International Institute of Information Technology
Bangalore

Visual Recognition
AI 825

Mini - Project

Technical Report

June 2, 2023

Contents

1 Introduction **1**

2 Encoder and Decoder Architecture **1**

3 Modified Baseline (Attention Mechanism) **2**

4 Results **5**

4.1 BLEU-4 Scores 7

4.2 Meteor Scores 8

4.3 Observations 8

5 Conclusion **8**

6 Team Members **9**

1 Introduction

The following report summarizes our progress and findings for the task of *Enhancing Image Captioning with a Modified Baseline model*.

The basic system, which serves as the foundation for our improvements, combines convolutional neural networks (CNNs) (Encoder) and long short-term memory (LSTM) (Decoder) into Bridge model. The CNN retrieves visual features from the input image, while the LSTM creates captions by processing visual and language embeddings. The specifications of the model used are provided in the next sections.

The Flickr8K dataset is used for training and testing the models. This dataset is a widely used benchmark dataset in the field of image captioning. It is designed for training and evaluating models that generate textual descriptions or captions for images. The dataset consists of a collection of 8,000 images gathered from the popular photo-sharing website Flickr.

Given the objective of the project to enhance the vision and language embeddings in the CNN-LSTM system, we have implemented the attention mechanism as a modification to our implemented baseline model.

To evaluate the performance of the models, objective metrics BLEU-4 (Bilingual Evaluation Understudy) and METEOR (Metric for Evaluation of Translation with Explicit ORdering) scores are used. The results includes a review of objective and subjective evaluation results, offering insights into the effectiveness of the implemented approaches.

2 Encoder and Decoder Architecture

As mentioned in the project document, the architecture of the models for the CNN Encoder and the LSTM architecture in the RNN decoder is retained even for the modified baseline, by accommodating the required changes in dimensionality of output of encoder and input to the decoder due to the addition of the attention mechanism.

Encoder: A `CNN_Encoder` class is defined using a pre-trained ResNet-50 model to extract image features and maps them to a lower-dimensional embedding space. The last linear and pooling layers are replaced with a linear layer to obtain the desired embedding size as we are not classifying images but passing a feature tensor onto the decoder. The images are passed through a transform function to resize them to match the input dimensions of the ResNet-50 model. The option to enable fine-tuning by training all weights of the CNN is also made available by using a flag `trainConvNet`.

Decoder: A `RNN` class is defined to generates captions based on the image features. The decoder class consists of an embedding layer, LSTM cells, and a linear layer for prediction. The CNN encoder's image feature tensor and training captions (word tokens) are sent into the decoder. The decoder sequentially interprets the input, providing predictions for each time step. The input embedding and the previous hidden state are fed into the LSTM cells, which create the next hidden state and cell state.

Bridge: The CNN Encoder and RNN Decoder are combined in the `Bridge` class. It accepts an image tensor and a caption tensor as input and then runs the image tensor through the CNN Encoder to retrieve image features. These features, together with the caption tensor, are then sent to the RNN Decoder. The `Bridge` class has a method `caption_image` for generating captions as well. During training, it computes the RNN Decoder predictions and during caption generation, the predicted word at each time step is used as input for the next time step until the end-of-sequence token (`<EOS>`) is generated.

3 Modified Baseline (Attention Mechanism)

The modification chosen for the baseline is the attention mechanism. This is a component used in deep learning models that helps the model focus on specific parts of the input data while generating outputs. It is particularly useful in tasks where the model needs to attend to different parts of the input dynamically, assigning varying levels of importance to different elements. In the context of image captioning, the attention mechanism allows the model to selectively attend to different regions of an image while generating a textual description or caption. This is important because images can contain multiple objects or regions of interest, and the model needs to understand which parts of the image are most relevant to each word or phrase in the caption.

Implementation of attention is done by modifications to the architecture as follows:

Encoder: Same architecture used as in baseline. Class is named `CNN_Encoder_Att`.

Attention: We create an Attention class that inherits from `nn.Module`, the base class for neural network modules in PyTorch. The class has three linear layers: `dec_att`, `enc_att` and `full_att` which map the decoder hidden state, the encoder output, and the attention vector to the same size (`att_dim`). The class has a forward method that takes two arguments: features and hidden-state, which are the encoder output and the decoder hidden state at a certain time step. The forward method applies the linear layers self.U and self.W to the features and the hidden state, and adds them element-wise. It then applies a `relu` function to get the combined-states tensor, which shows how well the encoder output and the decoder hidden state match. The forward method applies the linear layer `full_att` to the combined-states tensor to get the attention-scores tensor, which shows how important each encoder output is for generating the next word. The forward method squeezes and softmaxes the attention-scores tensor to get the `alpha` tensor, which shows the normalized attention weights for each encoder output. The forward method multiplies and sums the `features` tensor and the `alpha` tensor to get the attention-weights tensor, which shows the weighted average of the encoder output based on the attention weights.

Decoder: The only difference in this LSTM architecture is the additional attention layer. The code defines a class called `RNN_Att` that implements an attention decoder for image captioning. The class defines several layers, such as `self.embed`, `self.attention`, `self.init_h`, `self.init_c`, `self.lstm_cell`, and `self.fc`, which are used to embed the captions, compute the attention weights and context vector, initialize the LSTM state, scale the context vector, update the LSTM state, and generate the output words. The forward method embeds the captions, initializes the LSTM state, and iterates over the caption sequence. At each time step, it computes the attention weights and context vector using the `self.attention` layer, concatenates the word embedding and the context vector as the input to the LSTM cell, updates the LSTM state using the `self.lstm_cell` layer, generates the output word using the `self.fc` layer, and stores the predictions and alphas. The class also has an `init_hidden_state` method that takes one argument: encoder output. The init-hidden-state method computes the mean of the encoder output along the first dimension and uses it to initialize the LSTM hidden state and cell state.

Bridge: Same architecture is used as in baselines except in the `caption_image` method where the process similar to the `forward` method in `RNN_Att` is followed to generate captions for provided image. Class is named `Bridge_Att`.

The visualization of the attention mechanism is shown in the images below:

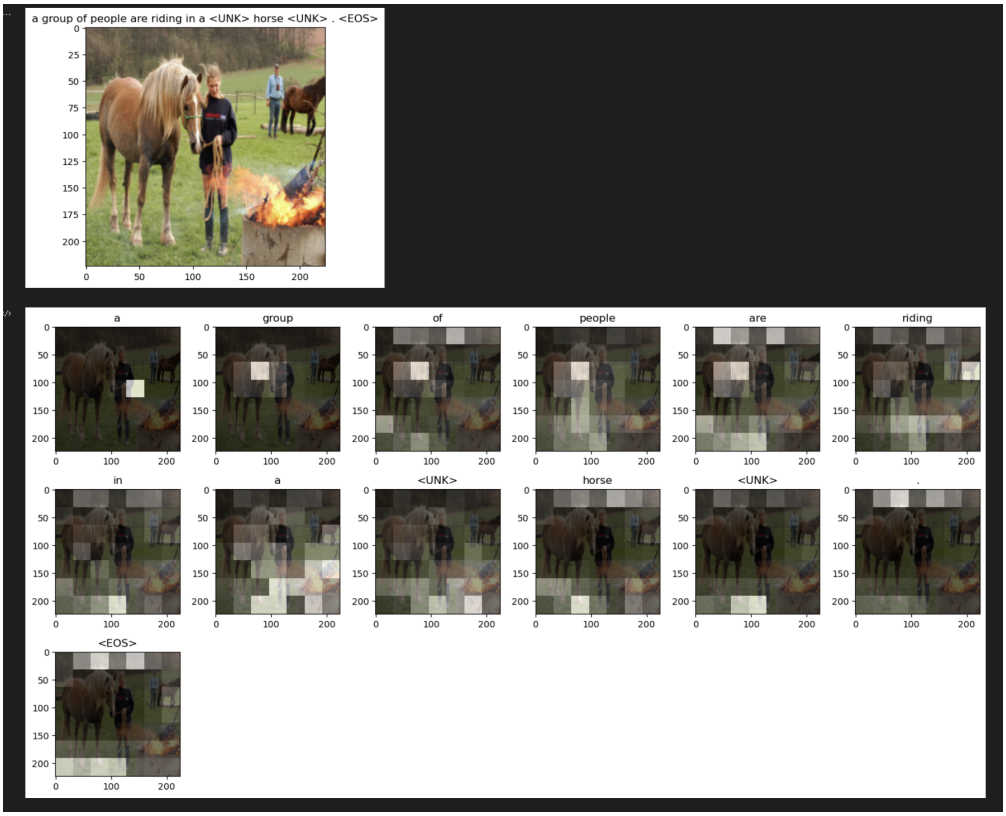

Figure 1: Example 1

Figure 2: Example 2

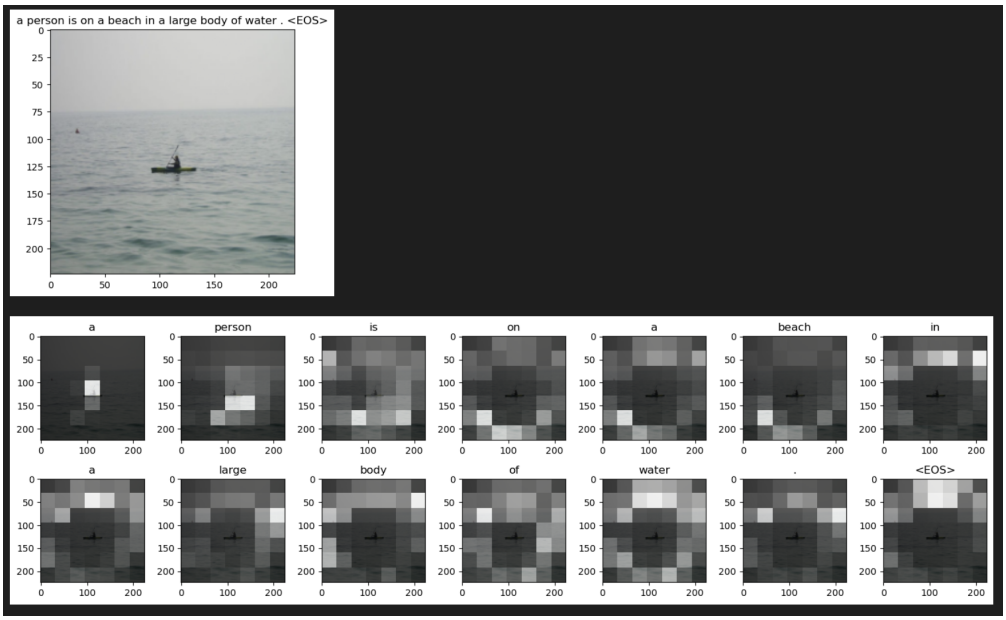

Figure 3: Example 3

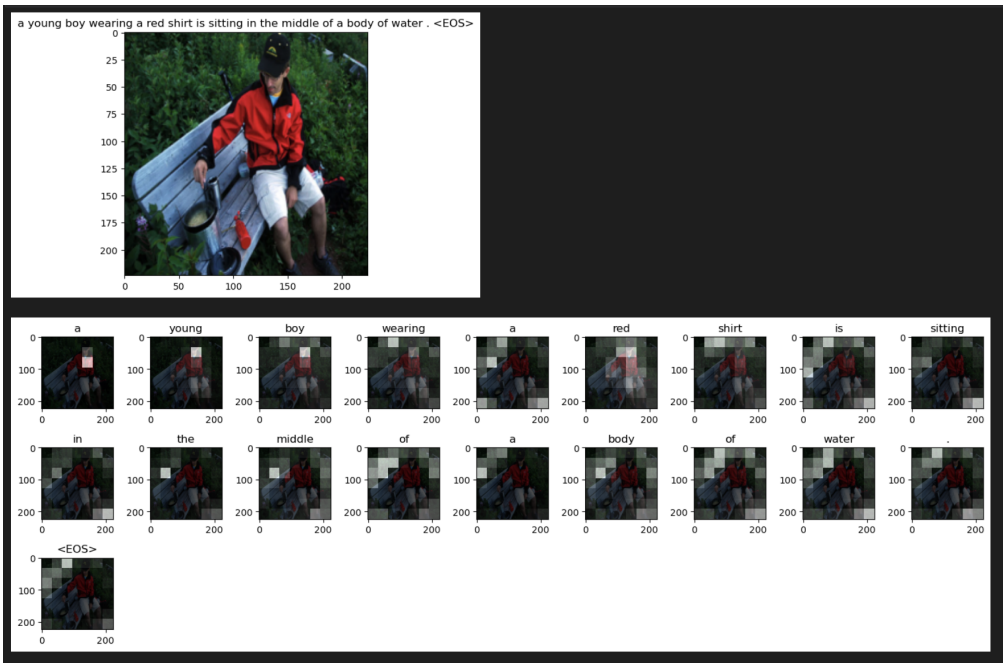

Figure 4: Example 4

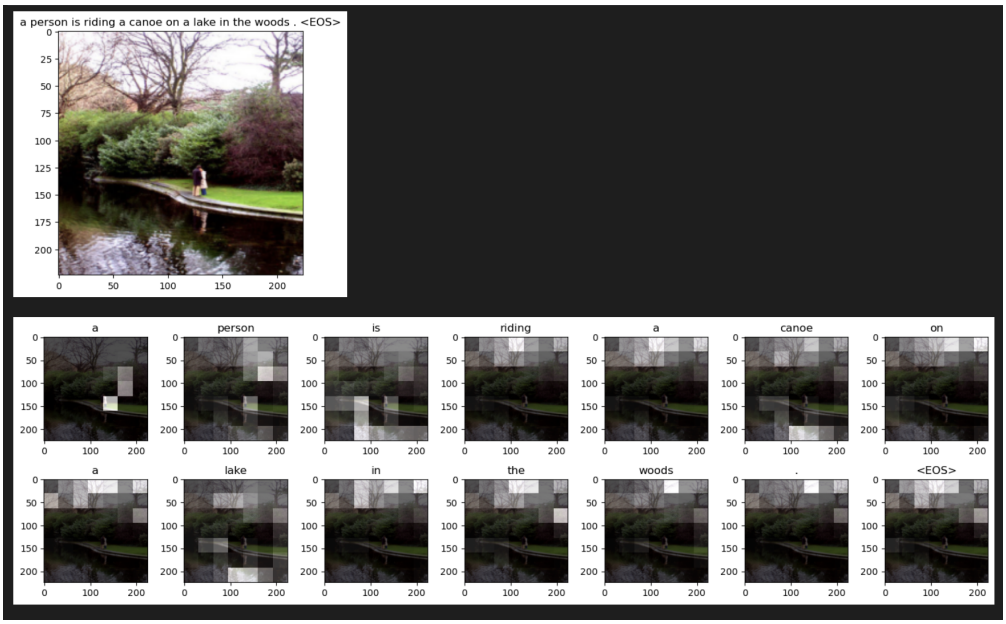

Figure 5: Example 5

4 Results

Each result is generated based on 4 reference samples against one output caption generated by our model. 5 test example images were used to evaluate both our models. The example images chosen are as shown below:

Figure 6: Example 1

Figure 7: Example 2

Figure 8: Example 3

Figure 9: Example 4

Figure 10: Example 5

4.1 BLEU-4 Scores

The BLEU-4 score measures the similarity between a machine-generated translation and a human reference translation based on n-gram matches. To calculate the score, n-grams in the machine-generated translation are counted for n=1,2,3,4 and compared to the maximum count for each n-gram type in any single reference translation to avoid favoring translations that closely match a particular reference translation. The total counts for each n-gram type in the machine-generated translation are clipped to their respective maximum reference counts to avoid overestimating the score when the machine-generated translation contains more n-grams than the reference translations. A brevity penalty is then calculated to penalize translations that are shorter than the reference translations, and the BLEU-4 score is calculated by taking the geometric mean of the clipped n-gram counts, weighted by their respective n-gram weights, and multiplied by the brevity penalty.

The BLEU-4 score ranges from 0 to 1, with higher scores indicating better translations.

Table 1: BLEU-4 Scores

	Example 1	**Example 2**	**Example 3**	**Example 4**	**Example 5**
Without Attention	0.80706	0.75983	0.86117	0.45746	0.72598
With Attention	3.61924 e-78	0.86334	0.85939	0.91217	1.31655 e-231

4.2 Meteor Scores

The METEOR score measures the quality of a machine-generated translation by comparing its bag-of-words representation to that of one or more reference translations. First, unigram precision and recall are calculated and then combined using a weighted harmonic mean to give a single unigram score. An alignment-based penalty is then calculated by comparing the number of chunks in the machine-generated translation that appear in the reference translations to the total number of chunks. The penalty score is calculated using an F-mean function, which is a weighted harmonic mean of precision and recall. Finally, the METEOR score is computed as a weighted sum of the unigram score and the alignment-based penalty score. The weights used can be adjusted to emphasize precision or recall.
The METEOR score ranges from 0 to 1, with higher scores indicating better translations.

Table 2: Meteor Scores

	Example 1	**Example 2**	**Example 3**	**Example 4**	**Example 5**
Without Attention	0.97990	0.97465	0.98573	0.62284	0.97080
With Attention	0.50223	0.98668	0.93491	0.88480	0.10869

4.3 Observations

We can draw the following observations from our test scores obtained with the 2 models:

- For Example 2 and Example 4 if we look at both the BLEU-4 and Meteor scores, our model With Attention mechanism performs significantly better than our model without Attention mechanism.
- For Example 3 both the models(with attention and without attention) are giving similar BLEU-4 and Meteor scores.
- For Example 1 and Example 5, our model without Attention mechanism is performing significantly better than our model with attention mechanism in both the BLEU-4 and Meteor score metrics. The reason for this might be the fact that the attention weight may not have adjusted to detect the particular word/token in the caption for example 1. For example 5, since the image has a plain uniform background and the objects of interest are small, attention may not prove to be useful.

The following link contains the checkpoint files for the saved models - Link to sharepoint.

5 Conclusion

From the above results we can conclude that the ability of the attention module to focus on different parts of the image while generating words for the caption allows for significantly better captioning

in certain cases where the image is focused more on the objects present in it. It is also observed that attention may, in some cases, not provide significant improvements or may even lead to worse results, such as for example images 1 and 5. But for images where the objects of interest are in focus in the image, the attention mechanism performs significantly well as shown in examples 2 and 4.

6 Team Members

Name	**Roll number**	**Email**
Shreyansh Rai	IMT2020501	shreyansh.rai@iiitb.ac.in
Aamod B K	IMT2020546	aamod.bk@iiitb.ac.in
Saket Gurjar	IMT2020520	saket.gurjar@iiitb.ac.in
Vatsal Dhama	IMT2020029	vatsal.dhama@iiitb.ac.in