

# dog\_app

July 19, 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: \* Download the [dog dataset](#). Unzip the folder and place it in this project’s home directory, at the location `/dogImages`.

- 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`.

```
[47]: import numpy as np
      from glob import glob
      import cv2
      import matplotlib.pyplot as plt

      from tqdm import tqdm
      import torch
      import torchvision.models as models
      from PIL import Image
      import torchvision.transforms as transforms

      # Set PIL to be tolerant of image files that are truncated.
      from PIL import ImageFile
      import os
      from torchvision import datasets
      import torch.nn as nn
      import torch.nn.functional as F
      import torch.optim as optim
      from PIL import ImageFile
      import torchvision.models as models
      import torch.nn as nn
      from matplotlib.pyplot import imshow

      %matplotlib inline
```

```
[48]: # load filenames for human and dog images
      human_files = np.array(glob("lfw/**/*.jpg"))
      dog_files = np.array(glob("dogImages/**/*.jpg"))

      # 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.

## 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.

```
[49]: # 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))

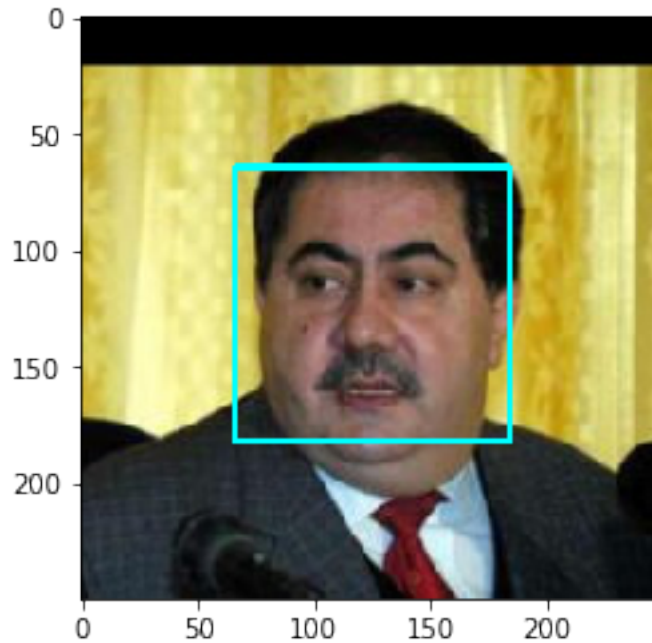
# get bounding box for each detected face
for (x,y,w,h) in faces:
    # add bounding box to color image
    cv2.rectangle(img,(x,y),(x+w,y+h),(255,255,0),2)

# 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.

```
[50]: # 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:**

human face detect accuracy = 98%

dog face detect accuracy = 13%

```
[51]: 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.

      human_correct = 0

      for h in human_files_short:
          if face_detector(h):
              human_correct += 1

      dog_correct = 0
      for d in dog_files_short:
          if face_detector(d):
              dog_correct += 1

      print(f'human face detect accuracy = {human_correct}%')
      print(f'dog face detect accuracy = {dog_correct}%')
```

human face detect accuracy = 98%

dog face detect accuracy = 13%

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`.

---

## 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.

```
[53]: # 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()
```

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](#).

```
[54]: ImageFile.LOAD_TRUNCATED_IMAGES = True

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
```

```
transform = transforms.Compose ([  
    transforms.Resize((224,224)),  
    transforms.ToTensor(),  
    transforms.Normalize( (0.5,0.5,0.5),  
                           (0.5,0.5,0.5))  
])
```

```
img = Image.open(img_path)  
img_transformed = transform(img).unsqueeze(0).cuda()  
out = VGG16(img_transformed)  
_, index = torch.max(out,1)
```

```
return index[0].item() # predicted class index
```

```
[55]: index = VGG16_predict('dogImages/train/001.Affenpinscher/Affenpinscher_00001.  
    ↪jpg')  
print(index)
```

252

```
[56]: img = Image.open('dogImages/train/001.Affenpinscher/Affenpinscher_00001.jpg')  
img
```

[56]:





### 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).

```
[57]: ### returns "True" if a dog is detected in the image stored at img_path
def dog_detector(img_path):

    index = VGG16_predict(img_path)
    if index >= 151 and index <= 268:
        return True
    return False
```

```
[58]: dog_detector('dogImages/train/001.Affenpinscher/Affenpinscher_00001.jpg')
```



[58]: True

### 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:

human face detect accuracy = 1%

dog face detect accuracy = 99%

```
[59]: ### TODO: Test the performance of the dog_detector function  
### on the images in human_files_short and dog_files_short.  
  
dog_in_human_files = 0  
  
for h in human_files_short:  
    if dog_detector(h):  
        dog_in_human_files += 1  
  
dog_correct = 0  
for d in dog_files_short:  
    if dog_detector(d):  
        dog_correct += 1  
  
print(f'human face detect accuracy = {dog_in_human_files}%')  
print(f'dog face detect accuracy = {dog_correct}%')
```

human face detect accuracy = 0%

dog face detect accuracy = 99%

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`.

---

### ## 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 imbalanced, 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 `dogImages/train`, `dogImages/valid`, and `dogImages/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](#)!

```
[62]: heights = []  
      widths = []
```

```

for d in tqdm(dog_files):
    img = Image.open(d)
    heights.append(img.height)
    widths.append(img.width)

print(f'mean height {np.mean(heights)}, mean width {np.mean(widths)}')

```

100%| | 8351/8351 [00:02<00:00, 2815.23it/s]

mean height 529.0449048018202, mean width 567.0325709495869

```

[63]: ### TODO: Write data loaders for training, validation, and test sets
      ## Specify appropriate transforms, and batch_sizes
      im_dim = 512

      transform = transforms.Compose([
          transforms.RandomRotation(20),
          transforms.RandomHorizontalFlip(0.2),
          transforms.Resize((im_dim,im_dim)),
          transforms.ToTensor(),
          transforms.Normalize( (0.5,0.5,0.5),(0.5,0.5,0.5))
      ])

      test_transform = transforms.Compose([
          transforms.Resize((im_dim,im_dim)),
          transforms.ToTensor(),
          transforms.Normalize( (0.5,0.5,0.5),(0.5,0.5,0.5))
      ])

      train_dataset = datasets.ImageFolder('dogImages/train/', transform=transform)
      valid_dataset = datasets.ImageFolder('dogImages/valid/', transform=transform)
      test_dataset = datasets.ImageFolder('dogImages/test/', transform=test_transform)

      batch_size = 64
      num_workers = 0
      train_loader = torch.utils.data.DataLoader(train_dataset,
          ↪batch_size=batch_size, num_workers=num_workers, shuffle=True)
      valid_loader = torch.utils.data.DataLoader(valid_dataset,
          ↪batch_size=batch_size, num_workers=num_workers, shuffle=True)
      test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=batch_size,
          ↪num_workers=num_workers, shuffle=True)

      loaders_scratch = {
          'train' : train_loader,

```

```

    'valid' : valid_loader,
    'test' : test_loader,
}

```

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:

Mean height and width of the dog images are aprox 529 and 568 respectively. So i chooses to resize the images to 512 which is nearest value of power of 2. (to help maximize GPU usage)

Then i am augmenting data by random rotations, horizontal flips, random crops. I am also normalizing the data. Data augmentation is only for train and valid dataset.

### 1.1.8 (IMPLEMENTATION) Model Architecture

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

```

[64]: dog_classes = set(np.array(glob("dogImages/train/*")))
      dog_num_classes = len(dog_classes)
      print(dog_num_classes)

```

133

```

[65]: # define the CNN architecture
      class Net(nn.Module):
          ### TODO: choose an architecture, and complete the class
          def __init__(self,im_dim=512,dog_num_classes=133):
              super(Net, self).__init__()
              ## Define layers of a CNN
              self.im_dim = im_dim
              # last conv layer has 32 depth, and 3 max pools are reducing image dim
              → to half each time
              out_im_dim = self.im_dim / 8
              self.lin_in = int(out_im_dim * out_im_dim * 64)
              self.features = nn.Sequential(
                  nn.Conv2d(3, 16, 3, padding=1, stride=1),
                  nn.ReLU(),
                  nn.MaxPool2d(2,2),
                  nn.Conv2d(16, 32, 3, padding=1, stride=1),
                  nn.ReLU(),
                  nn.MaxPool2d(2,2),
                  nn.Conv2d(32, 64, 3, padding=1, stride=1),

```

```

        nn.ReLU(),
        nn.MaxPool2d(2,2)
    )

    self.classifier = nn.Sequential(
        nn.Linear(self.lin_in,1024),
        nn.ReLU(),
        nn.Linear(1024,512),
        nn.ReLU(),
        nn.Linear(512,dog_num_classes),
    )

    def forward(self, x):
        ## Define forward behavior
        x = self.features(x)
        x = torch.flatten(x, start_dim=1)
        return self.classifier(x)

##-## You do NOT have to modify the code below this line. ##-##

# instantiate the CNN
model_scratch = Net()

# move tensors to GPU if CUDA is available
if use_cuda:
    model_scratch.cuda()

```

```
[66]: model_scratch
```

```

[66]: Net(
  (features): Sequential(
    (0): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (3): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): ReLU()
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (6): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU()
    (8): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (classifier): Sequential(
    (0): Linear(in_features=262144, out_features=1024, bias=True)
  )
)

```

```

(1): ReLU()
(2): Linear(in_features=1024, out_features=512, bias=True)
(3): ReLU()
(4): Linear(in_features=512, out_features=133, bias=True)
)
)

```

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

**Answer:**

I used Sequential module so that forward routine is trivial and model arch is easily describable. 3 series of conv2d layers increases the depth from 3 input channels to 16 and then to 32 and finally 64. Conv2d layers maintains the image dim same as input but maxpool2d would make it into half after each conv2d layer. Finally fully connected layer will transform output of conv2d layers to 512 len vector and finally to 133 which is number of classes of dogs. Each layer has Relu activation and last linear layer has sigmoid to get class probability

### 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.

```

[67]: ### 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'.

```

[68]: # the following import is required for training to be robust to truncated images

ImageFile.LOAD_TRUNCATED_IMAGES = True

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']):
    optimizer.zero_grad()
    # move to GPU
    if use_cuda:
        data, target = data.cuda(), target.cuda()
    ## find the loss and update the model parameters accordingly
    #print(f'data.shape {data.shape}')
    #print(f'target.shape {target.shape}')
    pred = model(data)
    #print(f'pred.shape {pred.shape}')
    loss = criterion(pred, target)
    loss.backward()
    optimizer.step()
    ## record the average training loss, using something like
    train_loss = train_loss + ((1 / (batch_idx + 1)) * (loss.data -
→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
    pred = model(data)
    vloss = criterion(pred, target)
    valid_loss += vloss.item()
    # print training/validation statistics
    valid_loss /= len(loaders['valid'])
    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_min > valid_loss:

```



```

        valid_loss_min = valid_loss
        print(f'saving model...{save_path}')
        torch.save(model.state_dict(), save_path)

    # return trained model
    return model

```

```

[69]: # train the model
model_scratch = train(20, loaders_scratch, model_scratch, optimizer_scratch,
                      criterion_scratch, use_cuda, 'model_scratch.pt')

```

Epoch: 1	Training Loss: 4.894238	Validation Loss: 4.785806
saving model...model_scratch.pt		
Epoch: 2	Training Loss: 4.638229	Validation Loss: 4.534277
saving model...model_scratch.pt		
Epoch: 3	Training Loss: 4.431794	Validation Loss: 4.433485
saving model...model_scratch.pt		
Epoch: 4	Training Loss: 4.213529	Validation Loss: 4.184511
saving model...model_scratch.pt		
Epoch: 5	Training Loss: 3.994592	Validation Loss: 4.100464
saving model...model_scratch.pt		
Epoch: 6	Training Loss: 3.807509	Validation Loss: 3.975826
saving model...model_scratch.pt		
Epoch: 7	Training Loss: 3.626223	Validation Loss: 4.041641
Epoch: 8	Training Loss: 3.452872	Validation Loss: 3.929790
saving model...model_scratch.pt		
Epoch: 9	Training Loss: 3.276466	Validation Loss: 3.989187
Epoch: 10	Training Loss: 3.024688	Validation Loss: 4.009722
Epoch: 11	Training Loss: 2.795467	Validation Loss: 4.172530
Epoch: 12	Training Loss: 2.511580	Validation Loss: 4.371286
Epoch: 13	Training Loss: 2.235348	Validation Loss: 4.602059
Epoch: 14	Training Loss: 1.922446	Validation Loss: 4.715621
Epoch: 15	Training Loss: 1.653450	Validation Loss: 5.403158
Epoch: 16	Training Loss: 1.436105	Validation Loss: 5.452211
Epoch: 17	Training Loss: 1.215536	Validation Loss: 5.770024
Epoch: 18	Training Loss: 1.088300	Validation Loss: 5.537965
Epoch: 19	Training Loss: 0.911539	Validation Loss: 6.389182
Epoch: 20	Training Loss: 0.811077	Validation Loss: 6.328019

### 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%.

```
[70]: 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
        #print(f'data shape = {data.shape}')
        output = model(data)
        #print(f'output shape = {output.shape}')
        # 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))
```

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

Test Loss: 3.999172

Test Accuracy: 10% (86/836)

---

## 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.

```
[72]: ## TODO: Specify data loaders

im_dim = 224

transform_transfer = transforms.Compose([
    transforms.RandomRotation(20),
    transforms.RandomHorizontalFlip(0.2),
    transforms.Resize((im_dim,im_dim)),
    transforms.ToTensor(),
    transforms.Normalize( (0.5,0.5,0.5),(0.5,0.5,0.5))
])

test_transform_transfer = transforms.Compose([
    transforms.Resize((im_dim,im_dim)),
    transforms.ToTensor(),
    transforms.Normalize( (0.5,0.5,0.5),(0.5,0.5,0.5))
])

train_dataset_transfer = datasets.ImageFolder('dogImages/train/',
    ↪transform=transform_transfer)
valid_dataset_transfer = datasets.ImageFolder('dogImages/valid/',
    ↪transform=transform_transfer)
test_dataset_transfer = datasets.ImageFolder('dogImages/test/',
    ↪transform=test_transform_transfer)

batch_size = 64
num_workers = 0
train_loader_transfer = torch.utils.data.DataLoader(train_dataset_transfer,
    ↪batch_size=batch_size, num_workers=num_workers, shuffle=True)
valid_loader_transfer = torch.utils.data.DataLoader(valid_dataset_transfer,
    ↪batch_size=batch_size, num_workers=num_workers, shuffle=True)
test_loader_transfer = torch.utils.data.DataLoader(test_dataset_transfer,
    ↪batch_size=batch_size, num_workers=num_workers, shuffle=True)

loaders_transfer = {
    'train' : train_loader_transfer,
```

```

    'valid' : valid_loader_transfer,
    'test' : test_loader_transfer,
}

```

### 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`.

```

[73]: ## TODO: Specify model architecture

class VGG16DOG(nn.Module):
    def __init__(self):
        super(VGG16DOG, self).__init__()
        self.model_vgg16 = models.vgg16(pretrained=True)

        for p in self.model_vgg16.features.parameters():
            p.requires_grad = False

        dog_pred_layer = nn.Sequential(nn.ReLU(),
                                       nn.Linear(1000, dog_num_classes),
                                       )

        self.model_vgg16.classifier.add_module('dog_pred_layer', dog_pred_layer)

    def parameters(self):
        return self.model_vgg16.classifier.parameters()

    def forward(self, x):
        return self.model_vgg16(x)

model_transfer = VGG16DOG()

if use_cuda:
    model_transfer = model_transfer.cuda()

```

```

[74]: print(model_transfer)

```

```

VGG16DOG(
  (model_vgg16): VGG(
    (features): Sequential(
      (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): ReLU(inplace=True)
      (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (3): ReLU(inplace=True)
      (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=True)
(7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(8): ReLU(inplace=True)
(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=True)
(12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(13): ReLU(inplace=True)
(14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(15): ReLU(inplace=True)
(16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
(17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(18): ReLU(inplace=True)
(19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(20): ReLU(inplace=True)
(21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(22): ReLU(inplace=True)
(23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
(24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(25): ReLU(inplace=True)
(26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(27): ReLU(inplace=True)
(28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(29): ReLU(inplace=True)
(30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
)
(avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
(classifier): Sequential(
  (0): Linear(in_features=25088, out_features=4096, bias=True)
  (1): ReLU(inplace=True)
  (2): Dropout(p=0.5, inplace=False)
  (3): Linear(in_features=4096, out_features=4096, bias=True)
  (4): ReLU(inplace=True)
  (5): Dropout(p=0.5, inplace=False)
  (6): Linear(in_features=4096, out_features=1000, bias=True)
  (dog_pred_layer): Sequential(
    (0): ReLU()
    (1): Linear(in_features=1000, 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:**

I created a new class that wraps the vgg19 model and adds new linear layer at the end in the classification stage. I inherited the parameters() methods to return only trainable params which are from classifier only.

#### 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.

```
[75]: criterion_transfer = nn.CrossEntropyLoss()
      optimizer_transfer = optim.Adam(model_transfer.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'.

```
[76]: # train the model
      n_epochs = 10
      model_transfer = train(n_epochs, loaders_transfer, model_transfer,
      ↪optimizer_transfer, criterion_transfer, use_cuda, 'model_transfer.pt')
```

Epoch: 1	Training Loss: 3.111947	Validation Loss: 1.686405
saving model...model_transfer.pt		
Epoch: 2	Training Loss: 1.990474	Validation Loss: 1.553914
saving model...model_transfer.pt		
Epoch: 3	Training Loss: 1.661002	Validation Loss: 1.193194
saving model...model_transfer.pt		
Epoch: 4	Training Loss: 1.503874	Validation Loss: 1.264740
Epoch: 5	Training Loss: 1.384212	Validation Loss: 1.067999
saving model...model_transfer.pt		
Epoch: 6	Training Loss: 1.353897	Validation Loss: 1.173093
Epoch: 7	Training Loss: 1.305635	Validation Loss: 1.127147
Epoch: 8	Training Loss: 1.233910	Validation Loss: 1.201705
Epoch: 9	Training Loss: 1.137117	Validation Loss: 1.171570
Epoch: 10	Training Loss: 1.171077	Validation Loss: 1.131685

### 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%.

```
[77]: # load the model that got the best validation accuracy (uncomment the line
      ↪below)
model_transfer.load_state_dict(torch.load('model_transfer.pt'))

with torch.no_grad():
    test(loaders_transfer, model_transfer, criterion_transfer, use_cuda)
```

Test Loss: 1.156848

Test Accuracy: 65% (544/836)

### 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.

```
[78]: ### 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 train_dataset_transfer.
               ↪classes]

def predict_breed_transfer(img_path):
    # load the image and return the predicted breed

    model_transfer = VGG16DOG()
    model_transfer.load_state_dict(torch.load('model_transfer.pt'))

    im_dim = 224
    test_transform_transfer = transforms.Compose([
        transforms.Resize((im_dim,im_dim)),
        transforms.ToTensor(),
        transforms.Normalize( (0.5,0.5,0.5),(0.5,0.5,0.5))
    ])

    img = Image.open(img_path)
    img = test_transform_transfer(img)
    img.unsqueeze_(0)
```



```

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

output = model_transfer(img)
_, index = torch.max(output,1)

return class_names[index[0].item()] # predicted class index

```

```

[79]: predict_breed_transfer('dogImages/train/001.Affenpinscher/Affenpinscher_00001.
    ↪jpg')

```

```

[79]: 'Affenpinscher'

```

---

### ## 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.


You are welcome to write your own functions for detecting humans and dogs in images, but feel free to use the `face_detector` and `dog_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!

```

hello, human!

```



```

You look like a ...
Chinese_shar-pei

```

### 1.1.18 (IMPLEMENTATION) Write your Algorithm

```
[80]: ### TODO: Write your algorithm.
      ### Feel free to use as many code cells as needed.
      %matplotlib inline
      def run_app(img_path):
          ## handle cases for a human face, dog, and neither

          if dog_detector(img_path):
              dog_breed = predict_breed_transfer(img_path)
              img = Image.open(img_path)
              plt.imshow(img)
              plt.show()
              print(str(dog_breed))
          elif face_detector(img_path):
              print(" hello human")
              img = Image.open(img_path)
              plt.imshow(img)
              plt.show()
              print("you look like a ..." + str(predict_breed_transfer(img_path)))
          else:
              print("Invalid image type")
```

---

### ## 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.

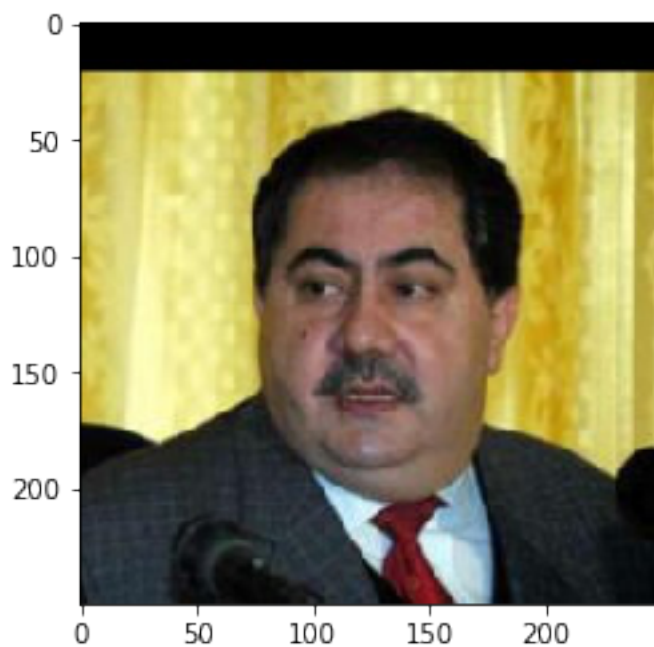
**Question 6:** Is the output better than you expected :) ? Or worse :( ? Provide at least three possible points of improvement for your algorithm.

**Answer:** (Three possible points for improvement)

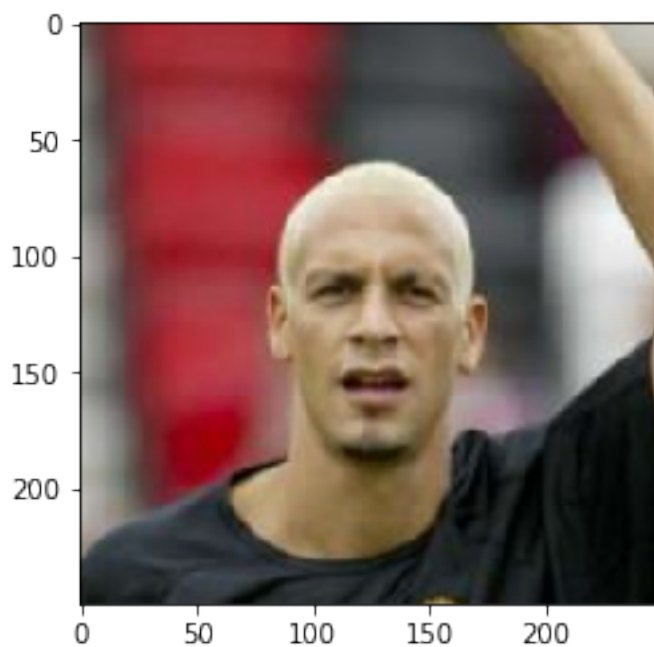
```
[81]: ## TODO: Execute your algorithm from Step 6 on
      ## at least 6 images on your computer.
      ## Feel free to use as many code cells as needed.

      ## suggested code, below
      for file in np.hstack((human_files[:3], dog_files[:3])):
          run_app(file)
```

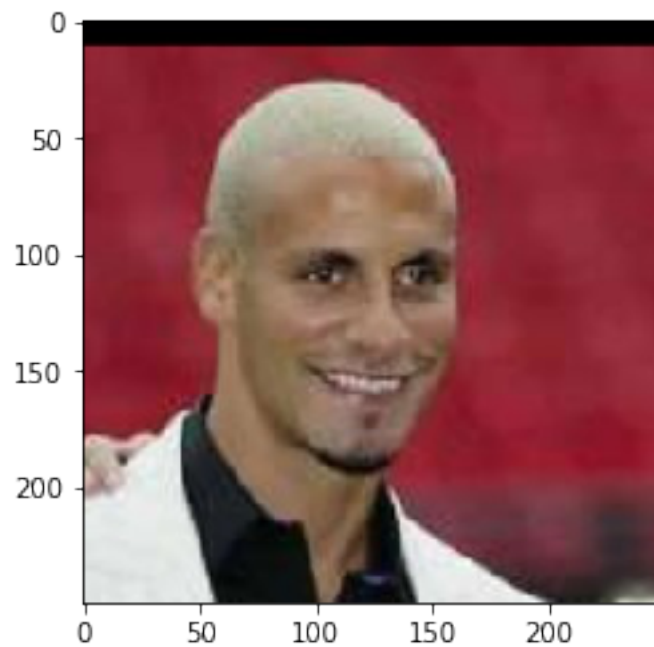
hello human



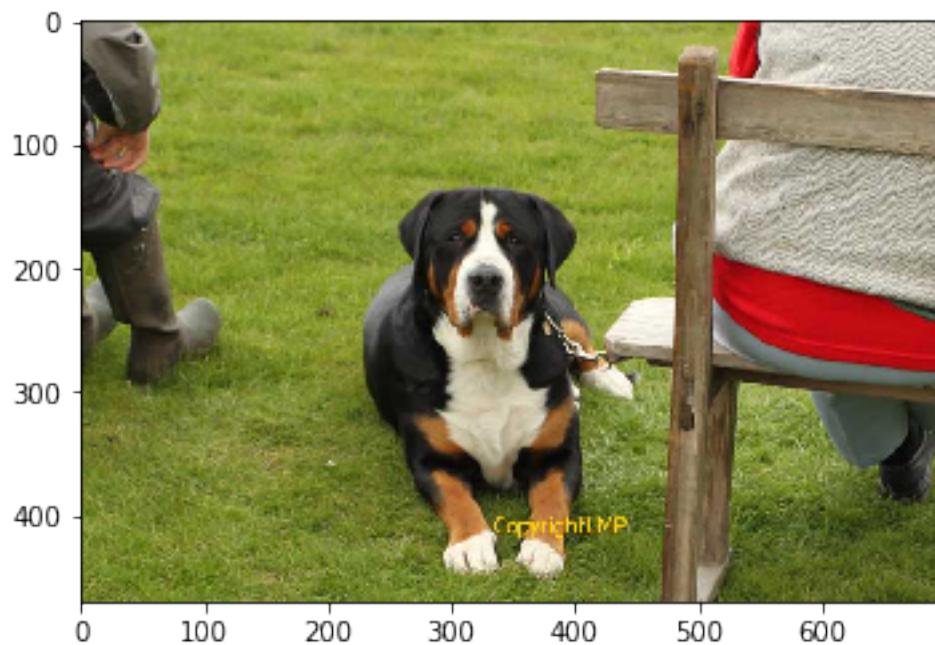
you look like a ...Pharaoh hound  
hello human



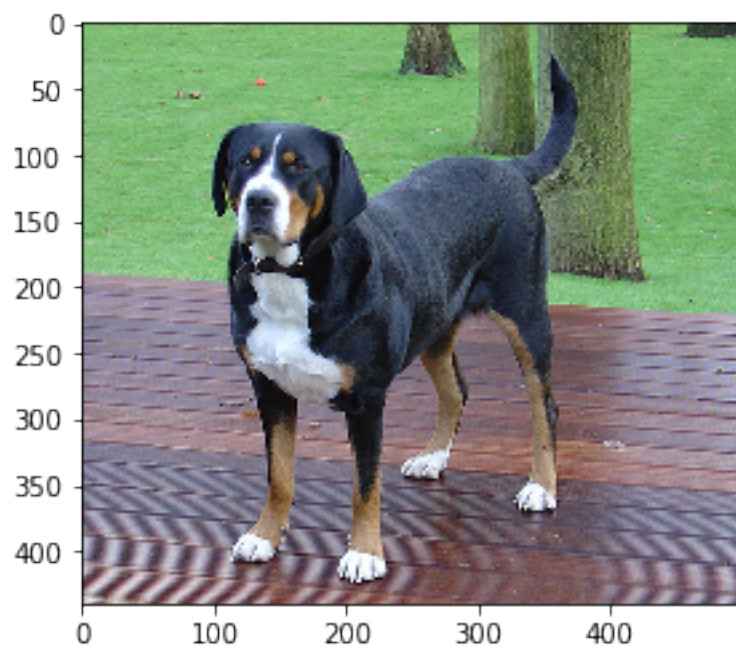
you look like a ...Dachshund  
hello human



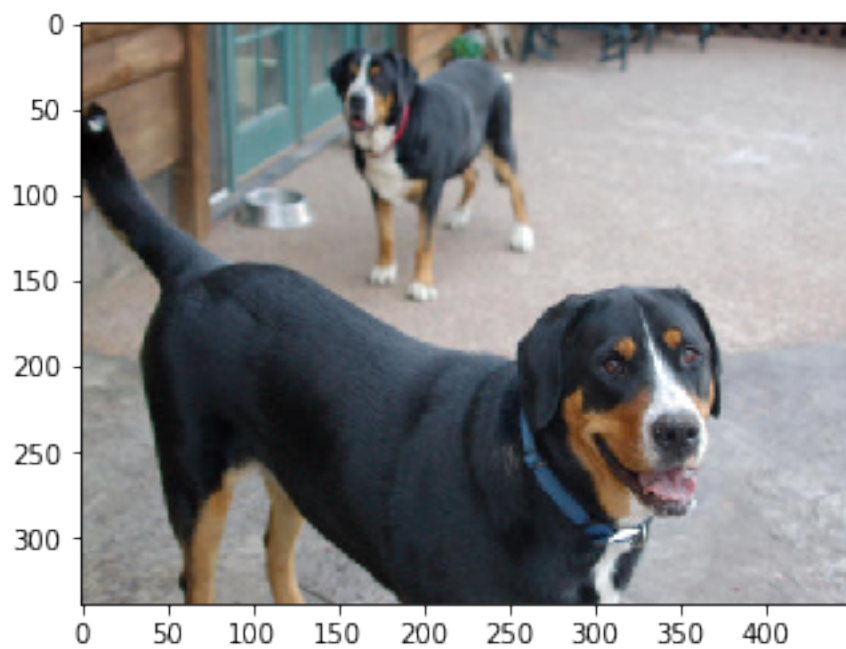
you look like a ...Chinese crested



Greater swiss mountain dog



Greater swiss mountain dog



Entlebucher mountain dog

[ ]:

[ ]:

[ ]:

[ ]: