Machine Learning Engineer Nanodegree **CHANG YONG LIK**
CAPSTONE PROJECT PROPOSAL **2021 May 27**

CNN Project: Dog Breed Classifier

Project Overview

The purpose of this project is to classify different dog breeds. There are a lot of dog breeds in the world, so distinguishing each dog breed is a challenging task. In this project, we are going to design a system and algorithm to solve the classification problem of dog breed. We would need to classify 133 dog breeds using convolutional neural networks (CNN).

Problem Statement

The aim of this project is to build a model that can accept an image and then distinguish whether that image is a dog or a human. If a user submits a dog photo, the model has to correctly predict its breed; and if a user submits a human photo, the model has to correctly predict the most resembling dog breed. So, we can divide the task up into two parts:

Dog face detector: Given a dog image, model will predict its breed

Dog breed detector: Given an image, if it's a dog, the model will predict its breed or if it's a human, the model will predict the closest resemblance of the breed.

Human face detector: Given a Human face image, model will predict most resembling dog breed

Datasets and Inputs

Inputs
Images of either dogs or humans

Dataset
The dataset is provided by Udacity and comprises two parts - dogs and humans.

Human images

Human images are distributed in 5749 folders named after human names. There are a total of 13233 human face images. All the images are not distributed evenly.

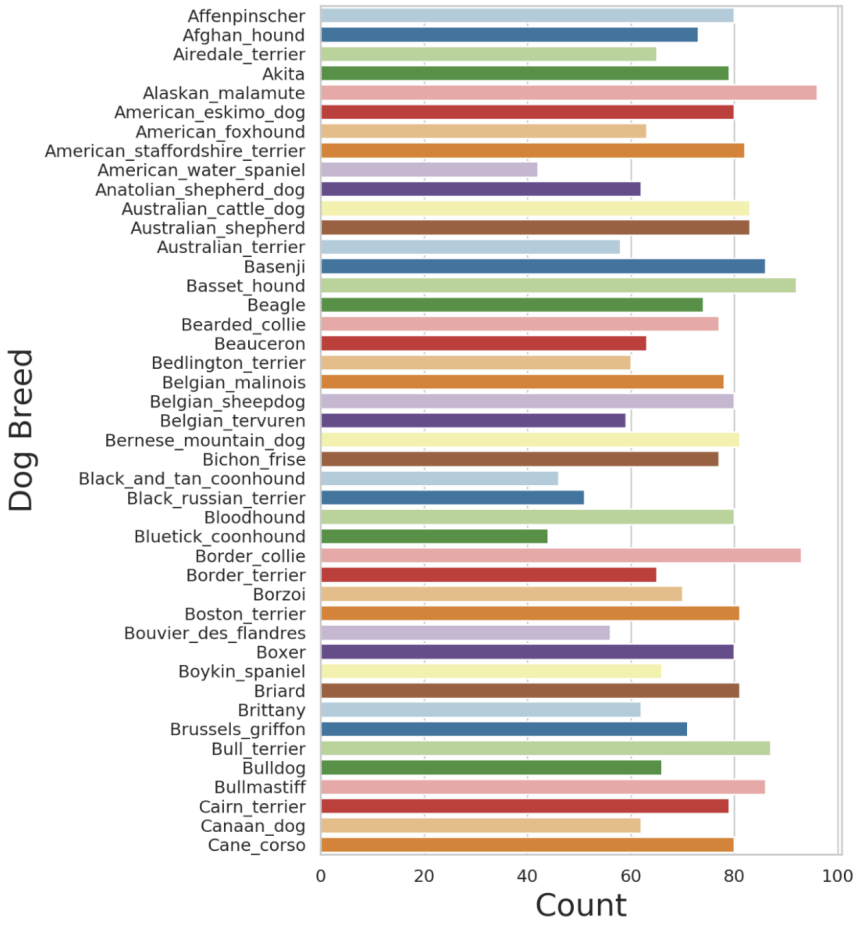
Dog images

Dog images are separated into 3 folders, namely train, test and valid. The folders are then further separated into 133 folders each representing the total number of dog breeds.

After doing EDA, I have compiled the information about the data in the following.

- Total number of human face images: 13233
- Total number of human face folders: 5749
- Total number of images in train: 6680
- Total number of images in test: 836
- Total number of images in valid: 835

Here is a distribution of dog breeds downloaded from Udacity.



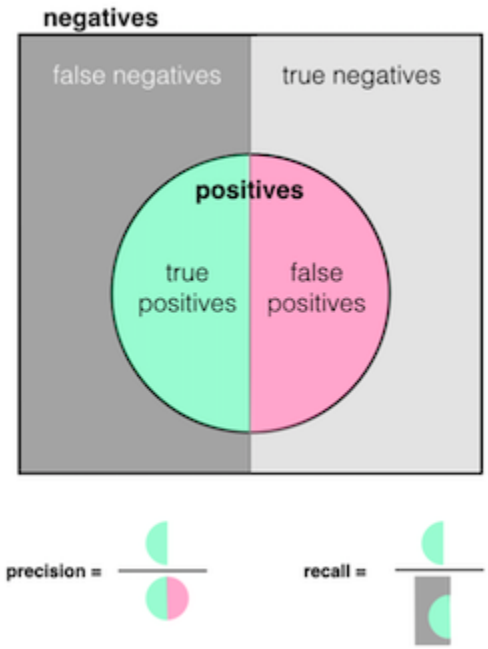
Dog Breed Distribution

Evaluation Metrics

This is a multiclass classification problem of dog vs human. The dataset is imbalanced, so we would need a good metric other than accuracy to gauge the performance of the model. Precision and recall are great choices for model evaluation. Precision and recall are just different metrics for measuring the performance of a trained model. Below are the definitions of precision and recall.

- Precision is defined as the number of true positives over all positives, and will be higher when the amount of false positives is low.
- Recall is defined as the number of true positives over true positives plus false negatives and will be higher when the number of false negatives is low.

Both take into account true positives and will be higher for high accuracy as well. Precision and recall can be visualized in the following diagram.



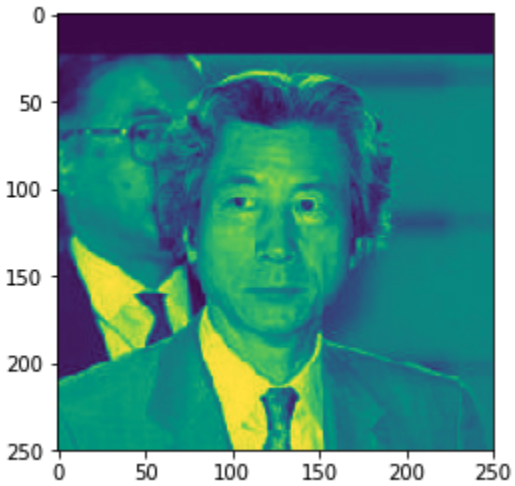
- Precision: True Positives/ (True Positives + False Positives)
- Recall: True Positives/ (True Positives + False Negatives)

In most cases, it may be worthwhile to optimize for a higher recall or precision, which gives you a more granular look at false positives and negatives.

Human Face Detection

OpenCV's Haar Cascade Classifier is used to detect human faces. This classifier is trained on many images with positive labels (with face) and negative labels (without face). The detectMultiScale returns a list of 4 bounding box coordinates for all detected faces in the same image. It is a standard practice to convert an RGB image into a grayscale image for all face detection algorithms. We have tested on two different classifiers namely, haarcascade_frontalface_alt.xml and haarcascade_frontalface_alt2.xml.

Example of an RGB images



Face Detector function

```
# 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
```

Model Performance:

Haar Cascade Classifier has an overall good performance

haarcascade_frontalface_alt.xml

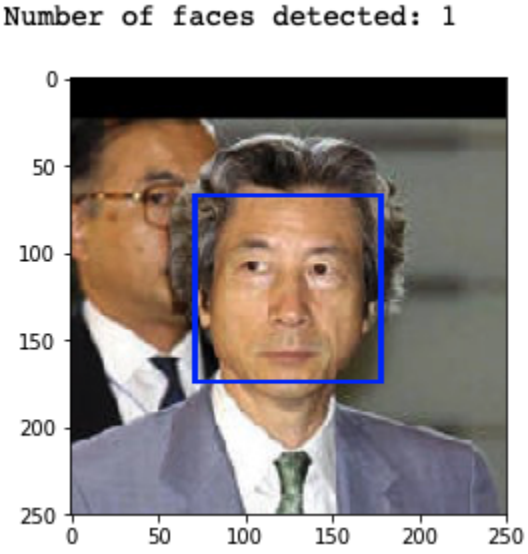
- Percentage of human faces detected in human images data: 98%
- Percentage of human faces detected in dog images data: 17%

haarcascade_frontalface_alt2.xml

- Percentage of human faces detected in human images data: 100%
- Percentage of human faces detected in dog images data: 21%

You can see that we have a better performance with the haarcascade_frontalface_alt2.xml. So we have an accuracy of 100 percent. We have an increased percentage of recognized humans in the dog_files_short. But this is not a problem as we will recognize dogs with a different detector.

Sample result given by Haar Cascade Classifier:



Dog Detection

Pretrained VGG16 model is used here to detect dog images. VGG16 is pretrained on 10M images of ImageNet data for 1000 classes. This model has been downloaded from torchvision. Before we start our training, we have to first load and transform the image in the required format (e.g. image size, RGB conversion and data normalization). The load_transform_image function transforms all images in a folder to the necessary format for training and prediction in a pipeline. The dog_detector function returns True if a dog is detected, else False is returned.

```
import torch
from PIL import Image
import torchvision.transforms as transforms
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()
def load_transform_image(img_path):
    '''
    Used load & transform image for prediction on single image
    '''
img = Image.open(img_path).convert('RGB')
    normalize = transforms.Normalize(
                        mean=[0.485, 0.456, 0.406],
                        std=[0.229, 0.224, 0.225])
    img_transform = transforms.Compose([
                        transforms.Resize(size=(224, 224)),
                        transforms.ToTensor(),
                        normalize])
    img = img_transform(img)[:3,:,:].unsqueeze(0)
    return img
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
    '''
    image = load_transform_image(img_path)
    if use_cuda:
        image = image.cuda()
    output = VGG16(image)
    return torch.max(output,1)[1].item()
def dog_detector(img_path):
    prediction = VGG16_predict(img_path)
    return (prediction>=151 and prediction<=268)
```

Model Performance:

The detection is performed only for the first 100 images in both datasets to save time.

- Percentage of dogs detected in human images data: 0%
- Percentage of dogs detected in dog images data: 100%

Dog Breed Detection

1) Baseline Model

I have built a CNN model from scratch to solve this problem. Here we will call it the baseline model. The CNN model has three layers with Relu activation. Using different kernel sizes, strides padding and Max-Pooling for each layer, the size of the original image has been reduced to (7,7) and the original depth of 3 has been transformed to 128: (224, 224, 3) -> (7, 7, 128). We can extract spatial features this way from a given image. We increase the depth or add more filters so that the network can learn more significant features in the image so that the model can generalize better. The Cross-Entropy Loss as a cost function and Adam as the optimizer are used. The model is trained for 50 epochs with batch size 20 on train data located in the dag image dataset, I got “Training Loss: 4.530296” and “Validation Loss: 4.330666”.

```python
import torch.nn as nn
import torch.nn.functional as F

# define the CNN architecture
class Net(nn.Module):
    ### TODO: choose an architecture, and complete the class
    def __init__(self):
        super(Net, self).__init__()
        ## Define layers of a CNN

        # Conv Layers
        self.conv1 = nn.Conv2d(3, 32, 3, stride=2, padding=1)
        self.conv2 = nn.Conv2d(32, 64, 3, stride=2, padding=1)
        self.conv3 = nn.Conv2d(64, 128, 3, padding=1)
        # maxpool
        self.pool = nn.MaxPool2d(2, 2)
        # fc layers
        self.fc4 = nn.Linear(7*7*128, 2048)
        self.fc5 = nn.Linear(2048, 512)
        self.fc6 = nn.Linear(512, 133)    #number of classes = 133
        # dropout
        self.dropout = nn.Dropout(0.25)    #dropout of 0.25
        self.batch_norm = nn.BatchNorm1d(512)

    def forward(self, x):
        ## Define forward behavior
        x = F.relu(self.conv1(x))
        x = self.pool(x)
        x = F.relu(self.conv2(x))
        x = self.pool(x)
        x = F.relu(self.conv3(x))
        x = self.pool(x)

        # flatten
        x = x.view(-1, 7*7*128)
        x = self.dropout(x)
        x = F.relu(self.fc4(x))
        x = self.dropout(x)
        x = F.relu(self.batch_norm(self.fc5(x)))
        x = self.dropout(x)
        x = self.fc6(x)
        return x

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

# instantiate the CNN
baseline_model = Net()

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

Model Performance:

Model performance of baseline model:

- Test Lost: 4.327450
- Test Accuracy: 4% (40/836)
- Precision: 0.0147228496059
- Recall: 0.0360961928631

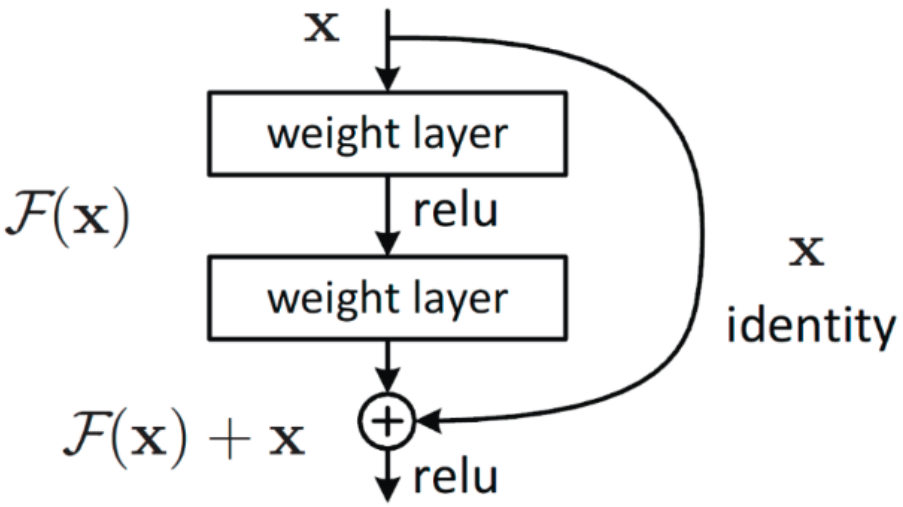
We can tell from the accuracy, precision and recall above that the model isn't performing well. More data is needed for training.

2) Transfer Learning Model

ResNet101 is used for this multi-class classification task. ResNet101 is chosen because it gives outstanding performance in image classification. ResNet is built with "Residual

Blocks" which are basically the skip connections between layers. This creates an identity connection between initial layers. This creates an identity connection between the initial layer to the final layers, reducing the risk of vanishing/exploding gradient problem that also helps in reduced risk of underfitting and overfitting on training data.

Model Architecture



Residual Block

The model is trained for 20 epochs with batch size 20 on train data located in the dag image dataset, I got "Training Loss: 2.109468" and "Validation Loss: 1.560796".

Model Performance:

Model performance of baseline model:

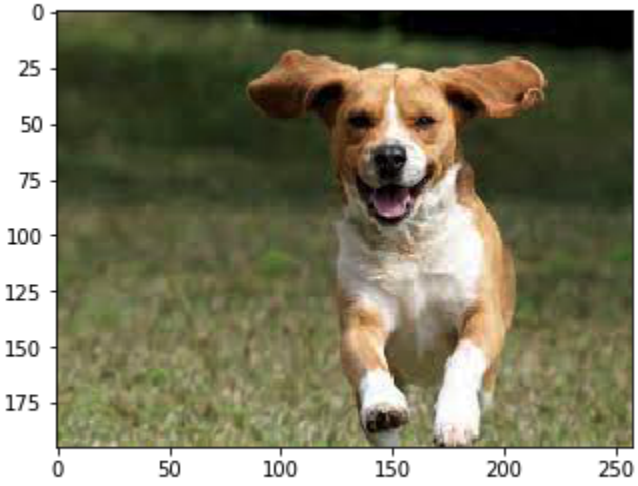
- Test Lost: 1.603196
- Test Accuracy: 75% (628/836)
- Precision: 0.687219986756
- Recall: 0.691618928273

The precision and recall have improved a lot. 0.69 precision indicates that our model is predicting the results with a precision of 69%, which means our model is 69% precise. Recall of 0.69 is also not bad considering we have only trained the model on 20 epochs only. We can further increase the accuracy, recall and precision by doing more hyperparameter tuning. Also, training the model for more epochs will help to boost these numbers!

Results

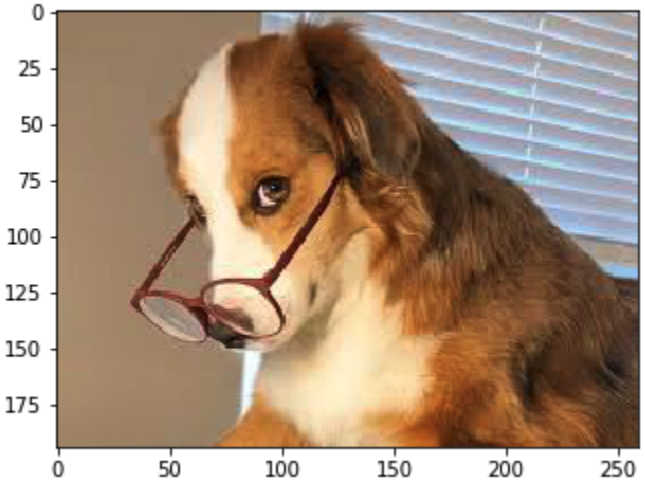
Below are some interesting results produced by our models. All the images were

Dog Detected!



This is a "Beagle" kind of dog!

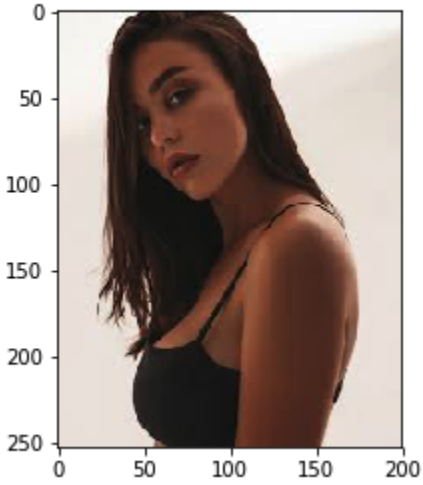
Ooops! Nothing to detect!



Dog Detected!

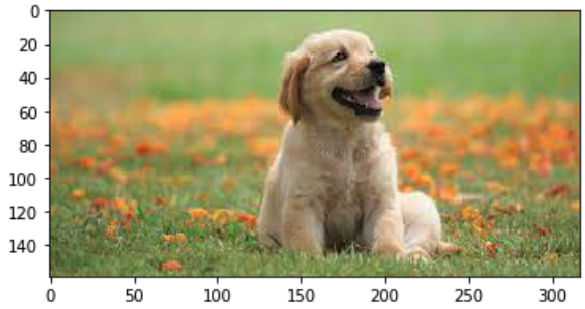
This is a "Pomeranian" kind of dog!

Ooops! Nothing to detect!



chosen from Google Photo.

Dog Detected!



This is a "Golden retriever" kind of dog!

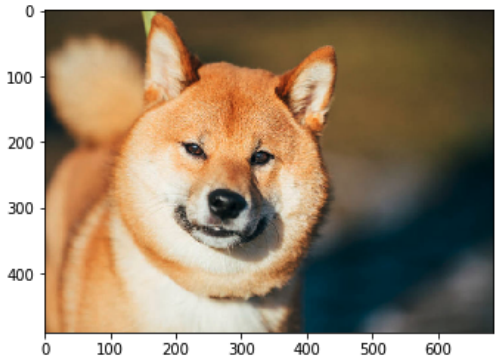
Dog Detected!

This is a "Australian shepherd" kind of dog!

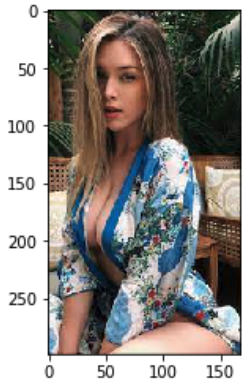
Human Detected!

Interesting, this human looks like "Japanese chin"!

Ooops! Nothing to detect!



Human Detected!



Interesting, this human looks like "Great dane"!

Improvements

- The algorithm could be made available in a web application to be more accessible to people. By involving more people in training the algorithm, we could develop more ways to fine-tune or create new algorithm models.
- Training on more data can help in this case, also augmentation of the data may help like cropping images to correct areas may help.
- Hyperparameter tuning always helps.
- Trying out better pretrained models.

Conclusion

It was fun being able to explore, play and find insight from data. Creating a model from scratch was really interesting. I got to experience the proper Machine Learning workflow of creating and experimenting in Machine Learning model creation. Modern deep learning frameworks like Tensorflow, PyTorch and etc; have really enabled us to work more easily and efficiently to train models and test them. This allows us to focus more on solving the problem at hand instead of being caught in technical issues.

References

1. https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html
2. https://neurohive.io/en/popular-networks/vgg16/
3. https://neurohive.io/en/popular-networks/resnet/