

Udacity Machine Learning Nanodegree

Dog Breed Classification Report

Mohammed Aydid Hasan

August 22nd, 2020

1. Definition

Project Overview:

Machine Learning Engineer Nanodegree course has been created upon different type of Supervised and Unsupervised Learning. Supervised Learning deals with Regression and Classification. And where classification is in high demand because now a days it is implementing across all the fields of the business aspects for different applications and are very prominent in Image classification for medical diagnosis and manufacturing. Not only that, it is one of the growing area of research with various real-world applications.

My purpose to select this project was to familiarize myself with convolutional neural networks (CNN). This project aims to predict the dog's breed from 133 classes. There are commercial and business-use cases for such an application as well. Here are some of the points that I want to show:

1. America's pet industry is estimated around \$69.51 Billion as of 2017 and estimated to grow by 3.7% to \$72.13 Billion. ¹
2. Each sponsored post can command around \$2,000. ²
3. Grumpy Cat's Net Worth: ~\$100 million. ³
4. Sponsored posts by high-profile pets can range from \$10,000-\$15,000 per post. ⁴
5. "On average, dog people post a picture or talk about their dog on social media six times per week." ⁵

Problem Statement:

The main objective of this project is to use Deep Learning techniques to build a classifier that could first detect whether a dog or a human was detected in the provided image and then, classify the detected dog into one of over 133 dog breed categories (and in case of a human, what dog breed the detected human looks like). I have used classification method against this dataset because classification method is the perfect method to train our model to detect a class from 133 different classes. When given an image sample in a computer readable format (such as a .jpeg/.png file), we want our model to be able to determine if it contains one of the target class with a corresponding likelihood score. Conversely, if none of the target image were detected, we will be presented with an unknown score. This model is using Convolutional Neural Network with Transfer learning to classify the dog breeds. For achieving the accuracy, I used ResNet50 as a transfer learning method. Which brings up to 77% accuracy where the requirement was to get more than 60% accuracy, that means my model worked better than the expectation.

Datasets and Inputs:

The data set was imported from sklearn provided in a Udacity's workspace consists of 8351 dog images in 133 different categories. The project has been done at Udacity's workspace (GPU-enabled). We needed to split the dataset into training set, validation set and testing set. The model will be tested on 100 images of each (human and dog). There are 6680 training dog images, 835 validation dog images and 836 test dog images. The human face detector will be relied on OpenCV's Haar Feature-based Cascade Classifiers. It has been trained on 13233 human faces which will be imported from sklearn. And the dog detector will be built on a pre-trained ResNet50 model. Its purpose is to confirm if an image is a dog or not. Input should be any sort of computer readable image.

Metrics:

The evaluation metric for this problem is simply the Accuracy Score. The most popular metrics used in CNN are "accuracy", "binary accuracy", "categorical accuracy", "sparse categorical accuracy", etc. But as we are classifying dog breeds, so we can't use binary or sparse categorical accuracy. Accuracy is the ratio of number of correct predictions to the total number of input samples. However, it works well only if there are equal number of samples belonging to each class. In this project, there are total 133 dog breed as class labels. Based on the distribution of training/validation/testing selected,

the classes were approximately evenly distributed, except a couple of classes. Therefore, we should be able to use “accuracy” metric in training.

Here is the equation of accuracy:

$$\text{Accuracy} = (TP+TN)/(TP+TN+FP+FN)$$

	Predicted No	Predicted Yes
Actual No	TN	FN
Actual Yes	FP	TP

Where, TP = True Positive, TN = True Negative, FP = False Positive and FN = False Negative.

Recall is defined as the number of true positives (TP) over the number of true positives plus the number of false negatives (FN).

$$\text{Recall} = \frac{TP}{TP + FN}$$

Sensitivity refers to the rate of correctly classified positive and is equal to TP divided by the sum of TP and FN.

Sensitivity may be referred as a True Positive Rate.

$$\text{Sensitivity} = \frac{TP}{FN + TP}$$

Specificity refers to the rate of correctly classified negative and is equal to the ratio of TN to the sum of TN and FP

$$\text{Specificity} = \frac{TN}{TN + FP}$$

1. Analysis

Data Exploration and Visualisation:

The data was imported from sklearn, where there are total 133 dog categories, which is consist of total 8351 dog images. And also 13233 human images. I splitted

the images in 3 sections, which are training data set, validation data set and test data set. In training data set, we have total 6680 training images. In validation and test data set we have 835 and 836 images.

```
#Displaying Trainign data
#####
def display_img(inp):
    inp = inp.numpy().transpose((1, 2, 0))
    inp = np.clip(inp, 0, 1)

    fig = plt.figure(figsize=(50, 25))
    plt.axis('off')
    plt.imshow(inp)
    plt.pause(0.001)
# Display!
dataiter = iter(test_loader)
images, labels = dataiter.next()
# Convert the batch to a grid.
grid = utils.make_grid(images[:5])
display_img(grid)
```

Num training images: 6680

Num test images: 836

Num validaion images: 835



To detect humans, we first convert the tri-channel RGB image (Red-Green-Blue) into grayscale. An OpenCV function works on grayscale images and gives us the coordinates of a box of where the human face is detected. Later, we can use this to draw a blue

rectangle on the image we are plotting as shown below.

```
In [19]: 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[10])
# convert BGR image to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

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

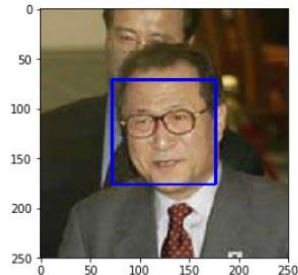
# print number of faces detected in the image
print('Number of faces detected:', len(faces))

# 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



Algorithms and Techniques:

Here I created a 5-layer CNN in PyTorch with Relu activation function with the convolution of kernel size = 3, stride =1 as it is a standard measurement for CNN..

The convolution layers squeeze images by reducing width and height while increasing the depth layer by layer.

Batch normalization is applied after every max pool layer. As we can see, first layer took 3 inputs for RGB and produces 16 outputs. So the next step has 16 convolution layers which is consist of 16 different filters. Batch Normalization 2D is a technique to provide inputs that are 0 mean or variance 1.

1. Layer 1: (3,16) input channels =3, output channels = 16
2. Layer 2: (16,32) input channels = 16, output channels = 32
3. Layer 3: (32,64) input channels =32, output channels = 64
4. Layer 4: (64,128) input channels =64, output channels = 128
5. Layer 5: (128,256) input channels =128, output channels = 256

The CNN Model architecture is:

```

In [48]: 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
        self.conv1 = nn.Conv2d(3, 16, 3)
        self.conv2 = nn.Conv2d(16, 32, 3)
        self.conv3 = nn.Conv2d(32, 64, 3)
        self.conv4 = nn.Conv2d(64, 128, 3)
        self.conv5 = nn.Conv2d(128, 256, 3)

        self.fc1 = nn.Linear(256 * 6 * 6, 133)
        self.max_pool = nn.MaxPool2d(2, 2, ceil_mode=True)

        self.dropout = nn.Dropout(0.20)

        self.conv_bn1 = nn.BatchNorm2d(224,3)
        self.conv_bn2 = nn.BatchNorm2d(16)
        self.conv_bn3 = nn.BatchNorm2d(32)
        self.conv_bn4 = nn.BatchNorm2d(64)
        self.conv_bn5 = nn.BatchNorm2d(128)
        self.conv_bn6 = nn.BatchNorm2d(256)

    def forward(self, x):
        ## Define forward behavior
        x = F.relu(self.conv1(x))
        x = self.max_pool(x)
        x = self.conv_bn2(x)

        x = F.relu(self.conv2(x))
        x = self.max_pool(x)
        x = self.conv_bn3(x)

        x = F.relu(self.conv3(x))
        x = self.max_pool(x)
        x = self.conv_bn4(x)

        x = F.relu(self.conv4(x))
        x = self.max_pool(x)
        x = self.conv_bn5(x)

        x = F.relu(self.conv5(x))
        x = self.max_pool(x)
        x = self.conv_bn6(x)

```

I put batch size 10. I have used a very common method for pre-processing which is “RandomResizeCrop(224)”. It resizes the images with fixed size of 224 and transforms it into tensor. I augmented the dataset by using random rotation and random horizontal flip.

```

x = self.conv_bn2(x)

x = F.relu(self.conv2(x))
x = self.max_pool(x)
x = self.conv_bn3(x)

x = F.relu(self.conv3(x))
x = self.max_pool(x)
x = self.conv_bn4(x)

x = F.relu(self.conv4(x))
x = self.max_pool(x)
x = self.conv_bn5(x)

x = F.relu(self.conv5(x))
x = self.max_pool(x)
x = self.conv_bn6(x)

x = x.view(-1, 256 * 6 * 6)

x = self.dropout(x)
x = self.fc1(x)
return x

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

# instantiate the CNN
model_scratch = Net()
print(model_scratch)
# move tensors to GPU if CUDA is available
if use_cuda:
    model_scratch.cuda()

Net(
  (conv1): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1))
  (conv2): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
  (conv3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
  (conv4): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1))
  (conv5): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1))
  (fc1): Linear(in_features=9216, out_features=133, bias=True)
  (max_pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=True)
  (dropout): Dropout(p=0.2)
  (conv_bn1): BatchNorm2d(224, eps=3, momentum=0.1, affine=True, track_running_stats=True)
  (conv_bn2): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv_bn3): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv_bn4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv_bn5): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv_bn6): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
)

```

Train a CNN to Classify Dog Breeds (via transfer learning):

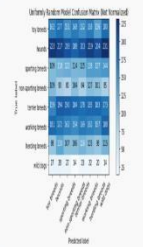
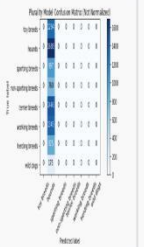
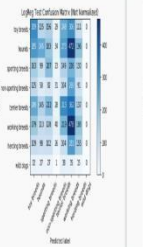
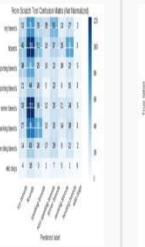
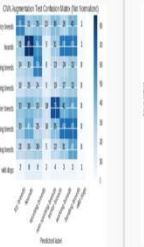
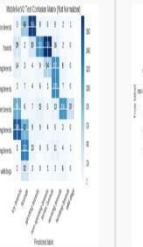

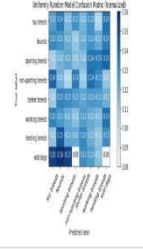
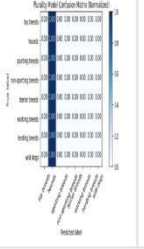
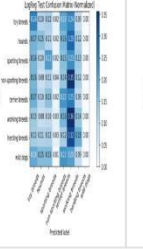
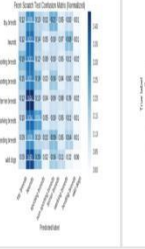
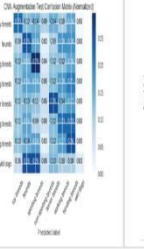


Next, I used pre-extracted “bottleneck features” which are an output of a pre-trained library applied on our train, test and validation datasets. I used ResNet which is already trained to extract the feature. The connected layers, I have created produce 133 breed as an output against it's input. Some of the advantages of using ResNet50 are given below:

1. It is a pre-trained image classification model.
2. ResNet50 is a powerful backbone model that is used very frequently in many computer visions tasks.
3. ResNet50 uses skip connection to add the output from an earlier layer to a later layer. This helps it mitigate the vanishing gradient problem.

In fact this model is perfect for our problem because this problem has RGB images with different sizes. And that sort of problem can easily be solved by using ResNet50.

Benchmark Model:

For this problem, I tried to beat the existing model by CNN using RestNet50 transfer learning. From the image below, we can see that the accuracy score for RestNet50 model is 83.28%. I was able to get 77% accuracy by using ResNet50 Model.

Model:	Random	Plurality	Logistic	CNN From Scratch	CNN with Data Augmentation	MobileNetV2	ResNet50
Categorical Loss	30.2838	27.7437	1.9771	5.8862	2.5033	0.7060	0.6864
Accuracy Score	12.32%	19.67%	20.26%	22.54%	22.88%	79.51%	83.28%
Time	< 1 sec	< 1 sec	19.6 mins	17.8 mins	18.5 mins	20.8 mins	50.3 mins
Confusion Matrix							
Confusion Matrix (Normalized)							

Source: <https://hljames.github.io/dog-breed-classification/>

Here is the image of my accuracy test.

(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 [62]: test(loaders, model_transfer, criterion_transfer, use_cuda)
```

Test Loss: 1.375510

Test Accuracy: 77% (646/836)

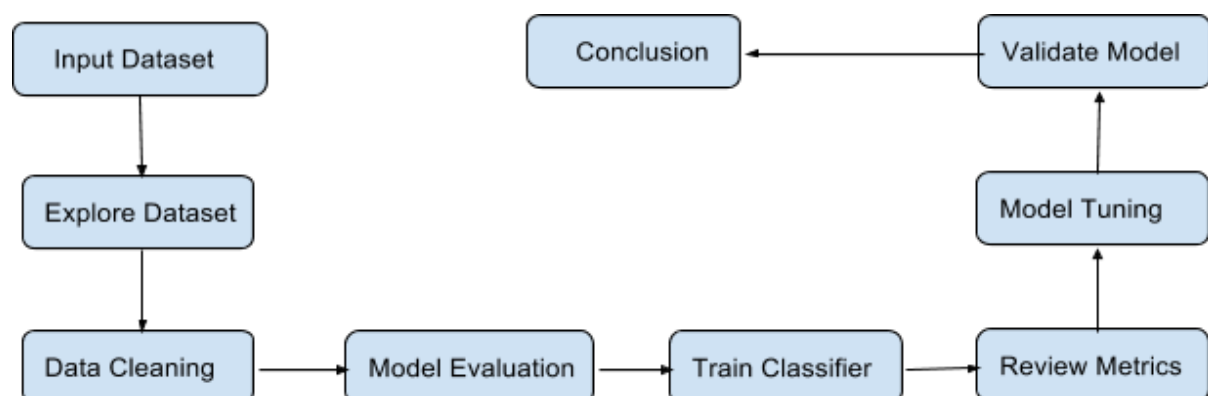
3. Methodology:

Data Pre-Processing:

As I mentioned previously, I split the dataset into 3 sections such as train, valid and test. I tried to make it as balanced as possible, but it is slightly imbalanced. We have 835 data in valid section and 836 in test section. I used torchvision to split my data sets. I have used a very common method for pre-processing which is RandomResizeCrop(224). It resizes the images with fixed size of 224 and transforms it into tensor. I augmented the dataset by using random rotation and random horizontal flip.

Implementation:

The project follows typical predictive analytics hierarchy as shown below:



At first, I created CNN from Scratch and then created CNN again with Transfer Learning where I used ResNet50 model to get the best accuracy.

In my first CNN(from Scratch), I implemented 10 epochs which basically took care of my training loss and validation loss. It brings down training loss from 0.000711 to 0.000597 and Validation loss from 0.005364 to 0.004585

In the second CNN where I used transfer learning, I implemented 20 epochs which actually helped me to get 77% accuracy where my requirement was to get more than

60% accuracy. When I implemented 15 epochs, I got only 66% accuracy and then I changed it to 20.

4. Result:

Result of my model is pretty much satisfactory. Udacity already set minimum score that I should achieve for both CNN and I meet all the requirements and was able to get more score than the minimum score. The required score for CNN from scratch was 10% and I was able to get 14%. And for CNN with transfer learning, minimum score was set to 60% where I achieved 77% which is even close to the benchmark score.

Justification:

As I am new in Machine learning, it is tough to beat benchmark score. In fact, the project I choose is one of the common projects with lot of solutions. But I was close to the benchmark score which indicates, I am in the right track. And it will be possible in near future to beat any benchmark score for different machine learning problem. At last I would say, there are many things that could be improved such as different hyperparameter tuning. The model accuracy could improve by data augmentation and adding more training labelled data since the model was slightly imbalanced. With acquiring more data, the Convnet models would be more capable of learning the more relevant and specific dog features from the images. But it requires a considerable amount of time and memory usage. As I mentioned before, one improvement could be applying more appropriate fine-tune strategies to adjust the weights based on the data.

Conclusion:

In this work, I have implemented the haar cascade classifier and ResNet50 pre-trained weights to detect human faces and dogs. I also spent some time exploring the data to find images characteristics, which helped me to apply more appropriate techniques. The model performed better than a random guess, 1 in 133, but still lot of things to improve.

In the end, I have seen that my model is predicting human and dog accurately but the classification for dog breed are not 100% accurate.