```
! wget https://storage.googleapis.com/4705-hw5-data/hw5data-20220809T182644Z-001.zip
! unzip hw5data-20220809T182644Z-001.zip
```

```
  inflating: hw5data/Flickr8k_Dataset/3188044631_ca3a9cc737.jpg
  inflating: hw5data/Flickr8k_Dataset/3738789925_7d17dbdf25.jpg
  inflating: hw5data/Flickr8k_Dataset/3425414048_fa14d33067.jpg
  inflating: hw5data/Flickr8k_Dataset/3205214191_29b42b9b09.jpg
  inflating: hw5data/Flickr8k_Dataset/3239021459_a6b71bb400.jpg
  inflating: hw5data/Flickr8k_Dataset/3215081286_d55541aa6b.jpg
  inflating: hw5data/Flickr8k_Dataset/3168841415_c0705a327a.jpg
  inflating: hw5data/Flickr8k_Dataset/3227594168_3351722aae.jpg
  inflating: hw5data/Flickr8k_Dataset/3259992164_94600858b3.jpg
  inflating: hw5data/Flickr8k_Dataset/3184206563_5435f2b494.jpg
  inflating: hw5data/Flickr8k_Dataset/322791392_aa3b142f43.jpg
  inflating: hw5data/Flickr8k_Dataset/3250589803_3f440ba781.jpg
  inflating: hw5data/Flickr8k_Dataset/3205754736_32c29b5208.jpg
  inflating: hw5data/Flickr8k_Dataset/3184031654_34b5c4ffe1.jpg
  inflating: hw5data/Flickr8k_Dataset/3258391809_38fc6211f7.jpg
  inflating: hw5data/Flickr8k_Dataset/464251704_b0f0c4c87a.jpg
  inflating: hw5data/Flickr8k_Dataset/3342272425_804316cb3d.jpg
  inflating: hw5data/Flickr8k_Dataset/3457784061_8f77f43a9c.jpg
  inflating: hw5data/Flickr8k_Dataset/3180806542_49b6de312d.jpg
  inflating: hw5data/Flickr8k_Dataset/3333017828_b930b9d41b.jpg
  inflating: hw5data/Flickr8k_Dataset/3256275785_9c3af57576.jpg
  inflating: hw5data/Flickr8k_Dataset/530661899_94655d7d0e.jpg
  inflating: hw5data/Flickr8k_Dataset/3707077198_efd6aa808d.jpg
  inflating: hw5data/Flickr8k_Dataset/3282121432_648dac8a29.jpg
  inflating: hw5data/Flickr8k_Dataset/3163477256_073605e06e.jpg
  inflating: hw5data/Flickr8k_Dataset/3532194771_07faf20d76.jpg
  inflating: hw5data/Flickr8k_Dataset/3230101918_7d81cb0fc8.jpg
  inflating: hw5data/Flickr8k_Dataset/3215108916_0473007b47.jpg
  inflating: hw5data/Flickr8k_Dataset/3482879314_d3387e95b1.jpg
  inflating: hw5data/Flickr8k_Dataset/3286543624_7a327f79ae.jpg
  inflating: hw5data/Flickr8k_Dataset/3547000169_40191e02ca.jpg
  inflating: hw5data/Flickr8k_Dataset/3315110972_1090d11728.jpg
  inflating: hw5data/Flickr8k_Dataset/3257103624_e76f25ff9e.jpg
  inflating: hw5data/Flickr8k_Dataset/3482237861_605b4f0fd9.jpg
  inflating: hw5data/Flickr8k_Dataset/3264937930_9623496b64.jpg
  inflating: hw5data/Flickr8k_Dataset/325005410_e1ff5041b5.jpg
  inflating: hw5data/Flickr8k_Dataset/743571049_68080e8751.jpg
  inflating: hw5data/Flickr8k_Dataset/3213395965_2a823c6865.jpg
  inflating: hw5data/Flickr8k_Dataset/3294202771_e8ee78a439.jpg
  inflating: hw5data/Flickr8k_Dataset/3381392182_db2c42430e.jpg
  inflating: hw5data/Flickr8k_Dataset/3264464625_c711cc40c6.jpg
  inflating: hw5data/Flickr8k_Dataset/3264350290_f50494e835.jpg
  inflating: hw5data/Flickr8k_Dataset/3385956569_a849218e34.jpg
  inflating: hw5data/Flickr8k_Dataset/3270083123_fcc1208053.jpg
  inflating: hw5data/Flickr8k_Dataset/3271178748_630d269811.jpg
  inflating: hw5data/Flickr8k_Dataset/3264650118_be7df266e7.jpg
  inflating: hw5data/Flickr8k_Dataset/3479050296_65bcea69a0.jpg
  inflating: hw5data/Flickr8k_Dataset/3258394043_a0b6a94dce.jpg
  inflating: hw5data/Flickr8k_Dataset/3371887001_44ab0c2f17.jpg
  inflating: hw5data/Flickr8k_Dataset/3419238351_ac18b440c0.jpg
  inflating: hw5data/Flickr8k_Dataset/3524519277_bd0c3e7382.jpg
  inflating: hw5data/Flickr8k_Dataset/3271252073_0a1b9525fc.jpg
```

```
        inflating: hw5data/Flickr8k_Dataset/3719461451_07de35af3a.jpg
        inflating: hw5data/Flickr8k_Dataset/3254645823_a7c072481c.jpg
        inflating: hw5data/Flickr8k_Dataset/3346289227_198fced308.jpg
        inflating: hw5data/Flickr8k_Dataset/3328495660_ed0e3f29cf.jpg
        inflating: hw5data/Flickr8k_Dataset/3232252882_05db7c2216.jpg
        inflating: hw5data/Flickr8k_Dataset/349889354_4b2889a9bd.jpg
```

```
import os
from collections import defaultdict
import numpy as np
import PIL
from matplotlib import pyplot as plt
%matplotlib inline

from keras import Sequential, Model
from keras.layers import Embedding, LSTM, Dense, Input, Bidirectional, RepeatVector, Concaten
from keras.activations import softmax
from keras.utils import to_categorical
#from keras.preprocessing.sequence import pad_sequences
from keras.utils import pad_sequences

from keras.applications.inception_v3 import InceptionV3

from keras.optimizers import Adam


import os
from collections import defaultdict
import numpy as np
import PIL
from matplotlib import pyplot as plt
%matplotlib inline

from tensorflow.keras import Sequential, Model
from tensorflow.keras.layers import Embedding, LSTM, Dense, Input, Bidirectional, RepeatVecto
from tensorflow.keras.activations import softmax
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing.sequence import pad_sequences

from tensorflow.keras.applications.inception_v3 import InceptionV3

from tensorflow.keras.optimizers import Adam


FLICKR_PATH="hw5data"

def load_image_list(filename):
    with open(filename,'r') as image_list_f:
        return [line.strip() for line in image_list_f]

train_list = load_image_list(os.path.join(FLICKR_PATH, 'Flickr_8k.trainImages.txt'))
dev_list = load_image_list(os.path.join(FLICKR_PATH,'Flickr_8k.devImages.txt'))
```

```
test_list = load_image_list(os.path.join(FLICKR_PATH,'Flickr_8k.testImages.txt'))

print(len(train_list), len(dev_list), len(test_list))
print(dev_list[20])
IMG_PATH = os.path.join(FLICKR_PATH, "Flickr8k_Dataset")

image = PIL.Image.open(os.path.join(IMG_PATH, dev_list[20]))
print(image)
print(plt.imshow(image))
print(np.asarray(image).shape)
print(np.asarray(image))
new_image = np.asarray(image.resize((299,299))) / 255.0
print(plt.imshow(new_image))
print(new_image.shape)

def get_image(image_name):
    image = PIL.Image.open(os.path.join(IMG_PATH, image_name))
    return np.asarray(image.resize((299,299))) / 255.0

print(plt.imshow(get_image(dev_list[25])))
```

```
6000 1000 1000
3693961165_9d6c333d5b.jpg
<PIL.JpegImagePlugin.JpegImageFile image mode=RGB size=500x333 at 0x7FC436906580>
AxesImage(54,36;334.8x217.44)
(333, 500, 3)
[[[118 161  89]
  [120 164  89]
  [111 157  82]
  ...
  [ 68 106  65]
  [ 64 102  61]
  [ 65 104  60]]

 [[125 168  96]
  [121 164  92]
  [119 165  90]
  ...
  [ 72 115  72]
  [ 65 108  65]
  [ 72 115  70]]

 [[129 175 102]
  [123 169  96]
  [115 161  88]
  ...
  [ 88 129  87]
  [ 75 116  72]
  [ 75 116  72]]

 ...

 [[ 41 118  46]
  [ 36 113  41]
  [ 45 111  49]
  ...
  [ 23  77  15]
  [ 60 114  62]
  [ 19  59   0]]
```

```
img_model = InceptionV3(weights = 'imagenet')
img_model.summary()
```

```
batch_normalization_83 (BatchN  (None, 8, 8, 384)   1152        ['conv2d_83[0][0]']
ormalization)

conv2d_84 (Conv2D)             (None, 8, 8, 192)    245760      ['average_pooling2d_

batch_normalization_76 (BatchN  (None, 8, 8, 320)   960         ['conv2d_76[0][0]']
ormalization)

activation_78 (Activation)     (None, 8, 8, 384)    0           ['batch_normalizatio

activation_79 (Activation)     (None, 8, 8, 384)    0           ['batch_normalizatio

activation_82 (Activation)     (None, 8, 8, 384)    0           ['batch_normalizatio
```

```
activation_83 (Activation)     (None, 8, 8, 384)    0           ['batch_normalizatio

batch_normalization_84 (BatchN  (None, 8, 8, 192)   576         ['conv2d_84[0][0]']
ormalization)

activation_76 (Activation)     (None, 8, 8, 320)    0           ['batch_normalizatio

mixed9_0 (Concatenate)         (None, 8, 8, 768)    0           ['activation_78[0][0
                                                                 'activation_79[0][0

concatenate (Concatenate)      (None, 8, 8, 768)    0           ['activation_82[0][0
                                                                 'activation_83[0][0

activation_84 (Activation)     (None, 8, 8, 192)    0           ['batch_normalizatio

mixed9 (Concatenate)           (None, 8, 8, 2048)   0           ['activation_76[0][0
                                                                 'mixed9_0[0][0]',
                                                                 'concatenate[0][0]'
                                                                 'activation_84[0][0

conv2d_89 (Conv2D)             (None, 8, 8, 448)    917504      ['mixed9[0][0]']

batch_normalization_89 (BatchN  (None, 8, 8, 448)   1344        ['conv2d_89[0][0]']
ormalization)

activation_89 (Activation)     (None, 8, 8, 448)    0           ['batch_normalizatio

conv2d_86 (Conv2D)             (None, 8, 8, 384)    786432      ['mixed9[0][0]']

conv2d_90 (Conv2D)             (None, 8, 8, 384)    1548288     ['activation_89[0][0

batch_normalization_86 (BatchN  (None, 8, 8, 384)   1152        ['conv2d_86[0][0]']
ormalization)

batch_normalization_90 (BatchN  (None, 8, 8, 384)   1152        ['conv2d_90[0][0]']
ormalization)

activation_86 (Activation)     (None, 8, 8, 384)    0           ['batch_normalizatio

activation_90 (Activation)     (None, 8, 8, 384)    0           ['batch_normalizatio

conv2d_87 (Conv2D)             (None, 8, 8, 384)    442368      ['activation_86[0][0

conv2d_88 (Conv2D)             (None, 8, 8, 384)    442368      ['activation_86[0][0
```

```
new_input = img_model.input
new_output = img_model.layers[-2].output
img_encoder = Model(new_input, new_output)

encoded_image = img_encoder.predict(np.array([new_image]))
print(encoded_image)
```

```
1/1 [==============================] - 5s 5s/step
[[0.6380656  0.48873064 0.0552624  ... 0.64255786 0.2959525  0.49004254]]
```

```
# PART I
def img_generator(img_list):
    for image in img_list:
        x = get_image(image)
        x.shape = (1,299,299,3)
        yield x


enc_train = img_encoder.predict_generator(img_generator(train_list), steps=len(train_list), v
print(enc_train[11])

enc_dev = img_encoder.predict_generator(img_generator(dev_list), steps=len(dev_list), verbose
enc_test = img_encoder.predict_generator(img_generator(test_list), steps=len(test_list), verb
```

```
   7/6000 [..............................] - ETA: 1:55<ipython-input-25-37450bed5cf0>:1
  enc_train = img_encoder.predict_generator(img_generator(train_list), steps=len(train_
6000/6000 [==============================] - 100s 17ms/step
[0.26818597 1.0321672  0.58516276 ... 1.231674   0.17969306 0.22405323]
   9/1000 [..............................] - ETA: 15s<ipython-input-25-37450bed5cf0>:4:
  enc_dev = img_encoder.predict_generator(img_generator(dev_list), steps=len(dev_list),
1000/1000 [==============================] - 16s 16ms/step
   8/1000 [..............................] - ETA: 16s<ipython-input-25-37450bed5cf0>:5:
  enc_test = img_encoder.predict_generator(img_generator(test_list), steps=len(test_lis
1000/1000 [==============================] - 16s 16ms/step
```

```
OUTPUT_PATH = "hw5output"
if not os.path.exists(OUTPUT_PATH):
    os.mkdir(OUTPUT_PATH)

np.save(os.path.join(OUTPUT_PATH,"encoded_images_train.npy"), enc_train)
np.save(os.path.join(OUTPUT_PATH,"encoded_images_dev.npy"), enc_dev)
np.save(os.path.join(OUTPUT_PATH,"encoded_images_test.npy"), enc_test)


#PART II
def read_image_descriptions(filename):
    image_descriptions = defaultdict(list)
    file = open(filename, "r")
    lines = file.read().splitlines()
    file.close()
    captions = list()

    test = 0
    for line in lines:
        split_line = line.split()
        name = split_line[0]

        file_name = name.split('#')[0] #this will be the dict key
        number = name.split('#')[1]
```

```
            tokens = split_line[1:]
            lower_tokens = list() # this will be appended to captions, captions is the dict val
            lower_tokens.append('<START>')
            for t in tokens:
                lower_tokens.append(t.lower())
            lower_tokens.append('<END>')

            captions.append(lower_tokens)

            if number == '4' and len(captions) == 5: #once captions is 5 token lists long we need
                # create dictionary item and reset captions
                image_descriptions[file_name] = captions
                captions = list()

        return image_descriptions

    descriptions = read_image_descriptions(f"{FLICKR_PATH}/Flickr8k.token.txt")
    print(descriptions[dev_list[0]])

    path = os.path.join(FLICKR_PATH, "Flickr_8k.trainImages.txt")
    file = open(path, "r")
    lines = file.read().splitlines()
    file.close()
    tokens = set()
    desc = read_image_descriptions(f"{FLICKR_PATH}/Flickr8k.token.txt")
    for filename in lines:
        caps = desc[filename]
        for caption in caps:
            for tok in caption:
                tokens.add(tok)

    tokens_list = list(tokens)
    tokens_list.sort()

    id_to_word = defaultdict(int)
    for i in range(0, len(tokens_list)):
        id_to_word[i+1] = tokens_list[i]

    word_to_id = defaultdict(str)
    for i in range(0, len(tokens_list)):
        word_to_id[tokens_list[i]] = i+1

    print(word_to_id['dog'])
    print(id_to_word[1985])
```

```
     [['<START>', 'the', 'boy', 'laying', 'face', 'down', 'on', 'a', 'skateboard', 'is', 'bei
     1986
     does
```

```
    #PART III
```

```
print(len(description) for image_id in train_list for description in descriptions[image_id])

MAX_LEN = 40
EMBEDDING_DIM=300
vocab_size = len(word_to_id)+1

# Text input
text_input = Input(shape=(MAX_LEN,))
embedding = Embedding(vocab_size, EMBEDDING_DIM, input_length=MAX_LEN)(text_input)
x = Bidirectional(LSTM(512, return_sequences=False))(embedding)
pred = Dense(vocab_size, activation='softmax')(x)
model = Model(inputs=[text_input],outputs=pred)
model.compile(loss='categorical_crossentropy', optimizer='RMSprop', metrics=['accuracy'])

print(model.summary())

tokens = list()
for image in train_list:
    for x in descriptions[image]:
        for tok in x:
            tokens.append(tok)
print(tokens[:50]) #tokens is now a flattened list of all the tokens from train_list
```

```
<generator object <genexpr> at 0x7fc4a6c46c80>
Model: "model_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_6 (InputLayer)        [(None, 40)]              0

 embedding_3 (Embedding)     (None, 40, 300)           2312400

 bidirectional_3 (Bidirectio  (None, 1024)             3330048
 nal)

 dense_4 (Dense)             (None, 7708)              7900700

=================================================================
Total params: 13,543,148
Trainable params: 13,543,148
Non-trainable params: 0
_________________________________________________________________
None
['<START>', 'a', 'black', 'dog', 'is', 'running', 'after', 'a', 'white', 'dog', 'in', 't
```

```
#PART III

def create_batch(tok_pos, curr_inp, batch_size):
    inp = np.zeros((batch_size, MAX_LEN))
    outp = np.zeros((batch_size, vocab_size))
```

```
        for row in range(0, batch_size):
            if tok_pos == len(tokens):
                tok_pos = 0
            curr_inp.append(word_to_id[tokens[tok_pos]])
            for i in range(0, MAX_LEN):
                if i >= len(curr_inp):
                    inp[row, i] = 0
                else:
                    inp[row, i] = curr_inp[i]
            curr_outp = tokens[tok_pos+1]
            for i in range(0, vocab_size):
                if i == word_to_id[curr_outp]:
                    outp[row, i] = 1
                else:
                    outp[row, i] = 0
            if curr_outp == '<END>':
                tok_pos += 2
                curr_inp.clear()
            else:
                tok_pos+=1

        return (tok_pos, inp, outp, curr_inp)

    def text_training_generator(batch_size=128):
        tok_pos = 0
        curr_inp = list()
        tok_pos, inp, outp, curr_inp = create_batch(tok_pos, curr_inp, batch_size)
        while True:
            yield (inp, outp)


    #PART III CONT.

    batch_size = 128
    generator = text_training_generator(batch_size)
    steps = len(train_list) * MAX_LEN // batch_size

    model.fit_generator(generator, steps_per_epoch=steps, verbose=True, epochs=10)
```

```
    <ipython-input-8-b467f0f3fc71>:5: UserWarning: `Model.fit_generator` is deprecated and w
      model.fit_generator(generator, steps_per_epoch=steps, verbose=True, epochs=10)
    Epoch 1/10
    1875/1875 [==============================] - 488s 255ms/step - loss: 4.2500 - accuracy:
    Epoch 2/10
    1875/1875 [==============================] - 468s 250ms/step - loss: 3.6945 - accuracy:
    Epoch 3/10
    1875/1875 [==============================] - 470s 250ms/step - loss: 3.5171 - accuracy:
    Epoch 4/10
    1875/1875 [==============================] - 468s 250ms/step - loss: 3.4122 - accuracy:
    Epoch 5/10
    1875/1875 [==============================] - 475s 253ms/step - loss: 3.3681 - accuracy:
    Epoch 6/10
    1875/1875 [==============================] - 472s 252ms/step - loss: 3.3050 - accuracy:
```

```
Epoch 7/10
1875/1875 [==============================] - 471s 251ms/step - loss: 3.2695 - accuracy:
Epoch 8/10
1875/1875 [==============================] - 471s 251ms/step - loss: 3.2482 - accuracy:
Epoch 9/10
1875/1875 [==============================] - 474s 253ms/step - loss: 3.2602 - accuracy:
Epoch 10/10
1875/1875 [==============================] - 473s 252ms/step - loss: 3.2695 - accuracy:
<keras.callbacks.History at 0x7fc4a977ca90>
```

```
# PART III CONT.

def decoder():
    seq = np.zeros((1,40))
    seq[0,0] = word_to_id['<START>']
    for i in range(1, MAX_LEN):
        print(seq)
        vocab_dist = model.predict(seq)
        x = np.argmax(vocab_dist)
        print(x)
        seq[0,i] = x
        predicted_word = id_to_word[x]
        print(predicted_word)
        if predicted_word == '<END>':
            break
    return seq

d = decoder()[0]
for w in d:
    print(id_to_word[w])
```

```
[[60.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.]]
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-215-6b2cf6dc9692> in <module>
     16     return seq
     17
---> 18 d = decoder()[0]
     19 for w in d:
     20     print(id_to_word[w])

                                 2 frames
/usr/local/lib/python3.8/dist-packages/keras/engine/training.py in
tf__predict_function(iterator)
     13                 try:
     14                     do_return = True
---> 15                     retval_ = ag__.converted_call(ag__.ld(step_function),
```

```
# PART III CONT.

def sample_decoder():
    seq = np.zeros((1,40))
    seq[0,0] = word_to_id['<START>']
    for i in range(1, MAX_LEN):
        #print(seq)
        vocab_dist = model.predict(seq, verbose = 0)
        multinom_dist = np.random.multinomial(100, vocab_dist[0])

        x = np.argmax(multinom_dist)
        #print(x)
        seq[0,i] = x
        predicted_word = id_to_word[x]
        #print(predicted_word)
        if predicted_word == '<END>':
            break
    return seq

for i in range(10):
    print('Sentence '+str(i)+': ')
    temp = sample_decoder()[0]
    for i in temp:
        w = id_to_word[i]
        if w != '<START>' and  w != '<END>' and w != '0':
          print(w, end = ' ')
    print()
```

```
Sentence 0:
making 30 accents accents discuss structures admiring all-white advertisment antenna rea
Sentence 1:
6 formed goose adorned aerobatic algae litttle accross loose 50 next agile crossing muzz
Sentence 2:
nice piste african-american hooping activities anticipation - cake 1 braids linked snowp
```

```
Sentence 3:
advertising 12 breeds trike adhd acroos ' accordion admire banners 's-eye-view after 10
Sentence 4:
react aboard ac ? 12 11 'n' welcome gates portion 6 abandoned 1950s pees creative adoles
Sentence 5:
paraphernalia accompanies aloft ages fetches 281 's 4-wheel ashen agile direct abandon f
Sentence 6:
( 12 gontaga sinks 19 stumbling backbends racquet 25 tap 8 acoustic spaniel browsing hur
Sentence 7:
wrinkled 75 75 coutryside 25 aluminum burning clad saturated ! 42 - devotion advertiseme
Sentence 8:
# 21 4x4 submissive trampled ' aid aboriginal allowing temple ace aerobics ' 19 attempt
Sentence 9:
dappled ability ac accent automobile cramped cyclist cars 3-wheeler funky 19 communal 's
```

```
#PART 4

MAX_LEN = 40
EMBEDDING_DIM=300
IMAGE_ENC_DIM=300

# Image input
img_input = Input(shape=(2048,))
img_enc = Dense(300, activation="relu") (img_input)
images = RepeatVector(MAX_LEN)(img_enc)

# Text input
text_input = Input(shape=(MAX_LEN,))
embedding = Embedding(vocab_size, EMBEDDING_DIM, input_length=MAX_LEN)(text_input)
x = Concatenate()([images,embedding])
y = Bidirectional(LSTM(256, return_sequences=False))(x)
pred = Dense(vocab_size, activation='softmax')(y)
model2 = Model(inputs=[img_input,text_input],outputs=pred)
model2.compile(loss='categorical_crossentropy', optimizer="RMSProp", metrics=['accuracy'])

print(model2.summary())

enc_train = np.load(f"{OUTPUT_PATH}/encoded_images_train.npy")
enc_dev = np.load(f"{OUTPUT_PATH}/encoded_images_dev.npy")
```

```
Model: "model_7"
__________________________________________________________________________________________
 Layer (type)                Output Shape              Param #     Connected to
==========================================================================================
 input_11 (InputLayer)       [(None, 2048)]            0           []

 dense_9 (Dense)             (None, 300)               614700      ['input_11[0][0]']

 input_12 (InputLayer)       [(None, 40)]              0           []

 repeat_vector_3 (RepeatVector)  (None, 40, 300)       0           ['dense_9[0][0]']
```

```
 embedding_6 (Embedding)        (None, 40, 300)      2312400     ['input_12[0][0]']

 concatenate_5 (Concatenate)    (None, 40, 600)      0           ['repeat_vector_3[0][0]
                                                                  'embedding_6[0][0]']

 bidirectional_6 (Bidirectional  (None, 512)         1755136     ['concatenate_5[0][0]']
 )

 dense_10 (Dense)               (None, 7708)         3954204     ['bidirectional_6[0][0]

==================================================================================================
Total params: 8,636,440
Trainable params: 8,636,440
Non-trainable params: 0
__________________________________________________________________________________________________
None
```

```
#PART 4 CONT.
def create_batch(token, caption, image, curr_inp, batch_size):
    inp = np.zeros((batch_size, MAX_LEN))
    outp = np.zeros((batch_size, vocab_size))
    img_inp = np.zeros((batch_size, 2048))
    image = image%len(train_list)
    img_arr = enc_train[image]
    for row in range(batch_size):
        if token+1 >= len(descriptions[train_list[image]][caption]):
            token = 0
            caption += 1
        if caption >= len(descriptions[train_list[image]]): #should always be a length of 5
            caption = 0
            image += 1
            image = image%len(train_list)
            img_arr = enc_train[image]

        curr_token = descriptions[train_list[image]][caption][token]
        curr_inp.append(curr_token)
        for i in range(len(curr_inp)):
            inp[row, i] = word_to_id[curr_inp[i]]
        for i in range(len(img_arr)):
            img_inp[row, i] = img_arr[i]
        curr_outp = descriptions[train_list[image]][caption][token+1]
        outp[row,word_to_id[curr_outp]] = 1
        if curr_outp == '<END>':
            curr_inp.clear()
        token += 1

    return (token, caption, image, img_inp, inp, outp, curr_inp)

def training_generator(batch_size=128):
    token, caption, image = 0,0,0
    curr_inp = list()
```

```
        while True:
            token, caption, image, img_inp, inp, outp, curr_inp = create_batch(token, caption, im
            yield ([img_inp, inp], outp)


    #PART 4 CONT.
    print(model)
    batch_size = 128
    generator = training_generator(batch_size)
    print(generator)
    steps = len(train_list) * MAX_LEN // batch_size

    print(model2.summary())

    model2.fit(generator, steps_per_epoch=steps, verbose=True, epochs=20)
```

```
<keras.engine.functional.Functional object at 0x7fc4a55b9640>
<generator object training_generator at 0x7fc4a52446d0>
Model: "model_7"
```

Layer (type)	Output Shape	Param #	Connected to
input_11 (InputLayer)	[(None, 2048)]	0	[]
dense_9 (Dense)	(None, 300)	614700	['input_11[0][0]']
input_12 (InputLayer)	[(None, 40)]	0	[]
repeat_vector_3 (RepeatVector)	(None, 40, 300)	0	['dense_9[0][0]']
embedding_6 (Embedding)	(None, 40, 300)	2312400	['input_12[0][0]']
concatenate_5 (Concatenate)	(None, 40, 600)	0	['repeat_vector_3[0][0]', 'embedding_6[0][0]']
bidirectional_6 (Bidirectional)	(None, 512)	1755136	['concatenate_5[0][0]']
dense_10 (Dense)	(None, 7708)	3954204	['bidirectional_6[0][0]

```
Total params: 8,636,440
Trainable params: 8,636,440
Non-trainable params: 0
```

```
#PART 4 CONT.
model2.save_weights(f"{OUTPUT_PATH}/model.h5")
model2.load_weights(f"{OUTPUT_PATH}/model.h5")
```

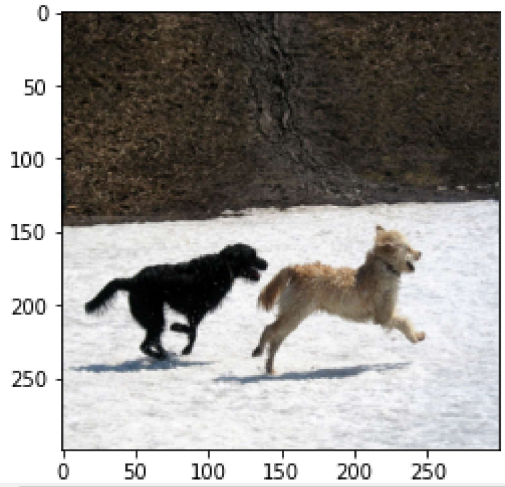
```
1875/1875 [==============================] - 225s 120ms/step - loss: 3.3780 - accuracy:
```

```
#PART 4 CONT.

def image_decoder(enc_image):
    seq = np.zeros((1,40))
    seq[0,0] = word_to_id['<START>']

    img_list = np.empty((1,2048))
    for i in range(2048):
        img_list[0,i] = enc_image[i]

    for i in range(1, MAX_LEN):
        #print(seq)
        #print(enc_image.shape)
        #print(seq.shape)
        vocab_dist = model2.predict([img_list, seq], verbose = False)
        #print('test')
        x = np.argmax(vocab_dist)
        #print(x)
        seq[0,i] = x
```

```
            predicted_word = id_to_word[x]
            #print(predicted_word)
            if predicted_word == '<END>':
                break
        return seq
```

```
# PART 4 CONT.

plt.imshow(get_image(train_list[0]))
x = image_decoder(enc_train[0])[0]

for w in x:
    print(id_to_word[w], end = ' ')
```

```
 <START> a dog running in the snow . <END> 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```




```
# PART 4 CONT.

plt.imshow(get_image(dev_list[1]))
x = image_decoder(enc_dev[1])[0]

for w in x:
    print(id_to_word[w], end = ' ')
```

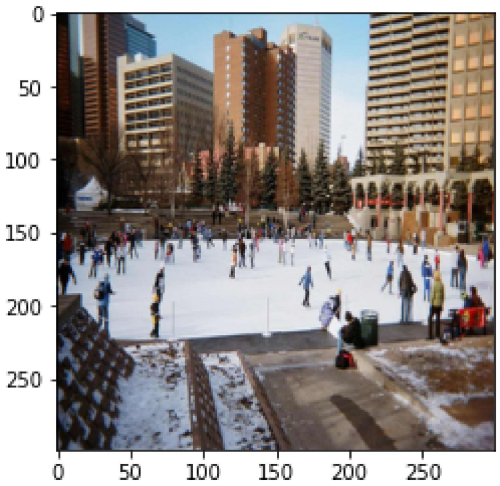
```
 <START> a man in a red shirt is in a red shirt and red and white pants is in a large lar
```

```
#PART 5

def img_beam_decoder(n, image_enc):
    seqs = list() # list of (prob, sequence) tuples
    initial_seq = np.zeros((1,40))
    initial_seq[0,0] = word_to_id['<START>']
    for i in range(n):
        seqs.append((1,initial_seq))

    img_list = np.empty((1,2048))
    for i in range(1,2048):
        img_list[0,i] = image_enc[i]

    for i in range(1, MAX_LEN): #one iteration per new word
        new_seqs = list()
        for prob, seq in seqs:
            vocab_dist = model2.predict([img_list, seq], verbose = False)
            n_best_words = (-vocab_dist[0]).argsort()[:n]

            temp_seq = seq
            for pos in n_best_words:
            #    print(pos)
                w_prob = vocab_dist[0, pos]
                temp_seq[0,i] = pos
                new_seqs.append((w_prob*prob, temp_seq))

        seqs = sorted(new_seqs, reverse = True)[:n]

    return sorted(seqs, reverse = True)[0][1]


# PART 5 CONT.

plt.imshow(get_image(dev_list[1]))
x = img_beam_decoder(3, enc_dev[1])[0]

for w in x:
    print(id_to_word[w], end = ' ')
```

```
<START> on on , blue and jacket blue blue a his jumping front two of red jumping an <END
```

```
#PART 5 CONT.

img_num = 6
plt.imshow(get_image(dev_list[img_num]))

print('Greedy:')
x = image_decoder(enc_dev[img_num])[0]
for w in x:
    print(id_to_word[w], end = ' ')

print('\nBeam with 3:')
y = img_beam_decoder(3, enc_dev[img_num])[0]
for w in y:
    print(id_to_word[w], end = ' ')

print('\nBeam with 5:')
z = img_beam_decoder(5, enc_dev[img_num])[0]
for w in z:
    print(id_to_word[w], end = ' ')
```

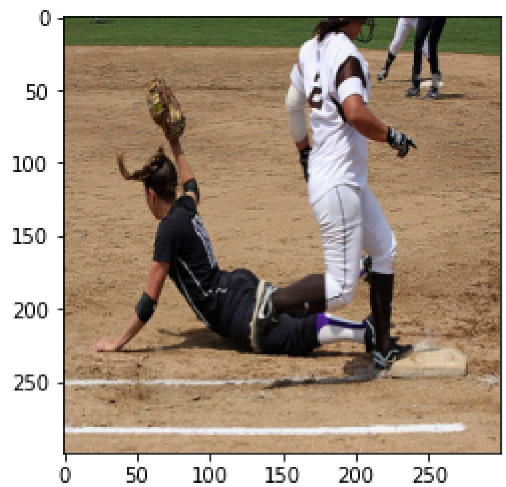
```
Greedy:
<START> a man in a red shirt is standing on a red and white and white dog in the air . <
Beam with 3:
<START> person on white blue ramp is water on two sidewalk in . . <END> in in in with ba
Beam with 5:
<START> red field women outdoor snow fence two with brown , and red an , black in black
```



```
#PART 5 CONT.

img_num = 11
plt.imshow(get_image(dev_list[img_num]))
```

```
print('Greedy:')
x = image_decoder(enc_dev[img_num])[0]
for w in x:
    print(id_to_word[w], end = ' ')

print('\nBeam with 3:')
y = img_beam_decoder(3, enc_dev[img_num])[0]
for w in y:
    print(id_to_word[w], end = ' ')

print('\nBeam with 5:')
z = img_beam_decoder(5, enc_dev[img_num])[0]
for w in z:
    print(id_to_word[w], end = ' ')
```

```
Greedy:
<START> a man in a red shirt is in a field . <END> 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 (
Beam with 3:
<START> on riding riding ground with the with in jacket . the a woman and an girls on <E
Beam with 5:
<START> and on their front two person scooter an red at three dogs is . are people by th
```



```
#PART 5 CONT.

img_num = 25
plt.imshow(get_image(dev_list[img_num]))

print('Greedy:')
x = image_decoder(enc_dev[img_num])[0]
for w in x:
    print(id_to_word[w], end = ' ')

print('\nBeam with 3:')
y = img_beam_decoder(3, enc_dev[img_num])[0]
for w in y:
```

```
    print(id_to_word[w], end = ' ')

print('\nBeam with 5:')
z = img_beam_decoder(5, enc_dev[img_num])[0]
for w in z:
    print(id_to_word[w], end = ' ')
```

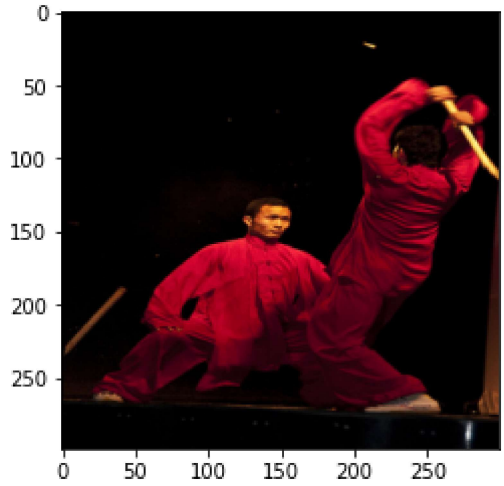
```
 Greedy:
 <START> a black dog is running in a field . <END> 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 Beam with 3:
 <START> black are dogs grass at red with open his and in red stick his on <END> grass by
 Beam with 5:
 <START> brown black in grass and field <END> dog and out blue and and red with yard toy
```


```
#PART 5 CONT.

img_num = 80
plt.imshow(get_image(dev_list[img_num]))

print('Greedy:')
x = image_decoder(enc_dev[img_num])[0]
for w in x:
    print(id_to_word[w], end = ' ')

print('\nBeam with 3:')
y = img_beam_decoder(3, enc_dev[img_num])[0]
for w in y:
    print(id_to_word[w], end = ' ')

print('\nBeam with 5:')
z = img_beam_decoder(5, enc_dev[img_num])[0]
for w in z:
    print(id_to_word[w], end = ' ')
```

```
Greedy:
<START> a dog is running in a field . <END> 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Beam with 3:
<START> brown in , black on in with open <END> red in brown with brown brown brown brown
Beam with 5:
<START> brown play water in carrying toy . white water the the a he his on dog black a t
```


```
#PART 5 CONT.

img_num = 30
plt.imshow(get_image(dev_list[img_num]))

print('Greedy:')
x = image_decoder(enc_dev[img_num])[0]
for w in x:
    print(id_to_word[w], end = ' ')

print('\nBeam with 3:')
y = img_beam_decoder(3, enc_dev[img_num])[0]
for w in y:
    print(id_to_word[w], end = ' ')

print('\nBeam with 5:')
z = img_beam_decoder(5, enc_dev[img_num])[0]
for w in z:
    print(id_to_word[w], end = ' ')
```

```
Greedy:
<START> a man in a red shirt is jumping on a metal fence . <END> 0 0 0 0 0 0 0 0 0 0 0 0
Beam with 3:
<START> on , woman with . in background with red the <END> the <END> his in a the backgr
Beam with 5:
<START> group in in area playing in in the yellow yellow jacket blue on blue and black c
```



Colab paid products - Cancel contracts here

0s completed at 11:56 PM