Deep Learning - Spring 2025

Assignment 3

Image Caption Generation and Object Detection

Due Date: 12:15 AM on Saturday, 03-May -2025 **Total Marks:** 130

Submission:
Submit all of your codes and results in a single zip file with the name FirstName_RollNumber_01.zip

- Submit a single zip file containing
  (a) codes (b) report (c) Saved Models (d) Readme.txt
- There should be **Report.pdf** detailing your experience and highlighting any interesting results. Kindly don't explain your code in the report, just explain the results. Your report should include your comments on the results of all the steps, with images, for example, what happened when you changed the learning rate, etc.
- Readme.txt should explain how to run your code, preferably it should accept the command line arguments e.g dataset path used for training the model.
- In the root directory, there should be 2 python files, a report, and a folder containing saved models.
- Root directory should be named as **FirstName_RollNumber_03**
- Your code script files should be named as **'rollNumber_03_task1.py'.**
- You have to submit a .py code file**.** Follow all the naming conventions.
- For each convention, there is a 3% penalty if you don't follow it.
- **Email the instructor or TA if there are any questions**. You cannot look at others' code or use others' code, however, you can discuss it with each other. **Plagiarism will lead to a straight zero with additional consequences as well.**
- 10% (of obtained marks) deduction per day for late submission.
- The late submissions will only be accepted till **06 May at midnight.**
- **DON'T RESUBMIT THE DATASETS PROVIDED IN YOUR SUBMISSION.**
- Use the dataset provided for this assignment, do not use any other dataset.
- Marks will be allocated based on viva and written code. Prepare yourself accordingly.

Note: For this assignment (and for others in general) **you are not allowed to search online for any kind of implementation.** Do not share code or look at the other's code. You should not be in possession of any implementation related to the assignment, other than your own. In case of any confusion please contact the TAs (email them or visit them during their office hours).

Objectives: In this assignment, you will write the code implementing an image captioning generation model and object detection model. The goals of this assignment are as follows.

- Combination of CNN and LSTM layer use. (Task-1)
- How to process text along with Image. (Task-1)
- Feature extraction of text and images and used as a single source. (Task-1)
- How to play with YOLO models for object detection. (Task-2)

⚠ **Important:** With your submission, you also need to submit one .py file named **'rollNumber_03_allCode.py'** in your zip folder. This file should contain the code of all 'code files' in your project. NO SUBMISSION WILL BE ACCEPTED WITHOUT IT.

Task-01 Image Captioning Generation

(100 Marks)

1. **Dataset**:

   In this assignment, the "flicker" dataset is shared with you. This dataset contains Images and captions against each image. There are a total of 8000 images, and there are multiple captions for each image. For an in-depth understanding of the dataset, refer to the following link: Flicker Dataset.

```
1  data = pd.read_csv("./dataset/captions.txt")
2  data.head()
```

[3] ✓ 0.0s

	image	caption
0	1000268201_693b08cb0e.jpg	A child in a pink dress is climbing up a set o...
1	1000268201_693b08cb0e.jpg	A girl going into a wooden building .
2	1000268201_693b08cb0e.jpg	A little girl climbing into a wooden playhouse .
3	1000268201_693b08cb0e.jpg	A little girl climbing the stairs to her playh...
4	1000268201_693b08cb0e.jpg	A little girl in a pink dress going into a woo...

Figure 1: CSV File head

Figure 2: Sample Image with its caption

2. **Split Data:**

   The dataset contained the Image directory in which images are stored. You can select any number greater than 1000 for training and a minimum of 200 samples for testing and validation.

2. **Data Loader**

   The data-loader task for this problem is easy, you have to load preprocessed text and image. In the data-loader you have to make a function to process the images and text. You have to research how you can set you text into label form e.g: convert into other form (on-hot) or any other proper form. Necessary steps to follow on image

and text are as

Text-Preprocessing:

1. Convert all the captions into lowercases.
2. Add starting and ending token on each sentence to check the starting and ending sequence of the sentence.
3. Remove special letters (optional).
4. Add any extra preprocessing by your knowledge (one is mandatory).

Image-Preprocessing:

1. Load image and convert into RGB (popular options are OpenCV and pillow), you can use any type of method to load the image.
2. Resize the image to a standard size and normalize the image.
3. For normalization use the values as:
    1. Mean = [0.485, 0.456, 0.406]
    2. Standard Deviation = [0.229, 0.224, 0.225]
4. Any other if you want any

3. Vocabulary/Tokenizer:

1. To process the text, you need to convert the text into numbers, for which you have to prepare vocabulary.
2. To prepare vocabulary, you need to process all the captions text and select unique words.
3. Each word contains a unique number as shown in Figure 3.
4. The encodings is passed to a embedding layer to generate embeddings.

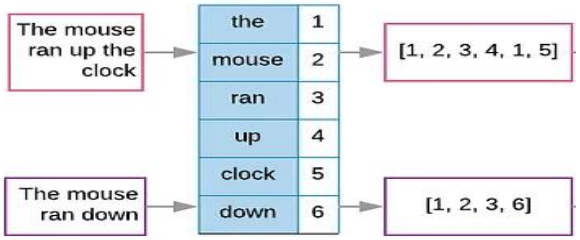

Figure 3: Example of creating and assigning ids to words.

5. The text from text to tokenization and tokenization to text are as follows:

```
18
19 print(tokenizer.texts_to_sequences([captions[1]]))
20 print(tokenizer.sequences_to_texts([[1, 18, 315, 63, 195, 116, 2]]))
95] ✓ 0.6s
... [[1, 18, 315, 63, 195, 116, 2]]
['startsentence girl going into wooden building endeentence']
```

6. You can use **word2vec from nltk or any other library**. Do not implement it from scratch.

4. **Model Architecture**
    1. **Image embeddings:**
        For this task, to generate image embeddings you can use any pretrained model or write your own custom model. Get the features till the last layer of convolution of the model.
        Note: Whatever model you choose/design, you have to explain why you select it in the report and also can be asked in the viva.
    2. **Text Embeddings:**
        For text embeddings you can use embeddings layers to generate meaningful embeddings for the model.
    3. **Combined Model:**
        Concatenate the image and text embeddings into a single layer and use multiple LSTM layers. Further moving from LSTM you can write down layers as per your understanding to generate a captioning system.
5. **Model Setting**:
    1. Use loss function as per the setting of your labels.
    2. Use any Optimizer that fits best to the problem.
6. Metrics:
    1. BLEU: (article link)
        1. Calculate BLEU on training and validation and draw a graph as well.
7. **Callbacks**
    To improve training stability, implement the following callbacks:
    1. **Learning Rate Decay**: Reduce the learning rate after a certain predefined value.
    2. **Save Best Weights**: Save or overwrite model weights whenever validation loss improves.
    3. **Early Stopping**: When the loss does not converge for certain epochs, the training of the model stops to save time and resources.
    4. **ReduceLROnPlateu**: When model is about to stop converging, the learning rate will be reduced.

8. **Train & Test Methods**
    Write separate methods for **training** and **testing**. Ensure that your code is modular by dividing it into functions. If any operation is used multiple times, create a reusable function and call it where necessary.

    Evaluation
    Create a separate file named test_eval.py, where:
    - The best model is loaded.
    - The user provides image paths as input.
    - The model predicts the caption and displays results.
      While you may use Jupyter Notebook for experimentation, ensure that test_eval.py is included in your submission.

9. **Visualization**
   - **Visualizations**: Show generated text and actual text with each image (make a graph and add image and text into it).
   - **Graphs**: Make sure to add loss and metrics training and validation graphs for each experiment.

Deliverables

```
project/
|-- rollNumber_03_task1.py'      # All code from data-loader to test & visualization.
|-- train.py                     # Training script
|-- test.py                      # Testing Script
|-- model.py                     # Captioning Model class
|-- data_utils.py                # Dataset/dataloaders
|-- weights                      # If weights are too large then upload it on your drive
                                 and share link in submission.
|-- Report.pdf                   # Detailed report including analysis.
|-- graphs                       # contain all the graphs as mentioned.
|-- requirements.txt             # Dependencies
'-- README.md                    # Execution instructions
```

Task-02 Object Detection Using YOLO

(30-Marks)

1. **Dataset**:

   In this task, the "Traffic Vehicles Detection" dataset is shared with you. This dataset contains Images and objects information against each image. The dataset contain 923 samples in training and test sample only contained images.

2. CSV File:

   Create a CSV file in which two columns on contain the image path and another one contains the label path.

   a. Split Dataframe:

      Split the dataframe into training and validation. Now pick every dataframe and iterate over each row and pick each image and label and save it in the form of yolo dataset structure.

      Structure of YOLO dataset are as follows:

```
dataset/
├ images/
|   ├ train/
|   ├ val/
|   ├ test/      # Optional
├ labels/
|   ├ train/
|   ├ val/
|   ├ test/      # Optional
├— data.yaml      # Dataset configuration file
```

b. Data.yaml:

Yaml file contains information about the dataset directories and classes information.

3. Model Training:

You can use ultralytics library for training the model. There is 2 to 3 lines of code that can start the training of the model, you have to YOLOV8 or YOLOV10 or YOLOV11 for this task. Also, you have to implement the two variants of the selected version e.g; nano, small, large, medium etc and compare the results. For more read the documentation of Ultralytics.

While passing the parameters

4. Model Testing:
   a. You must test the model in another file and show the metrics on validation and visualization of the predicted results on the testing set. Use best saved model for testing.
   b. Show IoU (>0.5 and >0.75) score on the validation and testing set.
   c. Another file in which you have to pass an image and in return it will show the predicted results on the image.
5. Report:
   a. In the report, show comparison of the two variants (nano, small, medium or large) models.
   b. Add graphs of loss function and IoU.
   c. Visualization of actual and predicted objects to display on the image.
   d. Loss function is used and why that loss function is used.
   e. Mention the model performance on the basis of graphs.

6. Deliverables

```
project/
|-- rollNumber_03_task2.py'      # All code from data-loader to test & visualization.
|-- train.py                     # Training script
|-- test.py                      # Testing Script
|-- weights                      # If weights are too large then upload it on your drive
                                 and share link in submission.
|-- Report.pdf                   # Detailed report including analysis.
|-- graphs                       # contain all the graphs as mentioned.
|-- requirements.txt             # Dependencies
'-- README.md                    # Execution instructions
```

Note

A file with named as "rollNumber_03_allCode.py" contained the code of task-1 and task-2 must be included in the main directory. Otherwise, mentioned penalty will be applied.

Good Luck