Configuration Manual

MSc Research Project
MSc. Data Analytics

Sai SriMaha Vishnu Valluri
Student ID: X19208758

School of Computing
National College of Ireland

Supervisor: Dr. Catherine Mulwa

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Sai SriMaha Vishnu Valluri		
Student ID:	X19208758		
Programme:	MSc. Data Analytics	**Year:**	2020/21
Module:	Research Project		
Lecturer:	Dr. Catherine Mulwa		
Submission Due Date:	16th August 2021		
Project Title:	Deep Learning and Natural Language Processing Approach for Real Estate Property Description Generation		
Word Count:	1578 **Page Count:** 15		

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: Sai SriMaha Vishnu Valluri

Date: 23rd August 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

Sai SriMaha Vishnu Valluri
Student ID: X19208758

1 Introduction

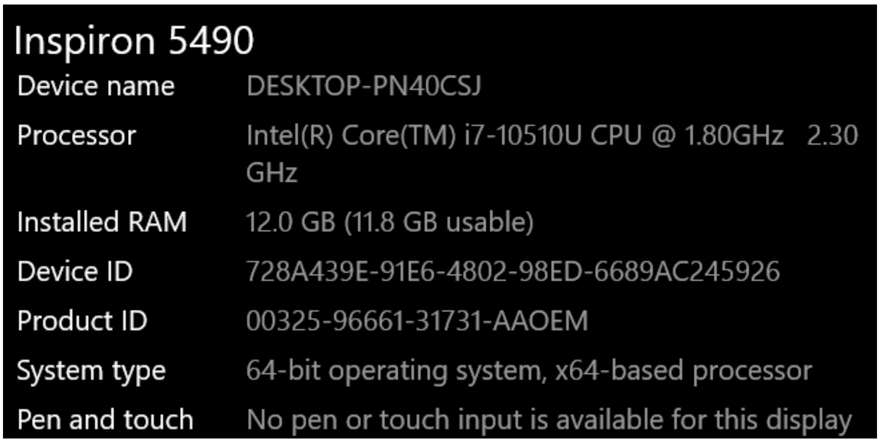
This manual provides information regarding the process involved in carrying out the implementation of the research. The manual provides information regarding system configuration, environment setup, important code snippets and necessary information required to replicate this research.

2 Hardware and IDE Specification

2.1 Hardware Specification

The hardware configuration involved in this project as mentioned below. A screenshot of the system hardware specification is also provided.

- Host device : Dell Inspiron 5490
- Processor: 2.30 GHz Intel Core i7-10510U
- Memory: 12gb
- Storage: 512 GB SSD
- Graphics: Nvidia MX230



2.2 Software Specification

Majority of the work done in this research has been done on Google Colab[1]. Google Colab is an implementation of Jupyter Notebooks[2]. It provides GPU access which is necessary for deep learning tasks. The entirety of this research is executed in Python language. Google Drive has also been used for storing processed data. The GPU provided alternates between an Nvidia P100, k80 or T4.

The following sections would provide information regarding the models that have been executed as a part of the research.

3 Image Classification

The first part of the research involved creating an image classification model. The process of achieving this has been explained in the following sections.

3.1 Import Packages

The following packages have to be imported before running the image classification model.

```
import tensorflow as tf
import matplotlib.image as img
%matplotlib inline
import numpy as np
from collections import defaultdict
import collections
from shutil import copy
from shutil import copytree, rmtree
import tensorflow.keras.backend as K
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
import matplotlib.pyplot as plt
import numpy as np
import os
import random
import tensorflow as tf
import tensorflow.keras.backend as K
from tensorflow.keras import regularizers
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten
from tensorflow.keras.layers import Convolution2D, MaxPooling2D, ZeroPadding2D, GlobalAveragePooling2D, AveragePooling2D
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import ModelCheckpoint, CSVLogger
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.regularizers import l2
from tensorflow import keras
from tensorflow.keras import models
import cv2
import pickle
```

3.2 Data Pre-processing

The data used for image classification is a Housing Image Dataset[3] produced by Poursaeed et. al., and is made available publicly. The dataset is to be downloaded first and uploaded to google drive. This is followed by unzipping the file. After the file is unzipped, the data has been split into training and testing sets in the proportions of 80% for training and 20% for testing. The data is stored in folders according to class names. The following code snippet shows how this splitting this dataset into training and testing datasets was done.

[1] Colab.research.google.com
[2] https://jupyter.org/
[3] https://omidpoursaeed.github.io/publication/vision-based-real-estate-price-estimation/

```
import splitfolders

splitfolders.ratio("/content/Train/kitchen", output="/content/output", seed=1337, ratio=(0.8, 0.2)) # default values
```

The dataset contains around 118,000 images split into 7 classes. This dataset was randomly sampled to reduce the quantity of data to help with computational restrictions. The following code snippet shows how this has been done.

```
#random sampling training data from 118,000 images to 7000 images
import random
import glob, shutil

#loop through each class from the list of classes
for room in room_list:
  mod_dir = os.path.join('/content/drive/MyDrive/output_ic2/output_ic1/train/', room) #assign filepath of current class in accordance to the for loop
  os.chdir(mod_dir) #change directory to training data file
  sample_path = os.path.join('/content/drive/MyDrive/output_ic2/output_ic1/train_main/', room) #set path to where sampled data must be stored
  #sample path directories must be created first in the sample_path folder
  #sample 1000 images per class
  for c in random.sample(glob.glob('*.jpg'), 1000):
    shutil.copy(c, sample_path) #copy all files to the sample path
```

Similarly, the testing dataset has also been randomly sampled to contain 300 images per class. This completes the data pre-processing task.

3.3 Feature Extraction

The next step of the process would be to execute feature extraction. Along with this the images will first need to be augmented. This has been done using the Keras ImageDataGenerator[4] function which can augment data in real time as processing is carried out. ImageDataGenerator has to be used with both training and testing data. The data is processed in batches of 140 and 75 for training and testing respectively. These values have been chosen to as they are factors of the number of images in those datasets.

[4] https://keras.io/api/preprocessing/image/

```
#instantiate imagedatagenerator for batch processing and image augmentation
train_datagen = ImageDataGenerator(
    rescale=1. / 255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True)

test_datagen = ImageDataGenerator(rescale=1. / 255)

#get images from Training dataset and process through ImageDataGenerator
train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(image_height, image_width),
    batch_size=batch_size_tr,
    class_mode='categorical')

#get images from Testing Dataset and process through ImageDataGenerator
test_generator = test_datagen.flow_from_directory(
    testing_data_dir,
    target_size=(image_height, image_width),
    batch_size=batch_size_test,
    class_mode='categorical')

Found 7000 images belonging to 7 classes.
Found 2100 images belonging to 7 classes.
```

3.4 Modelling

The first step of modelling is to load the InceptionV3 model with ImageNet weights. The InceptionV3 model is already imported in the beginning when importing packages.

```
model_inception = InceptionV3(weights='imagenet', include_top=False)
model_inception.summary()

conv2d_96 (Conv2D)              (None, None, None, 6 18432       activation_95[0][0]
batch_normalization_96 (BatchNo (None, None, None, 6 192         conv2d_96[0][0]
activation_96 (Activation)      (None, None, None, 6 0           batch_normalization_96[0][0]
max_pooling2d_4 (MaxPooling2D)  (None, None, None, 6 0           activation_96[0][0]
conv2d_97 (Conv2D)              (None, None, None, 8 5120        max_pooling2d_4[0][0]
batch_normalization_97 (BatchNo (None, None, None, 8 240         conv2d_97[0][0]
activation_97 (Activation)      (None, None, None, 8 0           batch_normalization_97[0][0]
conv2d_98 (Conv2D)              (None, None, None, 1 138240      activation_97[0][0]
batch_normalization_98 (BatchNo (None, None, None, 1 576         conv2d_98[0][0]
activation_98 (Activation)      (None, None, None, 1 0           batch_normalization_98[0][0]
```

Following this, GlobalAveragePooling is done and dropout is used to reduce overfitting.

```
x = model_inception.output
x = GlobalAveragePooling2D()(x) #takes average of each feature map
x = Dense(128,activation='relu')(x)
x = Dropout(0.2)(x) #use dropout to tackle overfitting
```

Activation functions are set up. Checkpointers are used to save the model at multiple checkpoints for future use. Early stopping has been used to stop training the model at the right time[5].

```
predictions = Dense(class_n,kernel_regularizer=regularizers.l2(0.005), activation='softmax')(x)
model = Model(inputs=model_inception.input, outputs=predictions)
model.compile(optimizer=SGD(lr=0.0001, momentum=0.9), loss='categorical_crossentropy', metrics=['accuracy'])
checkpointer = ModelCheckpoint(filepath='checkpoint_15epoch_3class.hdf5', verbose=1, save_best_only=True)
csv_logger = CSVLogger('imgcls_history_15epoch_3class.log')
```

Following this, the model is run for 10, 15 and 20 epochs. Each of the three files are saved and can be loaded to use without having to train the model again.

```
history = model.fit(train_generator,
                    steps_per_epoch = nb_train_samples // batch_size_tr,
                    validation_data=validation_generator,
                    validation_steps=nb_validation_samples // batch_size_val,
                    epochs=10,
                    verbose=1,
                    callbacks=[csv_logger, checkpointer]) #model training

model.save('classification_10epoch_3class.hdf5')
```

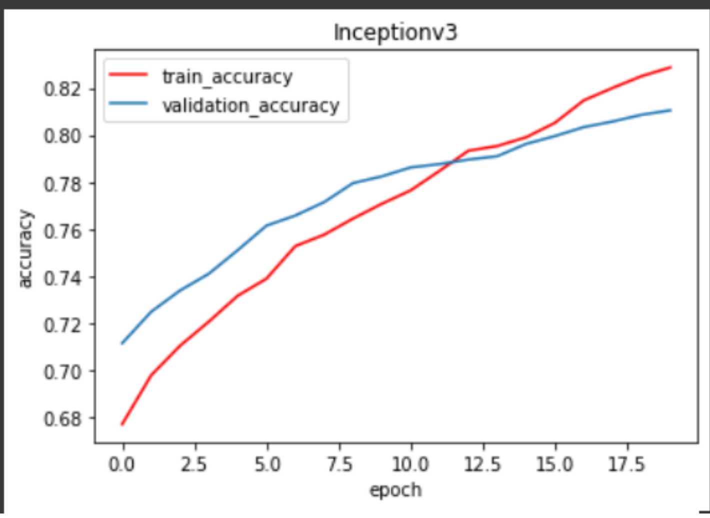
```
Epoch 1/10
50/50 [==============================] - 1679s 33s/step - loss: 2.0529 - accuracy: 0.1629 - val_loss: 1.9723 - val_accuracy: 0.2014

Epoch 00001: val_loss improved from inf to 1.97234, saving model to classificatiion_best_10epoch_3class.hdf5
Epoch 2/10
50/50 [==============================] - 177s 4s/step - loss: 1.9325 - accuracy: 0.2519 - val_loss: 1.8477 - val_accuracy: 0.3214
```

3.5 Model Evaluation

Following modelling, the model needs to be evaluated to analyse performance. Evaluation is done using loss and accuracy metrics. These values have been plotted in a graph for 10, 15 and 20 epochs.

```
[ ]  plot_accuracy_10(history_20,'Inceptionv3')
     plot_loss_10(history_20,'Inceptionv3')
```



[5] https://machinelearningmastery.com/how-to-stop-training-deep-neural-networks-at-the-right-time-using-early-stopping/

The model has been tested qualitatively as well by testing on a few images to see how the model would perform. The following function has been written to get the classification results.

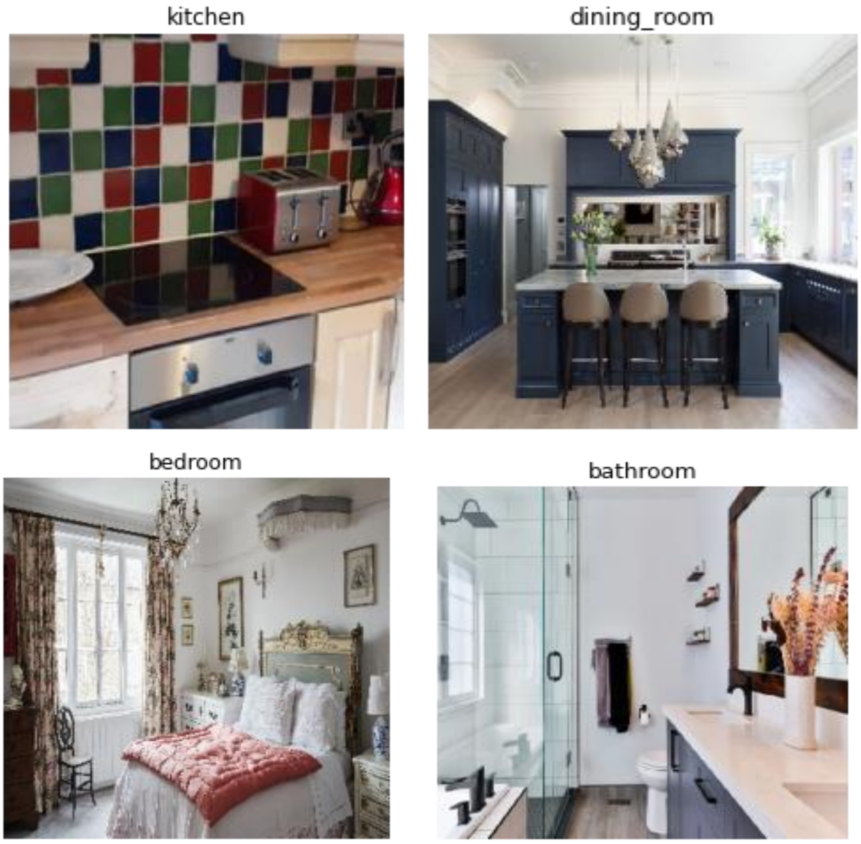
```
def predict_img(model, image1, show = True):

  for img in image1:
    img = image.load_img(img, target_size=(299, 299))
    img = image.img_to_array(img)
    img = np.expand_dims(img, axis=0)
    img /= 255.

    pred = model.predict(img)
    index = np.argmax(pred)
    room_list.sort()
    pred_value = room_list[index]

    if show:
      plt.imshow(img[0])
      plt.axis('off')
      plt.title(pred_value)
      plt.show()
```

The results are qualitatively analysed for 10, 15 and 20 epochs to understand the capabilities of the model. The following image is a sample output of classification results.



4 Image Captioning

In this section, parts of the image captioning implementation will be explained. Two experiments have been performed in this section. The following sections explain the code with the help of snippets.

4.1 Import Packages

The following packages have to be imported first before running the models.

```
import os

import pandas as pd
import pickle
import numpy as np
import random
from collections import defaultdict
from collections import OrderedDict
from nltk.translate.bleu_score import corpus_bleu
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, Bidirectional, RepeatVector, Concatenate
from keras.models import Sequential, Model
from keras.utils import np_utils
from keras.utils.vis_utils import plot_model
from keras.callbacks import ModelCheckpoint
from keras.callbacks import EarlyStopping
from keras.callbacks import Callback
from keras.preprocessing import image, sequence

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

4.2 Data Pre-processing

Flickr8k Image captioning dataset[6] has been used for training and testing the models. The data is first unzipped. The image files are stored in a common folder. The dataset is split into training, testing and validation sets. The file names of the files which are meant for training and testing are stored in text files. The first step is to parse these text files containing filenames to the images and storing them in variables. The following code snippet shows this.

[6] https://www.kaggle.com/shadabhussain/flickr8k

```
[2] #helper function to load images
    def ImageLoader(filename):
      with open(filename, 'r') as image_list_f:
        return [line.strip() for line in image_list_f]

    text_path = '/content/drive/MyDrive/Flickr_folder/Flickr_Data/Flickr_TextData'
    image_path = '/content/drive/MyDrive/Flickr_folder/Flickr_Data/Images'
```

```
[3] #load file names from the text files and parse images accordingly
    train = ImageLoader(os.path.join(text_path,'Flickr_8k.trainImages.txt'))
    test = ImageLoader(os.path.join(text_path,'Flickr_8k.testImages.txt'))
```

4.3 Feature Extraction

Feature extraction is done in both experiments. The first experiment requires features only to be extracted by the InceptionV3 model. In the second experiment, features are to be extracted from both InceptionV3 model and VGG16 Places365 CNN model as well. Both the models need to be loaded first before extracting features. Model loading is shown below.

```
[4] inceptionModel = InceptionV3(weights='imagenet')  #load inceptionv3 model

    Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception
    96116736/96112376 [==============================] - 1s 0us/step
    96124928/96112376 [==============================] - 1s 0us/step

[5] new_input = inceptionModel.input
    new_output = inceptionModel.layers[-2].output
    img_encoder = Model(new_input, new_output) #used for image encoding
```

```
    #loading VGG16 Places365 Model
    vgg_model = VGG16_Places365(weights = 'places', include_top = True)
    vgg_model.summary()

[23] inp = vgg_model.input
     out = vgg_model.layers[-2].output
     vgg_encoder = Model(inp, out) #used for encoding training and testing data with VGG16 Places365 CNN
```

Image feature extraction can now be done.

```
[ ] trainEnc = img_encoder.predict_generator(img_gen(train), steps = len(train), verbose =1)

    /usr/local/lib/python3.7/dist-packages/keras/engine/training.py:1976: UserWarning: `Model.predict_ge
      warnings.warn('`Model.predict_generator` is deprecated and '
    6000/6000 [==============================] - 1923s 316ms/step

[ ] testEnc = img_encoder.predict_generator(img_gen(test), steps = len(test), verbose=1)

    /usr/local/lib/python3.7/dist-packages/keras/engine/training.py:1976: UserWarning: `Model.predict_ge
      warnings.warn('`Model.predict_generator` is deprecated and '
    1000/1000 [==============================] - 303s 303ms/step
```

Following feature extraction, text data has to be processed. This involves splitting words by whitespace, removing numbers, punctuation and storing them as a list. Starting and ending tags must be added to sentences to indicate the start and end. The following function was defined to process text data.

```
#text processing
def text_processing(filename):
  imageDesc = defaultdict(list)
  with open(os.path.join(text_path, filename), 'r') as description_list:
    for line in description_list:
      img_val = line.split()   #split lines into words according to white spaces
      img_key = img_val[0]     #first word of the text is the image file name followed by caption number(#0-#5)
      img_key = img_key[0:-2]  #removing caption numbers from image file names
      image_value = img_val[1:] #storing all words after the 0th word
      new_val = []
      for word in image_value:
        new_val.append(word.lower()) #converting to lower case
      new_val.insert(0, "<startseq>") #adding a start tag
      new_val.append("<endseq>")  #adding an end tag
      if img_key not in imageDesc.keys():
        descriptions = []
      else:
        descriptions = imageDesc[img_key]
      descriptions.append(new_val)
      imageDesc[img_key] = descriptions #store all captions as sequence of words in a list

  return imageDesc
```

4.4 Modelling

In this section, both CNN models, along with LSTM text generation model are executed. The process involves merging the features which are extracted using InceptionV3 model and VGG16 model. Images need to be processed in batches for modelling. The code for processing images in batches is provided in the code files submitted. Model building can be done after creating the image generator. The following code shows how the model had been set up.

```
def create_vgg_lstm():

  MAX_LEN = max_len
  embed_dim = 300
  IMAGE_ENC_DIM = 300
  vocab_size = len(vector_id)

  #image feature inputs
  img_input = Input(shape=(2048,))#inceptionv3 dimensions are 2048
  img_enc = Dense(embed_dim, activation="relu") (img_input)
  images = RepeatVector(MAX_LEN)(img_enc)
  plcimg_input = Input(shape=(4096,)) #vgg16 dimensions are 4096
  plcimg_enc = Dense(embed_dim, activation = "relu")(plcimg_input)
  plcimages = RepeatVector(MAX_LEN)(plcimg_enc)
  merge_img = Concatenate()([images, plcimages]) #concatenate features extracted from vgg16 places365 CNN and inception V3

  #text input
  text_input = Input(shape=(MAX_LEN,))
  embedding = Embedding(vocab_size, embed_dim, input_length = MAX_LEN)(text_input) #convert vectors into dense vectors


  x = Concatenate()([merge_img, embedding])
  y = LSTM(256, return_sequences=False)(x) #setting up LSTM network for text generation
  pred = Dense(vocab_size, activation='softmax')(y)
  model = Model(inputs = [img_input, plcimg_input, text_input], outputs=pred)
  model.compile(loss = 'categorical_crossentropy', optimizer = "RMSProp", metrics=['accuracy'])

  model.summary()

  return model
```

Image features are merged in this and processed as a single set of features. This is then passed to the LSTM network along with text inputs processed from the text data. The following is a summary of the model that had just been created,

```
[30] mod_vg = create_vgg_lstm()

Model: "model_2"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_4 (InputLayer)            [(None, 2048)]       0
__________________________________________________________________________________________________
input_5 (InputLayer)            [(None, 4096)]       0
__________________________________________________________________________________________________
dense (Dense)                   (None, 300)          614700      input_4[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 300)          1229100     input_5[0][0]
__________________________________________________________________________________________________
repeat_vector (RepeatVector)    (None, 37, 300)      0           dense[0][0]
__________________________________________________________________________________________________
repeat_vector_1 (RepeatVector)  (None, 37, 300)      0           dense_1[0][0]
__________________________________________________________________________________________________
input_6 (InputLayer)            [(None, 37)]         0
__________________________________________________________________________________________________
concatenate_2 (Concatenate)     (None, 37, 600)      0           repeat_vector[0][0]
                                                                 repeat_vector_1[0][0]
__________________________________________________________________________________________________
embedding (Embedding)           (None, 37, 300)      2658600     input_6[0][0]
__________________________________________________________________________________________________
concatenate_3 (Concatenate)     (None, 37, 900)      0           concatenate_2[0][0]
                                                                 embedding[0][0]
__________________________________________________________________________________________________
lstm (LSTM)                     (None, 256)          1184768     concatenate_3[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 8862)         2277534     lstm[0][0]
==================================================================================================
Total params: 7,964,702
Trainable params: 7,964,702
Non-trainable params: 0
```

After this, a path has to be specified for saving the model. Early stopping and checkpoints have also been established to stop the model from overfitting and saving the best model after execution. Models had been run after this for 10 epochs and 15 epochs and results have been collected. Following image is of model training.

```
history = mod_vg.fit_generator(vgg_img_generator, steps_per_epoch=steps, verbose=True, epochs=15, callbacks=[checkpoint_save,early_stop])

/usr/local/lib/python3.7/dist-packages/keras/engine/training.py:1915: UserWarning: `Model.fit_generator` is deprecated and will be removed in a
  warnings.warn('`Model.fit_generator` is deprecated and '
Epoch 1/15
740/740 [==============================] - 421s 554ms/step - loss: 5.5912 - accuracy: 0.1324

Epoch 00001: accuracy improved from -inf to 0.13155, saving model to experiment_vgg_15_ep.hdf5
Epoch 2/15
740/740 [==============================] - 409s 553ms/step - loss: 5.2466 - accuracy: 0.1370
```

After running the models, greedy search algorithm has been used to form the descriptions. Words are arranged in a sequence with the based on predictions made by the model created in the previous steps. Greedy search implementation is given below.

```
#greedy search algorithm for obtaining the next word according to probabilites
def greedy_search(image):
  vgg_img = vgg_encoded_test[image]
  inc_img = testdict[image]
  sequence = ["<startseq>"] #set up a list to append all the words chhosen by the algorithm
  for i in range(max_len):
    txt_sequence = [vector_id[word] for word in sequence if word in vector_id]
    while len(txt_sequence) < max_len: #append as long as maximum sentence length is not exceeded
      txt_sequence.append(0)
    img = np.reshape(inc_img, (-1, 2048))
    txt_sequence = np.reshape(txt_sequence, (-1, max_len))
    plc_img = np.reshape(vgg_img, (-1,4096))
    next_word = mod_vg.predict([np.array(img), np.array(plc_img), np.array(txt_sequence)]) #predict the next word based on processing image features
    next_word = np.argmax(next_word)
    best_word = vector_word[next_word+1]
    sequence.append(best_word)
    if best_word == "<endseq>":
      return sequence
  return sequence
```

5 Evaluation

Evaluation of the model is based on analysing BLEU scores and qualitative analysis. Initially all the predicted captions for the testing set are collected along with the actual captions of the images. This has been done in the following way:

1. Create lists for actual and predicted captions
2. Create a for loop to loop through the entire testing set
3. Apply the greedy_search algorithm from the previous step on the entire testing set to obtain predicted captions and store in predicted captions list
4. Store all actual captions in the actual captions list
5. Remove starting and ending tags from the predicted and actual captions.

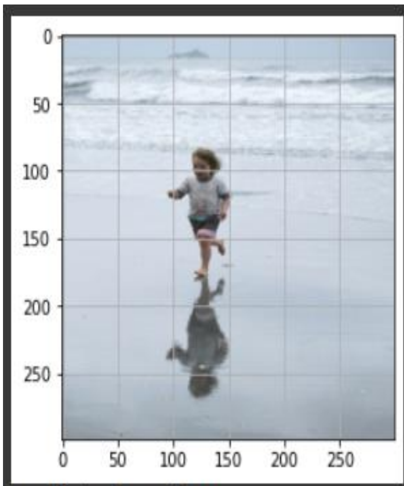
After processing the predicted and actual captions, BLEU scores can be calculated. BLEU Scores are calculated using the corpus_bleu function which is a part of the NLTK package[7]. The following image shows the BLEU scores of running the model for 10 epochs.

```
print('bleu-1: %f' % corpus_bleu(act_caps, pred_caps, weights=(1.0, 0, 0, 0)))
print('bleu-2: %f' % corpus_bleu(act_caps, pred_caps, weights=(0,5, 0.5, 0, 0)))
print('bleu-3: %f' % corpus_bleu(act_caps, pred_caps, weights=(0.3, 0.3, 0.3, 0)))
print('bleu-4: %f' % corpus_bleu(act_caps, pred_caps, weights=(0.25, 0.25, 0.25, 0.25)))
```

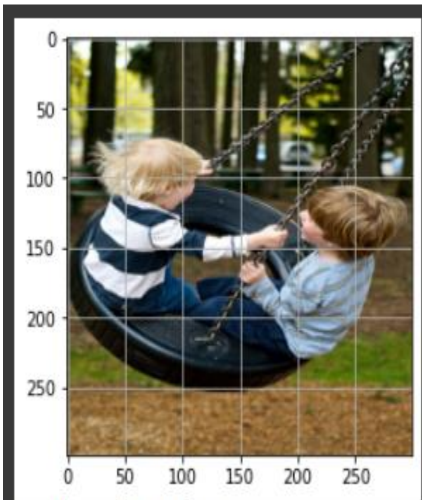
```
bleu-1: 0.545666
bleu-2: 0.000199
bleu-3: 0.255641
bleu-4: 0.133034
```

After calculating BLEU scores, the outputs have been analysed qualitatively. Samples of outputs are displayed below.

[7] https://www.nltk.org/_modules/nltk/translate/bleu_score.html




```
predicted caption:
a child in a red shirt is running on the beach
--------------
a little boy runs away from the approaching waves of the ocean
a little girl runs across the wet beach
a little girl runs on the wet sand near the ocean
a young girl runs across a wet beach with the ocean in the background
child running on the beach
```




```
predicted caption:
a boy in a blue shirt is playing on a swing
--------------
two children are playing on a swing made out of a tire
two children swing around in a tire swing at a park
two children swinging on a tire swing
two small children happily playing on a tire swing
two small children wearing blue jeans enjoy swinging in a tire swing
```