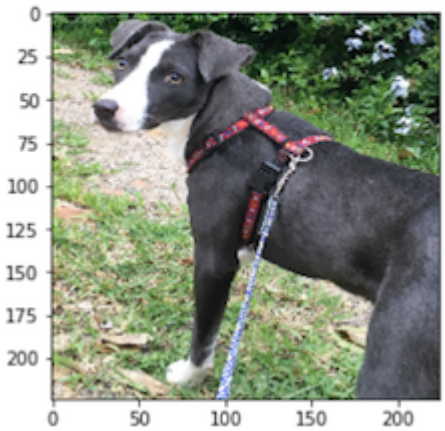
Convolutional Neural Networks

Project: Write an Algorithm for a Dog Identification App

Why We're Here

In this notebook, you will make the first steps towards developing an algorithm that could be used as part of a mobile or web app. At the end of this project, your code will accept any user-supplied image as input. If a dog is detected in the image, it will provide an estimate of the dog's breed. If a human is detected, it will provide an estimate of the dog breed that is most resembling. The image below displays potential sample output of your finished project (... but we expect that each student's algorithm will behave differently!).

```
hello, dog!
your predicted breed is ...
American Staffordshire terrier
```



In this real-world setting, you will need to piece together a series of models to perform different tasks; for instance, the algorithm that detects humans in an image will be different from the CNN that infers dog breed. There are many points of possible failure, and no perfect algorithm exists. Your imperfect solution will nonetheless create a fun user experience!

The Road Ahead

We break the notebook into separate steps. Feel free to use the links below to navigate the 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`.

```
!cd ..
!wget https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/dogImages.zip
!wget https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/lfw.zip
```

```
--2021-05-07 12:51:21--  https://s3-us-west-1.amazonaws.com/udacity-aind/dog-p
Resolving s3-us-west-1.amazonaws.com (s3-us-west-1.amazonaws.com)... 52.219.11
Connecting to s3-us-west-1.amazonaws.com (s3-us-west-1.amazonaws.com)|52.219.1
HTTP request sent, awaiting response... 200 OK
Length: 1132023110 (1.1G) [application/zip]
Saving to: 'dogImages.zip'

dogImages.zip       100%[===================>]   1.05G  20.7MB/s    in 62s

2021-05-07 12:52:24 (17.4 MB/s) - 'dogImages.zip' saved [1132023110/1132023110

--2021-05-07 12:52:24--  https://s3-us-west-1.amazonaws.com/udacity-aind/dog-p
Resolving s3-us-west-1.amazonaws.com (s3-us-west-1.amazonaws.com)... 52.219.12
Connecting to s3-us-west-1.amazonaws.com (s3-us-west-1.amazonaws.com)|52.219.1
HTTP request sent, awaiting response... 200 OK
Length: 196739509 (188M) [application/zip]
Saving to: 'lfw.zip'

lfw.zip             100%[===================>] 187.62M  19.6MB/s    in 11s

2021-05-07 12:52:36 (17.1 MB/s) - 'lfw.zip' saved [196739509/196739509]
```

```
!rm -rf __MACOSX/
!rm -rf dogImages/
!unzip dogImages.zip
!rm -rf lfw/
!unzip lfw.zip
;
```

```
import numpy as np
from glob import glob

# load filenames for human and dog images
```

```
human_files = np.array(glob("lfw/*/*"))
dog_files = np.array(glob("dogImages/*/*/*"))

# 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.
```

```
dog_names = [item[27:-1] for item in sorted(glob("dogImages/train/*"))]

print('Total dog classes: %d' % len(dog_names))
```

```
Total dog classes: 133
```

```
import matplotlib.pyplot as plt
%matplotlib inline

num_dogs = []
for files in sorted(glob("dogImages/train/*/")):
    num_dogs.append(len(files))


plt.figure(figsize=(25,5))
plt.xticks(rotation=90)
plt.bar(dog_names, num_dogs)
plt.xlabel('Classes')
plt.ylabel('# Dogs')

plt.show()
```

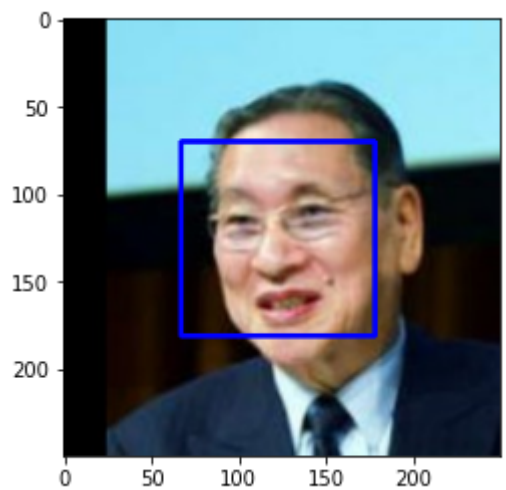
Step 1: Detect Humans

In this section, we use OpenCV's implementation of [Haar feature-based cascade classifiers](http://docs.opencv.org/trunk/d7/d8b/tutorial_py_face_detection.html) to detect human faces in images.

OpenCV provides many pre-trained face detectors, stored as XML files on [github](https://github.com/opencv/opencv/tree/master/data/haarcascades). 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.

```
import cv2

# extract pre-trained face detector
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalfa

# load color (BGR) image
img = cv2.imread(human_files[255])
# 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.

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.

```
# 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
```

(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)

```
from tqdm import tqdm

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

#-#-# Do NOT modify the code above this line. #-#-#
faces_count_human_files = 0
faces_count_dog_files = 0

for img in human_files_short:
    faces_count_human_files += face_detector(img)
```

```
    faces_count_human_files += face_detector(img)

for img in dog_files_short:
    faces_count_dog_files += face_detector(img)

print("{:.2f}% faces found on human files".format(faces_count_human_files))
print("{:.2f}% faces found on dogs files".format(faces_count_dog_files))
```

```
100.00% faces found on human files
8.00% faces found on dogs files
```

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`.

```
### (Optional)
### TODO: Test performance of another face detection algorithm.
### Feel free to use as many code cells as needed.
```

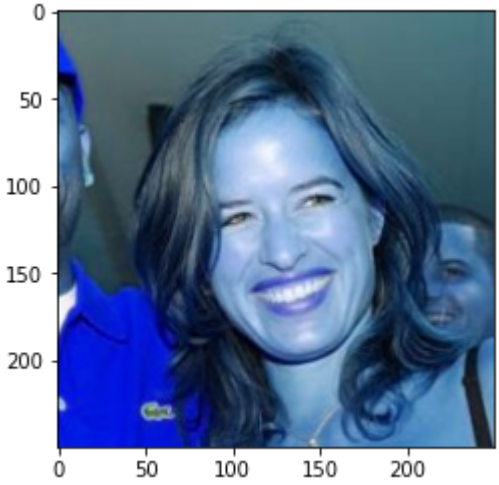
```
!pip install face-recognition
```

```
Collecting face-recognition
  Downloading https://files.pythonhosted.org/packages/1e/95/f6c9330f54ab07bfa0
Requirement already satisfied: Click>=6.0 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: Pillow in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: dlib>=19.7 in /usr/local/lib/python3.7/dist-pac
Collecting face-recognition-models>=0.3.0
  Downloading https://files.pythonhosted.org/packages/cf/3b/4fd8c534f6c0d1b80c
     |████████████████████████████████| 100.2MB 41kB/s
Building wheels for collected packages: face-recognition-models
  Building wheel for face-recognition-models (setup.py) ... done
  Created wheel for face-recognition-models: filename=face_recognition_models-
  Stored in directory: /root/.cache/pip/wheels/d2/99/18/59c6c8f01e39810415c0e6
Successfully built face-recognition-models
Installing collected packages: face-recognition-models, face-recognition
Successfully installed face-recognition-1.3.0 face-recognition-models-0.3.0
```

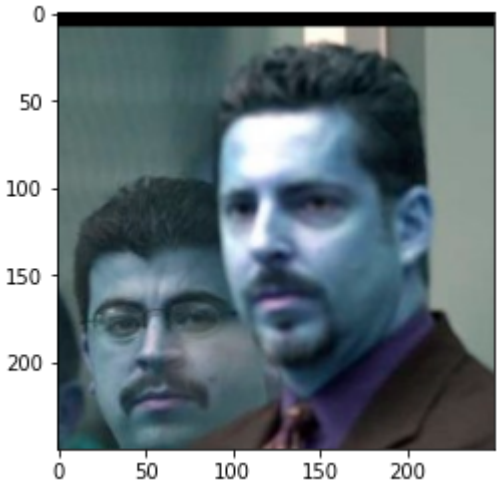
```
import face_recognition

faces_count_human_files = 0
faces_count_dog_files = 0

for img in human_files_short:
    image = face_recognition.load_image_file(img)
    face_locations = face_recognition.face_locations(image)
    faces_count_human_files += len(face_locations)
    if (len(face_locations) > 1):
        print("{} -> I found {} face(s) in this photograph.".format(img, len(face_l
        plt.imshow(cv2.imread(img))
        plt.show()
```

```
        plt.show()

for img in dog_files_short:
    image = face_recognition.load_image_file(img)
    face_locations = face_recognition.face_locations(image)
    faces_count_dog_files += len(face_locations)
    if (len(face_locations) > 0):
        print("{} -> I found {} face(s) in this photograph.".format(img, len(face_l
        plt.imshow(cv2.imread(img))
        plt.show()

print("{:.2f}% faces found on human files".format(faces_count_human_files))
print("{:.2f}% faces found on dogs files".format(faces_count_dog_files))
```

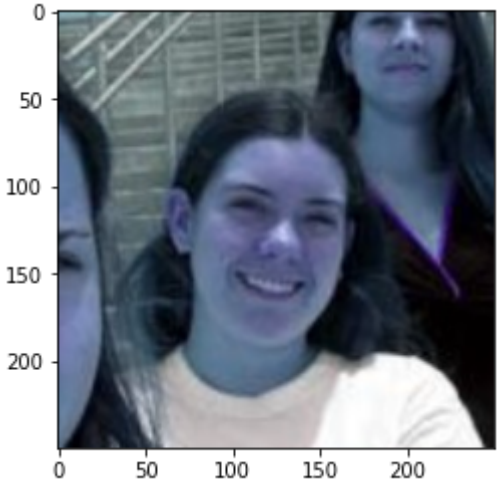
```
lfw/Jade_Jagger/Jade_Jagger_0001.jpg -> I found 2 face(s) in this photograph.
```



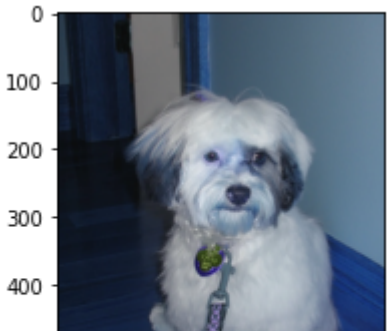
```
lfw/Philip_Zalewski/Philip_Zalewski_0001.jpg -> I found 2 face(s) in this phot
```



```
lfw/Colleen_Donovan/Colleen_Donovan_0001.jpg -> I found 2 face(s) in this phot
```



```
dogImages/test/082.Havanese/Havanese_05631.jpg -> I found 1 face(s) in this ph
```



Here I compared the performance between OpenCV with haarcascade_frontalface_alt.xml and `Face Recognition` library:

OpenCV haarcascade_frontalface_alt.xml:

- Humans detected in human_files_short: 100

- Humans detected in dog_files_short: 8

Face Recognition Lib:

- Humans detected in human_files_short: 101
- Humans detected in dog_files_short: 7

As we can see that we have a better performance with `Face Recognition` Lib, which detected all faces in human pictures including pictures with more than 1 visible face.

We also observed a decrease in the number of human faces detected in the dog_files_short. However, some dogs were still misclassified as human faces.

Step 2: Detect Dogs

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

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.

```
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
100%                                528M/528M [00:22<00:00, 24.6MB/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.

(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.

[ ] ↳ *2 cells hidden*

(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).

[ ] ↳ *1 cell hidden*

(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?

[ ] ↳ *10 cells hidden*

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

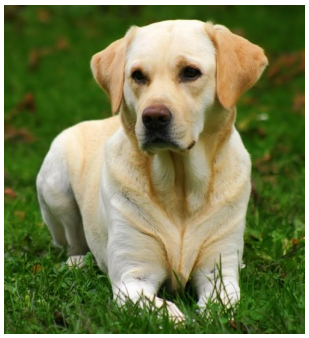
It is not difficult to find other dog breed pairs with minimal inter-class variation (for instance, Curly-Coated Retrievers and American Water Spaniels).

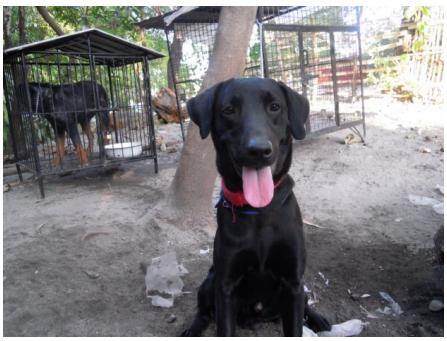
Curly-Coated Retriever	American Water Spaniel

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 | Black 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!

(IMPLEMENTATION) Specify Data Loaders for the Dog Dataset

Use the code cell below to write three separate [data loaders](http://pytorch.org/docs/master/data.html#torch.utils.data.DataLoader) 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](http://pytorch.org/docs/master/torchvision/datasets.html) to be a useful resource. If you are interested in augmenting your training and/or validation data, check out the wide variety of [transforms](http://pytorch.org/docs/master/torchvision/transforms.html)!

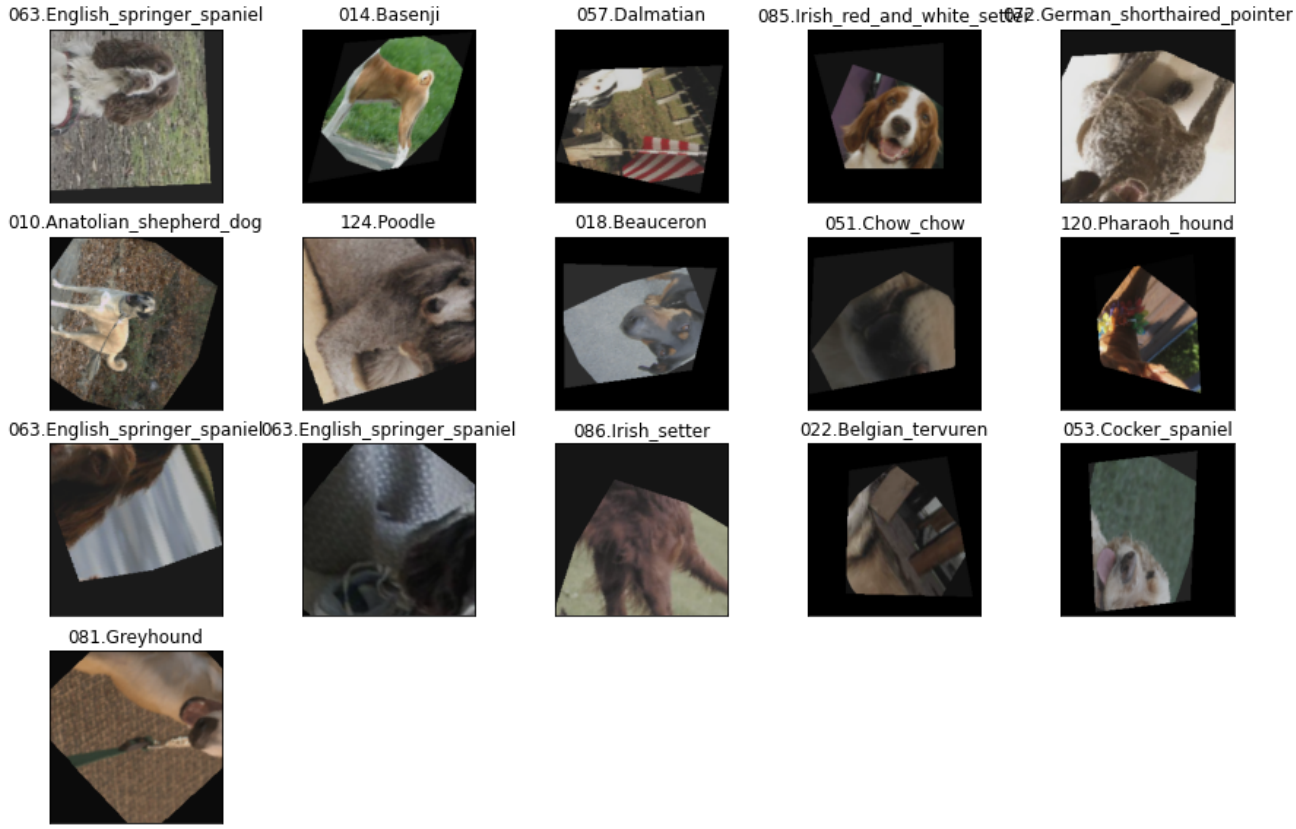
```
import os
import torch
from torchvision import datasets, transforms

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

data_dir = 'dogImages'

batch_size = 16
```

```
batch_size = 16

normalization = transforms.Normalize(mean=[0.485, 0.456, 0.406], # Normalizing the
                                      std=[0.229, 0.224, 0.225])

train_transforms = transforms.Compose([transforms.Resize(size=256),
                                       transforms.RandomResizedCrop(224),
                                       transforms.RandomHorizontalFlip(),
                                       transforms.RandomRotation(30),
                                       transforms.RandomAffine((0, 180), translate=
                                       transforms.ColorJitter(contrast=(0.5, 1.0),
                                       transforms.RandomPerspective(),
                                       transforms.ToTensor()])

validTest_transforms = transforms.Compose([transforms.Resize(size=256),
                                           transforms.CenterCrop(224),
                                           transforms.ToTensor()])

train_dataset = datasets.ImageFolder(os.path.join(data_dir, 'train'), transform=tra
valid_dataset = datasets.ImageFolder(os.path.join(data_dir, 'valid'), transform=val
test_dataset = datasets.ImageFolder(os.path.join(data_dir, 'test'), transform=valid

trainLoader = torch.utils.data.DataLoader(train_dataset,
                                          batch_size=batch_size,
                                          shuffle=True,
                                          num_workers=0)

validLoader = torch.utils.data.DataLoader(valid_dataset,
                                          batch_size=batch_size,
                                          shuffle=True,
                                          num_workers=0)

testLoader = torch.utils.data.DataLoader(test_dataset,
                                          batch_size=batch_size,
                                          shuffle=False,
                                          num_workers=0)
```

Check generated sample data

```
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline

iterator = iter(trainLoader)
images, labels = iterator.next()
images = images.numpy()

# plot the images in the batch, along with the corresponding labels
fig = plt.figure(figsize=(15,10))
for idx in np.arange(batch_size):
    ax = fig.add_subplot(4, 5, idx+1, xticks=[], yticks=[])
    plt.imshow(np.transpose(images[idx], (1, 2, 0)))
```

```
    ax.set_title(train_dataset.classes[labels[idx].numpy()])
```



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:

- *How does your code resize the images (by cropping, stretching, etc)? What size did you pick for the input tensor, and why?*
    - Images were resized to 256 then cropped to a size of 224 x 224. Since the images come in various sizes resizing and cropping them to an image of 224 x 224 should be good enough especially for training time.

- *Did you decide to augment the dataset? If so, how (through translations, flips, rotations, etc)? If not, why not?*
    - Yes, I have decided for data augmentation to improve the performance of the model. I applied a random resized crop, then a random horizontal flip and finally a random rotation of 10 degrees. It was also applied a few transformations, like Translation, Perspective distortion, Color Jitter

(IMPLEMENTATION) Model Architecture

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

[ ] *4 cells hidden*

(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.

[ ] *1 cell hidden*

(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'`.

[ ] *2 cells hidden*

(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%.

[ ] *1 cell hidden*

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.

(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.

```
## TODO: Specify data loaders
data_dir = 'dogImages'

batch_size = 16

train_transforms = transforms.Compose([transforms.Resize(size=256),
                                       transforms.RandomResizedCrop(224),
                                       transforms.RandomHorizontalFlip(),
                                       transforms.RandomRotation(30),
                                       #transforms.RandomAffine((0, 180), translate
                                       #transforms.ColorJitter(contrast=(0.5, 1.0),
                                       #transforms.RandomPerspective(),
                                       transforms.ToTensor()])

validTest_transforms = transforms.Compose([transforms.Resize(size=256),
                                           transforms.CenterCrop(224),
                                           transforms.ToTensor()])

train_dataset_transfer = datasets.ImageFolder(os.path.join(data_dir, 'train'), tran
valid_dataset_transfer = datasets.ImageFolder(os.path.join(data_dir, 'valid'), tran
test_dataset_transfer = datasets.ImageFolder(os.path.join(data_dir, 'test'), transf

trainLoader_transfer = torch.utils.data.DataLoader(train_dataset_transfer,
                                           batch_size=batch_size,
                                           shuffle=True,
                                           num_workers=0)

validLoader_transfer = torch.utils.data.DataLoader(valid_dataset_transfer,
                                           batch_size=batch_size,
                                           shuffle=True,
                                           num_workers=0)

testLoader_transfer = torch.utils.data.DataLoader(test_dataset_transfer,
                                           batch_size=batch_size,
                                           shuffle=False,
                                           num_workers=0)
```

(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`.

```
import torchvision.models as models
import torch.nn as nn

## TODO: Specify model architecture
model_transfer = models.vgg16(pretrained=True)

for param in model_transfer.parameters():
```

```
    param.requires_grad = False

# Define dog breed classifier
model_transfer.classifier = nn.Sequential(nn.Linear(25088, 4096), nn.ReLU(), nn.Dro

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

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 chose the VGG16 model as backbone using transfer learning approach considering it was trained with several dog images, among other classes. Then I freeze all the parameters and weights of the hidden layers need to be fixed (freezed) by executing the method requires_grad set to 'False'. Finally I trained only the classifier with 3 fully connected layers 2 dropout layers to reduce the number of parameters and then 2 ReLU activations.

(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.

```
criterion_transfer = nn.CrossEntropyLoss()
optimizer_transfer = optim.SGD(model_transfer.classifier.parameters(), lr= 0.001, m
#optimizer_transfer = optim.Adam(model_transfer.classifier.parameters(), lr= 0.001)
```

(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'`.

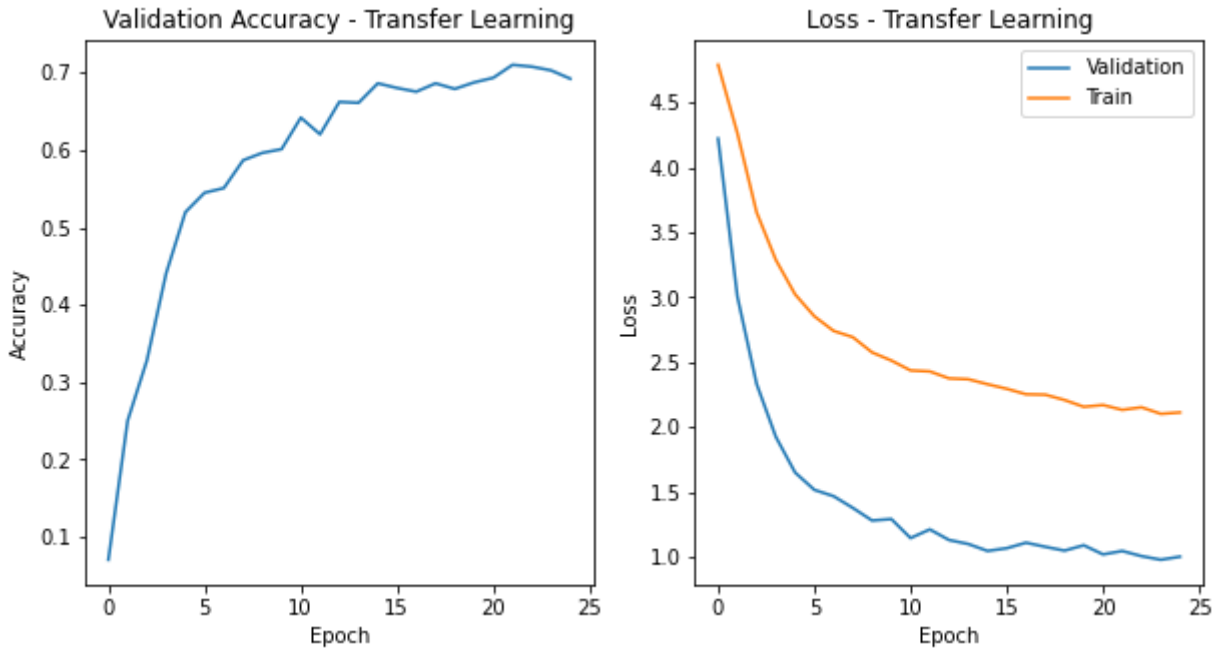
```
loaders_transfer = {'train': trainLoader_transfer, 'valid': validLoader_transfer, '

# train the model
model_transfer, train_loss_transfer, valid_loss_transfer, valid_acc_transfer = trai


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

```
    CUDA available... Using CUDA
    Epoch: 1
            Duration: 0:01:57.345298        Training Loss: 4.792060         Vali
    Validation loss decreased (inf --> 4.223718).  Saving model ...
```

```
Epoch: 2
        Duration: 0:01:58.782093        Training Loss: 4.266456         Vali
Validation loss decreased (4.223718 --> 3.007105).  Saving model ...
Epoch: 3
        Duration: 0:01:58.481012        Training Loss: 3.655352         Vali
Validation loss decreased (3.007105 --> 2.332606).  Saving model ...
Epoch: 4
        Duration: 0:01:58.414903        Training Loss: 3.288230         Vali
Validation loss decreased (2.332606 --> 1.923953).  Saving model ...
Epoch: 5
        Duration: 0:01:56.771905        Training Loss: 3.022867         Vali
Validation loss decreased (1.923953 --> 1.649451).  Saving model ...
Epoch: 6
        Duration: 0:01:58.155060        Training Loss: 2.853704         Vali
Validation loss decreased (1.649451 --> 1.515747).  Saving model ...
Epoch: 7
        Duration: 0:01:57.721400        Training Loss: 2.741003         Vali
Validation loss decreased (1.515747 --> 1.466867).  Saving model ...
Epoch: 8
        Duration: 0:01:58.140708        Training Loss: 2.692789         Vali
Validation loss decreased (1.466867 --> 1.376504).  Saving model ...
Epoch: 9
        Duration: 0:01:56.940037        Training Loss: 2.575206         Vali
Validation loss decreased (1.376504 --> 1.279289).  Saving model ...
Epoch: 10
        Duration: 0:01:56.644941        Training Loss: 2.512721         Vali
Epoch: 11
        Duration: 0:01:57.841955        Training Loss: 2.436260         Vali
Validation loss decreased (1.279289 --> 1.144045).  Saving model ...
Epoch: 12
        Duration: 0:01:57.939128        Training Loss: 2.428619         Vali
Epoch: 13
        Duration: 0:01:57.681754        Training Loss: 2.374759         Vali
Validation loss decreased (1.144045 --> 1.129050).  Saving model ...
Epoch: 14
        Duration: 0:01:57.099254        Training Loss: 2.368468         Vali
Validation loss decreased (1.129050 --> 1.098302).  Saving model ...
Epoch: 15
        Duration: 0:01:56.806719        Training Loss: 2.329848         Vali
Validation loss decreased (1.098302 --> 1.044848).  Saving model ...
Epoch: 16
        Duration: 0:01:56.866917        Training Loss: 2.294707         Vali
Epoch: 17
        Duration: 0:01:57.188658        Training Loss: 2.252424         Vali
Epoch: 18
        Duration: 0:01:56.825388        Training Loss: 2.249166         Vali
Epoch: 19
        Duration: 0:01:56.921753        Training Loss: 2.208196         Vali
Epoch: 20
        Duration: 0:01:57.491411        Training Loss: 2.155894         Vali
Epoch: 21
        Duration: 0:01:57.845575        Training Loss: 2.169488         Vali
Validation loss decreased (1.044848 --> 1.018228).  Saving model ...
Epoch: 22
```

```
fig, axs = plt.subplots(1, 2, figsize=(10,5))

# val accuracy
axs[0].set_title('Validation Accuracy - Transfer Learning')
axs[0].plot(valid_acc_transfer[1:])
```

```
    axs[0].set_ylabel('Accuracy')
    axs[0].set_xlabel('Epoch')

    # loss
    axs[1].set_title('Loss - Transfer Learning')
    axs[1].plot(valid_loss_transfer[1:])
    axs[1].plot(train_loss_transfer[1:])
    axs[1].set_ylabel('Loss')
    axs[1].set_xlabel('Epoch')
    axs[1].legend(['Validation', 'Train'], loc='upper right')

    fig.show()
```



(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%.

```
y_test_transfer, y_pred_transfer = test(loaders_transfer, model_transfer, criterion
```

SHOW HIDDEN OUTPUT

Overral Dog Breed Classification Report and Classification Performance per Class

[ ] ↳ *1 cell hidden*

(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.

[ ] ↳ *1 cell hidden*

Evaluating predictions

[ ] ↳ *1 cell hidden*

Step 5: Write your Algorithm

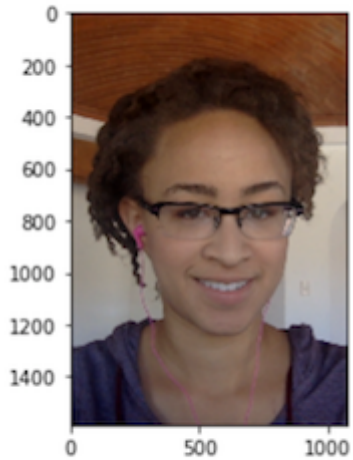
Write an algorithm that accepts a file path to an image and first determines whether the image contains a human, dog, or neither. Then,

- 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
```

(IMPLEMENTATION) Write your Algorithm

```
### TODO: Write your algorithm.
### Feel free to use as many code cells as needed.

def display_img(img_path):
    img = Image.open(img_path)
    plt.imshow(img)
    plt.show()


def run_app(img_path):
    ## handle cases for a human face, dog, and neither

    predicted_breed, top4_breed_names, top4_probabilites = predict_breed_transfer(i

    if dog_detector(img_path):
        print ('\n\n Woof Woof... it is a dog ')
        display_img(img_path)

        print ('Predicted breed:', predicted_breed)

        for k in range(4):
            print('Top {0:2} prediction:{1:7.2f}% - {2}'.format(k+2, top4_probabili

        return


    elif face_detector(img_path):
        print ('\n\n Hello, Human')
        display_img(img_path)

        print (' If you were a dog you could look like a ', predicted_breed)

        for k in range(4):
            print('Top {0:2} prediction:{1:7.2f}% - {2}'.format(k+2, top4_probabili

        return

    else:
        display_img(img_path)
        print ('\n\n error: Woow ,You are neither human, nor dog ')
    predict_breed_transfer
```

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?

(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:

The algorithm works pretty well, in general the predictions seem to be quite accurate.However there is always room for improvements:

- Improved face detecting algorithm, replacing OpenCV Haar Cascades by some Object Detection network trained to detect faces or use Dlib capabilities to do so.
- Better control the scenario where both dog and human are present in the same picture
- Try different alternatives for classifier layers of the transfer learning model. Try to chance only the FC layer and compare the performance.
- Expand training with more epochs and test the influence of the batch_size in the overall performance
- Try to improve model's accuracy by fine-tuning hyperparameters and testing a different backbone models (Inception, ResNet, VGG19)

```
## 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)
```