COL775 Deep Learning
Assignment 1.2
Text 2 Math Problem

Shashwat Bhardwaj
Roll Number: 2023AIY7528
Indian Institute of Technology Delhi

April 8, 2024

0.1 Seq2Seq Architecture

Seq2Seq (Sequence-to-Sequence) translation is a deep learning architecture used for converting sequences from one domain to another. It is particularly popular in tasks such as machine translation, where the input and output sequences can have different lengths.

The basic idea behind Seq2Seq translation is to use two recurrent neural networks (RNNs) - an encoder and a decoder. The encoder processes the input sequence and produces a fixed-length vector representation, capturing the semantic meaning of the input sequence. This vector, also known as the context or thought vector, is then fed into the decoder, which generates the output sequence one token at a time based on the context vector and the previously generated tokens.

Seq2Seq models are capable of handling variable-length input and output sequences, making them suitable for a wide range of tasks beyond machine translation, such as text summarization, question answering, and speech recognition.

The training process involves optimizing the parameters of both the encoder and decoder networks to minimize the difference between the predicted output sequence and the ground truth sequence. This is typically done using techniques like teacher forcing and attention mechanisms to improve the model's ability to capture long-range dependencies and produce accurate translations.

0.2 Different Seq2Seq Architectures and Embeddings

0.2.1 Seq2Seq with GloVe Embeddings and Bi-LSTM Encoder/LSTM Decoder

Seq2Seq models with various architectures and embeddings have been proposed to address different challenges in sequence-to-sequence tasks, such as machine translation., and question answering. Here are some examples of Seq2Seq models with different configurations:

0.2.2 Seq2Seq with GloVe Embeddings and Bi-LSTM Encoder/LSTM Decoder

This model utilizes GloVe (Global Vectors for Word Representation) embeddings to represent words in the input and output sequences. It employs a Bidirectional Long Short-Term Memory (Bi-LSTM) encoder to capture bidirectional context information from the input sequence and an LSTM decoder to generate the output sequence. Bi-LSTM helps in capturing contextual information from both directions, enhancing the model's understanding of the input sequence.

0.2.3 Seq2Seq+Attention with GloVe Embeddings and Bi-LSTM Encoder/LSTM Decoder

This model extends the basic Seq2Seq architecture by incorporating attention mechanisms. Attention allows the decoder to focus on different parts of the input sequence dynamically, improving the model's ability to generate accurate translations. It uses GloVe embeddings for word representation and a Bi-LSTM encoder to capture bidirectional context. The attention mechanism helps in aligning the decoder's output with relevant parts of the input sequence.

0.2.4 Seq2Seq+Attention with Pre-trained BERT Encoder and LSTM Decoder

This model leverages pre-trained BERT (Bidirectional Encoder Representations from Transformers) embeddings for word representation in both the encoder and decoder. BERT embeddings capture rich semantic information from large text corpora, enhancing the model's understanding of the input sequence. The encoder consists of a frozen pre-trained BERT model, which encodes the input sequence, while the decoder is an LSTM network. The attention mechanism is used to align the decoder's output with relevant parts of the input sequence.

0.2.5 Seq2Seq+Attention with Fine-tuned BERT Encoder and LSTM Decoder

In this model, the pre-trained BERT encoder is fine-tuned along with the remaining network during training. Fine-tuning allows the model to adapt the pre-trained BERT embeddings to the specific task, potentially improving performance on downstream tasks. The decoder remains an LSTM network, and attention mechanisms are used to facilitate alignment between the encoder and decoder states.

Each of these Seq2Seq models offers unique advantages and trade-offs in terms of computational complexity, memory requirements, and performance. The choice of model architecture and embeddings depends on the specific requirements of the task and the available resources.

1 Different Architectures

1.1 LSTM and BiLSTM Cells

Long Short-Term Memory (LSTM) networks are a type of recurrent neural network (RNN) architecture designed to capture long-term dependencies in sequential data. They are particularly effective in tasks such as language modeling, speech recognition, and machine translation.

The LSTM cell consists of several components:

- Input gate: Controls the flow of information into the cell.
- Forget gate: Determines which information from the previous cell state to forget.
- Cell state: Represents the memory of the cell.
- Output gate: Regulates the information to be output from the cell.

The equations governing the operations of an LSTM cell are as follows:

$$
\begin{aligned}
i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \\
f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \\
g_t &= \tanh(W_{xg}x_t + W_{hg}h_{t-1} + b_g) \\
o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}
$$

where:

- x_t is the input at time step t.
- h_{t-1} is the hidden state of the previous time step.

- i_t, f_t, g_t, and o_t are the input gate, forget gate, cell input, and output gate activations, respectively.
- c_{t-1} and c_t are the cell states at time steps $t-1$ and t, respectively.
- σ denotes the sigmoid activation function, and $\odot$ represents element-wise multiplication.

Bidirectional LSTM (BiLSTM) extends the LSTM architecture by processing the input sequence in both forward and backward directions. It consists of two LSTM layers: one processes the input sequence from left to right, while the other processes it from right to left. The outputs from both directions are concatenated at each time step, providing context from both past and future observations.

BiLSTM is especially useful for tasks where contextual information from both past and future contexts is crucial, such as named entity recognition, part-of-speech tagging, and sentiment analysis.

1.2 Attention Mechanism and Bahdanau Attention

Attention mechanism is a powerful concept in neural network architectures, particularly in sequence-to-sequence (Seq2Seq) models. It enables the model to focus on specific parts of the input sequence when making predictions, allowing for more accurate and contextually relevant outputs.

The basic idea behind attention is to compute attention weights for each input element, indicating its importance or relevance to the current decoding step. These attention weights are then used to compute a weighted sum of the input sequence, which serves as additional context for the decoder.

One popular form of attention mechanism is Bahdanau Attention, proposed by Dzmitry Bahdanau et al. in the paper "Neural Machine Translation by Jointly Learning to Align and Translate" (2015). Bahdanau Attention introduces a set of learnable parameters to compute attention scores dynamically based on both the current decoder hidden state and the encoder hidden states.

The attention scores are computed using a small neural network, typically a feedforward network with a softmax activation function, which generates a probability distribution over the input sequence. The attention weights are then used to compute a context vector, which is a weighted sum of the encoder hidden states.

The equations governing Bahdanau Attention are as follows:

$$
\begin{aligned}
e_{ij} &= v_a^T \tanh(W_a s_{i-1} + U_a h_j) \\
\alpha_{ij} &= \text{softmax}(e_{ij}) \\
c_i &= \sum_{j=1}^{T} \alpha_{ij} h_j
\end{aligned}
$$

where:

- e_{ij} represents the attention score for the i-th decoder step and the j-th encoder step.
- α_{ij} is the attention weight, indicating the importance of the j-th encoder hidden state at the i-th decoder step.
- s_{i-1} is the previous decoder hidden state.
- h_j is the j-th encoder hidden state.
- v_a, W_a, and U_a are learnable parameters.
- c_i is the context vector for the i-th decoder step.

Bahdanau Attention allows the model to selectively attend to different parts of the input sequence at each decoding step, improving the model's ability to capture complex dependencies and generate accurate outputs.

1.3 Bidirectional Encoder Representations from Transformers (BERT)

Bidirectional Encoder Representations from Transformers (BERT) is a revolutionary language representation model introduced by Jacob Devlin et al. from Google AI Language in 2018. BERT leverages the Transformer architecture, a powerful deep learning model introduced by Vaswani et al. in the paper "Attention Is All You Need" (2017).

BERT is designed to pre-train deep bidirectional representations of text by jointly conditioning on both left and right context in all layers. Unlike traditional language models that are unidirectional or shallowly bidirectional, BERT learns contextualized word representations by considering the entire input sentence or paragraph simultaneously.

The key innovation of BERT lies in its pre-training objectives, which consist of two tasks:

1. **Masked Language Model (MLM)**: BERT pre-trains by randomly masking some of the input tokens and then predicting those masked tokens based on the surrounding context. This enables the model to capture bidirectional context during pre-training.
2. **Next Sentence Prediction (NSP)**: BERT is also trained on a next sentence prediction task, where it learns to predict whether two input sentences are consecutive or not. This task helps BERT understand the relationships between sentences and improves its ability to perform tasks like text classification and question answering.

Additionally, BERT introduces a special token called the **[CLS]** token. This token is prepended to the input sequence, and its output representation is used in downstream tasks, such as classification or regression. The [CLS] token representation is used to encode the entire input sequence and is especially important for tasks like sentence classification, where the model needs to understand the overall semantics of the input.

2 Model Implementations

2.1 Seq2Seq Model with GloVe Embeddings

It consists of an encoder, which encodes input sequences into a fixed-size context vector, and a decoder, which generates output sequences based on the context vector. The model is trained to minimize the difference between predicted and actual output sequences. This architecture is commonly used for tasks like machine translation and sequence generation.

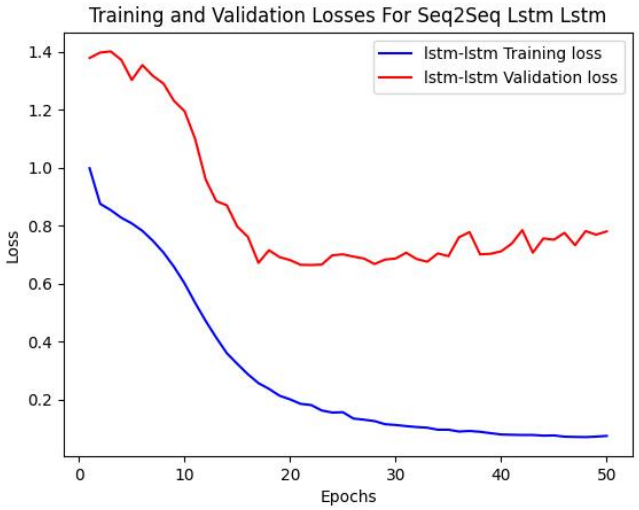
2.1.1 Accuracy and Performance

Metric	**Test (%)**	**Validation (%)**
Execution Accuracy	46.13	44.11
Exact Match Accuracy	43.58	41.64

Table 1: Performance metrics for the model on test and validation sets.

2.1.2 Hyperparameters

- Encoder embedding dimension : 300
- Decoder embedding dimension : 100
- Encoder hidden dimension : 512
- Decoder hidden dimension : 1024
- Learning rate: 0.001



(a) Loss curves for the Seq2Seq model with GloVe Embeddings.

Figure 1: Model Loss Plots

2.2 Seq2Seq + Attention Model with GloVe Embeddings

Description of the Seq2Seq model with GloVe Embeddings.

2.2.1 Accuracy and Performance

Teacher Forcing Ratio	Metric	Value (%)
0.3	Test Execution Accuracy	40.10
	Test Exact Match Accuracy	38.54
	Valid Execution Accuracy	41.80
	Valid Exact Match Accuracy	38.76
0.6	Test Execution Accuracy	41.40
	Test Exact Match Accuracy	39.50
	Valid Execution Accuracy	38.22
	Valid Exact Match Accuracy	44.67
0.9	Test Execution Accuracy	38.94
	Test Exact Match Accuracy	39.77
	Valid Execution Accuracy	40.23
	Valid Exact Match Accuracy	39.22

Table 2: Test and validation metrics for beam width 10 with various teacher forcing ratios.

2.2.2 Hyperparameters

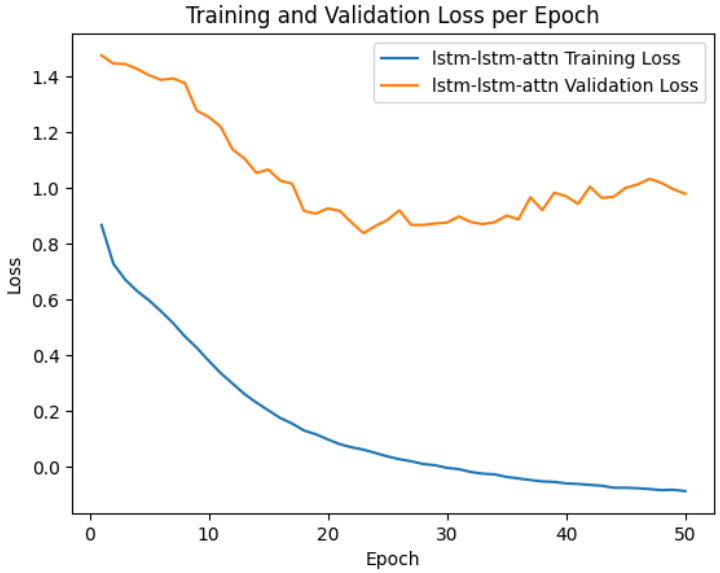
- Encoder embedding dimension : 100
- Decoder embedding dimension : 512
- Encoder hidden dimension : 512
- Decoder hidden dimension : 512
- Learning rate: 0.001 foe epochs upto 20 and 0.0001 for epochs greater than 20

(a) Loss curves for the Seq2Seq + Attention model with GloVe Embeddings with 0.3 Teacher forcing ratio.

Figure 2: Model Loss Plots

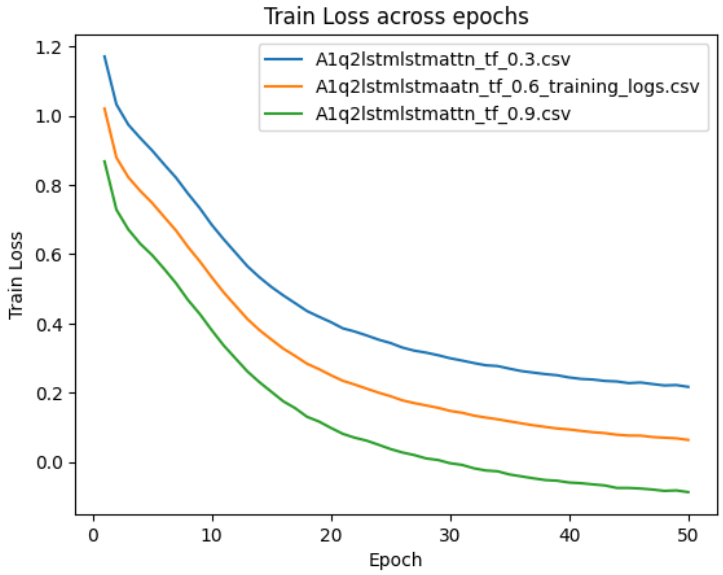
(a) Loss curves for the Seq2Seq + Attention model with GloVe Embeddings with 0.6 teacher forcing ratio.

Figure 3: Model Loss Plots



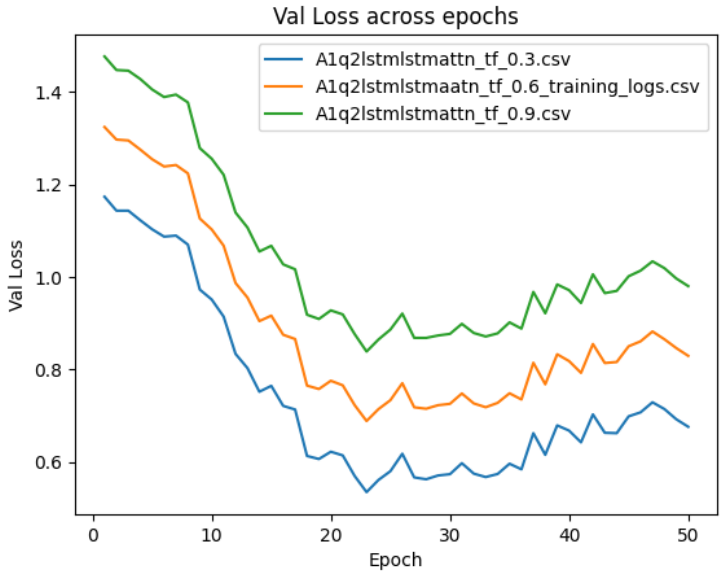
(a) Loss curves for the Seq2Seq + Attention model with GloVe Embeddings with 0.9 teacher forcing ratio.

Figure 4: Model Loss Plots



(a) Comparision of Train Loss curves for the Seq2Seq + Attention model with different teacher forcing ratios

Figure 5: Model Loss Plots



(a) Comparision of Validation Loss curves for the Seq2Seq + Attention model with different teacher forcing ratios

Figure 6: Model Loss Plots

2.3 BERT Seq2Seq + Attention Model with Frozen BERT-base-cased

Description of the BERT Seq2Seq + Attention model with Frozen BERT-base-cased.

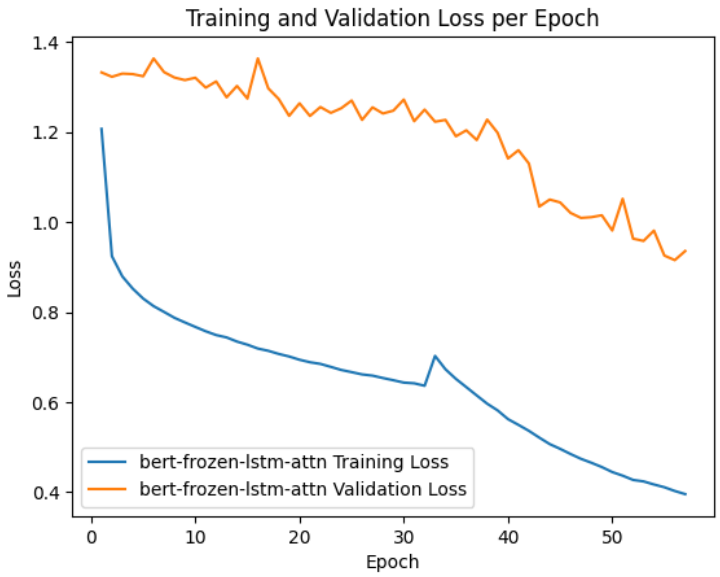
2.3.1 Accuracy and Performance

Metric	Test (%)	Validation (%)
Execution Accuracy	10.64	10.19
Exact Match Accuracy	9.50	9.18

Table 3: Performance metrics for the BERT Seq2Seq + Attention model with Frozen BERT-base-cased.

2.3.2 Hyperparameters

- Decoder embedding dimension (DEC_EMB_DIM): 100
- Encoder hidden dimension (ENC_HID_DIM): 128
- Decoder hidden dimension (DEC_HID_DIM): 256
- Decoder dropout rate (DEC_DROPOUT): 0.5
- Learning rate: 0.001



(a) Loss curves for the BERT Seq2Seq + Attention model with Frozen BERT-base-cased.

Figure 7: Model Loss Plots

2.4 BERT Seq2Seq + Attention Model with Fine-tuned BERT-base-cased

Description of the BERT Seq2Seq + Attention model with Fine-tuned BERT-base-cased.

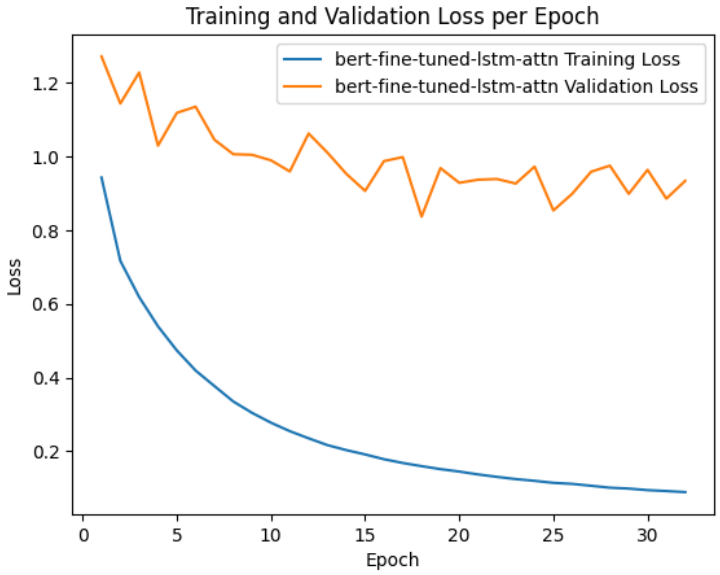
2.4.1 Accuracy and Performance

Beam Width	Metric	Test Value (%)
1	Execution Accuracy	10.64
	Exact Match Accuracy	9.50
10	Execution Accuracy	11.23
	Exact Match Accuracy	11.70
20	Execution Accuracy	11.37
	Exact Match Accuracy	10.65

Table 4: Test metrics for different beam widths.

2.4.2 Hyperparameters

Detail the hyperparameters used for this model configuration.

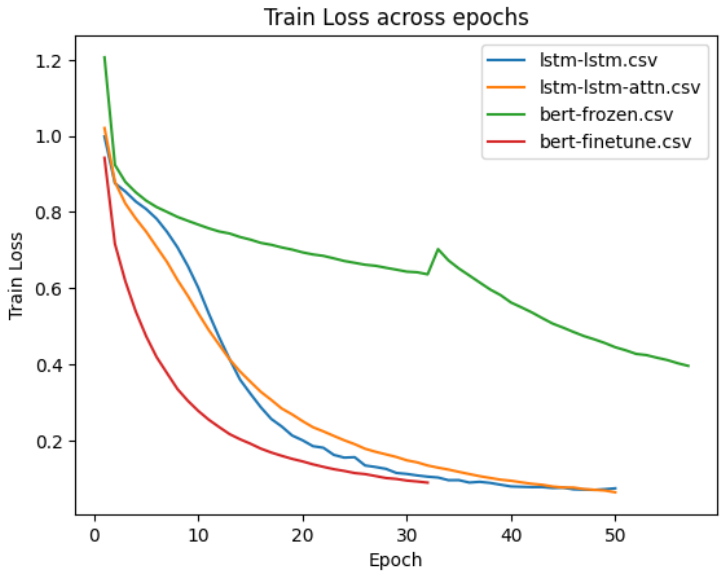


(a) Loss curves for the BERT Seq2Seq + Attention model with Fine-tuned BERT-base-cased.

Figure 8: Model Loss Plots

3 Model Comparison

This section compares all the models based on their execution and exact match accuracies, loss curves.



(a) Loss curves for the all the 4 models

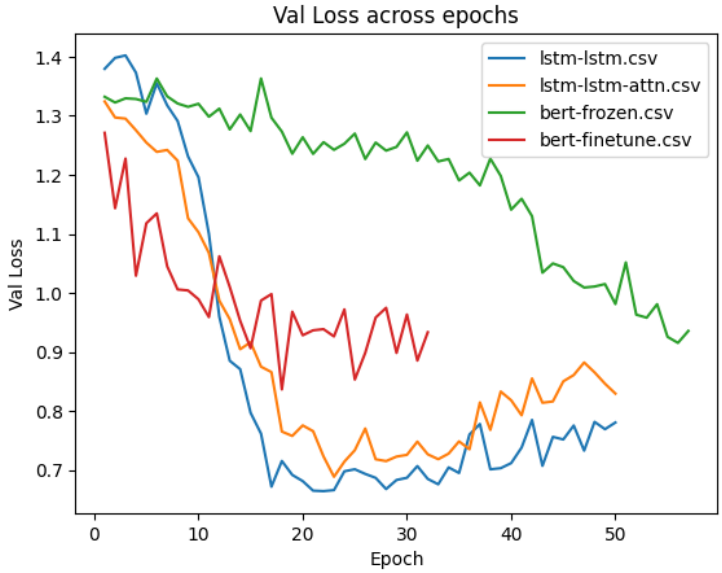
Figure 9: Model Loss Plots

3.0.1 Analysis of Training and Validation Loss

The provided figures depict the training and validation loss curves for four distinct model configurations: LSTM, LSTM with attention, BERT with frozen parameters, and BERT with fine-tuning. These curves are indicative of the models' learning progress over a number of epochs during training.

In the first figure representing training loss, all models start with a high loss that rapidly decreases, showcasing initial learning. As the epochs progress, all models show diminishing returns in learning, which is natural as they begin to converge to a minimal loss. The LSTM with attention mechanism (LSTM-lstm-attn.csv) and the fine-tuned BERT model (bert-finetune.csv) show a more stable and consistent decline in training loss compared to the others, suggesting more effective learning patterns.

The second figure illustrates the validation loss, which is a measure of how well the model generalizes to unseen data. A common issue in training deep learning models is overfitting, where the model learns the training data too well, including its noise and outliers, but fails to generalize this learning to new data.



(a) Loss curves for the all the 4 models

Figure 10: Model Loss Plots

In this case, the validation loss for the LSTM model without attention (lstm-lstm.csv) seems to fluctuate significantly, implying potential overfitting. The BERT with fine-tuning (bert-finetune.csv) demonstrates a generally downward trend, with some volatility, indicating that it is learning generalizable patterns but may still be somewhat sensitive to the training data specifics or may require regularization.

On the other hand, the BERT with frozen parameters (bert-frozen.csv) shows an initial decrease in validation loss followed by a plateau. This could be interpreted as the model quickly reaching its potential learning capacity with the given frozen parameters. It is worth noting that the fine-tuned BERT model eventually outperforms the frozen BERT model, showing the benefits of fine-tuning pre-trained models on specific tasks.

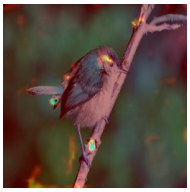
The analysis of these curves must also consider the trade-off between bias and variance, the complexity of the model, and the diversity of the training and validation datasets. In conclusion, the fine-tuned BERT model appears to have the best performance in terms of generalizability, as evidenced by the consistent downward trend in validation loss, despite some fluctuations which may need to be addressed through additional regularization or parameter tuning.

Accuracies across models especially for BERT models, could be improoved, by more hyper-parameter tuning and availability of powerful computational hardware, but due to lack of time and resources, was unable to do these.

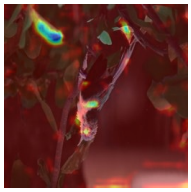
I was havibg too many mnay difficulties at home due to grandmother's ill-

ness, so did not even get time to fine tune hyper-parameters. Submitted Medical certificate and doctor prescription. Kindly consider this point while evaluation. Thanks and Regards

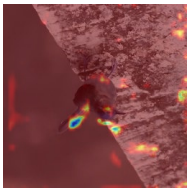
1 Gradcam



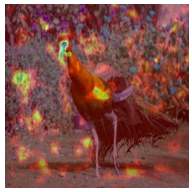
(a) Image 1



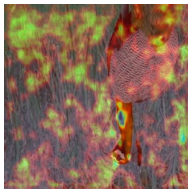
(b) Image 2



(c) Image 3

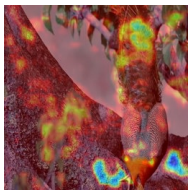


(d) Image 4

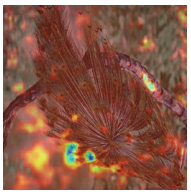


(e) Image 5

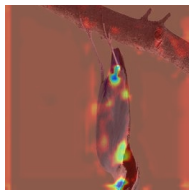
(f) Image 6



(g) Image 7



(h) Image 8



(i) Image 9

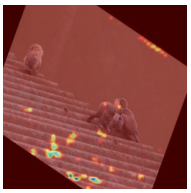
(j) Image 10

Correct Class Peacock

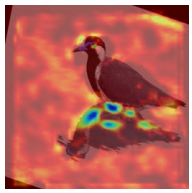
Incorrect Class

(a) Image 1

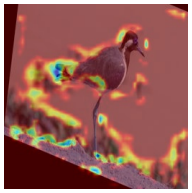
(b) Image 2



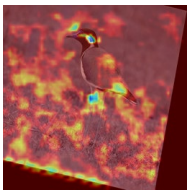
(c) Image 3



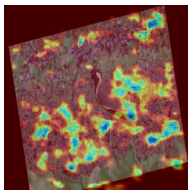
(d) Image 4



(e) Image 5

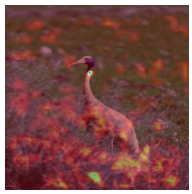


(f) Image 6



(g) Image 7

(h) Image 8



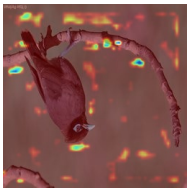
(i) Image 9

Correct Class Red Wattled Lapwing

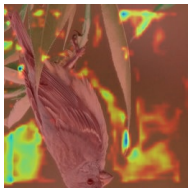
Incorrect Class

Correct Class Rudy Shelduck

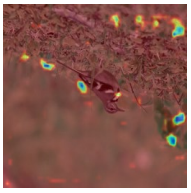
Incorrect Class



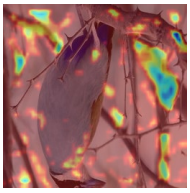
(a) Image 1



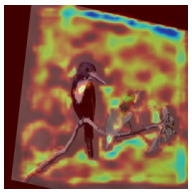
(b) Image 2



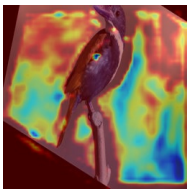
(c) Image 3



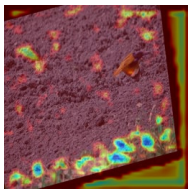
(d) Image 4



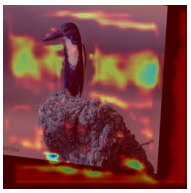
(e) Image 5



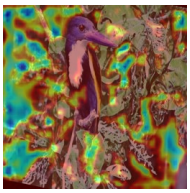
(f) Image 6



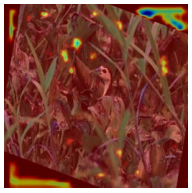
(g) Image 7



(h) Image 8



(i) Image 9



(j) Image 10

Correct Class Common Myna

Incorrect Class