

# Task1 - Import Libraries and Data set

## 1.1 Import Required Libraries

```
#got images from https://www.kaggle.com/datasets/adityajn105/flickr8k?resource=download&se
#got the captions from https://www.kaggle.com/datasets/adityajn105/flickr8k?resource=downl
!pip install keras
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/r
Requirement already satisfied: keras in /usr/local/lib/python3.7/dist-packages (2.9.0)
```

```
from keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.utils import load_img
from tensorflow.keras.utils import img_to_array
from keras.applications.inception_v3 import preprocess_input
from keras.models import Model
import os
import nltk
from nltk.translate.bleu_score import sentence_bleu
```

```
import warnings
warnings.filterwarnings('ignore')
```

```
nltk.download('stopwords')
nltk.download('punkt')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
True
```

## 1.2 Import the data sets

The pickle file containing image captions and the Flickr Dataset having images are imported to Google Drive

## 1.3 Check Availaility of GPU

Google Colab Pro which provides GPU/TPU & Higher RAM caapcity is used to run deep learning models.

```
from google.colab import drive
```

```
drive.mount('/content/gdrive/')
```

```
Mounted at /content/gdrive/
```

## Task2 - Data Processing

### 2.1 Read the Pickle File

```
file1 = open('gdrive/My Drive/DLAssgn2/captions.txt', 'r')
new_dict = file1.readlines()
new_dict = [line[:-1] for line in new_dict]
new_dict = new_dict[1:]
new_dict[0:3]

['1000268201_693b08cb0e.jpg,A child in a pink dress is climbing up a set of stairs
in an entry way .',
'1000268201_693b08cb0e.jpg,A girl going into a wooden building .',
'1000268201_693b08cb0e.jpg,A little girl climbing into a wooden playhouse .']
```

### 2.2 Processing the Captions

#### 2.2.1 Caption Processor

```
# load the caption file & read it
def load_caption_file(path):

    # dictionary to store captions
    captions_dict = {}

    # iterate through the file
    for caption in path:

        # caption has format-> 2872963574_52ab5182cb.jpg#4,Two oppose hockey player collid
        tokens = caption.split(',')
        caption_id, caption_text = tokens[0].split('.')[0], tokens[1]
        #caption_text = ' '.join(caption_text)

        # save it in the captions dictionary
        if caption_id not in captions_dict:
            captions_dict[caption_id] = [caption_text]
        else:
            captions_dict[caption_id].append(caption_text)

    return captions_dict

# call the function
captions_dict = load_caption_file(new_dict)
```

```
len(captions_dict)
```

```
8091
```

```
captions_dict[list(captions_dict.keys())[0]]
```

```
['A child in a pink dress is climbing up a set of stairs in an entry way .',
 'A girl going into a wooden building .',
 'A little girl climbing into a wooden playhouse .',
 'A little girl climbing the stairs to her playhouse .',
 'A little girl in a pink dress going into a wooden cabin .']
```

## 2.2.2 Preprocess the captions

1) Convert the captions into lowercase

2) Tokenize the captions into different tokens

3) Remove all the punctuations from the tokens

4) add "start\_index" and "end\_index" as pointers to tell the model start of the caption and end of the caption

```
# clean the captions
import string
```

```
# dictionary to store the cleaned captions
new_captions_dict = {}
```

```
# prepare translation table for removing punctuation. third argument is the list of punctu
table = str.maketrans('', '', string.punctuation)
```

```
# loop through the dictionary
```

```
for caption_id, caption_texts in captions_dict.items():
```

```
    #iterate through the 5 caption texts for each image
```

```
    for caption_text in caption_texts:
```

```
        # tokenize the caption_text
```

```
        caption_text = caption_text.split()
```

```
        # convert it into lower case
```

```
        caption_text = [token.lower() for token in caption_text]
```

```
        # remove punctuation from each token
```

```
        caption_text = [token.translate(table) for token in caption_text]
```

```
        # remove all the single letter tokens like 'a', 's'
```

```
        caption_text = [token for token in caption_text if len(token)>1]
```

```
        # store the cleaned captions
```

```
        if caption_id not in new_captions_dict:
```

```
            new_captions_dict[caption_id] = ['startseq ' + ' '.join(caption_text) + ' endseq']
```

```
        else:
```

```
            new_captions_dict[caption_id].append('startseq ' + ' '.join(caption_text) + ' ends
```

```
# delete unwanted
del captions_dict
```

```
print('' + list(new_captions_dict.keys())[1] + '' + ' : ' + str(new_captions_dict[list(n
    "1001773457_577c3a7d70" : ['startseq black dog and spotted dog are fighting endseq',
```



```
len(new_captions_dict)
```

```
8091
```

Make a list of only those images with captions

```
import os
```

```
caption_images_list = []
```

```
image_dataset_path = 'gdrive/My Drive/DLAssgn2/FlickrImages' # CHANGE THIS TO THE PATH WHE
```

```
image_index = list(new_captions_dict.keys())
```

```
caption_images_list = [ image.split('.')[0] for image in os.listdir(image_dataset_path) if
```

```
caption_images_list[0]
```

```
'416788726_5b4eb1466e'
```

```
len(caption_images_list)
```

```
8091
```

```
#need to change this to appropriate number
```

```
8091-1010
```

```
7081
```

## 2.3 Make training, validation and test data

Take 7081 images for training, 1000 for validation and rest 10 for testing

```
train_validate_images = caption_images_list[0:8081]
```

```
test_images = caption_images_list[8081:8092]
```

```
test_images
```

```
['109823397_e35154645f',
 '1141718391_24164bf1b1',
 '1096165011_cc5eb16aa6',
 '1104133405_c04a00707f',
 '1107471216_4336c9b328',
 '111497985_38e9f88856',
 '1144288288_e5c9558b6a',
 '112178718_87270d9b4d',
 '1110208841_5bb6806afe',
 '1130401779_8c30182e3e']
```

## 2.4 Plot Two samples and their captions


```
from IPython.display import Image
type(Image(image_dataset_path+'/3165936115_cb4017d94e.jpg'))
```

```
IPython.core.display.Image
```

```
new_captions_dict.get('3165936115_cb4017d94e')
```


```
['startseq two men juggling red endseq',
 'startseq two men look like they are playing with boxes in mall endseq',
 'startseq two men stand juggling colored boxes while other men stand on balcony
endseq',
 'startseq two young asian men juggle colored boxes in mall endseq',
 'startseq two young men are juggling multicolored blocks with people watching
endseq']
```

```
Image(image_dataset_path+'/3083016677_5782bc337c.jpg')
```



```
new_captions_dict.get('3083016677_5782bc337c')
```

```
['startseq motorcyclists near the beach endseq',
 'startseq there are two motorcycles with man and woman on it endseq',
 'startseq two guys with helmets are on motorcycles endseq',
 'startseq two motorcycles and four riders are on the road endseq',
 'startseq two motorcycles with two riders each endseq']
```




## Task3 - Model Building



### 3.1 Image Feature Extractor

Use Pretrained Inception model trained on ImageNet dataset for image feature extraction.



```
# extract features from each photo in the directory
def extract_features(directory, image_keys):
    # load the model
    model = InceptionV3(input_shape = (299, 299, 3))

    # re-structure the model
    model = Model(inputs=model.inputs, outputs=model.layers[-2].output)

    # summarize
    print(model.summary())

    # extract features from each photo
    features = dict()

    for name in image_keys:

        # load an image from file
        filename = directory + '/' + name + '.jpg'

        # load the image and convert it into target size of 299*299
        image = load_img(filename, target_size=(299, 299))

        # 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 Inception 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

```

## Extracting Image Features

```

# extracting image features for train_validate_images
#train_validate_features = extract_features(image_dataset_path, train_validate_images)

#from pickle import dump
#dump(train_validate_features, open('./train_validate_features.pkl', 'wb'))

from pickle import load
with open('gdrive/My Drive/DLAssgn2/train_validate_features_IV3_LSTM.pkl', 'rb') as handle
    train_validate_features = load(handle)

print("{} : {}".format(list(train_validate_features.keys())[0], train_validate_features[list(
len(train_validate_features)

416788726_5b4eb1466e : [[0.87184024 0.09597223 0.58854485 ... 0.38987753 0.96008      (
8081

```



## 3.2 Preparing the input data

Each caption will be split into words. The model will be provided one word and the image and generate the next word. Then the first two words of the description will be provided to the model as input with the image to generate the next word. This is how the model will be trained. So we will have two features,  $x_1$  (image) ,  $x_2$  (text\_sequence) and one target variable,  $y$  (generated\_word).

Image, text sequence, generated\_word

photo startseq, little

photo startseq, little, girl

photo startseq, little, girl, running

photo startseq, little, girl, running, in

photo startseq, little, girl, running, in, field

photo startseq, little, girl, running, in, field, endseq

### 3.2.1 Load Required Libraies

```
# load libraries
import numpy as np
from keras.models import Model, load_model
from keras.layers import Input, Dense, Dropout, GRU, Embedding
from keras.preprocessing.text import Tokenizer
from keras_preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical, plot_model
from tensorflow.keras import regularizers
from tensorflow.keras.layers import add
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau

max_len = max(len(caption.split()) for image in new_captions_dict for caption in new_capti
max_len

31

# make a dictionary of image with caption for train_validate_images
train_validate_image_caption = {}

for image, captions in new_captions_dict.items():

    # check whether the image is available in both train_validate_images list and train_va
    if image in train_validate_images and image in list(train_validate_features.keys()):

        for caption in captions:
            if image not in train_validate_image_caption:
                train_validate_image_caption[image] = [caption]
            else:
                train_validate_image_caption[image].append(caption)

len(train_validate_image_caption)

8081
```

### 3.2.2 make sure the correct caption is mapped with the correct image

```
list(train_validate_image_caption.values())[1]

['startseq black dog and spotted dog are fighting endseq',
 'startseq black dog and tricolored dog playing with each other on the road endseq',
 'startseq black dog and white dog with brown spots are staring at each other in the
```



```
street endseq',
'startseq two dogs of different breeds looking at each other on the road endseq',
'startseq two dogs on pavement moving toward each other endseq']
```

```
Image(image_dataset_path+'/'+list(train_validate_image_caption.keys())[1]+' .jpg')
```



```
# initialise tokenizer
tokenizer = Tokenizer()

# create word count dictionary on the captions list
tokenizer.fit_on_texts(list(np.concatenate(list(train_validate_image_caption.values()))))

# how many words are there in the vocabulary? store the total length in vocab_len and add
vocab_len = len(tokenizer.word_index) + 1

# store the length of the maximum sentence
max_len = max(len(caption.split()) for image in train_validate_image_caption for caption i

def prepare_data(image_keys):

    # x1 will store the image feature, x2 will store one sequence and y will store the nex
    x1, x2, y = [], [], []

    # iterate through all the images
    for image in image_keys:

        # store the caption of that image
        captions = train_validate_image_caption[image]

        for caption in captions:

            # split the image into tokens
```

```

caption = caption.split()

# generate integer sequences of the
seq = tokenizer.texts_to_sequences([caption])[0]

length = len(seq)

for i in range(1, length):

    x2_seq, y_seq = seq[:i] , seq[i]

    # pad the sequences
    x2_seq = pad_sequences([x2_seq], maxlen = max_len)[0]

    # encode the output sequence
    y_seq = to_categorical([y_seq], num_classes = vocab_len)[0]

    x1.append( train_validate_features[image][0] )

    x2.append(x2_seq)

    y.append(y_seq)

return np.array(x1), np.array(x2), np.array(y)

train_x1, train_x2, train_y = prepare_data( train_validate_images[0:7081] )
validate_x1, validate_x2, validate_y = prepare_data( train_validate_images[7081:8081] )

shuffler = np.random.RandomState(seed=40).permutation(len(train_x1))
train_x1 = train_x1[shuffler]
train_x2 = train_x2[shuffler]
train_y = train_y[shuffler]

len(train_x1)

347037

len(validate_x1)

49954

```

## Task4 - Create 3-layered GRU layer model and other relevant layers for image caption generation and Compile the Model

### 4.1 Add Regularization, Dropout and Choose Appropriate Activation Function & Loss Function and Print Model Summary

```
# feature extractor model
input_1 = Input(shape=(2048,))
droplayer = Dropout(0.5)(input_1)
denselayer = Dense(256, activation='relu')(droplayer)

# sequence model
input_2 = Input(shape=(max_len,))
embedding = Embedding(vocab_len, 256, mask_zero=True)(input_2)
droplayer_ = Dropout(0.5)(embedding)
gru = GRU(256)(droplayer_)

# decoder model
decoder1 = add([denselayer, gru])
decoder2 = Dense(256, activation='relu', kernel_regularizer=regularizers.l2(0.005), activity_regularizer=regularizers.l2(0.005))(decoder1)
outputs = Dense(vocab_len, activation='softmax')(decoder2)

# tie it together [image, seq] [word]
model = Model(inputs=[input_1, input_2], outputs=outputs)
model.compile(loss='categorical_crossentropy', optimizer='adam')

# summarize model
print(model.summary())
```

Model: "model"

Layer (type)	Output Shape	Param #	Connected to
input_2 (InputLayer)	[(None, 31)]	0	[]
input_1 (InputLayer)	[(None, 2048)]	0	[]
embedding (Embedding)	(None, 31, 256)	2203136	['input_2[0][0]']
dropout (Dropout)	(None, 2048)	0	['input_1[0][0]']
dropout_1 (Dropout)	(None, 31, 256)	0	['embedding[0][0]']
dense (Dense)	(None, 256)	524544	['dropout[0][0]']
gru (GRU)	(None, 256)	394752	['dropout_1[0][0]']
add (Add)	(None, 256)	0	['dense[0][0]', 'gru[0][0]']
dense_1 (Dense)	(None, 256)	65792	['add[0][0]']
dense_2 (Dense)	(None, 8606)	2211742	['dense_1[0][0]']

```
=====
Total params: 5,399,966
Trainable params: 5,399,966
Non-trainable params: 0
```

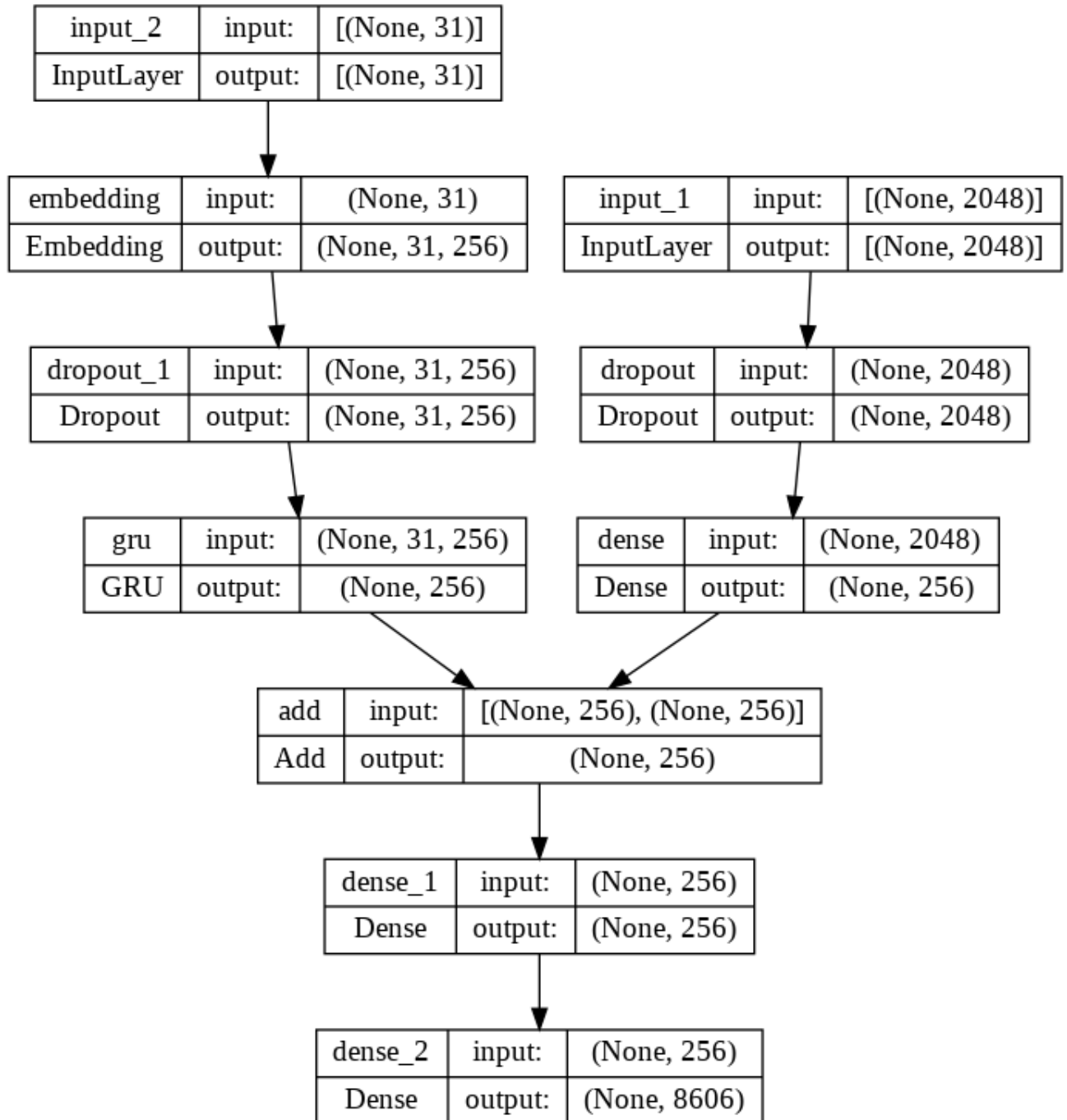
None



## 4.2 Plot the model architecture

```
# This is formatted as code
```

```
plot_model(model, to_file='model.png', show_shapes=True)
```



## 4.3 Make sure feature data and target data share the same first dimension

```
print("shape of train_x1 ", train_x1.shape)
print("shape of train_x2 ", train_x2.shape)
```

```
print("shape of train_y ", train_y.shape)
print()
print("shape of validate_x1 ", validate_x1.shape)
print("shape of validate_x2 ", validate_x2.shape)
print("shape of validate_y ", validate_y.shape)
```

```
shape of train_x1 (347037, 2048)
shape of train_x2 (347037, 31)
shape of train_y (347037, 8606)

shape of validate_x1 (49954, 2048)
shape of validate_x2 (49954, 31)
shape of validate_y (49954, 8606)
```

#### 4.4 Initialize a model checkpoint object to capture the model instance giving the least validation loss

```
model_checkpoint = ModelCheckpoint('model_IV3_GRU.h5',
                                   save_best_only = True,
                                   monitor = 'val_loss',
                                   verbose = 1)
```

### Task5 - Train the model using training data and validation data

#### 5.1 Train the model for an appropriate number of epochs. Print the train and validation loss for each epoch. Use the appropriate batch size.

```
# fit model
history = model.fit([train_x1, train_x2],
                    train_y,
                    verbose = 1,
                    batch_size = 512,
                    epochs = 15,
                    validation_data=([validate_x1, validate_x2], validate_y),
                    callbacks = [model_checkpoint])

Epoch 1: val_loss improved from inf to 4.36938, saving model to model_IV3_GRU.h5
678/678 [=====] - 188s 270ms/step - loss: 5.1049 - val_loss: 4.36938
Epoch 2/15
678/678 [=====] - ETA: 0s - loss: 4.0944
Epoch 2: val_loss improved from 4.36938 to 4.06787, saving model to model_IV3_GRU.h5
678/678 [=====] - 181s 266ms/step - loss: 4.0944 - val_loss: 4.06787
Epoch 3/15
678/678 [=====] - ETA: 0s - loss: 3.8074
Epoch 3: val_loss improved from 4.06787 to 3.92450, saving model to model_IV3_GRU.h5
678/678 [=====] - 182s 269ms/step - loss: 3.8074 - val_loss: 3.92450
Epoch 4/15
678/678 [=====] - ETA: 0s - loss: 3.6301
Epoch 4: val_loss improved from 3.92450 to 3.82879, saving model to model_IV3_GRU.h5
678/678 [=====] - 183s 270ms/step - loss: 3.6301 - val_loss: 3.82879
```

```

Epoch 5/15
678/678 [=====] - ETA: 0s - loss: 3.5001
Epoch 5: val_loss improved from 3.82879 to 3.77062, saving model to model_IV3_GRU.
678/678 [=====] - 185s 273ms/step - loss: 3.5001 - val_loss: 3.77062
Epoch 6/15
678/678 [=====] - ETA: 0s - loss: 3.3972
Epoch 6: val_loss improved from 3.77062 to 3.73127, saving model to model_IV3_GRU.
678/678 [=====] - 186s 274ms/step - loss: 3.3972 - val_loss: 3.73127
Epoch 7/15
678/678 [=====] - ETA: 0s - loss: 3.3141
Epoch 7: val_loss improved from 3.73127 to 3.71608, saving model to model_IV3_GRU.
678/678 [=====] - 186s 274ms/step - loss: 3.3141 - val_loss: 3.71608
Epoch 8/15
678/678 [=====] - ETA: 0s - loss: 3.2403
Epoch 8: val_loss improved from 3.71608 to 3.71512, saving model to model_IV3_GRU.
678/678 [=====] - 188s 277ms/step - loss: 3.2403 - val_loss: 3.71512
Epoch 9/15
678/678 [=====] - ETA: 0s - loss: 3.1757
Epoch 9: val_loss improved from 3.71512 to 3.68882, saving model to model_IV3_GRU.
678/678 [=====] - 184s 272ms/step - loss: 3.1757 - val_loss: 3.68882
Epoch 10/15
678/678 [=====] - ETA: 0s - loss: 3.1189
Epoch 10: val_loss did not improve from 3.68882
678/678 [=====] - 189s 279ms/step - loss: 3.1189 - val_loss: 3.68882
Epoch 11/15
678/678 [=====] - ETA: 0s - loss: 3.0672
Epoch 11: val_loss did not improve from 3.68882
678/678 [=====] - 190s 280ms/step - loss: 3.0672 - val_loss: 3.68882
Epoch 12/15
678/678 [=====] - ETA: 0s - loss: 3.0209
Epoch 12: val_loss did not improve from 3.68882
678/678 [=====] - 188s 277ms/step - loss: 3.0209 - val_loss: 3.68882
Epoch 13/15
678/678 [=====] - ETA: 0s - loss: 2.9772
Epoch 13: val_loss did not improve from 3.68882
678/678 [=====] - 189s 279ms/step - loss: 2.9772 - val_loss: 3.68882
Epoch 14/15
678/678 [=====] - ETA: 0s - loss: 2.9390
Epoch 14: val_loss did not improve from 3.68882
678/678 [=====] - 188s 278ms/step - loss: 2.9390 - val_loss: 3.68882
Epoch 15/15
678/678 [=====] - ETA: 0s - loss: 2.9020
Epoch 15: val_loss did not improve from 3.68882

```

5.2 Plot the loss and accuracy history graphs for both train and validation set. Print the total time taken for training.

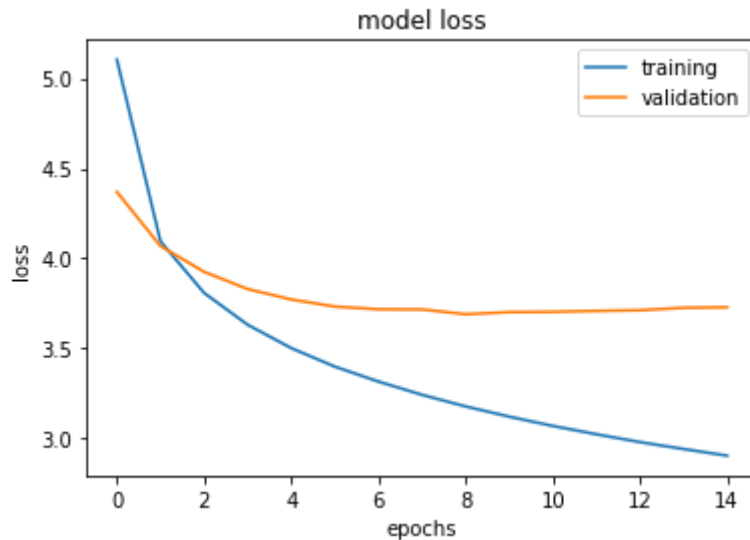
```

# plot training loss and validation loss
import matplotlib.pyplot as plt

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epochs')

```

```
plt.legend(['training', 'validation'], loc='upper right')
plt.show()
```



## Task6 - Model Evaluation

```
# extract features from each photo in the directory
def extract_feat(filename):
    # load the model
    model = InceptionV3(input_shape = (299, 299, 3))
    # re-structure the model
    model = Model(inputs=model.inputs, outputs=model.layers[-2].output)
    # load the photo
    image = load_img(filename, target_size=(299, 299))
    # 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)
    return feature

# map an integer to a word
def word_for_id(integer, tokenizer):
    for word, index in tokenizer.word_index.items():
        if index == integer:
            return word
    return None

# generate a description for an image
def generate_desc(model, tokenizer, photo, max_length):
    # seed the generation process
    in_text = 'startseq'
```

```

# iterate over the whole length of the sequence
for i in range(max_length):
    # integer encode input sequence
    sequence = tokenizer.texts_to_sequences([in_text])[0]
    # pad input
    sequence = pad_sequences([sequence], maxlen=max_length)
    # predict next word
    yhat = model.predict([photo, sequence], verbose=0)
    # convert probability to integer
    yhat = np.argmax(yhat)
    # map integer to word
    word = word_for_id(yhat, tokenizer)
    # stop if we cannot map the word
    if word is None:
        break
    # append as input for generating the next word
    in_text += ' ' + word
    # stop if we predict the end of the sequence
    if word == 'endseq':
        break
return in_text

```

## 6.1 Evaluating model on training images using the latest model

```

from keras.models import load_model
# load the model
modl = load_model('model_IV3_GRU.h5')

# generate description
tokenizr = Tokenizer()
tokenizr.fit_on_texts([caption for image in train_validate_images for caption in new_capti
max_length = 31

for count in range(10):

    photo = extract_feat('{} .jpg'.format(image_dataset_path+'/' + train_validate_images[coun

# reference captions list of the training image for BLEU Score calculation
ref_bleu = []
for caption in new_captions_dict[train_validate_images[count*100]]:
    ref_bleu.append(caption.split())

# generate description
description = generate_desc(modl, tokenizr, photo, max_length)
display(Image(image_dataset_path+'/' + train_validate_images[count*100] + '.jpg'))
print('Predicted caption -> ', description)
print()
print('Actual captions -> ')
for caption in new_captions_dict[train_validate_images[count*100]]: print(caption)
print()
print('BLEU-1 score -> {}'.format(sentence_bleu(ref_bleu, description.split(), weights
print('BLEU-2 score -> {}'.format(sentence_bleu(ref_bleu, description.split(), weights
print('BLEU-3 score -> {}'.format(sentence_bleu(ref_bleu, description.split(), weights

```



```
print('BLEU-4 score -> {}'.format(sentence_bleu(ref_bleu, description.split(), weights
print('*****'))
```



Downloading data from [https://storage.googleapis.com/tensorflow/keras-applications/inception\\_v3/96112376/96112376](https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/96112376/96112376) [=====] - 0s 0us/step



Predicted caption -> startseq two men in black and white shirts are dancing in the s

Actual captions ->

startseq family of five is watching performer standing on stepstool endseq

startseq group of people watch man standing on stool dressed in thong endseq

startseq halfnaked performer is performing for small crowd outside endseq

startseq man in small bathing suit standing on stool at tourist location endseq

startseq mostly nude man standing on stool while family of look at him questioningly

BLEU-1 score -> 0.21428571428571425

BLEU-2 score -> 6.905081759897607e-155

BLEU-3 score -> 5.351494544342064e-204

BLEU-4 score -> 1.2395288183339461e-231

\*\*\*\*\*





Predicted caption -> startseq two children are playing in the water endseq

Actual captions ->

startseq boy at the end of slip and slide endseq

startseq boy is being splashed in the face whilst riding blue ring along yellow water

startseq young boy slides down water slide with his eyes closed endseq

startseq young boy slides on slip slide endseq

startseq young boy splashes down water slide on lawn endseq

BLEU-1 score -> 0.5555555555555556

BLEU-2 score -> 0.26352313834736496

BLEU-3 score -> 1.236983879722816e-102

BLEU-4 score -> 7.657404561915943e-155

\*\*\*\*\*



Predicted caption -> startseq two dogs are running through the snow endseq

Actual captions ->

startseq group of dogs standing by car endseq

startseq pack of dogs fighting near car in an alley endseq

startseq pack of dogs roughhousing by car on dirty street endseq

startseq several dogs brawl in an alley near silver car endseq

startseq three dogs look on as two dogs attack third dog in the streets endseq

BLEU-1 score -> 0.5555555555555556

BLEU-2 score -> 0.26352313834736496

BLEU-3 score -> 1.236983879722816e-102

BLEU-4 score -> 7.657404561915943e-155

\*\*\*\*\*







Predicted caption -> startseq man in red shirt is standing in front of building ends

Actual captions ->

startseq boy in black and white shirt holds red skateboard as he stands over the holl

startseq skateboarder examines the hollywood walk of fame endseq

startseq teenage boy wearing striped shirt walks on street paved with stars carrying

startseq man walking with red skateboard endseq

startseq person wearing black and white striped shirt holding red skateboard standing

BLEU-1 score -> 0.6666666666666666

BLEU-2 score -> 0.24618298195866542

BLEU-3 score -> 1.1826438038915621e-102

BLEU-4 score -> 7.401184483348608e-155

\*\*\*\*\*



Predicted caption -> startseq dog is running through the grass endseq

Actual captions ->

startseq black dog looks up at ball in the air and prepares to catch it endseq

startseq black dog waits on the grass for falling yellow ball endseq

startseq dog is looking up at ball attached to the string endseq

startseq the animal is near the road endseq



startseq the animal is near the pond endseq  
 startseq the dinosaur exhibit is next to tree that has balloon stuck in its branches endseq

BLEU-1 score -> 0.75

BLEU-2 score -> 0.5669467095138409

BLEU-3 score -> 0.3806693978654565

BLEU-4 score -> 5.87583260478785e-78

\*\*\*\*\*



Predicted caption -> startseq man in red eyes is in the water endseq

Actual captions ->

startseq four children lay on tube endseq

startseq several girls on raft floating in rough waters endseq

startseq the four kids are riding raft on the water endseq

startseq the kids ride boat in the water endseq

startseq three girls and one boy are holding onto raft as they are being pulled by rope endseq

BLEU-1 score -> 0.5

BLEU-2 score -> 0.408248290463863

BLEU-3 score -> 0.350372724490772

BLEU-4 score -> 0.2777619034011791

\*\*\*\*\*



Predicted caption -> startseq two children are sitting on swing endseq

Actual captions ->

startseq boy and girl are riding in red seat on fairground ride endseq

startseq girl and boy enjoy fast amusement ride endseq

startseq young girl and boy on ride at an amusement park endseq

startseq two children ride in red seat on fair ride and smile endseq

startseq two kids are on fair ride and are slipping to one side of the car endseq

BLEU-1 score -> 0.6618726769384466

BLEU-2 score -> 0.4085166851999189

BLEU-3 score -> 0.29386714887104765

BLEU-4 score -> 4.685541510640215e-78

\*\*\*\*\*



Predicted caption -> startseq two girls in red and white dresses are sitting on the

Actual captions ->

startseq group of kids plays in the spray of water from fountain endseq

startseq children are being splashed with water endseq

startseq children are playing outside in fountain endseq

startseq five children are being sprayed by water fountain endseq

startseq four children are playing in water fountain endseq

BLEU-1 score -> 0.3571428571428572

BLEU-2 score -> 8.914422220094638e-155

BLEU-3 score -> 6.3341037752361424e-204

BLEU-4 score -> 1.408376648685561e-231

\*\*\*\*\*







## 6.2 Evaluating model on test images



```
# load the model
modl = load_model('model_IV3_GRU.h5')

# generate description
tokenizr = Tokenizer()
tokenizr.fit_on_texts([caption for image in test_images for caption in new_captions_dict[image]
max_length = 31

for count in range(10):

    photo = extract_feat('{} .jpg'.format(image_dataset_path+'/' +test_images[count]))

    # reference captions list of the testing image for BLEU Score calculation
    ref_bleu = []
    for caption in new_captions_dict[test_images[count]]:
        ref_bleu.append(caption.split())

    # generate description
    description = generate_desc(modl, tokenizr, photo, max_length)
    display(Image('{} .jpg'.format(image_dataset_path+'/' +test_images[count])))
    print('Predicted caption -> ', description)
    print()
    print('Actual caption ->')
    for caption in new_captions_dict[test_images[count]]: print(caption)
    print()
    print('BLEU-1 score -> {}'.format(sentence_bleu(ref_bleu, description.split(), weights
    print('BLEU-2 score -> {}'.format(sentence_bleu(ref_bleu, description.split(), weights
    print('BLEU-3 score -> {}'.format(sentence_bleu(ref_bleu, description.split(), weights
    print('BLEU-4 score -> {}'.format(sentence_bleu(ref_bleu, description.split(), weights
    print('*****'))
```



Predicted caption -> startseq white wearing front blue and

Actual caption ->

startseq man jumps gin the air while riding an atv endseq

startseq man on fourwheeler jumps near small building endseq

startseq an atv is airborne over field in front of white structure endseq

startseq person dressed in tan jacket jumps quad over harvested cornfield in front of

startseq man on four wheeler in the air endseq

BLEU-1 score -> 0.3032653298563167

BLEU-2 score -> 6.397495320955232e-155

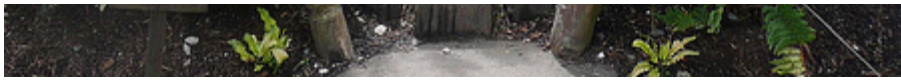
BLEU-3 score -> 4.2929930481953196e-204

BLEU-4 score -> 9.291879812217675e-232

\*\*\*\*\*







Predicted caption -> startseq boy in dogs while man are and the

Actual caption ->

startseq bridge through high green plants endseq  
 startseq man and woman are crossing over rope bridge with greenery all over them endseq  
 startseq man and woman are walking across rope bridge endseq  
 startseq man and woman crossing suspension bridge in tropical setting endseq  
 startseq woman and man walking across wooden rope bridge with caution sign beside it

BLEU-1 score -> 0.49713295378576094

BLEU-2 score -> 9.94903642112384e-155

BLEU-3 score -> 6.557713753086882e-204

BLEU-4 score -> 1.40745738725685e-231

\*\*\*\*\*



Predicted caption -> startseq brown in person

Actual caption ->

startseq boy smiles underwater endseq  
 startseq redheaded boy swimming underwater endseq  
 startseq small boy swimming underwater endseq  
 startseq smiling boy swims underwater in pool endseq  
 startseq the boys smiles underwater at the pool endseq

BLEU-1 score -> 0.38940039153570244

BLEU-2 score -> 8.214546595247418e-155

BLEU-3 score -> 5.512312187546572e-204

BLEU-4 score -> 1.1931009847695213e-231

\*\*\*\*\*





Predicted caption -> startseq white high girl setting endseq

Actual caption ->

startseq boy carrying soccer ball endseq

startseq boy walks with ball tucked under his arm endseq

startseq boy walks with soccer ball near fence endseq

startseq boy wearing white tshirt walks on the grass and carries soccer ball endseq

startseq small boy carries soccer ball on field endseq

BLEU-1 score -> 0.5

BLEU-2 score -> 1.0547686614863434e-154

BLEU-3 score -> 7.077948953527403e-204

BLEU-4 score -> 1.5319719891192393e-231

\*\*\*\*\*



Predicted caption -> startseq white two catch front snowboard in the camera endseq

Actual caption ->

startseq little girl is holding cine camera in front of her face endseq

startseq young girl is looking through an old fashioned video camera endseq

startseq young girl steadies her aim with camera endseq

startseq girl with rosy cheeks and lips holding black toy gun endseq

startseq there is kid with gun endseq

BLEU-1 score -> 0.5

BLEU-2 score -> 0.2357022603955158

BLEU-3 score -> 1.149168560061151e-102

BLEU-4 score -> 7.241926111174567e-155

\*\*\*\*\*





Predicted caption -> startseq boy in dogs while man are and the

Actual caption ->

startseq kid rock climbing against the backdrop of green valley endseq

startseq woman in striped shirt climbs up mountain endseq

startseq young man climbs rocky hill endseq

startseq the person has striped shirt on and is holding on to rope on mountain endseq

startseq the person in the striped shirt is mountain climbing endseq

BLEU-1 score -> 0.5555555555555556

BLEU-2 score -> 1.1118237916213198e-154

BLEU-3 score -> 7.328370166425364e-204

BLEU-4 score -> 1.5728604687011317e-231

\*\*\*\*\*





Predicted caption -> startseq boy in dogs while man snowboard and the camera endseq

Actual caption ->

startseq man in shorts is jogging along street with headset endseq

startseq person in blue shorts and wearing walkman jogs endseq

startseq woman in white shirt and blue shorts is wearing headphones endseq

startseq woman walks on street wearing headphones endseq

startseq woman wearing white hat and shirt is jogging down street with plant store or

BLEU-1 score -> 0.5454545454545454

BLEU-2 score -> 1.1016699370024176e-154

BLEU-3 score -> 7.284129382816144e-204

BLEU-4 score -> 1.5656618337072542e-231

\*\*\*\*\*



Predicted caption -> startseq brown in person while man to and the carries endseq

Actual caption ->

startseq guy stands in the sand with snowboard behind him endseq

startseq man holds surfboard on the beach endseq

startseq man holds his snowboard in the sand endseq



startseq man with his surfboard stands in the sand endseq  
 startseq man is standing on white sand and holding snowboard endseq

BLEU-1 score -> 0.5454545454545454

BLEU-2 score -> 1.1016699370024176e-154

BLEU-3 score -> 7.284129382816144e-204

BLEU-4 score -> 1.5656618337072542e-231

\*\*\*\*\*



Predicted caption -> startseq with man soccer and the camera endseq

Actual caption ->

startseq black and white border collie catches frisbee in front of an audience endseq

startseq brown and white dog catches frisbee in it mouth in front of group of people

startseq brown and white dog jumping up to catch frisbee while an audience watches er

startseq dog jumps to catch frisbee endseq

startseq an agile dog catches frisbee while crowd of onlookers watches closely endseq

BLEU-1 score -> 0.375

BLEU-2 score -> 9.134564559628536e-155

BLEU-3 score -> 6.43691305555604e-204

BLEU-4 score -> 1.4256605770826504e-231

\*\*\*\*\*





Predicted caption -> startseq jumps with man soccer walks the frisbee endseq

..

startseq two brown dogs in the water endseq

startseq two brown dogs running through water endseq

startseq two brown dogs runs through the water endseq

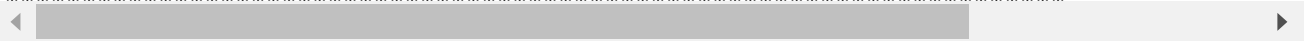
startseq two dogs splash through the water endseq

BLEU-1 score -> 0.3333333333333333

BLEU-2 score -> 8.612150057732663e-155

BLEU-3 score -> 6.19152043543562e-204

BLEU-4 score -> 1.384292958842266e-231



[Colab paid products](#) - [Cancel contracts here](#)

✓ 0s completed at 2:56 PM

