

# 3\_Inference

May 17, 2021

## 1 Computer Vision Nanodegree

### 1.1 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: - Section ??: Get Data Loader for Test Dataset - Section ??: Load Trained Models - Section ??: Finish the Sampler - Section ??: Clean up Captions - Section ??: 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 [3]: 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
            transforms.RandomCrop(224),             # get 224x224 crop from random loca
            transforms.RandomHorizontalFlip(),       # horizontally flip image with prob
            transforms.ToTensor(),                   # convert the PIL Image to a tensor
            transforms.Normalize((0.485, 0.456, 0.406), # normalize image for pre-trained m
                                (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 [4]: 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 [5]: import torch

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

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

### 1.1.1 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).

### 1.1.2 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 [7]: # 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

# TODO #2: Specify the saved models to load.
encoder_file = "encoder-3.pkl"
decoder_file = "decoder-3.pkl"

# TODO #3: Select appropriate values for the Python variables below.
embed_size = 512
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)
```

The autoreload extension is already loaded. To reload it, use:

```
%reload_ext autoreload
```

```
Out[7]: DecoderRNN(
  (wordEmbedding): Embedding(9955, 512)
  (lstm): LSTM(512, 512, batch_first=True)
  (linear): Linear(in_features=512, out_features=9955, 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 [9]: # 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 th
```

```
example output: [0, 3, 14, 21, 3, 535, 6, 214, 18, 1, 18, 1, 18, 1, 18, 1, 18, 1, 18, 1]
```

### ## 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 [12]: # TODO #4: Complete the function.
        def clean_sentence(output):
            sentence = ''
            for token in output:
                word = data_loader.dataset.vocab.idx2word[token]
                if token != 0:
                    if token == 1:
                        break
                    if token == 18:
```

```

        sentence = sentence + word
    else:
        sentence = sentence + ' ' + word
    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 [13]: sentence = clean_sentence(output)
        print('example sentence:', sentence)

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

```

example sentence: a kitchen with a stove and refrigerator.

### ## 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 [14]: 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)

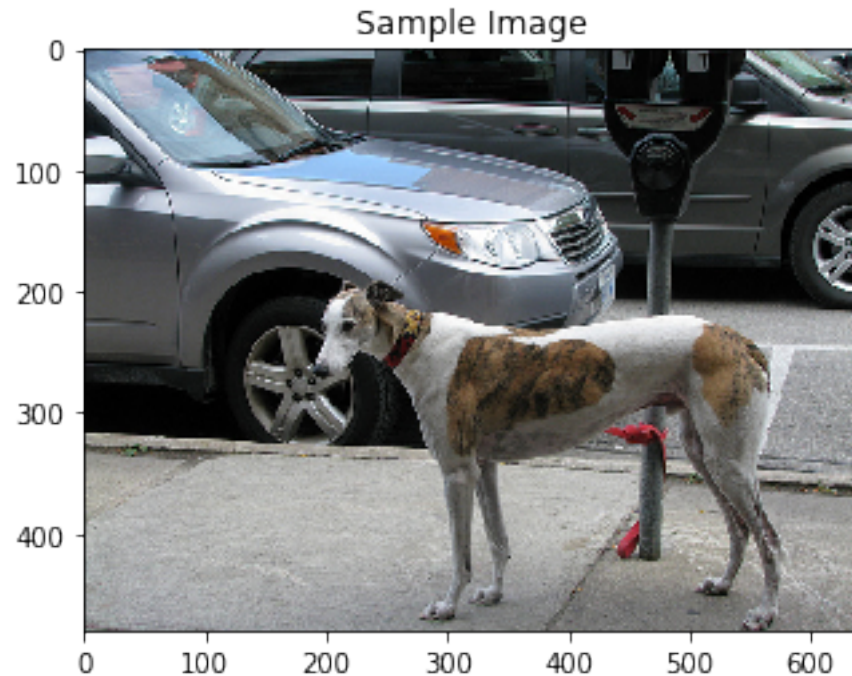
```

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

```

In [42]: get_prediction()

```



a dog is standing in the middle of a road.

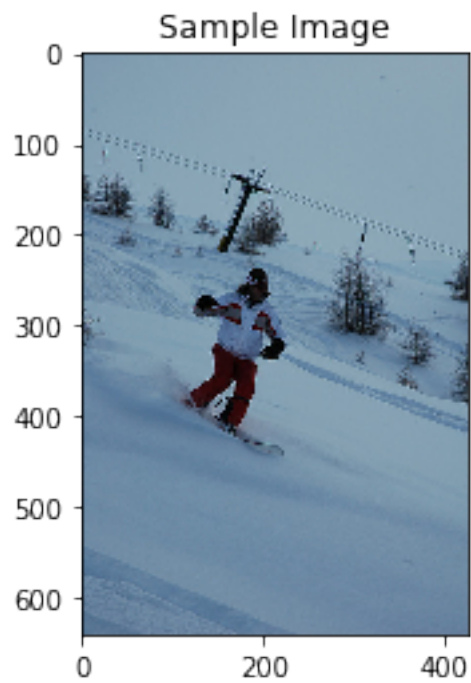
As the last task in this project, you will loop over the images until you find four image-caption pairs of interest: - Two should include image-caption pairs that show instances when the model performed well. - Two should highlight image-caption pairs that highlight instances where the model did not perform well.

Use the four code cells below to complete this task.

### 1.1.3 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 [33]: get_prediction()
```



a snowboarder is going down the slope.

```
In [21]: get_prediction()
```

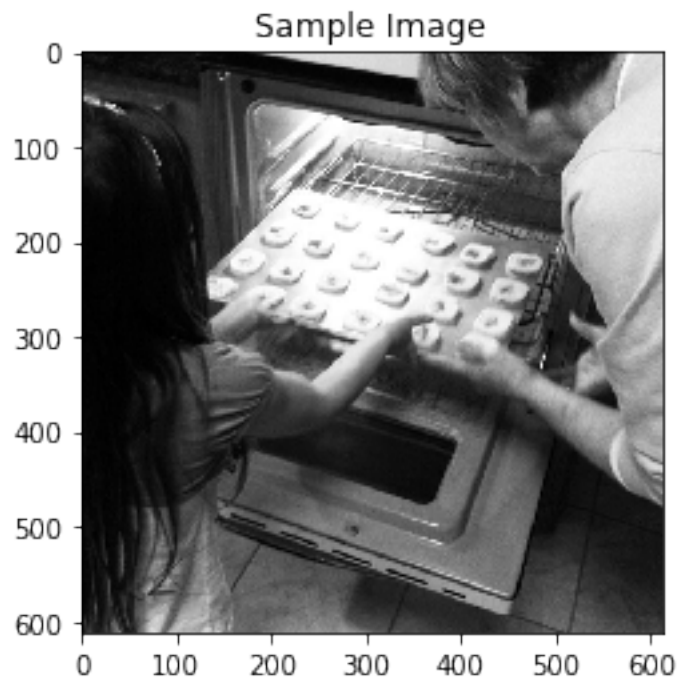


a man riding a bicycle on a street.

#### 1.1.4 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.

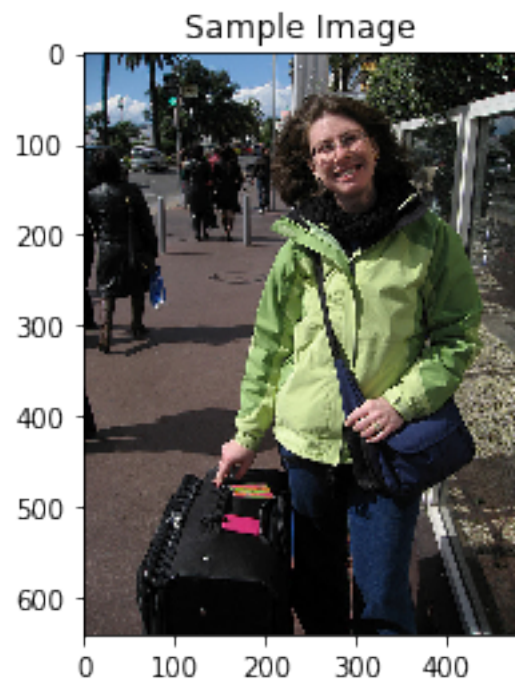
```
In [22]: get_prediction()
```



a woman sitting on a laptop computer on a table.

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





a man is standing in the snow with a luggage bag.

In [ ]: