# AI 825 – Visual Recognition

## Mini Project Report
## CNN + LSTM Image Captioning

## GROUP-23

IMT2020051 – Naititksinh Solanki
IMT2020081 – Arin Awasthi
IMT2020084 – Arya Kondawar

## Aim of the Project

To design a CNN-LSTM network that performs Image Captioning on Flickr8k Dataset.

## About Flick8k Dataset

For testing, validation, and training, we used the Flickr dataset. Each of the 8,000 images in the collection has a caption that describes the significant people, places, and events in one of five different ways. The images were hand-selected from six different Flickr groups and don't include any famous people or locations, but they do show a variety of situations and events.

## Preprocessing

Text Preprocessing:

Despite having access to the lemmatized text descriptions along with the dataset, we preferred the original descriptions. Lemmatized output captions are produced via lemmatized training, which lowers the bleu score. Therefore, we went with the original descriptions. We now cleaned up these original texts using the text preparation techniques listed below.

Image Processing:

Since we have used Resnet 50 CNN architecture to extract visual features from the images, hence we need to perform some basic transformations to the images before giving them as input to our architecture, like:

- Resizing the image to (224, 224)
- Added an extra dimension since the CNN expects batches of images as input rather than a single image.

Word Embeddings:

We used one-hot enocding in baselined model and glove embeddings in modified model. We use here is just a pretrained version of this model for around 6 billion tokens called the GloVe6b. We used a particular variant which has 200 feature embeddings for every word. We needed to find meaningful word embedding hence we used GloVe embedding because it boosts generalization and performance by representing words as semantically-meaningful i.e learn the required spatial relations between words by modifying these vectors along with reducing the dimensionality of the vocabulary.

Building Vocabulary:

We build a dictionary for all the words present in the dataset for each train, test, and eval separately. We store 2 maps where each word is given an index and the reverse mapping corresponding to each index. This helps to create some sort of label encoding. However, in addition to the words present in the dataset we have introduced 2 different tokens:

- startseq, indicates the start of the caption.
- endseq, indicates the end of the caption.

Building Dataset:

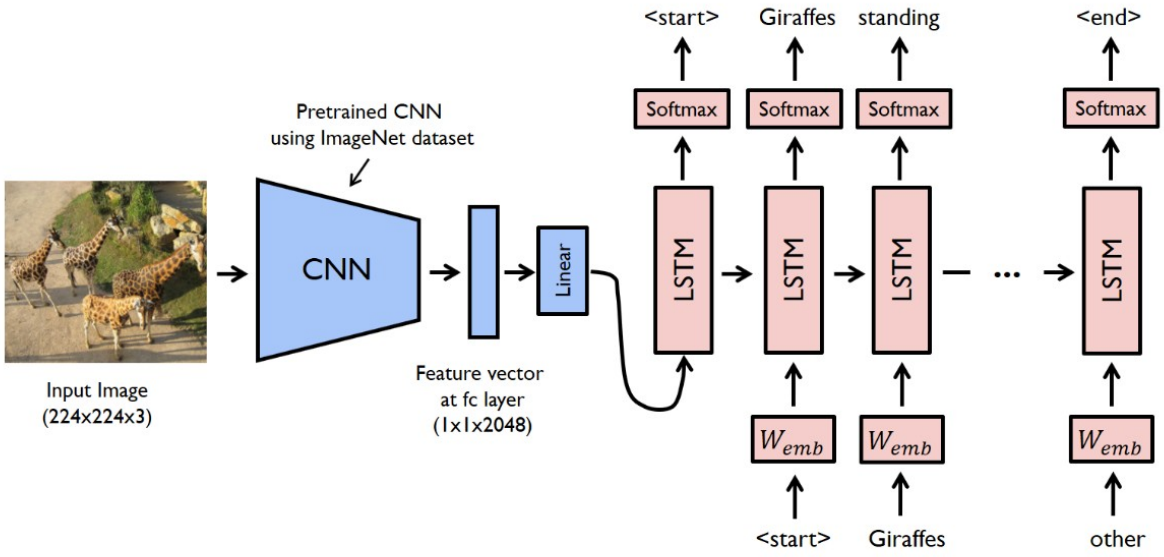
For baseline we have used a dataframe to store the filename of the image and the corresponding caption. This was replicated for all three train, test, and eval set given to us in the dataset.

## Approaches:

- CNN + LSTM baseline with DenseNet201 CNN and Label encoded text data.(Baseline)
- CNN + LSTM with ResNet50 CNN and pre-trained GloVe embedddings.(Modified 1)
- CNN + LSTM with ResNet50 CNN and trained on pre-trained GloVe embedddings.(Modified 2)

- CNN + LSTM with ResNet50 CNN, single attention head and pre-trained GloVe embedddings.(Modified 3)
- CNN + LSTM with ResNet50 CNN, single attention head and trained on top of pre-trained GloVe embedddings.(Modified 4)
- CNN + LSTM with ResNet50 CNN, multiple attention head and pre-trained GloVe embedddings.(Modified 5)
- CNN + LSTM with ResNet50 CNN, multiple attention head and trained on top of pre-trained GloVe embedddings. (Modified 6)

## Architecture Details:




Representation of the model architecture

### Encoder CNN:

We utilised ResNet50 as a feature extractor and DenseNet201 as a feature extractor baseline model, these enables us to obtain the visual characteristics for input photos with a size of 224x224x3. For the reasons listed below, we utilised Resnet 50:

- Depth: ResNet-50 is a 50-layer deep neural network that can extract more intricate characteristics from photos. ResNet-50 can learn hierarchical representations of pictures thanks to its depth.

- Pre-trained weights: A feature extractor that uses pre-trained weights can help a model become more accurate and effective.
- Performance: The forward pass per image using resnet is less than other networks like inceptionv3 and VGG and hence was a clear choice as we had limited time to train the model and couldn't allocate much time for the forward pass of the CNN.

Decoder LSTM architecture:

For all the models we used Categorical Cross Entropy loss for updating the parameters. Also, we used Adam optimizer with a constant learning rate. We trained for around 15 epochs and observed that the loss almost converged within those epochs.

Baselined:

The inputs to the model are the preprocessed image (224, 244, 3) and the integer tokens of the captions. The image's features are extracted (we have already done this) and reduced to 256 dimensions using a Linear layer with ReLU activation. Word embeddings are label encoded and are expanded to 256 dimensions. After getting both the word and Image embeddings we pass it through the LSTM and then apply various dropouts, dense layers and finally softmax.

Caption maximum length computation:

We now determine the longest possible caption. This aids in determining the training input size and guarantees that each caption is a consistent length for more accurate calculations. Additionally, being aware of the maximum length aids in determining the amount of padding or truncation point as well as aids in avoiding the creation of excessively long captions. As a result, anytime a caption size bigger than the current max was met, the max value was updated to determine the maximum caption length using a straightforward for loop.

Calculating vocabulary size:

Next, we determine the size and vocabulary of our dataset.

The vocabulary size is the total number of different or unique words found in all of the dataset's captions. A better, more varied dataset is indicated by a greater vocabulary size. We can calculate the word embedding matrix that we shall see below with the aid of the vocabulary size. We repeatedly go through the pictures in the dictionary to see if every word in the caption has already been used or not if no, then the word gets added to a list called vocab_list. The vocab_size is given by len(idxtoword1) + 1 (or) len(vocab_list).

BUILDING THE DATALOADER:

In order to create batch_size number of training examples at once, we built a function named data_generator() that accepts the picture features, caption, and batch_size. Thus, this dataloader will provide quicker data access during training, assist in parallelizing data loading to optimise memory utilisation, utilise several GPUs, and decrease the amount of time spent waiting for data.

We use these below in all the architectures.

| Layer (type) | Output Shape | Param # | Connected to |
|---|---|---|---|
| input_1 (InputLayer) | [(None, 1920)] | 0 | [] |
| dense (Dense) | (None, 256) | 491776 | ['input_1[0][0]'] |
| input_2 (InputLayer) | [(None, 34)] | 0 | [] |
| reshape (Reshape) | (None, 1, 256) | 0 | ['dense[0][0]'] |
| embedding (Embedding) | (None, 34, 256) | 1885696 | ['input_2[0][0]'] |
| concatenate (Concatenate) | (None, 35, 256) | 0 | ['reshape[0][0]', 'embedding[0][0]'] |
| lstm (LSTM) | (None, 256) | 525312 | ['concatenate[0][0]'] |
| dropout (Dropout) | (None, 256) | 0 | ['lstm[0][0]'] |
| add (Add) | (None, 256) | 0 | ['dropout[0][0]', 'dense[0][0]'] |
| dense_1 (Dense) | (None, 128) | 32896 | ['add[0][0]'] |
| dropout_1 (Dropout) | (None, 128) | 0 | ['dense_1[0][0]'] |
| dense_2 (Dense) | (None, 7366) | 950214 | ['dropout_1[0][0]'] |

CNN + LSTM with ResNet50 and GloVe embeddings:

We have a similar architecture over here to that of baselined just the change is we have used GloVe embedding for word

embeddings and the word embeddinngs dimensions have been changed to 200 and the image embedding dimensions have also been reduced to 200. After passing the concatenated features through LSTM we added it with image embeddings.

WORD EMBEDDINGS GENERATION:

As we move forward, word embeddings will be generated. One-hot encodings need a lot of memory and aren't appropriate for huge vocabulary sets. We make advantage of GloVe embeddings, which provide dense vector representations and bridge words according to how frequently they appear concurrently. Each word has an embedded fixed dimension with length determined by the vocabulary capacity. Each line of the text file we prepare for GloVe embedding contains the word itself followed by its 200 dimensional embedding. Next, we add the relevant embedding for each word in the glove embedding to a glossary. This will be used below in all the architectures with GloVe.

```
input_4 (InputLayer)           [(None, 2048)]       0           []

dropout_3 (Dropout)            (None, 2048)         0           ['input_4[0][0]']

dense_3 (Dense)                (None, 200)          409800      ['dropout_3[0][0]']

input_5 (InputLayer)           [(None, 36)]         0           []

reshape_1 (Reshape)            (None, 1, 200)       0           ['dense_3[0][0]']

embedding_1 (Embedding)        (None, 36, 200)      1455400     ['input_5[0][0]']

concatenate_1 (Concatenate)    (None, 37, 200)      0           ['reshape_1[0][0]',
                                                                 'embedding_1[0][0]']

lstm_2 (LSTM)                  (None, 200)          320800      ['concatenate_1[0][0]']

dropout_4 (Dropout)            (None, 200)          0           ['lstm_2[0][0]']

add_1 (Add)                    (None, 200)          0           ['dense_3[0][0]',
                                                                 'dropout_4[0][0]']
...
```

Now we used attention, before looking at the architectures we will see what is attention and how it is applied.

Attention Mechanism:

We use Query, Key and Value as learnt in class. The algorithm is specifically called as the Bahdanau Attention and is as follows:

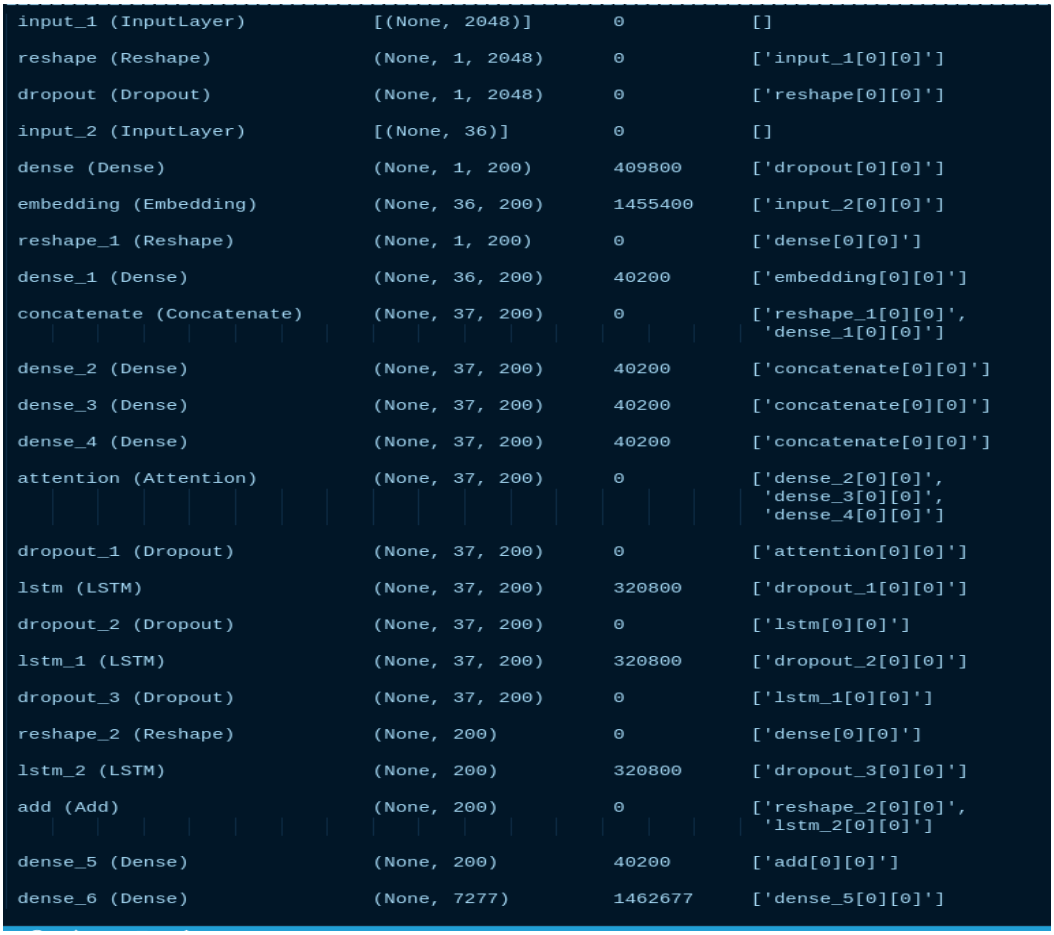
- The encoder generates a set of annotations, $h_i$, from the input sentence.
- These annotations are fed to an alignment model and the previous hidden decoder state. The alignment model uses this information to generate the attention scores, $e_{t,i}$.
- A softmax function is applied to the attention scores, effectively normalizing them into weight values, $\alpha_{t,i}$, in a range between 0 and 1.
- Together with the previously computed annotations, these weights are used to generate a context vector,$c_t$, through a weighted sum of the annotations. That is:

$$c_t = \Sigma^{T}_{i=1} \alpha_{t,i} h_i$$

- The context vector is fed to the decoder together with the previous hidden decoder state and the previous output to compute the final output, $y_t$.
- Above steps are repeated until the end of the sequence.

CNN + LSTM with ResNet50 and GloVe embeddings and single attention head:

Here the architecture is same as above until concatenation, after concatenation we create query, value and key vectors which takes the conctenated feature as an input and reduces it to 200 dimension, after creating them respectively we pass them through an Attention Layer(in Keras) and then pass this output to various LSTMs.



| | | | |
|---|---|---|---|
| input_1 (InputLayer) | [(None, 2048)] | 0 | [] |
| reshape (Reshape) | (None, 1, 2048) | 0 | ['input_1[0][0]'] |
| dropout (Dropout) | (None, 1, 2048) | 0 | ['reshape[0][0]'] |
| input_2 (InputLayer) | [(None, 36)] | 0 | [] |
| dense (Dense) | (None, 1, 200) | 409800 | ['dropout[0][0]'] |
| embedding (Embedding) | (None, 36, 200) | 1455400 | ['input_2[0][0]'] |
| reshape_1 (Reshape) | (None, 1, 200) | 0 | ['dense[0][0]'] |
| dense_1 (Dense) | (None, 36, 200) | 40200 | ['embedding[0][0]'] |
| concatenate (Concatenate) | (None, 37, 200) | 0 | ['reshape_1[0][0]', 'dense_1[0][0]'] |
| dense_2 (Dense) | (None, 37, 200) | 40200 | ['concatenate[0][0]'] |
| dense_3 (Dense) | (None, 37, 200) | 40200 | ['concatenate[0][0]'] |
| dense_4 (Dense) | (None, 37, 200) | 40200 | ['concatenate[0][0]'] |
| attention (Attention) | (None, 37, 200) | 0 | ['dense_2[0][0]', 'dense_3[0][0]', 'dense_4[0][0]'] |
| dropout_1 (Dropout) | (None, 37, 200) | 0 | ['attention[0][0]'] |
| lstm (LSTM) | (None, 37, 200) | 320800 | ['dropout_1[0][0]'] |
| dropout_2 (Dropout) | (None, 37, 200) | 0 | ['lstm[0][0]'] |
| lstm_1 (LSTM) | (None, 37, 200) | 320800 | ['dropout_2[0][0]'] |
| dropout_3 (Dropout) | (None, 37, 200) | 0 | ['lstm_1[0][0]'] |
| reshape_2 (Reshape) | (None, 200) | 0 | ['dense[0][0]'] |
| lstm_2 (LSTM) | (None, 200) | 320800 | ['dropout_3[0][0]'] |
| add (Add) | (None, 200) | 0 | ['reshape_2[0][0]', 'lstm_2[0][0]'] |
| dense_5 (Dense) | (None, 200) | 40200 | ['add[0][0]'] |
| dense_6 (Dense) | (None, 7277) | 1462677 | ['dense_5[0][0]'] |

CNN + LSTM with ResNet50 and GloVe embeddings and multiple attention head:

We have a similar architecture but here we have used MultiAttentionHead with 8 number of heads and dimension of key to be 64.

```
Layer (type)                 Output Shape              Param #   Connected to
==================================================================================================
input_12 (InputLayer)        [(None, 2048)]            0         []

dropout_9 (Dropout)          (None, 2048)              0         ['input_12[0][0]']

input_13 (InputLayer)        [(None, 36)]              0         []

dense_13 (Dense)             (None, 200)               409800    ['dropout_9[0][0]']

embedding_5 (Embedding)      (None, 36, 200)           1455400   ['input_13[0][0]']

reshape_5 (Reshape)          (None, 1, 200)            0         ['dense_13[0][0]']

lstm_9 (LSTM)                (None, 36, 200)           320800    ['embedding_5[0][0]']

multi_head_attention_2 (MultiH  (None, 1, 200)         25896     ['reshape_5[0][0]',
eadAttention)                                                     'lstm_9[0][0]']

concatenate_4 (Concatenate)  (None, 1, 400)            0         ['multi_head_attention_2[0][0]',
                                                                  'reshape_5[0][0]']

lstm_10 (LSTM)               (None, 200)               480800    ['concatenate_4[0][0]']

dropout_10 (Dropout)         (None, 200)               0         ['lstm_10[0][0]']

add_4 (Add)                  (None, 200)               0         ['dense_13[0][0]',
                                                                  'dropout_10[0][0]']

dense_14 (Dense)             (None, 200)               40200     ['add_4[0][0]']

dense_15 (Dense)             (None, 7277)              1462677   ['dense_14[0][0]']

==================================================================================================
Total params: 4,195,573
Trainable params: 2,740,173
Non-trainable params: 1,455,400
```

In the above we have used the same architectures for both pre-trained GloVe and trained upon pre-trained GloVe. Here we create a randomweight matrix which is used to extract embeddings and then we check in our vocabulary if the word in vocabulary is present in glove we replace it with glove's embeddings else leave it random, we tried to enhance the ones which weren't present in GloVe by training them.

## Evaluation Metric:

The 2 metrics mentioned to us were the BLEU score and METEOR Score

### Bleu Score:

BLEU stands for bilingual evaluation understudy in its full form. BLEU will generate a number between 0 and 1. The score represents how similar the given text is to the reference text, with values closer to 1 indicating more comparable texts. In actuality, a perfect score is impossible to get because a translation must precisely match the reference. Human translators are incapable of accomplishing this. The sentence_bleu() function in NLTK is used to compare a candidate sentence to one or more reference sentences. However, on NLTK Sentence Bleu by default gives BLEU-4 with weights = (0.25,0.25,0.25,0.25) and we have considered that to be our metric. There is also a corpus bleu available that rates the corpus but for our task, it didn't hold much relevance and we felt that sentence bleu can give a much more accurate view of how our model is performing.

Meteor Score:
The Meteor score is based on a combination of precision, recall, and unigram matching between the machine-translated output and a set of reference translations. It also takes into account semantic and syntactic information by using WordNet-based synonymy and paraphrase matching. The Meteor score ranges from 0 to 1, with higher scores indicating better machine translation quality.

Since the meteor score uses unigram we can already foresee that even for a good BLEU-4 score (default for sentence BLEU), the Meteor score will be very low as compared to it and later we see that, that is exactly what we observed.

## Results:

| Model | BLEU-4 | BLEU-3 | BLEU-2 | BLEU-1 | Meteor |
|---|---|---|---|---|---|
| Baseline | 0.4633 | 0.3156 | 0.1853 | 0.1024 | 0.312563 |
| Modified-1 | 0.5131 | 0.3276 | 0.2011 | 0.1246 | 0.330673 |
| Modified-2 | 0.4908 | 0.3407 | 0.1905 | 0.1373 | 0.330667 |
| Modified-3 | 0.5438 | 0.3436 | 0.2096 | 0.1253 | 0.331017 |
| Modified-4 | 0.4602 | 0.3067 | 0.1773 | 0.0843 | 0.291690 |
| Modified-5 | 0.4870 | 0.3325 | 0.2041 | 0.1254 | 0.3555861 |
| Modified-6 | 0.4870 | 0.3325 | 0.2041 | 0.1254 | 0.3555861 |

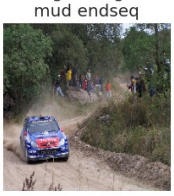
## Average Sentence BLEU scores:

| Model | Avg. BLEU Score |
|---|---|
| Baseline | 0.50527 |
| Modified-1 | 0.52641 |
| Modified-2 | 0.52257 |
| Modified-3 | 0.54327 |
| Modified-4 | 0.48766 |
| Modified-5 | 0.54539 |
| Modified-6 | 0.54539 |

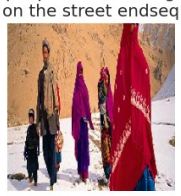
## Some Predictions:

### 1. Baseline:

startseq the car is driving through the mud endseq



startseq group of people are standing on the street endseq

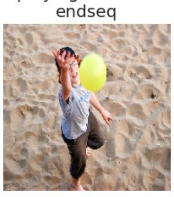


startseq two young girls are playing on the grass endseq

startseq two children are playing in the grass endseq

startseq two people are sitting on the beach endseq

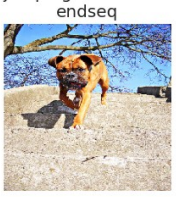
startseq the man is playing with ball endseq



startseq boy in red shirt is playing in the street endseq

startseq little boy is sitting on bed endseq

startseq dog is jumping into the air endseq



startseq black dog is running through the grass endseq

startseq black and white dog is playing with toy endseq

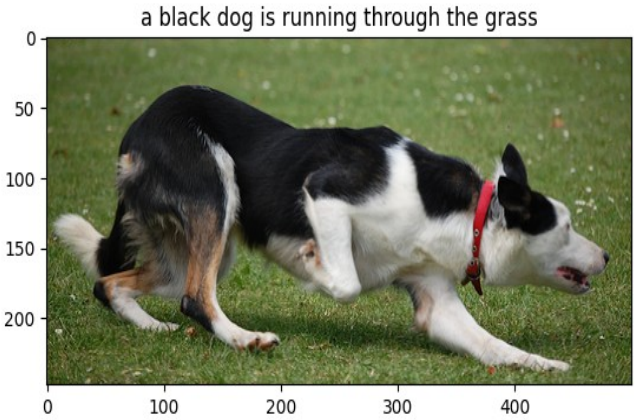
startseq man is jumping over the grass endseq

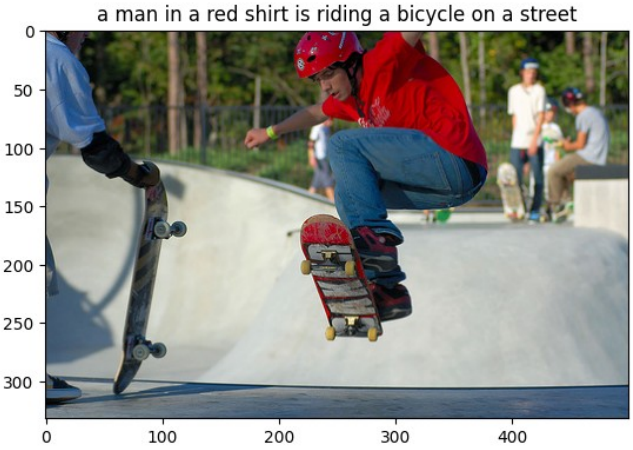
startseq group of people are walking on the street endseq

startseq person in red jacket is skiing down snow covered hill endseq

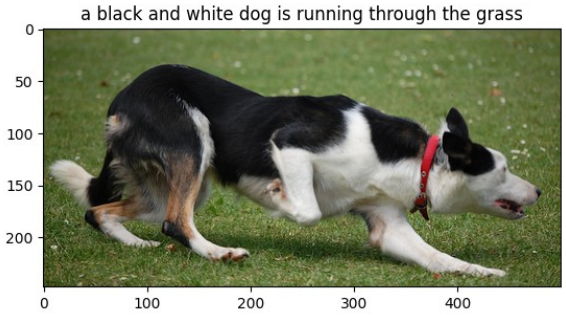
startseq football player in red uniform is playing game endseq

## 2. Modified-1:




## 3. Modified-2:

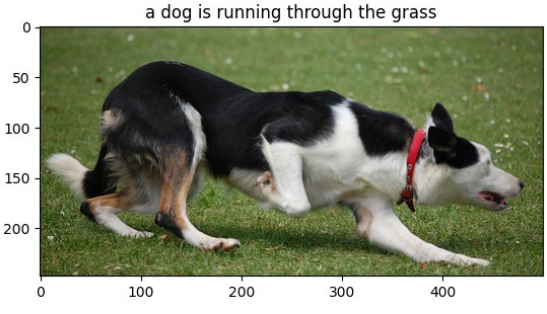




## 4. Modified-3:



## 5. Modified-4:

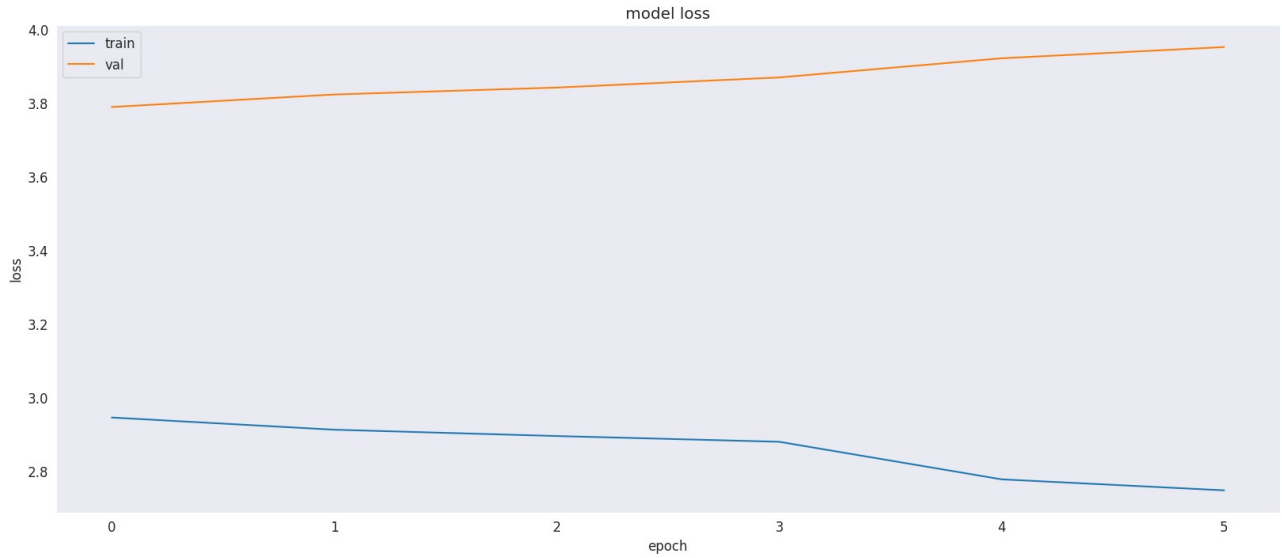



## 6. Modified-5:



## 7. Modified-6:

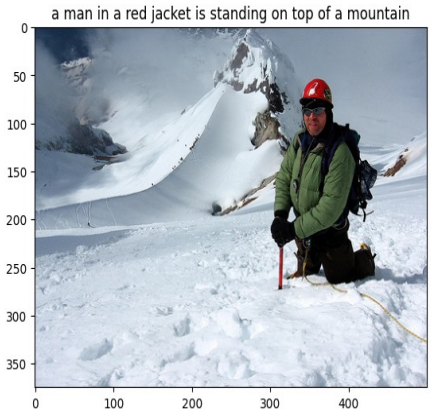

# Analysis Of Results:

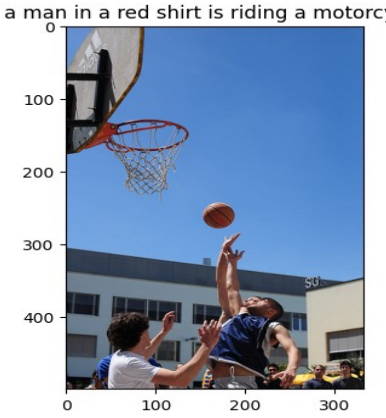


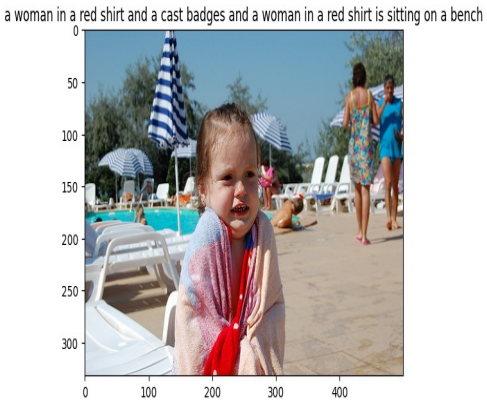

- As we observe(in above graph) this is the validation loss and train loss in baselined model, here we observe the model is overfitting on the train data.
- As seen from the above results of scores and predictions of captions we observe attention mechanisms in image captioning have been shown to be effective in improving the quality of generated captions. The main reason for this is that attention allows the model to focus on different parts of the image when generating the caption.
- On observing the results we have seen that The LSTM based image captioning can ‘blindly’ learn the structure of the language and predict meaningful sentences even without learning much insight to the content of the image. This is termed as “language bias” of the system. So we see in few of the above cases of our outputs I observed it predicted the phrase “a man in red shirt” blidly just by looking at the structure.
- We observe from the results that the models trained with multiple attention heads give the best results. A possible explanation for this would be that as we know the normal CNN-LSTM baseline has limitations in capturing complex spatial and temporal dependencies in the image and text domains, respectively. This is where attention helps us The use of multi-head attention in the CNN-LSTM architecture allows

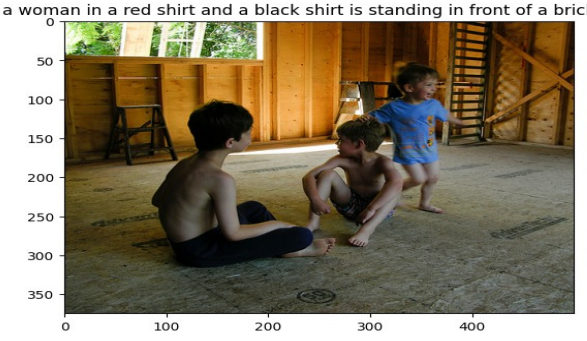
the model to attend to multiple parts of the input simultaneously, and to learn multiple representations of the input at different levels of abstraction. This can be particularly useful for image captioning tasks, where there may be multiple objects or regions of interest in the image that are relevant to the generated description. By attending to multiple parts of the input, the model can capture more complex relationships between the image and text domains, leading to better results. Additionally, the use of multi-head attention can also improve the interpretability of the model, by providing insights into which parts of the input are most relevant to the generated description.

- We also observe that the predictions predicted using GloVe embeddings are more meaningful compared to that of normal label encoding.
- There are a lot of training captions in which we find the “in a red shirt / red jacket etc.” i.e. phrases containing red attached to a human. All the images do not contain the red color shirt but are still captioned that way this may occur due to less training data or less intra class examples and overfitting.




Ways to overcome this problem would be:

- Increasing training data
- Data augmentation
- Architecture modifications such as models with transformers can potentially improve the model's ability to generate diverse and contextually relevant captions.
- Applying techniques like dropout, batch normalization, layer normalization etc in our models to prevent overfitting etc.

- One observation we also made was we trained the model on only 15 epochs due to gpu and time constraints so this may be considered less or the model has not been trained on sufficient epochs. Hence we could try training it on more epochs to get more meaningful captions.
- As we observe not only phrases like "red shirt" or "red jacket" there are many such phrases that get predicted just looking at the structure of the sentence or the features of image if hhuman is there red shirt and etc. This is where attention helped us by not giving all the features equal importance and weighing the features correspondingly.
- We also observed in the single-head attention mechanism, the model learns to attend to the most relevant parts of the input, but it may not capture all the relevant information. However, multi-head attention allows the model to attend to different parts of the input in parallel, and each head can capture different types of information. By doing so, multi-head attention can help the model capture more complex relationships between the image and caption, leading to better performance.
- Also we see on training on top of pre-trained glve embeddings led to a decline in the BLEU score compared to their heir with only pre trained embeddings. Few reasons for this could be:
  - Overfitting
  - Domain mismatch: using pre-trained embeddings which are well not suited and training upon them would lead to even bad results. This can interfere with the pre-trained weights, which may negatively impact the model's performance.

- We also see the model with single attention head and training upon pre-trained GloVe embeddings(modified-4) led to a BLEU score that was less compared to the Baseline's BLEU score. This may be due to domain-mismatch as mentioned above training on bad embeddings and weighing them more may have led it to such a bad BLEU score.

## Things we tried more:

- We did try to implement beam-search algorithm for prediction, we had it tested on our baseline model but it led to very bad BLEU score which was around 0.12, Hence we did not try it for other cases. Also it took a lot of time hence this testing was done randomly on a list of 100 images, these results were not as satisfying as previous attention algorithms mainly due to time constraints and limited testing.