Double-click (or enter) to edit

Image Captioning - DL Assignment2

Assignment Details

1. **Group:90**
2. **Members**
    - Afrah Khan - 2019AD04096
    - Inderdeep Singh - 2019AD04061
    - Partha Pratim Saha - 2019AD04100

Import Libraries/Dataset

```
import pandas as pd
import numpy as np
import re
import os.path
import tensorflow as tf
from tensorflow.keras.preprocessing.image import load_img
import matplotlib.pyplot as plt
import glob
from PIL import Image
import string
import time
pd.set_option('display.max_colwidth', None)
```

Checking GPU availability

```
print("Version: ", tf.__version__)
print("Eager mode: ", tf.executing_eagerly())
print("GPU is", "available" if tf.config.list_physical_devices("GPU") else "NOT AVAILAB
print("GPU device name:", tf.test.gpu_device_name())
```

```
Version:  2.5.0
Eager mode:  True
GPU is available
GPU device name: /device:GPU:0
```

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

```
Mounted at /content/drive
```

Data Visualization and augmentation:

Read the pickle file

```
pkl_df = pd.read_pickle("/content/set_4.pkl")
```

```
pkl_list=[]
for d in pkl_df:
  pkl_list.append(re.split('#|\t',d,maxsplit=2))
```

```
mdf = pd.DataFrame(pkl_list,columns =['ImageName', 'id','Caption'])
```

```
mdf = mdf.groupby(['ImageName']).sum()
mdf.drop(columns=['id'],inplace=True)
mdf.reset_index(inplace=True)
```

```
mdf.head()
```

	ImageName	Caption
0	1000268201_693b08cb0e.jpg	A girl go into a wooden building .A child in a pink dress be climb up a set of stair in an entry way .A little girl climb into a wooden playhouse .A little girl climb the stair to her playhouse .A little girl in a pink dress go into a wooden cabin .
1	1001773457_577c3a7d70.jpg	A black dog and a tri-colored dog play with each other on a road .A black dog and a white dog with brown spot be stare at each other in a street .Two dog on pavement move toward each other .
2	1002674143_1b742ab4b8.jpg	There be a girl with pigtail sit in front of a rainbow paint .A little girl be sit in front of a large painted rainbow .
		A man lay on a bench while his dog sit by him .A man lay on a

Image dataset

```
import os
os.getcwd()
```

```
'/content/drive/My Drive/Flicker8k_Dataset'
```

Convert the data into the correct format which could be used for ML model.

```
from tqdm import tqdm
image_dict = {}
for filename in tqdm(glob.glob('*.jpg')):
    img=load_img(filename,target_size=(224,224))
    image_dict[filename]=img
    image_dict[filename]=img
```

```
100%|██████████| 8091/8091 [46:45<00:00,  2.88it/s]
```

```
len(image_dict)
```

```
8091
```

```
dic={}
for i in mdf.ImageName:
  for j in image_dict:
    if i in j:
      dic[i]=image_dict[j]
```

```
df_test = pd.DataFrame.from_dict(dic,orient='index',columns=['Image'])
df_test.reset_index(inplace=True)
df_test.rename(columns={'index':'ImageName'},inplace=True)
```

```
main_df = pd.merge(mdf,df_test,on=['ImageName'])
```

```
main_df.head()
```

	ImageName	Caption	Image
0	1000268201_693b08cb0e.jpg	A girl go into a wooden building .A child in a pink dress be climb up a set of stair in an entry way .A little girl climb into a wooden playhouse .A little girl climb the stair to her playhouse .A little girl in a pink dress go into a wooden cabin .	<PIL.Image.Image image mode=RGB size=224x224 at 0x7F069F0719D0>
1	1001773457_577c3a7d70.jpg	A black dog and a tri-colored dog play with each other on a road .A black dog and a white dog with brown spot be stare at each other in a street .Two dog on pavement move toward each other .	<PIL.Image.Image image mode=RGB size=224x224 at 0x7F069EEF1550>
2	1002674143_1b742ab4b8.jpg	There be a girl with pigtail sit in front of a rainbow paint .A little girl be sit in front of a	<PIL.Image.Image image mode=RGB size=224x224 at

```
main_df.info()
```

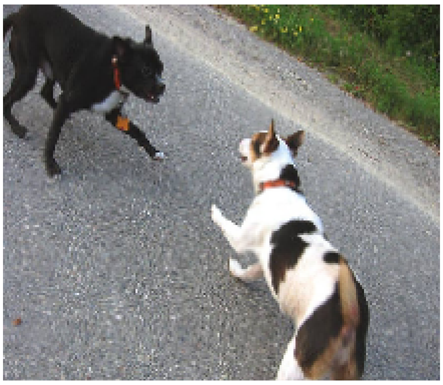
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8032 entries, 0 to 8031
Data columns (total 3 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   ImageName  8032 non-null   object
 1   Caption    8032 non-null   object
 2   Image      8032 non-null   object
dtypes: object(3)
memory usage: 251.0+ KB
```

Plot at least two samples and their captions (use matplotlib/seaborn/any other library)

```
for i in range(len(main_df.head())):
  print("Caption: "+ str(main_df['Caption'][i].split(".")[:-1]))
  plt.imshow(main_df['Image'][i])
  plt.axis(False)
  plt.show()
```

```
Caption: ['A child in a pink dress be climb up a set of stair in an entry way ', 'A 
```

```
Caption: ['Two dog of different breed look at each other on a road ']
```

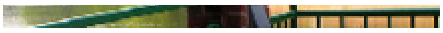


Bring the train and test data in the required format

```
image = main_df['Image']
for i in range(len(image)):
  image[i]=np.asarray(image[i])
```

```
from sklearn.model_selection import train_test_split
# keeping 50% size, as colab session crashes if we increase the train size
train, test = train_test_split(main_df, test_size=0.5)
```

```
caption_dict={}
for i in range(len(train)):
  caption_dict[train.ImageName.values[i]]=["START "+sent.strip()+" END" for sent in tra
```



```
caption_dict.get('1000268201_693b08cb0e.jpg')
```

```
['START a girl go into a wooden building END',
 'START a child in a pink dress be climb up a set of stair in an entry way END',
 'START a little girl climb into a wooden playhouse END',
 'START a little girl climb the stair to her playhouse END',
 'START a little girl in a pink dress go into a wooden cabin END']
```

```
count_words = {}
count=1
for k,vv in caption_dict.items():
    for v in vv:
        for word in v.split():
            if word not in count_words:
              count_words[word] = count
              count +=1
```

```
len(count_words)
```

```
4453
```

```
for k,vv in caption_dict.items():
    for v in vv:
        encode=[]
        for word in v.split():
          encode.append(count_words[word])
        caption_dict[k][vv.index(v)]=encode
```

```
caption_dict.get('1000268201_693b08cb0e.jpg')
```

```
[[1, 2, 12, 473, 85, 2, 72, 382, 11],
 [1,
  2,
  37,
  4,
  2,
  249,
  119,
  38,
  168,
  158,
  2,
  182,
  17,
  181,
  4,
  223,
  1356,
  457,
  11],
 [1, 2, 331, 12, 168, 85, 2, 72, 1357, 11],
 [1, 2, 331, 12, 168, 9, 181, 78, 210, 1357, 11],
 [1, 2, 331, 12, 4, 2, 249, 119, 473, 85, 2, 72, 1358, 11]]
```

Model Building

Use Pretrained Resnet-50 model trained on ImageNet dataset

```
from tensorflow.keras.applications import ResNet50
resnet_model = ResNet50()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/r
102973440/102967424 [==============================] - 3s 0us/step
```

```
from tensorflow.keras.models import Model
```

```
image_model = Model(inputs=resnet_model.input, outputs=resnet_model.layers[-2].output)
```

Bring the train in the required format.

```
image_feat = {}
for im in range(len(train)):
  img = train['Image'].values[im].reshape(1,224,224,3)
  pred = image_model.predict(img).reshape(2048,)
  image_feat[train.ImageName.values[im]] = pred
```

```
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

```
MAX_LEN = 0
for k, vv in caption_dict.items():
    for v in vv:
        if len(v) > MAX_LEN:
            MAX_LEN = len(v)
```

```
MAX_LEN
```

```
46
```

```
VOCAB_SIZE = len(count_words)
def generator(photo, caption):
    n_samples = 0
    X = []
    y_in = []
    y_out = []
    for k, vv in caption.items():
        for v in vv:
            for i in range(1, len(v)):
              try:
                X.append(photo[k])
                in_seq= [v[:i]]
                out_seq = v[i]
                in_seq = pad_sequences(in_seq, maxlen=MAX_LEN, padding='post', truncati
                out_seq = to_categorical([out_seq], num_classes=VOCAB_SIZE+1)[0]
                y_in.append(in_seq)
                y_out.append(out_seq)
              except:
                pass

    return X, y_in, y_out
```

```
X, y_in, y_out = generator(image_feat, caption_dict)
```

```
X = np.asarray(X)
y_in = np.asarray(y_in)
y_out = np.asarray(y_out)
```

```
X.shape,y_in.shape,y_out.shape
```

```
    ((143214, 2048), (143214, 46), (143214, 4454))
```

```
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras import layers
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
from tensorflow.keras.layers import Embedding
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import RepeatVector
from tensorflow.keras.layers import TimeDistributed
from tensorflow.keras.layers import Concatenate
```

Four(4) layered RNN - We are using LSTM as it is an RNN architecture

Add one layer of dropout at the appropriate position and give reasons.

Adding **L2 regularization** to all the **RNN** layers and one layer of 20% dropout - adding before last layer of **LSTM** abecause they are the one with the greater number of parameters and thus they're likely to excessively co-adapting themselves causing overfitting. However, since it's a stochastic regularization technique, we can really place it everywhere.

Choose the appropriate activation function for all the layers.

Adding **softmax** as appropriate activation function - **Adam** optimizer as it converges really faster and adding **categorical crossentropy** as loss function with **accuracy** as performance metric

Give reasons for the choice of learning rate and its value: Adding learning rate as 0.0001 as it was hypertuned and gave better results

Model Compilation

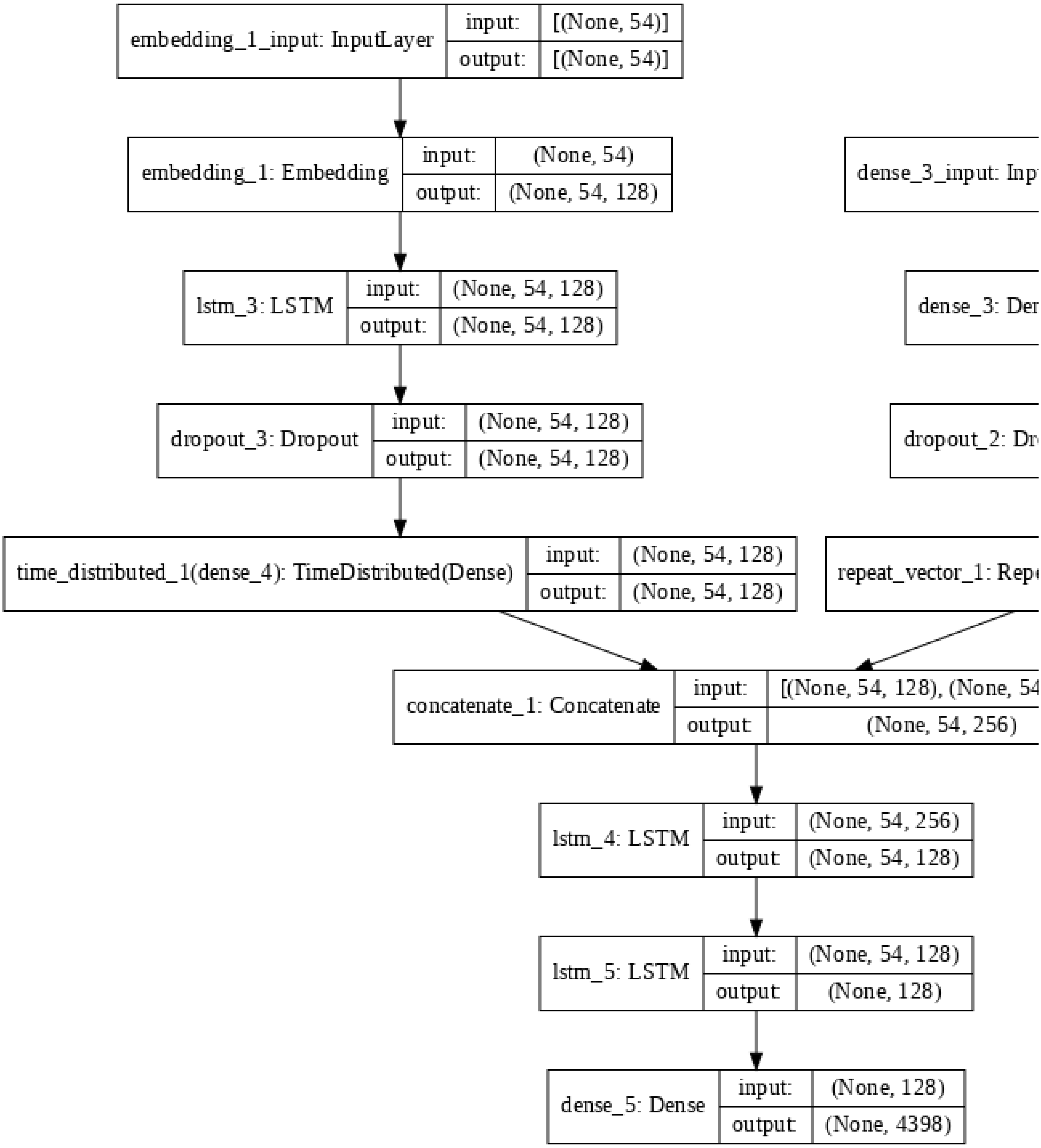
```
embedding_size = 128
max_len = MAX_LEN
vocab_size = len(count_words) + 1
image_model1 = Sequential()
image_model1.add(Dense(embedding_size, input_shape=(2048,), activation='relu'))
image_model1.add(Dropout(0.2))
```

```
image_model1.add(Dropout(0.2))
image_model1.add(RepeatVector(max_len))
language_model = Sequential()
language_model.add(Embedding(input_dim=vocab_size, output_dim=embedding_size, input_len
language_model.add(LSTM(128, kernel_regularizer=tf.keras.regularizers.l2(l2=0.0001), re
language_model.add(Dropout(0.2))
language_model.add(TimeDistributed(Dense(embedding_size)))
conca = Concatenate()([image_model1.output, language_model.output])2
#Add L2 regularization to all the RNN layers.
x = LSTM(128, kernel_regularizer=tf.keras.regularizers.l2(l2=0.0001), return_sequences=
x = LSTM(128, kernel_regularizer=tf.keras.regularizers.l2(l2=0.0001), return_sequences=
out = Dense(vocab_size,activation='softmax')(x)
model = Model(inputs=[image_model1.input, language_model.input], outputs = out)

#Compiling the model with above parameters
model.compile(optimizer=tf.keras.optimizers.Adam(1e-3), loss=tf.losses.categorical_cros
#Print the model summary
model.summary()
```

```
Model: "model_2"
__________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==========================================================================================
embedding_1_input (InputLayer)  [(None, 46)]         0
__________________________________________________________________________________________
dense_3_input (InputLayer)      [(None, 2048)]       0
__________________________________________________________________________________________
embedding_1 (Embedding)         (None, 46, 128)      570112      embedding_1_input[0]
__________________________________________________________________________________________
dense_3 (Dense)                 (None, 128)          262272      dense_3_input[0][0]
__________________________________________________________________________________________
lstm_3 (LSTM)                   (None, 46, 128)      131584      embedding_1[0][0]
__________________________________________________________________________________________
dropout_2 (Dropout)             (None, 128)          0           dense_3[0][0]
__________________________________________________________________________________________
dropout_3 (Dropout)             (None, 46, 128)      0           lstm_3[0][0]
__________________________________________________________________________________________
repeat_vector_1 (RepeatVector)  (None, 46, 128)      0           dropout_2[0][0]
__________________________________________________________________________________________
time_distributed_1 (TimeDistrib (None, 46, 128)      16512       dropout_3[0][0]
__________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 46, 256)      0           repeat_vector_1[0][0
                                                                 time_distributed_1[0
__________________________________________________________________________________________
lstm_4 (LSTM)                   (None, 46, 128)      197120      concatenate_1[0][0]
__________________________________________________________________________________________
lstm_5 (LSTM)                   (None, 128)          131584      lstm_4[0][0]
__________________________________________________________________________________________
dense_5 (Dense)                 (None, 4454)         574566      lstm_5[0][0]
==========================================================================================
Total params: 1,883,750
Trainable params: 1,883,750
Non-trainable params: 0
__________________________________________________________________________________________
```

```
from tensorflow.keras.utils import plot_model
plot_model(model,show_shapes=True,dpi=72)
```



Model Training

Print the train and validation loss for each epoch. Use the appropriate batch size

```
start = time.time()
history = model.fit([X, y_in], y_out, batch_size=720, epochs=35, validation_split=0.1,
end = time.time()
```

```
Epoch 1/35
180/180 [==============================] - 33s 146ms/step - loss: 5.5007 - accurac
Epoch 2/35
180/180 [==============================] - 25s 136ms/step - loss: 5.2176 - accurac
Epoch 3/35
180/180 [==============================] - 25s 137ms/step - loss: 5.1917 - accurac
```

```
Epoch 4/35
180/180 [==============================] - 25s 138ms/step - loss: 5.1286 - accuracy
Epoch 5/35
180/180 [==============================] - 25s 138ms/step - loss: 5.0974 - accuracy
Epoch 6/35
180/180 [==============================] - 25s 136ms/step - loss: 5.0780 - accuracy
Epoch 7/35
180/180 [==============================] - 25s 136ms/step - loss: 5.0485 - accuracy
Epoch 8/35
180/180 [==============================] - 25s 139ms/step - loss: 5.0293 - accuracy
Epoch 9/35
180/180 [==============================] - 25s 139ms/step - loss: 5.0088 - accuracy
Epoch 10/35
180/180 [==============================] - 25s 138ms/step - loss: 4.9890 - accuracy
Epoch 11/35
180/180 [==============================] - 25s 138ms/step - loss: 4.9679 - accuracy
Epoch 12/35
180/180 [==============================] - 25s 139ms/step - loss: 4.9458 - accuracy
Epoch 13/35
180/180 [==============================] - 25s 138ms/step - loss: 4.9252 - accuracy
Epoch 14/35
180/180 [==============================] - 25s 137ms/step - loss: 4.8974 - accuracy
Epoch 15/35
180/180 [==============================] - 25s 137ms/step - loss: 4.8752 - accuracy
Epoch 16/35
180/180 [==============================] - 25s 138ms/step - loss: 4.8574 - accuracy
Epoch 17/35
180/180 [==============================] - 25s 140ms/step - loss: 4.8331 - accuracy
Epoch 18/35
180/180 [==============================] - 25s 138ms/step - loss: 4.8143 - accuracy
Epoch 19/35
180/180 [==============================] - 25s 138ms/step - loss: 4.7929 - accuracy
Epoch 20/35
180/180 [==============================] - 25s 139ms/step - loss: 4.7757 - accuracy
Epoch 21/35
180/180 [==============================] - 25s 139ms/step - loss: 4.7597 - accuracy
Epoch 22/35
180/180 [==============================] - 25s 138ms/step - loss: 4.7465 - accuracy
Epoch 23/35
180/180 [==============================] - 25s 137ms/step - loss: 4.7344 - accuracy
Epoch 24/35
180/180 [==============================] - 25s 138ms/step - loss: 4.7160 - accuracy
Epoch 25/35
180/180 [==============================] - 25s 139ms/step - loss: 4.7021 - accuracy
Epoch 26/35
180/180 [==============================] - 25s 138ms/step - loss: 4.6927 - accuracy
Epoch 27/35
180/180 [==============================] - 25s 138ms/step - loss: 4.6763 - accuracy
Epoch 28/35
180/180 [==============================] - 25s 139ms/step - loss: 4.6613 - accuracy
Epoch 29/35
180/180 [==============================] - 25s 139ms/step - loss: 4.6546 - accuracy
```

total time taken for training: ~15 mins

```
# Print the total time taken for training
print("Total time taken for training: ", (end - start)/60, "mins")
```

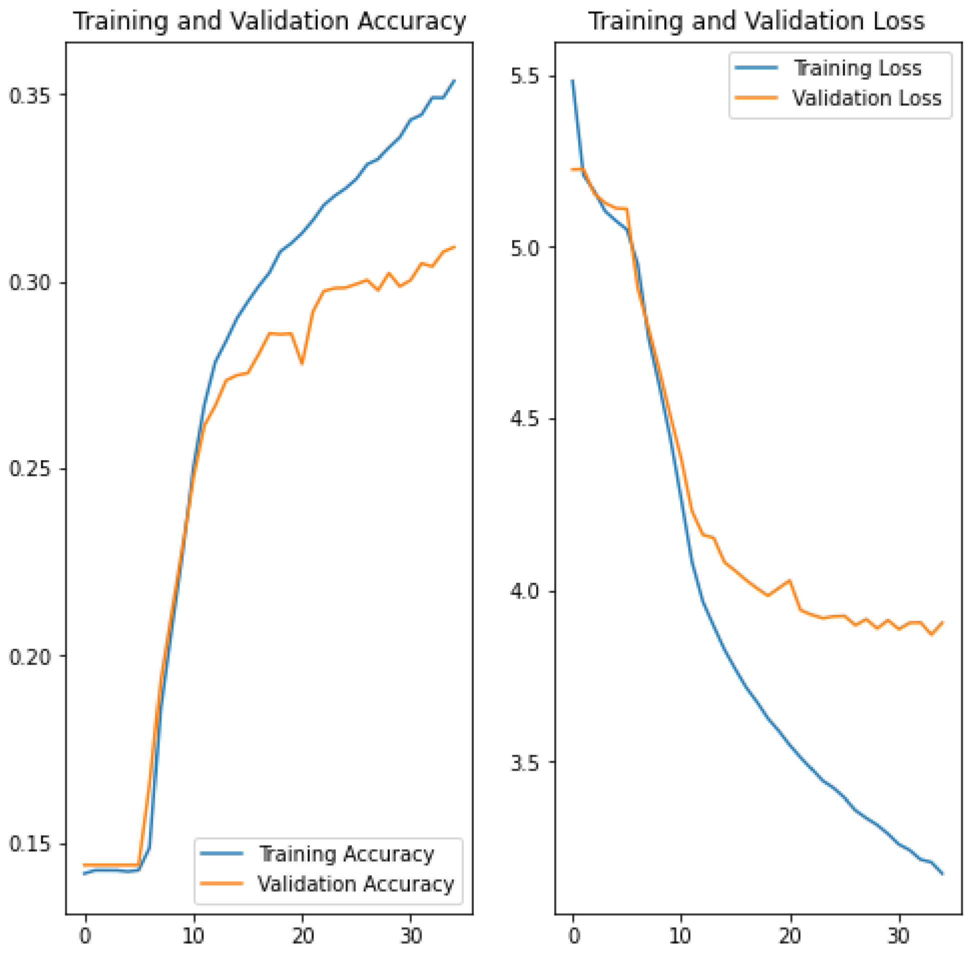
```
Total time taken for training:  15.453375518321991 mins
```

Plot the loss and accuracy history graphs for both train and validation set

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs_range = range(len(history.epoch))

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```



```
inv_dict = {v:k for k, v in count_words.items()}
```

```
def getImage(x):
```

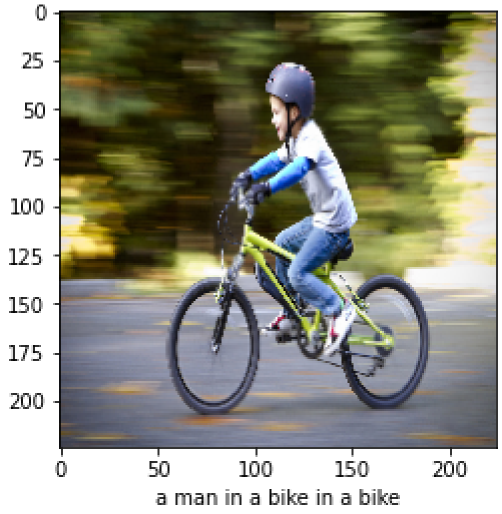
```
def getImage(x):
    test_img = np.reshape(x, (1,224,224,3))
    return test_img
```

Model Evaluation: Take a random image from google and generate caption for that image

```
testImg = load_img('/content/test_image3.jpg',target_size=(224,224,3))
```

```
test_pred = image_model.predict(getImage(testImg)).reshape(1,2048)
text_inp = ['START']
count = 0
caption = ''
while count < 25:
            count += 1
            encoded = []
            for i in text_inp:
                encoded.append(count_words[i])
            encoded = [encoded]
            encoded = pad_sequences(encoded, padding='post', truncating='post', maxlen=
            prediction = np.argmax(model.predict([test_pred, encoded]))
            sampled_word = inv_dict[prediction]
            caption = caption + ' ' + sampled_word
            if sampled_word == 'END':
                break
            text_inp.append(sampled_word)
plt.figure()
plt.imshow(testImg)
plt.xlabel(caption.replace("END","").strip())
```

```
Text(0.5, 0, 'a man in a bike in a bike')
```



✓ 1s completed at 23:02 ● ×