

Image Captioning using Deep Learning

Introduction

Caption generation is an artificial intelligence problem where a textual description must be generated for a given photograph. It requires both methods from computer vision and natural language processing. Computer vision is used to understand the content of the image while natural language processing is used to turn the understanding of the image into words in the right order.

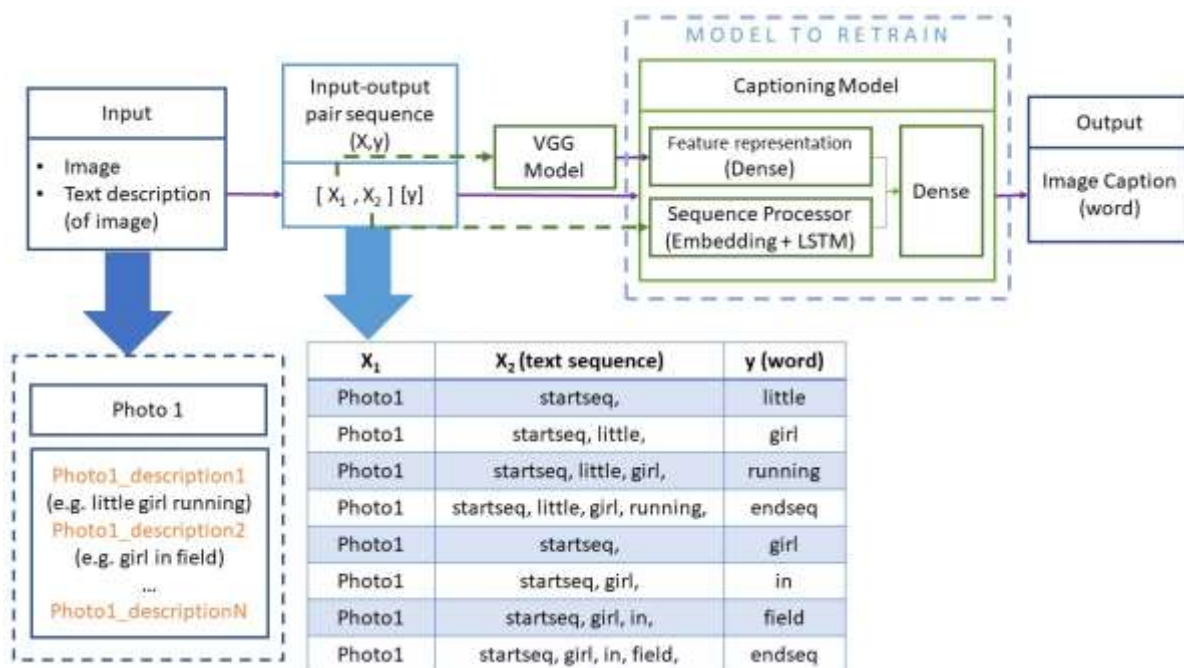
In this lab, you will be guided to create a deep learning model for photo captioning using pre-trained model and smaller dataset for faster training. In the last part of this guide, the homework is explained.

Objective

The students are expected to learn how to prepare photo and text data for training a deep learning model, how to retrain pre-trained model of caption generation model, how to evaluate a trained caption captioning model and use it to caption entirely new photographs.

Background

The task of image captioning can be divided into two modules: (1) image-based model, which extracts the features of the image, and (2) captioning model, which handles the text description of each image, merges with the image feature extracted and makes prediction over the entire output vocabulary for the next word in the sequence which describe the input image, as shown in the figure below.



The input image will be fed to the VGG model (without the output layer) and use the extracted features predicted as input to a Dense layer to produce a 256-element representation of the photo. Whereas the text descriptions which are split into words and encoded as integers are fed to the sequence processor which compose of Embedding layer that handles text input and then followed by LSTM layer with 256 memory units. Both the input models (Feature Extractor and Sequence Processor) produce a 256-element vector which are then merge using an add operation and fed to a 256-neuron Dense layer. Then to a final output Dense layer that makes the *softmax* prediction over the entire output vocabulary for the next word in the sequence.

However, creating this model from scratch takes time in training, thus, in this Lab exercise, we will use pre-trained model and retrain it on smaller dataset. By retraining means, we use pre-trained model as our starting point. To do this, we will follow the steps below:

1. Caption testing on pre-trained model
2. Retraining pre-trained model using smaller dataset (transfer learning) by following the steps below:
 - Prepare photo dataset by extracting its features using the VGG16 model
 - Prepare the text dataset by cleaning each text descriptions of each photo and by mapping the words to unique integer values using the *tokenizer*
 - Transform data into input-output pair for training the model
 - Load the pre-trained model and fit on the prepared data.

Implementation

Before training, we can examine the pre-trained model by testing it to caption a sample photo.

Caption testing on pre-trained model

In this section, we can test the pre-trained model by following the steps below.

Step 1: Download the model. You can download the model by running the script below. Doing so, it will download and unzip a compressed file (.zip file) which contains the pre-trained model and the dataset we will be using on this Lab.

```
1 # to download the dataset and pretrained model
2 !gdown --id 1D79dNcLXu6mV1Uueo7EJJXtQm2A09yUb
3 # to unzip the file
4 !unzip dataset.zip
```

Step 2: Import all necessary libraries for data and model loading.

```
3 from pickle import load
4 from numpy import argmax
5 from keras.preprocessing.sequence import pad_sequences
6 from keras.applications.vgg16 import VGG16
7 from keras.preprocessing.image import load_img
8 from keras.preprocessing.image import img_to_array
9 from keras.applications.vgg16 import preprocess_input
10 from keras.models import Model
11 from keras.models import load_model
12
13 import IPython.display as display
14 from PIL import Image
```

Step 3: Load the pre-trained model, the image to caption and the tokenizer. The tokenizer is used to encode generated words for the model while generating the sequence. This tokenizer is fitted on the pre-trained model which contains the dictionary of descriptions of the images used.

```
67 # load the tokenizer
68 tokenizer = load(open('/content/tokenizer.pkl', 'rb'))
69 # pre-define the max sequence length (from pre-training)
70 max_length = 34
71 # load the pre-trained model
72 model = load_model('/content/model_4.h5')
73 # load and prepare the photograph
74 photo = extract_features('/content/example.jpg')
75 # display the image
76 display.display(Image.open('/content/example.jpg'))
77 # generate description
78 description = generate_desc(model, tokenizer, photo, max_length)
79 print(description)
```

Running above script will render the following output. You can access the full code in [here](#).



Training using pre-trained model and smaller dataset

Before training, let's prepare the photo dataset and text data first.

Step 1: Extract features on photo images. We will use the VGG model to extract features on our images. We can pre-compute the “photo features” of our training and testing dataset and save to a file. Then, we can load these features later and feed into the model.

```
35 # extract features from each photo in the directory
36 def extract_features(directory, dataset):
37     # load the model
38     model = VGG16()
39     # re-structure the model
40     model = Model(inputs=model.inputs, outputs=model.layers[-2].output)
41     # summarize
42     print(model.summary())
43     # extract features from each photo
44     features = dict()
45     for name in dataset:
46         # load an image from file
47         filename = directory + '/' + name + '.jpg'
48         image = load_img(filename, target_size=(224,224))
49         # convert the image pixels to a numpy array
50         image = img_to_array(image)
51         # reshape data for the model
52         image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
53         # prepare the image for the VGG model
54         image = preprocess_input(image)
55         # get features
56         feature = model.predict(image, verbose=0)
57         # get image id
58         image_id = name.split('.')[0]
59         # store feature
60         features[image_id] = feature
61         print('>%s' % name)
62     return features
```

```
64 # extract features from training images
65 filename = '/content/Flickr8k_text/Flickr_100.trainImages.txt'
66 train = load_set(filename)
67 train_features = extract_features('/content/Flickr8k_Dataset', train)
68 print('Extracted Features of training data: %d' % len(train_features))
69 # save to file
70 dump(train_features, open('/content/features_train.pkl', 'wb'))
71
72 # extract features from testing images
73 filename = '/content/Flickr8k_text/Flickr_50.testImages.txt'
74 test = load_set(filename)
75 test_features = extract_features('/content/Flickr8k_Dataset', test)
76 print('Extracted Features of test data: %d' % len(test_features))
77 # save to file
78 dump(test_features, open('/content/features_test.pkl', 'wb'))
```

Step 2: Prepare text data. The dataset contains multiple descriptions for each photograph and each photo has a unique identifier. This identifier is used on the photo filename and in the text file of descriptions.

Step 2.1: Load the file containing all of the descriptions.

```
1 import string
2
3 # Load doc into memory
4 def load_doc(filename):
5     # open the file as read only
6     file = open(filename, 'r')
7     # read all text
8     text = file.read()
9     # close the file
10    file.close()
11    return text
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73 filename = 'dataset/Flickr8k_text/Flickr8k.token.txt'
74 # Load descriptions
75 doc = load_doc(filename)
```

Step 2.2: Map the descriptions to corresponding photo, each photo identifier maps to a list of one or more textual descriptions.

```
13 # extract descriptions for images
14 def load_descriptions(doc):
15     mapping = dict()
16     # process lines
17     for line in doc.split('\n'):
18         # split line by white space
19         tokens = line.split()
20         if len(tokens) < 2:
21             continue
22         # take the first token as the image id, the rest as the description
23         image_id, image_desc = tokens[0], tokens[1:]
24         # remove filename from image id
25         image_id = image_id.split('.')[0]
26         # convert description tokens back to string
27         image_desc = ' '.join(image_desc)
28         # create the list if needed:
29         if image_id not in mapping:
30             mapping[image_id] = list()
31         # store description
32         mapping[image_id].append(image_desc)
33     return mapping
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76 # parse descriptions
77 descriptions = load_descriptions(doc)
78 print('Loaded: %d ' % len(descriptions))
```

Step 2.3: The text of the descriptions requires some minimal cleaning to reduce the size of the vocabulary of words. The following cleaning will be applied:

- Convert all words to lowercase
- Remove all punctuations
- Remove all words that are one character or less in length (e.g. 'a')
- Remove all words that contain numbers

```

35 def clean_descriptions(descriptions):
36     # prepare translation table for removing punctuation
37     table = str.maketrans('', '', string.punctuation)
38     for key, desc_list in descriptions.items():
39         for i in range(len(desc_list)):
40             desc = desc_list[i]
41             # tokenize
42             desc = desc.split()
43             # convert to lower case
44             desc = [word.lower() for word in desc]
45             # remove punctuation from each token
46             desc = [w.translate(table) for w in desc]
47             # remove hanging 's' and 'a'
48             desc = [word for word in desc if len(word)>1]
49             # remove tokens with numbers
50             desc = [word for word in desc if word.isalpha()]
51             # store as string
52             desc_list[i] = ' '.join(desc)
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79 # clean descriptions
80 clean_descriptions(descriptions)

```

Step 2.4: Summarize the size of the vocabulary. Ideally, we want a vocabulary that is both expressive and as small as possible. A smaller vocabulary will result in a smaller that will train faster.

```

54 # convert the loaded descriptions into a vocabulary of words
55 def to_vocabulary(descriptions):
56     # build a list of all description strings
57     all_desc = set()
58     for key in descriptions.keys():
59         [all_desc.update(d.split()) for d in descriptions[key]]
60     return all_desc
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81 # summarize vocabulary
82 vocabulary = to_vocabulary(descriptions)
83 print('Vocabulary size: %d' % len(vocabulary))

```

Step 2.5: Finally, save the dictionary of image identifiers and descriptions to a new file named “*descriptions.txt*”, with one image identifier and description per line.

```

62 # save descriptions to file with one per line
63 def save_descriptions(descriptions, filename):
64     lines = list()
65     for key, desc_list in descriptions.items():
66         for desc in desc_list:
67             lines.append(key + ' ' + desc)
68     data = '\n'.join(lines)
69     file = open(filename, 'w')
70     file.write(data)
71     file.close()
72
73
74
75
76
77
78
79
80
81
82
83
84 # save to file
85 save_descriptions(descriptions, 'descriptions.txt')

```


Running the scripts above, will display the following output and save a file which look like as below. (The order of descriptions in your file may vary)

```
Loaded: 8092
Vocabulary size: 8763

1000268201_693b08cb0e child in pink dress is climbing up set of stairs in an entry way
1000268201_693b08cb0e girl going into wooden building
1000268201_693b08cb0e little girl climbing into wooden playhouse
1000268201_693b08cb0e little girl climbing the stairs to her playhouse
1000268201_693b08cb0e little girl in pink dress going into wooden cabin
1001773457_577c3a7d70 black dog and spotted dog are fighting
1001773457_577c3a7d70 black dog and tricolored dog playing with each other on the road
1001773457_577c3a7d70 black dog and white dog with brown spots are staring at each other in the street
1001773457_577c3a7d70 two dogs of different breeds looking at each other on the road
```

We can now retrain the model using the prepared data.

Step 1: Load the training data. The train dataset has been predefined in the *Flickr_100.trainImages.txt* that contains lists of photo file names.

```
3 from numpy import array
4 from pickle import load
5 from numpy import argmax
6 from keras.preprocessing.sequence import pad_sequences
7 from tensorflow.keras.utils import to_categorical
8 from keras.preprocessing.image import load_img
9 from keras.preprocessing.image import img_to_array
10 from keras.applications.vgg16 import preprocess_input
11 from keras.models import Model
12 from keras.models import load_model
13
14 # load doc into memory
15 def load_doc(filename):
16     # open the file as read only
17     file = open(filename, 'r')
18     # read all text
19     text = file.read()
20     # close the file
21     file.close()
22     return text
23
24 # load a pre-defined list of photo identifiers
25 def load_set(filename):
26     doc = load_doc(filename)
27     dataset = list()
28     # process line by line
29     for line in doc.split('\n'):
30         # skip empty lines
31         if len(line) < 1:
32             continue
33         # get the image identifier
34         identifier = line.split('.')[0]
35         dataset.append(identifier)
36     return set(dataset)
37
107 # load training dataset (100)
108 filename = '/content/Flickr8k_text/Flickr_100.trainImages.txt'
109 train = load_set(filename)
110 print('Dataset: %d' % len(train))
```

Step 2: Load the prepared descriptions, saved as “*descriptions.txt*”, which contains a set of identifiers and text descriptions. The model will generate a caption given a photo, and the caption will be generated one word at a time. The sequence of previously generated will be provided as input, therefore, we

will need of “*first word*” to kick-off the generation process and a “*last word*” to signal the end of the caption. With this, we will use the strings “*startseq*” and “*endseq*” (line 54). These tokens are added to the loaded descriptions as they are loaded. It is important to add these tokens first before encoding the text so that it will be encoded correctly.

```
38 # load clean descriptions into memory
39 def load_clean_descriptions(filename, dataset):
40     # load document
41     doc = load_doc(filename)
42     descriptions = dict()
43     for line in doc.split('\n'):
44         # split line by white space
45         tokens = line.split()
46         # split id from description
47         image_id, image_desc = tokens[0], tokens[1:]
48         # skip images not in the set
49         if image_id in dataset:
50             # create list
51             if image_id not in descriptions:
52                 descriptions[image_id] = list()
53             # wrap description in tokens
54             desc = 'startseq ' + ' '.join(image_desc) + ' endseq'
55             # store
56             descriptions[image_id].append(desc)
57     return descriptions
```

```
111 # descriptions
112 train_descriptions = load_clean_descriptions('/content/descriptions.txt', train)
113 print('Descriptions: train=%d' % len(train_descriptions))
```

Step 3: Load the prepared photos.

```
59 # load photo features
60 def load_photo_features(filename, dataset):
61     # load all features
62     all_features = load(open(filename, 'rb'))
63     # filter features
64     features = {k: all_features[k] for k in dataset}
65     return features
```

```
114 # photo features
115 train_features = load_photo_features('/content/features_train.pkl', train)
116 print('Photos: train=%d' % len(train_features))
```

Step 4: Load the tokenizer.

```
117 # load the tokenizer
118 tokenizer = load(open('/content/tokenizer.pkl', 'rb'))
119 vocab_size = len(tokenizer.word_index) + 1
120 print('Vocabulary Size: %d' % vocab_size)
121 # pre-define the max sequence length (from pre-training)
122 max_length = 34
123 print('Description Length: %d' % max_length)
```


Step 5: Transform data to input-output pairs for training the model. There are two input arrays to the model: (1) one for the photos and (2) one for the encoded text. And there is one output for the model which is the encoded next word in the text sequence. The input text is encoded as integers, which will be fed to a word embedding layer. The photo features will be fed directly to other part of the model. The model will output a prediction, which will be a probability distribution of all words in the vocabulary. The output data will therefore be a one-hot encoded version of each word, representing an idealized probability distribution with 0 values at all word positions except the actual word position, which has a value of 1.

```

67 # create sequences of images, input sequences and output word for an image
68 def create_sequences(tokenizer, max_length, desc_list, photo, vocab_size):
69     X1, X2, y = list(), list(), list()
70     # walk through each description for the image
71     for desc in desc_list:
72         # encode the sequence
73         seq = tokenizer.texts_to_sequences([desc])[0]
74         # split one sequence into multiple X,y pairs
75         for i in range(1, len(seq)):
76             # split into input and output pair
77             in_seq, out_seq = seq[:i], seq[i]
78             # pad input sequence
79             in_seq = pad_sequences([in_seq], maxlen=max_length)[0]
80             # encode output sequence
81             out_seq = to_categorical([out_seq], num_classes=vocab_size)[0]
82             # store
83             X1.append(photo)
84             X2.append(in_seq)
85             y.append(out_seq)
86     return array(X1), array(X2), array(y)

```

Step 6: Define the model. Since we're retraining from pre-trained model, we can just load the pre-trained model.

```

88 # define captioning model
89 def define_model():
90     model = load_model('/content/model_4.h5')
91     # summarize model
92     print(model.summary())
93     return model

```

```

127 # define the model
128 model = define_model()

```

Step 7: Fit the model and the save model after each epoch.

```

95 # intended to be used in a call to model.fit_generator()
96 def data_generator(descriptions, photos, tokenizer, max_length, vocab_size):
97     # loop forever over images
98     while 1:
99         for key, desc_list in descriptions.items():
100             # retrieve the photo feature
101             photo = photos[key][0]
102             in_img, in_seq, out_word = create_sequences(tokenizer, max_length, desc_list, photo, vocab_size)
103             yield [in_img, in_seq], out_word

```

```

129 # train the model, run epochs and save model after each epoch
130 epochs = 5
131 steps = len(train_descriptions)
132 for i in range(epochs):
133     # create data generator
134     generator = data_generator(train_descriptions, train_features, tokenizer, max_length, vocab_size)
135     # fit for one epoch
136     model.fit_generator(generator, epochs=1, steps_per_epoch=steps, verbose=1)
137     # save model
138     model.save('/content/lab_model_' + str(i) + '.h5')

```

After running the above scripts, it will train model, save the models and display the following.

```

Dataset: 100
Descriptions: train=100
Photos: train=100
Vocabulary Size: 7579
Description Length: 34
Model: "model_7"

```

Layer (type)	Output Shape	Param #	Connected to
input_17 (InputLayer)	[(None, 34)]	0	[]
input_16 (InputLayer)	[(None, 4096)]	0	[]
embedding_7 (Embedding)	(None, 34, 256)	1940224	['input_17[0][0]']
dropout_14 (Dropout)	(None, 4096)	0	['input_16[0][0]']
dropout_15 (Dropout)	(None, 34, 256)	0	['embedding_7[0][0]']
dense_20 (Dense)	(None, 256)	1048832	['dropout_14[0][0]']
lstm_7 (LSTM)	(None, 256)	525312	['dropout_15[0][0]']
add_7 (Add)	(None, 256)	0	['dense_20[0][0]', 'lstm_7[0][0]']
dense_21 (Dense)	(None, 256)	65792	['add_7[0][0]']
dense_22 (Dense)	(None, 7579)	1947803	['dense_21[0][0]']

```

=====
Total params: 5,527,963
Trainable params: 5,527,963
Non-trainable params: 0
=====
None
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:136: UserWarning: "Model.fit_generator"
100/100 [=====] - 31s 281ms/step - loss: 3.7674
/usr/local/lib/python3.7/dist-packages/keras/engine/functional.py:1410: CustomMaskWarning: Custom ma
layer_config = serialize_layer_fn(layer)
100/100 [=====] - 28s 282ms/step - loss: 3.3883
100/100 [=====] - 29s 289ms/step - loss: 3.1597
100/100 [=====] - 30s 301ms/step - loss: 2.9971
100/100 [=====] - 28s 284ms/step - loss: 2.8479

```

Once the model is fit, it can be evaluated on the holdout test dataset. We will evaluate a model by generating descriptions for all photos in the test dataset and evaluating those predictions with a standard cost function.

Step 1: Load the test dataset, its features and descriptions, and the tokenizer.

```

121 # load test set
122 filename = '/content/Flickr8K_text/Flickr_80.testImages.txt'
123 test = load_set(filename)
124 print('Dataset: %d' % len(test))
125 # descriptions
126 test_descriptions = load_clean_descriptions('/content/descriptions.txt', test)
127 print('Descriptions: test=%d' % len(test_descriptions))
128 # photo features
129 test_features = load_photo_features('/content/features_test.pkl', test)
130 print('Photos: test=%d' % len(test_features))
131 # load the tokenizer
132 tokenizer = load(open('/content/tokenizer.pkl', 'rb'))
133 # pre-define the max sequence length (from pre-training)
134 max_length = 34

```

Step 2: Define a function that can generate a description for a photo using the trained model. This involves passing in the start description token “*startseq*”, generating one word, then calling the model recursively with generated words as input until the end of sequence token is reached “*endseq*” or the maximum length is reached.

```
66 # map an integer to a word
67 def word_for_id(integer, tokenizer):
68     for word, index in tokenizer.word_index.items():
69         if index == integer:
70             return word
71     return None
72
73 # generate a description for an image
74 def generate_desc(model, tokenizer, photo, max_length):
75     # seed the generation process
76     in_text = 'startseq'
77     # iterate over the whole length of the sequence
78     for i in range(max_length):
79         # integer encode input sequence
80         sequence = tokenizer.texts_to_sequences([in_text])[0]
81         # pad input
82         sequence = pad_sequences([sequence], maxlen=max_length)
83         # predict next word
84         yhat = model.predict([photo, sequence], verbose=0)
85         # convert probability to integer
86         yhat = argmax(yhat)
87         # map integer to word
88         word = word_for_id(yhat, tokenizer)
89         # stop if cannot map the word
90         if word is None:
91             break
92         # append as input for generating the next word
93         in_text += ' ' + word
94         # stop if the end of the sequence is predicted
95         if word == 'endseq':
96             break
97     return in_text
```

Step 3: Evaluate a trained model against a given test dataset. The actual and predicted descriptions are collected and evaluated collectively using the corpus BLEU score that summarizes how close the generated text is to the expected text. BLEU scores are used in text translation for evaluating translated text against one or more reference translations. The NLTK Python library implements the BLEU score calculation in the *corpus_bleu()* function. A higher score close to 1 is better, a score closer to zero is worse. Then, we display the last photo with its generated caption.

```
99 # evaluate the skill of the model
100 def evaluate_model(model, descriptions, photos, tokenizer, max_length):
101     actual, predicted = list(), list()
102     # step over the whole set
103     for key, desc_list in descriptions.items():
104         # generate description
105         yhat = generate_desc(model, tokenizer, photos[key], max_length)
106         # store actual and predicted
107         references = [d.split() for d in desc_list]
108         actual.append(references)
109         predicted.append(yhat.split())
110     # calculate BLEU score
111     print('BLEU-1: %f' % corpus_bleu(actual, predicted, weights=(1.0, 0, 0, 0)))
112     print('BLEU-2: %f' % corpus_bleu(actual, predicted, weights=(0.5, 0.5, 0, 0)))
113     print('BLEU-3: %f' % corpus_bleu(actual, predicted, weights=(0.3, 0.3, 0.3, 0)))
114     print('BLEU-4: %f' % corpus_bleu(actual, predicted, weights=(0.25, 0.25, 0.25, 0.25)))
115
116 # display the predicted caption of the last photo
117 display.display(Image.open('/content/101chick Dataset/' + key + '-.jpg'))
118 print(yhat)

137 # load the model
138 filename = '/content/lab_model_4.h5'
139 model = load_model(filename)
140 # evaluate model
141 evaluate_model(model, test_descriptions, test_features, tokenizer, max_length)
```

Running the scripts above, render the following results.

```
Dataset: 50  
Descriptions: test=50  
Photos: test=50  
BLEU-1: 0.562609  
BLEU-2: 0.304598  
BLEU-3: 0.212766  
BLEU-4: 0.104708
```



startseq two people are playing on the beach endseq

Homework

Now that you know how to develop and evaluate a caption generation model, you can now use it to generate captions for entirely new photographs. Write a code that can generate caption for entirely new data (You can download new photos or used the image below).

For this homework, you are required to turn in the following:

- a. Code for generating a description for a new photograph.
- b. Generated caption result/s of the new photograph/s.

