Configuration Manual

MSc Research Project
MSc. Data Analytics

Davis Munachimso Agughalam
Student ID: 19143354

School of Computing
National College of Ireland

Supervisors: Dr Paul Stynes
Dr Pramod Pathak

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Davis Munachimso Agughalam	
Student ID:	19143354	
Programme:	MSc. Data Analytics	**Year:** 2019/2020
Module:	Research Project	
Supervisor:	Dr Paul Stynes	
Submission Due Date:	17th August 2020	
Project Title:	Bidirectional LSTM approach to image captioning with scene features	
Word Count:	1426	**Page Count:** 12

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Davis Munachimso Agughalam
Date:	17th August 2020

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

1 Introduction

In this configuration manual, the steps followed to carry out the implementation of this research are detailed. These steps include data collection and processing, feature extraction and model development. Code snippets, screenshots and directions to execute the codes are also present to ensure seamless reproducibility.

2 Hardware and Software configurations

Table 1: Hardware specifications

Host machine/Operating System	MacBook Pro/MacOS Catalina
RAM	16 GB, core i7 processor.
Hard Disk	256GB
Cloud compute (GPU)	Free GPU Tesla K80 offered by Colab with 2496 CUDA cores and 12GB RAM.

Table 2: Software specifications

Programming language	Python (Anaconda distribution)
IDE	Jupyter notebook
Cloud environment	Google Collaboratory
Browser	Google chrome

Due to the large data size, the data is first processed on the local machine and results are saved as pickle files before they are moved to Google Colab for modelling.

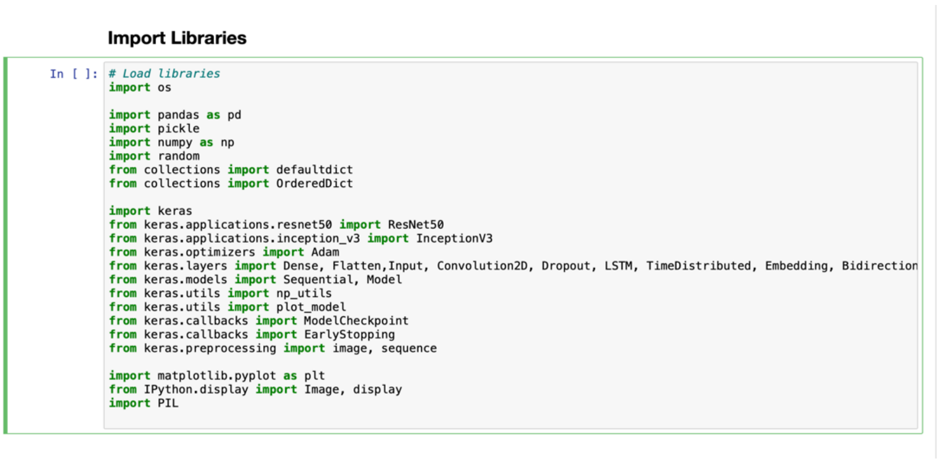
3 Data Preparation

3.1 Collecting the Flickr8k dataset

1. Download dataset from the URL https://www.kaggle.com/shadabhussain/flickr8k onto a work station and store dataset in new separate folder. The dataset contains images and text descriptions to be parsed in next section.

3.2 Data parsing

1. Open a new Jupyter notebook environment in the same folder as stored data.
2. Import required libraries to process data.

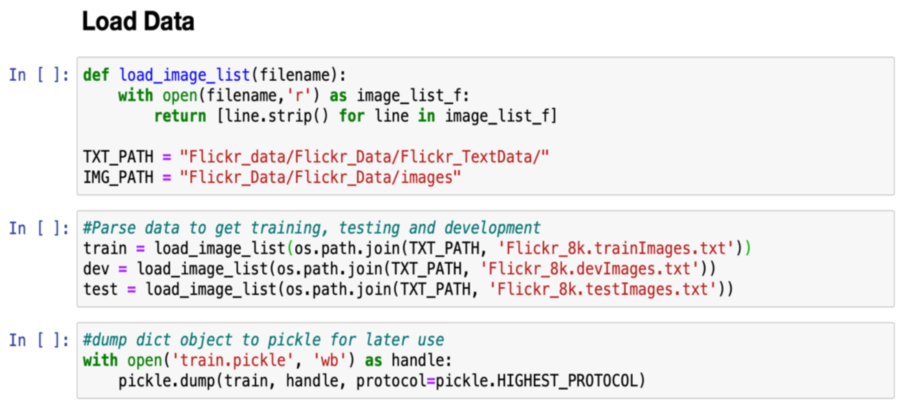


Import Libraries

```
In [ ]: # Load libraries
        import os

        import pandas as pd
        import pickle
        import numpy as np
        import random
        from collections import defaultdict
        from collections import OrderedDict

        import keras
        from keras.applications.resnet50 import ResNet50
        from keras.applications.inception_v3 import InceptionV3
        from keras.optimizers import Adam
        from keras.layers import Dense, Flatten,Input, Convolution2D, Dropout, LSTM, TimeDistributed, Embedding, Bidirection
        from keras.models import Sequential, Model
        from keras.utils import np_utils
        from keras.utils import plot_model
        from keras.callbacks import ModelCheckpoint
        from keras.callbacks import EarlyStopping
        from keras.preprocessing import image, sequence

        import matplotlib.pyplot as plt
        from IPython.display import Image, display
        import PIL
```

Figure 1: Import required libraries

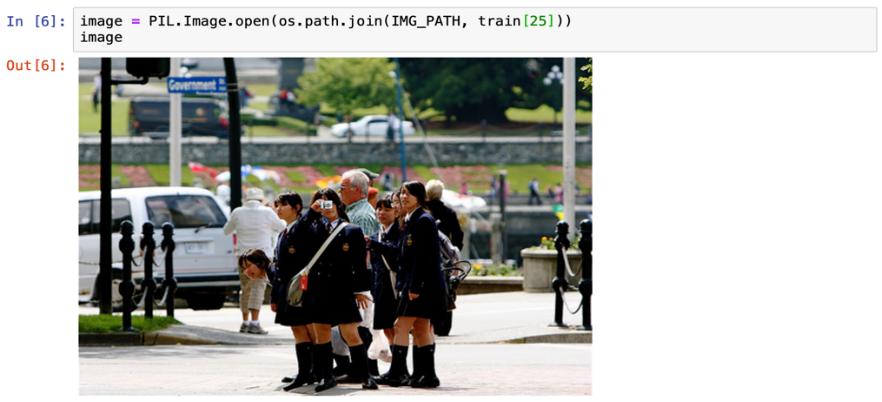
3. The code for parsing the data to get training, testing and development splits from the specified file paths in the downloaded datasets is given in the Jupyter notebook submitted alongside this configuration manual. The function load_image_list takes the filenames from the file paths and returns the image names for images in train, test and development splits. The train images are dumped to a pickle file for later use and figure 3 shows a sample image from the training set.



Load Data

```
In [ ]: def load_image_list(filename):
            with open(filename,'r') as image_list_f:
                return [line.strip() for line in image_list_f]

        TXT_PATH = "Flickr_data/Flickr_Data/Flickr_TextData/"
        IMG_PATH = "Flickr_Data/Flickr_Data/images"

In [ ]: #Parse data to get training, testing and development
        train = load_image_list(os.path.join(TXT_PATH, 'Flickr_8k.trainImages.txt'))
        dev = load_image_list(os.path.join(TXT_PATH, 'Flickr_8k.devImages.txt'))
        test = load_image_list(os.path.join(TXT_PATH, 'Flickr_8k.testImages.txt'))

In [ ]: #dump dict object to pickle for later use
        with open('train.pickle', 'wb') as handle:
            pickle.dump(train, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

Figure 2: Parse data



```
In [6]: image = PIL.Image.open(os.path.join(IMG_PATH, train[25]))
        image
Out[6]:
```

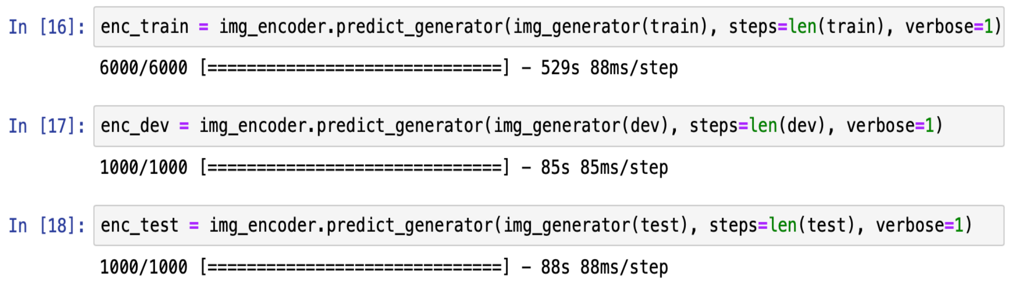
Figure 3: Sample training set image

4 Image Feature extraction

After arranging the images into train, test, and development sets, the global image features are extracted using a pretrained Inceptionv3 CNN and Places365 pretrained CNN is used for image scene feature extractions. For the Inceptionv3, the pretrained model has already been imported from Keras.applications while importing libraries. An external implementation[1] of the Places365CNN is used and the codes for these extractions are given in the Jupyter notebook. The features from the second to last layer of both models are extracted as the image global features and deep scene features respectively. These features are stored in an ordered dictionary with image names as keys and these extracted feature arrays as values before being pickled for easy access as 'traindict.pickle' and 'train_plc.pickle'.

For the global image features;

1. Convert image to array using the get_image function.
2. Load Inceptionv3 weights.
3. Set up inceptionv3 model taking the second to last layer as the output layer
4. Use the image_generator function to progressively extract the image features using the loaded model.
5. Create an ordered dictionary taking image name as key and feature vector as value.
6. Save dictionary to pickle files for future use.



```
In [16]: enc_train = img_encoder.predict_generator(img_generator(train), steps=len(train), verbose=1)
         6000/6000 [==============================] - 529s 88ms/step

In [17]: enc_dev = img_encoder.predict_generator(img_generator(dev), steps=len(dev), verbose=1)
         1000/1000 [==============================] - 85s 85ms/step

In [18]: enc_test = img_encoder.predict_generator(img_generator(test), steps=len(test), verbose=1)
         1000/1000 [==============================] - 88s 88ms/step
```

Figure 4: Finished image feature extraction process

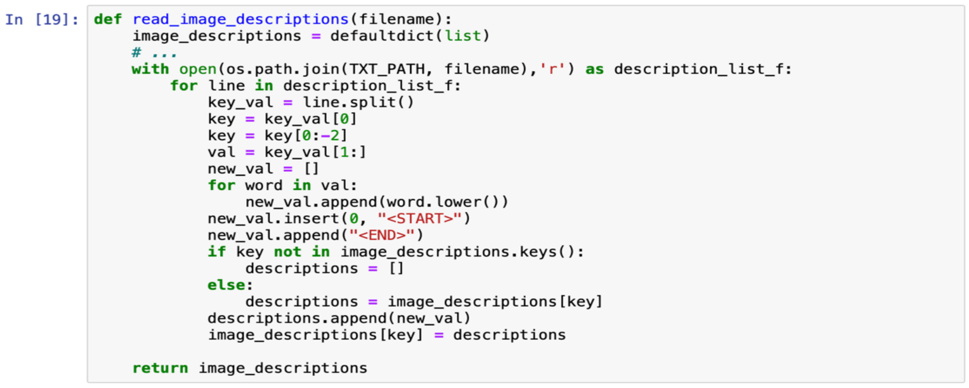
For the image scene features,

1. Import VGG16 places365 model from class file and load weights.
2. Load the images to array using the res_image function.
3. Set up loaded Places365 model to use second to last layer as output layer.
4. Use place_gen function to progressively extract image scene features using loaded Places 365 model.
5. Create ordered dictionary with image name as keys and scene feature vector as values.
6. Save files to pickle for future use.

[1] https://github.com/GKalliatakis/Keras-Application-Zoo

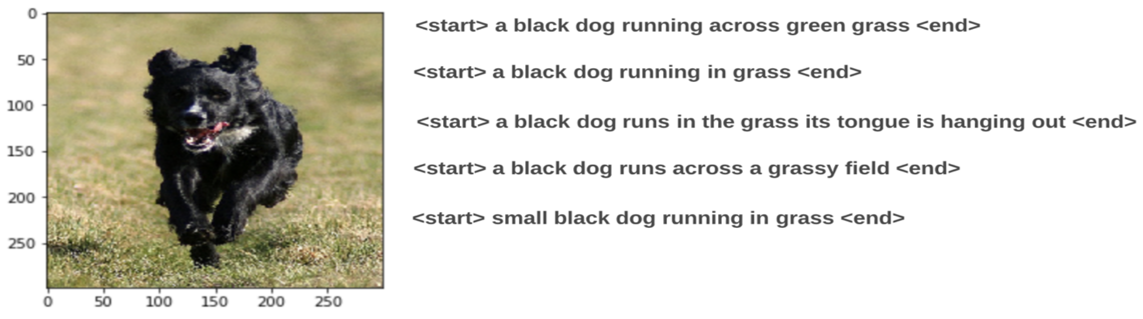
5 Text processing

During data handling, the text captions are cleaned. As part of the cleaning process, they are converted to lower case with punctuations and numbers removed. Start and end indicators '<start>' and '<end>' are added to the sentences as part of the cleaning process. The code for the text cleaning is also present in the submitted Jupyter notebook.



```
In [19]: def read_image_descriptions(filename):
             image_descriptions = defaultdict(list)
             # ...
             with open(os.path.join(TXT_PATH, filename),'r') as description_list_f:
                 for line in description_list_f:
                     key_val = line.split()
                     key = key_val[0]
                     key = key[0:-2]
                     val = key_val[1:]
                     new_val = []
                     for word in val:
                         new_val.append(word.lower())
                     new_val.insert(0, "<START>")
                     new_val.append("<END>")
                     if key not in image_descriptions.keys():
                         descriptions = []
                     else:
                         descriptions = image_descriptions[key]
                     descriptions.append(new_val)
                     image_descriptions[key] = descriptions

             return image_descriptions
```

Figure 5: Text processing function



```
<start> a black dog running across green grass <end>
<start> a black dog running in grass <end>
<start> a black dog runs in the grass its tongue is hanging out <end>
<start> a black dog runs across a grassy field <end>
<start> small black dog running in grass <end>
```

Figure 6: Sample image with processed text descriptions

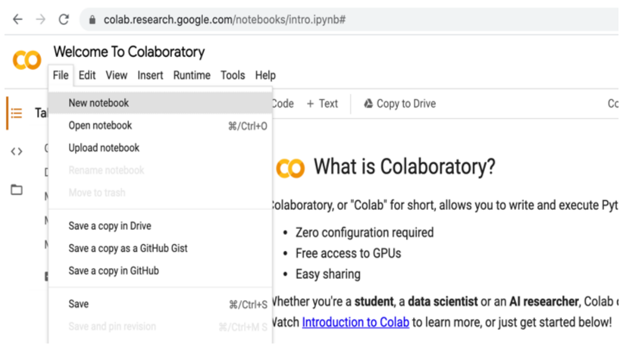
The processed text descriptions are stored in a pickle file and uploaded to Google Collaboratory for modelling.

6 Modelling

6.1 Google Colab setup

1. A Google Colab[2] free tier account is used to access a new Jupyter notebook IDE. This IDE is already preloaded with most of the required libraries for seamless python development.

[2] https://colab.research.google.com/notebooks/intro.ipynb#recent=true



What is Colaboratory?

:olaboratory, or "Colab" for short, allows you to write and execute Pyt

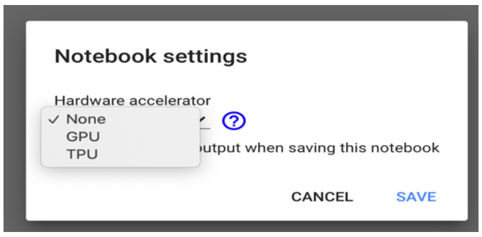
- Zero configuration required
- Free access to GPUs
- Easy sharing

Vhether you're a **student**, a **data scientist** or an **AI researcher**, Colab

Vatch Introduction to Colab to learn more, or just get started below!

Figure 7: New Google Colab environment

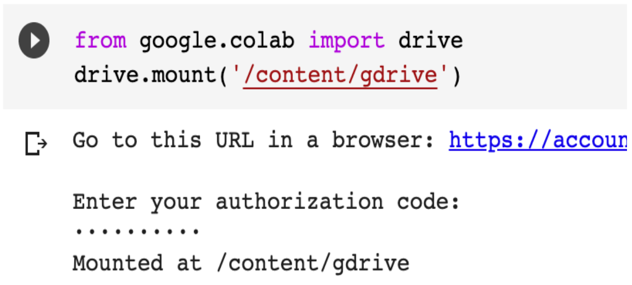
2. As part of the Colab environment set up, a GPU environment is selected to ensure faster model training times by going to Runtime → Change runtime type → GPU.



ıutput when saving this notebook

Figure 8: Selecting GPU for Google Colab

3. Upload pickle files ‘traindict.pickle’, ‘des.pickle’, ‘train.pickle’, ‘train_plc.pickle’ saved during data processing to Google drive.
4. Connect Jupyter notebook on Colab to Drive by mounting the drive on the IDE. Follow given link and copy authorization code to give access to Google drive.

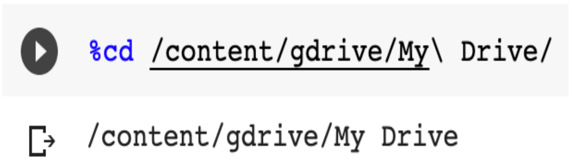


```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Go to this URL in a browser: https://accoun

Enter your authorization code:
··········
Mounted at /content/gdrive
```

Figure 9: Mount Google drive

5. Connect mounted drive to Jupyter notebook to ensure the notebook has access to uploaded pickle files.



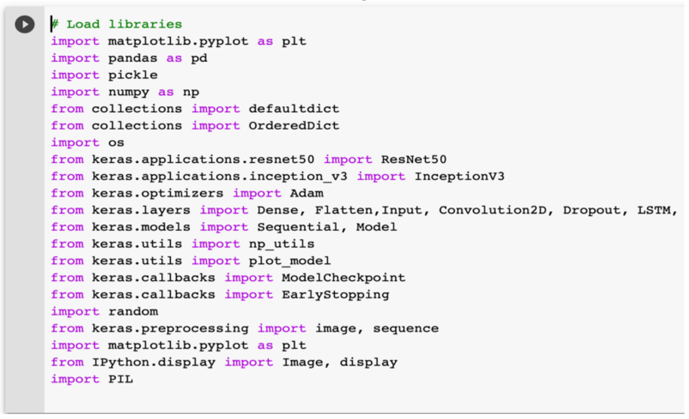
```
%cd /content/gdrive/My\ Drive/
```

```
/content/gdrive/My Drive
```

Figure 10: Connect mounted drive to Jupyter IDE

6.2 Model building

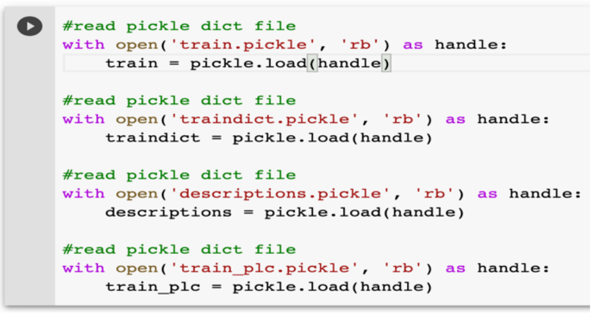
1. Import required keras[3] libraries for modelling.



```
# Load libraries
import matplotlib.pyplot as plt
import pandas as pd
import pickle
import numpy as np
from collections import defaultdict
from collections import OrderedDict
import os
from keras.applications.resnet50 import ResNet50
from keras.applications.inception_v3 import InceptionV3
from keras.optimizers import Adam
from keras.layers import Dense, Flatten,Input, Convolution2D, Dropout, LSTM,
from keras.models import Sequential, Model
from keras.utils import np_utils
from keras.utils import plot_model
from keras.callbacks import ModelCheckpoint
from keras.callbacks import EarlyStopping
import random
from keras.preprocessing import image, sequence
import matplotlib.pyplot as plt
from IPython.display import Image, display
import PIL
```

Figure 11: Import libraries for modelling

2. Read in pickle files.



```
#read pickle dict file
with open('train.pickle', 'rb') as handle:
    train = pickle.load(handle)

#read pickle dict file
with open('traindict.pickle', 'rb') as handle:
    traindict = pickle.load(handle)

#read pickle dict file
with open('descriptions.pickle', 'rb') as handle:
    descriptions = pickle.load(handle)

#read pickle dict file
with open('train_plc.pickle', 'rb') as handle:
    train_plc = pickle.load(handle)
```

Figure 12: Read in pickle files

3. Create Word to ID and ID to word dictionaries for sequence vectorization and obtain maximum length of sequences. The code to achieve this is shown in the Jupyter notebook submitted alongside.
4. Due to RAM computational limitations, use a data generator to load the data into the model in batches of 128. The code for this is also in the accompanying code file.
5. Build models.

[3] Keras.io

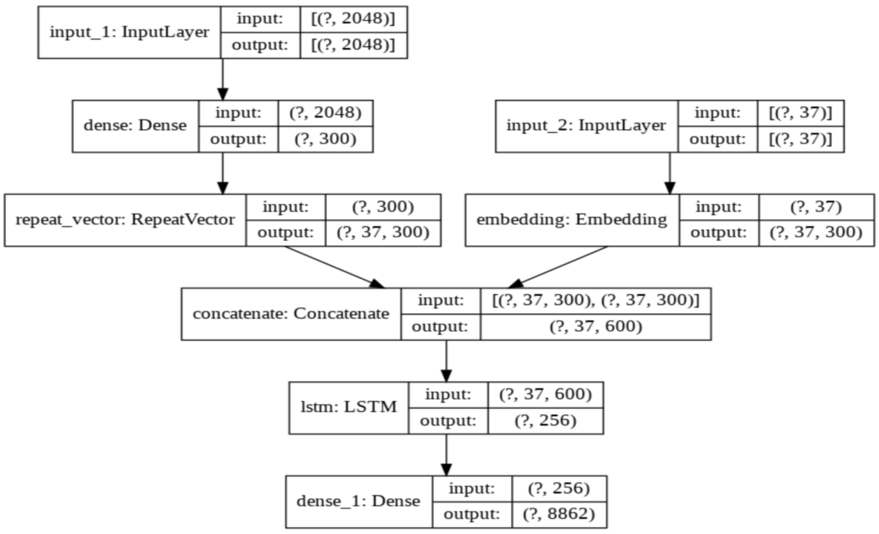


Figure 13: Experiment 1 model architecture

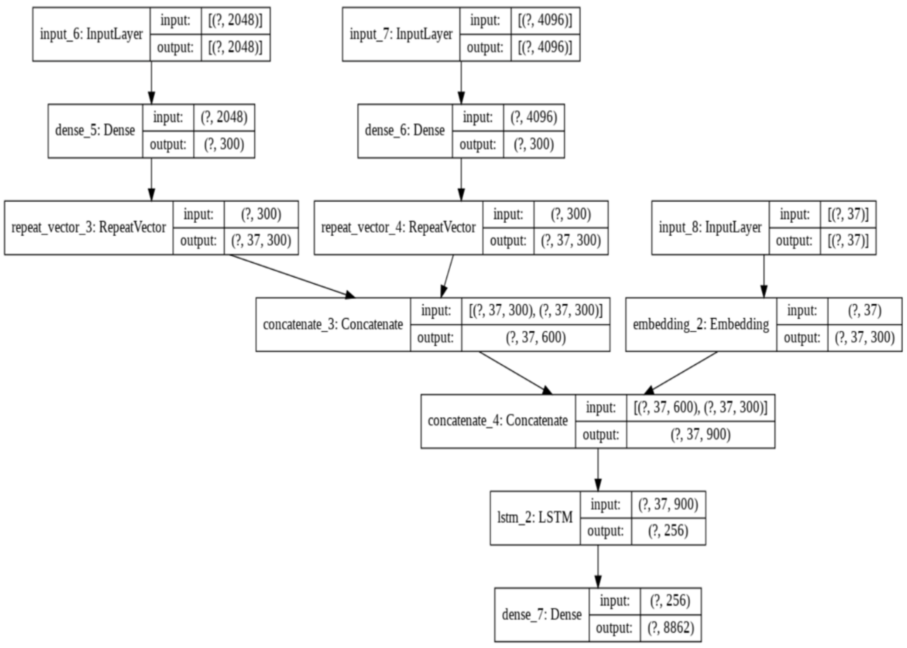


Figure 14: Experiment 2 model architecture

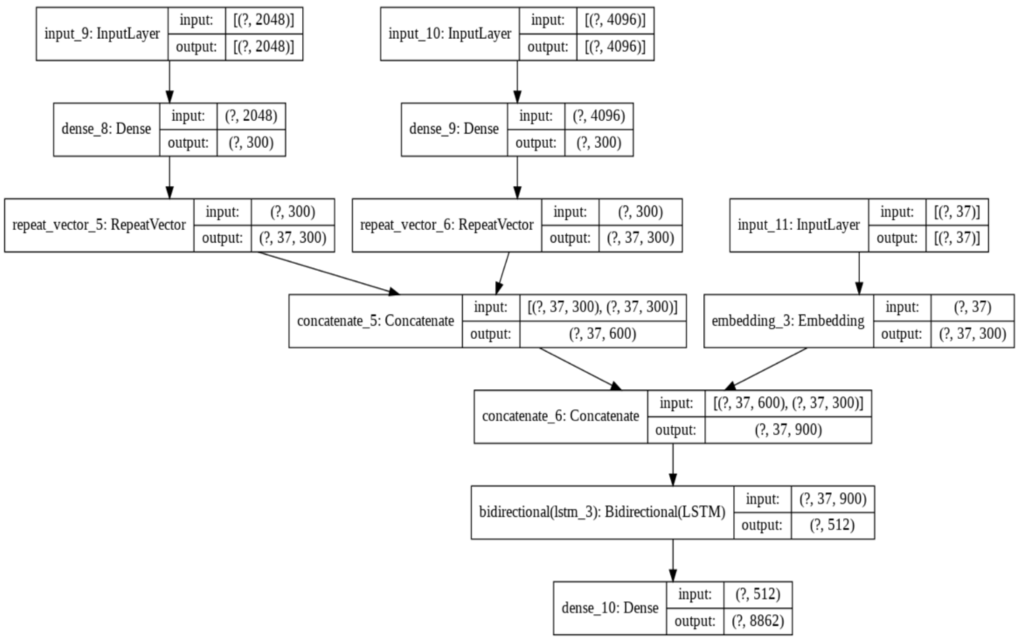

input_9: InputLayer | input: [(?, 2048)] | output: [(?, 2048)]
input_10: InputLayer | input: [(?, 4096)] | output: [(?, 4096)]
dense_8: Dense | input: (?, 2048) | output: (?, 300)
dense_9: Dense | input: (?, 4096) | output: (?, 300)
repeat_vector_5: RepeatVector | input: (?, 300) | output: (?, 37, 300)
repeat_vector_6: RepeatVector | input: (?, 300) | output: (?, 37, 300)
input_11: InputLayer | input: [(?, 37)] | output: [(?, 37)]
concatenate_5: Concatenate | input: [(?, 37, 300), (?, 37, 300)] | output: (?, 37, 600)
embedding_3: Embedding | input: (?, 37) | output: (?, 37, 300)
concatenate_6: Concatenate | input: [(?, 37, 600), (?, 37, 300)] | output: (?, 37, 900)
bidirectional(lstm_3): Bidirectional(LSTM) | input: (?, 37, 900) | output: (?, 512)
dense_10: Dense | input: (?, 512) | output: (?, 8862)

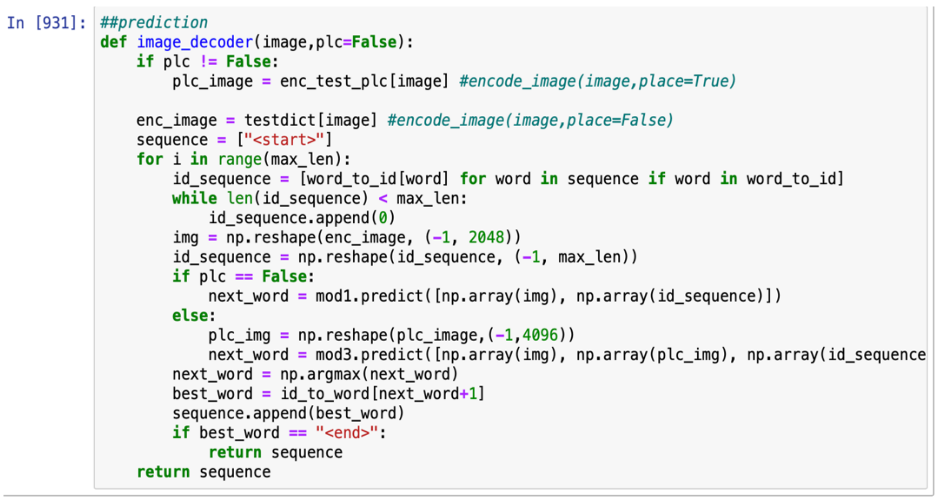
Figure 15: Experiment 3 model architecture

6. Instantiate checkpoint paths and early stopping.
7. Fit models for 15 epochs saving weights and testing at 5, 10 and 15 epochs.
8. Download saved model weights to local machine.

7 Inference and Evaluation

7.1 Inference

1. A Custom function is written using the saved weights of the models to generate sentence descriptions on the test set of the data. This function takes as input the image name and obtains the extracted image features from the saved ordered dictionary and returns a sentence description for that image.
2. Change prediction model inside the image decoder function according to experiment for which inference is being done. Note that mod2 for inference using experiment 2 model and mod3 for experiment 3 model.



```
In [931]: ##prediction
          def image_decoder(image,plc=False):
              if plc != False:
                  plc_image = enc_test_plc[image] #encode_image(image,place=True)

              enc_image = testdict[image] #encode_image(image,place=False)
              sequence = ["<start>"]
              for i in range(max_len):
                  id_sequence = [word_to_id[word] for word in sequence if word in word_to_id]
                  while len(id_sequence) < max_len:
                      id_sequence.append(0)
                  img = np.reshape(enc_image, (-1, 2048))
                  id_sequence = np.reshape(id_sequence, (-1, max_len))
                  if plc == False:
                      next_word = mod1.predict([np.array(img), np.array(id_sequence)])
                  else:
                      plc_img = np.reshape(plc_image,(-1,4096))
                      next_word = mod3.predict([np.array(img), np.array(plc_img), np.array(id_sequence
                  next_word = np.argmax(next_word)
                  best_word = id_to_word[next_word+1]
                  sequence.append(best_word)
                  if best_word == "<end>":
                      return sequence
              return sequence
```

Figure 16: Inference function

```
PREDICTED CAPTION
a man is standing on a rock above the sand .
-----------------------
GROUND TRUTH CAPTIONS
a hiker ascends a snowy hill .
a man reaches the top of a tall sand dune .
a person is hiking to the top of a hill .
a person walks up a white sandy hill against the blue sky .
the person is wearing shorts and climbing a gray sand hill under a blue sky .
```

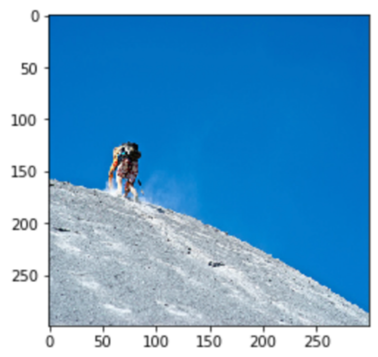
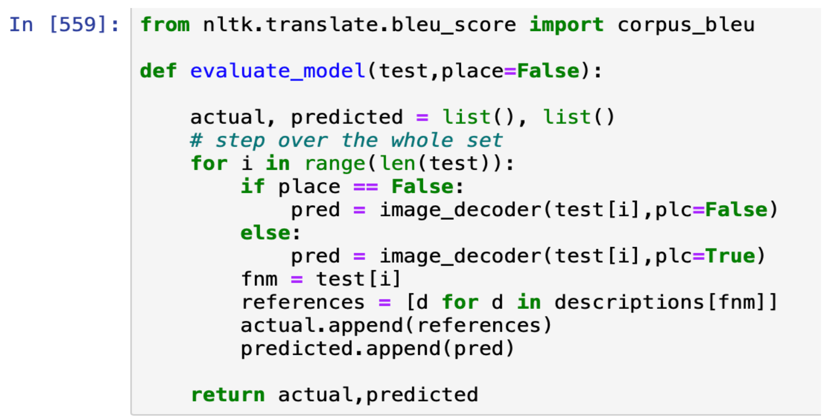


Figure 17: sample predicted and ground truth captions

7.2 Metrics Evaluation

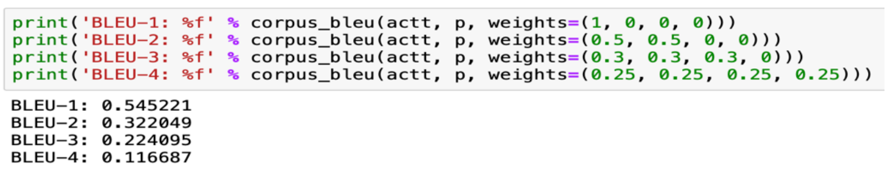
1. Another custom function is written using the previous inference function to generate sentence descriptions across the entire text corpus. The function takes as input a list containing test image names and returns the actual and predicted captions.



```
In [559]: from nltk.translate.bleu_score import corpus_bleu

          def evaluate_model(test,place=False):

              actual, predicted = list(), list()
              # step over the whole set
              for i in range(len(test)):
                  if place == False:
                      pred = image_decoder(test[i],plc=False)
                  else:
                      pred = image_decoder(test[i],plc=True)
                  fnm = test[i]
                  references = [d for d in descriptions[fnm]]
                  actual.append(references)
                  predicted.append(pred)

              return actual,predicted
```

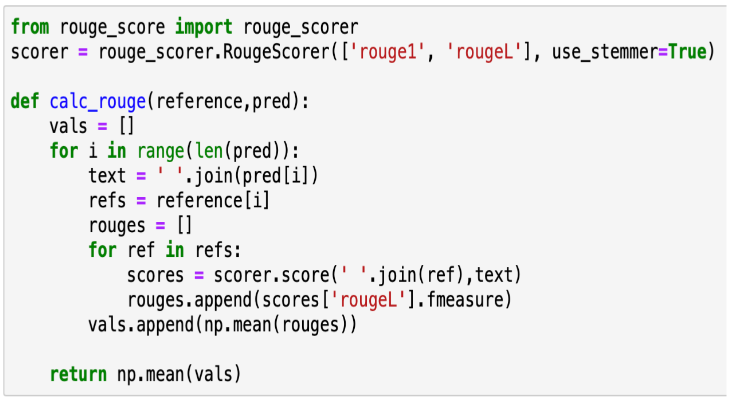
Figure 18: Function to make predictions across the entire test corpus

2. Remove <start> and <end> tokens from predicted and actual sentence descriptions.
3. Evaluate actual and predicted captions using BLEU and ROUGE scores.



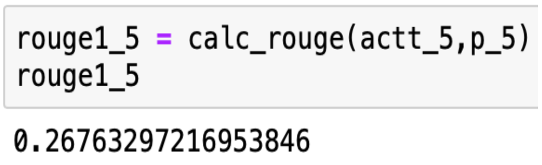
```
print('BLEU-1: %f' % corpus_bleu(actt, p, weights=(1, 0, 0, 0)))
print('BLEU-2: %f' % corpus_bleu(actt, p, weights=(0.5, 0.5, 0, 0)))
print('BLEU-3: %f' % corpus_bleu(actt, p, weights=(0.3, 0.3, 0.3, 0)))
print('BLEU-4: %f' % corpus_bleu(actt, p, weights=(0.25, 0.25, 0.25, 0.25)))

BLEU-1: 0.545221
BLEU-2: 0.322049
BLEU-3: 0.224095
BLEU-4: 0.116687
```

Figure 19: Sample BLEU evaluation



```python
from rouge_score import rouge_scorer
scorer = rouge_scorer.RougeScorer(['rouge1', 'rougeL'], use_stemmer=True)

def calc_rouge(reference,pred):
    vals = []
    for i in range(len(pred)):
        text = ' '.join(pred[i])
        refs = reference[i]
        rouges = []
        for ref in refs:
            scores = scorer.score(' '.join(ref),text)
            rouges.append(scores['rougeL'].fmeasure)
        vals.append(np.mean(rouges))

    return np.mean(vals)
```

Figure 20: ROUGE evaluation function



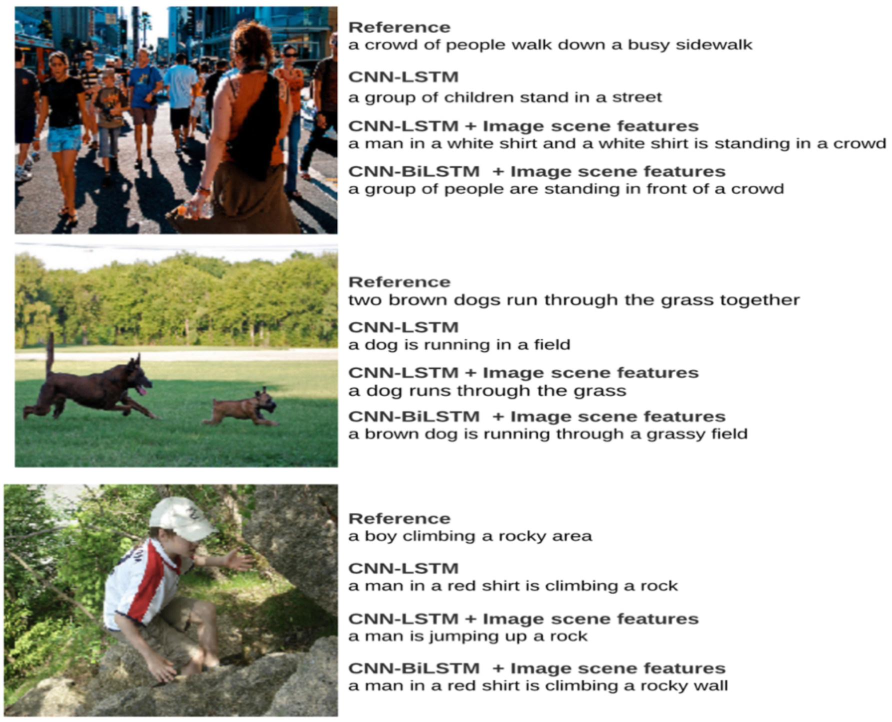
```python
rouge1_5 = calc_rouge(actt_5,p_5)
rouge1_5
```

```
0.26763297216953846
```

Figure 21: Sample ROUGE evaluation

7.3 Qualitative Evaluation

A qualitative analysis of the experiments is undertaken to evaluate the effect of scene features on image captioning and samples are shown in the figure below.



Reference
a crowd of people walk down a busy sidewalk

CNN-LSTM
a group of children stand in a street

CNN-LSTM + Image scene features
a man in a white shirt and a white shirt is standing in a crowd

CNN-BiLSTM + Image scene features
a group of people are standing in front of a crowd

Reference
two brown dogs run through the grass together

CNN-LSTM
a dog is running in a field

CNN-LSTM + Image scene features
a dog runs through the grass

CNN-BiLSTM + Image scene features
a brown dog is running through a grassy field

Reference
a boy climbing a rocky area

CNN-LSTM
a man in a red shirt is climbing a rock

CNN-LSTM + Image scene features
a man is jumping up a rock

CNN-BiLSTM + Image scene features
a man in a red shirt is climbing a rocky wall

Figure 22: Model qualitative analysis