landmark

January 20, 2022

1 Convolutional Neural Networks

1.1 Project: Write an Algorithm for Landmark Classification

In this notebook, some template code has already been provided for you, and you will need to implement additional functionality to successfully complete this project. You will not need to modify the included code beyond what is requested. Sections that begin with **'(IMPLEMENTATION)'** in the header indicate that the following block of code will require additional functionality which you must provide. Instructions will be provided for each section, and the specifics of the implementation are marked in the code block with a 'TODO' statement. Please be sure to read the instructions carefully!

> **Note**: Once you have completed all the code implementations, you need to finalize your work by exporting the Jupyter Notebook as an HTML document. Before exporting the notebook to HTML, all the code cells need to have been run so that reviewers can see the final implementation and output. You can then export the notebook by using the menu above and navigating to **File -> Download as -> HTML (.html)**. Include the finished document along with this notebook as your submission.

In addition to implementing code, there will be questions that you must answer which relate to the project and your implementation. Each section where you will answer a question is preceded by a **'Question X'** header. Carefully read each question and provide thorough answers in the following text boxes that begin with **'Answer:'**. Your project submission will be evaluated based on your answers to each of the questions and the implementation you provide.

> **Note:** Code and Markdown cells can be executed using the **Shift + Enter** keyboard shortcut. Markdown cells can be edited by double-clicking the cell to enter edit mode.

The rubric contains *optional* "Stand Out Suggestions" for enhancing the project beyond the minimum requirements. If you decide to pursue the "Stand Out Suggestions", you should include the code in this Jupyter notebook.

Step 0: Download Datasets and Install Python Modules

Note: if you are using the Udacity workspace, *YOU CAN SKIP THIS STEP*. The dataset can be found in the `/data` folder and all required Python modules have been installed in the workspace.

Download the landmark dataset. Unzip the folder and place it in this project's home directory, at the location `/landmark_images`.

Install the following Python modules: * cv2 * matplotlib * numpy * PIL * torch * torchvision

Step 1: Create a CNN to Classify Landmarks (from Scratch)

In this step, you will create a CNN that classifies landmarks. You must create your CNN *from scratch* (so, you can't use transfer learning *yet*!), and you must attain a test accuracy of at least 20%.

Although 20% may seem low at first glance, it seems more reasonable after realizing how difficult of a problem this is. Many times, an image that is taken at a landmark captures a fairly mundane image of an animal or plant, like in the following picture.

Just by looking at that image alone, would you have been able to guess that it was taken at the Haleakal National Park in Hawaii?

An accuracy of 20% is significantly better than random guessing, which would provide an accuracy of just 2%. In Step 2 of this notebook, you will have the opportunity to greatly improve accuracy by using transfer learning to create a CNN.

Remember that practice is far ahead of theory in deep learning. Experiment with many different architectures, and trust your intuition. And, of course, have fun!

1.1.1 (IMPLEMENTATION) Specify Data Loaders for the Landmark Dataset

Use the code cell below to create three separate data loaders: one for training data, one for validation data, and one for test data. Randomly split the images located at `landmark_images/train` to create the train and validation data loaders, and use the images located at `landmark_images/test` to create the test data loader.

Note: Remember that the dataset can be found at `/data/landmark_images/` in the workspace.

All three of your data loaders should be accessible via a dictionary named `loaders_scratch`. Your train data loader should be at `loaders_scratch['train']`, your validation data loader should be at `loaders_scratch['valid']`, and your test data loader should be at `loaders_scratch['test']`.

You may find this documentation on custom datasets to be a useful resource. If you are interested in augmenting your training and/or validation data, check out the wide variety of transforms!

In [1]:
```
### TODO: Write data loaders for training, validation, and test sets
## Specify appropriate transforms, and batch_sizes
import torch
import numpy as np

import torchvision
from torchvision import datasets, transforms
from torch.utils.data.sampler import SubsetRandomSampler

# Defining the data path
data_path = '/data/landmark_images/'
# Defining batch size to load
batch_size = 32
# % of train set to use in validation
valid_percent = 0.2

# Specifying the transforms
```

```
train_transforms = transforms.Compose([transforms.RandomRotation(30),
                                       transforms.RandomResizedCrop(224),
                                       transforms.RandomHorizontalFlip(),
                                       transforms.ToTensor(),
                                       #transforms.Normalize(0.5836, 0.1628) # zip argum
                                       transforms.Normalize((0.485, 0.456, 0.406), # Was
                                                            (0.229, 0.224, 0.225)) # Sol
                                      ])

test_transforms = transforms.Compose([transforms.Resize(255),
                                      transforms.CenterCrop(224),
                                      transforms.ToTensor(),
                                      #transforms.Normalize(0.5836, 0.1628)
                                      transforms.Normalize((0.485, 0.456, 0.406),
                                                           (0.229, 0.224, 0.225))
                                     ])

# Loading the datasets
train_data = datasets.ImageFolder(data_path + '/train', transform=train_transforms)
test_data = datasets.ImageFolder(data_path + '/test', transform=test_transforms)
print(len(train_data))
print(len(test_data))

# Spliting data to train and validation
num_train = len(train_data)
indices = list(range(num_train))
np.random.shuffle(indices)
split = int(np.floor(valid_percent * num_train))
train_idx, valid_idx = indices[split:], indices[:split]

# Defining samplers
train_sampler = SubsetRandomSampler(train_idx)
valid_sampler = SubsetRandomSampler(valid_idx)

# Preparing the data loaders
train_load_scr = torch.utils.data.DataLoader(train_data, batch_size=batch_size,
    sampler=train_sampler)
valid_load_scr = torch.utils.data.DataLoader(train_data, batch_size=batch_size,
    sampler=valid_sampler)
test_load_scr = torch.utils.data.DataLoader(test_data, batch_size=batch_size)

# Compressing in a dict
loaders_scratch = {'train': train_load_scr, 'valid': valid_load_scr, 'test': test_load_s
```

```
4996
1250
```

Question 1: Describe your chosen procedure for preprocessing the data. - How does your code resize the images (by cropping, stretching, etc)? What size did you pick for the input tensor, and why? - Did you decide to augment the dataset? If so, how (through translations, flips, rotations, etc)? If not, why not?

Answer: - I chose this 224 size because I read in discussions online that was a good resolution to train it without using to much memory and it was also the size used for the VGG transfered NN, so some time saved there just copying the code.

- My code resize the train data to 224x224 with a random resized crop included. It also rotate the data in a random direction, apart from flipping it horizontally. Then, the 3 rgb channels get normalized and the data converted to tensor.
- Yes, with this augmentations mentioned above in the training data, I tried to get higher accuracy both on the scratch NN and the transfered one. As suggested in the *Stand Out Suggestions 1*
- For the test set I just used a stretching resizer and the a center crop to 224 with no augmentations.

1.1.2 (IMPLEMENTATION) Visualize a Batch of Training Data

Use the code cell below to retrieve a batch of images from your train data loader, display at least 5 images simultaneously, and label each displayed image with its class name (e.g., "Golden Gate Bridge").

Visualizing the output of your data loader is a great way to ensure that your data loading and preprocessing are working as expected.

In [2]:
```
import matplotlib.pyplot as plt
%matplotlib inline

## TODO: visualize a batch of the train data loader
# Obtaining one batch of training images
dataiter = iter(loaders_scratch['train'])
images, labels = dataiter.next()
# print(images[0].mean(),images[0].std())
print(images.size())
images = images.numpy() # convert images to numpy for display
## the class names can be accessed at the `classes` attribute
## of your dataset object (e.g., `train_dataset.classes`)
classes = train_data.classes
# print(len(classes))
# Plotting the images in the batch, along with the corresponding labels
fig = plt.figure(figsize=(25, 4))
for idx in np.arange(5):
    ax = fig.add_subplot(2, 8, idx+1, xticks=[], yticks=[])
    image = images[idx] / 2 + 0.5 # @Tejas J forum tip
    plt.imshow(np.transpose(image, (1, 2, 0)).clip(0,1)) # okey clip
    ax.set_title(classes[labels[idx]])
```

```
torch.Size([32, 3, 224, 224])
```

1.1.3 Initialize use_cuda variable

```
In [3]: # useful variable that tells us whether we should use the GPU
        use_cuda = torch.cuda.is_available()
```

1.1.4 (IMPLEMENTATION) Specify Loss Function and Optimizer

Use the next code cell to specify a loss function and optimizer. Save the chosen loss function as `criterion_scratch`, and fill in the function `get_optimizer_scratch` below.

```
In [4]: ## TODO: select loss function
        import torch.nn as nn
        import torch.optim as optim

        criterion_scratch = nn.CrossEntropyLoss()

        def get_optimizer_scratch(model):
            ## TODO: select and return an optimizer
            optimizer = optim.Adamax(model.parameters(), lr=0.003)
            return optimizer
```

1.1.5 (IMPLEMENTATION) Model Architecture

Create a CNN to classify images of landmarks. Use the template in the code cell below.

```
In [5]: import torch.nn.functional as F
        # define the CNN architecture
        class Net(nn.Module):
            ## TODO: choose an architecture, and complete the class
            def __init__(self):
                super(Net, self).__init__()
                ## Define layers of a CNN
                self.conv1 = nn.Conv2d(3, 16, 3, padding=1)
                self.conv2 = nn.Conv2d(16, 32, 3, padding=1)
                self.conv3 = nn.Conv2d(32, 64, 3, padding=1)
                self.pool = nn.MaxPool2d(2, 2)

                self.fc1 = nn.Linear(64 * 28 * 28, 500)
                self.fc2 = nn.Linear(500, 50)
```

```
        self.dropout = nn.Dropout(0.20)

    def forward(self, x):
        ## Define forward behavior
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.pool(F.relu(self.conv3(x)))
        # print(x.shape)
        x = x.view(-1, 64 * 28 * 28)

        x = self.dropout(x)
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)

        return x

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

# instantiate the CNN
model_scratch = Net()
print(model_scratch)
# move tensors to GPU if CUDA is available
if use_cuda:
    model_scratch.cuda()
```

```
Net(
  (conv1): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc1): Linear(in_features=50176, out_features=500, bias=True)
  (fc2): Linear(in_features=500, out_features=50, bias=True)
  (dropout): Dropout(p=0.2)
)
```

Question 2: Outline the steps you took to get to your final CNN architecture and your reasoning at each step.

Answer:

I didnt want to use a really big NN so it was going to need to much GPU time for train. So I chose a network similar to which I have used in my CIFAR exercise due to it was a similar multi label task, in which I reached a good accuracy. Then I arranged the shapes of the data used so it will work good using the print(x.shape).

Everything worked fine so quickly and I was training it to 40 epochs in a moment. Then the val loss stop decreasing to I stopped the training with a 44% accu.

1.1.6 (IMPLEMENTATION) Implement the Training Algorithm

Implement your training algorithm in the code cell below. Save the final model parameters at the filepath stored in the variable `save_path`.

```
In [6]: from tqdm import tqdm_notebook as tqdm
        def train(n_epochs, loaders, model, optimizer, criterion, use_cuda, save_path):
            """returns trained model"""
            # initialize tracker for minimum validation loss
            valid_loss_min = np.Inf # Need a change for updating this whenever u want to continu

            for epoch in range(1, n_epochs+1):
                # initialize variables to monitor training and validation loss
                train_loss = 0.0
                valid_loss = 0.0

                ###################
                # train the model #
                ###################
                # set the module to training mode
                model.train()
                for batch_idx, (data, target) in tqdm(enumerate(loaders['train'])):
                    # move to GPU
                    if use_cuda:
                        data, target = data.cuda(), target.cuda()

                    ## TODO: find the loss and update the model parameters accordingly
                    ## record the average training loss, using something like
                    ## train_loss = train_loss + ((1 / (batch_idx + 1)) * (loss.data.item() - tr

                    # First, reset the optimizer
                    optimizer.zero_grad()
                    # Second, pass the data through the model
                    output = model(data)
                    # Calculate the loss
                    loss = criterion(output, target)
                    # Backpropagate the losss
                    loss.backward()
                    # Optimize the parameters
                    optimizer.step()
                    # Update the train loss
                    train_loss = train_loss + (1 / (batch_idx + 1)) * (loss.data.item() - train_

                ######################
                # validate the model #
                ######################
                # set the model to evaluation mode
                model.eval()
                for batch_idx, (data, target) in tqdm(enumerate(loaders['valid'])):
```

```
            # move to GPU
            if use_cuda:
                data, target = data.cuda(), target.cuda()

            ## TODO: update average validation loss

            # Pass the input to the model
            output = model(data)
            # Calculate the loss
            loss = criterion(output, target)
            # Update the validation loss
            valid_loss = valid_loss + (1 / (batch_idx + 1)) * (loss.data.item() - valid_


        # print training/validation statistics
        print('Epoch: {} \tTraining Loss: {:.6f} \tValidation Loss: {:.6f}'.format(
            epoch,
            train_loss,
            valid_loss
            ))

        ## TODO: if the validation loss has decreased, save the model at the filepath st
        if valid_loss <= valid_loss_min:
            print('Validation loss decreased ({:.6f} --> {:.6f}).  Saving model ...'.for
            valid_loss_min,
            valid_loss))
            torch.save(model.state_dict(), save_path)
            valid_loss_min = valid_loss


    return model
```

1.1.7 (IMPLEMENTATION) Experiment with the Weight Initialization

Use the code cell below to define a custom weight initialization, and then train with your weight initialization for a few epochs. Make sure that neither the training loss nor validation loss is nan.

Later on, you will be able to see how this compares to training with PyTorch's default weight initialization.

```
In [19]: def custom_weight_init(m):
             ## TODO: implement a weight initialization strategy
             classname = m.__class__.__name__
             if classname.find('Linear') != -1:
                 n = m.in_features
                 y = (1.01234/np.sqrt(n))
                 m.weight.data.normal_(0.0012321, y)
                 m.bias.data.fill_(0.04129)
             elif classname.find('Conv2d') != -1:
```

```
            torch.nn.init.kaiming_normal_(m.weight)
            if m.bias is not None:
                m.bias.data.fill_(0.014184)


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

    model_scratch.apply(custom_weight_init)
    model_scratch = train(20, loaders_scratch, model_scratch, get_optimizer_scratch(model_s
                          criterion_scratch, use_cuda, 'ignore.pt')
```

```
HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))

Epoch: 1         Training Loss: 9.263409         Validation Loss: 3.921264
Validation loss decreased (inf --> 3.921264).  Saving model ...

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))

Epoch: 2         Training Loss: 3.911889         Validation Loss: 3.915108
Validation loss decreased (3.921264 --> 3.915108).  Saving model ...

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))
```

```
Epoch: 3        Training Loss: 3.908436        Validation Loss: 3.912008
Validation loss decreased (3.915108 --> 3.912008).  Saving model ...


HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))




HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))


Epoch: 4        Training Loss: 3.904149        Validation Loss: 3.894633
Validation loss decreased (3.912008 --> 3.894633).  Saving model ...


HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))




HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))


Epoch: 5        Training Loss: 3.884599        Validation Loss: 3.875511
Validation loss decreased (3.894633 --> 3.875511).  Saving model ...


HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))




HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))


Epoch: 6        Training Loss: 3.856737        Validation Loss: 3.845820
Validation loss decreased (3.875511 --> 3.845820).  Saving model ...


HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))
```

```
HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))


Epoch: 7         Training Loss: 3.827743         Validation Loss: 3.816581
Validation loss decreased (3.845820 --> 3.816581).  Saving model ...

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))



HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))


Epoch: 8         Training Loss: 3.780379         Validation Loss: 3.752029
Validation loss decreased (3.816581 --> 3.752029).  Saving model ...

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))



HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))


Epoch: 9         Training Loss: 3.758815         Validation Loss: 3.744243
Validation loss decreased (3.752029 --> 3.744243).  Saving model ...

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))



HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))


Epoch: 10         Training Loss: 3.730066         Validation Loss: 3.701447
Validation loss decreased (3.744243 --> 3.701447).  Saving model ...
```

```
HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))
```

```
---------------------------------------------------------------------------

KeyboardInterrupt                         Traceback (most recent call last)

<ipython-input-19-ca9ac84c34a2> in <module>()
     17 model_scratch.apply(custom_weight_init)
     18 model_scratch = train(20, loaders_scratch, model_scratch, get_optimizer_scratch(mode
---> 19                       criterion_scratch, use_cuda, 'ignore.pt')


<ipython-input-6-9fcd06033abe> in train(n_epochs, loaders, model, optimizer, criterion,
     15         # set the module to training mode
     16         model.train()
---> 17         for batch_idx, (data, target) in tqdm(enumerate(loaders['train'])):
     18             # move to GPU
     19             if use_cuda:


/opt/conda/lib/python3.6/site-packages/tqdm/_tqdm_notebook.py in __iter__(self, *args, *
    185     def __iter__(self, *args, **kwargs):
    186         try:
--> 187             for obj in super(tqdm_notebook, self).__iter__(*args, **kwargs):
    188                 # return super(tqdm...) will not catch exception
    189                 yield obj


/opt/conda/lib/python3.6/site-packages/tqdm/_tqdm.py in __iter__(self)
    831 """, fp_write=getattr(self.fp, 'write', sys.stderr.write))
    832
--> 833             for obj in iterable:
    834                 yield obj
    835                 # Update and print the progressbar.


/opt/conda/lib/python3.6/site-packages/torch/utils/data/dataloader.py in __next__(self)
    262         if self.num_workers == 0:  # same-process loading
    263             indices = next(self.sample_iter)  # may raise StopIteration
--> 264             batch = self.collate_fn([self.dataset[i] for i in indices])
    265             if self.pin_memory:
    266                 batch = pin_memory_batch(batch)


/opt/conda/lib/python3.6/site-packages/torch/utils/data/dataloader.py in <listcomp>(.0)
    262         if self.num_workers == 0:  # same-process loading
```

```
    263             indices = next(self.sample_iter)  # may raise StopIteration
--> 264             batch = self.collate_fn([self.dataset[i] for i in indices])
    265             if self.pin_memory:
    266                 batch = pin_memory_batch(batch)


/opt/conda/lib/python3.6/site-packages/torchvision-0.2.1-py3.6.egg/torchvision/datasets/
     99         """
    100         path, target = self.samples[index]
--> 101         sample = self.loader(path)
    102         if self.transform is not None:
    103             sample = self.transform(sample)


/opt/conda/lib/python3.6/site-packages/torchvision-0.2.1-py3.6.egg/torchvision/datasets/
    145         return accimage_loader(path)
    146     else:
--> 147         return pil_loader(path)
    148
    149


/opt/conda/lib/python3.6/site-packages/torchvision-0.2.1-py3.6.egg/torchvision/datasets/
    128     with open(path, 'rb') as f:
    129         img = Image.open(f)
--> 130         return img.convert('RGB')
    131
    132


/opt/conda/lib/python3.6/site-packages/PIL/Image.py in convert(self, mode, matrix, dithe
    890         """
    891
--> 892         self.load()
    893
    894         if not mode and self.mode == "P":


/opt/conda/lib/python3.6/site-packages/PIL/ImageFile.py in load(self)
    233
    234                             b = b + s
--> 235                             n, err_code = decoder.decode(b)
    236                             if n < 0:
    237                                 break


KeyboardInterrupt:
```

1.1.8 (IMPLEMENTATION) Train and Validate the Model

Run the next code cell to train your model.

```
In [7]: ## TODO: you may change the number of epochs if you'd like,
        ## but changing it is not required
        num_epochs = 100

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

        # function to re-initialize a model with pytorch's default weight initialization
        def default_weight_init(m):
            reset_parameters = getattr(m, 'reset_parameters', None)
            if callable(reset_parameters):
                m.reset_parameters()

        # reset the model parameters
        model_scratch.apply(default_weight_init)

        # train the model
        model_scratch.load_state_dict(torch.load('model_scratch.pt'))
        model_scratch = train(num_epochs, loaders_scratch, model_scratch, get_optimizer_scratch(
                              criterion_scratch, use_cuda, 'model_scratch.pt')
```

```
HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))



HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))


Epoch: 1         Training Loss: 4.075562         Validation Loss: 3.888436
Validation loss decreased (inf --> 3.888436).  Saving model ...


HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))



HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))


Epoch: 2         Training Loss: 3.839315         Validation Loss: 3.774903
Validation loss decreased (3.888436 --> 3.774903).  Saving model ...
```

```
HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))

Epoch: 3         Training Loss: 3.747534         Validation Loss: 3.737884
Validation loss decreased (3.774903 --> 3.737884).  Saving model ...

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))

Epoch: 4         Training Loss: 3.691145         Validation Loss: 3.653192
Validation loss decreased (3.737884 --> 3.653192).  Saving model ...

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))

Epoch: 5         Training Loss: 3.628869         Validation Loss: 3.545792
Validation loss decreased (3.653192 --> 3.545792).  Saving model ...

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))
```

```
Epoch: 6        Training Loss: 3.561285        Validation Loss: 3.534850
Validation loss decreased (3.545792 --> 3.534850).  Saving model ...


HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))




HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))


Epoch: 7        Training Loss: 3.464760        Validation Loss: 3.374228
Validation loss decreased (3.534850 --> 3.374228).  Saving model ...


HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))




HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))


Epoch: 8        Training Loss: 3.406613        Validation Loss: 3.353082
Validation loss decreased (3.374228 --> 3.353082).  Saving model ...


HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))




HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))


Epoch: 9        Training Loss: 3.346248        Validation Loss: 3.275855
Validation loss decreased (3.353082 --> 3.275855).  Saving model ...


HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))
```

```
HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))

Epoch: 10         Training Loss: 3.281782         Validation Loss: 3.225167
Validation loss decreased (3.275855 --> 3.225167).  Saving model ...

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))

Epoch: 11         Training Loss: 3.188290         Validation Loss: 3.200020
Validation loss decreased (3.225167 --> 3.200020).  Saving model ...

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))

Epoch: 12         Training Loss: 3.145452         Validation Loss: 3.114134
Validation loss decreased (3.200020 --> 3.114134).  Saving model ...

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))

Epoch: 13         Training Loss: 3.070233         Validation Loss: 3.113539
Validation loss decreased (3.114134 --> 3.113539).  Saving model ...
```

```
HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))

Epoch: 14         Training Loss: 3.004590         Validation Loss: 3.025699
Validation loss decreased (3.113539 --> 3.025699).  Saving model ...

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))

Epoch: 15         Training Loss: 2.983421         Validation Loss: 3.092224

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))

Epoch: 16         Training Loss: 2.903275         Validation Loss: 2.942105
Validation loss decreased (3.025699 --> 2.942105).  Saving model ...

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))
```

```
Epoch: 17         Training Loss: 2.827653         Validation Loss: 2.931641
Validation loss decreased (2.942105 --> 2.931641).  Saving model ...
```

```
HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))
```

```
HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))
```

```
Epoch: 18         Training Loss: 2.804891         Validation Loss: 2.940915
```

```
HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))
```

```
HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))
```

```
Epoch: 19         Training Loss: 2.747423         Validation Loss: 2.882969
Validation loss decreased (2.931641 --> 2.882969).  Saving model ...
```

```
HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))
```

```
HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))
```

```
Epoch: 20         Training Loss: 2.719472         Validation Loss: 2.811672
Validation loss decreased (2.882969 --> 2.811672).  Saving model ...
```

```
HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))
```

```
HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))

Epoch: 21         Training Loss: 2.662407         Validation Loss: 2.844345

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))

Epoch: 22         Training Loss: 2.616170         Validation Loss: 2.731724
Validation loss decreased (2.811672 --> 2.731724).  Saving model ...

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))

Epoch: 23         Training Loss: 2.588323         Validation Loss: 2.787043

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))

Epoch: 24         Training Loss: 2.552168         Validation Loss: 2.741770

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))
```

```
HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))


Epoch: 25         Training Loss: 2.527035         Validation Loss: 2.790179

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))



HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))


Epoch: 26         Training Loss: 2.465996         Validation Loss: 2.724202
Validation loss decreased (2.731724 --> 2.724202).  Saving model ...

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))



HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))


Epoch: 27         Training Loss: 2.500872         Validation Loss: 2.685225
Validation loss decreased (2.724202 --> 2.685225).  Saving model ...

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))



HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))


Epoch: 28         Training Loss: 2.446638         Validation Loss: 2.685790
```

```
HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))



HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))


Epoch: 29         Training Loss: 2.385067         Validation Loss: 2.704247

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))



HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))


Epoch: 30         Training Loss: 2.369251         Validation Loss: 2.655194
Validation loss decreased (2.685225 --> 2.655194).  Saving model ...

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))



HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))


Epoch: 31         Training Loss: 2.336515         Validation Loss: 2.615879
Validation loss decreased (2.655194 --> 2.615879).  Saving model ...

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))



HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))
```

```
Epoch: 32        Training Loss: 2.288093        Validation Loss: 2.586198
Validation loss decreased (2.615879 --> 2.586198).  Saving model ...

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))

Epoch: 33        Training Loss: 2.250560        Validation Loss: 2.629656

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))

Epoch: 34        Training Loss: 2.212519        Validation Loss: 2.662029

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))

Epoch: 35        Training Loss: 2.200109        Validation Loss: 2.676562

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))
```

```
HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))


Epoch: 36         Training Loss: 2.184821         Validation Loss: 2.535587
Validation loss decreased (2.586198 --> 2.535587).  Saving model ...


HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))


HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))


Epoch: 37         Training Loss: 2.123658         Validation Loss: 2.652949


HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))


HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))


Epoch: 38         Training Loss: 2.137784         Validation Loss: 2.654183


HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))


HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))


Epoch: 39         Training Loss: 2.077990         Validation Loss: 2.660160


HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))
```

```
HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))
```

```
Epoch: 40        Training Loss: 2.090369        Validation Loss: 2.511839
Validation loss decreased (2.535587 --> 2.511839).  Saving model ...
```

```
HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))
```

```
HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))
```

```
Epoch: 41        Training Loss: 2.049895        Validation Loss: 2.651264
```

```
HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))
```

```
    ---------------------------------------------------------------------------

    KeyboardInterrupt                         Traceback (most recent call last)

    <ipython-input-7-234244f91f3a> in <module>()
     16 # train the model
     17 model_scratch = train(num_epochs, loaders_scratch, model_scratch, get_optimizer_scra
---> 18                       criterion_scratch, use_cuda, 'model_scratch.pt')


    <ipython-input-6-9fcd06033abe> in train(n_epochs, loaders, model, optimizer, criterion,
     35             optimizer.step()
     36             # Update the train loss
---> 37             train_loss = train_loss + (1 / (batch_idx + 1)) * (loss.data.item() - tr
     38
     39         #####################


    KeyboardInterrupt:
```

1.1.9 (IMPLEMENTATION) Test the Model

Run the code cell below to try out your model on the test dataset of landmark images. Run the code cell below to calculate and print the test loss and accuracy. Ensure that your test accuracy is greater than 20%.

```
In [7]: def test(loaders, model, criterion, use_cuda):

            # monitor test loss and accuracy
            test_loss = 0.
            correct = 0.
            total = 0.

            # set the module to evaluation mode
            model.eval()

            for batch_idx, (data, target) in enumerate(loaders['test']):
                # move to GPU
                if use_cuda:
                    data, target = data.cuda(), target.cuda()
                # forward pass: compute predicted outputs by passing inputs to the model
                output = model(data)
                # calculate the loss
                loss = criterion(output, target)
                # update average test loss
                test_loss = test_loss + ((1 / (batch_idx + 1)) * (loss.data.item() - test_loss))
                # convert output probabilities to predicted class
                pred = output.data.max(1, keepdim=True)[1]
                # compare predictions to true label
                correct += np.sum(np.squeeze(pred.eq(target.data.view_as(pred))).cpu().numpy())
                total += data.size(0)

            print('Test Loss: {:.6f}\n'.format(test_loss))

            print('\nTest Accuracy: %2d%% (%2d/%2d)' % (
                100. * correct / total, correct, total))
```

```
In [8]: # load the model that got the best validation accuracy
        model_scratch.load_state_dict(torch.load('model_scratch.pt'))
        test(loaders_scratch, model_scratch, criterion_scratch, use_cuda)
```

```
Test Loss: 2.305133


Test Accuracy: 44% (551/1250)
```

Step 2: Create a CNN to Classify Landmarks (using Transfer Learning)

You will now use transfer learning to create a CNN that can identify landmarks from images. Your CNN must attain at least 60% accuracy on the test set.

1.1.10 (IMPLEMENTATION) Specify Data Loaders for the Landmark Dataset

Use the code cell below to create three separate data loaders: one for training data, one for validation data, and one for test data. Randomly split the images located at `landmark_images/train` to create the train and validation data loaders, and use the images located at `landmark_images/test` to create the test data loader.

All three of your data loaders should be accessible via a dictionary named `loaders_transfer`. Your train data loader should be at `loaders_transfer['train']`, your validation data loader should be at `loaders_transfer['valid']`, and your test data loader should be at `loaders_transfer['test']`.

If you like, **you are welcome to use the same data loaders from the previous step**, when you created a CNN from scratch.

```
In [8]: ### TODO: Write data loaders for training, validation, and test sets
        ## Specify appropriate transforms, and batch_sizes
        # Defining the data path
        data_path = '/data/landmark_images/'
        # Defining batch size to load
        batch_size = 32
        # % of train set to use in validation
        valid_percent = 0.2

        # Specifying the transforms
        train_transforms = transforms.Compose([transforms.RandomRotation(30),
                                               transforms.RandomResizedCrop(224),
                                               transforms.RandomHorizontalFlip(),
                                               transforms.ToTensor(),
                                               #transforms.Normalize(0.5836, 0.1628) # zip argum
                                               transforms.Normalize((0.485, 0.456, 0.406), # Was
                                                                    (0.229, 0.224, 0.225)) # Sol
                                              ])

        test_transforms = transforms.Compose([transforms.Resize(255),
                                              transforms.CenterCrop(224),
                                              transforms.ToTensor(),
                                              #transforms.Normalize(0.5836, 0.1628)
                                              transforms.Normalize((0.485, 0.456, 0.406),
                                                                   (0.229, 0.224, 0.225))
                                             ])

        # Loading the datasets
        train_data = datasets.ImageFolder(data_path + '/train', transform=train_transforms)
        test_data = datasets.ImageFolder(data_path + '/test', transform=test_transforms)
        print(len(train_data))
```

```
print(len(test_data))

# Spliting data to train and validation
num_train = len(train_data)
indices = list(range(num_train))
np.random.shuffle(indices)
split = int(np.floor(valid_percent * num_train))
train_idx, valid_idx = indices[split:], indices[:split]

# Defining samplers
train_sampler = SubsetRandomSampler(train_idx)
valid_sampler = SubsetRandomSampler(valid_idx)

# Preparing the data loaders
train_load_tr = torch.utils.data.DataLoader(train_data, batch_size=batch_size,
    sampler=train_sampler)
valid_load_tr = torch.utils.data.DataLoader(train_data, batch_size=batch_size,
    sampler=valid_sampler)
test_load_tr = torch.utils.data.DataLoader(test_data, batch_size=batch_size)

loaders_transfer = {'train': train_load_tr, 'valid': valid_load_tr, 'test': test_load_tr
```

```
4996
1250
```

1.1.11 (IMPLEMENTATION) Specify Loss Function and Optimizer

Use the next code cell to specify a loss function and optimizer. Save the chosen loss function as `criterion_transfer`, and fill in the function `get_optimizer_transfer` below.

```
In [9]: ## TODO: select loss function
        from torch import optim
        criterion_transfer = nn.CrossEntropyLoss()

        def get_optimizer_transfer(model):
            ## TODO: select and return optimizer
            optimizer = optim.Adamax(model.classifier.parameters(), lr=0.003)
            return optimizer
```

1.1.12 (IMPLEMENTATION) Model Architecture

Use transfer learning to create a CNN to classify images of landmarks. Use the code cell below, and save your initialized model as the variable `model_transfer`.

```
In [10]: ## TODO: Specify model architecture
         import torch.nn as nn
         from torchvision import models
         from collections import OrderedDict
```

```
model_transfer = models.vgg19(pretrained=True)

for param in model_transfer.features.parameters():
    param.requires_grad_(False)

print(model_transfer)
#print(model_transfer.fc.in_features)  # Inception and ResNet50 werent working properly
#print(model_transfer.fc.out_features)

#classifier = nn.Sequential(OrderedDict([ # VGG is more comfortable, I didnt even need
#                           ('fc1', nn.Linear(model_transfer.fc.in_features, 1000)),
#                           ('relu', nn.ReLU()),
#                           ('drop', nn.Dropout(p=0.2)),
#                           ('fc2', nn.Linear(1000, 500)),
#                           ('relu2', nn.ReLU()),
#                           ('drop2', nn.Dropout(p=0.2)),
#                           ('fc3', nn.Linear(500, 50)),
#                           ('output', nn.Softmax(dim=1))
#                           ]))

model_transfer.classifier[6] = nn.Linear(in_features=4096, out_features=50)
#for param in model_transfer.fc.parameters(): # Code used with ResNet50 trying to unfre
#    param.requires_grad_(True)

print(model_transfer)

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

if use_cuda:
    model_transfer = model_transfer.cuda()
```

```
Downloading: "https://download.pytorch.org/models/vgg19-dcbb9e9d.pth" to /root/.torch/models/vgg
100%|| 574673361/574673361 [00:06<00:00, 94410542.56it/s]


VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
```

```
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace)
    (16): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (17): ReLU(inplace)
    (18): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (19): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace)
    (23): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (24): ReLU(inplace)
    (25): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (26): ReLU(inplace)
    (27): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace)
    (30): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (31): ReLU(inplace)
    (32): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (33): ReLU(inplace)
    (34): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (35): ReLU(inplace)
    (36): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace)
    (2): Dropout(p=0.5)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace)
    (5): Dropout(p=0.5)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
```

```
    (8): ReLU(inplace)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace)
    (16): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (17): ReLU(inplace)
    (18): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (19): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace)
    (23): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (24): ReLU(inplace)
    (25): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (26): ReLU(inplace)
    (27): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace)
    (30): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (31): ReLU(inplace)
    (32): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (33): ReLU(inplace)
    (34): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (35): ReLU(inplace)
    (36): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace)
    (2): Dropout(p=0.5)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace)
    (5): Dropout(p=0.5)
    (6): Linear(in_features=4096, out_features=50, bias=True)
  )
)
```

Question 3: Outline the steps you took to get to your final CNN architecture and your reasoning at each step. Describe why you think the architecture is suitable for the current problem.

Answer:

- I googled for the best available models in Torch 0.4.0 which is the version allowed in this workspace. And I tried the highest accu nets. First InceptionV3 was giving me a problem I wasn't managing to fix, so I decided to change to a ResNet50.

- With the ResNet I had 2 problems, first I was having problems with freezing some parameters but getting all frozen. Finally I managed to fix it but the net didnt learn well so I got a 30% accu what wasn't enough.

- Finally, I changed to the VGG19 which was easier to build, freeze and train. Changed the classifier so it fit my output classes and trained it for some epochs. It easily got pretty above the minimum accu.

- I think VGG is suitable because it was trained in a multilabel task with 1000 classes so the initial layers learned well how to extract the images features from a huge a varied dataset as ImageNet. I suppose it was easy to generalize to simple landmark images.

1.1.13 (IMPLEMENTATION) Train and Validate the Model

Train and validate your model in the code cell below. Save the final model parameters at filepath `'model_transfer.pt'`.

```
In [11]: # TODO: train the model and save the best model parameters at filepath 'model_transfer.
         num_epochs = 15

         model_transfer.load_state_dict(torch.load('model_transfer.pt')) # For retraining after
         model_transfer = train(num_epochs, loaders_transfer, model_transfer, get_optimizer_tran
                               criterion_transfer, use_cuda, 'model_transfer.pt')
```

```
HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))

Epoch: 1         Training Loss: 1.885878         Validation Loss: 1.233629
Validation loss decreased (inf --> 1.233629).  Saving model ...

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))

Epoch: 2         Training Loss: 1.560930         Validation Loss: 1.258994
```

```
HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))

Epoch: 3         Training Loss: 1.402346         Validation Loss: 1.154829
Validation loss decreased (1.233629 --> 1.154829).  Saving model ...

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))

Epoch: 4         Training Loss: 1.331803         Validation Loss: 1.103056
Validation loss decreased (1.154829 --> 1.103056).  Saving model ...

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))

Epoch: 5         Training Loss: 1.340903         Validation Loss: 1.093652
Validation loss decreased (1.103056 --> 1.093652).  Saving model ...

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))

HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))
```

```
Epoch: 6        Training Loss: 1.291270         Validation Loss: 1.073554
Validation loss decreased (1.093652 --> 1.073554).  Saving model ...
```

```
HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))
```

```
HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))
```

```
Epoch: 7        Training Loss: 1.221157         Validation Loss: 1.084225
```

```
HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))
```

```
HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))
```

```
Epoch: 8        Training Loss: 1.190087         Validation Loss: 1.105835
```

```
HBox(children=(IntProgress(value=1, bar_style='info', max=1), HTML(value='')))
```

```
    ---------------------------------------------------------------------------

    KeyboardInterrupt                         Traceback (most recent call last)

    <ipython-input-11-21b36c5e178c> in <module>()
      4 model_transfer.load_state_dict(torch.load('model_transfer.pt')) # For retraining aft
      5 model_transfer = train(num_epochs, loaders_transfer, model_transfer, get_optimizer_t
----> 6                        criterion_transfer, use_cuda, 'model_transfer.pt')


    <ipython-input-6-9fcd06033abe> in train(n_epochs, loaders, model, optimizer, criterion,
     35             optimizer.step()
     36             # Update the train loss
---> 37             train_loss = train_loss + (1 / (batch_idx + 1)) * (loss.data.item() - tr
```

```
38
39         #####################


KeyboardInterrupt:
```

1.1.14 (IMPLEMENTATION) Test the Model

Try out your model on the test dataset of landmark images. Use the code cell below to calculate and print the test loss and accuracy. Ensure that your test accuracy is greater than 60%.

```
In [12]: #-#-# Do NOT modify the code below this line. #-#-#

         # load the model that got the best validation accuracy
         model_transfer.load_state_dict(torch.load('model_transfer.pt'))
         test(loaders_transfer, model_transfer, criterion_transfer, use_cuda)
```

```
Test Loss: 0.923124


Test Accuracy: 75% (947/1250)
```

Step 3: Write Your Landmark Prediction Algorithm
Great job creating your CNN models! Now that you have put in all the hard work of creating accurate classifiers, let's define some functions to make it easy for others to use your classifiers.

1.1.15 (IMPLEMENTATION) Write Your Algorithm, Part 1

Implement the function `predict_landmarks`, which accepts a file path to an image and an integer k, and then predicts the **top k most likely landmarks**. You are **required** to use your transfer learned CNN from Step 2 to predict the landmarks.

An example of the expected behavior of `predict_landmarks`:

```
>>> predicted_landmarks = predict_landmarks('example_image.jpg', 3)
>>> print(predicted_landmarks)
['Golden Gate Bridge', 'Brooklyn Bridge', 'Sydney Harbour Bridge']
```

```
In [13]: import cv2
         from PIL import Image

         ## the class names can be accessed at the `classes` attribute
         ## of your dataset object (e.g., `train_dataset.classes`)
         model_transfer.load_state_dict(torch.load('model_transfer.pt'))

         def predict_landmarks(img_path, k):
```

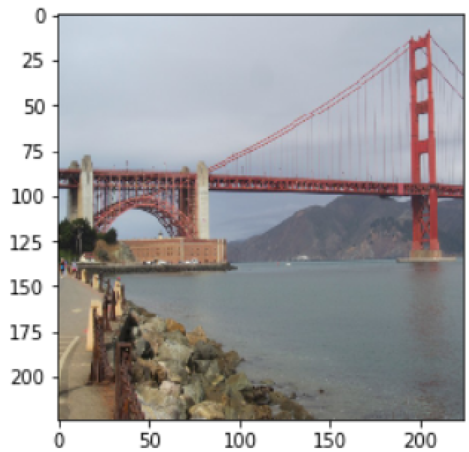
```
    ## TODO: return the names of the top k landmarks predicted by the transfer learned
    loader = transforms.Compose([transforms.Resize(255),
                                  transforms.CenterCrop(224),
                                  transforms.ToTensor(),
                                  transforms.Normalize((0.485, 0.456, 0.406),
                                                       (0.229, 0.224, 0.225))
                                 ])
    image = Image.open(img_path)
    image = loader(image).float()
    image.unsqueeze_(0)

    if use_cuda:
        image = image.cuda()

    model_transfer.eval()
    pred = model_transfer(image)
    _ , top_class = pred.topk(k, dim=1)
    #print(top_class) # Was testing
    #indices = top_class.tolist() # Not needed at end
    #print(indices) # More testing
    classes = train_data.classes
    #print(classes[1]) # Another 1
    predictions = []
    for idx in top_class[0]:
        #print(classes[idx]) # Last 1
        predictions.append(classes[idx])

    return predictions

# test on a sample image
predict_landmarks('images/test/09.Golden_Gate_Bridge/190f3bae17c32c37.jpg', 5)
```

```
Out[13]: ['09.Golden_Gate_Bridge',
          '38.Forth_Bridge',
          '30.Brooklyn_Bridge',
          '28.Sydney_Harbour_Bridge',
          '06.Niagara_Falls']
```

1.1.16 (IMPLEMENTATION) Write Your Algorithm, Part 2

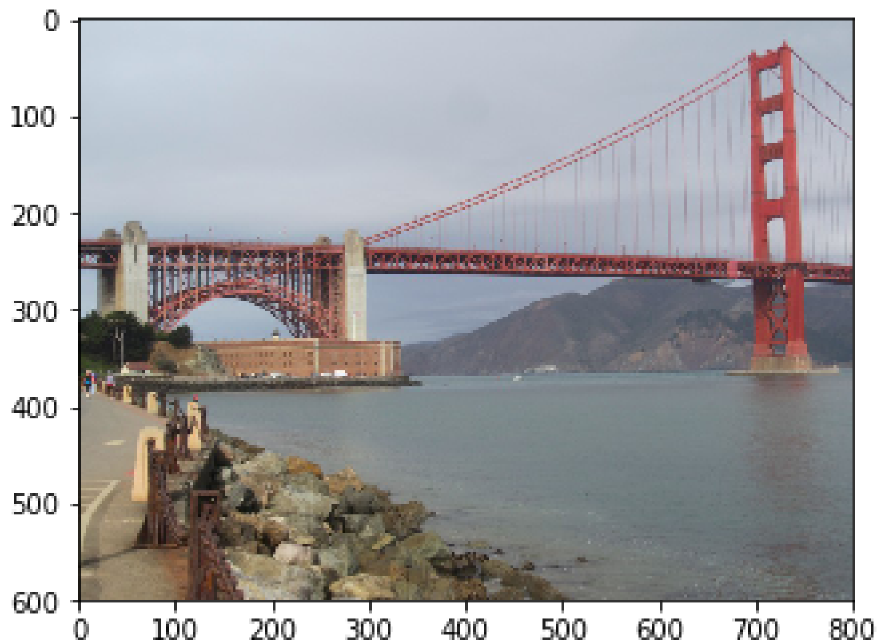
In the code cell below, implement the function `suggest_locations`, which accepts a file path to an image as input, and then displays the image and the **top 3 most likely landmarks** as predicted by `predict_landmarks`.

Some sample output for `suggest_locations` is provided below, but feel free to design your own user experience!




```
Is this picture of the
Golden Gate Bridge, Brooklyn Bridge, or Sydney Harbour Bridge?
```

In [14]:
```
import matplotlib.pyplot as plt
def suggest_locations(img_path):
    # get landmark predictions
    predicted_landmarks = predict_landmarks(img_path, 3)
    ## TODO: display image and display landmark predictions
    img = Image.open(img_path)
    plt.imshow(img)
    plt.show()
    print('The Top 3 Predicted Landmarks for this foto are: ')
    print(predicted_landmarks[0])
    print(predicted_landmarks[1])
    print(predicted_landmarks[2])

# test on a sample image
suggest_locations('images/test/09.Golden_Gate_Bridge/190f3bae17c32c37.jpg')
```



```
The Top 3 Predicted Landmarks for this foto are:
09.Golden_Gate_Bridge
38.Forth_Bridge
30.Brooklyn_Bridge
```

1.1.17 (IMPLEMENTATION) Test Your Algorithm

Test your algorithm by running the `suggest_locations` function on at least four images on your computer. Feel free to use any images you like.

Question 4: Is the output better than you expected :) ? Or worse :(? Provide at least three possible points of improvement for your algorithm.

Answer:

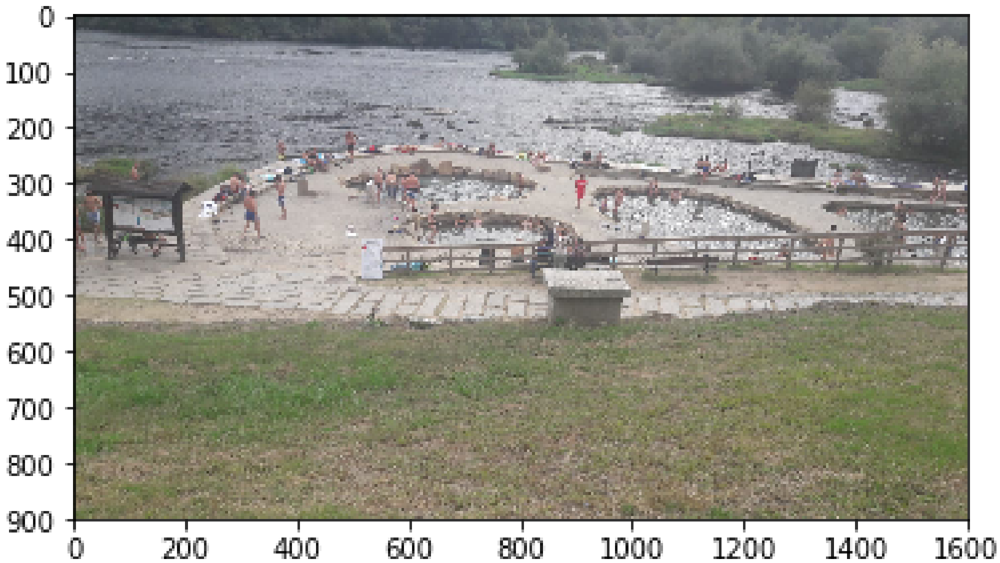
- Well the output is as expected. The photos used from my computer were from Spain. Where no landmarks were used for training the model, so it tried to match them to similar ones in its classes.

It classified well the Golden Gate, from Google Images, but failed with the Sagrada Familia.

- Improv 1: More landmarks to train, for example from Spain, :)
- Improv 2: I was reading some discussion about lr scheduler, so I would definitely have used it to improve performance if I hadn't come up with my current attempt. Luckily wasnt neccesary, but next time I will use it.
- Improv 3: Unfreeze the last layers of the pre-entrained network could help learning more advanced features from the data

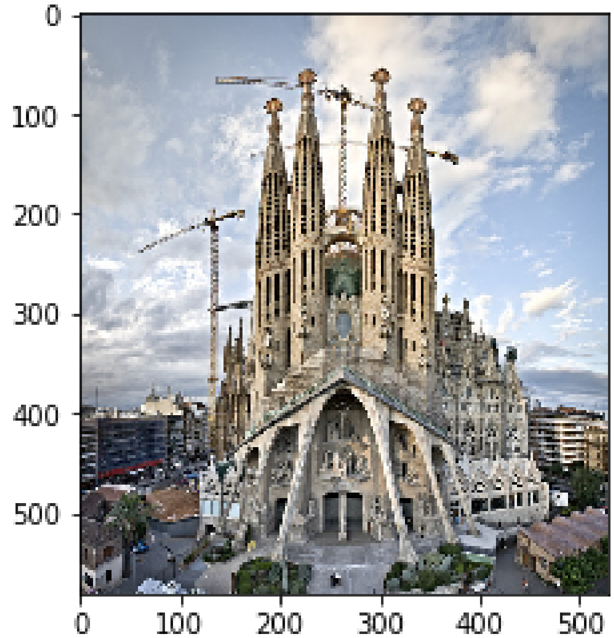
- Improv 4: Different or more types or augmentation, also duplicating the images so it had more impact, 1 normal and 1 augmentated

In [15]:
```
## TODO: Execute the `suggest_locations` function on
## at least 4 images on your computer.
## Feel free to use as many code cells as needed.
suggest_locations('my_samples/055.jpg') # Ourense's Thermal Pools
```



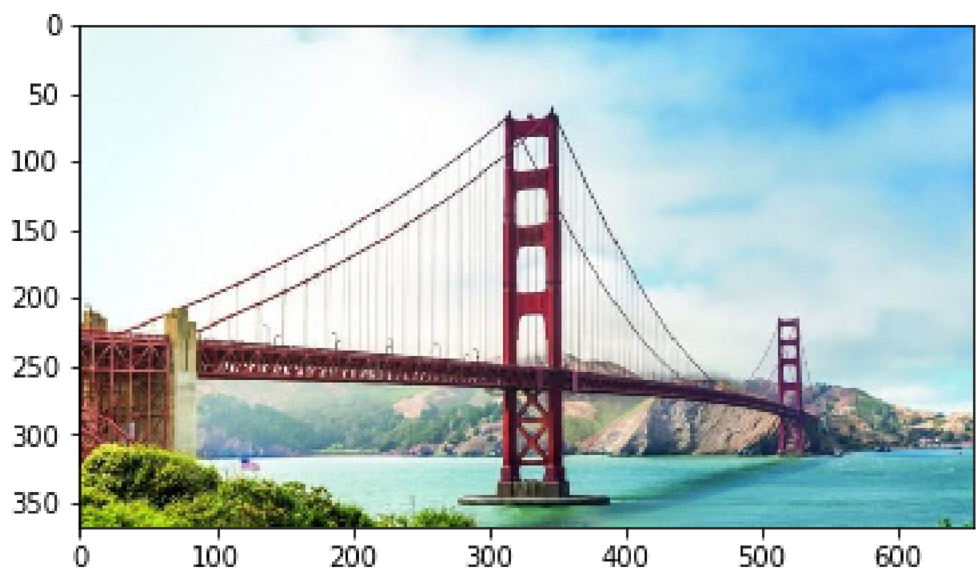
```
The Top 3 Predicted Landmarks for this foto are:
03.Dead_Sea
02.Ljubljana_Castle
48.Whitby_Abbey
```

In [16]:
```
suggest_locations('my_samples/57.jpg') # Sagrada Familia Barcelona
```



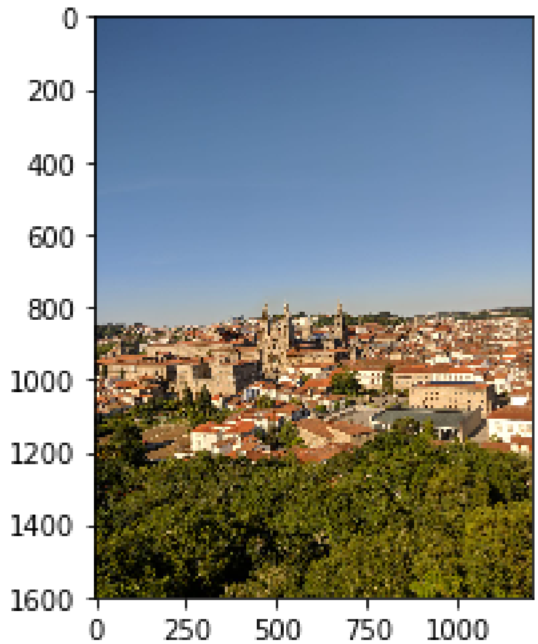
```
The Top 3 Predicted Landmarks for this foto are:
19.Vienna_City_Hall
16.Eiffel_Tower
28.Sydney_Harbour_Bridge
```

```
In [17]: suggest_locations('my_samples/94.jpg') # Golden Gate
```



```
The Top 3 Predicted Landmarks for this foto are:
09.Golden_Gate_Bridge
30.Brooklyn_Bridge
38.Forth_Bridge
```

```
In [18]: suggest_locations('my_samples/2905.jpg') # Catedral of Santiago de Compostela
```




```
The Top 3 Predicted Landmarks for this foto are:
10.Edinburgh_Castle
21.Taj_Mahal
19.Vienna_City_Hall
```

2 *Stand Out Suggestion 3*

Additional use cases: I read online that VGG is being used from numerous multilabel classifications of all kind from brain diseases to clothing material, and even to facial attributes. So it may be useful for a huge number of possible use cases, as can be seen in these examples. One could be, helping police agents to identify criminal photos and classify their location based on photos taken from their social media site. Similar to what we have done here with landmarks but in a higher quality.