UDACITY

Machine Learning Engineer Nanodegree

Capstone Project:

Dog Breed Classification

Shivam Tawari

24th May 2020

Definition

Domain Background:

It is very reasonable to see dogs on the streets every day. Seeing them doesn’t mean knowing them. Out of so many dog breeds, I can identify the German Shepherd and the Pug only. There are so many breeds looking similar, and we cannot tell the differences. There are around 350 breeds of dog in the whole world. Identifying the breeds of the dogs is not a task humans can easily do, if we don't know much about dogs. There are thousands of dog breeds in the world. Some of these dog breeds are too close to differentiate from their images.
Dog Breed Classification is a prevalent problem in Machine Learning. We can use supervised machine learning to solve this problem with the help of image classification using Convolutional Neural Network (CNN).

Fig. Belgian Malinois Dog vs. German Shepherd Dog

Fig. Whippet vs. Italian Greyhound

Problem Statement:

Identification of dog breeds is not an easy thing. Our main aim is to create a machine learning model which will do the following tasks:

- The model will identify which breed is present in the image input of a dog supplied by the user.
- The model will also identify whether the input of a human image resembles any dog breed.

Metrics:

The metrics to judge a Convolutional Neural Network model performs are to look at its accuracy, Cross-entropy loss, or log loss. We need to see how the model functions and check it's validation loss values and prediction accuracy against the test dataset.

$$\text{accuracy} = \frac{\text{correct predictions}}{\text{all predictions}}$$

Accuracy is defined as the percentage of correct predictions for the test data. It can be calculated easily by dividing the number of accurate predictions by the number of total predictions.

$$logloss_{(\mathrm{N}=1)} = y\log(p) + (1-y)\log(1-p)$$

Whereas Log Loss quantifies the accuracy of a classifier by penalizing false classifications. Minimizing the Log Loss is equivalent to maximizing the efficiency of the classifier, but there is a subtle twist which we'll get to in a moment. Also, during model training, we compare the test data prediction with the validation dataset and calculate Multi-class log loss to find the best performing model. We have more metrics such as Precision and Recall, but we won't be using it for our model.

Analysis

Data Exploration

An extensive set of dogs and human pictures were provided for training and testing purposes in the dataset. The dataset for the project has already been provided by the Udacity Machine Learning Engineer Nanodegree Program.

```
In [8]: # load filenames for human and dog images
        human_files = np.array(glob("./data/lfw/*/*"))
        dog_files = np.array(glob("./data/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.
```

Fig. Data Structure

Humans Dataset: There are 13233 total human images which are sorted by their names.

Dogs Dataset: The dog image dataset has 8351 total images that have 133 breeds of dog. The dataset is sorted in the following manner:

Train: The train folder has 6680 images of 133 breeds of dog with 25-70 images per breed.
Test: The test folder has 836 images of 133 breeds of dog with 5-10 images per breed.
Valid: The valid folder has 835 images of 133 breeds of dog with 5-10 images per breed.

The number of images provided is imbalanced per breed. And each image does not have a clear background, and some have one or more humans or dogs in an image.

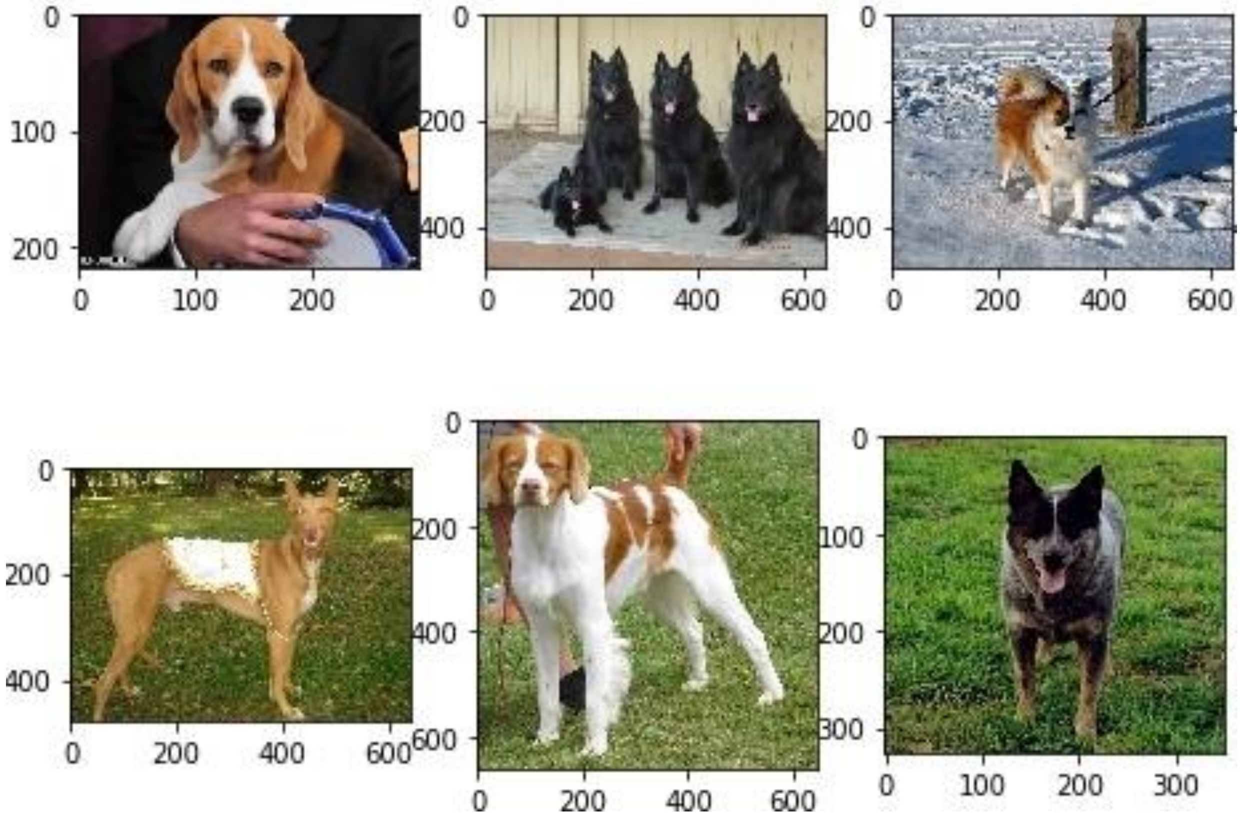


Fig. Some Sample Images of Dogs from the Dataset

The human dataset is similarly structured, with most pictures featuring a single human portrait in the middle of the frame. Some images are showing multiple humans in a frame, but one human is emphasized.

Fig. Some Sample Images of Humans from the Dataset

Exploratory Visualization

Most of the data contain images with one primary human or dog in the image. But few photos also include one or more humans or dogs in the picture. This is a challenge in the project.

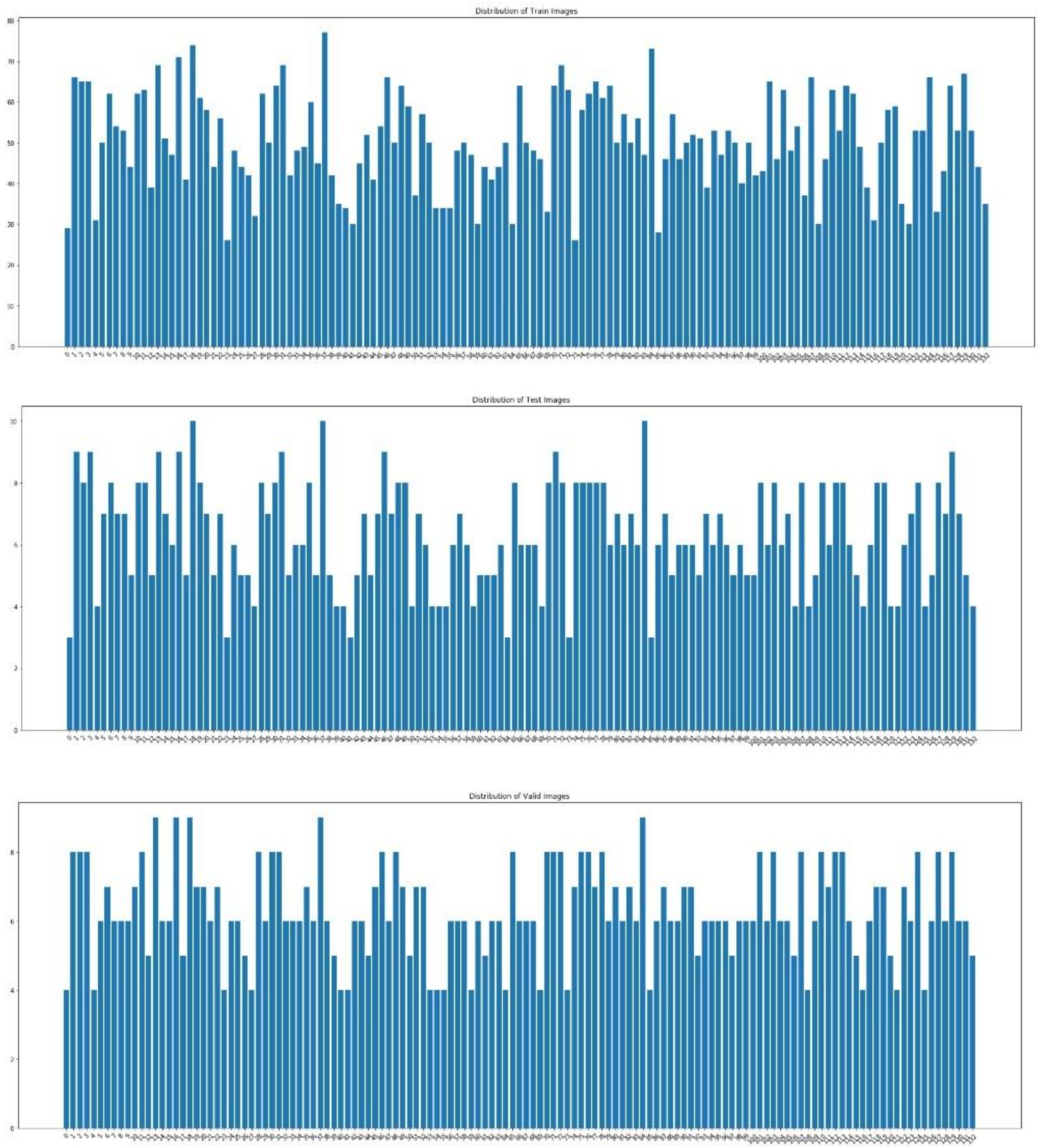


Fig. Distribution of Train, Test and Valid Images

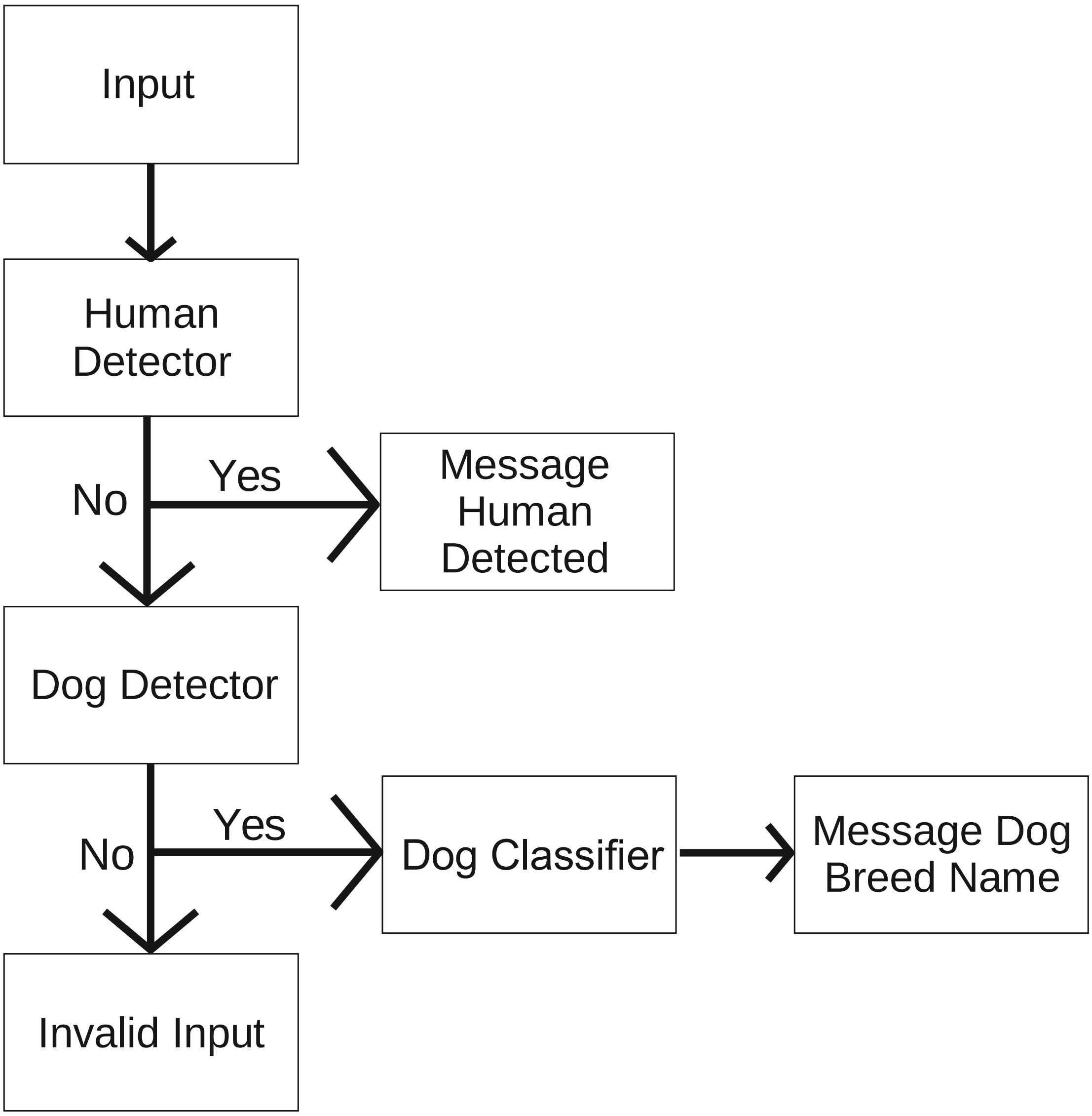
Algorithms and Techniques

For creating this Machine Learning Model of Dog Breeds Classification, we will use Convolutional Neural Network (CNN). A Convolutional Neural Network (CNN) is a type of Artificial Neural Network used in image recognition and processing that is specifically designed to process pixel data. CNN is designed to automatically and adaptively learn spatial hierarchies of features through backpropagation by using multiple building blocks, such as convolution layers, pooling layers, and fully connected layers.

- *A convolutional neural network is a class of deep learning methods that have become dominant in various computer vision tasks.*
- *Convolutional neural network is composed of multiple building blocks, such as convolution layers, pooling layers, and fully connected layers. It is designed to automatically and adaptively learn spatial hierarchies of features through a backpropagation algorithm.*
- *CNN works exceptionally well with image classification applications because it follows a hierarchical model that resembles the human brain. The established model features fully connected layers where all the neurons are connected with specified outputs. Images are composed of clouds, edges, colors, etc. These features are easily extractable by convolutional filters in the CNN models.*

We use OpenCV's Haar Cascades for face detection. Then a pre-trained model will help to detect dogs. Finally, after identification of the input image, we will use a trained model using Convolutional Neural Network (CNN) to identify the breed of the dog out of 133 breeds based on the input.

Initially, the input (image) will go through a human face detector. If a face is detected, it will show the message with the name of the dog breed resembling. If a dog is detected in an image, the dog detector will display the name of the breed to which the dog belongs. The flowchart is given below:



Dog Classifier Flowchart

Benchmark Model:

★ The Convolutional Neural Network model prepared from scratch should have an accuracy of at least 10%.
★ The Convolutional Neural Network model created using transfer learning must have an accuracy of 60% and above.

Methodology

Data Preprocessing

We have initially have done normalization to the whole dataset. Then we have resized all the images to 256x256 pixels. Three directories are created Train, Test, and Valid.

```
In [27]: import os
         from torchvision import datasets

         # Build full paths of train, valid and test directories
         data_dir = 'data/dogImages/'
         train_dir = os.path.join(data_dir, 'train/')
         valid_dir = os.path.join(data_dir, 'valid/')
         test_dir = os.path.join(data_dir, 'test/')
```

Fig. Train, Test, and Validation Directories

And then cropped them to 224x224 pixels and centered them all the three train, test, and valid datasets. Only the train dataset is randomly horizontally flipped. Random flip helps in data variation and thus creates a robust dataset. And then, finally, all three dataset directories train, test, and valid are converted into tensors before processing them into the model.

```
In [28]: # Define standard normalization for all transormations
         normalization = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                              std=[0.229, 0.224, 0.225])

         train_transform = transforms.Compose([transforms.Resize(256),
                                               transforms.RandomResizedCrop(224),
                                               transforms.RandomHorizontalFlip(),
                                               transforms.RandomRotation(10),
                                               transforms.ToTensor(),
                                               normalization])


         valid_transform = transforms.Compose([transforms.Resize(256),
                                               transforms.CenterCrop(224),
                                               transforms.ToTensor(),
                                               normalization])

         test_transform  = transforms.Compose([transforms.Resize(256),
                                               transforms.CenterCrop(224),
                                               transforms.ToTensor(),
                                               normalization])
```

Fig. Data Preprocessing

Implementation

Human Detector: The input image first passes through the human face detector. This Human Face Detector is an OpenCV's pre-trained model named Haar-cascade classifier.

In [16]:

```python
import cv2
import matplotlib.pyplot as plt
%matplotlib inline

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

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

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

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

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

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

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

Fig. Implementation of Haar feature-based cascade classifiers

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

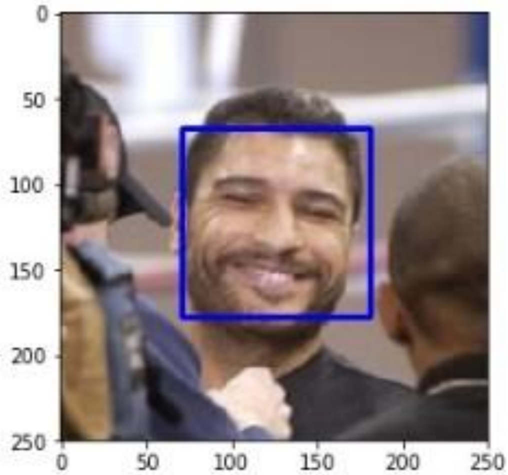


Fig. Human Face Sample Result

Dog Detector: If the haar-cascade face detector doesn't detect any human face, the image is passed to the dog breed detector. This is constructed with the help of a pre-trained model called VGG-16. VGG-16 is a convolutional neural network architecture, and its name VGG-16 comes from the fact that it has 16 layers. Its layers consist of Convolutional layers, Max Pooling layers, Activation layers, Fully connected layers. VGG-16 network is trained on the ImageNet dataset, which has over 14 million images and 1000 classes.

```
In [21]: from PIL import Image
         import torchvision.transforms as transforms

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

             Args:
                 img_path: path to an image

             Returns:
                 Index corresponding to VGG-16 model's prediction
             '''

             ## TODO: Complete the function.
             ## Load and pre-process an image from the given img_path
             ## Return the *index* of the predicted class for that image
             image = Image.open(img_path).convert('RGB')

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

             transformations = transforms.Compose([transforms.Resize(size=(224, 224)),
                                                   transforms.ToTensor(),
                                                   normalize])

             transformed_image = transformations(image)[:3,:,:].unsqueeze(0)

             if use_cuda:
                 transformed_image = transformed_image.cuda()

             output = VGG16(transformed_image)

             return torch.max(output,1)[1].item() # predicted class index
```

Dog Detector using VGG16 model

```
In [22]: # test functions
         dog_img_example = Image.open(dog_files_short[11])
         plt.imshow(dog_img_example)

Out[22]: <matplotlib.image.AxesImage at 0x7f81fb338780>
```

Sample Prediction

Dog Breed Classifier: When the dog detector detects a dog, it is passed to a dog breed classifier. We have created a Convolutional Neural Network from scratch for predicting the dog breed. The CNN model uses the following things:

- First convolution layer uses 32 filters, max pooling, and stride reduced the image size.
- The second convolution layer uses 64 filters, max pooling, and stride to reduce the image size.
- Third convolution layer uses 128 filters, and max pooling reduced the image size to 7x7
- Two successive layers are assigned, with 133 as the output size.
- Dropout is set to be 0.3 to avoid overfitting.

But the scratch CNN model is straightforward and cannot achieve the accuracy we required.

Refinement

The CNN model, which we created from scratch has an accuracy of just 4%. It doesn't even meet our benchmark. Thus we need to improve our model with the help of transfer learning. Transfer learning is a machine learning technique where a model trained on one task is repurposed on a second related task. In this project, we are using ResNet-101. It is a convolutional neural network that is 101 layers deep. We only need to add a fully connected layer to produce 133-dimensional output. The model performed extremely well when compared to CNN from scratch.

Results

Model Evaluation and Validation

As mentioned earlier, the scratch CNN model does not perform well. After 10 epochs of training, the validation loss is 4.526055. This result shows us that the CNN model we created has too simple architecture.

```
In [34]: # train the model
         model_scratch = train(10, loaders_scratch, model_scratch, optimizer_scratch,
                               criterion_scratch, use_cuda, 'model_scratch.pt')

Epoch: 1        Training Loss: 4.884570         Validation Loss: 4.872018
Validation loss decreased (inf --> 4.872018).  Saving model ...
Epoch: 2        Training Loss: 4.859816         Validation Loss: 4.849079
Validation loss decreased (4.872018 --> 4.849079).  Saving model ...
Epoch: 3        Training Loss: 4.820254         Validation Loss: 4.791597
Validation loss decreased (4.849079 --> 4.791597).  Saving model ...
Epoch: 4        Training Loss: 4.770636         Validation Loss: 4.750432
Validation loss decreased (4.791597 --> 4.750432).  Saving model ...
Epoch: 5        Training Loss: 4.748075         Validation Loss: 4.726555
Validation loss decreased (4.750432 --> 4.726555).  Saving model ...
Epoch: 6        Training Loss: 4.712607         Validation Loss: 4.701014
Validation loss decreased (4.726555 --> 4.701014).  Saving model ...
Epoch: 7        Training Loss: 4.676020         Validation Loss: 4.638260
Validation loss decreased (4.701014 --> 4.638260).  Saving model ...
Epoch: 8        Training Loss: 4.606761         Validation Loss: 4.565671
Validation loss decreased (4.638260 --> 4.565671).  Saving model ...
Epoch: 9        Training Loss: 4.574089         Validation Loss: 4.566793
Epoch: 10       Training Loss: 4.564557         Validation Loss: 4.526055
Validation loss decreased (4.565671 --> 4.526055).  Saving model ...
```

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

Test Loss: 4.529026


Test Accuracy:  4% (280/6680)
```

Fig. Accuracy of the CNN model from scratch

However, with the help of the Transfer Learning technique, we implemented ResNet-101. We trained that model for 20 epochs, and results were much better than the model we created from scratch. The test loss was 1.719057, and the accuracy was equal to 70% (4740/6680).

```
In [42]: test(loaders_transfer, model_transfer, criterion_transfer, use_cuda)

         Test Loss: 1.726822


         Test Accuracy: 70% (4736/6680)
```

Fig. Accuracy of the transferred ResNet-101 model

```
In [40]: # train the model
model_transfer = train(20, loaders_transfer, model_transfer, optimizer_transfe
t')

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

```
Epoch: 1        Training Loss: 6.416148         Validation Loss: 5.850319
Validation loss decreased (inf --> 5.850319).  Saving model ...
Epoch: 2        Training Loss: 5.505236         Validation Loss: 5.091931
Validation loss decreased (5.850319 --> 5.091931).  Saving model ...
Epoch: 3        Training Loss: 4.905421         Validation Loss: 4.582201
Validation loss decreased (5.091931 --> 4.582201).  Saving model ...
Epoch: 4        Training Loss: 4.483241         Validation Loss: 4.205105
Validation loss decreased (4.582201 --> 4.205105).  Saving model ...
Epoch: 5        Training Loss: 4.153215         Validation Loss: 3.886796
Validation loss decreased (4.205105 --> 3.886796).  Saving model ...
Epoch: 6        Training Loss: 3.879205         Validation Loss: 3.570375
Validation loss decreased (3.886796 --> 3.570375).  Saving model ...
Epoch: 7        Training Loss: 3.644866         Validation Loss: 3.319936
Validation loss decreased (3.570375 --> 3.319936).  Saving model ...
Epoch: 8        Training Loss: 3.422944         Validation Loss: 3.121242
Validation loss decreased (3.319936 --> 3.121242).  Saving model ...
Epoch: 9        Training Loss: 3.228610         Validation Loss: 2.977206
Validation loss decreased (3.121242 --> 2.977206).  Saving model ...
Epoch: 10       Training Loss: 3.045317         Validation Loss: 2.759387
Validation loss decreased (2.977206 --> 2.759387).  Saving model ...
Epoch: 11       Training Loss: 2.896891         Validation Loss: 2.593393
Validation loss decreased (2.759387 --> 2.593393).  Saving model ...
Epoch: 12       Training Loss: 2.748847         Validation Loss: 2.440891
Validation loss decreased (2.593393 --> 2.440891).  Saving model ...
Epoch: 13       Training Loss: 2.625856         Validation Loss: 2.341729
Validation loss decreased (2.440891 --> 2.341729).  Saving model ...
Epoch: 14       Training Loss: 2.501039         Validation Loss: 2.210972
Validation loss decreased (2.341729 --> 2.210972).  Saving model ...
Epoch: 15       Training Loss: 2.406114         Validation Loss: 2.137880
Validation loss decreased (2.210972 --> 2.137880).  Saving model ...
Epoch: 16       Training Loss: 2.298508         Validation Loss: 2.009991
Validation loss decreased (2.137880 --> 2.009991).  Saving model ...
Epoch: 17       Training Loss: 2.213571         Validation Loss: 1.959886
Validation loss decreased (2.009991 --> 1.959886).  Saving model ...
Epoch: 18       Training Loss: 2.137583         Validation Loss: 1.893592
Validation loss decreased (1.959886 --> 1.893592).  Saving model ...
Epoch: 19       Training Loss: 2.061157         Validation Loss: 1.801663
Validation loss decreased (1.893592 --> 1.801663).  Saving model ...
Epoch: 20       Training Loss: 1.996163         Validation Loss: 1.732064
Validation loss decreased (1.801663 --> 1.732064).  Saving model ...

Out[40]: <All keys matched successfully>
```

Fig. Training Log of ResNet-101 Model

```
In [43]: ### TODO: Write your algorithm.
### Feel free to use as many code cells as needed.
def load_image(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

    if face_detector(img_path):
        print ("Hello Human!")
        predicted_breed = predict_breed_transfer(img_path)
        print("Predicted breed: ",predicted_breed)
        load_image(img_path)

    elif dog_detector(img_path):
        print ("Hello Dog!")
        predicted_breed = predict_breed_transfer(img_path)
        print("Predicted breed: ",predicted_breed)
        load_image(img_path)

    else:
        print ("Invalid image")
```

Fig. Final Algorithm

Fig. Sample Outputs

Justification

The model got the best accuracy of 70% and a test loss of ~1.72, which is better than the CNN model, which we created from scratch, which only had an accuracy of 4%. We improved our project's efficiency from 4% to 70%.

Improvement

The model which we created is working well. Some improvement areas which I think can be worked on are:

- Increasing the dataset could improve accuracy.
- Tuning the parameters could improve the results.
- We can use different pre-trained models for improving the result.

References

1. Original repository for Project
https://github.com/udacity/deep-learning-v2-pytorch/blob/master/project-dog-classification
2. What’s considered a good Log Loss in Machine Learning?
https://medium.com/@fzammito/whats-considered-a-good-log-loss-in-machine-learning-a529d400632d
3. VGG-16 | CNN model
https://www.geeksforgeeks.org/vgg-16-cnn-model/
4. Evaluating a machine learning model
https://www.jeremyjordan.me/evaluating-a-machine-learning-model/
5. Making Sense of Logarithmic Loss
https://datawookie.netlify.app/blog/2015/12/making-sense-of-logarithmic-loss/
6. ResNet-101 | Kaggle
https://www.kaggle.com/pytorch/resnet101