Computer Vision Nanodegree

Project: Image Captioning

In this notebook, you will use your trained model to generate captions for images in the test dataset.

This notebook **will be graded**.

Feel free to use the links below to navigate the notebook:

- Step 1: Get Data Loader for Test Dataset
- Step 2: Load Trained Models
- Step 3: Finish the Sampler
- Step 4: Clean up Captions
- Step 5: Generate Predictions!

Step 1: Get Data Loader for Test Dataset

Before running the code cell below, define the transform in `transform_test` that you would like to use to pre-process the test images.

Make sure that the transform that you define here agrees with the transform that you used to pre-process the training images (in **2_Training.ipynb**). For instance, if you normalized the training images, you should also apply the same normalization procedure to the test images.

In [18]:

```
import sys
sys.path.append('/opt/cocoapi/PythonAPI')
from pycocotools.coco import COCO
from data_loader import get_loader
from torchvision import transforms

# TODO #1: Define a transform to pre-process the testing images.
transform_test = transforms.Compose([
    transforms.Resize(256),                          # smaller edge of image resized to 256
    transforms.RandomCrop(224),                      # get 224x224 crop from random location
    transforms.RandomHorizontalFlip(),               # horizontally flip image with probability=(
    transforms.ToTensor(),                           # convert the PIL Image to a tensor
    transforms.Normalize((0.485, 0.456, 0.406),      # normalize image for pre-trained model
                         (0.229, 0.224, 0.225))])

#-#-#-# Do NOT modify the code below this line. #-#-#-#

# Create the data loader.
data_loader = get_loader(transform=transform_test,
                         mode='test')
```

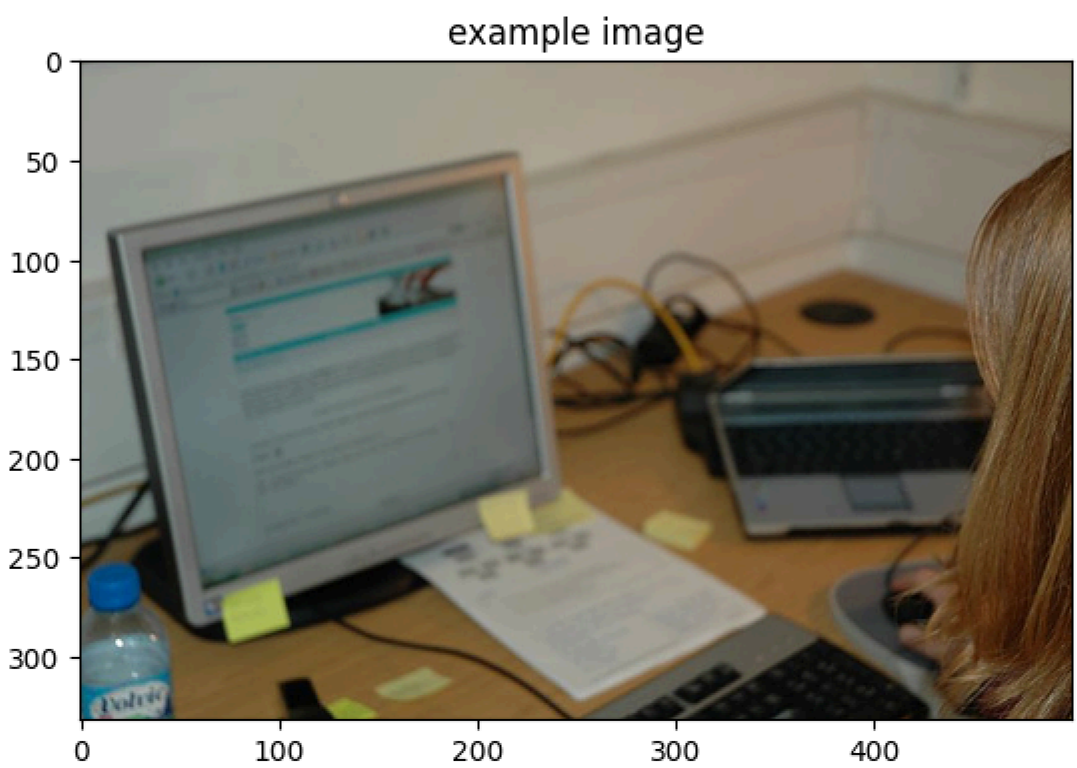
```
Vocabulary successfully loaded from vocab.pkl file!
```

Run the code cell below to visualize an example test image, before pre-processing is applied.

In [19]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

# Obtain sample image before and after pre-processing.
orig_image, image = next(iter(data_loader))

# Visualize sample image, before pre-processing.
plt.imshow(np.squeeze(orig_image))
plt.title('example image')
plt.show()
```



Step 2: Load Trained Models

In the next code cell we define a `device` that you will use move PyTorch tensors to GPU (if CUDA is available). Run this code cell before continuing.

In [20]:
```
import torch

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

In [21]:
```
device
```

Out[21]:
```
device(type='cuda')
```

Before running the code cell below, complete the following tasks.

Task #1

In the next code cell, you will load the trained encoder and decoder from the previous notebook (**2_Training.ipynb**). To accomplish this, you must specify the names of the saved encoder and decoder files in the `models/` folder (e.g., these names should be `encoder-5.pkl` and `decoder-5.pkl`, if you trained the model for 5 epochs and saved the weights after each epoch).

Task #2

Plug in both the embedding size and the size of the hidden layer of the decoder corresponding to the selected pickle file in `decoder_file`.

In [22]:
```
# Watch for any changes in model.py, and re-load it automatically.
# % load_ext autoreload
# % autoreload 2

import os
import torch
from model import EncoderCNN, DecoderRNN

# Specify the saved models to load.
encoder_file = 'encoder-3.pkl'
decoder_file = 'decoder-3.pkl'

# Select appropriate values for the Python variables below.
embed_size = 256
hidden_size = 512

# The size of the vocabulary.
vocab_size = len(data_loader.dataset.vocab)

# Initialize the encoder and decoder, and set each to inference mode.
encoder = EncoderCNN(embed_size)
encoder.eval()
decoder = DecoderRNN(embed_size, hidden_size, vocab_size)
decoder.eval()

# Load the trained weights.
encoder.load_state_dict(torch.load(os.path.join('./models', encoder_file)))
decoder.load_state_dict(torch.load(os.path.join('./models', decoder_file)))

# Move models to GPU if CUDA is available.
encoder.to(device)
decoder.to(device)
```

```
/tmp/ipykernel_111425/1919389756.py:27: FutureWarning: You are using `torch.load` with `weights_only=False` (the current default value), which uses the default pickle module implicitly. It is possible to construct malicious pickle data which will execute arbitrary code during unpickling (See https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for more details). In a future release, the default value for `weights_only` will be flipped to `True`. This limits the functions that could be executed during unpickling. Arbitrary objects will no longer be allowed to be loaded via this mode unless they are explicitly allowlisted by the user via `torch.serialization.add_safe_globals`. We recommend you start setting `weights_only=True` for any use case where you don't have full control of the loaded file. Please open an issue on GitHub for any issues related to this experimental feature.
  encoder.load_state_dict(torch.load(os.path.join('./models', encoder_file)))
/tmp/ipykernel_111425/1919389756.py:28: FutureWarning: You are using `torch.load` with `weights_only=False` (the current default value), which uses the default pickle module implicitly. It is possible to construct malicious pickle data which will execute arbitrary code during unpickling (See https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for more details). In a future release, the default value for `weights_only` will be flipped to `True`. This limits the functions that could be executed during unpickling. Arbitrary objects will no longer be allowed to be loaded via this mode unless they are explicitly allowlisted by the user via `torch.serialization.add_safe_globals`. We recommend you start setting `weights_only=True` for any use case where you don't have full control of the loaded file. Please open an issue on GitHub for any issues related to this experimental feature.
  decoder.load_state_dict(torch.load(os.path.join('./models', decoder_file)))
```

Out[22]:

```
DecoderRNN(
  (word_embedding): Embedding(9947, 256)
  (lstm): LSTM(256, 512, batch_first=True)
  (fc): Linear(in_features=512, out_features=9947, bias=True)
)
```

Step 3: Finish the Sampler

Before executing the next code cell, you must write the `sample` method in the `DecoderRNN` class in **model.py**. This method should accept as input a PyTorch tensor `features` containing the embedded input features corresponding to a single image.

It should return as output a Python list `output`, indicating the predicted sentence. `output[i]` is a nonnegative integer that identifies the predicted `i`-th token in the sentence. The correspondence between integers and tokens can be explored by examining either `data_loader.dataset.vocab.word2idx` (or `data_loader.dataset.vocab.idx2word`).

After implementing the `sample` method, run the code cell below. If the cell returns an assertion error, then please follow the instructions to modify your code before proceeding. Do **not** modify the code in the cell below.

In [23]:

```
# Move image Pytorch Tensor to GPU if CUDA is available.
image = image.to(device)

# Obtain the embedded image features.
features = encoder(image).unsqueeze(1)

# Pass the embedded image features through the model to get a predicted caption.
output = decoder.sample(features)
print('example output:', output)

assert (type(output)==list), "Output needs to be a Python list"
assert all([type(x)==int for x in output]), "Output should be a list of integers."
assert all([x in data_loader.dataset.vocab.idx2word for x in output]), "Each entry in the output
```

```
example output: [5305, 1575, 21, 3, 2, 39, 32, 121, 18]
```

Step 4: Clean up the Captions

In the code cell below, complete the `clean_sentence` function. It should take a list of integers (corresponding to the variable `output` in **Step 3**) as input and return the corresponding predicted sentence (as a single Python string).

In [24]:
```
# TODO #4: Complete the function.
def clean_sentence(output):

    return sentence
```

In [25]:
```
def clean_sentence(output):
    cleaned_list = []
    for index in output:
        if  (index == 1) :
            continue
        cleaned_list.append(data_loader.dataset.vocab.idx2word[index])
    cleaned_list = cleaned_list[1:-1] # Discard <start> and <end>

    sentence = ' '.join(cleaned_list) # Convert list of string to
    sentence = sentence.capitalize()
    return sentence
```

After completing the `clean_sentence` function above, run the code cell below. If the cell returns an assertion error, then please follow the instructions to modify your code before proceeding.

In [26]:
```
sentence = clean_sentence(output)
print('example sentence:', sentence)

assert type(sentence)==str, 'Sentence needs to be a Python string!'
```

```
example sentence: Racket with a <unk> on the front
```

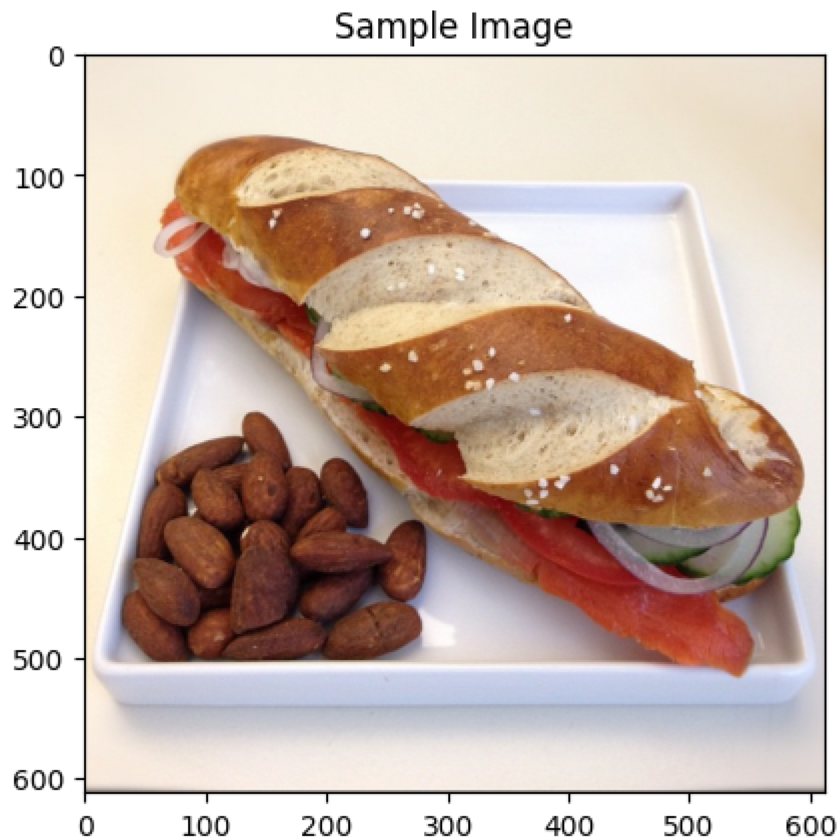
Step 5: Generate Predictions!

In the code cell below, we have written a function (`get_prediction`) that you can use to use to loop over images in the test dataset and print your model's predicted caption.

In [27]:
```
def get_prediction():
    orig_image, image = next(iter(data_loader))
    plt.imshow(np.squeeze(orig_image))
    plt.title('Sample Image')
    plt.show()
    image = image.to(device)
    features = encoder(image).unsqueeze(1)
    output = decoder.sample(features)
    sentence = clean_sentence(output)
    print(sentence)
```

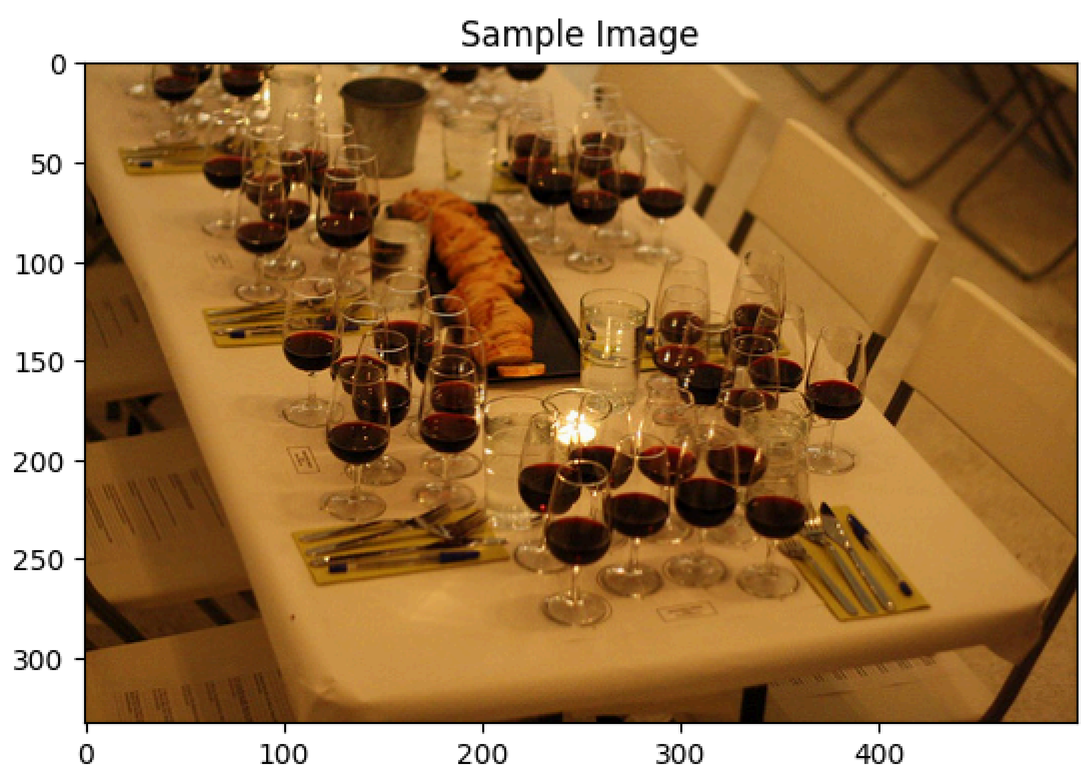
Run the code cell below (multiple times, if you like!) to test how this function works.

In [28]:
```
get_prediction()
```




```
A bowl of food with broccoli and carrots
```

In [29]:
```
get_prediction()
```




```
A small table with a laptop and a mouse
```

As the last task in this project, you will loop over the images until you find four image-caption pairs of interest:

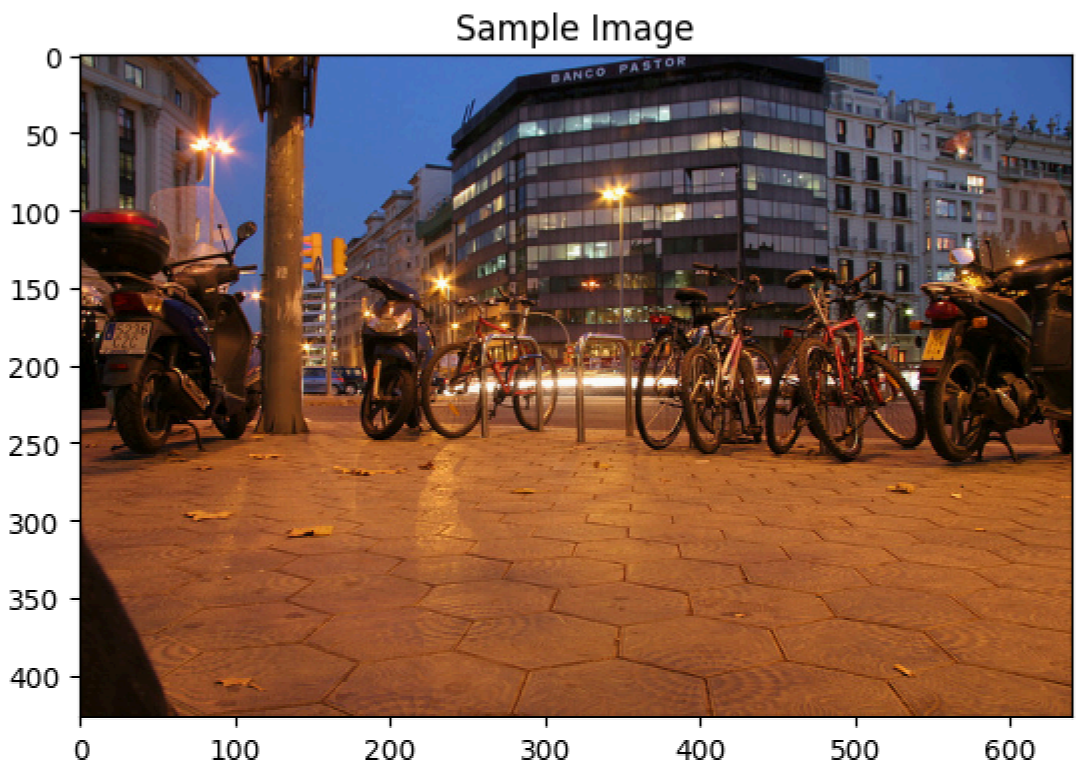
- Two should include image-caption pairs that show instances when the model performed well.
- Two should highlight image-caption pairs that highlight instances where the model did not perform well.

Use the four code cells below to complete this task.

The model performed well!

Use the next two code cells to loop over captions. Save the notebook when you encounter two images with relatively accurate captions.

In [30]:
```
get_prediction()
```

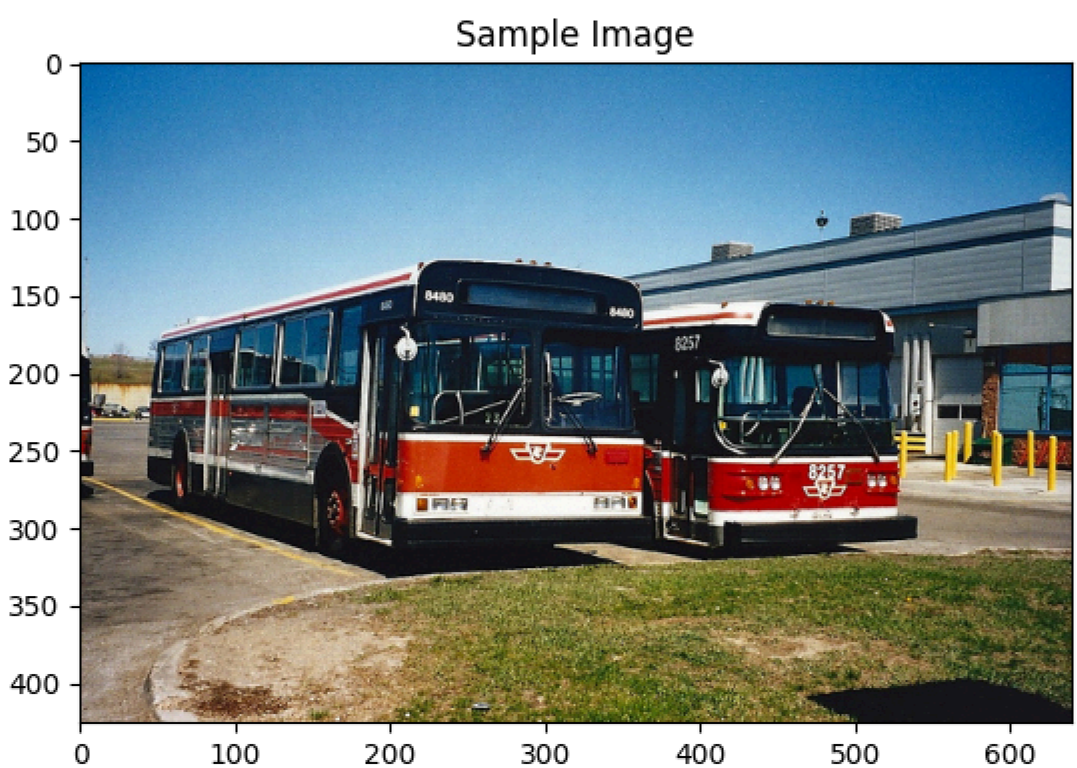



```
A group of people walking down a sidewalk
```

In [31]:
```
get_prediction()
```

A desk with a computer and a

In [32]:
```
get_prediction()
```

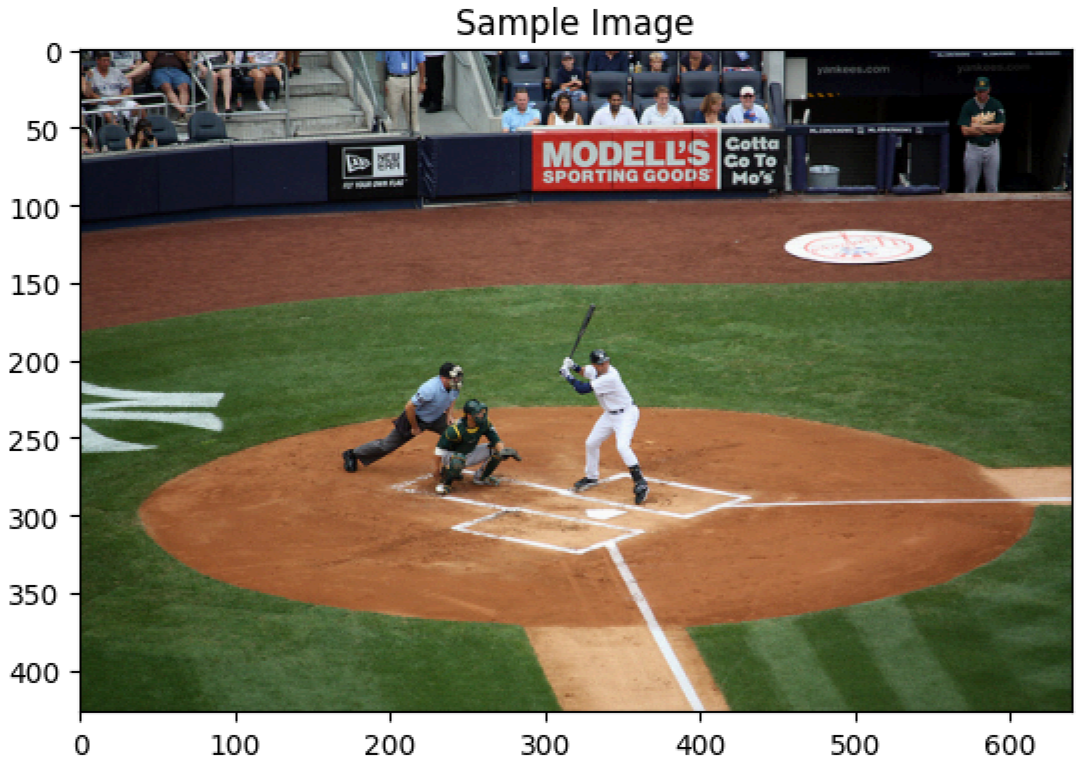


A bus is parked on the side of the street

The model could have performed better ...

Use the next two code cells to loop over captions. Save the notebook when you encounter two images with relatively inaccurate captions.

In [33]:
```
get_prediction()
```



A baseball player swinging a bat on a field

In [34]:
```
get_prediction()
```

A man is holding a large umbrella in the rain

In [ ]:
```

```