Convolutional Neural Networks

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.

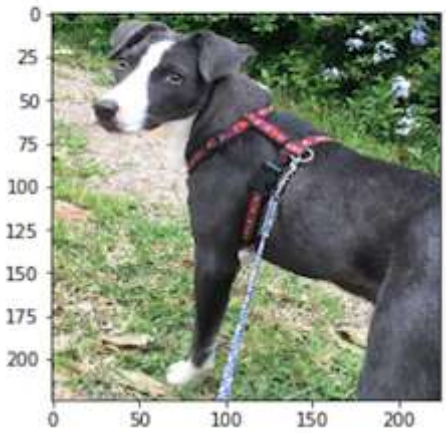
> **Note:** Code and Markdown cells can be executed using the **Shift + Enter** keyboard shortcut. Markdown cells can be edited by double-clicking the cell to enter edit mode.

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.

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
- Step 1: Detect Humans
- Step 2: Detect Dogs
- Step 3: Create a CNN to Classify Dog Breeds (from Scratch)
- Step 4: Create a CNN to Classify Dog Breeds (using Transfer Learning)
- Step 5: Write your Algorithm
- Step 6: Test Your Algorithm

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`.

In [ ]:
```
# !wget https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/dogImages.zip
# !unzip dogImages.zip
# !rm dogImages.zip

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

In [ ]:
```
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.
```

Step 1: Detect Humans

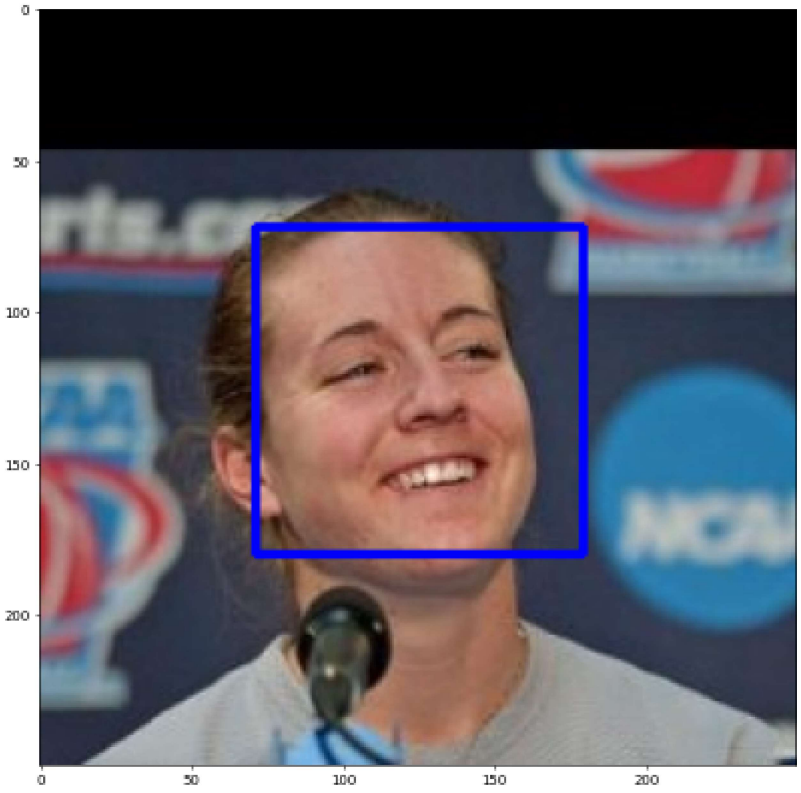
In this section, we use OpenCV's implementation of Haar feature-based cascade classifiers to detect human faces in images.

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

plt.rcParams['figure.figsize'] = [15, 10]
plt.rcParams['figure.dpi'] = 50

# 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.

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 [ ]:
```
# 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:

Human faces detected in dog files: 7%

Human faces detected in human files: 99%

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.

faces_found_in_dogs = 0
faces_found_in_humans = 0

file_list = tqdm(zip(human_files_short, dog_files_short))
for human_file, dog_file in file_list:
    if face_detector(human_file):
        faces_found_in_humans += 1
    if face_detector(dog_file):
        faces_found_in_dogs += 1

print('Human faces detected in dog files: {}%'.format(faces_found_in_dogs))
print('Human faces detected in human files: {}%'.format(faces_found_in_humans))
```

```
100it [00:06, 16.36it/s]
Human faces detected in dog files: 7%
Human faces detected in human files: 99%
```

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 another face 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.

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 [ ]:
```
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()
```

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.

In [ ]:
```
import json
```

```
from PIL import Image
import torchvision.transforms as transforms

# Set PIL to be tolerant of image files that are truncated.
from PIL import ImageFile
ImageFile.LOAD_TRUNCATED_IMAGES = True

class_names = None
with open('imagenet1000_clsidx_to_labels.txt', 'r') as f:
    class_names = json.loads(f.read())


def Predict(model: torch.nn.Module, img_path: str):
    '''
    Use pre-trained model to obtain index corresponding to
    predicted ImageNet class for image at specified path

    Args:
        model:    torch model to use and make the prediction
        img_path: path to an image

    Returns:
        Index corresponding to 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

    TX = transforms.Compose([
        transforms.ToTensor(),
        transforms.Resize((300, 300)),
        transforms.Normalize(
            mean=[0.485, 0.456, 0.406],
            std=[0.229, 0.224, 0.225]),
        ])

    X = None
    with Image.open(img_path) as img:
        X = torch.unsqueeze(TX(img), 0)

    if use_cuda:
        X = X.cuda()

    model.eval()
    Y = model(X)
    predicted = torch.max(Y, 1)[1]

    return predicted # predicted class index
```

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

(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:

Dogs detected in dog files: 100%

Dogs detected in human files: 0%

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

def dog_vs_human_eval(model, dogs, humans):
    dogs_found_in_dogs = 0
    dogs_found_in_humans = 0

    files = tqdm(zip(humans, dogs))
    for human_file, dog_file in files:
        if dog_detector(model, human_file):
            dogs_found_in_humans += 1
        if dog_detector(model, dog_file):
            dogs_found_in_dogs += 1

    print('Dogs detected in dog files: {}%'.format(dogs_found_in_dogs))
    print('Dogs detected in human files: {}%'.format(dogs_found_in_humans))

%time dog_vs_human_eval(VGG16, dog_files_short, human_files_short)
```

```
0it [00:00, ?it/s]/home/dakkar/.conda/envs/udacity/lib/python3.9/site-packages/torch/nn/
functional.py:718: UserWarning: Named tensors and all their associated APIs are an exper
imental feature and subject to change. Please do not use them for anything important unt
il they are released as stable. (Triggered internally at  /opt/conda/conda-bld/pytorch_1
623448238472/work/c10/core/TensorImpl.h:1156.)
  return torch.max_pool2d(input, kernel_size, stride, padding, dilation, ceil_mode)
100it [00:02, 34.27it/s]
Dogs detected in dog files: 100%
Dogs detected in human files: 0%
CPU times: user 34 s, sys: 973 ms, total: 35 s
Wall time: 2.92 s
```

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 [ ]:

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

MobileNetV3_small = models.mobilenet_v3_small(pretrained=True)

if use_cuda:
    MobileNetV3_small = MobileNetV3_small.cuda()

%time dog_vs_human_eval(MobileNetV3_small, dog_files_short, human_files_short)
```

```
100it [00:01, 60.95it/s]
Dogs detected in dog files: 100%
Dogs detected in human files: 0%
CPU times: user 19 s, sys: 677 ms, total: 19.7 s
Wall time: 1.64 s
```

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	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 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!

In [ ]:

```
import os
from torchvision import datasets


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

class DogDataset(datasets.ImageFolder):
    def __init__(self, root, batch_size=32, size=(512, 512), txs=None, load=False, save
```

```
        super(DogDataset, self).__init__(root)
        np.random.shuffle(self.imgs)

        self.txs = transforms.Normalize(
                mean=[0.485, 0.456, 0.406],
                std=[0.229, 0.224, 0.225]
            ) if txs is None else txs

        self.to_tensor = transforms.Compose([
            transforms.ToTensor(),
            transforms.Resize(size),
        ])

        self.batch_size = batch_size
        self.img_size = size
        self.memory = None

        if load:
            self.load()
            if save:
                self.save(overwrite)

    def __len__(self):
        return int(np.ceil(len(self.imgs) / self.batch_size))

    def __getitem__(self, index):
        start = self.batch_size * index
        end = start + self.batch_size
        if start >= len(self.imgs):
            raise StopIteration
        if end > len(self.imgs):
            end = len(self.imgs)

        if self.memory is not None:
            return self.txs(self.memory['x'][start:end]), self.memory['y'][start:end]
        else:
            total = end - start
            batchx = torch.empty((total, 3, *self.img_size))
            batchy = torch.empty((total), dtype=torch.int64)
            for bdx, idx in enumerate(range(start, end)):
                x, y = super(DogDataset, self).__getitem__(idx)
                batchx[bdx, ...] = self.to_tensor(x)
                batchy[bdx] = int(y)
            return batchx, batchy

    def load(self, reload=False, use_disk=True):
        print('Load {} data to memory...'.format(self.root))

        if self.memory is not None:
            print('Data already loaded in memory.')
            if not reload:
                return
            print('Reloading...')

        xcache_path = os.path.join(self.root, 'xcache.pt')
        ycache_path = os.path.join(self.root, 'ycache.pt')
        cache_exists = os.path.exists(xcache_path) and os.path.exists(ycache_path)

        if cache_exists and use_disk:
            print('Loading image files using cache files...')
```

```
            self.memory = {
                'x': torch.load(xcache_path),
                'y': torch.load(ycache_path)
            }
        else:
            print('Loading image files one by one...')
            total = len(self.imgs)
            self.memory = {
                'x': torch.empty((total, 3, *self.img_size)),
                'y': torch.empty((total), dtype=torch.int64)
            }
            for imgx in tqdm(range(total)):
                x, y = super(DogDataset, self).__getitem__(imgx)
                self.memory['x'][imgx, ...] = self.to_tensor(x)
                self.memory['y'][imgx] = int(y)

    def save(self, overwrite=False):
        print('Saving {} cache to disk...'.format(self.root))

        if self.memory is None:
            print('Data not loaded in memory. Call `load` first')
            return

        xcache_path = os.path.join(self.root, 'xcache.pt')
        ycache_path = os.path.join(self.root, 'ycache.pt')
        cache_exists = os.path.exists(xcache_path) and os.path.exists(ycache_path)
        if cache_exists:
            print('Cache already exists.')
            if not overwrite:
                return
            print('Overwrite...')
        torch.save(self.memory['x'], xcache_path)
        torch.save(self.memory['y'], ycache_path)

image_shape=(320, 320)

augmentations = transforms.Compose([
    transforms.ColorJitter(hue=0.1, contrast=0.5, saturation=0.5),
    transforms.RandomHorizontalFlip(),
    transforms.RandomAffine(0, translate=(.15, .15)),
    transforms.RandomResizedCrop(image_shape, scale=(.75, 1)),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

loaders_scratch = {
    'train': DogDataset('dogImages/train', 20, size=image_shape, txs=augmentations, loa
    'valid': DogDataset('dogImages/valid', 8, size=image_shape, load=True),
    'test':  DogDataset('dogImages/test' , 8, size=image_shape, load=True),
}
```

```
Load dogImages/train data to memory...
Loading image files using cache files...
Load dogImages/valid data to memory...
Loading image files using cache files...
Load dogImages/test data to memory...
Loading image files using cache files...
```

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:

- Use 320x320 to avoid having a very large model but with an acceptable resolutions to have good results.
- Color jitter: this transformation changes the hue, contrast and saturation with probabilities of 0.1, 0.5 and 0.5 respectively. I'm interested in teaching the model to distinguish the same dog shape without the proper color information, hence the change in hue. Hue changes are suppressed in the second set of transformation, since I'm expecting the model to distinguish color variations as much as possible.
- Random mirror: this transformation applies a horizontal flip to the image with a 0.5 probability of occurrence each time.
- Random translation: with this transformation we teach the model how a dog looks like in different positions in the image.
- Random resize and crop: this transformation crops the image at random location and resize it to a specific size, which in this case is the expected model input size, which is 320x320.
- Normalization: this transformation normalizes the image using the mean and standard deviation values recommended in Pytorch documentation [6]. For the R, G and B channels, these values are [0.485, 0.456, 0.406] for mean and [0.229, 0.224, 0.225] for standard deviation. These values were calculated using millions of color images.

(IMPLEMENTATION) Model Architecture

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

In [ ]:

```
import torch.nn as nn
import torch.nn.functional as F
from collections import namedtuple

def scale(filters, multiplier=1.0, base=8):
  rounded = int(int(filters) * multiplier / base + 0.5)
  result = int(rounded * base)
  return max(result, base)

class Swish6(nn.Module):
    def __init__(self):
        super(Swish6, self).__init__()
    def forward(self, x):
        return x * F.relu6(x + 3.) * (1. / 6.)

class GlobalAvgPool2d(nn.Module):
    def __init__(self, reshape=False):
        super(GlobalAvgPool2d, self).__init__()
        self.reshape = reshape
    def forward(self, x):
        out = F.avg_pool2d(x, x.shape[2:], stride=1, padding=0)
        return out.reshape(out.shape[:2]) if self.reshape else out
```

```
class Reshape(nn.Module):
    def __init__(self, shape):
        super(Reshape, self).__init__()
        self.shape = shape
    def forward(self, x):
        return x.reshape(x.shape[0], *self.shape)

class RegularConv2d(nn.Module):
    Params = namedtuple('RegularConv2dParams', 'filters kernel_size stride norm activat
    def __init__(self, inputs, filters, kernel_size=3, stride=1, norm=nn.BatchNorm2d, a
        super(RegularConv2d, self).__init__()
        layers = [ nn.Conv2d(inputs, filters, kernel_size, stride=stride, padding=kerne
        if norm:
            layers.append(norm(filters, eps=1e-3, momentum=0.1))
        layers.append(activation())
        self.net = nn.Sequential(*layers)
    def forward(self, x):
        return self.net(x)

class SqueezeAndExcite(nn.Module):
    def __init__(self, inputs, filters, activation=nn.ReLU6):
        super(SqueezeAndExcite, self).__init__()
        self.net = nn.Sequential(
            GlobalAvgPool2d(),
            RegularConv2d(inputs, filters, 1, norm=None, activation=activation),
            RegularConv2d(filters, inputs, 1, norm=None, activation=nn.Sigmoid)
        )
    def forward(self, x):
        return self.net(x) * x

class DepthwiseConv2d(nn.Module):
    def __init__(self, inputs, filters, kernel_size, stride=1, activation=nn.ReLU6):
        super(DepthwiseConv2d, self).__init__()
        self.net = nn.Sequential(
            nn.Conv2d(inputs, filters, kernel_size, stride=stride, padding=kernel_size/
            nn.BatchNorm2d(filters, eps=1e-3, momentum=0.1),
            activation()
        )
    def forward(self, x):
        return self.net(x)

class Bottleneck(nn.Module):
    Params = namedtuple('BottleneckParams', 'filters kernel_size stride expansion use_r
    def __init__(self, inputs, filters, kernel_size=3, stride=1, expansion=4, use_resid
        super(Bottleneck, self).__init__()
        expand_filters = int(inputs) * expansion
        layers = []
        if expansion > 1:
            layers.append(RegularConv2d(inputs, expand_filters, 1, activation=activatio
        layers.append(DepthwiseConv2d(expand_filters, expand_filters, kernel_size, stri
        if use_se:
            hidden_dim = scale(expand_filters, 0.25)
            layers.append(SqueezeAndExcite(expand_filters, hidden_dim, activation))
        layers.append(RegularConv2d(expand_filters, filters, 1, activation=nn.Identity)
        self.net = nn.Sequential(*layers)
        self.use_residual = stride == 1 and inputs == filters and use_residual
    def forward(self, x):
        out = self.net(x)
        return out + x if self.use_residual else out
```

```
class Fused(nn.Module):
    Params = namedtuple('FusedParams', 'filters kernel_size stride expansion use_residu
    def __init__(self, inputs, filters, kernel_size=3, stride=1, expansion=8, use_resid
        super(Fused, self).__init__()
        expand_filters = int(inputs) * expansion
        layers = [ RegularConv2d(inputs, expand_filters, kernel_size, stride, activatio
        if use_se:
            hidden_dim = scale(expand_filters, 0.25)
            layers.append(SqueezeAndExcite(expand_filters, hidden_dim, activation))
        layers.append(RegularConv2d(expand_filters, filters, 1, activation=nn.Identity)
        self.net = nn.Sequential(*layers)
        self.use_residual = stride == 1 and inputs == filters and use_residual
    def forward(self, x):
        out = self.net(x)
        return out + x if self.use_residual else out

class Tucker(nn.Module):
    Params = namedtuple('TuckerParams', 'filters kernel_size stride in_ratio out_ratio
    def __init__(self, inputs, filters, kernel_size=3, stride=1, in_ratio=0.25, out_rat
        super(Tucker, self).__init__()
        self.use_residual = stride == 1 and inputs == filters and use_residual
        hidden_in = scale(inputs, in_ratio)
        hidden_out = scale(filters, out_ratio)
        self.net = nn.Sequential(
            RegularConv2d(inputs, hidden_in, 1, activation=activation),
            RegularConv2d(hidden_in, hidden_out, kernel_size, stride, activation=activa
            RegularConv2d(hidden_out, filters, 1, activation=nn.Identity),
        )
    def forward(self, x):
        out = self.net(x)
        return out + x if self.use_residual else out

LAYERS = {
    'conv': RegularConv2d,
    'fused': Fused,
    'tucker': Tucker,
    'bottleneck': Bottleneck
}

SETTINGS = [
    ('conv'      , RegularConv2d.Params(16, stride=2)),
    ('bottleneck', Bottleneck.Params(8, expansion=1)),
    ('fused'     , Fused.Params(16, stride=2, expansion=4)),
    ('bottleneck', Bottleneck.Params(16)),
    ('tucker'    , Tucker.Params(16, out_ratio=0.75)),
    ('tucker'    , Tucker.Params(16, out_ratio=0.75)),
    ('bottleneck', Bottleneck.Params(32, 5, 2, 8)),
    ('bottleneck', Bottleneck.Params(32)),
    ('tucker'    , Tucker.Params(32, in_ratio=0.75, out_ratio=0.75)),
    ('tucker'    , Tucker.Params(32, in_ratio=0.75, out_ratio=0.75)),
    ('bottleneck', Bottleneck.Params(72, 5, 2, 8)),
    ('bottleneck', Bottleneck.Params(72)),
    ('bottleneck', Bottleneck.Params(72)),
    ('tucker'    , Tucker.Params(72, out_ratio=0.75)),
    ('bottleneck', Bottleneck.Params(96, kernel_size=5, expansion=8)),
    ('bottleneck', Bottleneck.Params(96, expansion=8)),
    ('bottleneck', Bottleneck.Params(96, kernel_size=5, expansion=8)),
    ('bottleneck', Bottleneck.Params(96, kernel_size=5)),
    ('bottleneck', Bottleneck.Params(104, kernel_size=5, stride=2)),
```

```
    ('tucker'    , Tucker.Params(104, 3, 1, 0.25, 0.75)),
    ('tucker'    , Tucker.Params(104, 3, 1, 0.75, 0.75)),
    ('tucker'    , Tucker.Params(104, 3, 1, 0.25, 0.75)),
    ('bottleneck', Bottleneck.Params(192, kernel_size=5, expansion=8))
]

# define the CNN architecture
class Net(nn.Module):
    ### TODO: choose an architecture, and complete the class
    def __init__(self, n_classes=133, settings=SETTINGS, hidden_top=256, dropout=0.25,
        super(Net, self).__init__()
        ## Define layers of a CNN
        inputs = 3
        layers = []
        for layer, params in settings:
            if layer in ['fused', 'bottleneck']:
                params = params._replace(use_se=use_se)
            params = params._replace(filters=scale(params[0], depth_multiplier), activa
            layers.append(LAYERS[layer](inputs, **params._asdict()))
            inputs = params[0]
        layers.extend([
            GlobalAvgPool2d(True),
            nn.Dropout(dropout),
            nn.Linear(inputs, hidden_top),
            nn.Linear(hidden_top, n_classes)
        ])
        self.net = nn.Sequential(*layers)

    def forward(self, x):
        ## Define forward behavior
        return self.net(x)

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

# instantiate the CNN
model_scratch = Net(hidden_top=512, use_se=True, depth_multiplier=1.5)

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

from torchsummary import summary
summary(model_scratch, (3, *image_shape))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 24, 160, 160]             648
       BatchNorm2d-2         [-1, 24, 160, 160]              48
            Swish6-3         [-1, 24, 160, 160]               0
    RegularConv2d-4          [-1, 24, 160, 160]               0
            Conv2d-5         [-1, 24, 160, 160]             216
       BatchNorm2d-6         [-1, 24, 160, 160]              48
            Swish6-7         [-1, 24, 160, 160]               0
  DepthwiseConv2d-8          [-1, 24, 160, 160]               0
  GlobalAvgPool2d-9              [-1, 24, 1, 1]               0
           Conv2d-10               [-1, 8, 1, 1]             200
           Swish6-11               [-1, 8, 1, 1]               0
   RegularConv2d-12               [-1, 8, 1, 1]               0
           Conv2d-13              [-1, 24, 1, 1]             216
```

```
         Sigmoid-14            [-1, 24, 1, 1]               0
   RegularConv2d-15            [-1, 24, 1, 1]               0
SqueezeAndExcite-16         [-1, 24, 160, 160]               0
          Conv2d-17         [-1, 16, 160, 160]             384
     BatchNorm2d-18         [-1, 16, 160, 160]              32
        Identity-19         [-1, 16, 160, 160]               0
   RegularConv2d-20         [-1, 16, 160, 160]               0
      Bottleneck-21         [-1, 16, 160, 160]               0
          Conv2d-22           [-1, 64, 80, 80]           9,216
     BatchNorm2d-23           [-1, 64, 80, 80]             128
          Swish6-24           [-1, 64, 80, 80]               0
   RegularConv2d-25           [-1, 64, 80, 80]               0
 GlobalAvgPool2d-26             [-1, 64, 1, 1]               0
          Conv2d-27             [-1, 16, 1, 1]           1,040
          Swish6-28             [-1, 16, 1, 1]               0
   RegularConv2d-29             [-1, 16, 1, 1]               0
          Conv2d-30             [-1, 64, 1, 1]           1,088
         Sigmoid-31             [-1, 64, 1, 1]               0
   RegularConv2d-32             [-1, 64, 1, 1]               0
SqueezeAndExcite-33           [-1, 64, 80, 80]               0
          Conv2d-34           [-1, 24, 80, 80]           1,536
     BatchNorm2d-35           [-1, 24, 80, 80]              48
        Identity-36           [-1, 24, 80, 80]               0
   RegularConv2d-37           [-1, 24, 80, 80]               0
           Fused-38           [-1, 24, 80, 80]               0
          Conv2d-39           [-1, 96, 80, 80]           2,304
     BatchNorm2d-40           [-1, 96, 80, 80]             192
          Swish6-41           [-1, 96, 80, 80]               0
   RegularConv2d-42           [-1, 96, 80, 80]               0
          Conv2d-43           [-1, 96, 80, 80]             864
     BatchNorm2d-44           [-1, 96, 80, 80]             192
          Swish6-45           [-1, 96, 80, 80]               0
 DepthwiseConv2d-46           [-1, 96, 80, 80]               0
 GlobalAvgPool2d-47             [-1, 96, 1, 1]               0
          Conv2d-48             [-1, 24, 1, 1]           2,328
          Swish6-49             [-1, 24, 1, 1]               0
   RegularConv2d-50             [-1, 24, 1, 1]               0
          Conv2d-51             [-1, 96, 1, 1]           2,400
         Sigmoid-52             [-1, 96, 1, 1]               0
   RegularConv2d-53             [-1, 96, 1, 1]               0
SqueezeAndExcite-54           [-1, 96, 80, 80]               0
          Conv2d-55           [-1, 24, 80, 80]           2,304
     BatchNorm2d-56           [-1, 24, 80, 80]              48
        Identity-57           [-1, 24, 80, 80]               0
   RegularConv2d-58           [-1, 24, 80, 80]               0
      Bottleneck-59           [-1, 24, 80, 80]               0
          Conv2d-60            [-1, 8, 80, 80]             192
     BatchNorm2d-61            [-1, 8, 80, 80]              16
          Swish6-62            [-1, 8, 80, 80]               0
   RegularConv2d-63            [-1, 8, 80, 80]               0
          Conv2d-64           [-1, 16, 80, 80]           1,152
     BatchNorm2d-65           [-1, 16, 80, 80]              32
          Swish6-66           [-1, 16, 80, 80]               0
   RegularConv2d-67           [-1, 16, 80, 80]               0
          Conv2d-68           [-1, 24, 80, 80]             384
     BatchNorm2d-69           [-1, 24, 80, 80]              48
        Identity-70           [-1, 24, 80, 80]               0
   RegularConv2d-71           [-1, 24, 80, 80]               0
          Tucker-72           [-1, 24, 80, 80]               0
          Conv2d-73            [-1, 8, 80, 80]             192
```

BatchNorm2d-74	[-1, 8, 80, 80]	16
Swish6-75	[-1, 8, 80, 80]	0
RegularConv2d-76	[-1, 8, 80, 80]	0
Conv2d-77	[-1, 16, 80, 80]	1,152
BatchNorm2d-78	[-1, 16, 80, 80]	32
Swish6-79	[-1, 16, 80, 80]	0
RegularConv2d-80	[-1, 16, 80, 80]	0
Conv2d-81	[-1, 24, 80, 80]	384
BatchNorm2d-82	[-1, 24, 80, 80]	48
Identity-83	[-1, 24, 80, 80]	0
RegularConv2d-84	[-1, 24, 80, 80]	0
Tucker-85	[-1, 24, 80, 80]	0
Conv2d-86	[-1, 192, 80, 80]	4,608
BatchNorm2d-87	[-1, 192, 80, 80]	384
Swish6-88	[-1, 192, 80, 80]	0
RegularConv2d-89	[-1, 192, 80, 80]	0
Conv2d-90	[-1, 192, 40, 40]	4,800
BatchNorm2d-91	[-1, 192, 40, 40]	384
Swish6-92	[-1, 192, 40, 40]	0
DepthwiseConv2d-93	[-1, 192, 40, 40]	0
GlobalAvgPool2d-94	[-1, 192, 1, 1]	0
Conv2d-95	[-1, 48, 1, 1]	9,264
Swish6-96	[-1, 48, 1, 1]	0
RegularConv2d-97	[-1, 48, 1, 1]	0
Conv2d-98	[-1, 192, 1, 1]	9,408
Sigmoid-99	[-1, 192, 1, 1]	0
RegularConv2d-100	[-1, 192, 1, 1]	0
SqueezeAndExcite-101	[-1, 192, 40, 40]	0
Conv2d-102	[-1, 48, 40, 40]	9,216
BatchNorm2d-103	[-1, 48, 40, 40]	96
Identity-104	[-1, 48, 40, 40]	0
RegularConv2d-105	[-1, 48, 40, 40]	0
Bottleneck-106	[-1, 48, 40, 40]	0
Conv2d-107	[-1, 192, 40, 40]	9,216
BatchNorm2d-108	[-1, 192, 40, 40]	384
Swish6-109	[-1, 192, 40, 40]	0
RegularConv2d-110	[-1, 192, 40, 40]	0
Conv2d-111	[-1, 192, 40, 40]	1,728
BatchNorm2d-112	[-1, 192, 40, 40]	384
Swish6-113	[-1, 192, 40, 40]	0
DepthwiseConv2d-114	[-1, 192, 40, 40]	0
GlobalAvgPool2d-115	[-1, 192, 1, 1]	0
Conv2d-116	[-1, 48, 1, 1]	9,264
Swish6-117	[-1, 48, 1, 1]	0
RegularConv2d-118	[-1, 48, 1, 1]	0
Conv2d-119	[-1, 192, 1, 1]	9,408
Sigmoid-120	[-1, 192, 1, 1]	0
RegularConv2d-121	[-1, 192, 1, 1]	0
SqueezeAndExcite-122	[-1, 192, 40, 40]	0
Conv2d-123	[-1, 48, 40, 40]	9,216
BatchNorm2d-124	[-1, 48, 40, 40]	96
Identity-125	[-1, 48, 40, 40]	0
RegularConv2d-126	[-1, 48, 40, 40]	0
Bottleneck-127	[-1, 48, 40, 40]	0
Conv2d-128	[-1, 40, 40, 40]	1,920
BatchNorm2d-129	[-1, 40, 40, 40]	80
Swish6-130	[-1, 40, 40, 40]	0
RegularConv2d-131	[-1, 40, 40, 40]	0
Conv2d-132	[-1, 40, 40, 40]	14,400
BatchNorm2d-133	[-1, 40, 40, 40]	80

```
         Swish6-134          [-1, 40, 40, 40]              0
   RegularConv2d-135          [-1, 40, 40, 40]              0
          Conv2d-136          [-1, 48, 40, 40]          1,920
     BatchNorm2d-137          [-1, 48, 40, 40]             96
        Identity-138          [-1, 48, 40, 40]              0
   RegularConv2d-139          [-1, 48, 40, 40]              0
          Tucker-140          [-1, 48, 40, 40]              0
          Conv2d-141          [-1, 40, 40, 40]          1,920
     BatchNorm2d-142          [-1, 40, 40, 40]             80
         Swish6-143          [-1, 40, 40, 40]              0
   RegularConv2d-144          [-1, 40, 40, 40]              0
          Conv2d-145          [-1, 40, 40, 40]         14,400
     BatchNorm2d-146          [-1, 40, 40, 40]             80
         Swish6-147          [-1, 40, 40, 40]              0
   RegularConv2d-148          [-1, 40, 40, 40]              0
          Conv2d-149          [-1, 48, 40, 40]          1,920
     BatchNorm2d-150          [-1, 48, 40, 40]             96
        Identity-151          [-1, 48, 40, 40]              0
   RegularConv2d-152          [-1, 48, 40, 40]              0
          Tucker-153          [-1, 48, 40, 40]              0
          Conv2d-154         [-1, 384, 40, 40]         18,432
     BatchNorm2d-155         [-1, 384, 40, 40]            768
         Swish6-156         [-1, 384, 40, 40]              0
   RegularConv2d-157         [-1, 384, 40, 40]              0
          Conv2d-158         [-1, 384, 20, 20]          9,600
     BatchNorm2d-159         [-1, 384, 20, 20]            768
         Swish6-160         [-1, 384, 20, 20]              0
 DepthwiseConv2d-161         [-1, 384, 20, 20]              0
 GlobalAvgPool2d-162           [-1, 384, 1, 1]              0
          Conv2d-163            [-1, 96, 1, 1]         36,960
         Swish6-164            [-1, 96, 1, 1]              0
   RegularConv2d-165            [-1, 96, 1, 1]              0
          Conv2d-166           [-1, 384, 1, 1]         37,248
         Sigmoid-167           [-1, 384, 1, 1]              0
   RegularConv2d-168           [-1, 384, 1, 1]              0
SqueezeAndExcite-169         [-1, 384, 20, 20]              0
          Conv2d-170         [-1, 112, 20, 20]         43,008
     BatchNorm2d-171         [-1, 112, 20, 20]            224
        Identity-172         [-1, 112, 20, 20]              0
   RegularConv2d-173         [-1, 112, 20, 20]              0
      Bottleneck-174         [-1, 112, 20, 20]              0
          Conv2d-175         [-1, 448, 20, 20]         50,176
     BatchNorm2d-176         [-1, 448, 20, 20]            896
         Swish6-177         [-1, 448, 20, 20]              0
   RegularConv2d-178         [-1, 448, 20, 20]              0
          Conv2d-179         [-1, 448, 20, 20]          4,032
     BatchNorm2d-180         [-1, 448, 20, 20]            896
         Swish6-181         [-1, 448, 20, 20]              0
 DepthwiseConv2d-182         [-1, 448, 20, 20]              0
 GlobalAvgPool2d-183           [-1, 448, 1, 1]              0
          Conv2d-184           [-1, 112, 1, 1]         50,288
         Swish6-185           [-1, 112, 1, 1]              0
   RegularConv2d-186           [-1, 112, 1, 1]              0
          Conv2d-187           [-1, 448, 1, 1]         50,624
         Sigmoid-188           [-1, 448, 1, 1]              0
   RegularConv2d-189           [-1, 448, 1, 1]              0
SqueezeAndExcite-190         [-1, 448, 20, 20]              0
          Conv2d-191         [-1, 112, 20, 20]         50,176
     BatchNorm2d-192         [-1, 112, 20, 20]            224
        Identity-193         [-1, 112, 20, 20]              0
```

```
   RegularConv2d-194         [-1, 112, 20, 20]              0
      Bottleneck-195         [-1, 112, 20, 20]              0
          Conv2d-196         [-1, 448, 20, 20]         50,176
     BatchNorm2d-197         [-1, 448, 20, 20]            896
          Swish6-198         [-1, 448, 20, 20]              0
   RegularConv2d-199         [-1, 448, 20, 20]              0
          Conv2d-200         [-1, 448, 20, 20]          4,032
     BatchNorm2d-201         [-1, 448, 20, 20]            896
          Swish6-202         [-1, 448, 20, 20]              0
 DepthwiseConv2d-203         [-1, 448, 20, 20]              0
 GlobalAvgPool2d-204           [-1, 448, 1, 1]              0
          Conv2d-205           [-1, 112, 1, 1]         50,288
          Swish6-206           [-1, 112, 1, 1]              0
   RegularConv2d-207           [-1, 112, 1, 1]              0
          Conv2d-208           [-1, 448, 1, 1]         50,624
         Sigmoid-209           [-1, 448, 1, 1]              0
   RegularConv2d-210           [-1, 448, 1, 1]              0
SqueezeAndExcite-211         [-1, 448, 20, 20]              0
          Conv2d-212         [-1, 112, 20, 20]         50,176
     BatchNorm2d-213         [-1, 112, 20, 20]            224
        Identity-214         [-1, 112, 20, 20]              0
   RegularConv2d-215         [-1, 112, 20, 20]              0
      Bottleneck-216         [-1, 112, 20, 20]              0
          Conv2d-217          [-1, 32, 20, 20]          3,584
     BatchNorm2d-218          [-1, 32, 20, 20]             64
          Swish6-219          [-1, 32, 20, 20]              0
   RegularConv2d-220          [-1, 32, 20, 20]              0
          Conv2d-221          [-1, 88, 20, 20]         25,344
     BatchNorm2d-222          [-1, 88, 20, 20]            176
          Swish6-223          [-1, 88, 20, 20]              0
   RegularConv2d-224          [-1, 88, 20, 20]              0
          Conv2d-225         [-1, 112, 20, 20]          9,856
     BatchNorm2d-226         [-1, 112, 20, 20]            224
        Identity-227         [-1, 112, 20, 20]              0
   RegularConv2d-228         [-1, 112, 20, 20]              0
          Tucker-229         [-1, 112, 20, 20]              0
          Conv2d-230         [-1, 896, 20, 20]        100,352
     BatchNorm2d-231         [-1, 896, 20, 20]          1,792
          Swish6-232         [-1, 896, 20, 20]              0
   RegularConv2d-233         [-1, 896, 20, 20]              0
          Conv2d-234         [-1, 896, 20, 20]         22,400
     BatchNorm2d-235         [-1, 896, 20, 20]          1,792
          Swish6-236         [-1, 896, 20, 20]              0
 DepthwiseConv2d-237         [-1, 896, 20, 20]              0
 GlobalAvgPool2d-238           [-1, 896, 1, 1]              0
          Conv2d-239           [-1, 224, 1, 1]        200,928
          Swish6-240           [-1, 224, 1, 1]              0
   RegularConv2d-241           [-1, 224, 1, 1]              0
          Conv2d-242           [-1, 896, 1, 1]        201,600
         Sigmoid-243           [-1, 896, 1, 1]              0
   RegularConv2d-244           [-1, 896, 1, 1]              0
SqueezeAndExcite-245         [-1, 896, 20, 20]              0
          Conv2d-246         [-1, 144, 20, 20]        129,024
     BatchNorm2d-247         [-1, 144, 20, 20]            288
        Identity-248         [-1, 144, 20, 20]              0
   RegularConv2d-249         [-1, 144, 20, 20]              0
      Bottleneck-250         [-1, 144, 20, 20]              0
          Conv2d-251        [-1, 1152, 20, 20]        165,888
     BatchNorm2d-252        [-1, 1152, 20, 20]          2,304
          Swish6-253        [-1, 1152, 20, 20]              0
```

```
   RegularConv2d-254        [-1, 1152, 20, 20]               0
          Conv2d-255        [-1, 1152, 20, 20]          10,368
     BatchNorm2d-256        [-1, 1152, 20, 20]           2,304
          Swish6-257        [-1, 1152, 20, 20]               0
 DepthwiseConv2d-258        [-1, 1152, 20, 20]               0
 GlobalAvgPool2d-259          [-1, 1152, 1, 1]               0
          Conv2d-260           [-1, 288, 1, 1]         332,064
          Swish6-261           [-1, 288, 1, 1]               0
   RegularConv2d-262           [-1, 288, 1, 1]               0
          Conv2d-263          [-1, 1152, 1, 1]         332,928
         Sigmoid-264          [-1, 1152, 1, 1]               0
   RegularConv2d-265          [-1, 1152, 1, 1]               0
SqueezeAndExcite-266        [-1, 1152, 20, 20]               0
          Conv2d-267         [-1, 144, 20, 20]         165,888
     BatchNorm2d-268         [-1, 144, 20, 20]             288
        Identity-269         [-1, 144, 20, 20]               0
   RegularConv2d-270         [-1, 144, 20, 20]               0
      Bottleneck-271         [-1, 144, 20, 20]               0
          Conv2d-272        [-1, 1152, 20, 20]         165,888
     BatchNorm2d-273        [-1, 1152, 20, 20]           2,304
          Swish6-274        [-1, 1152, 20, 20]               0
   RegularConv2d-275        [-1, 1152, 20, 20]               0
          Conv2d-276        [-1, 1152, 20, 20]          28,800
     BatchNorm2d-277        [-1, 1152, 20, 20]           2,304
          Swish6-278        [-1, 1152, 20, 20]               0
 DepthwiseConv2d-279        [-1, 1152, 20, 20]               0
 GlobalAvgPool2d-280          [-1, 1152, 1, 1]               0
          Conv2d-281           [-1, 288, 1, 1]         332,064
          Swish6-282           [-1, 288, 1, 1]               0
   RegularConv2d-283           [-1, 288, 1, 1]               0
          Conv2d-284          [-1, 1152, 1, 1]         332,928
         Sigmoid-285          [-1, 1152, 1, 1]               0
   RegularConv2d-286          [-1, 1152, 1, 1]               0
SqueezeAndExcite-287        [-1, 1152, 20, 20]               0
          Conv2d-288         [-1, 144, 20, 20]         165,888
     BatchNorm2d-289         [-1, 144, 20, 20]             288
        Identity-290         [-1, 144, 20, 20]               0
   RegularConv2d-291         [-1, 144, 20, 20]               0
      Bottleneck-292         [-1, 144, 20, 20]               0
          Conv2d-293         [-1, 576, 20, 20]          82,944
     BatchNorm2d-294         [-1, 576, 20, 20]           1,152
          Swish6-295         [-1, 576, 20, 20]               0
   RegularConv2d-296         [-1, 576, 20, 20]               0
          Conv2d-297         [-1, 576, 20, 20]          14,400
     BatchNorm2d-298         [-1, 576, 20, 20]           1,152
          Swish6-299         [-1, 576, 20, 20]               0
 DepthwiseConv2d-300         [-1, 576, 20, 20]               0
 GlobalAvgPool2d-301           [-1, 576, 1, 1]               0
          Conv2d-302           [-1, 144, 1, 1]          83,088
          Swish6-303           [-1, 144, 1, 1]               0
   RegularConv2d-304           [-1, 144, 1, 1]               0
          Conv2d-305           [-1, 576, 1, 1]          83,520
         Sigmoid-306           [-1, 576, 1, 1]               0
   RegularConv2d-307           [-1, 576, 1, 1]               0
SqueezeAndExcite-308         [-1, 576, 20, 20]               0
          Conv2d-309         [-1, 144, 20, 20]          82,944
     BatchNorm2d-310         [-1, 144, 20, 20]             288
        Identity-311         [-1, 144, 20, 20]               0
   RegularConv2d-312         [-1, 144, 20, 20]               0
      Bottleneck-313         [-1, 144, 20, 20]               0
```

```
          Conv2d-314         [-1, 576, 20, 20]          82,944
     BatchNorm2d-315         [-1, 576, 20, 20]           1,152
          Swish6-316         [-1, 576, 20, 20]               0
   RegularConv2d-317         [-1, 576, 20, 20]               0
          Conv2d-318         [-1, 576, 10, 10]          14,400
     BatchNorm2d-319         [-1, 576, 10, 10]           1,152
          Swish6-320         [-1, 576, 10, 10]               0
 DepthwiseConv2d-321         [-1, 576, 10, 10]               0
 GlobalAvgPool2d-322           [-1, 576, 1, 1]               0
          Conv2d-323           [-1, 144, 1, 1]          83,088
          Swish6-324           [-1, 144, 1, 1]               0
   RegularConv2d-325           [-1, 144, 1, 1]               0
          Conv2d-326           [-1, 576, 1, 1]          83,520
         Sigmoid-327           [-1, 576, 1, 1]               0
   RegularConv2d-328           [-1, 576, 1, 1]               0
SqueezeAndExcite-329         [-1, 576, 10, 10]               0
          Conv2d-330         [-1, 160, 10, 10]          92,160
     BatchNorm2d-331         [-1, 160, 10, 10]             320
        Identity-332         [-1, 160, 10, 10]               0
   RegularConv2d-333         [-1, 160, 10, 10]               0
      Bottleneck-334         [-1, 160, 10, 10]               0
          Conv2d-335          [-1, 40, 10, 10]           6,400
     BatchNorm2d-336          [-1, 40, 10, 10]              80
          Swish6-337          [-1, 40, 10, 10]               0
   RegularConv2d-338          [-1, 40, 10, 10]               0
          Conv2d-339         [-1, 120, 10, 10]          43,200
     BatchNorm2d-340         [-1, 120, 10, 10]             240
          Swish6-341         [-1, 120, 10, 10]               0
   RegularConv2d-342         [-1, 120, 10, 10]               0
          Conv2d-343         [-1, 160, 10, 10]          19,200
     BatchNorm2d-344         [-1, 160, 10, 10]             320
        Identity-345         [-1, 160, 10, 10]               0
   RegularConv2d-346         [-1, 160, 10, 10]               0
          Tucker-347         [-1, 160, 10, 10]               0
          Conv2d-348         [-1, 120, 10, 10]          19,200
     BatchNorm2d-349         [-1, 120, 10, 10]             240
          Swish6-350         [-1, 120, 10, 10]               0
   RegularConv2d-351         [-1, 120, 10, 10]               0
          Conv2d-352         [-1, 120, 10, 10]         129,600
     BatchNorm2d-353         [-1, 120, 10, 10]             240
          Swish6-354         [-1, 120, 10, 10]               0
   RegularConv2d-355         [-1, 120, 10, 10]               0
          Conv2d-356         [-1, 160, 10, 10]          19,200
     BatchNorm2d-357         [-1, 160, 10, 10]             320
        Identity-358         [-1, 160, 10, 10]               0
   RegularConv2d-359         [-1, 160, 10, 10]               0
          Tucker-360         [-1, 160, 10, 10]               0
          Conv2d-361          [-1, 40, 10, 10]           6,400
     BatchNorm2d-362          [-1, 40, 10, 10]              80
          Swish6-363          [-1, 40, 10, 10]               0
   RegularConv2d-364          [-1, 40, 10, 10]               0
          Conv2d-365         [-1, 120, 10, 10]          43,200
     BatchNorm2d-366         [-1, 120, 10, 10]             240
          Swish6-367         [-1, 120, 10, 10]               0
   RegularConv2d-368         [-1, 120, 10, 10]               0
          Conv2d-369         [-1, 160, 10, 10]          19,200
     BatchNorm2d-370         [-1, 160, 10, 10]             320
        Identity-371         [-1, 160, 10, 10]               0
   RegularConv2d-372         [-1, 160, 10, 10]               0
          Tucker-373         [-1, 160, 10, 10]               0
```

```
          Conv2d-374         [-1, 1280, 10, 10]         204,800
     BatchNorm2d-375         [-1, 1280, 10, 10]           2,560
          Swish6-376         [-1, 1280, 10, 10]               0
   RegularConv2d-377         [-1, 1280, 10, 10]               0
          Conv2d-378         [-1, 1280, 10, 10]          32,000
     BatchNorm2d-379         [-1, 1280, 10, 10]           2,560
          Swish6-380         [-1, 1280, 10, 10]               0
 DepthwiseConv2d-381         [-1, 1280, 10, 10]               0
 GlobalAvgPool2d-382           [-1, 1280, 1, 1]               0
          Conv2d-383            [-1, 320, 1, 1]         409,920
          Swish6-384            [-1, 320, 1, 1]               0
   RegularConv2d-385            [-1, 320, 1, 1]               0
          Conv2d-386           [-1, 1280, 1, 1]         410,880
        Sigmoid-387           [-1, 1280, 1, 1]               0
   RegularConv2d-388           [-1, 1280, 1, 1]               0
SqueezeAndExcite-389         [-1, 1280, 10, 10]               0
          Conv2d-390          [-1, 288, 10, 10]         368,640
     BatchNorm2d-391          [-1, 288, 10, 10]             576
        Identity-392          [-1, 288, 10, 10]               0
   RegularConv2d-393          [-1, 288, 10, 10]               0
      Bottleneck-394          [-1, 288, 10, 10]               0
 GlobalAvgPool2d-395                  [-1, 288]               0
         Dropout-396                  [-1, 288]               0
          Linear-397                  [-1, 512]         147,968
          Linear-398                  [-1, 133]          68,229
================================================================
Total params: 6,109,565
Trainable params: 6,109,565
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 1.17
Forward/backward pass size (MB): 431.85
Params size (MB): 23.31
Estimated Total Size (MB): 456.33
----------------------------------------------------------------
```

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

Answer:

The proposed model is a MobileDet architecture [4]. The authors use a neural architecture search (NAS) to find the best architecture configuration based on the target hardware the model is going to be deployed. Instead of using only inverted bottlenecks (IBN) for convolution layers, which are the default choice for state-of-the-art architectures due their low latency and high performance, they add two convolution layer families to the search space; Fused (expansion) and Tucker (compression) convolution layers.

The reason behind these additions is that an IBN-only architecture, with less floating-point operations per second (FLOPS) is going to be fast and optimized in CPU but will be slower than full convolutions with orders of magnitude more FLOPS in TPU and GPU, since these hardware architectures are designed to accelerate these types of operations. Since my motivation is to study the MobileDet family, I have implemented the Bottleneck, Tucker and Fused layer classes in the jupyter notebook. As support material, I have used previous integrations using bottleneck

architectures such as MobileNetV1 to V3, EfficientNetV1Bx [1, 2, 3], the MobileDet paper [4] and the Tensorflow research repository [5].

1. A. G. Howard, W. Wang et al., *"MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications"*, https://arxiv.org/pdf/1704.04861.pdf
2. M. Sandler, A. Howard et al., *"MobileNetV2: Inverted Residual and Linear Bottlenecks"*, https://arxiv.org/pdf/1801.04381.pdf
3. A. Howard, M. Sandler et al., *"Searching for MobileNetV3"*, https://arxiv.org/pdf/1905.02244.pdf
4. Y. Xiong, H. Liu et al., *"MobileDets: Searching for Object Detection Architectures for Mobile Accelerators"*, https://arxiv.org/pdf/2004.14525.pdf
5. Tensorflow models research repository, https://github.com/tensorflow/models

(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 [ ]:
```
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)
scheduler_scratch = optim.lr_scheduler.ReduceLROnPlateau(optimizer_scratch, patience=5,
```

(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 [ ]:
```
# the following import is required for training to be robust to truncated images
from PIL import ImageFile
ImageFile.LOAD_TRUNCATED_IMAGES = True

def compute_accuracy(outputs, target):
    size = target.shape[0]
    # convert output probabilities to predicted class
    pred = outputs.max(1, keepdim=True)[1]
    # compare predictions to true label
    correct = pred.eq(target.data.view_as(pred)).squeeze().sum().item()
    return correct / size

def train(n_epochs, loaders, model, optimizer, scheduler, criterion, use_cuda, save_pat
    """returns trained model"""
    # initialize tracker for minimum validation loss
    valid_loss_min = np.Inf
    epoch_improve = 0

    metrics = {
        'epoch': [],
        'lr': [],
```

```
        'loss': [],
        'val_loss': [],
        'val_acc': []
    }

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

        len_train = len(loaders['train'])
        print('Epoch {}/{}'.format(epoch, n_epochs))
        pbar = tqdm(enumerate(loaders['train']), total=len_train, unit='step', ascii="
        ###################
        # train the model #
        ###################
        model.train()
        for batch_idx, (images, labels) in pbar:
            # move to GPU
            if use_cuda:
                images, labels = images.cuda(), labels.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()
            outputs = model(images)
            loss = criterion(outputs, labels)
            train_loss = train_loss + (loss.item() - train_loss) / (batch_idx + 1)
            loss.backward()
            optimizer.step()
            if (batch_idx + 1) == len_train:
                ######################
                # validate the model #
                ######################
                model.eval()
                lr = optimizer.param_groups[0]['lr']
                for val_idx, (images, labels) in enumerate(loaders['valid']):
                    # move to GPU

                    if use_cuda:
                        images, labels = images.cuda(), labels.cuda()

                    with torch.no_grad():
                        outputs = model(images)

                    loss = criterion(outputs, labels)
                    valid_loss = valid_loss + (loss.item() - valid_loss) / (val_idx + 1

                    accuracy = compute_accuracy(outputs, labels)
                    valid_acc = valid_acc + (accuracy - valid_acc) / (val_idx + 1)
                pbar.set_description_str('loss: {:.6f}, val_loss: {:.6f}, val_acc: {:.6
            else:
                pbar.set_description_str('loss: {:.6f}'.format(train_loss))
        scheduler.step(valid_loss)
        metrics['epoch'].append(epoch)
        metrics['lr'].append(lr)
        metrics['loss'].append(train_loss)
        metrics['val_acc'].append(valid_acc)
        metrics['val_loss'].append(valid_loss)
```

```
        ## TODO: save the model if validation loss has decreased
        if valid_loss < valid_loss_min:
            print('Validation loss improved from {:.6f} to {:.6f}. Saving model to {}.'
                valid_loss_min,
                valid_loss,
                save_path
            ))
            epoch_improve = epoch
            valid_loss_min = valid_loss
            torch.save(model.state_dict(), save_path)
        else:
            print('Validation loss hasn\'t improved since epoch {}.'.format(epoch_impro

        if early_stop == (epoch - epoch_improve):
            print('Stopping training early.')
            break

    # return trained model
    return model, metrics

n_epochs = 30
early_stop = 10

# train the model
model_scratch, metrics = train(n_epochs, loaders_scratch, model_scratch, optimizer_scra

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

```
Epoch 1/30
334/334 [==============================================] - 02:05  2.67step/s - loss:
4.815824, val_loss: 4.636381, val_acc: 0.033333, lr: 1.00e-03
Validation loss improved from inf to 4.636381. Saving model to model_scratch.pt.
Epoch 2/30
334/334 [==============================================] - 02:03  2.70step/s - loss:
4.523171, val_loss: 4.630255, val_acc: 0.030952, lr: 1.00e-03
Validation loss improved from 4.636381 to 4.630255. Saving model to model_scratch.pt.
Epoch 3/30
334/334 [==============================================] - 02:07  2.61step/s - loss:
4.334764, val_loss: 4.190006, val_acc: 0.052381, lr: 1.00e-03
Validation loss improved from 4.630255 to 4.190006. Saving model to model_scratch.pt.
Epoch 4/30
334/334 [==============================================] - 02:07  2.63step/s - loss:
4.184640, val_loss: 4.036045, val_acc: 0.047619, lr: 1.00e-03
Validation loss improved from 4.190006 to 4.036045. Saving model to model_scratch.pt.
Epoch 5/30
334/334 [==============================================] - 02:07  2.62step/s - loss:
4.024582, val_loss: 4.016069, val_acc: 0.071429, lr: 1.00e-03
Validation loss improved from 4.036045 to 4.016069. Saving model to model_scratch.pt.
Epoch 6/30
334/334 [==============================================] - 02:08  2.61step/s - loss:
3.891518, val_loss: 3.871686, val_acc: 0.109524, lr: 1.00e-03
Validation loss improved from 4.016069 to 3.871686. Saving model to model_scratch.pt.
Epoch 7/30
334/334 [==============================================] - 02:07  2.62step/s - loss:
3.731757, val_loss: 3.676553, val_acc: 0.139286, lr: 1.00e-03
```

```
Validation loss improved from 3.871686 to 3.676553. Saving model to model_scratch.pt.
Epoch 8/30
334/334 [==============================================] - 02:05  2.65step/s - loss: 3.578462, val_loss: 3.509506, val_acc: 0.140079, lr: 1.00e-03
Validation loss improved from 3.676553 to 3.509506. Saving model to model_scratch.pt.
Epoch 9/30
334/334 [==============================================] - 02:04  2.67step/s - loss: 3.403889, val_loss: 3.357620, val_acc: 0.171429, lr: 1.00e-03
Validation loss improved from 3.509506 to 3.357620. Saving model to model_scratch.pt.
Epoch 10/30
334/334 [==============================================] - 02:04  2.68step/s - loss: 3.190116, val_loss: 3.373256, val_acc: 0.176190, lr: 1.00e-03
Validation loss hasn't improved since epoch 9.
Epoch 11/30
334/334 [==============================================] - 02:04  2.67step/s - loss: 3.023290, val_loss: 3.111387, val_acc: 0.221032, lr: 1.00e-03
Validation loss improved from 3.357620 to 3.111387. Saving model to model_scratch.pt.
Epoch 12/30
334/334 [==============================================] - 02:05  2.67step/s - loss: 2.821896, val_loss: 2.821053, val_acc: 0.278571, lr: 1.00e-03
Validation loss improved from 3.111387 to 2.821053. Saving model to model_scratch.pt.
Epoch 13/30
334/334 [==============================================] - 02:04  2.68step/s - loss: 2.651683, val_loss: 2.619890, val_acc: 0.290079, lr: 1.00e-03
Validation loss improved from 2.821053 to 2.619890. Saving model to model_scratch.pt.
Epoch 14/30
334/334 [==============================================] - 02:04  2.67step/s - loss: 2.488146, val_loss: 2.595762, val_acc: 0.328175, lr: 1.00e-03
Validation loss improved from 2.619890 to 2.595762. Saving model to model_scratch.pt.
Epoch 15/30
334/334 [==============================================] - 02:04  2.69step/s - loss: 2.322708, val_loss: 2.378557, val_acc: 0.367857, lr: 1.00e-03
Validation loss improved from 2.595762 to 2.378557. Saving model to model_scratch.pt.
Epoch 16/30
334/334 [==============================================] - 02:04  2.68step/s - loss: 2.217798, val_loss: 2.260492, val_acc: 0.379365, lr: 1.00e-03
Validation loss improved from 2.378557 to 2.260492. Saving model to model_scratch.pt.
Epoch 17/30
334/334 [==============================================] - 02:04  2.67step/s - loss: 2.064838, val_loss: 2.167741, val_acc: 0.413889, lr: 1.00e-03
Validation loss improved from 2.260492 to 2.167741. Saving model to model_scratch.pt.
Epoch 18/30
334/334 [==============================================] - 02:04  2.67step/s - loss: 1.946257, val_loss: 2.077069, val_acc: 0.417063, lr: 1.00e-03
Validation loss improved from 2.167741 to 2.077069. Saving model to model_scratch.pt.
Epoch 19/30
334/334 [==============================================] - 02:04  2.68step/s - loss: 1.845522, val_loss: 1.906539, val_acc: 0.446825, lr: 1.00e-03
Validation loss improved from 2.077069 to 1.906539. Saving model to model_scratch.pt.
Epoch 20/30
334/334 [==============================================] - 02:04  2.68step/s - loss: 1.753267, val_loss: 1.897296, val_acc: 0.468651, lr: 1.00e-03
Validation loss improved from 1.906539 to 1.897296. Saving model to model_scratch.pt.
Epoch 21/30
334/334 [==============================================] - 02:05  2.67step/s - loss: 1.649322, val_loss: 1.934997, val_acc: 0.470635, lr: 1.00e-03
```

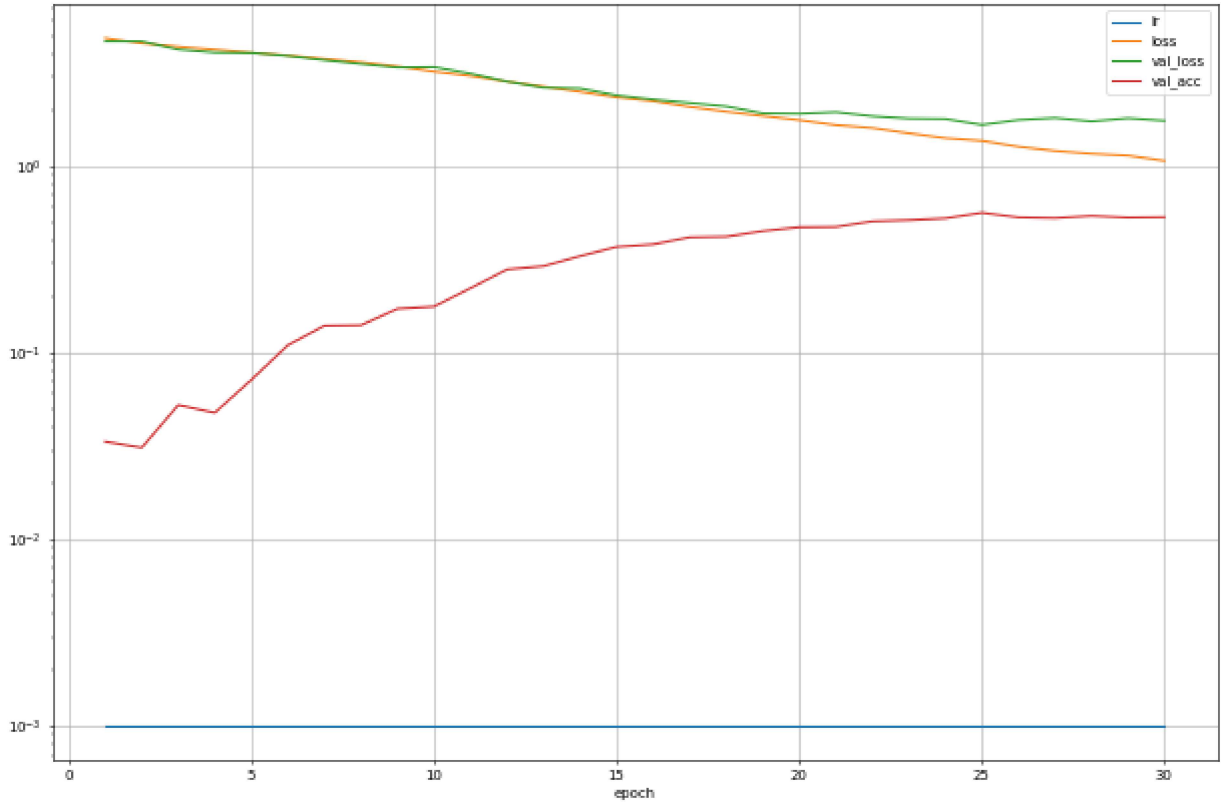
```
Validation loss hasn't improved since epoch 20.
Epoch 22/30
334/334 [==============================================] - 02:04  2.68step/s - loss:
1.591024, val_loss: 1.840181, val_acc: 0.503968, lr: 1.00e-03
Validation loss improved from 1.897296 to 1.840181. Saving model to model_scratch.pt.
Epoch 23/30
334/334 [==============================================] - 02:04  2.68step/s - loss:
1.490278, val_loss: 1.786369, val_acc: 0.511111, lr: 1.00e-03
Validation loss improved from 1.840181 to 1.786369. Saving model to model_scratch.pt.
Epoch 24/30
334/334 [==============================================] - 02:05  2.67step/s - loss:
1.403178, val_loss: 1.778217, val_acc: 0.523016, lr: 1.00e-03
Validation loss improved from 1.786369 to 1.778217. Saving model to model_scratch.pt.
Epoch 25/30
334/334 [==============================================] - 02:04  2.69step/s - loss:
1.358406, val_loss: 1.653972, val_acc: 0.559921, lr: 1.00e-03
Validation loss improved from 1.778217 to 1.653972. Saving model to model_scratch.pt.
Epoch 26/30
334/334 [==============================================] - 02:05  2.67step/s - loss:
1.263310, val_loss: 1.755864, val_acc: 0.528968, lr: 1.00e-03
Validation loss hasn't improved since epoch 25.
Epoch 27/30
334/334 [==============================================] - 02:04  2.67step/s - loss:
1.199310, val_loss: 1.798990, val_acc: 0.524603, lr: 1.00e-03
Validation loss hasn't improved since epoch 25.
Epoch 28/30
334/334 [==============================================] - 02:04  2.68step/s - loss:
1.158079, val_loss: 1.725940, val_acc: 0.538492, lr: 1.00e-03
Validation loss hasn't improved since epoch 25.
Epoch 29/30
334/334 [==============================================] - 02:04  2.68step/s - loss:
1.131802, val_loss: 1.793293, val_acc: 0.527778, lr: 1.00e-03
Validation loss hasn't improved since epoch 25.
Epoch 30/30
334/334 [==============================================] - 02:04  2.68step/s - loss:
1.061462, val_loss: 1.739957, val_acc: 0.530159, lr: 1.00e-03
Validation loss hasn't improved since epoch 25.
```

Out[ ]:
```
<All keys matched successfully>
```

Plot training metrics

Following chart shows how the training loss, validation loss and accuracy improved and what was the learning rate at each step.

In [ ]:
```python
import pandas as pd

df = pd.DataFrame(metrics)
df.set_index(['epoch'], inplace=True)

df.plot()
plt.yscale('log')
plt.legend()
plt.grid()
plt.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 10%.

In [ ]:

```
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 + (loss.data - test_loss) / (batch_idx + 1)
        # 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: 1.975264


Test Accuracy: 48% (409/836)
```

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.

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

class RandomGaussianBlur:
    def __init__(self, kernel_size, sigma=(0.1, 2), p=0.5):
        self.prob = p
        self.blur = transforms.GaussianBlur(kernel_size, sigma)
    def __call__(self, x):
        return self.blur(x) if np.random.random() > self.prob else x


augmentations = transforms.Compose([
    transforms.ColorJitter(contrast=0.5, saturation=0.5),
    transforms.RandomHorizontalFlip(),
    RandomGaussianBlur((3, 3)),
    transforms.RandomAffine((-20, 20), translate=(.15, .15)),
    transforms.RandomResizedCrop(image_shape, scale=(.75, 1)),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

loaders_transfer = loaders_scratch
loaders_transfer['train'].txs = augmentations
```

(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 [ ]:
```
import torchvision.models as models
import torch.nn as nn

## TODO: Specify model architecture
del model_scratch
model_transfer = Net(settings=SETTINGS, hidden_top=512, use_se=True, dropout=0.1, depth

if use_cuda:
    model_transfer = model_transfer.cuda()

# Load model scratch
model_transfer.load_state_dict(torch.load('model_scratch.pt'))
```

Out[ ]:
```
<All keys matched successfully>
```

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:

In the transfer learning step, the following changes were made to the augment transformation:

- Remove hue variations in the color jitter transform.
- Random rotations of -20^0 to 20^0were added to the images.
- Random gaussian blur is applied with 0.5 probability, using a kernel size of 3x3, sigma x and y of 0.1 and 2 respectively.

A dropout rate of 0.1 is used to let the network rely on more connections. The model was trained for 30 epochs with an early stop of 10 epochs if the validation loss doesn't improve. The learning rate of the optimizer was initialized with 0.0001 to let the model improve at a much slower step than before.

A reduce-on-plateau learning rate scheduler to limit the maximum lambda to be used for the Adam optimizer was used in both trainings. This scheduler doesn't do much since the Adam adapts the learning rate for each parameter accordingly.

The results are not as impressive as if the model would have been trained using transfer learning from a previous long running training in the ImageNet dataset but given the fact it was trained from literally scratch, with randomly initialized weights, achieving 62% accuracy after 25 + 3 epochs is quite remarkable.

Further work is required to implement a proper transfer learning approach, training the model in the ImageNet dataset.

(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 [ ]:
```
criterion_transfer = nn.CrossEntropyLoss()
optimizer_transfer = optim.Adam(model_transfer.parameters(), lr=0.0001)
```

```
scheduler_transfer = optim.lr_scheduler.ReduceLROnPlateau(optimizer_transfer, patience=
```

(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 [ ]:

```
# train the model
model_transfer, metrics = train(n_epochs, loaders_transfer, model_transfer, optimizer_t

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

```
Epoch 1/30
334/334 [==============================================] - 01:41  3.29step/s - loss:
0.945261, val_loss: 1.330037, val_acc: 0.613492, lr: 1.00e-05
Validation loss improved from inf to 1.330037. Saving model to model_transfer.pt.
Epoch 2/30
334/334 [==============================================] - 01:42  3.25step/s - loss:
0.948468, val_loss: 1.337477, val_acc: 0.618254, lr: 1.00e-05
Validation loss hasn't improved since epoch 1.
Epoch 3/30
334/334 [==============================================] - 01:43  3.21step/s - loss:
0.958652, val_loss: 1.322802, val_acc: 0.620635, lr: 1.00e-06
Validation loss improved from 1.330037 to 1.322802. Saving model to model_transfer.pt.
Epoch 4/30
334/334 [==============================================] - 01:42  3.24step/s - loss:
0.947222, val_loss: 1.360466, val_acc: 0.609921, lr: 1.00e-06
Validation loss hasn't improved since epoch 3.
Epoch 5/30
334/334 [==============================================] - 01:41  3.29step/s - loss:
0.960123, val_loss: 1.348707, val_acc: 0.621825, lr: 1.00e-06
Validation loss hasn't improved since epoch 3.
Epoch 6/30
334/334 [==============================================] - 01:39  3.34step/s - loss:
0.947728, val_loss: 1.323250, val_acc: 0.612302, lr: 1.00e-06
Validation loss hasn't improved since epoch 3.
Epoch 7/30
334/334 [==============================================] - 01:40  3.34step/s - loss:
0.931063, val_loss: 1.347735, val_acc: 0.606349, lr: 1.00e-06
Validation loss hasn't improved since epoch 3.
Epoch 8/30
334/334 [==============================================] - 01:40  3.34step/s - loss:
0.975639, val_loss: 1.337202, val_acc: 0.618254, lr: 1.00e-06
Validation loss hasn't improved since epoch 3.
Epoch 9/30
334/334 [==============================================] - 01:39  3.34step/s - loss:
0.921839, val_loss: 1.359315, val_acc: 0.613492, lr: 1.00e-06
Validation loss hasn't improved since epoch 3.
Epoch 10/30
334/334 [==============================================] - 01:40  3.33step/s - loss:
0.958462, val_loss: 1.341368, val_acc: 0.607540, lr: 1.00e-07
Validation loss hasn't improved since epoch 3.
Epoch 11/30
```

```
334/334 [==============================================] - 01:39  3.34step/s - loss:
0.972801, val_loss: 1.338655, val_acc: 0.618254, lr: 1.00e-07
Validation loss hasn't improved since epoch 3.
Epoch 12/30
334/334 [==============================================] - 01:39  3.35step/s - loss:
0.922257, val_loss: 1.336849, val_acc: 0.618254, lr: 1.00e-07
Validation loss hasn't improved since epoch 3.
Epoch 13/30
334/334 [==============================================] - 01:39  3.34step/s - loss:
0.937691, val_loss: 1.349450, val_acc: 0.620635, lr: 1.00e-07
Validation loss hasn't improved since epoch 3.
Stopping training early.
```

Out[ ]:
```
<All keys matched successfully>
```

(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 [ ]:
```
test(loaders_transfer, model_transfer, criterion_transfer, use_cuda)
```

```
Test Loss: 1.338512


Test Accuracy: 62% (523/836)
```

(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 [ ]:
```
### 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'].classes

def predict_breed_transfer(model, img_path):
    pred = Predict(model, img_path)
    return class_names[pred]
```

Step 5: Write your Algorithm

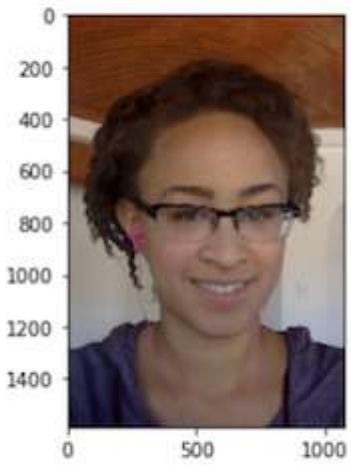
Write an algorithm that accepts a file path to an image and first determines whether the image contains a human, dog, or neither. Then,

- 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

In [ ]:

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

def run_app(model, img_path):
    ## handle cases for a human face, dog, and neither
    is_face = face_detector(img_path)
    is_dog = dog_detector(MobileNetV3_small, img_path)

    predicted_dogbreed = predict_breed_transfer(model, img_path)

    dog_pics = [ pic for pic in dog_files if pic.find(predicted_dogbreed.replace(' ', '
    dog = Image.open(dog_pics[np.random.randint(0, len(dog_pics))])
    img = Image.open(img_path)

    if is_face and not is_dog:
        print('This is a human :)... and looks like a {}'.format(predicted_dogbreed))
        print('Do you see the resemblance?')
    elif is_face and is_dog:
        print('Ahm... I\'m confused, this might be a dog with its owner or a dog that a
        print('Is it a {}?'.format(predicted_dogbreed))
    elif not is_dog:
        print('Doesn\'t look like a human. Is it a {}?'.format(predicted_dogbreed))
    else:
        print('This is a {}, isn\'t it?'.format(predicted_dogbreed))

    plt.subplot(121)
    plt.imshow(img)
```

```
    plt.subplot(122)
    plt.imshow(dog)
    plt.show()
```

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.

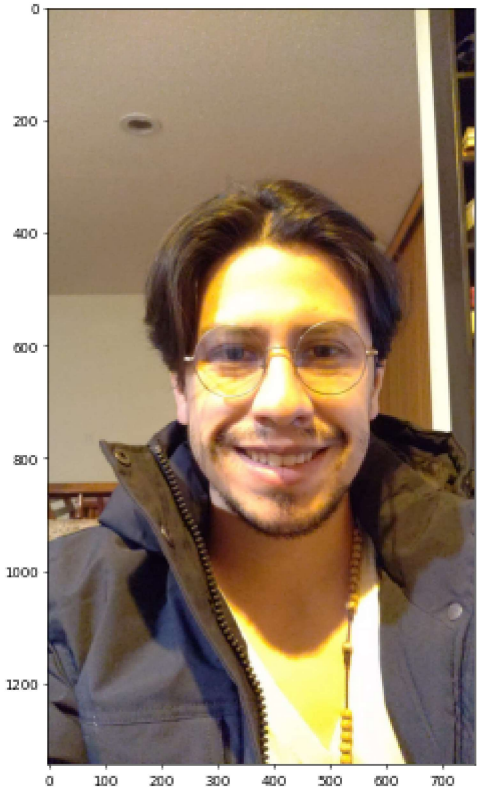
Answer: (Three possible points for improvement)

1. Further work is required to implement a proper transfer learning approach, training the model in the ImageNet dataset.
2. Further experimentation is needed to implement a good data augmentation strategy.
3. The NAS approach offers a good starting point to design hardware and possible problem aware CNN architectures, where we could have a model specially designed to maximize dog breed classification accuracy.
4. I tried to export this model using ONNX to develop a PWA that could do funny things, but the ReLU6 operation was not supported. This might be a good starting point to deploy the model in a web application. With this many parameters a TFLite model would weight as much as 7MB.

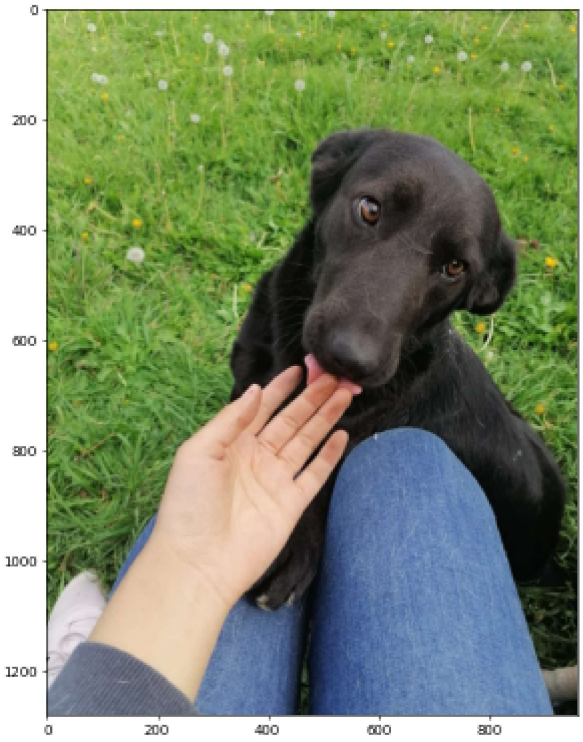
In [ ]:
```
## TODO: Execute your algorithm from Step 6 on
## at least 6 images on your computer.
## Feel free to use as many code cells as needed.

for file in glob('testsImgs/*'):
    run_app(model_transfer, file)
```

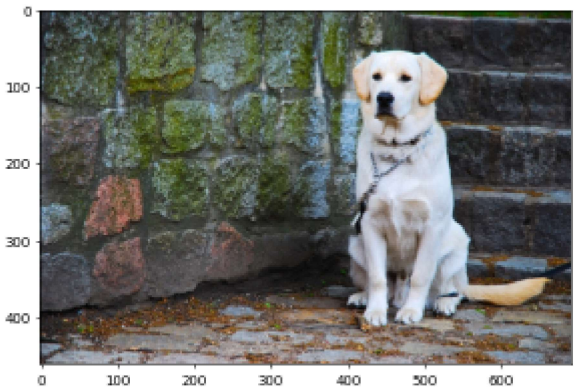
```
This is a Chinese crested, isn't it?
```

This is a human :)... and looks like a German shorthaired pointer
Do you see the resemblance?

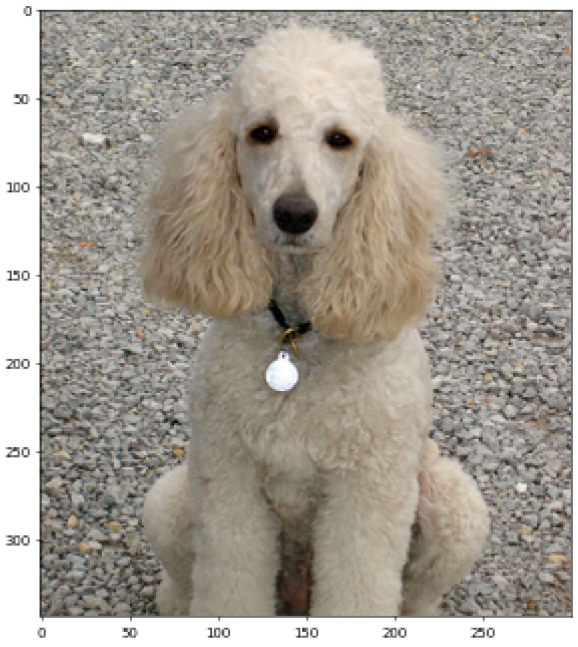


This is a Labrador retriever, isn't it?

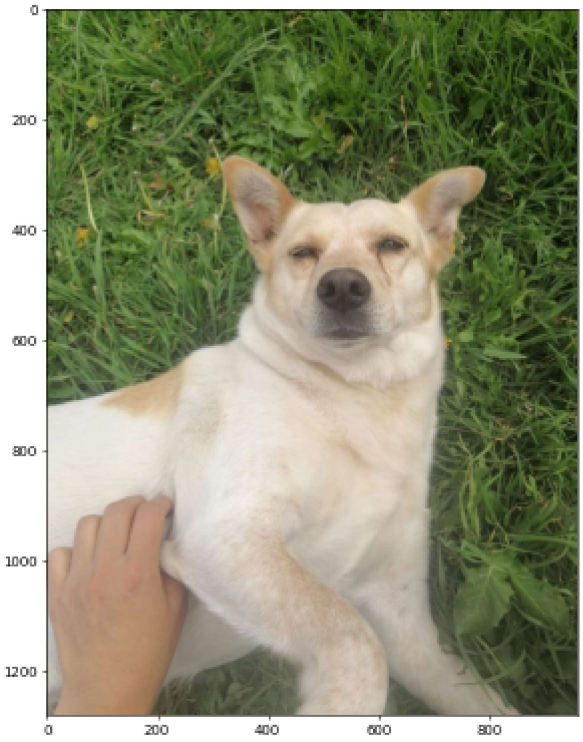


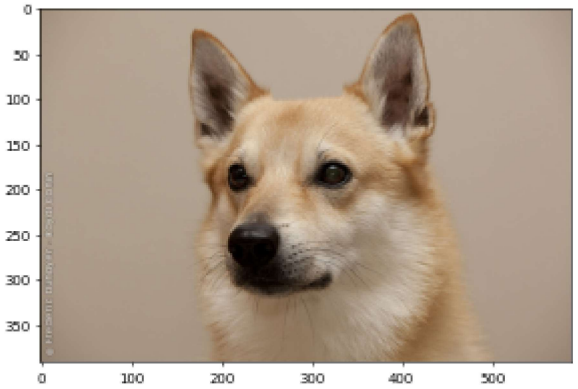


This is a human :)... and looks like a Poodle
Do you see the resemblance?

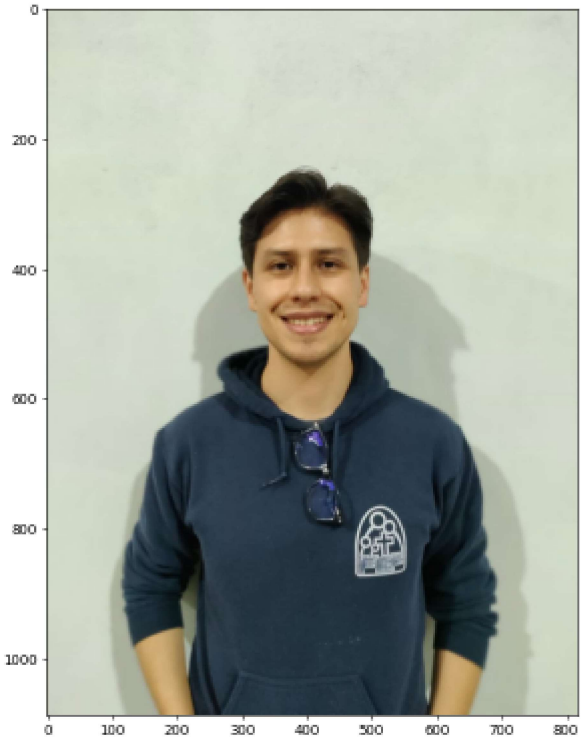


This is a Norwegian buhund, isn't it?

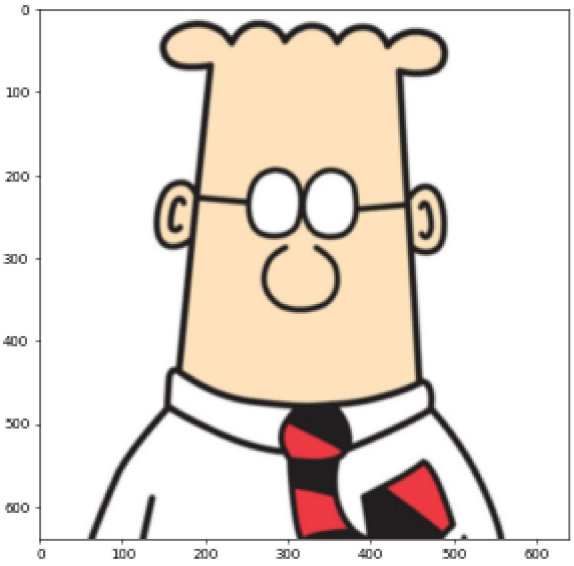




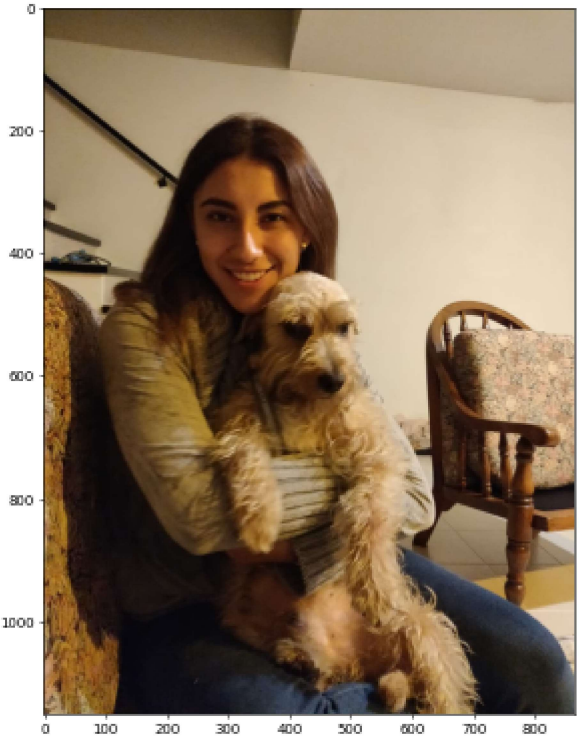
This is a human :)... and looks like a German pinscher
Do you see the resemblance?



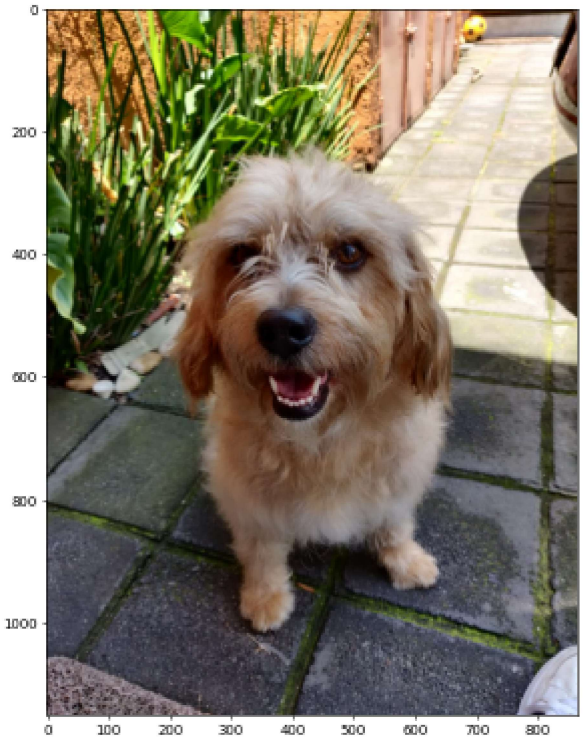
Doesn't look like a human. Is it a Great dane?



This is a human :)... and looks like a Field spaniel
Do you see the resemblance?



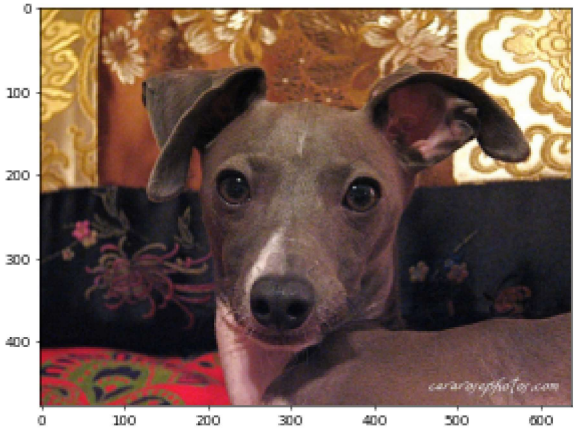
This is a Dandie dinmont terrier, isn't it?



This is a human :)... and looks like a Chesapeake bay retriever
Do you see the resemblance?

This is a human :)... and looks like a Italian greyhound
Do you see the resemblance?


0
200
400
600
800
1000
0
100
200
300
400
500
600
700
800


0
100
200
300
400
0
100
200
300
400
500
600

In [ ]: