

Assignment 2 Problem Set 3

DL Group 193

Authors:

1. JITENDRA PAL SINGH - 2019AP04008
2. ASHISH SHARMA - 2018AD04053

Pre-Requisites

1.

Visit the link [<https://drive.google.com/file/d/1-mPKMpphaKqtT26ZzbR5hCHGedkNyAf1/view?usp=sharing>]

Go to drive.google.com

Under 'Shared with me', find the file named **Image_Captioning_Dataset.zip**

Right click -> Add Shortcut to Drive

Select 'My Drive'

Click 'Add Shortcut'

2. Visit the link [https://drive.google.com/file/d/1QG2_tAFLroc4ATSOLRiEAwYTYTsL_xVc/view?usp=sharing]

Go to drive.google.com

Under 'Shared with me', find the file named **set_3.pkl**

Right click -> Add Shortcut to Drive

Select 'My Drive'

Click 'Add Shortcut'

Upload a file to gdrive/MyDrive

rename it as: **google-image.jpg**

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

import pickle
from google.colab import drive
import re
from nltk.tokenize import RegexpTokenizer
import nltk
```

```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from itertools import islice
from collections import defaultdict
import seaborn as sns
import matplotlib.pyplot as plt
import zipfile
import os

from operator import itemgetter
import matplotlib.pyplot as plt
import pandas as pd

from keras.preprocessing.image import load_img
from IPython.display import Image, display
import math
from textwrap import wrap

import numpy as np
from keras.applications.vgg16 import VGG16
from keras.utils.vis_utils import plot_model
from tensorflow.keras.utils import to_categorical

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 keras.layers.merge import add
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau

from pickle import dump
from keras.preprocessing.image import img_to_array
from keras.applications.vgg16 import preprocess_input
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
True
```

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

Drive already mounted at /content/gdrive; to attempt to forcibly remount, call

root_path = '{}/MyDrive'.format(drive_path)

pickle_path = '{}/set_3.pkl'.format(root_path)

dataset_zip_path = '{}/Image_captioning_Dataset.zip'.format(root_path)
dataset_path = '{}/Flicker8k_Dataset'.format(root_path)

glove_zip_name = 'glove.6B.zip'
```

```

glove_zip_path = '{} / {}'.format(root_path, glove_zip_name)
glove_dataset_file = '{} / glove.6B.200d.txt'.format(root_path)

# download and copy/move glove dataset to drive
if not os.path.exists(glove_zip_path):
    print("downloading glove dataset, please wait for some time...")
    !wget 'https://nlp.stanford.edu/data/glove.6B.zip'

print("uploading glove dataset to drive, this may take a few minutes...")
!mv './' + glove_zip_name} {glove_zip_path}
with zipfile.ZipFile(glove_zip_path, "r") as zip_ref:
    zip_ref.extractall(os.path.dirname(glove_zip_path))

# extract the dataset
if not os.path.exists(dataset_path):
    print("extracting image_captioning dataset, this may take a couple of minutes...")
    with zipfile.ZipFile(dataset_zip_path, "r") as zip_ref:
        zip_ref.extractall(os.path.dirname(dataset_zip_path))
else:
    print("dataset extracted already!")

    dataset extracted already!

# read the pickle dataset
objects = []
with open(pickle_path, "rb") as openfile:
    while True:
        try:
            objects.append(pickle.load(openfile))
        except EOFError:
            break
print(len(objects))
objects[0][:10]

1
['3033668641_5905f73990.jpg#0\tA man in a white helmet cling to a sheer rock',
 '2608289957_044849f73e.jpg#0\tA baby sit on a tire and hold a toy .',
 '2723477522_d89f5ac62b.jpg#4\ttwo dog play in the grass .',
 '3453019315_cfd5c10dae.jpg#2\tA man do a bike trick on a dirt path .',
 '3524975665_7bec41578b.jpg#1\tA woman in a yellow jacket and brown hat stand',
 '3461041826_0e24cdf597.jpg#2\tA black and white dog be splash in a stream .'
 '414568315_5adcfc23c0.jpg#4\tA child be use chopstick on overturned kitchen',
 '2657484284_daa07a3a1b.jpg#0\tA kayak on a river be pass by a wire bridge th',
 '2860667542_95abec3380.jpg#2\tA white water bird take flight from a lake .',
 '2222498879_9e82a100ab.jpg#3\tA tan and black dog walk happy through a field']

# convert to required format, remove punctuations, lowercase the captions.

captions_dict = defaultdict(list)
tokenizer = RegexpTokenizer(r'\w+')

for record in objects[0]:

```

```
(name, caption_num, caption) = re.split(r'#\|\t', record, maxsplit=2)

# do not add it to captions_dict (or training/testing sets if we don't have a file)
if not os.path.exists(os.path.join(dataset_path, name)):
    continue

# remove stopwords, convert to lowercase, remove punctuations.
caption_words = tokenizer.tokenize(caption.lower())
captions_dict[name].append(' '.join(caption_words))

dict(islice(captions_dict.items(), 5))

{'2608289957_044849f73e.jpg': ['a baby sit on a tire and hold a toy',
    'boy in pajama with red and yellow toy seat on tire near plant',
    'a little kid with a red toy be sit on a tire',
    'child hold plastic car sit on a tire'],
 '2723477522_d89f5ac62b.jpg': ['two dog play in the grass',
    'a large brown dog be chase after a little brown dog',
    'a big dog chase a little dog on the grass',
    'two brown dog run through the grass together'],
 '3033668641_5905f73990.jpg': ['a man in a white helmet cling to a sheer rock',
    'a man with all his equipment rock climb',
    'a rock climber wear a white helmet'],
 '3453019315_cfd5c10dae.jpg': ['a man do a bike trick on a dirt path',
    'man perform a trick on a bicycle outside',
    'a helmeted male airborne on a bike on a dirt road'],
 '3524975665_7bec41578b.jpg': ['a woman in a yellow jacket and brown hat stand',
    'a young woman wait under a sculpture in a park',
    'a woman in a yellow coat be stand on a sidewalk']}
```

```
# display N sample images and their captions.
```

```
def display_images(dataset_path, filenames, captions_dict, title=True):

    N = len(filenames)
    ncols = 2    # number of columns in the grid

    # create a grid.
    fig, axes = plt.subplots(nrows=math.ceil(N/ncols), ncols=ncols, figsize=(24,24),
                           sharex=True, sharey=True)

    # adjust grid spacing to accomodate multiple captions.
    fig.subplots_adjust(left=None, bottom=None, right=None, top=1.5, wspace=None, hspace=None)

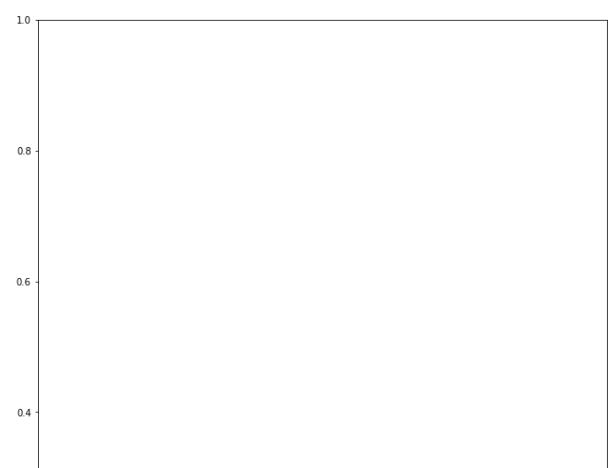
    count = 0
    for filename in filenames:
        (i, j) = (count // ncols, count % ncols)    # grid i, j coordinates
        image = load_img(os.path.join(dataset_path, filename))
        axes[i, j].imshow(image)

        # number the title, wrap long titles (to default 70 char width)
        if title:
            title = "\n".join(["{}.\n{}".format(i+1, "\n".join(wrap(v))) for i, v in enumerate(captions_dict[filename])])
            axes[i, j].set_title(title)
```

```
axes[i, j].title.set_text(title)

count += 1
if count >= N:
    break

display_images(dataset_path, list(captions_dict.keys())[:5], captions_dict)
```





```
# split captions data in training/test sets

train_ratio = 0.9

image_names = [x for x in captions_dict.keys()]

# NOTE:
# with shuffle = False, The code shall extract image features on the first run, and
# save it to disk for training and testing datasets, subsequent runs shall re-use t
#
# shuffle=True will re-extract training/test image features every time and will NOT
# save it to disk.
shuffle = False
if shuffle:
    np.random.shuffle(image_names)

train_size = int(len(image_names) * train_ratio)

train_data = dict({k: captions_dict[k] for k in image_names[:train_size]})
test_data = dict({k: captions_dict[k] for k in image_names[train_size:]})

print("sample captions from training dataset\n-----")
print(dict(islice(train_data.items(), 5)), "\n\n")
print("sample captions from testing dataset\n-----")
print(dict(islice(test_data.items(), 5)), "\n\n")
'size - training dataset: {}, testing dataset: {}'.format(len(train_data), len(test_data))

sample captions from training dataset
-----
{'3033668641_5905f73990.jpg': ['a man in a white helmet cling to a sheer rock wall'],
 '3471841031_a949645ba8.jpg': ['people at park sit on bench and a woman take photo'],
 '3471841031_a949645ba9.jpg': ['a man in a white shirt and blue jeans sitting on a bench'],
 '3471841031_a949645baa.jpg': ['a man in a white shirt and blue jeans sitting on a bench'],
 '3471841031_a949645bab.jpg': ['a man in a white shirt and blue jeans sitting on a bench']}

sample captions from testing dataset
-----
{'3471841031_a949645ba8.jpg': ['people at park sit on bench and a woman take photo'],
 '3471841031_a949645ba9.jpg': ['a man in a white shirt and blue jeans sitting on a bench'],
 '3471841031_a949645baa.jpg': ['a man in a white shirt and blue jeans sitting on a bench'],
 '3471841031_a949645bab.jpg': ['a man in a white shirt and blue jeans sitting on a bench'],
 '3471841031_a949645bab0.jpg': ['a man in a white shirt and blue jeans sitting on a bench']}

'size - training dataset: 7214, testing dataset: 802'

# update training and test data captions with BOS/EOS markers
for key, captions in train_data.items():
    new_captions = ['startseq {} endseq'.format(caption) for caption in captions]
    train_data[key] = new_captions

for key, captions in test_data.items():
    new_captions = ['startseq {} endseq'.format(caption) for caption in captions]
    test_data[key] = new_captions
```

```
print("sample captions from training dataset\n-----")
print(dict(islice(train_data.items(), 5)), "\n\n")
print("sample captions from testing dataset\n-----")
print(dict(islice(test_data.items(), 5)), "\n\n")
'size - training dataset: {}, testing dataset: {}'.format(len(train_data), len(test_data))

sample captions from training dataset
-----
{'3033668641_5905f73990.jpg': ['startseq a man in a white helmet cling to a s']

sample captions from testing dataset
-----
{'3471841031_a949645ba8.jpg': ['startseq people at park sit on bench and a wo']

'size - training dataset: 7214, testing dataset: 802'

# create a word count dictionary.

vocabulary = set()
word_count = defaultdict(lambda: 0)
for captions in train_data.values():
    for caption in captions:
        for word in str.split(caption, ' '):
            vocabulary.add(word)
            if word not in ['startseq', 'endseq']:
                word_count[word] += 1

# sort the word count dict by values.
word_count = dict(sorted(word_count.items(), key = itemgetter(1), reverse = True))

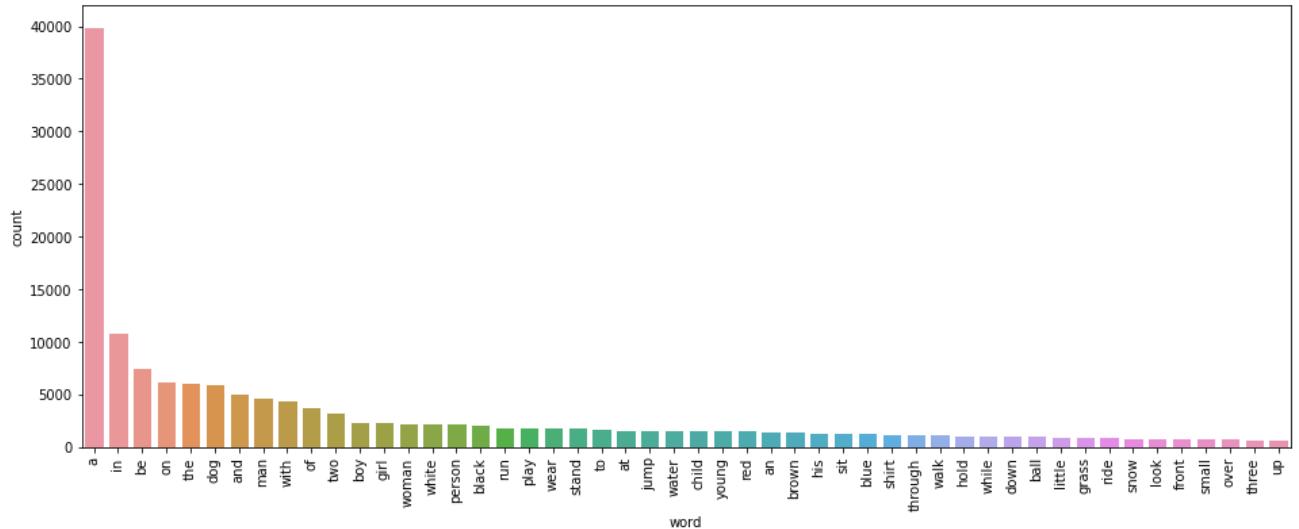
# total vocab size
print(len(vocabulary))

5142

# plot top 50 words from word_count dict.
plt.figure(figsize=(16,6))
plt.xticks(rotation='vertical')

df = pd.DataFrame(word_count.items(), columns=['word', 'count'])
sns.set(style="whitegrid")
sns.barplot(x='word', y='count', data=df[:50])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc140875f10>
```



```
# caption words (min, max, avg.)
```

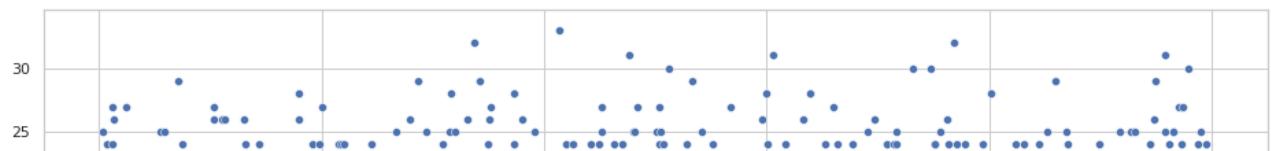
```
caption_sizes = {}
i=0
for captions in captions_dict.values():
    for caption in captions:
        caption_sizes[i] = len(str.split(caption, ' '))
        i += 1

df1 = pd.DataFrame(caption_sizes.items(), columns=['index', 'caption_size'])

plt.figure(figsize=(16,6))

sns.set(style="whitegrid")
sns.scatterplot(x='index', y='caption_size', data=df1)
df1['caption_size'].describe()
```

```
count      24995.000000
mean       10.775155
std        3.773125
min        1.000000
25%        8.000000
50%        10.000000
75%        13.000000
max        33.000000
Name: caption_size, dtype: float64
```



```
# use the VGG16 pretrained model to extract features from images.
model = VGG16(weights='imagenet')
print(model.summary())
plot_model(model, to_file='{}/vgg.png'.format(root_path))
```

```
Model: "vgg16"
```

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 224, 224, 3)]	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168

```
block3_conv3 (Conv2D) (None, 56, 56, 256) 590080
```

```
# recreate model removing last prediction layer
model1 = Model(inputs=model.inputs, outputs=model.layers[-2].output)
print(model1.summary())
plot_model(model1, to_file='{}/vgg-1.png'.format(root_path))
```

```
Model: "model"
```

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 224, 224, 3)]	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160

The model above takes images as **224 x 224 x 3** size.

```
block4_conv3 (Conv2D) (None, 28, 28, 512) 2359808
```

```
from numpy import expand_dims
```

```
# use above model to extract features from images.
def extract_image_features(model, data, dataset_path):
    image_features = dict()
    for filename in data.keys():

        filepath = os.path.join(dataset_path, filename)
        if not os.path.exists(filepath):
            continue
```

```
        image = load_img(filepath, target_size=(224, 224))
```

```
# reshape the image
```

```
image = img_to_array(image)
```

```
image = expand_dims(image, axis=0)
```

```
# preprocess
```

```
image = preprocess_input(image)
```

```
# extract features
```

```
image_features[filename] = model1.predict(image, verbose=0)
```

```
return image_features
```

```
train_data_dump_path = '{}/image_features_train-set.pkl'.format(root_path),
test_data_dump_path = '{}/image_features_test-set.pkl'.format(root_path)
```

```
if shuffle or not os.path.exists(train_data_dump_path):
    print("extracting image features from training data.. please wait for 5-10 minutes")
    image_features_train = extract_image_features(model, train_data, dataset_path)

# No point in saving shuffled image features to disk, as next shuffle won't have
# the same set of images in training or test dataset.
if not shuffle:
    with open(train_data_dump_path, "wb") as fh:
        dump(image_features_train, fh)

if shuffle or not os.path.exists(test_data_dump_path):
    print("extracting image features from test data.. please wait for 2-3 minutes.")

image_features_test = extract_image_features(model, test_data, dataset_path)
# No point in saving shuffled image features to disk, as next shuffle won't have
# the same set of images in training or test dataset.
if not shuffle:
    with open(test_data_dump_path, "wb") as fh:
        dump(image_features_test, fh)
```

▼

```
# load image features from test dump file, display samples.
```

```
print("loading training image features from pickle file")
with open(test_data_dump_path, "rb") as fh:
    image_features_test = pickle.load(fh)
```

```
print('feature set size: {}'.format(len(image_features_test)))
```

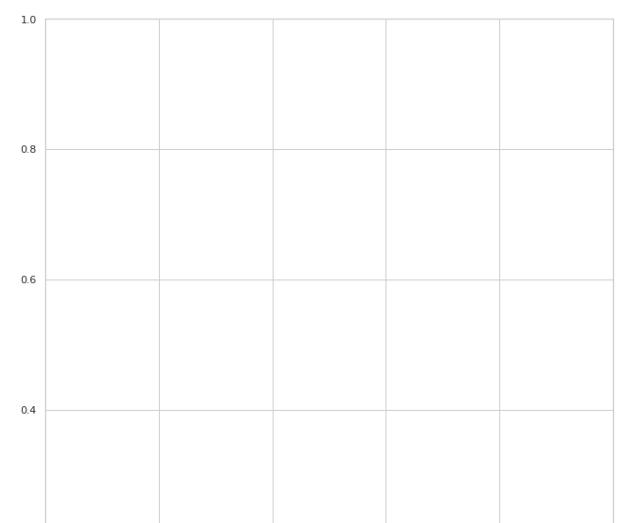
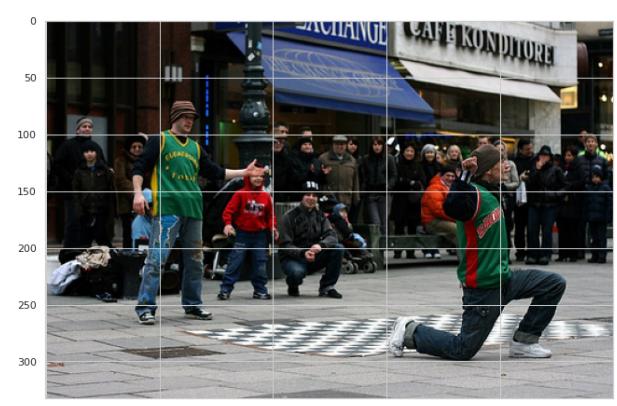
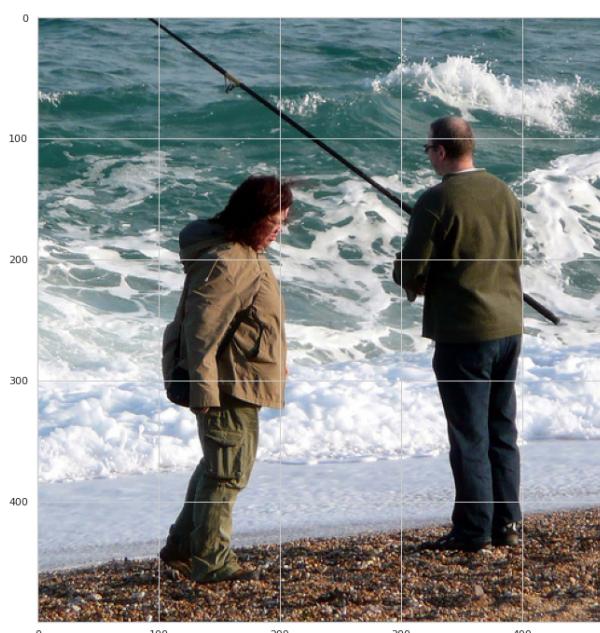
```
# print 5 items
dict(islice(image_features_test.items(), 5))
```

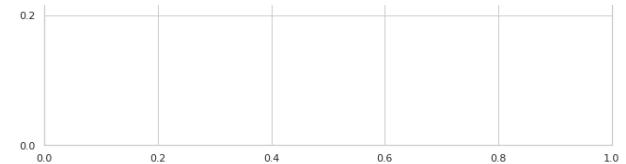
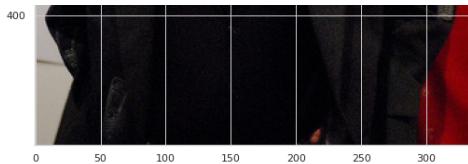
```
loading training image features from pickle file
feature set size: 802
{'1499495021_d295ce577c.jpg': array([[0.42266178, 0.           , 0.06778109, ...
0.886196   ]], dtype=float32),
'2167644298_100ca79f54.jpg': array([[1.9898707, 1.854141  , 0.           , ..., 0
2.3647237]], dtype=float32),
'3228517564_74b00a923b.jpg': array([[0.           , 0.           , 0.           , ...
2.6010358  ]], dtype=float32),
'3259992638_0612a40288.jpg': array([[0.           , 0.           , 0.           , ..., 0
0.5030978]], dtype=float32),
'3471841031_a949645ba8.jpg': array([[1.4077761, 1.5905311, 1.8254167, ..., 0
0.           ]], dtype=float32)}
```

↓

```
# display some test data, before generating captions for it.
```

```
display_images(dataset_path, list(test_data.keys())[:5], test_data, title=False)
```





```
# also display extaracted features for the above images.
def display_features(filename, layer=1):

    features = 64
    ncols = 8      # number of columns in the grid
    nrows = features//ncols

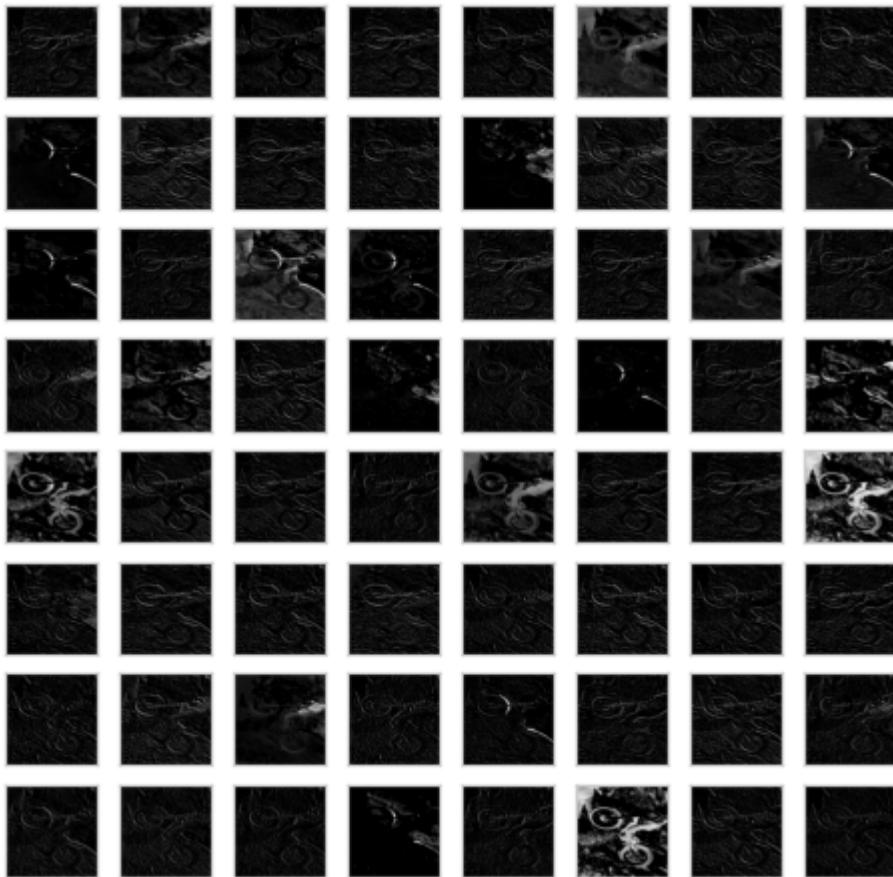
    layer_op = Model(inputs=model.inputs, outputs=model.layers[layer].output)
```

```
# create a grid.
fig, axes = plt.subplots(nrows=nrows, ncols=ncols, figsize=(8,8), squeeze=False)

image = load_img(os.path.join(dataset_path, filename), target_size=(224, 224))
image = img_to_array(image)
image = expand_dims(image, axis=0)
image = preprocess_input(image)

feature_map = layer_op.predict(image)
for fmap in feature_map:
    count = 0
    for r in range(nrows):
        for c in range(ncols):
            axes[r, c].set_xticks([])
            axes[r, c].set_yticks([])
            axes[r, c].imshow(fmap[ :, :, count], cmap='gray')
            count += 1

# displaying features (produced at layer 1) from the biker photo above (index 1).
photo_index = 1
display_features(list(test_data.keys())[photo_index], layer=1)
```



Let's generate captions for the images above.

```
# interpret the output of prediction layer.
from keras.applications.vgg16 import decode_predictions
```

```



```

```
embedding_matrix[i] = embedding_vector
```

```
from tensorflow.keras import regularizers
from tensorflow.keras.optimizers import Adam

# image feature extractor model
inputs1 = Input(shape=(4096,))
fe1 = Dropout(0.3)(inputs1)
fe2 = Dense(256, activation='relu')(fe1)

# partial caption sequence model
inputs2 = Input(shape=(max_caption_len,))
se1 = Embedding(vocab_len, output_dim=200, mask_zero=True)(inputs2)
se2 = Dropout(0.3)(se1)
# stacking multiple GRU layers
se3 = GRU(256, kernel_regularizer=regularizers.l2(0.01))(se2)

# decoder (feed forward) model
decoder1 = add([fe2, se3])
decoder2 = Dense(256, activation='relu')(decoder1)
outputs = Dense(vocab_len, activation='softmax')(decoder2)

# merge the two input models
model2 = Model(inputs=[inputs1, inputs2], outputs=outputs)
print(model2.layers)

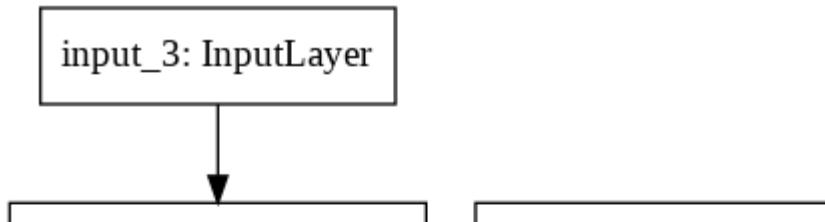
model2.layers[2].set_weights([embedding_matrix])
model2.layers[2].trainable = False
optimizer = Adam(learning_rate=0.001)

model2.compile(loss='categorical_crossentropy', optimizer=optimizer)
model2.summary()
plot_model(model2)
```

```
[<keras.engine.input_layer.InputLayer object at 0x7fc057bfce50>, <keras.engine.Model: "model_2"
```

Layer (type)	Output Shape	Param #	Connected to
input_3 (InputLayer)	[(None, 33)]	0	
input_2 (InputLayer)	[(None, 4096)]	0	
embedding (Embedding)	(None, 33, 200)	1028600	input_3[0][0]
dropout (Dropout)	(None, 4096)	0	input_2[0][0]
dropout_1 (Dropout)	(None, 33, 200)	0	embedding[0]
dense (Dense)	(None, 256)	1048832	dropout[0][0]
gru (GRU)	(None, 256)	351744	dropout_1[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, 5143)	1321751	dense_1[0][0]

Total params: 3,816,719
Trainable params: 2,788,119
Non-trainable params: 1,028,600



```

def data_gen(captions, features, word_index, max_caption_len, vocab_len, num_photos_per_desc):
    X1, X2, y = list(), list(), list()
    # word_index = tokenizer.word_index
    n=0
    # loop for ever over images
    while 1:
        for key, desc_list in captions.items():
            n+=1
            # retrieve the photo feature
            photo = features[key]
            for desc in desc_list:
                # encode the sequence
                seq = [word_index[word] for word in desc.split(' ') if word in word_index]
                # split one sequence into multiple x, y pairs
                for i in range(1, len(seq)):
                    # split into input and output pair
                    in_seq, out_seq = seq[:i], seq[i]
                    # pad input sequence
                    in_seq = pad_sequences([in_seq], maxlen=max_caption_len)[0]
                    # encode output sequence
                    out_seq = to_categorical([out_seq], num_classes=vocab_len)[0]

```

```

# store
X1.append(photo[0])
X2.append(in_seq)
y.append(out_seq)
# yield the batch data
if n==num_photos_per_batch:
    yield [np.array(X1), np.array(X2)], np.array(y) # [[np.array(X1), np.array(
X1, X2, y = list(), list(), list()
n=0

# train the model
import tensorflow as tf
tf.config.run_functions_eagerly(True)

print("loading training image features from pickle file")
with open(train_data_dump_path, "rb") as fh:
    image_features_train = pickle.load(fh)

epochs = 5
batch_size = 64
steps = len(train_data) // batch_size

model_save_dir = '{}/group-193'.format(root_path)
os.makedirs(model_save_dir, exist_ok=True)

histories = []
for i in range(epochs):
    print('epoch {} of {}'.format(i+1, epochs))
    gen = data_gen(train_data, image_features_train, tokenizer.word_index, max_captions)
    history = model2.fit(gen, epochs=1, steps_per_epoch=steps, batch_size=batch_size,
    model2.save('{} /model.{}'.format(model_save_dir, i)))
    histories.append(history)

# load the last saved model results.
recent_model_path = '{} /model.{}'.format(model_save_dir, epochs-1)
recent_model = load_model(recent_model_path)

loading training image features from pickle file
epoch 1 of 5
/usr/local/lib/python3.7/dist-packages/tensorflow/python/data/ops/dataset_ops.py:497: CustomMaskWarning: Even though the `tf.config.experimental_run_functions_eagerly` is set to True, the operation `tf.data.Dataset.map` will still be traced. This is because the function `tf.function` is used to wrap the mapping function. To avoid tracing, you can either use a standard Python function instead of a TensorFlow function or use the `tf.function` decorator with the `experimental_relax_shapes=True` argument.
WARNING:absl:Found untraced functions such as gru_cell_layer_call_fn, gru_cell_layer_call_fn
/usr/local/lib/python3.7/dist-packages/keras/utils/generic_utils.py:497: CustomMaskWarning: Even though the `tf.config.experimental_run_functions_eagerly` is set to True, the operation `tf.data.Dataset.map` will still be traced. This is because the function `tf.function` is used to wrap the mapping function. To avoid tracing, you can either use a standard Python function instead of a TensorFlow function or use the `tf.function` decorator with the `experimental_relax_shapes=True` argument.
INFO:tensorflow:Assets written to: /content/gdrive/MyDrive/group-193/model.0/
INFO:tensorflow:Assets written to: /content/gdrive/MyDrive/group-193/model.0/
epoch 2 of 5
/usr/local/lib/python3.7/dist-packages/tensorflow/python/data/ops/dataset_ops.py:497: CustomMaskWarning: Even though the `tf.config.experimental_run_functions_eagerly` is set to True, the operation `tf.data.Dataset.map` will still be traced. This is because the function `tf.function` is used to wrap the mapping function. To avoid tracing, you can either use a standard Python function instead of a TensorFlow function or use the `tf.function` decorator with the `experimental_relax_shapes=True` argument.
WARNING:absl:Found untraced functions such as gru_cell_layer_call_fn, gru_cell_layer_call_fn
/usr/local/lib/python3.7/dist-packages/keras/utils/generic_utils.py:497: CustomMaskWarning: Even though the `tf.config.experimental_run_functions_eagerly` is set to True, the operation `tf.data.Dataset.map` will still be traced. This is because the function `tf.function` is used to wrap the mapping function. To avoid tracing, you can either use a standard Python function instead of a TensorFlow function or use the `tf.function` decorator with the `experimental_relax_shapes=True` argument.

```

```

INFO:tensorflow:Assets written to: /content/gdrive/MyDrive/group-193/model.1/
INFO:tensorflow:Assets written to: /content/gdrive/MyDrive/group-193/model.1/
epoch 3 of 5
/usr/local/lib/python3.7/dist-packages/tensorflow/python/data/ops/dataset_ops
    "Even though the `tf.config.experimental_run_functions_eagerly`"
WARNING:absl:Found untraced functions such as gru_cell_layer_call_fn, gru_ce
/usr/local/lib/python3.7/dist-packages/keras/utils/generic_utils.py:497: Cust
    category=CustomMaskWarning)
INFO:tensorflow:Assets written to: /content/gdrive/MyDrive/group-193/model.2/
INFO:tensorflow:Assets written to: /content/gdrive/MyDrive/group-193/model.2/
epoch 4 of 5
/usr/local/lib/python3.7/dist-packages/tensorflow/python/data/ops/dataset_ops
    "Even though the `tf.config.experimental_run_functions_eagerly`"
WARNING:absl:Found untraced functions such as gru_cell_layer_call_fn, gru_ce
/usr/local/lib/python3.7/dist-packages/keras/utils/generic_utils.py:497: Cust
    category=CustomMaskWarning)
INFO:tensorflow:Assets written to: /content/gdrive/MyDrive/group-193/model.3/
INFO:tensorflow:Assets written to: /content/gdrive/MyDrive/group-193/model.3/
epoch 5 of 5
/usr/local/lib/python3.7/dist-packages/tensorflow/python/data/ops/dataset_ops
    "Even though the `tf.config.experimental_run_functions_eagerly`"
WARNING:absl:Found untraced functions such as gru_cell_layer_call_fn, gru_ce
/usr/local/lib/python3.7/dist-packages/keras/utils/generic_utils.py:497: Cust
    category=CustomMaskWarning)
INFO:tensorflow:Assets written to: /content/gdrive/MyDrive/group-193/model.4/
INFO:tensorflow:Assets written to: /content/gdrive/MyDrive/group-193/model.4/

```

```

from tensorflow.keras.models import load_model

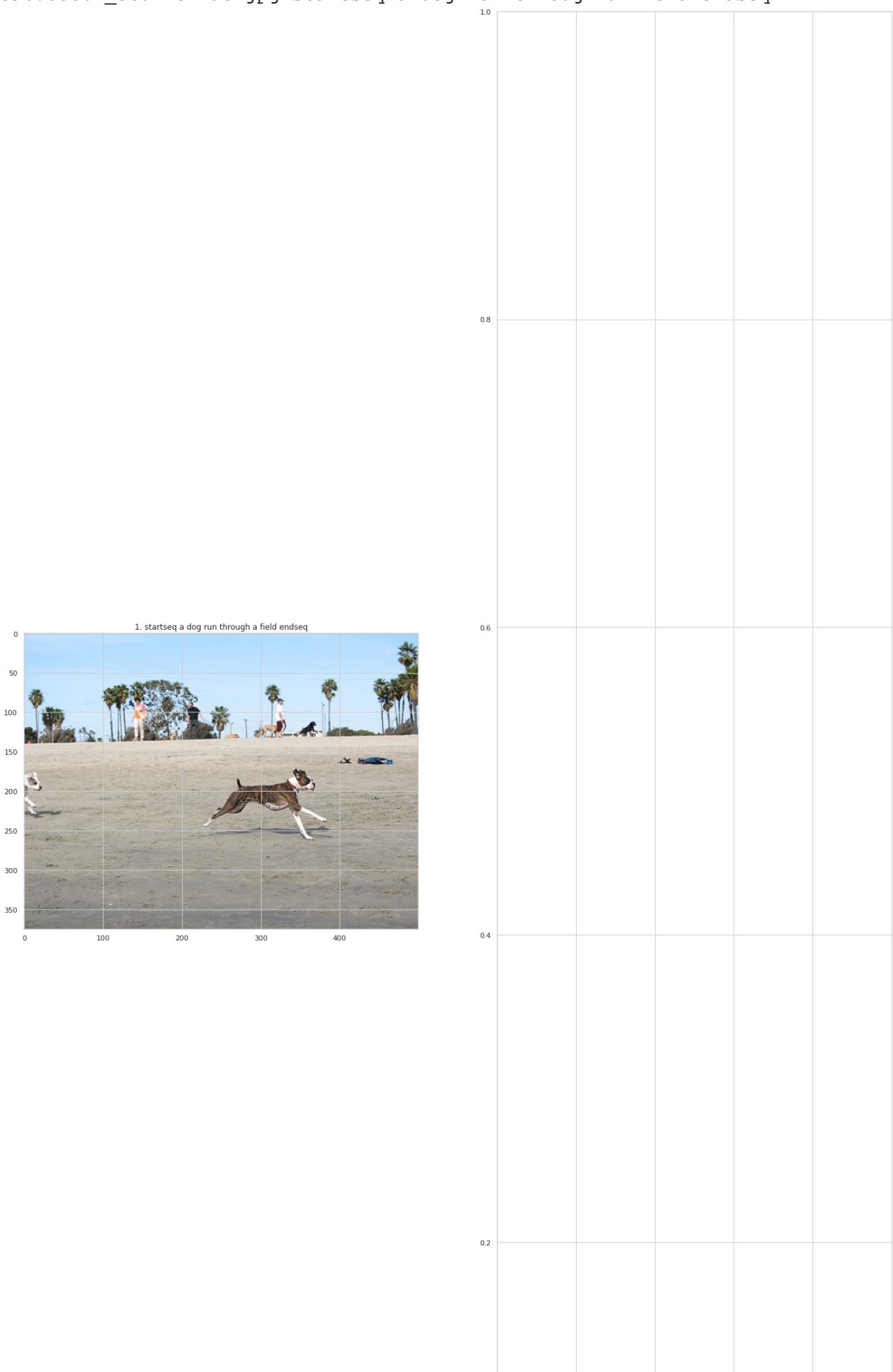
# generate captions.
def greedySearch(rnn_model, photo):
    in_text = 'startseq'
    for i in range(max_caption_len):
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        sequence = pad_sequences([sequence], maxlen=max_caption_len, padding='post')
        yhat = rnn_model.predict([photo, sequence], verbose=0)
        yhat = np.argmax(yhat)
        word = None
        if tokenizer.index_word.get(yhat):
            word = tokenizer.index_word[yhat]
            in_text += ' ' + word
        else:
            break
        if word == 'endseq':
            break
    return in_text

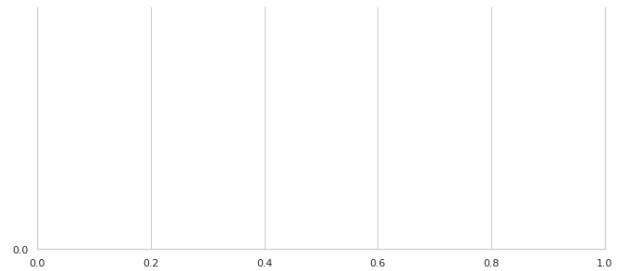
index = 101
key = list(image_features_test.keys())[index]
value = image_features_test[key]

label = greedySearch(recent_model, value)
print(key, label)
display_images(dataset_path, [key], {key: [label]}, title=True)

```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/data/ops/dataset_ops
  "Even though the `tf.config.experimental_run_functions_eagerly` "
396763804_3b7f1e12a8.jpg startseq a dog run through a field endseq
```





```
# load an image from google
```

```
google_image_name = 'google-image.jpg'
```

```
google image path = '{}/{}'.format(root_path, google_image_name)
```

```
https://colab.research.google.com/drive/1FMkb-WzHiJN9s31YnAWLSkms8zKiVIYm#scrollTo=qYq4pq-Y1Tla&printMode=true
```

```
if os.path.exists(google_image_path):
    image = load_img(google_image_path, target_size=(224, 224))
    image = img_to_array(image)
    image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
    image = preprocess_input(image)

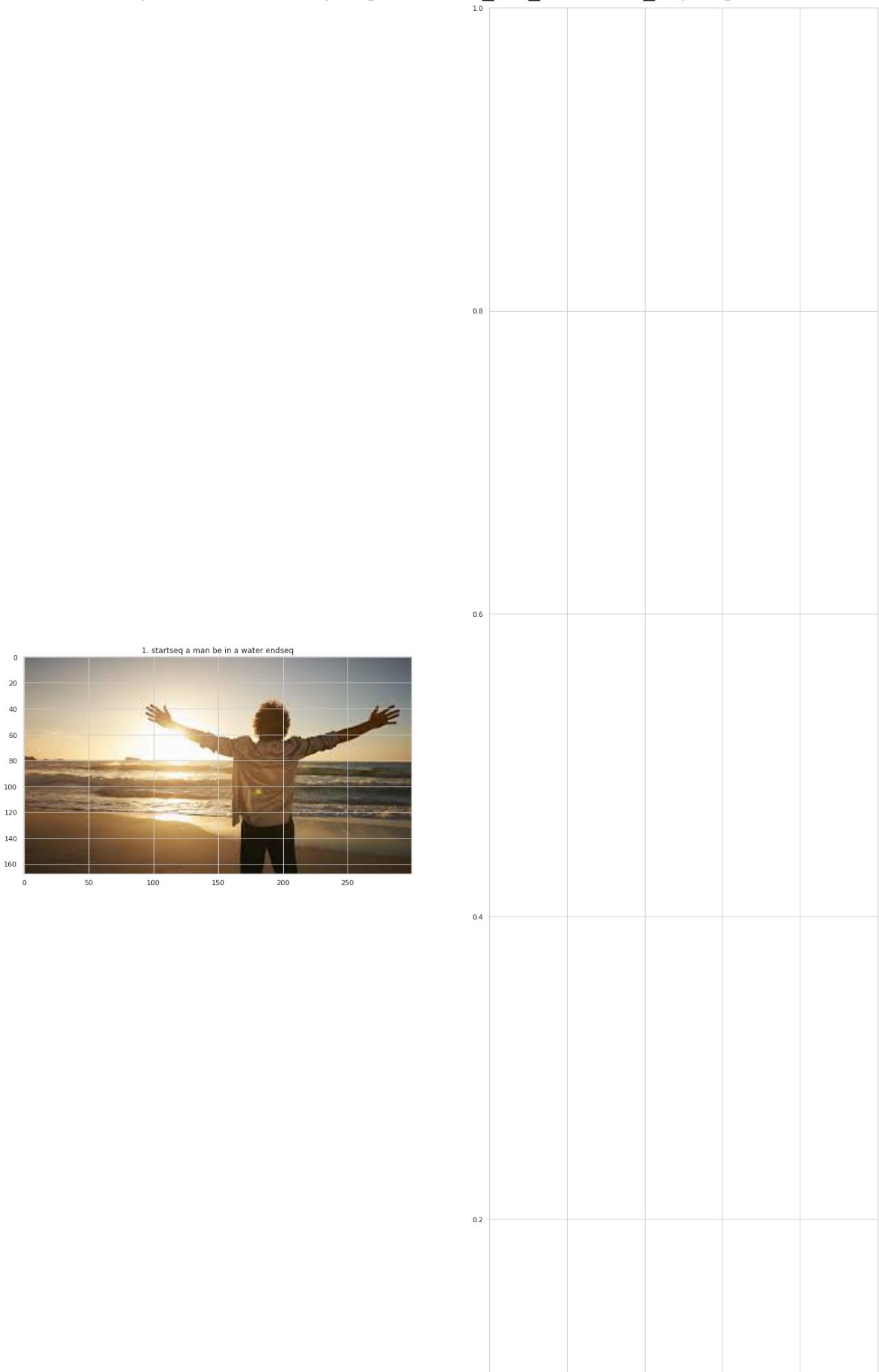
    # use model-1 here (to produce 4096 feature vector)
    img_feat = model1.predict(image)

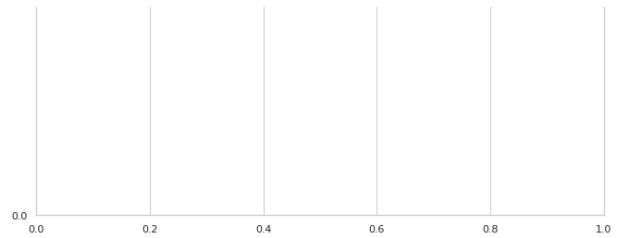
    label = greedySearch(recent_model, img_feat)

    display_images(root_path, [google_image_name], {google_image_name: [label]}, title)

else:
    print("no such file: {}".format(google_image_path))
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/data/ops/dataset_ops  
"Even though the `tf.config.experimental_run_functions_eagerly` "
```





```
# evaluation metrics.

for i, history in enumerate(histories):
    print("loss after epoch {} : {}".format(i, history.history['loss']))

loss after epoch 0 : [5.674670219421387]
loss after epoch 1 : [3.931886672973633]
loss after epoch 2 : [3.578935146331787]
loss after epoch 3 : [3.37323260307312]
loss after epoch 4 : [3.212104082107544]

from nltk.translate.bleu_score import corpus_bleu

def bleu_score(test_data, img_feat):
    actual, predicted = list(), list()
    for key, desc in test_data.items():
        label = greedySearch(recent_model, img_feat[key])
        label.replace('startseq ', '')
        label.replace(' endseq', '')
        actual_labels = [l.split() for l in test_data[key]]
        actual.append(actual_labels)
        predicted.append(label)

    print("BLEU-1: {}".format(corpus_bleu(actual, predicted, weights=(1.0, 0, 0, 0)))
    print("BLEU-2: {}".format(corpus_bleu(actual, predicted, weights=(0.5, 0.5, 0, 0)))
    print("BLEU-3: {}".format(corpus_bleu(actual, predicted, weights=(0.3, 0.3, 0.3, 0)))
    print("BLEU-4: {}".format(corpus_bleu(actual, predicted, weights=(0.25, 0.25, 0.25, 0)))

bleu_score(test_data, image_features_test)
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/data/ops/dataset_ops
  "Even though the `tf.config.experimental_run_functions_eagerly`"

KeyboardInterrupt                                     Traceback (most recent call last)
<ipython-input-29-cbf3b9d58229> in <module>()
    16     print("BLEU-4: {}".format(corpus_bleu(actual, predicted, weights=
(0.25, 0.25, 0.25, 0.25))))
    17
--> 18 bleu_score(test_data, image_features_test)

29 frames
```

bleu-scores for the test data are better for 4-grams compared to other n-grams, which suggests better / closer match to expected captions with longer sentences.

```
--> 81     def __init__(self, func, args, kwds):
```

Notes

We picked adam as the optimizer following a detailed analysis over here:

<https://ruder.io/optimizing-gradient-descent/>

While not always the best, adaptive learning algorithms may be more suited for sparse data, and work well with lesser memory requirements.

Both vanilla sgd and adam may be used here.

A learning rate of 0.001 was used for Adam optimizer, though various sources differ on an optimal value, many of them ranging from 1e-4 to 0.01, other on the higher side.

Due to time constraints we were not able to experiment with the learning rate and its optimal values, and went ahead with a typical learning rate of 0.001

A batch size of 64 was chosen, we tried with higher number but ran into memory issues, a lower batch size would result in a slower and may be inaccurate results.

Due to time constraints, we could not try with more than 3 epochs. Though the loss seems to be reducing with each epoch, a BLEU score for individual test data may be explored to see the accuracy of each of the generated captions.

Double-click (or enter) to edit

! 2m 23s completed at 23:49

