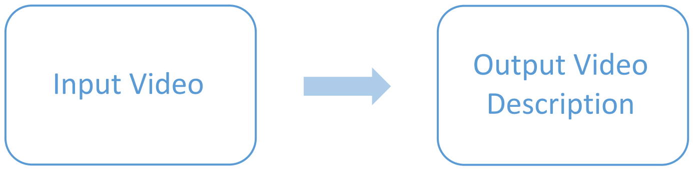
CPSC 8430: Deep Learning
Report: Homework 2

Name: Suraj B. Khamkar
skhamka@g.clemson.edu | CUID: C16741431

GitHub Repository: https://github.com/khamkarsuraj/Deep-Learning/tree/main/Homework_2

To do:
Show a video caption using a sequential-to-sequential model for an input video. A video will serve as the code's input, and the code will produce a stream of captions describing the film's actions. Recurrent neural networks are used to accomplish the aforementioned.



Prerequisites:
- Python
- torch
- scipy
- numpy
- pandas
- pickle

Format and Data:
- The Microsoft Video Description (MSVD) dataset is used for this homework. It has around 120K sentences.
- I have used 1450 videos to train the model.
- 100 videos to test the model.
- After preprocessing the data, we have stored data in video format of 80x4096.

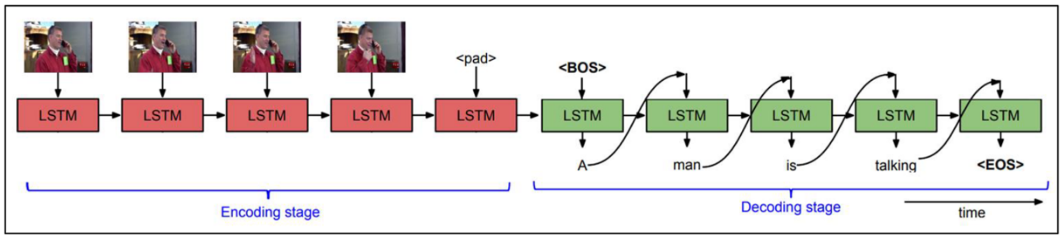
Preprocessing of the given dataset:
The slides specify that a dictionary is necessary for the seq2seq model. The fundamental word-to-phrase mapping is shown below.
Used tokens are,

<PAD>	: Pad the sentence to the same length.
<BOS>	: Begin of sentence, a sign to generate the output sentence.
<EOS>	: End of sentence, a sign of the end of the output sentence.
<UNK>	: Use this token when the word isn't in the dictionary or just.

All below data structures are stored as python directory,
word_dict: Contains a word's frequency in the training label file as determined by creating a vocabulary.
Every term with a frequency of fewer than four is disregarded.
w2i: Includes word to index mappings for the vocabulary terms.
i2w: Includes index to word reverse mappings for the vocabulary terms.

Trained Model:

Encoder and decoder are the model's two levels. Both of these layers are created using Gated Recurrent Units (GRU). The GRU (RNNs) operates more quickly than LSTM because it has fewer training parameters, which requires less memory. Since the dataset contains shorter data sequences, the GRU delivers quicker results with LSTM-like accuracy. The encoder (encoderRNN) processes the video, which then encodes it into the required format. The captions are then divided using the beginning and ending tokens, and video processing is applied to the words to create the actual words.
Hyperparameters used for encoders and decoders whereas dropout percentage is 0.3.



Attention Layer:

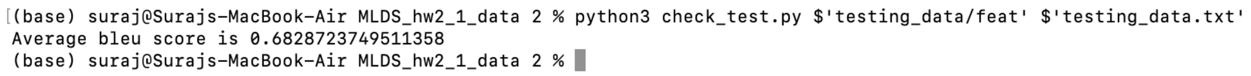
At every decoding time step, it enables the model to sneak a look at various input portions. To enhance the effectiveness of the underlying model, an attention layer is applied to the encoder hidden states. The model can peek at various input regions at each decoding time step. The final new confidential form is subsequently transmitted to the decoder's following time step.

Training and Testing:

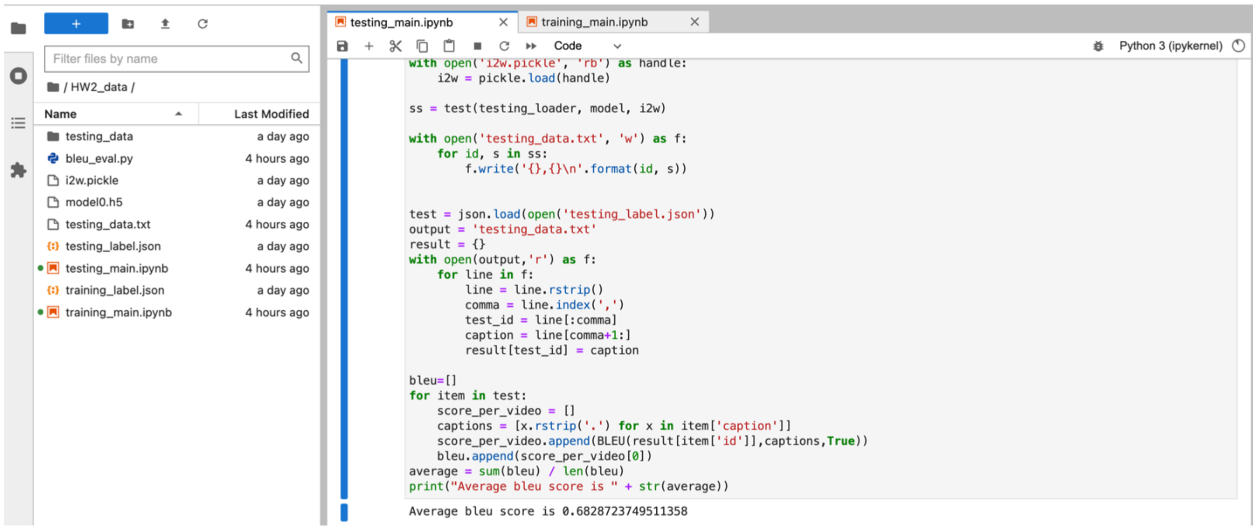
- Training model = training_main.ipynb
- Testing model = testing_main.ipynb
- Batch size = 32
- Epochs = 10
- Learning rate = 0.0001
- Frame dimension = 80
- Feature dimension = 4096

To execute test model, run check_test.py file. This file will generate prediction caption output for test videos. The created output caption is then contrasted with the ground truth caption to provide the BLEU score for the BLUE_eval function.

With given test data, BLEU score of **0.6828723749511358**, as below,



```
[(base) suraj@Surajs-MacBook-Air MLDS_hw2_1_data 2 % python3 check_test.py $'testing_data/feat' $'testing_data.txt'
 Average bleu score is 0.6828723749511358
 (base) suraj@Surajs-MacBook-Air MLDS_hw2_1_data 2 %
```



Name	Last Modified
testing_data	a day ago
bleu_eval.py	4 hours ago
i2w.pickle	a day ago
model0.h5	a day ago
testing_data.txt	4 hours ago
testing_label.json	a day ago
testing_main.ipynb	4 hours ago
training_label.json	a day ago
training_main.ipynb	4 hours ago

```
with open('i2w.pickle', 'rb') as handle:
    i2w = pickle.load(handle)

ss = test(testing_loader, model, i2w)

with open('testing_data.txt', 'w') as f:
    for id, s in ss:
        f.write('{},{}\n'.format(id, s))


test = json.load(open('testing_label.json'))
output = 'testing_data.txt'
result = {}
with open(output,'r') as f:
    for line in f:
        line = line.rstrip()
        comma = line.index(',')
        test_id = line[:comma]
        caption = line[comma+1:]
        result[test_id] = caption

bleu=[]
for item in test:
    score_per_video = []
    captions = [x.rstrip('.') for x in item['caption']]
    score_per_video.append(BLEU(result[item['id']],captions,True))
    bleu.append(score_per_video[0])
average = sum(bleu) / len(bleu)
print("Average bleu score is " + str(average))
```

```
Average bleu score is 0.6828723749511358
```