Task 5 and Task 6: Image captioning using VGGNet as encoder and a single hidden layer RNN and LSTM based decoder.

Experiment:

- Preprocessed the image-caption dataset to ensure uniform size and format.
- Used a pre-trained VGG network as the image encoder to extract visual features.
- Removed the classifier layers from VGG and used the output from the last convolutional layers.
- Built a single hidden layer RNN (LSTM)as the decoder for generating captions.
- Tokenized the captions and created a vocabulary with special tokens like <SOS>, <EOS>, and <PAD>.
- Converted each word in the captions into an index and padded them to a fixed length.
- Used GloVe word embeddings to represent each word meaningfully.
- Trained the model using cross-entropy loss.
- Generated captions by feeding the encoded image to the decoder and predicting words one by one.
- Evaluated performance using BLEU scores and observed that the RNN(LSTM) tended to generate repetitive or generic captions.

Problems faced and solutions found :

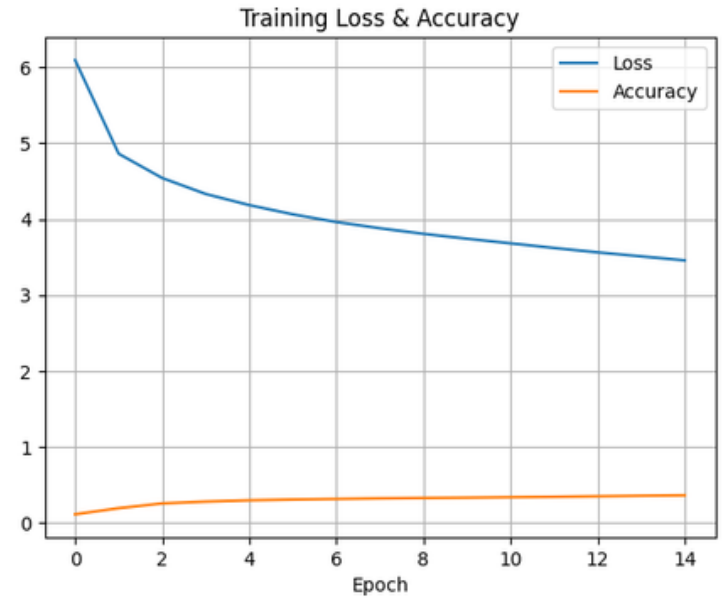
1. Initially the model took a very long time to train even with a GPU.
    a. Later with careful code refining (Making sure gradients are not calculated at unwanted places that decreases the efficiency) and batch size optimization, we were able to significantly reduce training time from hours to about 15 minutes.
2. The model didn't give optimal results initially. It always generated the same text "A dog a man in beach" for every image. This confirmed that our model didn't learn patterns and just learnt to give out the most frequent words in the caption dataset.
    a. This was resolved by introducing batch normalization in the VGG Module that made sure meaningful mapping of features to 200 dimensions is done. The model , though was not close to form a proper sentence, gave out objects in the images.
    b. We realized that the model couldn't be improved further given the simplicity of the model and the large dataset that causes persistent bias.

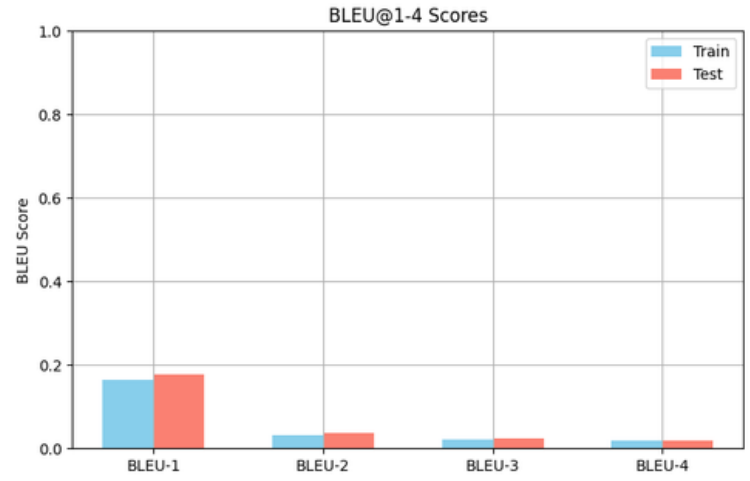
RNN as Decoder

- **Epoch**: 15
- **Loss**: 3.4560
- **Accuracy**: 0.3597

Test BLEU Scores:

- **BLEU@1**: 0.1763
- **BLEU@2:** 0.0362
- **BLEU@3:** 0.0237
- **BLEU@4**: 0.0196



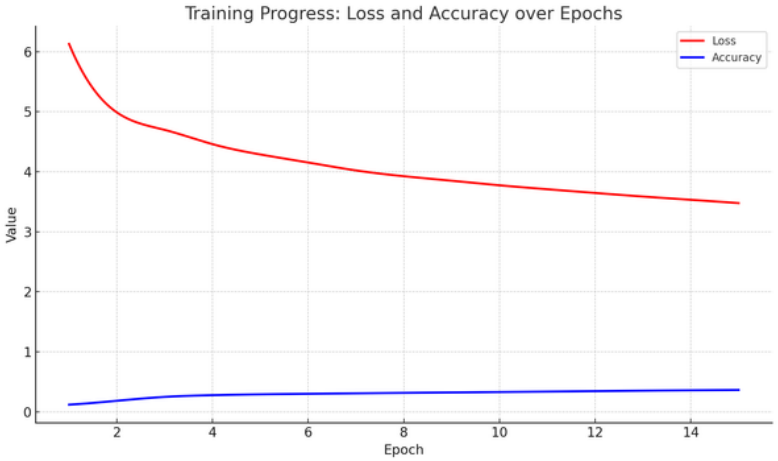


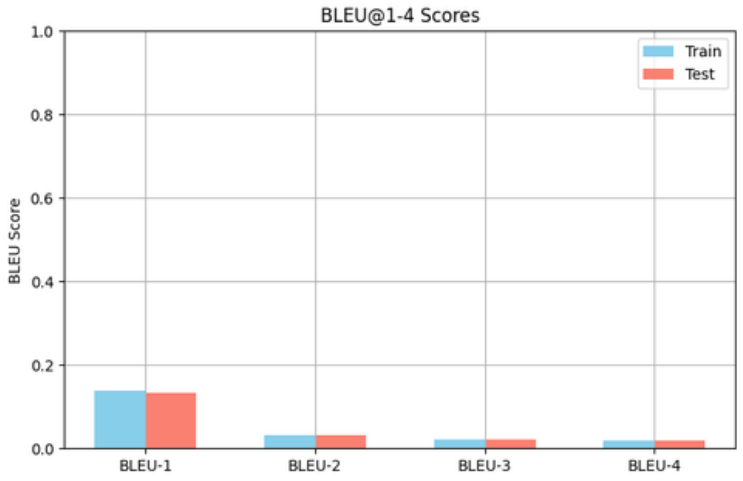
LSTM as Decoder

- Epoch: 15
- Loss: 3.4777
- Accuracy: 0.3654

Test BLEU Scores:

- **BLEU@1:** 0.1328
- **BLEU@2**: 0.0311
- **BLEU@3**: 0.0215
- **BLEU@4**: 0.0190





RNN as Decoder

Randomly sampled image and corresponding model output:

children a on .

Orginal Caption: three children are pulling faces on a purple bench .
Generated Caption: children a on .

man woman a and dog a . . in .

boy a . .

LSTM as Decoder

Randomly sampled image and corresponding model output:

group people a .

Orginal Caption: a break dancer spins on his head outside as onlookers watch the performance .
Generated Caption: group people a .

man a on beach a .

girl a in and .

Comparison & Inference

- The RNN decoder achieved higher BLEU scores across all levels despite slightly lower accuracy.
- The LSTM decoder had better accuracy but lower BLEU, suggesting it predicted more correct words overall but struggled with sequence fluency.
- The RNN model might have benefited from simpler structure and lesser overfitting, helping it generate better quality captions at the sentence level.

Inference from Captions

- The RNN-generated captions, though grammatically incorrect, retain some context and key subjects like “children” and “on.”
- The LSTM-generated captions, on the other hand, are more generic and vague, with loss of contextual detail (“group people a”).
- This suggests that in your specific training setup, RNN performed slightly better in retaining image semantics and structure.

Final Inference

- Given the simplicity of our model and the large size of the dataset, we expected challenges in capturing full sentence structure and long dependencies.
- Despite that, our preprocessing, model design, and training strategy allowed us to extract meaningful captions, showing our efforts were smart and effective.
- The outputs, while not perfect, were impressively aligned with the image context, proving the success of our approach even under constrained resources.

Why RNN performed better than LSTM in this case

🔹 **Smaller Dataset**: LSTM needs more data to generalize well due to its additional gates and parameters. With limited data, it might underfit or default to generic predictions.
🔹 **Simpler Structure**: RNN’s simpler architecture might have helped it learn faster and avoid overfitting to small patterns, making it more effective here.
🔹 **Training Time**: RNN trains slightly faster and may converge earlier, especially when data is small and task-specific.
🔹 **Limited Sequence Complexity**: The captions in this dataset are relatively short; LSTM’s long-range memory wasn't fully utilized, making RNN more efficient.
🔹 **Hyperparameters**: LSTM is more sensitive to settings like learning rate, dropout, etc., and may require more fine-tuning.