National College of Ireland

Configuration Manual

MSc Research Project
MSc In Data Analytics

Akash Singh
Student ID: x19210736

School of Computing
National College of Ireland

Supervisor: Noel Cosgrave

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Akash Singh
Student ID:	x19210736
Programme:	MSc In Data Analytics
Year:	2021
Module:	MSc Research Project
Supervisor:	Noel Cosgrave
Submission Due Date:	16/12/2021
Project Title:	Configuration Manual
Word Count:	530
Page Count:	5

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:	
Date:	16th December 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.	☐

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):	

Configuration Manual

Akash Singh
x19210736

1 Introduction

This configuration manual explains how to set up the system and environment for the code to work. It also outlines the steps taken to put the research into action. The manual explains feature extraction, data preprocessing, and modeling with the appropriate snippets.

2 System Configuration

2.1 Hardware Configuration

- **Host Machine/Operating System:** Lenovo-IdeaPad Gaming 3 15IMH05/Windows 10
- **RAM:** 8GB, Core-i7 processor
- **Cloud Config:** Google Colab Tesla-P100 GPU with 16GB RAM

2.2 Hardware Configuration

- **Programming Language:** Python 3.7
- **Distribution:** Anaconda Distribution
- **IDE:** Jupyter Notebook
- **Cloud Environment:** Google Collaboratory
- **Browser:** Google Chrome

3 Data Collection and Running the Model

3.1 Downloading the Data

The image data set can be downloaded from the kaggle link `https://www.kaggle.com/shadabhussain/flickr8k` and the machine translated text for the corresponding images can be downloaded from the github repository[1]. Load the data on the drive and then mount it on google colab to use it.

[1]https://github.com/rathiankit03/ImageCaptionHindi/blob/master/Flickr8kHindiDataset/Unclean-5Sentences.txt

3.2 Image Feature Extraction

Image Feature extraction is done by using pre trained CNN such as VGG16, InceptionV3 and ResNet50. Keras is used to load the CNN libraries.

```
from keras.applications.vgg16 import VGG16
from keras.applications.resnet50 import ResNet50
from keras.applications.inception_v3 import InceptionV3
```

Figure 1: Keras CNN libraries

For Image feature extraction remove the last layer of the CNN and and change the target size of the image according to the requirement of the CNN. For Vgg16 and Resnet its is 224*224 and for InceptionV3 it's 229*229.

```
def extract_features(directory):
    # load the model
    #model_resnet50 = ResNet50()
    #model_InceptionV3 = InceptionV3()
    model_vgg16 = VGG16()
    model = Sequential()

    # re-structure the model
    for layer in model_vgg16.layers[:-1]: # this is where I changed your code
        model.add(layer)

    # summarize
    print(model.summary())
    # extract features from each photo
    features = dict()
    for name in listdir(directory):
        # load an image from file
        filename = directory + '/' + name
        image = load_img(filename, target_size=(224, 224))
        # convert the image pixels to a numpy array
        image = img_to_array(image)
        # reshape data for the model
        image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
        # prepare the image for the VGG model
        image = preprocess_input(image)
        # get features
        feature = model.predict(image, verbose=0)
        # get image id
        image_id = name.split('.')[0]
        # store feature
        features[image_id] = feature
        print('>%s' % name)
    return features
```

Figure 2: Image Feature Extraction

After extracting the image, store the image in a pickle file so that it can be reused if needed.

```
# save to file
dump(features, open('VGG16.pkl', 'wb'))
```

Figure 3: Saving Pickle File

3.3 Text loading and Pr-processing

After extraction of image features we load the text in a dictionary and save the file.

```python
# extract descriptions for images
def load_descriptions(doc):
    mapping = dict()
    # process lines
    for line in doc.split('\n'):
        # split line by white space
        tokens = line.split()
        if len(line) < 2:
            continue
        # take the first token as the image id, the rest as the description
        image_id, image_desc = tokens[0], tokens[1:]
        # remove filename from image id
        image_id = image_id.split('.')[0]
        # convert description tokens back to string
        image_desc = ' '.join(image_desc)
        # create the list if needed
        if image_id not in mapping:
            mapping[image_id] = list()
        # store description
        mapping[image_id].append(image_desc)
    return mapping
```

Figure 4: Loading Description

The image descriptions are then added wrapping tags 'startseq' and 'endseq' so that the machine can understand start and end of the image description.

```python
def load_clean_descriptions(filename, dataset):
  # load document
  doc = load_doc(filename)
  descriptions = dict()
  for line in doc.split('\n'):
    # split line by white space
    tokens = line.split()
    # split id from description
    image_id, image_desc = tokens[0], tokens[1:]
    # skip images not in the set
    if image_id in dataset:
      # create list
      if image_id not in descriptions:
        descriptions[image_id] = list()
      # wrap description in tokens
      desc = 'startseq ' + ' '.join(image_desc) + ' endseq'
      # store
      descriptions[image_id].append(desc)
  return descriptions
```

Figure 5: Wrapping Tags

After wrapping the text we need to create tokens of the text so that machine can understand it. The text is tokenized by importing tokenizer from the keras library. Save the tokenzier in a pickle file

```
# fit a tokenizer given caption descriptions
def create_tokenizer(descriptions):
  lines = to_lines(descriptions)
  tokenizer = Tokenizer()
  tokenizer.fit_on_texts(lines)
  return tokenizer
```

Figure 6: Creating Tokenizer

3.4 Modelling

The encoded text and image features are passed through a model. This is where we define our LSTM and Bi-LSTM layer.

```
# define the captioning model
def define_model(vocab_size, max_length):
    # feature extractor model
    inputs1 = Input(shape=(4096,))
    fe1 = Dropout(0.5)(inputs1)
    fe2 = Dense(256, activation='relu')(fe1)
  # sequence model
    inputs2 = Input(shape=(max_length,))
    se1 = Embedding(vocab_size, 256, mask_zero=True)(inputs2)
    se2 = Dropout(0.5)(se1)
    se3 = Bidirectional(LSTM(128, return_sequences = True))(se2)
    se4 = Bidirectional(LSTM(128))(se3)

  # decoder model
    decoder1 = add([fe2, se4])
    decoder2 = Dense(256, activation='relu')(decoder1)
    outputs = Dense(vocab_size, activation='softmax')(decoder2)
  # tie it together [image, seq] [word]
    model = Model(inputs=[inputs1, inputs2], outputs=outputs)
    model.compile(loss='categorical_crossentropy', optimizer='adam')
  # summarize model
    print(model.summary())
    return model
```

Figure 7: Model

The data in this model is loaded progressively for that a generator function is defined which is called in model.fit_generator() when the model is compiled.

```
#Below code is used to progressively load the batch of data
# data generator, intended to be used in a call to model.fit_generator()
def data_generator(descriptions, photos, tokenizer, max_length):
  # loop for ever over images
  while 1:
    for key, desc_list in descriptions.items():
      # retrieve the photo feature
      photo = photos[key][0]
      in_img, in_seq, out_word = create_sequences(tokenizer, max_length, desc_list, photo)
      yield (in_img, in_seq), out_word
```

Figure 8: Generator Function

3.5 Quantitative Evaluation

The Model is evaluated on BLUE score. For calculating the BLUE score the predicted descriptions are compared to the actual captions.

```
# evaluate the skill of the model
def evaluate_model(model, descriptions, photos, tokenizer, max_length):
  actual, predicted = list(), list()
  # step over the whole set
  for key, desc_list in descriptions.items():
    # generate description
    yhat = generate_desc(model, tokenizer, photos[key], max_length)
    # store actual and predicted
    references = [d.split() for d in desc_list]
    actual.append(references)
    predicted.append(yhat.split())
  # calculate BLEU score
  print('BLEU-1: %f' % corpus_bleu(actual, predicted, weights=(1.0, 0, 0, 0)))
  print('BLEU-2: %f' % corpus_bleu(actual, predicted, weights=(0.5, 0.5, 0, 0)))
  print('BLEU-3: %f' % corpus_bleu(actual, predicted, weights=(0.3, 0.3, 0.3, 0)))
  print('BLEU-4: %f' % corpus_bleu(actual, predicted, weights=(0.25, 0.25, 0.25, 0.25)))
```

Figure 9: BLUE score evaluation

3.6 Qualitative Evaluation

The generated image captions are evaluated manually to see if they make sense and are grammatically correct.

एक लड़का एक पूल में कूद रहा है।

Translation: A boy is jumping in a pool in a pool.

Figure 10: Image with generated Text

4 Image Evaluation

After the model run is completed you can download the h5 file with the minimum loss and use it together with the pickle file of tokenizer to generate text for test images.