Image Caption Generator with CNN & LSTM

Python based project where we will use deep learning techniques of Convolutional Neural Networks and a type of Recurrent Neural Network (LSTM) together.

In [1]:

```
import string
import numpy as np
from PIL import Image
import os
from pickle import dump, load
import numpy as np

from keras.applications.xception import Xception, preprocess_input
from keras.preprocessing.image import load_img, img_to_array
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical
from keras.layers import add
from keras.models import Model, load_model
from keras.layers import Input, Dense, LSTM, Embedding, Dropout

# small library for seeing the progress of loops.
from tqdm.notebook import tqdm as tqdm
tqdm().pandas()
```

```
0it [00:00, ?it/s]
```

In [2]:

```
# Loading a text file into memory
def load_doc(filename):
    # Opening the file as read only
    file = open(filename, 'r')
    text = file.read()
    file.close()
    return text
```

In [3]:

```
# get all imgs with their captions
def all_img_captions(filename):
    file = load_doc(filename)
    captions = file.split('\n')
    descriptions ={}
    for caption in captions[:-1]:
        img, caption = caption.split('\t')
        if img[:-2] not in descriptions:
            descriptions[img[:-2]] = [caption]
        else:
            descriptions[img[:-2]].append(caption)
    return descriptions
```

In [4]:

```
##Data cleaning- lower casing, removing puntuations and words containing numbers
def cleaning_text(captions):
    table = str.maketrans('','',string.punctuation)
    for img,caps in captions.items():
        for i,img_caption in enumerate(caps):

            img_caption.replace("-"," ")
            desc = img_caption.split()

            #converts to lower case
            desc = [word.lower() for word in desc]
            #remove punctuation from each token
            desc = [word.translate(table) for word in desc]
            #remove hanging 's and a
            desc = [word for word in desc if(len(word)>1)]
            #remove tokens with numbers in them
            desc = [word for word in desc if(word.isalpha())]
            #convert back to string

            img_caption = ' '.join(desc)
            captions[img][i]= img_caption
    return captions
```

In [5]:

```
def text_vocabulary(descriptions):
    # build vocabulary of all unique words
    vocab = set()

    for key in descriptions.keys():
        [vocab.update(d.split()) for d in descriptions[key]]
    return vocab
```

In [6]:

```
#All descriptions in one file
def save_descriptions(descriptions, filename):
    lines = list()
    for key, desc_list in descriptions.items():
        for desc in desc_list:
            lines.append(key + '\t' + desc )
    data = "\n".join(lines)
    file = open(filename,"w")
    file.write(data)
    file.close()
```

In [7]:

```
# Set these path according to project folder in you system
dataset_text = "C:/Users/vidit/Desktop/MS in DS/Projects/Image Caption Generator/Flickr8k_text"
dataset_images = "C:/Users/vidit/Desktop/MS in DS/Projects/Image Caption Generator/Flicker8k_Dataset"
```

In [8]:

```
#we prepare our text data
filename = dataset_text + "/" + "Flickr8k.token.txt"
#loading the file that contains all data
#mapping them into descriptions dictionary img to 5 captions
descriptions = all_img_captions(filename)
print("Length of descriptions =" ,len(descriptions))
```

```
Length of descriptions = 8092
```

In [9]:

```
#cleaning the descriptions
clean_descriptions = cleaning_text(descriptions)
```

In [10]:

```
#building vocabulary
vocabulary = text_vocabulary(clean_descriptions)
print("Length of vocabulary = ", len(vocabulary))
```

```
Length of vocabulary =  8763
```

In [11]:

```
#saving each description to file
save_descriptions(clean_descriptions, "descriptions.txt")
```

In [12]:

```
def extract_features(directory):
        model = Xception( include_top=False, pooling='avg' )
        features = {}
        for img in tqdm(os.listdir(directory)):
            filename = directory + "/" + img
            image = Image.open(filename)
            image = image.resize((299,299))
            image = np.expand_dims(image, axis=0)
            #image = preprocess_input(image)
            image = image/127.5
            image = image - 1.0

            feature = model.predict(image)
            features[img] = feature
        return features
```

Extracting the feature vector from all images

In [23]:

```
#2048 feature vector
features = extract_features(dataset_images)
dump(features, open("features.p","wb"))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/xception/xception_weights_tf_dim_
ordering_tf_kernels_notop.h5
83683744/83683744 [==============================] - 35s 0us/step
  0%|          | 0/8091 [00:00<?, ?it/s]
```

```
1/1 [==============================] - 0s 180ms/step
1/1 [==============================] - 0s 186ms/step
1/1 [==============================] - 0s 183ms/step
1/1 [==============================] - 0s 186ms/step
1/1 [==============================] - 0s 183ms/step
1/1 [==============================] - 0s 183ms/step
1/1 [==============================] - 0s 214ms/step
1/1 [==============================] - 0s 194ms/step
1/1 [==============================] - 0s 205ms/step
1/1 [==============================] - 0s 198ms/step
1/1 [==============================] - 0s 206ms/step
1/1 [==============================] - 0s 201ms/step
1/1 [==============================] - 0s 186ms/step
1/1 [==============================] - 0s 190ms/step
1/1 [==============================] - 0s 185ms/step
1/1 [==============================] - 0s 192ms/step
1/1 [==============================] - 0s 179ms/step
1/1 [==============================] - 0s 181ms/step
1/1 [==============================] - 0s 188ms/step
1/1 [==============================] - 0s 180ms/step
1/1 [==============================] - 0s 181ms/step
1/1 [==============================] - 0s 183ms/step
1/1 [==============================] - 0s 190ms/step
1/1 [==============================] - 0s 185ms/step
1/1 [==============================] - 0s 193ms/step
1/1 [==============================] - 0s 218ms/step
1/1 [==============================] - 0s 197ms/step
```

In [13]:

```python
features = load(open("features.p","rb"))
```

In [14]:

```python
#load the data
def load_photos(filename):
    file = load_doc(filename)
    photos = file.split("\n")[:-1]
    return photos
```

In [15]:

```python
def load_clean_descriptions(filename, photos):
    #loading clean_descriptions
    file = load_doc(filename)
    descriptions = {}
    for line in file.split("\n"):
        words = line.split()
        if len(words)<1 :
            continue

        image, image_caption = words[0], words[1:]

        if image in photos:
            if image not in descriptions:
                descriptions[image] = []
            desc = '<start> ' + " ".join(image_caption) + ' <end>'
            descriptions[image].append(desc)

    return descriptions
```

In [16]:

```
def load_features(photos):
    #loading all features
    all_features = load(open("features.p","rb"))
    #selecting only needed features
    features = {k:all_features[k] for k in photos}
    return features
```

In [17]:

```
filename = dataset_text + "/" + "Flickr_8k.trainImages.txt"

#train = loading_data(filename)
train_imgs = load_photos(filename)
train_descriptions = load_clean_descriptions("descriptions.txt", train_imgs)
train_features = load_features(train_imgs)
```

Tokenizing the vocabulary

In [18]:

```
#converting dictionary to clean list of descriptions
def dict_to_list(descriptions):
    all_desc = []
    for key in descriptions.keys():
        [all_desc.append(d) for d in descriptions[key]]
    return all_desc
```

In [19]:

```
#creating tokenizer class
#this will vectorise text corpus
#each integer will represent token in dictionary

from keras.preprocessing.text import Tokenizer

def create_tokenizer(descriptions):
    desc_list = dict_to_list(descriptions)
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(desc_list)
    return tokenizer
```

In [20]:

```
# give each word a index, and store that into tokenizer.p pickle file
tokenizer = create_tokenizer(train_descriptions)
dump(tokenizer, open('tokenizer.p', 'wb'))
vocab_size = len(tokenizer.word_index) + 1
vocab_size
```

Out[20]:

```
7577
```

In [21]:

```
#calculate maximum length of descriptions
def max_length(descriptions):
    desc_list = dict_to_list(descriptions)
    return max(len(d.split()) for d in desc_list)

max_length = max_length(descriptions)
max_length
```

Out[21]:

```
32
```

In [22]:

```
features['1000268201_693b08cb0e.jpg'][0]
```

Out[22]:

```
array([0.47340965, 0.01730897, 0.07334232, ..., 0.08557969, 0.02102296,
       0.23765543], dtype=float32)
```

Define the model

- Photo feature extractor - we extracted features from pretrained model Xception.
- Sequence processor - word embedding layer that handles text, followed by LSTM

- Decoder - Both 1 and 2 model produce fixed length vector. They are merged together and processed by dense layer to make final prediction

In [23]:

```
#create input-output sequence pairs from the image description.

#data generator, used by model.fit_generator()
def data_generator(descriptions, features, tokenizer, max_length):
    while 1:
        for key, description_list in descriptions.items():
            #retrieve photo features
            feature = features[key][0]
            input_image, input_sequence, output_word = create_sequences(tokenizer, max_length, description_list, feat
            yield [[input_image, input_sequence], output_word]

def create_sequences(tokenizer, max_length, desc_list, feature):
    X1, X2, y = list(), list(), list()
    # walk through each description for the image
    for desc in desc_list:
        # encode the sequence
        seq = tokenizer.texts_to_sequences([desc])[0]
        # 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_length)[0]
            # encode output sequence
            out_seq = to_categorical([out_seq], num_classes=vocab_size)[0]
            # store
            X1.append(feature)
            X2.append(in_seq)
            y.append(out_seq)
    return np.array(X1), np.array(X2), np.array(y)
```

In [24]:

```
[a,b],c = next(data_generator(train_descriptions, features, tokenizer, max_length))
a.shape, b.shape, c.shape
```

Out[24]:

```
((47, 2048), (47, 32), (47, 7577))
```

Defining the CNN-RNN model

In [25]:

```
from keras.utils import plot_model

# define the captioning model
def define_model(vocab_size, max_length):

    # features from the CNN model squeezed from 2048 to 256 nodes
    inputs1 = Input(shape=(2048,))
    fe1 = Dropout(0.5)(inputs1)
    fe2 = Dense(256, activation='relu')(fe1)

    # LSTM sequence model
    inputs2 = Input(shape=(max_length,))
    se1 = Embedding(vocab_size, 256, mask_zero=True)(inputs2)
    se2 = Dropout(0.5)(se1)
    se3 = LSTM(256)(se2)

    # Merging both models
    decoder1 = add([fe2, se3])
    decoder2 = Dense(256, activation='relu')(decoder1)
    outputs = Dense(vocab_size, activation='softmax')(decoder2)

    # tie it together [image, seq] [word]
    model = Model(inputs=[inputs1, inputs2], outputs=outputs)
    model.compile(loss='categorical_crossentropy', optimizer='adam')

    # summarize model
    print(model.summary())
    plot_model(model, to_file='model.png', show_shapes=True)

    return model
```

Training the model

In [27]:

```
# train our model
print('Dataset: ', len(train_imgs))
print('Descriptions: train=', len(train_descriptions))
```

```python
print('Photos: train=', len(train_features))
print('Vocabulary Size:', vocab_size)
print('Description Length: ', max_length)

model = define_model(vocab_size, max_length)
epochs = 10
steps = len(train_descriptions)
# making a directory models to save our models
os.mkdir("models")
for i in range(epochs):
    generator = data_generator(train_descriptions, train_features, tokenizer, max_length)
    model.fit_generator(generator, epochs=1, steps_per_epoch= steps, verbose=1)
    model.save("models/model_" + str(i) + ".h5")
```

```
Dataset:  6000
Descriptions: train= 6000
Photos: train= 6000
Vocabulary Size: 7577
Description Length:  32
Model: "model_1"
__________________________________________________________________________________________________
 Layer (type)                Output Shape                 Param #   Connected to                  
==================================================================================================
 input_4 (InputLayer)        [(None, 32)]                 0         []                            
                                                                                                  
 input_3 (InputLayer)        [(None, 2048)]               0         []                            
                                                                                                  
 embedding_1 (Embedding)     (None, 32, 256)              1939712   ['input_4[0][0]']             
                                                                                                  
 dropout_2 (Dropout)         (None, 2048)                 0         ['input_3[0][0]']             
                                                                                                  
 dropout_3 (Dropout)         (None, 32, 256)              0         ['embedding_1[0][0]']         
                                                                                                  
 dense_3 (Dense)             (None, 256)                  524544    ['dropout_2[0][0]']           
                                                                                                  
 lstm_1 (LSTM)               (None, 256)                  525312    ['dropout_3[0][0]']           
                                                                                                  
 add_1 (Add)                 (None, 256)                  0         ['dense_3[0][0]',             
                                                                     'lstm_1[0][0]']              
                                                                                                  
 dense_4 (Dense)             (None, 256)                  65792     ['add_1[0][0]']               
                                                                                                  
 dense_5 (Dense)             (None, 7577)                 1947289   ['dense_4[0][0]']             
                                                                                                  
==================================================================================================
Total params: 5002649 (19.08 MB)
Trainable params: 5002649 (19.08 MB)
Non-trainable params: 0 (0.00 Byte)
__________________________________________________________________________________________________
None
You must install pydot (`pip install pydot`) and install graphviz (see instructions at https://graphviz.gitlab.io/do
wnload/) for plot_model to work.
C:\Users\vidit\AppData\Local\Temp\ipykernel_17764\2846483122.py:15: UserWarning: `Model.fit_generator` is deprecated
and will be removed in a future version. Please use `Model.fit`, which supports generators.
  model.fit_generator(generator, epochs=1, steps_per_epoch= steps, verbose=1)
6000/6000 [==============================] - 1446s 240ms/step - loss: 4.5192
```

```
C:\Users\vidit\AppData\Roaming\Python\Python310\site-packages\keras\src\engine\training.py:3079: UserWarning: You ar
e saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using in
stead the native Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
6000/6000 [==============================] - 1444s 241ms/step - loss: 3.6668
6000/6000 [==============================] - 1800s 300ms/step - loss: 3.3763
6000/6000 [==============================] - 1778s 296ms/step - loss: 3.2024
6000/6000 [==============================] - 1868s 311ms/step - loss: 3.0830
6000/6000 [==============================] - 1903s 317ms/step - loss: 2.9933
6000/6000 [==============================] - 1878s 313ms/step - loss: 2.9248
6000/6000 [==============================] - 1884s 314ms/step - loss: 2.8677
6000/6000 [==============================] - 1915s 319ms/step - loss: 2.8221
6000/6000 [==============================] - 1963s 327ms/step - loss: 2.7828
```

Testing the model

In [28]:

```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.applications.xception import Xception
from keras.models import load_model
from pickle import load
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
import argparse
```

In [59]:

```
img_path = "C:/Users/vidit/Desktop/MS in DS/Projects/Image Caption Generator/Flicker8k_Dataset/3665179773_dd217416fc.
#img_path = "C:/Users/vidit/Desktop/MS in DS/Projects/Image Caption Generator/Flicker8k_Dataset/3471463779_64084b686c
#img_path = "C:/Users/vidit/Desktop/MS in DS/Projects/Image Caption Generator/Flicker8k_Dataset/3457572788_e1fe4f6480
#img_path = "C:/Users/vidit/Desktop/MS in DS/Projects/Image Caption Generator/Flicker8k_Dataset/421808539_57abee6d55.
#img_path = "C:/Users/vidit/Desktop/MS in DS/Projects/Image Caption Generator/Flicker8k_Dataset/242109387_e497277e07.
```

In [70]:

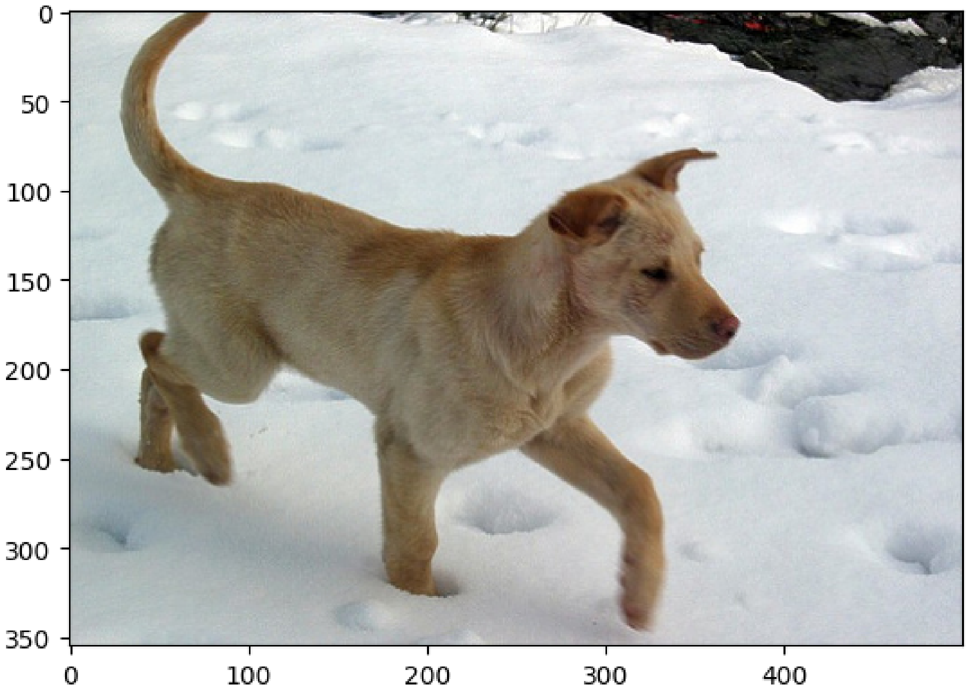
```
def result(img_path):
    def extract_features(filename, model):
        try:
            image = Image.open(filename)

        except:
            print("ERROR: Couldn't open image! Make sure the image path and extension is correct")
        image = image.resize((299,299))
```

```python
    image = np.array(image)
    # for images that has 4 channels, we convert them into 3 channels
    if image.shape[2] == 4:
        image = image[..., :3]
    image = np.expand_dims(image, axis=0)
    image = image/127.5
    image = image - 1.0
    feature = model.predict(image)
    return feature

def word_for_id(integer, tokenizer):
    for word, index in tokenizer.word_index.items():
        if index == integer:
            return word
    return None


def generate_desc(model, tokenizer, photo, max_length):
    in_text = 'start'
    for i in range(max_length):
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        sequence = pad_sequences([sequence], maxlen=max_length)
        pred = model.predict([photo,sequence], verbose=0)
        pred = np.argmax(pred)
        word = word_for_id(pred, tokenizer)
        if word is None:
            break
        in_text += ' ' + word
        if word == 'end':
            break
    return in_text



max_length = 32
tokenizer = load(open("tokenizer.p","rb"))
model = load_model('models/model_9.h5')
xception_model = Xception(include_top=False, pooling="avg")

photo = extract_features(img_path, xception_model)
img = Image.open(img_path)
```

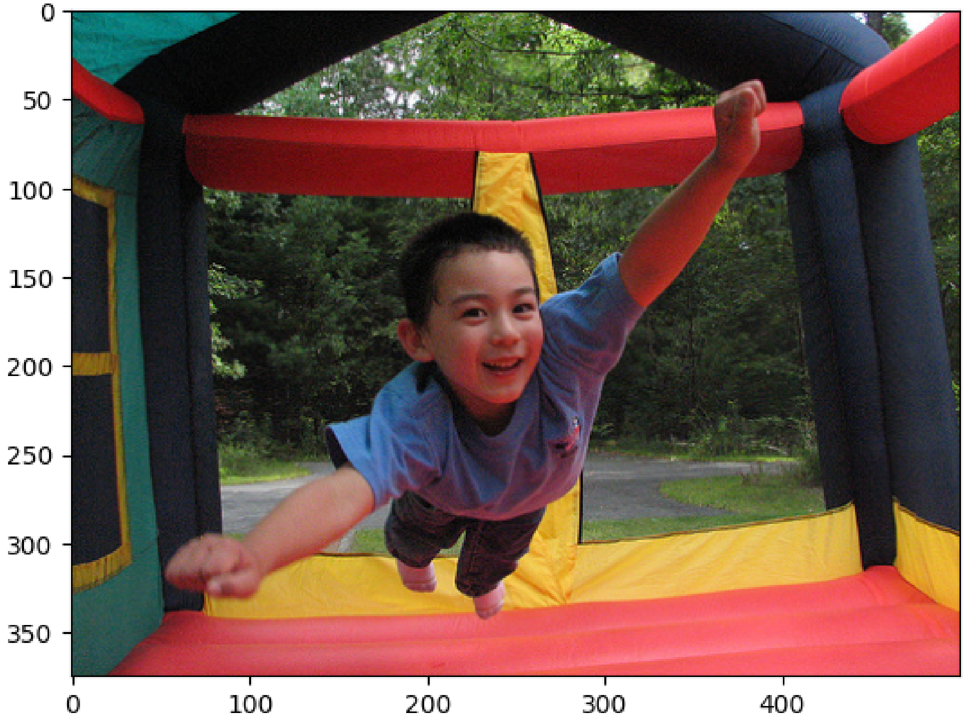
```
    description = generate_desc(model, tokenizer, photo, max_length)
    print("\n\n")
    print(description)
    plt.imshow(img)
    plt.show()
```

In [71]:
```
Arr=["421808539_57abee6d55","242109387_e497277e07","3457572788_e1fe4f6480","3665179773_dd217416fc","3471463779_64084

for i in Arr:
    img_path = "C:/Users/vidit/Desktop/MS in DS/Projects/Image Caption Generator/Flicker8k_Dataset/"+i+".jpg"
    result(img_path)
```

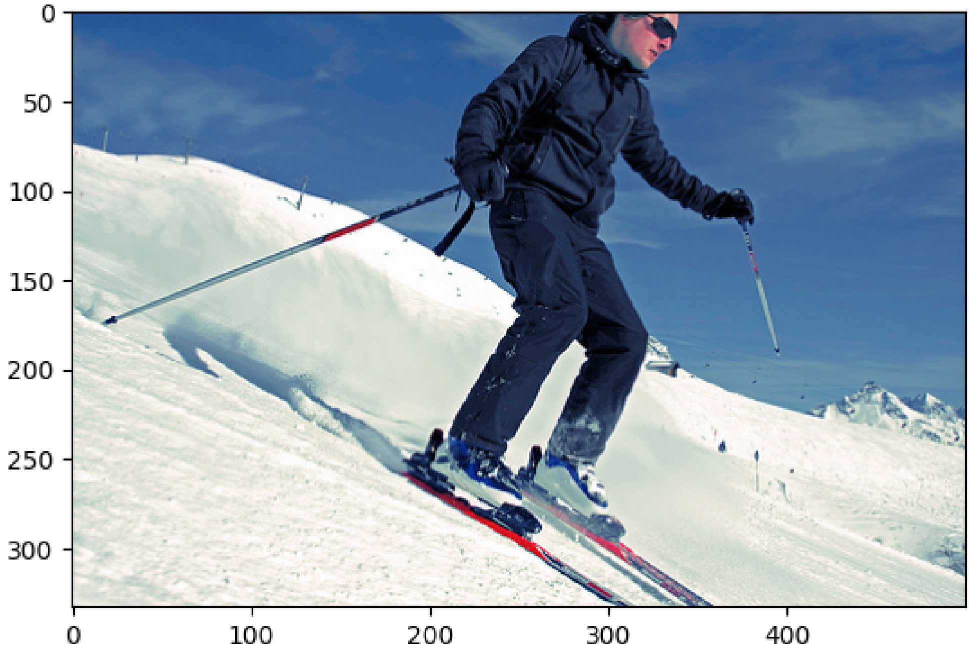
```
1/1 [==============================] - 1s 953ms/step



start dog is running through the snow end
```



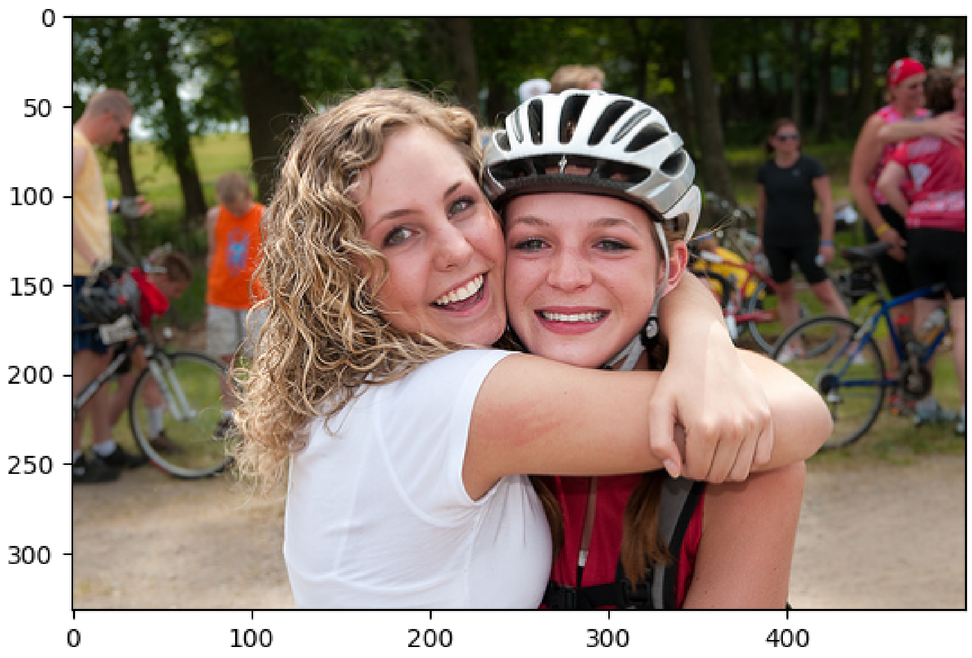
```
1/1 [==============================] - 1s 823ms/step


start little boy in pink shirt plays on playground end
```



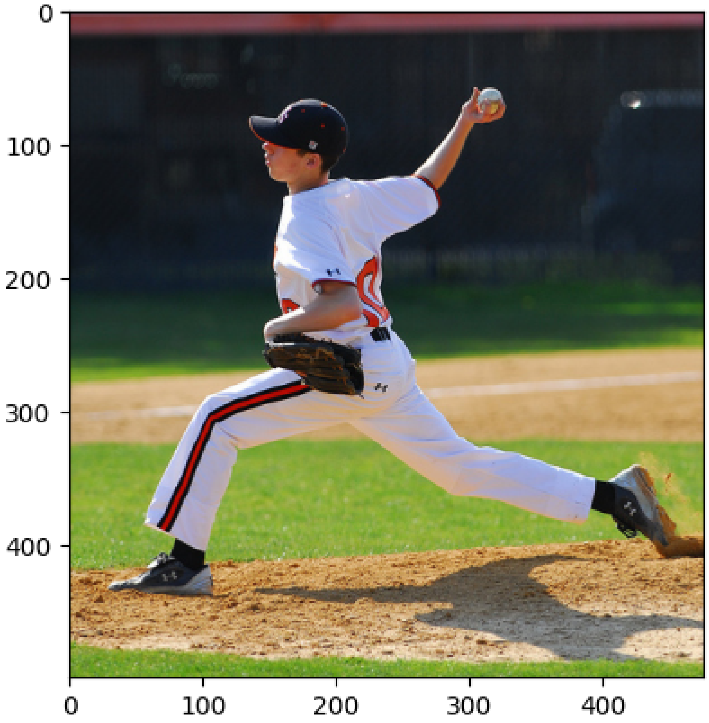
```
1/1 [==============================] - 1s 784ms/step


start skier is skiing down snowy hill end
```



```
1/1 [==============================] - 1s 1s/step


start man in red shirt is sitting on bench with dog end
```



```
1/1 [==============================] - 1s 878ms/step


start baseball player in white shirt is throwing ball end
```



```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```