohive.* [Online]. Available: https://neurohive.io/en/popular-networks/vgg16/. [Accessed: 15-May-2021].

[3] "Understanding and Coding a ResNet in Keras," *Towards Data Science.* [Online]. Available: https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33. [Accessed: 15-May-2021].

[4] M. Phi, "Illustrated Guide to LSTM's and GRU's: A step by step explanation," Medium, 28-Jun-2020. [Online]. Available: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21 [Accessed: 03-Apr-2021]

[5] A. Vaswani et. al., "Attention is All You Need," Arxiv, 12-June-2017. [Online]. Available: https://arxiv.org/abs/1706.03762. [Accessed: 14-May-2021].

[6] adityajn105, "Flickr 8k Dataset," *Kaggle*, 27-Apr-2020. [Online]. Available: https://www.kaggle.com/adityajn105/flickr8k. [Accessed: 03-Apr-2021].

[7] "IMAGENET," IMAGENET, [Online]. Available: http://www.image-net.org/index. [Accessed: 03-Apr-2021].

[8] colah, "Understanding LSTM Networks," *colah's blog,* 27-Aug-2015. [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/. [Accessed: 03-Apr-2021].

[9] D. Garg, "Automatic Image Captioning with PyTorch," *Medium,* 20-Aug-2020. [Online]. Available: https://medium.com/@deepeshrishu09/automatic-image-captioning-with-pytorch-cf576c98d319. [Accessed: 03-Apr-2021].

[10] S. Russell-Puleri, “Gated Recurrent Units explained using Matrices: Part 1,” *Medium*, 01-Dec-2020. [Online]. Available: https://towardsdatascience.com/gate-recurrent-units-explained-using-matrices-part-1-3c781469fc18. [Accessed: 04-Apr-2021].

[11] “Image Captioning using Attention Mechanism,” *Medium.* [Online]. Available: https://medium.com/swlh/image-captioning-using-attention-mechanism-f3d7fc96eb0e [Accessed: 15-May-2021].

[12] G. Loye, “Attention Mechanism,” FloydHub Blog, 17-Jan-2020. [Online]. Available: https://blog.floydhub.com/attention-mechanism/. [Accessed: 14-May-2021].

[13] Khandelwal, R., “BLEU-Bilingual Evaluation Understudy,” *Medium,* 26-Jan-2020. [Online]. Available: https://towardsdatascience.com/bleu-bilingual-evaluation-understudy-2b4eab9bcfd1. [Accessed: 14-May-2021].

[14] P. Joshi, “Transformers in NLP,” Analytics Vidhya, 19-June-2019. [Online]. Available: https://www.analyticsvidhya.com/blog/2019/06/understanding-transformers-nlp-state-of-the-art-models/. [Accessed: 14-May-2021].

[15] “Transformer-based image captioning extension of pytorch/fairseq,” PythonAwesome, 21-Jan-2020. [Online]. Available: https://pythonawesome.com/transformer-based-image-captioning-extension-of-pytorch-fairseq/. [Accessed: 14-May-2021].

[16] “Image captioning with visual attention : TensorFlow Core,” TensorFlow. [Online]. Available: https://www.tensorflow.org/tutorials/text/image_captioning. [Accessed: 05-Apr-2021].