Configuration Manual

MSc Research Project
Data Analytics

Namita Mohan
Student ID: 19212500

School of Computing
National College of Ireland

Supervisor: Dr Paul Stynes and Dr Pramod Pathak

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Namita Mohan
Student ID:	19212500
Programme:	Data Analytics
Year:	2020/2021
Module:	MSc Research Project
Supervisor:	Dr Paul Stynes and Dr Pramod Pathak
Submission Due Date:	16/08/2021
Project Title:	Configuration Manual
Word Count:	1109
Page Count:	11

Signature:	Namita Mohan
Date:	18th September 2021

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.	☐

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

Configuration Manual

Namita Mohan
19212500

1 Introduction

This configuration manual will discuss steps followed in the implementation of this research in detail. The document covers steps from data collection to the evaluation of the implemented remote sensing image captioning framework. It includes system configuration, code snippets and directions to execute to ensure that this work can be reproduced in future easily.

2 Hardware and Software configurations

All the execution for this research is done on the local machine. To track progress and save time, processed data and models were saved locally as pickle files. Table 1 and Table 2 presents hardware and software specifications for this research. It is also suggested to use the latest version of Tensorflow and Keras for this implementation.

Table 1: Hardware specifications

Host machine/Operating System	MacBook Pro/MacOS Mojave
RAM	8GB, 2.7 GHz Intel Core i5
Hard Disk	256GB
GPU	Intel Iris Graphics 6100 1536 MB

Table 2: Software specifications

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

3 Data Preparation

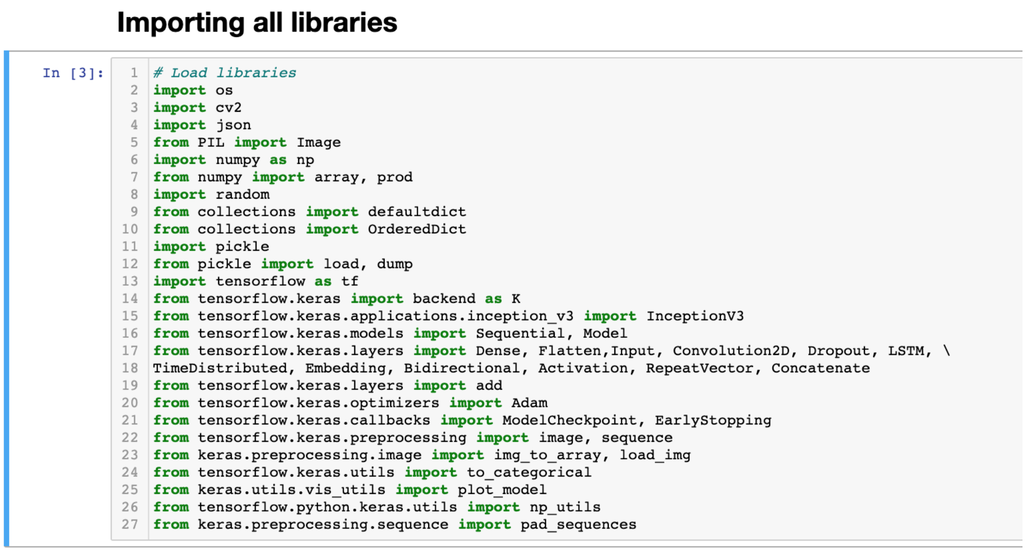
3.1 Collecting Data

Download the RSICD dataset on your local machine from GitHub using the URL https://github.com/201528014227051/RSICD_optimal. The dataset includes images folder and a JSON file with key as image name and value as an array of text descriptions.

3.2 Reading Data

1. Create a new Jupyter notebook environment in the same folder where the dataset is stored.

2. Import all required libraries for data processing.

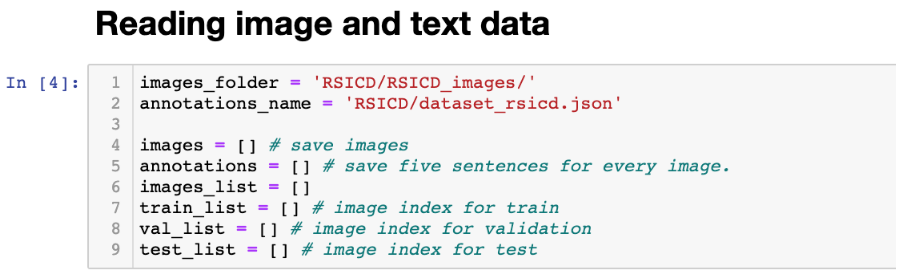


Importing all libraries

```
In [3]:
# Load libraries
import os
import cv2
import json
from PIL import Image
import numpy as np
from numpy import array, prod
import random
from collections import defaultdict
from collections import OrderedDict
import pickle
from pickle import load, dump
import tensorflow as tf
from tensorflow.keras import backend as K
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Dense, Flatten,Input, Convolution2D, Dropout, LSTM, \
TimeDistributed, Embedding, Bidirectional, Activation, RepeatVector, Concatenate
from tensorflow.keras.layers import add
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
from tensorflow.keras.preprocessing import image, sequence
from keras.preprocessing.image import img_to_array, load_img
from tensorflow.keras.utils import to_categorical
from keras.utils.vis_utils import plot_model
from tensorflow.python.keras.utils import np_utils
from keras.preprocessing.sequence import pad_sequences
```

Figure 1: Import required libraries.

3. Define directory parth for images and json file to read data.



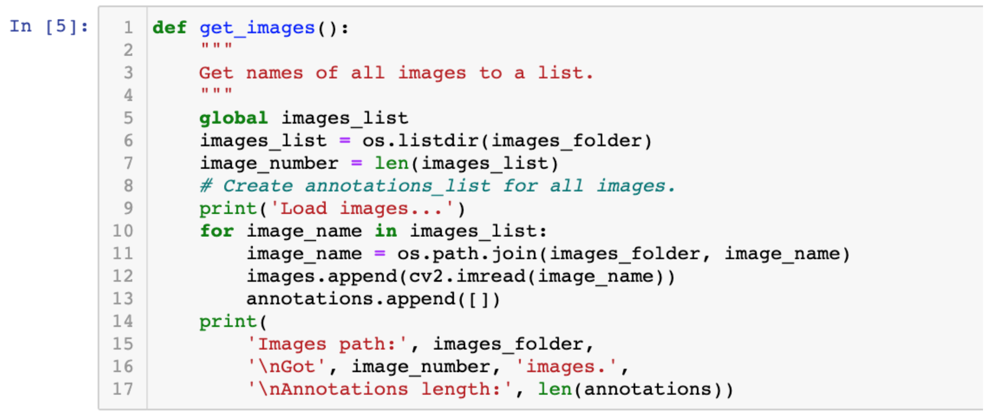
Reading image and text data

```
In [4]:
images_folder = 'RSICD/RSICD_images/'
annotations_name = 'RSICD/dataset_rsicd.json'

images = [] # save images
annotations = [] # save five sentences for every image.
images_list = []
train_list = [] # image index for train
val_list = [] # image index for validation
test_list = [] # image index for test
```

Figure 2: Defing directory paths for the data.

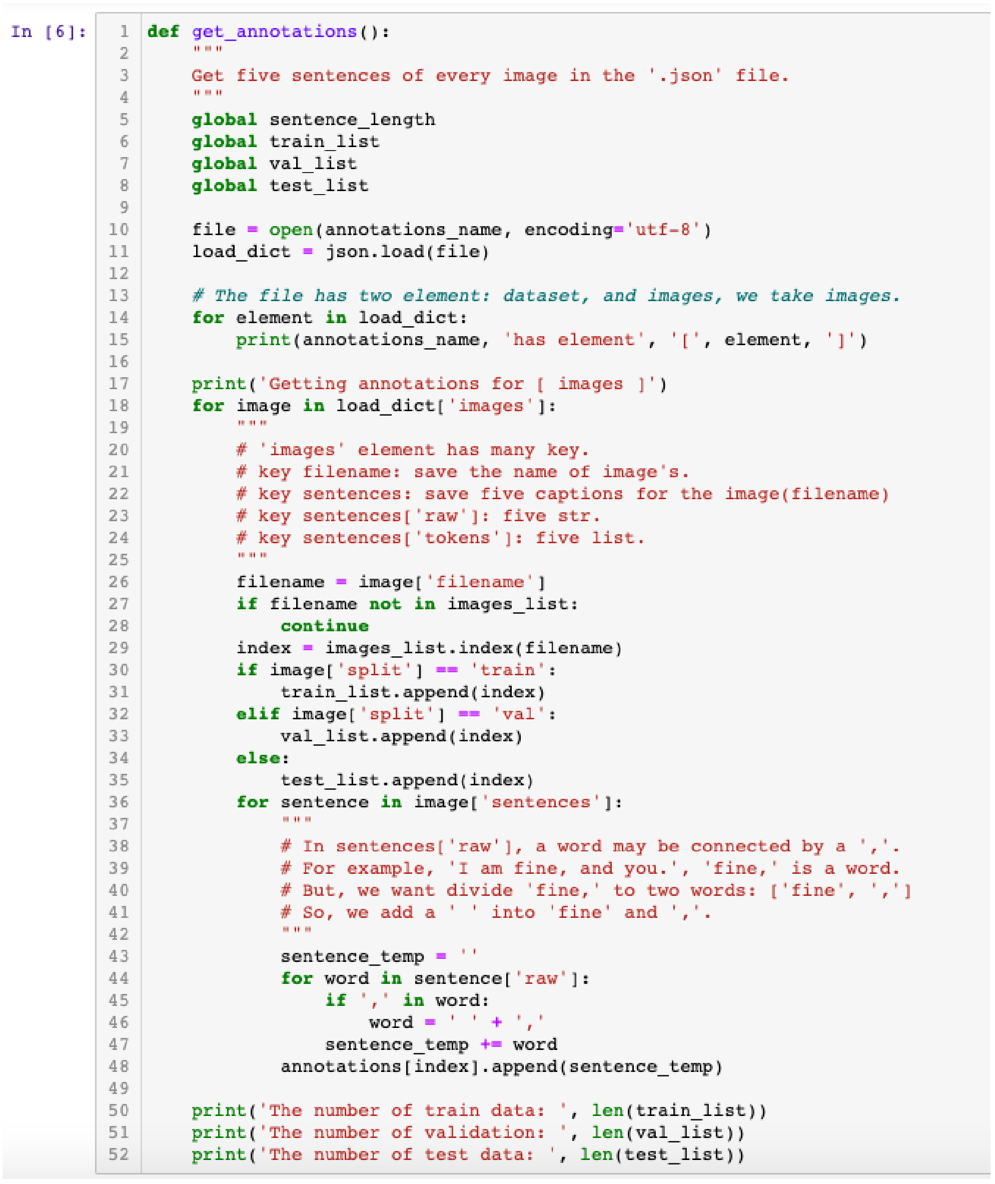
4. Load images into notebook using OS package of python. The get_image function defined in Figure 3 creates a global list containng names of the images in the given directory. Also, a list of images and empty list of same length is also created to store reference descriptions.

5. To load text data into notebook use JSON package of python. Split the loaded data into three parts using the 'split' key of the json for training, testing, and validation of the model. The fuction get_annotations is used to parse json data and create three lists containing index of the image in the global image_list for training, testing and validation.



```python
def get_images():
    """
    Get names of all images to a list.
    """
    global images_list
    images_list = os.listdir(images_folder)
    image_number = len(images_list)
    # Create annotations_list for all images.
    print('Load images...')
    for image_name in images_list:
        image_name = os.path.join(images_folder, image_name)
        images.append(cv2.imread(image_name))
        annotations.append([])
    print(
        'Images path:', images_folder,
        '\nGot', image_number, 'images.',
        '\nAnnotations length:', len(annotations))
```

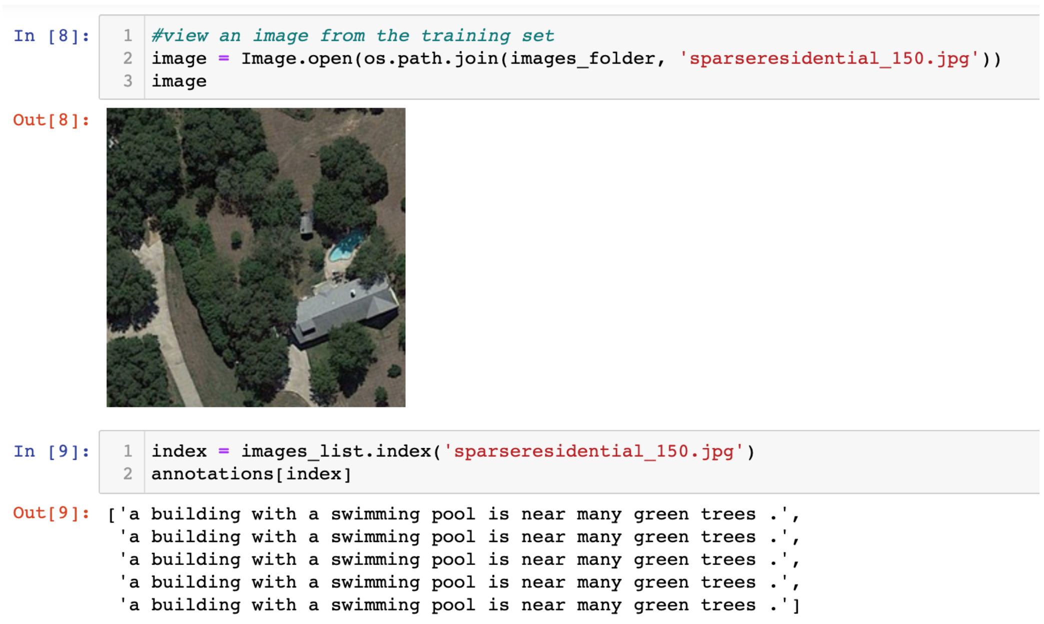
Figure 3: Parsing image data.



```python
def get_annotations():
    """
    Get five sentences of every image in the '.json' file.
    """
    global sentence_length
    global train_list
    global val_list
    global test_list

    file = open(annotations_name, encoding='utf-8')
    load_dict = json.load(file)

    # The file has two element: dataset, and images, we take images.
    for element in load_dict:
        print(annotations_name, 'has element', '[', element, ']')

    print('Getting annotations for [ images ]')
    for image in load_dict['images']:
        """
        # 'images' element has many key.
        # key filename: save the name of image's.
        # key sentences: save five captions for the image(filename)
        # key sentences['raw']: five str.
        # key sentences['tokens']: five list.
        """
        filename = image['filename']
        if filename not in images_list:
            continue
        index = images_list.index(filename)
        if image['split'] == 'train':
            train_list.append(index)
        elif image['split'] == 'val':
            val_list.append(index)
        else:
            test_list.append(index)
        for sentence in image['sentences']:
            """
            # In sentences['raw'], a word may be connected by a ','.
            # For example, 'I am fine, and you.', 'fine,' is a word.
            # But, we want divide 'fine,' to two words: ['fine', ',']
            # So, we add a ' ' into 'fine' and ','.
            """
            sentence_temp = ''
            for word in sentence['raw']:
                if ',' in word:
                    word = ' ' + ','
                sentence_temp += word
            annotations[index].append(sentence_temp)

    print('The number of train data: ', len(train_list))
    print('The number of validation: ', len(val_list))
    print('The number of test data: ', len(test_list))
```

Figure 4: Parsing text data.

6. Figure 5 represents a sample image from the loaded RSICD dataset along with references text descriptions.



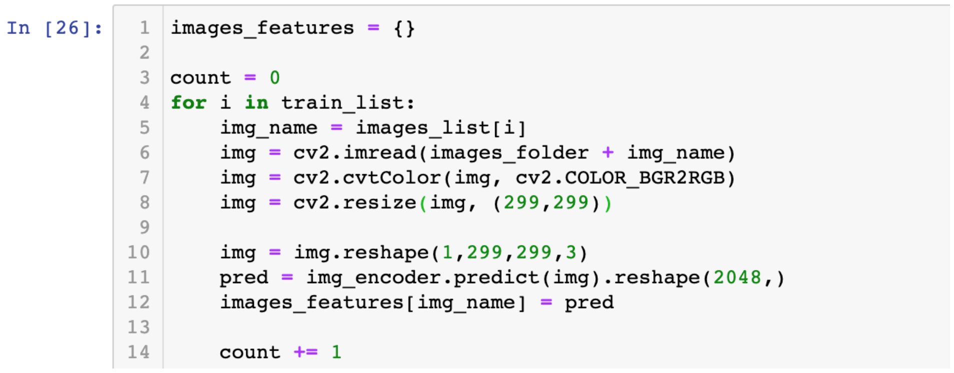
```
In [8]:  #view an image from the training set
         image = Image.open(os.path.join(images_folder, 'sparseresidential_150.jpg'))
         image
Out[8]:
```

```
In [9]:  index = images_list.index('sparseresidential_150.jpg')
         annotations[index]

Out[9]: ['a building with a swimming pool is near many green trees .',
         'a building with a swimming pool is near many green trees .',
         'a building with a swimming pool is near many green trees .',
         'a building with a swimming pool is near many green trees .',
         'a building with a swimming pool is near many green trees .']
```

Figure 5: Model Architecture.

4 Image Processing

The next step is image processing and transformation to generate image feature vectors. In image processing, loaded image are resized as required by the encoder model. Figure 6 shows the code for feature extraction including resizing and getting feature vector from the model.



```
In [26]:  images_features = {}

          count = 0
          for i in train_list:
              img_name = images_list[i]
              img = cv2.imread(images_folder + img_name)
              img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
              img = cv2.resize(img, (299,299))

              img = img.reshape(1,299,299,3)
              pred = img_encoder.predict(img).reshape(2048,)
              images_features[img_name] = pred

              count += 1
```

Figure 6: Image feature extraction process.

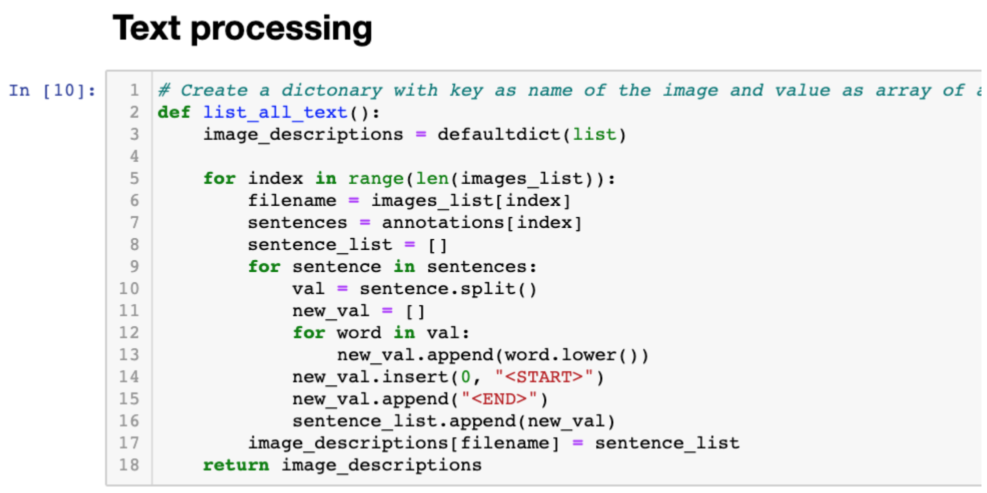
This research used two type of encoder model CNN and Capsule Network. For the CNN, a pre-trained InceptionV3 model is used which is avaible in keras applications [1]. For Capsule Network, a encoder model is created including 4 casule units which are

[1]https://keras.io/api/applications/inceptionv3/

flatten together in a layer. The architecture of the capsule network is inspired from the model proposed in the study Deka (2020), which is available on GitHub [2].

5 Text Processing

1. In this step, text descriptions are cleaned and transformed for modelling. For cleaning text descriptions first each word is converted into the lower case to avoid replication of words in the vocabulary and <START> and <END> tokens are added to mark the beginning and ending of a sentence. Figure 7 presents the code for this part.

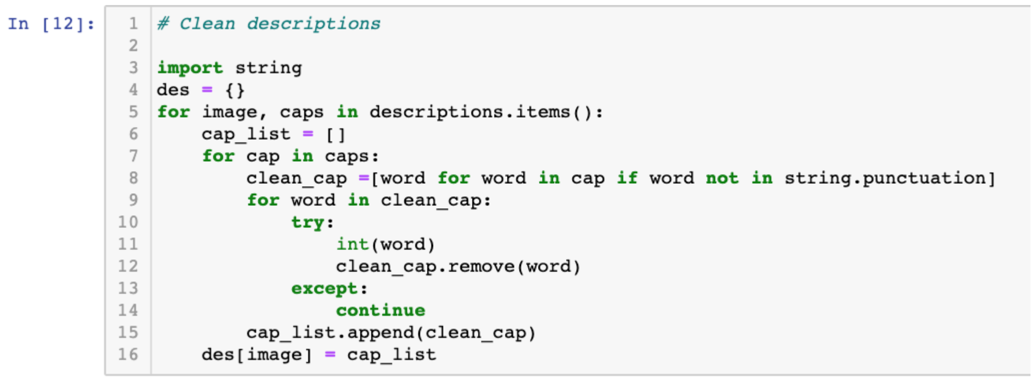


Text processing

In [10]:
```python
# Create a dictonary with key as name of the image and value as array of
def list_all_text():
    image_descriptions = defaultdict(list)

    for index in range(len(images_list)):
        filename = images_list[index]
        sentences = annotations[index]
        sentence_list = []
        for sentence in sentences:
            val = sentence.split()
            new_val = []
            for word in val:
                new_val.append(word.lower())
            new_val.insert(0, "<START>")
            new_val.append("<END>")
            sentence_list.append(new_val)
        image_descriptions[filename] = sentence_list
    return image_descriptions
```

Figure 7: Coverting to lower case and adding start and end tokens.

2. The next step is to remove punctuations from the description. Code used for removing punctuations is shown in Figure 8.



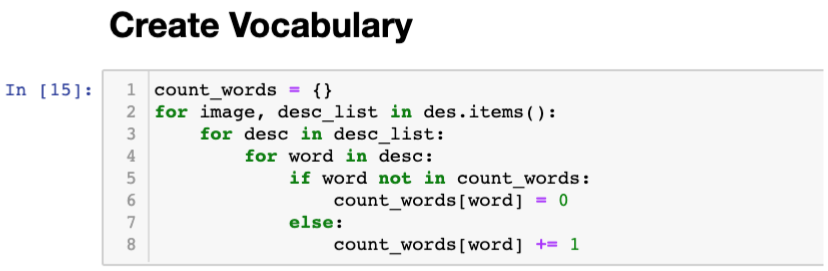
In [12]:
```python
# Clean descriptions

import string
des = {}
for image, caps in descriptions.items():
    cap_list = []
    for cap in caps:
        clean_cap =[word for word in cap if word not in string.punctuation]
        for word in clean_cap:
            try:
                int(word)
                clean_cap.remove(word)
            except:
                continue
        cap_list.append(clean_cap)
    des[image] = cap_list
```

Figure 8: Removing punctuations.

3. After cleaning the text descriptions, the next step is to calculate vocabulary size for modelling. The code in Figure 9 is followed for this step.

4. The last step in text processing is the tokenization of the text descriptions. A dictionary is created to convert words to numbers as showing in Figure 10.

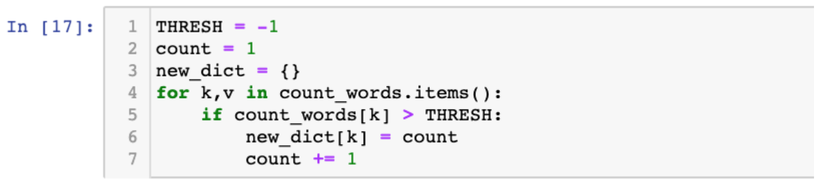
[2]https://github.com/jaydeepdeka/Flickr8k_image_captioning_using_CapsNet



Create Vocabulary

```
In [15]:
count_words = {}
for image, desc_list in des.items():
    for desc in desc_list:
        for word in desc:
            if word not in count_words:
                count_words[word] = 0
            else:
                count_words[word] += 1
```

Figure 9: Counting total words in vocabulary.

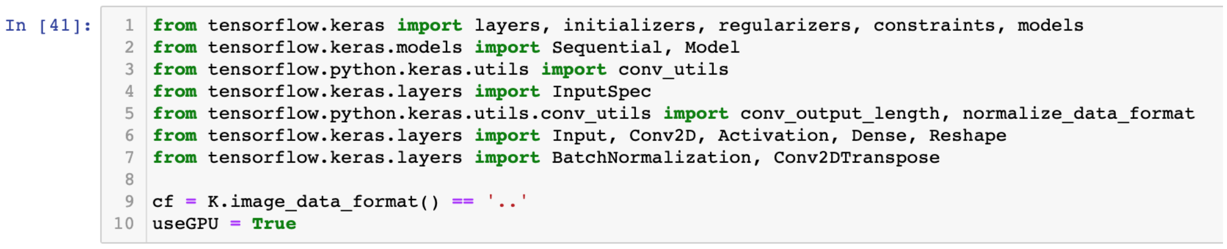


```
In [17]:
THRESH = -1
count = 1
new_dict = {}
for k,v in count_words.items():
    if count_words[k] > THRESH:
        new_dict[k] = count
        count += 1
```

Figure 10: Creating word to number dictionary.

6 Modelling

1. Importing remaining libraries required for modelling.



```
In [41]:
from tensorflow.keras import layers, initializers, regularizers, constraints, models
from tensorflow.keras.models import Sequential, Model
from tensorflow.python.keras.utils import conv_utils
from tensorflow.keras.layers import InputSpec
from tensorflow.python.keras.utils.conv_utils import conv_output_length, normalize_data_format
from tensorflow.keras.layers import Input, Conv2D, Activation, Dense, Reshape
from tensorflow.keras.layers import BatchNormalization, Conv2DTranspose

cf = K.image_data_format() == '..'
useGPU = True
```

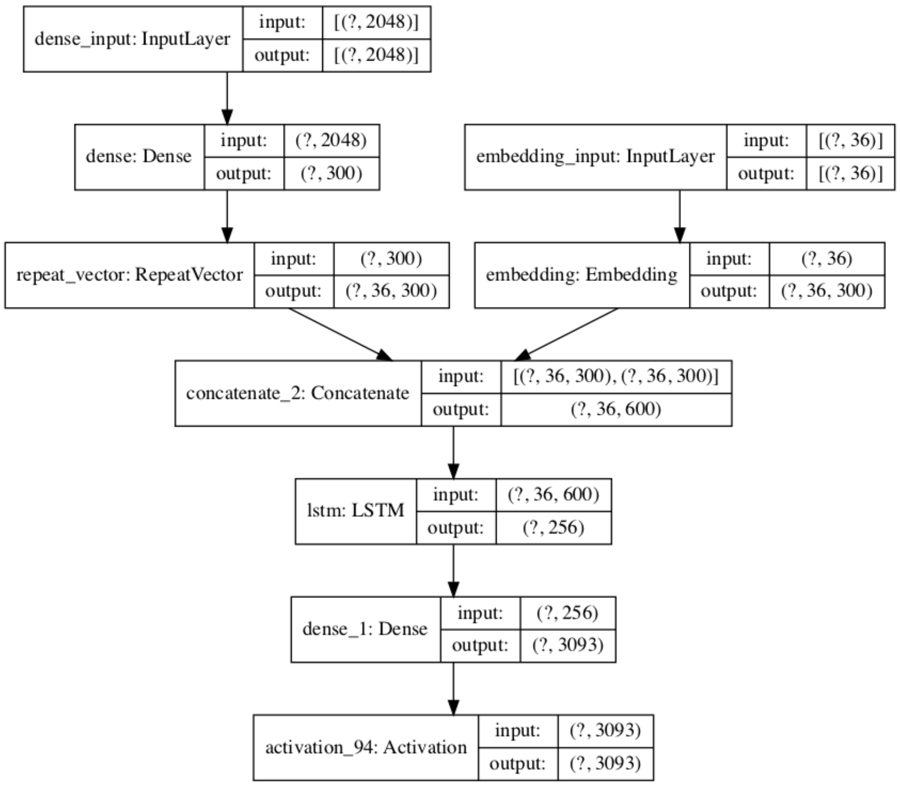
Figure 11: Importing libraries for modelling.

2. Creating a model as shown in Figure 12, which takes image feature vector and word vector to generate a description. The architecture of this model is inspired by the study of (Agughalam et al.; 2021).

3. For the second experiment, an encoder-decoder framework uses a feature vector generated by CapsNet encoder to generate text sequences. The architecture of the model is presented in Figure 13.

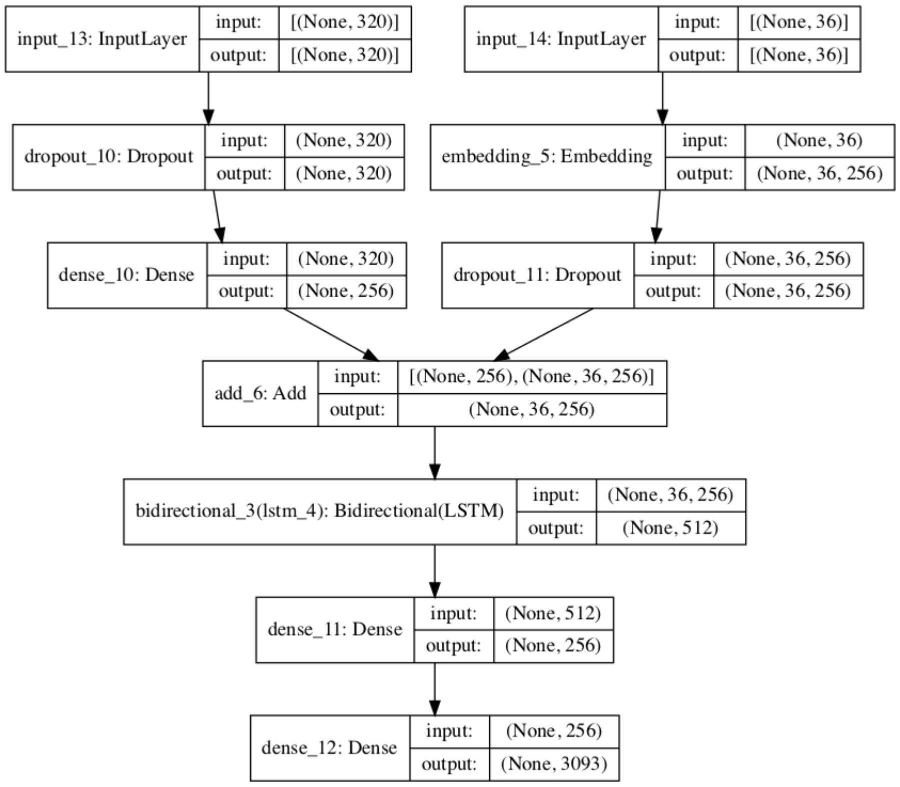
4. For using CNN and CapsNet features together for generating text descriptions in experiment three, the architecture in Figure 14 is implemented. It takes three inputs, first for CNN feature vector, second for CapsNet feature vector and third for word vector.

5. In the training of the models 10 epochs are used. The model with the lowest validation loss is saved and used for generating text sequences. The code in Figure 15 is used for the training of models.



dense_input: InputLayer | input: [(?, 2048)] | output: [(?, 2048)]
dense: Dense | input: (?, 2048) | output: (?, 300)
embedding_input: InputLayer | input: [(?, 36)] | output: [(?, 36)]
repeat_vector: RepeatVector | input: (?, 300) | output: (?, 36, 300)
embedding: Embedding | input: (?, 36) | output: (?, 36, 300)
concatenate_2: Concatenate | input: [(?, 36, 300), (?, 36, 300)] | output: (?, 36, 600)
lstm: LSTM | input: (?, 36, 600) | output: (?, 256)
dense_1: Dense | input: (?, 256) | output: (?, 3093)
activation_94: Activation | input: (?, 3093) | output: (?, 3093)

Figure 12: CNN-LSTM encoder-decoder architecture.



input_13: InputLayer | input: [(None, 320)] | output: [(None, 320)]
input_14: InputLayer | input: [(None, 36)] | output: [(None, 36)]
dropout_10: Dropout | input: (None, 320) | output: (None, 320)
embedding_5: Embedding | input: (None, 36) | output: (None, 36, 256)
dense_10: Dense | input: (None, 320) | output: (None, 256)
dropout_11: Dropout | input: (None, 36, 256) | output: (None, 36, 256)
add_6: Add | input: [(None, 256), (None, 36, 256)] | output: (None, 36, 256)
bidirectional_3(lstm_4): Bidirectional(LSTM) | input: (None, 36, 256) | output: (None, 512)
dense_11: Dense | input: (None, 512) | output: (None, 256)
dense_12: Dense | input: (None, 256) | output: (None, 3093)

Figure 13: CapsNet-BiLSTM encoder-decoder architecture.

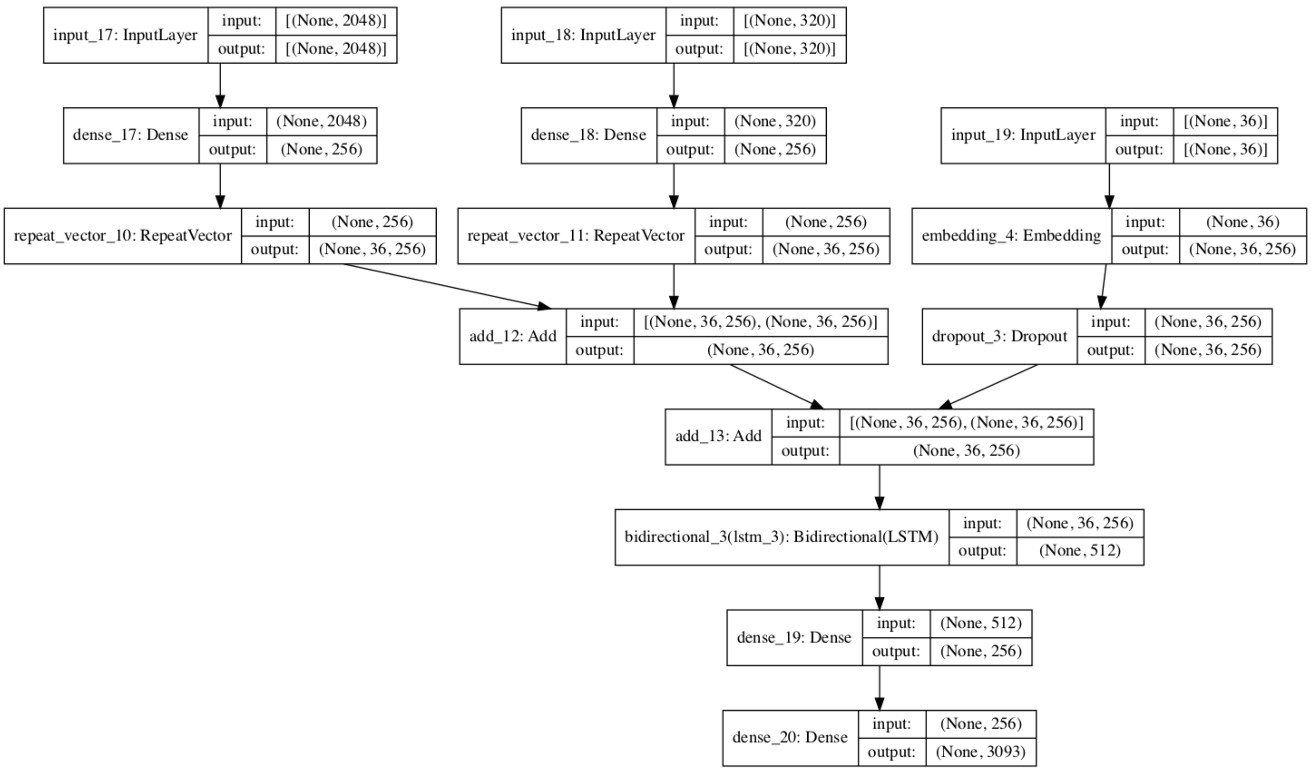
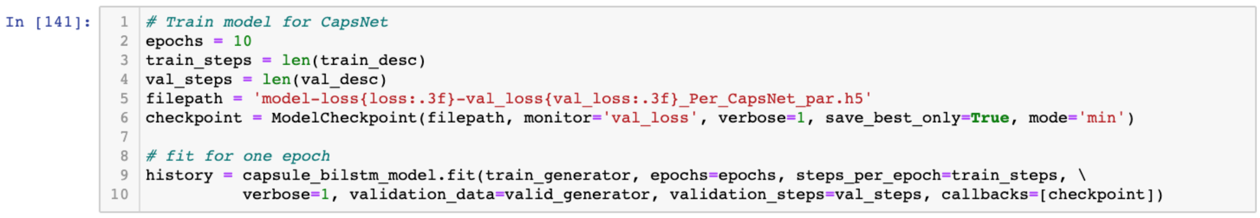

input_17: InputLayer | input: [(None, 2048)] | output: [(None, 2048)]
input_18: InputLayer | input: [(None, 320)] | output: [(None, 320)]
dense_17: Dense | input: (None, 2048) | output: (None, 256)
dense_18: Dense | input: (None, 320) | output: (None, 256)
input_19: InputLayer | input: [(None, 36)] | output: [(None, 36)]
repeat_vector_10: RepeatVector | input: (None, 256) | output: (None, 36, 256)
repeat_vector_11: RepeatVector | input: (None, 256) | output: (None, 36, 256)
embedding_4: Embedding | input: (None, 36) | output: (None, 36, 256)
add_12: Add | input: [(None, 36, 256), (None, 36, 256)] | output: (None, 36, 256)
dropout_3: Dropout | input: (None, 36, 256) | output: (None, 36, 256)
add_13: Add | input: [(None, 36, 256), (None, 36, 256)] | output: (None, 36, 256)
bidirectional_3(lstm_3): Bidirectional(LSTM) | input: (None, 36, 256) | output: (None, 512)
dense_19: Dense | input: (None, 512) | output: (None, 256)
dense_20: Dense | input: (None, 256) | output: (None, 3093)

Figure 14: CNN-CapsNet-BiLSTM encoder-decoder architecture.



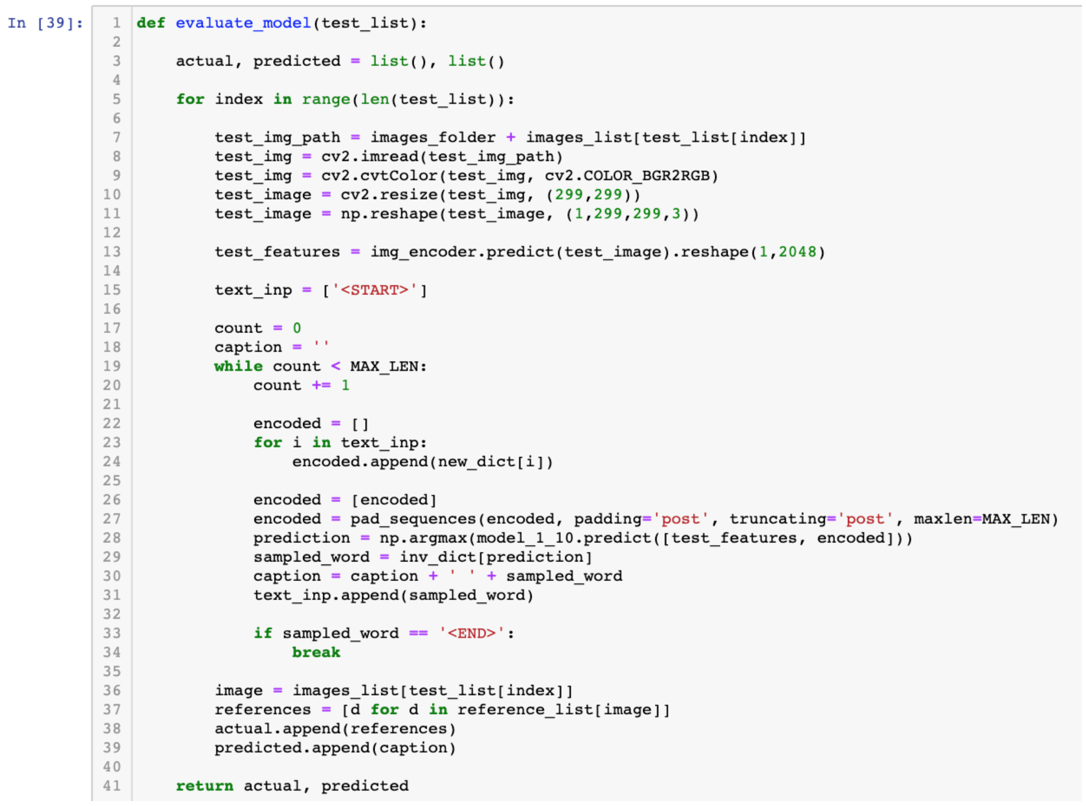
In [141]:
```
# Train model for CapsNet
epochs = 10
train_steps = len(train_desc)
val_steps = len(val_desc)
filepath = 'model-loss{loss:.3f}-val_loss{val_loss:.3f}_Per_CapsNet_par.h5'
checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=1, save_best_only=True, mode='min')

# fit for one epoch
history = capsule_bilstm_model.fit(train_generator, epochs=epochs, steps_per_epoch=train_steps, \
          verbose=1, validation_data=valid_generator, validation_steps=val_steps, callbacks=[checkpoint])
```

Figure 15: Training model.

7 Inference and Evaluation

7.1 Inference

1. The function evaluation_model takes a list of test images and generate a text description for each image. First, the image is resized as required for the encoder model and the feature vector is generated. Then text sequence is predicted using that feature vector.



```
In [39]:
def evaluate_model(test_list):

    actual, predicted = list(), list()

    for index in range(len(test_list)):

        test_img_path = images_folder + images_list[test_list[index]]
        test_img = cv2.imread(test_img_path)
        test_img = cv2.cvtColor(test_img, cv2.COLOR_BGR2RGB)
        test_image = cv2.resize(test_img, (299,299))
        test_image = np.reshape(test_image, (1,299,299,3))

        test_features = img_encoder.predict(test_image).reshape(1,2048)

        text_inp = ['<START>']

        count = 0
        caption = ''
        while count < MAX_LEN:
            count += 1

            encoded = []
            for i in text_inp:
                encoded.append(new_dict[i])

            encoded = [encoded]
            encoded = pad_sequences(encoded, padding='post', truncating='post', maxlen=MAX_LEN)
            prediction = np.argmax(model_1_10.predict([test_features, encoded]))
            sampled_word = inv_dict[prediction]
            caption = caption + ' ' + sampled_word
            text_inp.append(sampled_word)

            if sampled_word == '<END>':
                break

        image = images_list[test_list[index]]
        references = [d for d in reference_list[image]]
        actual.append(references)
        predicted.append(caption)

    return actual, predicted
```

Figure 16: .

2. The word with the highest probability is selected as the next word in a sentence. Like this, all words are appended together to form a sentence.

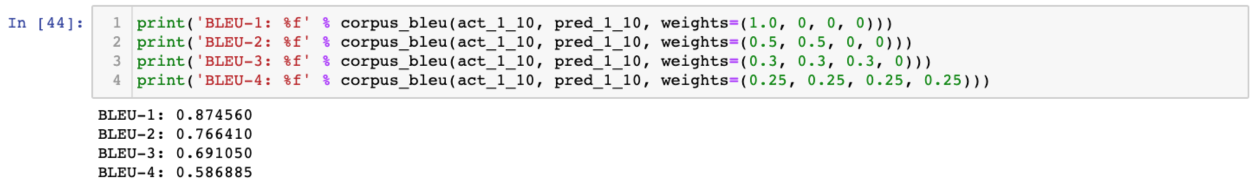
3. For all three experiments predicted text description is generated for the test images and stored in pickle files.

7.2 Evaluation

1. Remove <START> and <END> tokens from begining and ending of predicted descriptions for evaluation.

2. Calculate BLEU and ROUGE-L scores for each model. Figure 17 and Figure 18 presents the code for BLEU and ROUGE metrices.

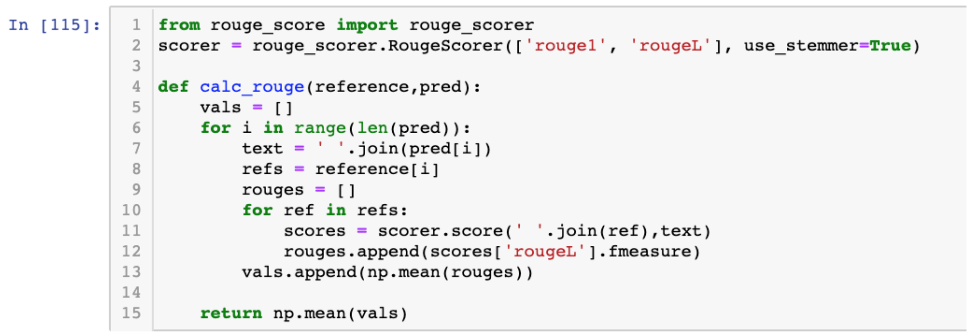
3. For qualitative analysis of the research results, predicted sentences by all models are compared with respect to the content of test image. Figure 19 shown one of test image with actual and predicted sentences by each model.



```
In [44]:
print('BLEU-1: %f' % corpus_bleu(act_1_10, pred_1_10, weights=(1.0, 0, 0, 0)))
print('BLEU-2: %f' % corpus_bleu(act_1_10, pred_1_10, weights=(0.5, 0.5, 0, 0)))
print('BLEU-3: %f' % corpus_bleu(act_1_10, pred_1_10, weights=(0.3, 0.3, 0.3, 0)))
print('BLEU-4: %f' % corpus_bleu(act_1_10, pred_1_10, weights=(0.25, 0.25, 0.25, 0.25)))
```

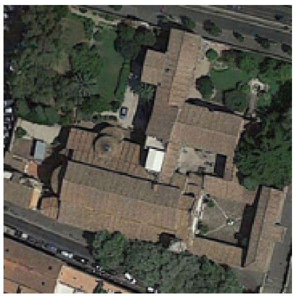
```
BLEU-1: 0.874560
BLEU-2: 0.766410
BLEU-3: 0.691050
BLEU-4: 0.586885
```

Figure 17: Model Architecture.



```
In [115]:
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 18: Model Architecture.



Reference Descriptions

1. a rectangular church with a big and four small round roofs is located on a rounded triangle red square enclosed by three roads with trees planted along
2. a rectangle church with a big and four small round roofs is located on a rounded triangle red square enclosed by three roads with trees planted along
3. the long black church with a umbelliform roof is in a delta region between two rows of trees
4. it is an i shaped church in the middle of two roads
5. three lines of green trees are near a church

CNN (InceptionV3)-LSTM

many buildings and green trees are in a medium residential area

CapsNet-BiLSTM

many green trees are around a building

Figure 19: Model Architecture.

References

Agughalam, D., Pathak, P. and Stynes, P. (2021). Bidirectional LSTM approach to image captioning with scene features, *in* X. Jiang and H. Fujita (eds), *Thirteenth International Conference on Digital Image Processing (ICDIP 2021)*, Vol. 11878, International Society for Optics and Photonics, SPIE, pp. 81 – 88.
URL: *https://doi.org/10.1117/12.2600465*

Deka, J. (2020). *Image captioning: Capsule network vs cnn approach*, Master's thesis, Dublin, National College of Ireland.
URL: *http://norma.ncirl.ie/4298/*