**Joost Zegers**

# CNN Dog Breed Classifier

**Capstone Project Report - Machine Learning Engineer Nanodegree**

## DEFINITION

## PROJECT OVERVIEW

In this project, we develop an algorithm that classifies the breed of a dog from an image. This is a multi-class classification problem that we solve using Convolutional Neural Networks (CNNs).

We create three classification models:

1. *Dog breed classifier* - identifies the most probable dog breed from an image (multi-class)
2. *Dog detector* - detects whether an image contains a dog (binary)
3. *Human face detector* - detects whether an image contains a human face (binary)

Finally, we combine these three models to construct the final algorithm. The final pipeline can be used as part of a mobile or web-based app for dog breed identification.

We use the *Stanford Dogs* dataset[1] and the *Labeled Faces in the Wild* (LFW) dataset[2] to develop and evaluate our models. Both datasets have been pre-processed by the creators of this project at Udacity.

The project design and objectives were created by Udacity, as part of their Nanodegree programs.

## PROJECT STATEMENT

The goal of the project is to build a pipeline that processes real-world images and uses CNNs to:

1. Classify whether an image contains a human face, a dog or neither
2. If a dog is detected: estimate the dog's breed (with >60% accuracy)
3. If a human face is detected: estimate the most-resembling dog breed

Once completed, the pipeline should be able to be used as part of a mobile or web-based app for dog breed identification.

---

[1] http://vision.stanford.edu/aditya86/ImageNetDogs/
[2] http://vis-www.cs.umass.edu/lfw/

## METRICS

### Validation

Model validation is done using a categorical cross-entropy loss function on a validation dataset. This is appropriate for training and validation of a multi-class classifier[3]. Cross-entropy loss is defined as:

$$-\sum_{i=1}^{N}\sum_{c=1}^{C} t_{i,c} \cdot log(p_{i,c})$$

$N$ - number of observations

$C$ - number of classes

$t$ - groundtruth: equals 1 for the target class and 0 for all other classes

$p$ - predicted probability that class $c$ is the true value for observation $i$

The categorical cross-entropy loss will be lower when a predicted class probability agrees with the true value.

### Evaluation

The final evaluation of the model is done using class prediction accuracy on a test dataset. The class prediction accuracy is defined as:

$$\frac{\sum_{i=1}^{N} I(t_i=\widehat{t_i})}{N}$$

$N$ - number of observations

$I$ - indicator function: returns 1 if the prediction equals the groundtruth, otherwise 0

$t$ - groundtruth: equals 1 for the target class and 0 for all other classes

$\widehat{t}$ - predicted class: the class that corresponds with the highest value of $p$

[3] Machine Learning Engineer Nanodegree > Extracurricular > 2. Additional Materials: Convolutional Neural Networks > Convolutional Neural Networks > 8. Loss & Optimization

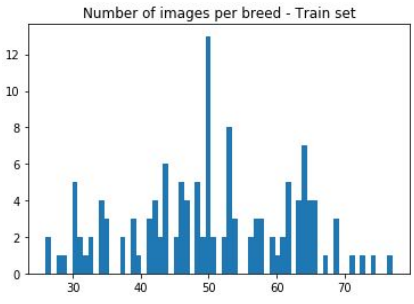
# ANALYSIS

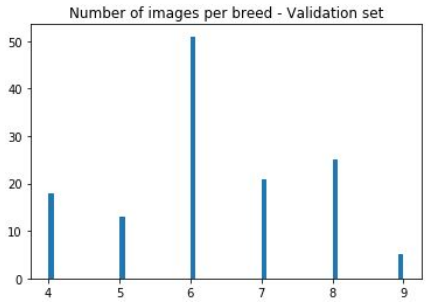
## DATA EXPLORATION AND VISUALIZATION

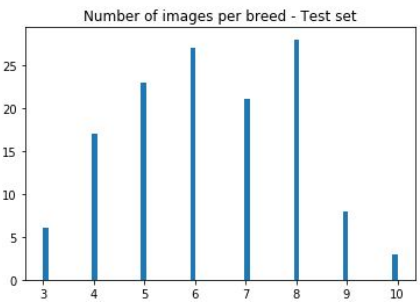
Both the *Stanford Dogs* dataset and the LFW dataset have been pre-processed by the project creators at Udacity.

### Dog images

The *dog image* dataset contains 8,351 labeled images of 133 dog breeds, already split into train (6,680 images), validation (835 images) and test (836 images) datasets. The images have different sizes, backgrounds and rotations. The classes in the dataset are not perfectly balanced, because the number of available images per dog breed varies. However, we do not expect the imbalance to greatly impact model performance, because the imbalance is not that big. Also, this variation might actually be representative of the variation in the number of dog breeds in the real world.




### Human face images

The *human faces* dataset contains 13,233 images that each contain one or more human faces. The background and orientation of the images varies. All images are sized 250x250. The sole purpose of this dataset in this project is to test that our pre-trained *human face detector* is working. Hence, balance is no concern here.

## ALGORITHMS AND TECHNIQUES

In the lessons about CNNs, we learned that they are used to obtain state-of-the-art results in the field of Computer Vision[4]. The image classification models that achieve highest accuracy on the Imagenet dataset all use CNNs[5]. Hence, it makes sense that we use CNNs to build our *dog breed classifier*.

[4] Machine Learning Engineer Nanodegree > Extracurricular > 2. Additional Materials: Convolutional Neural Networks > Convolutional Neural Networks > 2. Applications of CNNs

[5] https://paperswithcode.com/sota/image-classification-on-imagenet

An image classification CNN takes in an input image and pulls that image through several convolutional and pooling layers. This extracts a set of feature maps that help identify what is being shown in the image. The feature maps are flattened and pulled through several linear layers. This transforms the features into an assigned probability for each class. The class with the highest probability is the predicted class for this input image.[6]

Developing high-performing image classification CNNs from scratch is complex and requires very large datasets and a lot of computing power. Transfer learning is an approach that uses what state-of-the-art models have already learned and applies it to a new task[7]. Using transfer learning, we can build significantly more powerful CNN models compared to building from scratch.

For our *dog breed classifier*, we use a transfer learning approach and build upon the well-known VGG-16 model[8]. We expect the high-level features of our *dog image* dataset to be similar to the Imagenet dataset on which the VGG-16 model is trained. Hence, we:

- Keep all layers of the VGG-16 network intact, except the last layer
- Freeze the pre-trained weights, in order to prevent overfitting on our small data set
- Add and train a new linear layer with randomized weights and 133 output features, which corresponds with the number of dog breed classes

For the *dog detector* and *human face detector* functionality, we use fully pre-trained CNN algorithms. The *dog detector* uses a pre-trained VGG-16 model. The *human face detector* is an implementation from OpenCV that uses Haar Cascades[9].

## BENCHMARK

As a benchmark for the transfer learning *dog breed classifier*, we build a CNN from scratch (without using transfer learning) that serves as a benchmark model. By doing so, we can assess the benefit of using a transfer learning approach more clearly.

Additionally, the project creators have set an accuracy benchmark for both the CNN build from scratch (>10%) as well as the CNN that uses transfer learning (>60%). These accuracy benchmarks must be achieved on the *dog image* test set.

---

[6] Machine Learning Engineer Nanodegree > Extracurricular > 2. Additional Materials: Convolutional Neural Networks > Convolutional Neural Networks > 45. Summary of CNNs
[7] Machine Learning Engineer Nanodegree > Extracurricular > 2. Additional Materials: Convolutional Neural Networks > Transfer Learning > 1. Transfer Learning
[8] https://arxiv.org/abs/1409.1556
[9] https://docs.opencv.org/trunk/d7/d8b/tutorial_py_face_detection.html

# METHODOLOGY

## DATA PRE-PROCESSING

We follow the PyTorch documentation, which prescribes that all pre-trained models expect input images to be normalized in the same way[10]. Hence, we apply the following pre-processing steps to each image in the *dog image* dataset:

1. Convert to 3-channel RGB image
2. Resize to 224x224
3. Convert to Tensors
4. Normalize using mean = [0.485, 0.456, 0.406] and std = [0.229, 0.224, 0.225]

To improve the performance of our *dog breed classifiers*, we augment the train dataset with the following PyTorch functionality:

1. Random rotation of 25 degrees (RandomRotation)
2. Random translation and scaling (RandomResizedCrop)
3. Random horizontal flipping (RandomHorizontalFlip)

The aim of the augmentation is to increase the prediction accuracy by making the model more invariant to differences in object rotation, translation and scale.

## IMPLEMENTATION

We use the *categorical cross-entropy* loss function and a *Stochastic Gradient Descen*t optimizer for training and validation of both *dog breed classifiers*. Both models are trained for 20 epochs.

The implementation of the transfer learning CNN is described in the Algorithms and Techniques section of this report. To recap: we keep all layers of the VGG-16 network but the final layer, freeze the pre-trained weights and replace and re-train the final layer of the network. We used a learning rate of 0.001.

The implementation of the benchmark CNN is described below, in the Refinement section.

Both the *dog detector* and *human face detector* functions use fully pre-trained CNN algorithms, which are described in the Algorithms and Techniques section.

[10] https://pytorch.org/docs/stable/torchvision/models.html

## REFINEMENT

The implementation of the benchmark CNN (built from scratch) is inspired by the CNN architecture used to classify MNIST digits that was shown in the Udacity classroom. Starting from that architecture, we took the following steps:

- Preserved the (3, 3) kernels and (1, 1) padding for all convolutional layers, because:
    - it works well for the VGG models described in the Udacity classroom
    - it makes it easy to keep track of the down-sampling in the x-y dimensions
- Preserved the ReLu activations, since no real alternatives were discussed in the classroom and they appear to work well for most architectures
- Preserved the (2, 2) pooling layers after each convolution layer, to reduce the input's x-y dimensions
- Added a (2, 2) stride in the first convolutional layer to increase the down-sampling in x-y dimensions
- Added a fourth convolution layer, because this data set and objective (dog breed classification) seem more complex than the MNIST digit dataset and objective
- Preserved the 2 linear layers, ReLu activations and the (p=0.25) dropout

The final architecture of the benchmark CNN model is described below.

```
Net(
  (conv1): Conv2d(3, 16, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
  (conv2): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv4): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc1): Linear(in_features=6272, out_features=500, bias=True)
  (fc2): Linear(in_features=500, out_features=133, bias=True)
  (dropout): Dropout(p=0.25)
)
```

We tried various settings for the learning rate of the benchmark CNN and ended up using 0.05.

# RESULTS

## MODEL EVALUATION AND VALIDATION

Our *dog breed classifier* built on top of the VGG-16 model obtains an accuracy of 87% on the test set. The benchmark set by the Udacity project creators is 60%. We beat the benchmark on our first iteration. The model classifies 5/5 dog breeds correct for random dog images that we test.

Our benchmark model obtained an accuracy of 14%. The benchmark set by the Udacity project creators is 10%. We beat the benchmark on our fourth iteration.

**The large difference in performance between the transfer learning CNN and the benchmark CNN shows the power of the transfer learning approach.**

The fully pre-trained *dog detector* CNN correctly detected 100% of the dogs in a test set of 100 dog images. It falsely detected 1 dog in a test set of 100 human face images.

The fully pre-trained *human face detector* CNN correctly detected a human face 98% of the cases in a test set of 100 human face images. It falsely detected 11 human faces in a test set of 100 dog images.

## JUSTIFICATION

We are pleased with the 87% accuracy of the transfer learning CNN. It significantly beats the benchmark set by the Udacity project creators. It also classifies 5/5 dog breeds correct for random dog images that we tested. With this performance, we feel confident that we can implement the pipeline as part of a mobile or web-based app for dog breed identification.

## IMPROVEMENTS

To conclude, we list some approaches that can be explored to further improve the performance of the algorithm:

1. Optimization of hyperparameters (e.g., optimizer, learning rate, weight initialization, architecture)
2. Increase the size of the *dog image* dataset by adding more dog images
3. Increase the robustness of the *dog breed classifier* by investigating different augmentation techniques
4. Try different classification models and investigate if using an ensemble increases performance