

# Udacity Machine Learning Engineer Nanodegree

CapStone Project Report

## **DOG BREED CLASSIFIER**

Terrence Goh

May 6, 2022

---

<b>DOMAIN BACKGROUND</b>	<b>3</b>
<b>PROBLEM STATEMENT</b>	<b>3</b>
<b>DATASET AND INPUTS</b>	<b>4</b>
<b>SOLUTION STATEMENT</b>	<b>4</b>
Python Libraries	4
Google Colab	5
<b>BENCHMARK MODEL</b>	<b>5</b>
CNN model developed from scratch	5
Pre-trained CNN model	5
<b>EVALUATION METRICS</b>	<b>5</b>
<b>PROJECT DESIGN</b>	<b>6</b>
Import Data	6
Human Face Detection	6
Dog Detection	6
Dog Breed Classification	7
CNN built from scratch	7
Pretrained model using transfer learning	9
Test Algorithm	10
<b>CONCLUSION</b>	<b>10</b>
<b>References</b>	<b>11</b>

---

## DOMAIN BACKGROUND

When computer vision started in the 1960s, its aim was to try and mimic human vision systems and ask computers to tell us what they see, automating the process of image analysis. Early researchers were extremely optimistic about the future of this exciting field. This kind of technology is widely regarded as the precursor to artificial intelligence and potentially could transform the world.

Early research attempts met with limited success, and with computer hardware and computing power at that time, it was not able to solve even the basic tasks.

After more than 50 years had passed, the world now has the Internet with a large collection of data and photos uploaded by everyday users. Computing power has grown tremendously and hardware costs keep going down every year.

The growth of the internet and small, more powerful computers are a big catalyst for a team from the University of Toronto to win an image classification competition in 2012 at the ImageNet Large Scale Visual Recognition Challenge (ILSVRC). They created an artificial deep neural network named AlexNet that won the competition with an error rate of 15.3%, versus the runner up 26.2% [1].

Nowadays, the gold standard for image recognition tasks is a class of deep neural networks called **convolutional neural networks (CNNs)** for short) and this project will explore using CNNs to solve an image classification problem.

## PROBLEM STATEMENT

The goal of this project is to develop a machine learning algorithm that takes a photo or an image as an input, and classify it according to the conditions below:

1. Given an image of a dog, identify an estimate of the canine's breed.
2. Given an image of a human face, identify the resembling dog breed.

---

## DATASET AND INPUTS

The dataset is provided by Udacity which consists of thousands of images, both humans and dogs. The photos are sorted into three folders: test, train and validation. The datasets can be found at:

1. Labeled Faces in the Wild dataset:  
<https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/lfw.zip>
2. Dog dataset: <https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/dogImages.zip>

## SOLUTION STATEMENT

Referring to the problem statement, the algorithm needs to identify if the photo is a human or a dog. If it is a dog, the algorithm will find the closest match of the dog breed. We can use one of the models trained from the ImageNet dataset, which has a large number of images labelled with dog. Some model candidates are Alexnet, VGG16, Inception and Resnet50, just to name a few.

If the photo is a human, the algorithm will try to find the dog breed that closely resembles the human face...just for fun. However, to detect a human face in the first place, we will need another algorithm more suited for this approach, which is Haar Cascades<sup>[2]</sup> from the OpenCV software library.

Back to identifying the dog breed, the pre-trained models above can only identify if the image contains a dog, but not classifying the dog breed itself. We have to apply transfer learning<sup>[3]</sup> and customize the models to our needs.


## Python Libraries

Not all the libraries are listed here, but the important ones are:

1. Requests
2. Numpy
3. Glob
4. OpenCV
5. Matplotlib
6. PIL
7. Torch, torchvision

---

## Google Colab

Additionally, the project's notebook can be executed in Google Colab environment by clicking the link  at the notebook's title.

There are several benefits of this feature:

1. When the free AWS credits provided are run out, one can continue their work for FREE using another environment
2. The free tier includes the GPU runtime for accelerated model training
3. Frequently used Python libraries for data science and machine learning are pre-installed and up-to-date
4. Automatic code completion and doc string display for modules

The notebook also provides the ability to save the model checkpoints in Google Drive if the user didn't make a copy and save to his/her own Google Drive account, as the model parameters will be lost when the Colab session is terminated (i.e. when the notebook is closed).

## BENCHMARK MODEL

The benchmark model(s) for this project are developed with the performance metrics given below by the Capstone project's requirements.

### CNN model developed from scratch

Attain a test accuracy of **at least 10%**.

### Pre-trained CNN model

Attain a test accuracy of **at least 60%**.

## EVALUATION METRICS

The most common evaluation metric for image classification task is accuracy, which is the fraction of correct predictions over total number of predictions. The equation is expressed as a percentage as shown below:

$$accuracy \% = \frac{\text{correct predictions}}{\text{number of predictions}} \times 100$$

---

## PROJECT DESIGN

This section outlines the steps that were taken from beginning till the end of the project.

### Import Data

The dataset provided is imported and explored to check the quality of the data, whether the photos are clear or obscured, or is it “clean” as in for example, if there are human photos in the dog dataset. It is also good practice to check if the number of photos per category are sufficient(at least 10 or more) and equally distributed to avoid overfitting to a particular dog breed.

In the dog image folders, the training, validation and test samples are already split according to the table below:

Datasets	Train	Validation	Test	Total
Dog images	6680 (80%)	835 (10%)	836 (10%)	8351 images

Looks like the test and train sets are split nicely to a good rule-of-thumb of 80:10:10 ratio for training.

### Human Face Detection

Once the dataset preparation is done, the Haar Cascade classifier is applied to find human faces.

Using the `openCV` library, a pre-trained face detector model using Haar Cascade classifier is loaded. The input photo is converted into a grayscale image. The `detectMultiScale` function is applied on the image to find out the features of a human face. If a face is found, a bounding rectangle(s) is drawn and the number of faces is returned as an integer result.

A `face_detector` function is created to return True if a face is found, or False otherwise.

### Dog Detection

For this task, the goal is to find the presence of a dog in the image, which is a binary classification problem (i.e. dog is found or not found).

The pre-trained VGG16 model is used since it has already been trained on a very large ImageNet dataset (1000 categories, 1.2M images [4]) for many hours and fine-tuned to identify 1000 categories.

The code implementation begins with preprocessing the input image before sending it to the pretrained model. Firstly, the input image is resized to 256 x 256, center cropped to 224 x 224,

---

converted into a PyTorch tensor and the 3-channel RGB tensor is normalized to between 0 and 1 using the mean and standard deviation values of the ImageNet dataset. The input image is center cropped to 224 x 224 because the VGG16 model expects the input image of this dimension.

The preprocessed image is converted into a tensor and fed into the model for prediction. The output of the model is an integer representing the key of a dictionary of 1000-object classes [5]. A simple `dog_detector` function is written using the VGG model.

If the index falls within a value of “151-268” (both inclusive), most likely the image is a dog since the index represents different dog breeds in the ImageNet dataset.

## **Dog Breed Classification**

For this task, we developed 2 CNN models - CNN built from scratch and a modified, pretrained model to detect dog breed using transfer learning.

### **CNN built from scratch**

Similarly, I decided to build a preprocessed image pipeline using Pytorch Transforms.Compose function.

For the training dataset, I decided to augment the dataset to create more variety of images by randomly resizing and cropping training images to 256 by 256 dimension, rotating them by 20 degrees and flipping them horizontally.

As for the validation and test datasets, the images are not augmented using the steps above, other than resizing the input to 256 x 256. That is because the objective of validation and test is to evaluate the models actual performance after training, and not to improve the validation/test metrics. Otherwise, the model will overfit and perform poorly on the test dataset and in the real world scenario.

Using the `torchinfo` package, the CNN model that I have built has the architecture that looks like this:

```
=====
Layer (type:depth-idx)          Output Shape          Param #
=====
Net                               --                    --
├─Conv2d: 1-1                    [64, 32, 247, 247]    9,632
├─MaxPool2d: 1-2                 [64, 32, 123, 123]    --
├─Conv2d: 1-3                    [64, 64, 122, 122]    8,256
├─MaxPool2d: 1-4                 [64, 64, 61, 61]     --
├─Conv2d: 1-5                    [64, 128, 60, 60]    32,896
├─MaxPool2d: 1-6                 [64, 128, 30, 30]     --
├─Conv2d: 1-7                    [64, 128, 29, 29]    65,664
├─MaxPool2d: 1-8                 [64, 128, 14, 14]     --
├─Conv2d: 1-9                    [64, 256, 12, 12]    295,168
├─MaxPool2d: 1-10                [64, 256, 6, 6]       --
├─Linear: 1-11                   [64, 1330]            12,258,610
├─Dropout: 1-12                  [64, 1330]            --
├─Linear: 1-13                   [64, 133]             177,023
=====

Total params: 12,847,249
Trainable params: 12,847,249
Non-trainable params: 0
Total mult-adds (G): 60.10
=====

Input size (MB): 50.33
Forward/backward pass size (MB): 1797.96
Params size (MB): 51.39
Estimated Total Size (MB): 1899.68
=====
```

The CNN model has 5 conv layers with zero padding and stride of 1. The 1st conv layer expects an input image size of dimension 256 x 256 (width x height). A typical approach to CNN design is when the conv layers get deeper, the size of the image gets smaller but the number of channels gets higher(from 32 to 256). Higher number of channels allow the model to learn more features of the image like edges, shapes and objects.



---

The output dimension (width, height) of each conv layer is given by this formula:

$$D = \frac{W - F + 2P}{S} + 1$$

D = output dimension (i.e. width or height)  
W = input dimension (width or height)  
F = filter size  
P = padding  
S = stride

After each conv layer, there's a relu activation layer followed by a maxpool layer. After the 5th conv layer, the output 3D volume is flattened to a 1-dimension tensor to create a 9216-input (256 x 6 x 6) fully connected(FC) layer. The output of this FC layer is 1330 nodes that are fed into another relu activation layer. The outputs of the relu layer is then connected to a dropout layer which will randomly turn off 25% of the 1st FC layer's output nodes to prevent overfitting.

Lastly, it is connected into a 2nd FC layer that has 133 output classes. Dropout is not used at the model's output layer as we want all the 133 output nodes, which represent all the dog breed classes. This final FC layer produces the class scores for each 133 outputs. The highest score will be the most likely dog breed.

I have chosen the cross entropy loss function, which is widely used as a loss function for training a classification problem with many classes. Also, the Adam optimizer algorithm is chosen for faster convergence to the local minima when calculating the gradient descent. A small learning rate of 0.001 is to ensure the local minima is found and will not “bounce around” the valley.

Next, a training and validation loop is developed to train the model with 50 epochs with a batch size of 64 images. The model is saved at each iteration if the validation loss is smaller than the previous epoch's validation loss score, so we can save the model with the best model parameters.

On the test dataset, this model achieved an impressive accuracy of **28%**, almost triple the minimum benchmark score stated earlier of **10%** !

## Pretrained model using transfer learning

For the pretrained model, ResNet-152 is chosen because of its very high number of hidden layers. The intuition is that the high number of hidden layers contains more information about the image features, which helps to identify the difference in dog breeds.

---

Since we know that the model has been trained on a large dataset, the hidden layers contain useful information about object features that we want to keep for our goal to identify different dog classes. We want to “transfer” the knowledge learned from one dataset, to another similar dataset using *transfer learning*.

Using transfer learning approach, we instantiate the resnet model with argument `pretrained=True` to preserve all the model parameters in the hidden layers. However, only the last output FC layer is “unfrozen” so that its original model parameters are not kept, but recalculated during training with the dog images dataset. Also, this layer’s output features are changed from 1000 to 133 to fit our 133 dog classes.

```
ResNet-152 last fc layer: Linear(in_features=2048, out_features=1000, bias=True)
Our fc layer: Linear(in_features=2048, out_features=133, bias=True)
```

Before training, the images need to be preprocessed to the correct input dimensions of 224 x 224 for the Resnet model. I can re-use the preprocessing function that I have developed earlier, but a small adjustment is needed since it was resizing images and cropped to 256 x 256 instead.

After training, the model returns a test accuracy of **87%**, which meets the benchmark requirement of at least **60%**.

Finally, I developed the `predict_breed_transfer` function to return the predicted dog breed using the pretrained model earlier.

## Test Algorithm

Lastly, the pieces above are integrated into a final working demo of the project in the `run_app` function. Firstly, the dog breed is predicted given an input image using the `predict_breed_transfer` function. Next, `face_detector` is used to find a human face in the same image. If it returns True, then a message stating the image is actually a human face but closely resembles a particular dog breed. However, if the image is actually a dog using the `dog_detector` function, the dog breed will be returned. If the image is neither a dog or human face, an error message is displayed.

## CONCLUSION

From the project steps explained above, I have learned it is not a trivial task, even for humans, to predict a dog breed from images. Some dogs look very similar to a particular dog breed but it turns out the actual result might differ.

---

Another lesson learnt is to determine the input dimensions of the CNN model, and to calculate the output dimension of each layer carefully using the CNN output dimension formula. Otherwise, the model will return an error during the compilation step.

Also, when the dog dataset is normalized using mean and standard deviation values from another dataset (ImageNet), the impact is insignificant although I observe some blog post advice to recalculate those values for the specific dataset in use. Maybe because the mean and standard deviation is trained on a large dataset of 1 million-plus images.

Lastly, some possible improvements to the algorithm is to add more sample size to dog breed categories which have lower number of images, and to add dog photos taken with different angles or objects in the photo like a toy ball, scarf, different background environment etc.

Overall, this is a fun project and there are many lessons learnt. Looking forward to further exploring other computer vision tasks!

## References

- [1]. Jerry Wei, July 3rd, 2019. AlexNet: The Architecture That Challenged CNNs.  
<https://towardsdatascience.com/alexnet-the-architecture-that-challenged-cnns-e406d5297951>
- [2]. OpenCV v4.0 Cascade Classifier.  
[https://docs.opencv.org/4.0.0/db/d28/tutorial\\_cascade\\_classifier.html](https://docs.opencv.org/4.0.0/db/d28/tutorial_cascade_classifier.html)
- [3]. Jason Brownlee, Dec 20, 2017. A Gentle Introduction to Transfer Learning for Deep Learning.  
<https://machinelearningmastery.com/transfer-learning-for-deep-learning/>
- [4] Olga Russakovsky\*, Jia Deng\*, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg and Li Fei-Fei. (\* = equal contribution) ImageNet Large Scale Visual Recognition Challenge. IJCV, 2015.  
<https://www.image-net.org/challenges/LSVRC/2014/>
- [5] Yagnesh Revar, ImageNet 1000 class idx to human readable labels.  
[https://gist.github.com/yrevar/942d3a0ac09ec9e5eb3a#file-imagenet1000\\_clsidx\\_to\\_labels-txt](https://gist.github.com/yrevar/942d3a0ac09ec9e5eb3a#file-imagenet1000_clsidx_to_labels-txt)