dog_app

May 2, 2020

1 Convolutional Neural Networks

1.1 Project: Write an Algorithm for a Dog Identification App

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 of 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 of 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: Import Datasets

Make sure that you've downloaded the required human and dog datasets:

Note: if you are using the Udacity workspace, you *DO NOT* need to re-download these - they can be found in the `/data` folder as noted in the cell below.

- Download the dog dataset. Unzip the folder and place it in this project's home directory, at the location `/dog_images`.
- Download the human dataset. Unzip the folder and place it in the home directory, at location `/lfw`.

Note: If you are using a Windows machine, you are encouraged to use 7zip to extract the folder.

In the code cell below, we save the file paths for both the human (LFW) dataset and dog dataset in the numpy arrays `human_files` and `dog_files`.

```
In [2]: import numpy as np
        from glob import glob

        # load filenames for human and dog images
        human_files = np.array(glob("/data/lfw/*/*"))
        dog_files = np.array(glob("/data/dog_images/*/*/*"))

        # print number of images in each dataset
        print('There are %d total human images.' % len(human_files))
        print('There are %d total dog images.' % len(dog_files))
```

```
There are 13233 total human images.
There are 8351 total dog images.
```

```
In [3]: dog_files
```

```
Out[3]: array(['/data/dog_images/train/103.Mastiff/Mastiff_06833.jpg',
               '/data/dog_images/train/103.Mastiff/Mastiff_06826.jpg',
               '/data/dog_images/train/103.Mastiff/Mastiff_06871.jpg', ...,
               '/data/dog_images/valid/100.Lowchen/Lowchen_06682.jpg',
               '/data/dog_images/valid/100.Lowchen/Lowchen_06708.jpg',
               '/data/dog_images/valid/100.Lowchen/Lowchen_06684.jpg'],
              dtype='<U106')
```

Step 1: Detect Humans

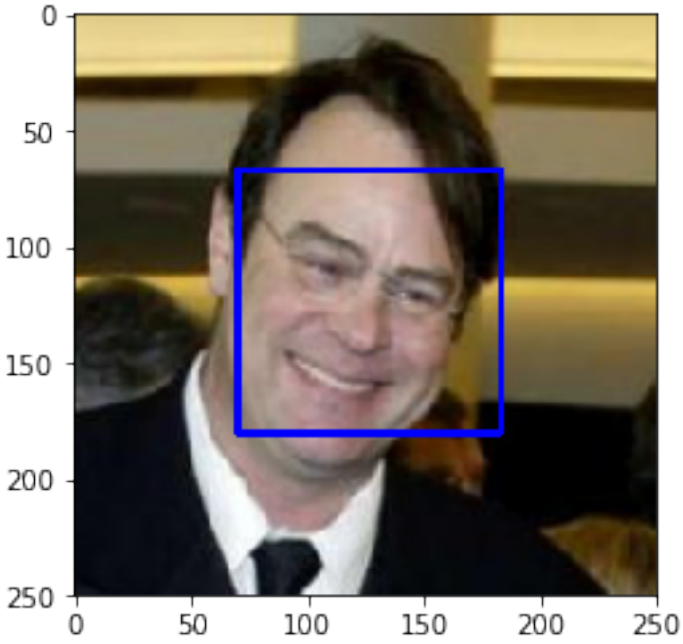
In this section, we use OpenCV's implementation of Haar feature-based cascade classifiers to detect human faces in images.

OpenCV provides many pre-trained face detectors, stored as XML files on github. We have downloaded one of these detectors and stored it in the `haarcascades` directory. In the next code cell, we demonstrate how to use this detector to find human faces in a sample image.

```
In [4]: import cv2
        import matplotlib.pyplot as plt
        %matplotlib inline

        # extract pre-trained face detector
        face_cascade = cv2.CascadeClassifier('haarcascades/haarcascade_frontalface_alt.xml')
```

```
# load color (BGR) image
img = cv2.imread(human_files[0])
# convert BGR image to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# find faces in image
faces = face_cascade.detectMultiScale(gray)

# print number of faces detected in the image
print('Number of faces detected:', len(faces))

# get bounding box for each detected face
for (x,y,w,h) in faces:
    # add bounding box to color image
    cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)

# convert BGR image to RGB for plotting
cv_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# display the image, along with bounding box
plt.imshow(cv_rgb)
plt.show()
```

```
Number of faces detected: 1
```



Before using any of the face detectors, it is standard procedure to convert the images to grayscale. The `detectMultiScale` function executes the classifier stored in `face_cascade` and takes the grayscale image as a parameter.

In the above code, `faces` is a numpy array of detected faces, where each row corresponds to a detected face. Each detected face is a 1D array with four entries that specifies the bounding box of the detected face. The first two entries in the array (extracted in the above code as `x` and `y`) specify the horizontal and vertical positions of the top left corner of the bounding box. The last two entries in the array (extracted here as `w` and `h`) specify the width and height of the box.

1.1.1 Write a Human Face Detector

We can use this procedure to write a function that returns `True` if a human face is detected in an image and `False` otherwise. This function, aptly named `face_detector`, takes a string-valued file path to an image as input and appears in the code block below.

```
In [5]: # returns "True" if face is detected in image stored at img_path
        def face_detector(img_path):
            img = cv2.imread(img_path)
            gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
            faces = face_cascade.detectMultiScale(gray)
            return len(faces) > 0
```

1.1.2 (IMPLEMENTATION) Assess the Human Face Detector

Question 1: Use the code cell below to test the performance of the `face_detector` function.
- What percentage of the first 100 images in `human_files` have a detected human face?
- What percentage of the first 100 images in `dog_files` have a detected human face?

Ideally, we would like 100% of human images with a detected face and 0% of dog images with a detected face. You will see that our algorithm falls short of this goal, but still gives acceptable performance. We extract the file paths for the first 100 images from each of the datasets and store them in the numpy arrays `human_files_short` and `dog_files_short`.

Answer: (You can print out your results and/or write your percentages in this cell) 1. 98% of humans have been detected as humans 2. 17% of dogs have been detected as humans

```
In [ ]: from tqdm import tqdm

        human_files_short = human_files[:100]
        dog_files_short = dog_files[:100]

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

        ## TODO: Test the performance of the face_detector algorithm
        ## on the images in human_files_short and dog_files_short.
        hum_cnt = 0
        an_cnt = 0
        for file in human_files_short:
            if face_detector(file):
                hum_cnt += 1
        for file in dog_files_short:
```

```
        if face_detector(file):
            an_cnt += 1

    print(f"{hum_cnt}/100")
    print(f"{an_cnt}/100")
```

We suggest the face detector from OpenCV as a potential way to detect human images in your algorithm, but you are free to explore other approaches, especially approaches that make use of deep learning :). Please use the code cell below to design and test your own face detection algorithm. If you decide to pursue this *optional* task, report performance on `human_files_short` and `dog_files_short`.

```
In [ ]: ### (Optional)
        ### TODO: Test performance of anotherface detection algorithm.
        ### Feel free to use as many code cells as needed.
```

Step 2: Detect Dogs

In this section, we use a pre-trained model to detect dogs in images.

1.1.3 Obtain Pre-trained VGG-16 Model

The code cell below downloads the VGG-16 model, along with weights that have been trained on ImageNet, a very large, very popular dataset used for image classification and other vision tasks. ImageNet contains over 10 million URLs, each linking to an image containing an object from one of 1000 categories.

```
In [6]: import torch
        import torchvision.models as models

        # define VGG16 model
        VGG16 = models.vgg16(pretrained=True)

        # check if CUDA is available
        use_cuda = torch.cuda.is_available()

        # move model to GPU if CUDA is available
        if use_cuda:
            VGG16 = VGG16.cuda()
```

```
Downloading: "https://download.pytorch.org/models/vgg16-397923af.pth" to /root/.torch/models/vgg
100%|| 553433881/553433881 [00:07<00:00, 70276969.29it/s]
```

Given an image, this pre-trained VGG-16 model returns a prediction (derived from the 1000 possible categories in ImageNet) for the object that is contained in the image.

1.1.4 (IMPLEMENTATION) Making Predictions with a Pre-trained Model

In the next code cell, you will write a function that accepts a path to an image (such as `'dogImages/train/001.Affenpinscher/Affenpinscher_00001.jpg'`) as input and returns the index corresponding to the ImageNet class that is predicted by the pre-trained VGG-16 model. The output should always be an integer between 0 and 999, inclusive.

Before writing the function, make sure that you take the time to learn how to appropriately pre-process tensors for pre-trained models in the PyTorch documentation.

```
In [7]: from PIL import Image
        import torchvision.transforms as transforms

        def load_image(img_path):
            image = Image.open(img_path).convert('RGB')
            transform = transforms.Compose([
                                transforms.Resize(size=(244, 244)),
                                transforms.ToTensor()])
            image = transform(image)[:3,:,:].unsqueeze(0)
            return image

        def VGG16_predict(img_path):
            '''
            Use pre-trained VGG-16 model to obtain index corresponding to
            predicted ImageNet class for image at specified path

            Args:
                img_path: path to an image

            Returns:
                Index corresponding to VGG-16 model's prediction
            '''
            ## TODO: Complete the function.
            ## Load and pre-process an image from the given img_path
            ## Return the *index* of the predicted class for that image
            img = load_image(img_path)
            if use_cuda:
                img = img.cuda()
            ret = VGG16(img)
            return torch.max(ret,1)[1].item() # predicted class index
```

1.1.5 (IMPLEMENTATION) Write a Dog Detector

While looking at the dictionary, you will notice that the categories corresponding to dogs appear in an uninterrupted sequence and correspond to dictionary keys 151-268, inclusive, to include all categories from `'Chihuahua'` to `'Mexican hairless'`. Thus, in order to check to see if an image is predicted to contain a dog by the pre-trained VGG-16 model, we need only check if the pre-trained model predicts an index between 151 and 268 (inclusive).

Use these ideas to complete the `dog_detector` function below, which returns `True` if a dog is detected in an image (and `False` if not).

```
In [8]: ### returns "True" if a dog is detected in the image stored at img_path
        def dog_detector(img_path):
            ## TODO: Complete the function.
            idx = VGG16_predict(img_path)
            return idx >= 151 and idx <= 268 # true/false
```

1.1.6 (IMPLEMENTATION) Assess the Dog Detector

Question 2: Use the code cell below to test the performance of your `dog_detector` function.
- What percentage of the images in `human_files_short` have a detected dog?
- What percentage of the images in `dog_files_short` have a detected dog?

Answer: 1. 0% of the humans have been detected as dog 2. 98% of the dogs have been detected as dog

```
In [ ]: ### TODO: Test the performance of the dog_detector function
        ### on the images in human_files_short and dog_files_short.
        dog_hum_cnt = 0
        dog_an_cnt = 0

        for file in human_files_short:
            if dog_detector(file):
                dog_hum_cnt += 1
        for file in dog_files_short:
            if dog_detector(file):
                dog_an_cnt += 1

        print(f"{dog_hum_cnt} out of 100 humans")
        print(f"{dog_an_cnt} out of 100 dogs")
```

We suggest VGG-16 as a potential network to detect dog images in your algorithm, but you are free to explore other pre-trained networks (such as Inception-v3, ResNet-50, etc). Please use the code cell below to test other pre-trained PyTorch models. If you decide to pursue this *optional* task, report performance on `human_files_short` and `dog_files_short`.

```
In [9]: ### (Optional)
        ### TODO: Report the performance of another pre-trained network.
        ### Feel free to use as many code cells as needed.
```

Step 3: Create a CNN to Classify Dog Breeds (from Scratch)

Now that we have functions for detecting humans and dogs in images, we need a way to predict breed from images. In this step, you will create a CNN that classifies dog breeds. 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 10%. In Step 4 of this notebook, you will have the opportunity to use transfer learning to create a CNN that attains greatly improved accuracy.

We mention that the task of assigning breed to dogs from images is considered exceptionally challenging. To see why, consider that *even a human* would have trouble distinguishing between a Brittany and a Welsh Springer Spaniel.

Brittany	Welsh Springer Spaniel

It is not difficult to find other dog breed pairs with minimal inter-class variation (for instance, Curly-Coated Retrievers and American Water Spaniels).

Curly-Coated Retriever	American Water Spaniel

Likewise, recall that labradors come in yellow, chocolate, and black. Your vision-based algorithm will have to conquer this high intra-class variation to determine how to classify all of these different shades as the same breed.

Yellow Labrador	Chocolate Labrador

We also mention that random chance presents an exceptionally low bar: setting aside the fact that the classes are slightly imabalanced, a random guess will provide a correct answer roughly 1 in 133 times, which corresponds to an accuracy of less than 1%.

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

1.1.7 (IMPLEMENTATION) Specify Data Loaders for the Dog Dataset

Use the code cell below to write three separate data loaders for the training, validation, and test datasets of dog images (located at `dog_images/train`, `dog_images/valid`, and `dog_images/test`, respectively). 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 [9]: import os
        import torch
        import numpy as np
        from torchvision import datasets, transforms

        ### TODO: Write data loaders for training, validation, and test sets
        ## Specify appropriate transforms, and batch_sizes

        standard_normalization = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                                      std=[0.229, 0.224, 0.225])

        train_transform = transforms.Compose([
            transforms.RandomHorizontalFlip(),
            transforms.RandomRotation(45),
            transforms.Resize((224,224)),
            transforms.ToTensor(),
```

```
    standard_normalization
])

test_transform = transforms.Compose([
    transforms.Resize((224,224)),
    transforms.ToTensor(),
    standard_normalization
])


trainset = datasets.ImageFolder(root='/data/dog_images/train', transform=train_transform
validset = datasets.ImageFolder(root='/data/dog_images/valid', transform=train_transform
testset = datasets.ImageFolder(root='/data/dog_images/test', transform=test_transform)

trainloader = torch.utils.data.DataLoader(trainset, batch_size=20, shuffle=True)
validloader = torch.utils.data.DataLoader(validset, batch_size=20, shuffle=True)
testloader = torch.utils.data.DataLoader(testset, batch_size=20, shuffle=True)

loaders_scratch = {
    'train': trainloader,
    'valid': validloader,
    'test': testloader
}
```

Question 3: 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: 1. I resized the code using transforms module. The code uses transforms.Resize((224,224)) to convert the dimensions of the image to target size of 224x224 as it widely accepted image dimension and also default input dimensions of various pre-trained model like VGG. Its also widely accepted practise to convert the image into square shape so I resized it to 224x224.

2. Yes, I decided to augement the data to make my model perform better and reduce overfitting. I used horizontalflip and rotation from transforms to augment the data.

1.1.8 (IMPLEMENTATION) Model Architecture

Create a CNN to classify dog breed. Use the template in the code cell below.

```
In [10]: import torch.nn as nn
         import torch.nn.functional as F
         import numpy as np

         # 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, 32, 3, stride=2, padding=1)
            self.conv2 = nn.Conv2d(32, 64, 3, stride=2, padding=1)
            self.conv3 = nn.Conv2d(64, 128, 3, padding=1)
            self.pool = nn.MaxPool2d(2, 2)
            self.fc1 = nn.Linear(7*7*128, 500)
            self.fc2 = nn.Linear(500, 133)
            self.dropout = nn.Dropout(0.25)

        def forward(self, x):
            ## Define forward behavior
            x = F.relu(self.conv1(x))
            x = self.pool(x)
            x = F.relu(self.conv2(x))
            x = self.pool(x)
            x = F.relu(self.conv3(x))
            x = self.pool(x)
            x = x.view(-1, 7*7*128)
            x= self.dropout(x)
            x = F.relu(self.fc1(x))
            x = self.dropout(x)
            x = self.fc2(x)
            return x


    #-#-# You so NOT have to 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, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
  (conv3): Conv2d(64, 128, 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=6272, out_features=500, bias=True)
  (fc2): Linear(in_features=500, out_features=133, bias=True)
  (dropout): Dropout(p=0.25)
)
```

Question 4: Outline the steps you took to get to your final CNN architecture and your reasoning at each step.

Answer: 1. First 2 conv layers I've applied kernel_size of 3 as its widely used kernel_size followed by MaxPooling layers 2. After 2 conv layers, maxpooling with stride 2 is placed and this will lead to downsize of input image by 2. 3. The 3rd conv layers is consist of kernel_size of 3 with stride 1, and this will not reduce input image. 4. After final maxpooling with stride 2, the total output image size is downsized by factor of 32 and the depth will be 128. 5. I've applied dropout of 0.25 in order to prevent overfitting before Fully-connected layer. 6. Fully connected layer is placed after than followed by 2nd fully-connected layer which is intended to produce final output_size which predicts classes of breeds. 7. All layers have relu activation functions except the last layers, as I will be selecting criterion=CrossEntropyLoss() which applies the softmax function to the output of last layer

1.1.9 (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 the optimizer as `optimizer_scratch` below.

```
In [11]: import torch.optim as optim

         ### TODO: select loss function
         criterion_scratch = nn.CrossEntropyLoss()

         ### TODO: select optimizer
         optimizer_scratch = optim.Adam(model_scratch.parameters(), lr=0.001)
```

1.1.10 (IMPLEMENTATION) Train and Validate the Model

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

```
In [30]: 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

             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 #
                 ###################
                 model.train()
                 for batch_idx, (data, target) in enumerate(loaders['train']):
                     # move to GPU
                     if use_cuda:
                         data, target = data.cuda(), target.cuda()
                     ## 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 - train_lo
            optimizer.zero_grad()
            output = model(data)
            loss = criterion(output, target)
            loss.backward()
            optimizer.step()

            train_loss += (1 / (batch_idx + 1)) * (loss.data - train_loss)
            if batch_idx % 100 == 0:
                print('Epoch %d, Batch %d loss: %.6f' %
                  (epoch, batch_idx + 1, train_loss))


        ######################
        # validate the model #
        ######################
        model.eval()
        for batch_idx, (data, target) in enumerate(loaders['valid']):
            # move to GPU
            if use_cuda:
                data, target = data.cuda(), target.cuda()
            ## update the average validation loss
            output = model(data)
            loss = criterion(output, target)
            valid_loss += (1 / (batch_idx + 1)) * (loss.data - valid_loss)


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

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

    # return trained model
    return model

# train the model
model_scratch = train(100, loaders_scratch, model_scratch, optimizer_scratch,
                      criterion_scratch, use_cuda, 'model_scratch.pt')
```

```
        # load the model that got the best validation accuracy
        model_scratch.load_state_dict(torch.load('model_scratch.pt'))
```

```
Epoch 1, Batch 1 loss: 4.539851
Epoch 1, Batch 101 loss: 4.192613
Epoch 1, Batch 201 loss: 4.199136
Epoch 1, Batch 301 loss: 4.193339
Epoch: 1         Training Loss: 4.192425         Validation Loss: 4.171508
Validation loss decreased (inf --> 4.171508).  Saving model ...
Epoch 2, Batch 1 loss: 3.889310
Epoch 2, Batch 101 loss: 4.078901
Epoch 2, Batch 201 loss: 4.067098
Epoch 2, Batch 301 loss: 4.058945
Epoch: 2         Training Loss: 4.061929         Validation Loss: 4.076018
Validation loss decreased (4.171508 --> 4.076018).  Saving model ...
Epoch 3, Batch 1 loss: 3.948102
Epoch 3, Batch 101 loss: 3.956148
Epoch 3, Batch 201 loss: 3.947497
Epoch 3, Batch 301 loss: 3.945089
Epoch: 3         Training Loss: 3.934783         Validation Loss: 4.036073
Validation loss decreased (4.076018 --> 4.036073).  Saving model ...
Epoch 4, Batch 1 loss: 3.598597
Epoch 4, Batch 101 loss: 3.896841
Epoch 4, Batch 201 loss: 3.875212
Epoch 4, Batch 301 loss: 3.843456
Epoch: 4         Training Loss: 3.840196         Validation Loss: 3.967484
Validation loss decreased (4.036073 --> 3.967484).  Saving model ...
Epoch 5, Batch 1 loss: 3.665117
Epoch 5, Batch 101 loss: 3.725762
Epoch 5, Batch 201 loss: 3.741030
Epoch 5, Batch 301 loss: 3.739758
Epoch: 5         Training Loss: 3.742875         Validation Loss: 3.915607
Validation loss decreased (3.967484 --> 3.915607).  Saving model ...
Epoch 6, Batch 1 loss: 3.378133
Epoch 6, Batch 101 loss: 3.651158
Epoch 6, Batch 201 loss: 3.655387
Epoch 6, Batch 301 loss: 3.645646
Epoch: 6         Training Loss: 3.642410         Validation Loss: 3.783824
Validation loss decreased (3.915607 --> 3.783824).  Saving model ...
Epoch 7, Batch 1 loss: 3.235156
Epoch 7, Batch 101 loss: 3.528157
Epoch 7, Batch 201 loss: 3.541631
Epoch 7, Batch 301 loss: 3.547986
Epoch: 7         Training Loss: 3.546667         Validation Loss: 3.849987
Epoch 8, Batch 1 loss: 3.694313
Epoch 8, Batch 101 loss: 3.450254
Epoch 8, Batch 201 loss: 3.453306
```

```
Epoch 8, Batch 301 loss: 3.465000
Epoch: 8         Training Loss: 3.467976         Validation Loss: 3.749968
Validation loss decreased (3.783824 --> 3.749968).  Saving model ...
Epoch 9, Batch 1 loss: 3.582586
Epoch 9, Batch 101 loss: 3.383260
Epoch 9, Batch 201 loss: 3.385746
Epoch 9, Batch 301 loss: 3.409111
Epoch: 9         Training Loss: 3.407104         Validation Loss: 3.757144
Epoch 10, Batch 1 loss: 3.526085
Epoch 10, Batch 101 loss: 3.296979
Epoch 10, Batch 201 loss: 3.305474
Epoch 10, Batch 301 loss: 3.329292
Epoch: 10         Training Loss: 3.330431         Validation Loss: 3.784858
Epoch 11, Batch 1 loss: 3.271615
Epoch 11, Batch 101 loss: 3.239656
Epoch 11, Batch 201 loss: 3.283058
Epoch 11, Batch 301 loss: 3.290442
Epoch: 11         Training Loss: 3.290467         Validation Loss: 3.741269
Validation loss decreased (3.749968 --> 3.741269).  Saving model ...
Epoch 12, Batch 1 loss: 3.030665
Epoch 12, Batch 101 loss: 3.190143
Epoch 12, Batch 201 loss: 3.235844
Epoch 12, Batch 301 loss: 3.230671
Epoch: 12         Training Loss: 3.234148         Validation Loss: 3.722297
Validation loss decreased (3.741269 --> 3.722297).  Saving model ...
Epoch 13, Batch 1 loss: 2.898851



        ---------------------------------------------------------------------------

        KeyboardInterrupt                         Traceback (most recent call last)

        <ipython-input-30-090bd1e279d2> in <module>()
         66 # train the model
         67 model_scratch = train(100, loaders_scratch, model_scratch, optimizer_scratch,
    ---> 68                        criterion_scratch, use_cuda, 'model_scratch.pt')
         69
         70 # load the model that got the best validation accuracy


        <ipython-input-30-090bd1e279d2> in train(n_epochs, loaders, model, optimizer, criterion,
         13         ####################
         14         model.train()
    ---> 15         for batch_idx, (data, target) in enumerate(loaders['train']):
         16             # move to GPU
         17             if use_cuda:
```

```
    /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/
    101         sample = self.loader(path)
    102         if self.transform is not None:
--> 103             sample = self.transform(sample)
    104         if self.target_transform is not None:
    105             target = self.target_transform(target)


    /opt/conda/lib/python3.6/site-packages/torchvision-0.2.1-py3.6.egg/torchvision/transform
     47     def __call__(self, img):
     48         for t in self.transforms:
---> 49             img = t(img)
     50         return img
     51


    /opt/conda/lib/python3.6/site-packages/torchvision-0.2.1-py3.6.egg/torchvision/transform
    173             PIL Image: Rescaled image.
    174         """
--> 175         return F.resize(img, self.size, self.interpolation)
    176
    177     def __repr__(self):


    /opt/conda/lib/python3.6/site-packages/torchvision-0.2.1-py3.6.egg/torchvision/transform
    204             return img.resize((ow, oh), interpolation)
    205     else:
--> 206         return img.resize(size[::-1], interpolation)
    207
    208
```

```
 /opt/conda/lib/python3.6/site-packages/PIL/Image.py in resize(self, size, resample, box)
   1763         self.load()
   1764
-> 1765         return self._new(self.im.resize(size, resample, box))
   1766
   1767     def rotate(self, angle, resample=NEAREST, expand=0, center=None,


KeyboardInterrupt:
```

```
In [29]: from PIL import ImageFile
         ImageFile.LOAD_TRUNCATED_IMAGES = True
```

1.1.11 (IMPLEMENTATION) Test the Model

Try out your model on the test dataset of dog images. Use the code cell below to calculate and print the test loss and accuracy. Ensure that your test accuracy is greater than 10%.

```
In [31]: def test(loaders, model, criterion, use_cuda):

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

             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 - 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))
```

```
        # call test function
        test(loaders_scratch, model_scratch, criterion_scratch, use_cuda)
```

```
Test Loss: 3.665728


Test Accuracy: 15% (132/836)
```

Step 4: Create a CNN to Classify Dog Breeds (using Transfer Learning)

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

1.1.12 (IMPLEMENTATION) Specify Data Loaders for the Dog Dataset

Use the code cell below to write three separate data loaders for the training, validation, and test datasets of dog images (located at `dogImages/train`, `dogImages/valid`, and `dogImages/test`, respectively).

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

```
In [12]: trainloader = torch.utils.data.DataLoader(trainset, batch_size=20, shuffle=True)
         validloader = torch.utils.data.DataLoader(validset, batch_size=20, shuffle=True)
         testloader = torch.utils.data.DataLoader(testset, batch_size=20, shuffle=True)

         loaders_transfer = {
             'train': trainloader,
             'valid': validloader,
             'test': testloader
         }
```

1.1.13 (IMPLEMENTATION) Model Architecture

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

```
In [23]: import torchvision.models as models
         import torch.nn as nn

         ## TODO: Specify model architecture
         model_transfer = models.resnet50(pretrained=True)

         model_transfer.fc = nn.Linear(2048, 133, bias=True)
         fc_parameters = model_transfer.fc.parameters()

         for param in fc_parameters:
```

```
            param.requires_grad = True


        if use_cuda:
            model_transfer = model_transfer.cuda()

        print(model_transfer)
```

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
      (downsample): Sequential(
        (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
    )
    (2): Bottleneck(
      (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
    )
  )
  (layer2): Sequential(
    (0): Bottleneck(
```

```
    (conv1): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace)
    (downsample): Sequential(
      (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
      (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (1): Bottleneck(
    (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace)
  )
  (2): Bottleneck(
    (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace)
  )
  (3): Bottleneck(
    (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace)
  )
)
(layer3): Sequential(
  (0): Bottleneck(
    (conv1): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
```

```
  (relu): ReLU(inplace)
  (downsample): Sequential(
    (0): Conv2d(512, 1024, kernel_size=(1, 1), stride=(2, 2), bias=False)
    (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
)
(1): Bottleneck(
  (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace)
)
(2): Bottleneck(
  (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace)
)
(3): Bottleneck(
  (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace)
)
(4): Bottleneck(
  (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace)
)
(5): Bottleneck(
  (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
  (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
```

```
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
    )
  )
  (layer4): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
      (downsample): Sequential(
        (0): Conv2d(1024, 2048, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
    )
    (2): Bottleneck(
      (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
    )
  )
  (avgpool): AvgPool2d(kernel_size=7, stride=1, padding=0)
  (fc): Linear(in_features=2048, out_features=133, bias=True)
)
```

Question 5: 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: 1. I chose ResNet as a transfer learning model because it performs great on Image Classification problems. 2. I looked into the structure and functions of ResNet. The core idea of ResNet is introducing a so-called "identity shortcut connection" that skips one or more layers. 3. I think this prevents overfitting when it's training. 4. I've replaced the final Fully-connected layer

and replaced it with Fully-connected layer with output of 133

1.1.14 (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 the optimizer as `optimizer_transfer` below.

```
In [21]: criterion_transfer = nn.CrossEntropyLoss()
         optimizer_transfer = optim.Adam(model_transfer.fc.parameters(), lr=0.001)
```

1.1.15 (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 [24]: # train the model
         #model_transfer = train(20, loaders_transfer, model_transfer, optimizer_transfer, crite

         # load the model that got the best validation accuracy (uncomment the line below)
         #model_transfer.load_state_dict(torch.load('model_transfer.pt'))

         if torch.cuda.is_available():
             map_location=lambda storage, loc: storage.cuda()
         else:
             map_location='cpu'

         checkpoint = torch.load('model_transfer.pt', map_location=map_location)
```

```
In [26]: model_transfer.load_state_dict(checkpoint)
```

1.1.16 (IMPLEMENTATION) Test the Model

Try out your model on the test dataset of dog 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 [37]: test(loaders_transfer, model_transfer, criterion_transfer, use_cuda)
```

```
Test Loss: 0.623861


Test Accuracy: 80% (677/836)
```

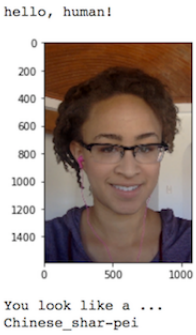
1.1.17 (IMPLEMENTATION) Predict Dog Breed with the Model

Write a function that takes an image path as input and returns the dog breed (`Affenpinscher`, `Afghan hound`, etc) that is predicted by your model.

```
In [13]: ### TODO: Write a function that takes a path to an image as input
         ### and returns the dog breed that is predicted by the model.

         # list of class names by index, i.e. a name can be accessed like class_names[0]
         class_names = [item[4:].replace("_", " ") for item in loaders_transfer['train'].dataset

         from PIL import Image
         def load_input_image(img_path):
             image = Image.open(img_path).convert('RGB')
             prediction_transform = transforms.Compose([transforms.Resize(size=(224, 224)),
                                                        transforms.ToTensor(),
                                                        standard_normalization])

             # discard the transparent, alpha channel (that's the :3) and add the batch dimensio
             image = prediction_transform(image)[:3,:,:].unsqueeze(0)
             return image

         def predict_breed_transfer(model, class_names, img_path):
             # load the image and return the predicted breed
             img = load_input_image(img_path)
             model = model.cpu()
             model.eval()
             idx = torch.argmax(model(img))
             return class_names[idx]

In [45]: for img_file in os.listdir('./images'):
             img_path = os.path.join('./images', img_file)
             predition = predict_breed_transfer(model_transfer, class_names, img_path)
             print("image_file_name: {0}, \t predition breed: {1}".format(img_path, predition))

image_file_name: ./images/Welsh_springer_spaniel_08203.jpg,          predition breed: Irish red
image_file_name: ./images/sample_human_output.png,          predition breed: Xoloitzcuintli
image_file_name: ./images/Labrador_retriever_06457.jpg,          predition breed: Labrador retri
image_file_name: ./images/Curly-coated_retriever_03896.jpg,          predition breed: Curly-coat
image_file_name: ./images/sample_cnn.png,          predition breed: Akita
image_file_name: ./images/Brittany_02625.jpg,          predition breed: Brittany
image_file_name: ./images/Labrador_retriever_06449.jpg,          predition breed: Labrador retri
image_file_name: ./images/American_water_spaniel_00648.jpg,          predition breed: Boykin spa
image_file_name: ./images/sample_dog_output.png,          predition breed: Smooth fox terrier
image_file_name: ./images/Labrador_retriever_06455.jpg,          predition breed: Labrador retri
```

Step 5: Write your Algorithm

Write an algorithm that accepts a file path to an image and first determines whether the image contains a human, dog, or neither. Then, - if a **dog** is detected in the image, return the predicted breed. - if a **human** is detected in the image, return the resembling dog breed. - if **neither** is detected in the image, provide output that indicates an error.



Sample Human Output

You are welcome to write your own functions for detecting humans and dogs in images, but feel free to use the `face_detector` and `human_detector` functions developed above. You are **required** to use your CNN from Step 4 to predict dog breed.

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

1.1.18 (IMPLEMENTATION) Write your Algorithm

```
In [14]: ### TODO: Write your algorithm.
         ### Feel free to use as many code cells as needed.
         import matplotlib.pyplot as plt

         def run_app(img_path):
             ## handle cases for a human face, dog, and neither
             img = Image.open(img_path)
             plt.imshow(img)
             plt.show()
             if dog_detector(img_path) is True:
                 prediction = predict_breed_transfer(model_transfer, class_names, img_path)
                 print("Dogs Detected!\nIt looks like a {0}".format(prediction))
             elif face_detector(img_path) > 0:
                 prediction = predict_breed_transfer(model_transfer, class_names, img_path)
                 print("Hello, human!\nIf you were a dog..You may look like a {0}".format(predic
             else:
                 print("Error! Can't detect anything..")
```

Step 6: Test Your Algorithm

In this section, you will take your new algorithm for a spin! What kind of dog does the algorithm think that *you* look like? If you have a dog, does it predict your dog's breed accurately? If you have a cat, does it mistakenly think that your cat is a dog?

1.1.19 (IMPLEMENTATION) Test Your Algorithm on Sample Images!

Test your algorithm at least six images on your computer. Feel free to use any images you like. Use at least two human and two dog images.

Question 6: Is the output better than you expected :) ? Or worse :(? Provide at least three possible points of improvement for your algorithm.

Answer: (Three possible points for improvement) 1. More image datasets of dogs will improve training models. Also, more image augmentations trials (flipping vertically, move left or right, etc.) will improve performance on test data. 2. Ensembles of models 3. Hyper-parameter tunings: weight initializings, learning rates, drop-outs, batch_sizes, and optimizers will be helpful to improve performances.

```
In [55]: !wget -O https://cdn.pixabay.com/photo/2016/03/23/04/01/beautiful-1274056_960_720.jpg
         !wget -O https://cdn.pixabay.com/photo/2016/03/23/08/34/beautiful-1274361_960_720.jpg
         !wget -O https://cdn.pixabay.com/photo/2017/06/09/17/11/model-2387582_960_720.jpg
         !wget -O https://cdn.pixabay.com/photo/2016/05/09/10/42/weimaraner-1381186_960_720.jpg
         !wget -O https://cdn.pixabay.com/photo/2016/02/19/15/46/dog-1210559_960_720.jpg
         !wget -O https://cdn.pixabay.com/photo/2016/12/13/05/15/puppy-1903313_960_720.jpg

--2020-05-02 12:15:59--  https://cdn.pixabay.com/photo/2016/03/23/04/01/beautiful-1274056_960_72
Resolving cdn.pixabay.com (cdn.pixabay.com)... 104.18.21.183, 104.18.20.183, 2606:4700::6812:14b
Connecting to cdn.pixabay.com (cdn.pixabay.com)|104.18.21.183|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 155272 (152K) [image/jpeg]
Saving to: /myimages

/myimages           100%[===================>] 151.63K  --.-KB/s    in 0.02s

2020-05-02 12:16:00 (6.00 MB/s) - /myimages saved [155272/155272]

--2020-05-02 12:16:00--  https://cdn.pixabay.com/photo/2016/03/23/08/34/beautiful-1274361_960_72
Resolving cdn.pixabay.com (cdn.pixabay.com)... 104.18.20.183, 104.18.21.183, 2606:4700::6812:15b
Connecting to cdn.pixabay.com (cdn.pixabay.com)|104.18.20.183|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 158275 (155K) [image/jpeg]
Saving to: /myimages

/myimages           100%[===================>] 154.57K  --.-KB/s    in 0.03s

2020-05-02 12:16:01 (6.04 MB/s) - /myimages saved [158275/158275]

--2020-05-02 12:16:02--  https://cdn.pixabay.com/photo/2017/06/09/17/11/model-2387582_960_720.jp
Resolving cdn.pixabay.com (cdn.pixabay.com)... 104.18.21.183, 104.18.20.183, 2606:4700::6812:15b
Connecting to cdn.pixabay.com (cdn.pixabay.com)|104.18.21.183|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 106218 (104K) [image/jpeg]
Saving to: /myimages

/myimages           100%[===================>] 103.73K  --.-KB/s    in 0.01s

2020-05-02 12:16:02 (8.50 MB/s) - /myimages saved [106218/106218]
```

```
--2020-05-02 12:16:03--  https://cdn.pixabay.com/photo/2016/05/09/10/42/weimaraner-1381186_960_7
Resolving cdn.pixabay.com (cdn.pixabay.com)... 104.18.20.183, 104.18.21.183, 2606:4700::6812:14b
Connecting to cdn.pixabay.com (cdn.pixabay.com)|104.18.20.183|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 75772 (74K) [image/jpeg]
Saving to: /myimages

/myimages           100%[===================>]  74.00K  --.-KB/s    in 0.01s

2020-05-02 12:16:03 (5.92 MB/s) - /myimages saved [75772/75772]

--2020-05-02 12:16:04--  https://cdn.pixabay.com/photo/2016/02/19/15/46/dog-1210559_960_720.jpg
Resolving cdn.pixabay.com (cdn.pixabay.com)... 104.18.20.183, 104.18.21.183, 2606:4700::6812:15b
Connecting to cdn.pixabay.com (cdn.pixabay.com)|104.18.20.183|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 108733 (106K) [image/jpeg]
Saving to: /myimages

/myimages           100%[===================>] 106.18K  --.-KB/s    in 0.01s

2020-05-02 12:16:04 (8.14 MB/s) - /myimages saved [108733/108733]

--2020-05-02 12:16:05--  https://cdn.pixabay.com/photo/2016/12/13/05/15/puppy-1903313_960_720.jp
Resolving cdn.pixabay.com (cdn.pixabay.com)... 104.18.20.183, 104.18.21.183, 2606:4700::6812:14b
Connecting to cdn.pixabay.com (cdn.pixabay.com)|104.18.20.183|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 85628 (84K) [image/jpeg]
Saving to: /myimages

/myimages           100%[===================>]  83.62K  --.-KB/s    in 0.01s

2020-05-02 12:16:05 (7.41 MB/s) - /myimages saved [85628/85628]
```

```
In [58]: !ls
```

```
beautiful-1274056_960_720.jpg  model-2387582_960_720.jpg
beautiful-1274361_960_720.jpg  model_scratch.pt
dog-1210559_960_720.jpg        model_transfer.pt
dog_app-cn.ipynb               puppy-1903313_960_720.jpg
dog_app.ipynb                       README.md
haarcascades                   weimaraner-1381186_960_720.jpg
images
```

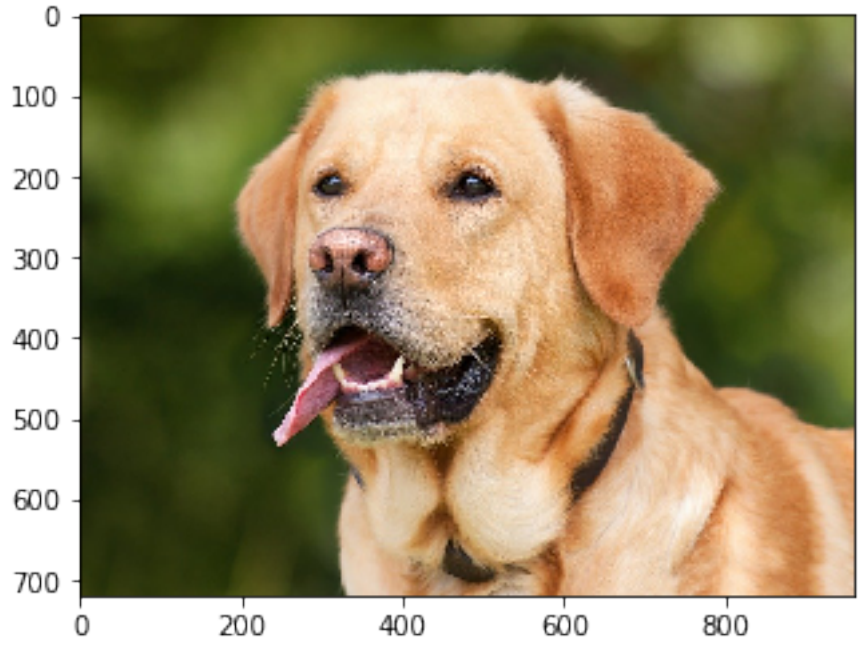
```
In [15]: my_human_files = ['beautiful-1274056_960_720.jpg', 'beautiful-1274361_960_720.jpg', 'we
         my_dog_files = ['dog-1210559_960_720.jpg', 'puppy-1903313_960_720.jpg', 'model-2387582_
```

In [27]:
```
## TODO: Execute your algorithm from Step 6 on
## at least 6 images on your computer.
## Feel free to use as many code cells as needed.

## suggested code, below
for file in np.hstack((my_human_files, my_dog_files)):
    run_app(file)
```

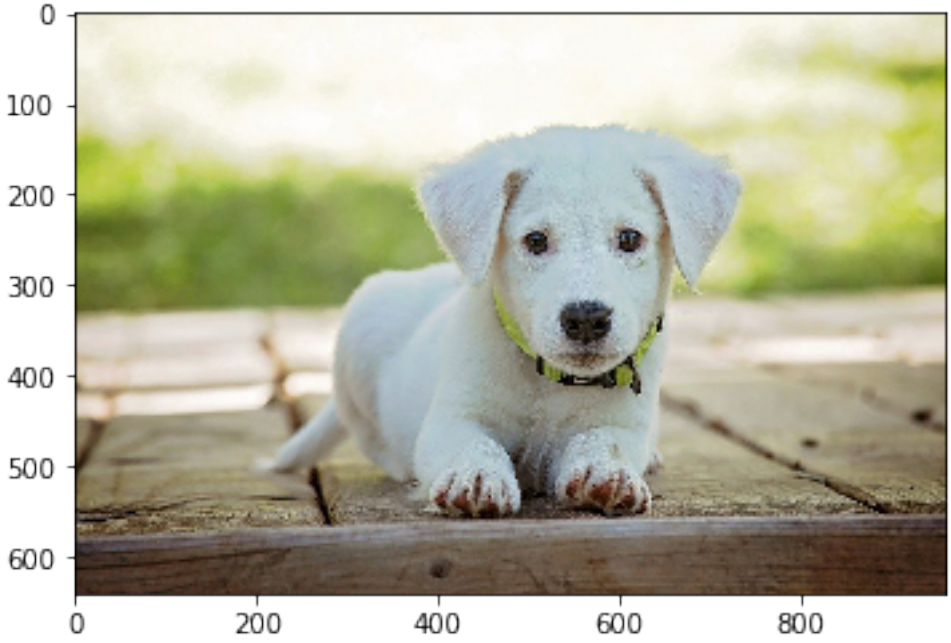

```
Hello, human!
If you were a dog..You may look like a English toy spaniel
```


```
Hello, human!
If you were a dog..You may look like a Lowchen
```

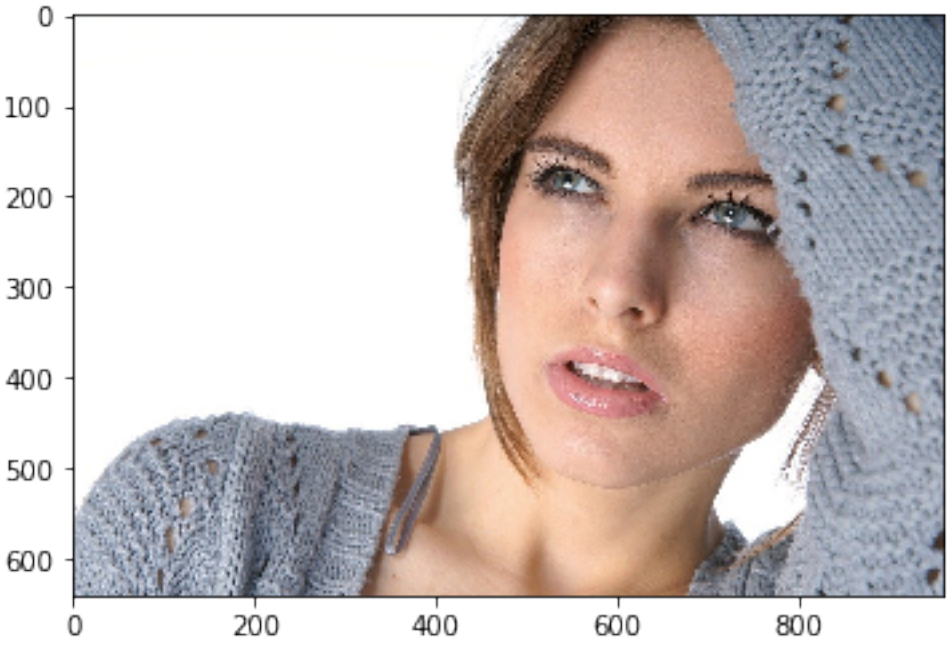
```
Dogs Detected!
It looks like a German shorthaired pointer
```




```
Dogs Detected!
It looks like a Golden retriever
```



```
Dogs Detected!
It looks like a Labrador retriever
```




```
Hello, human!
If you were a dog..You may look like a Bedlington terrier
```

In [ ]: