# Udacity Dog Classification Project- Report

## Section 1: Project Definition

### Project Overview

The aim of this project was to construct a Convolutional Neural Network (CNN) to classify dog breeds from pictures or, if the picture shows a human, to return what dog breed they "look like". While this application of CNNs was primarily a bit of fun directed at learning and understanding multiple aspects of image classification, these networks can perform important tasks: Whether they are used for machine "vision" in self-driving cars or robots, or assist doctors in identifying deadly tumours, CNNs are usually built around similar principles.

This project was a guided walk-through of creating a CNN, using Udacity-provided data of pictures of dogs and humans to train, validate, and test the model.

### Problem Statement

The target of the project was to create an algorithm which could be used in a mobile or web-app to classify any user-image. This involved creating constructing and training a classification model and wrapping it in an algorithm that takes the user input and outputs the prediction.

### Metrics

The main metric focused on in this project was the accuracy of the classification, i.e.: how many of the made classifications were correct. This makes sense for a non-binary classification like in this project. In other applications like tumour detection for example, other metrics should also be included as consequences of miss-classification can be more significant than in this context.

For the various models created as part of this project there were different minimum thresholds set for accuracy: 1% for the CNN build from scratch, 60% for the final model built with transfer learning.

## Section 2: Analysis

### Data Exploration

The input data consisted of 8,351 dog images, containing dogs from 133 dog categories (or breeds). This dataset was split into 6,680 images for training, 835 for validation, and a further 836 for testing.

The input data also contained 13,233 human images.

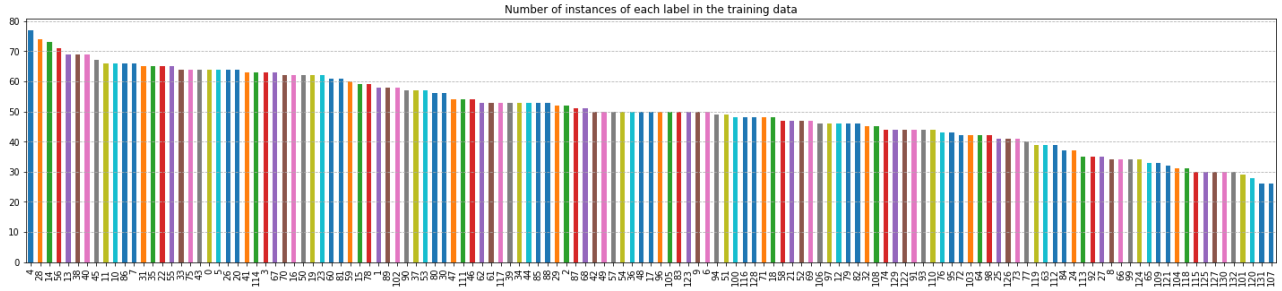
All images were coloured.

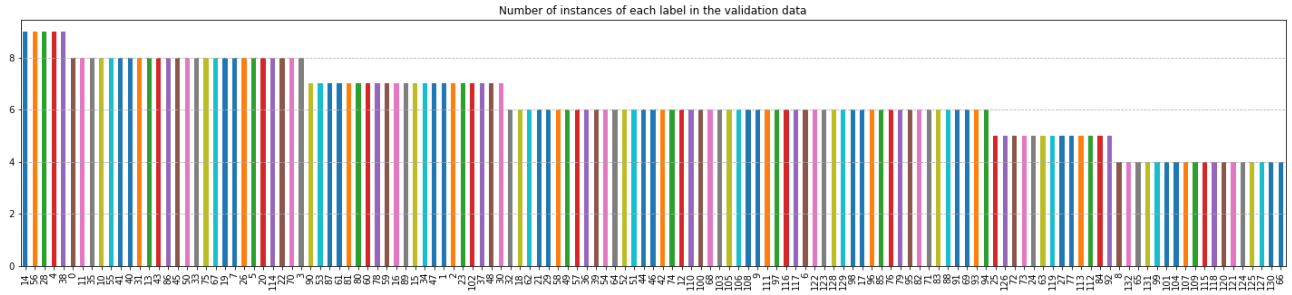
As there 133 is a very high number of possible labels, I conducted some additional analysis to check whether train, validation, and test set all contained at least one instance of each dog breed. I also checked the most and least common dog breeds in each dataset. The analysis revealed that no label had less than 26, 4, and 5 instances in training, validation, and test data respectively. While it would

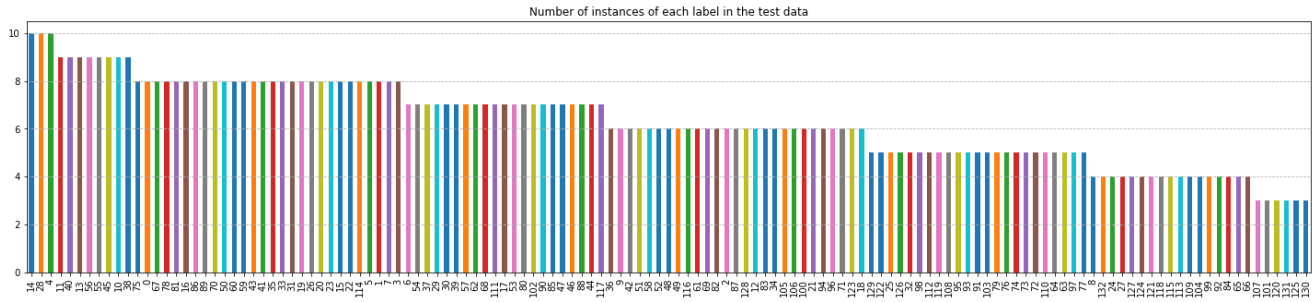
be unpractical to compare the frequency of all 133 labels across datasets, a look at the visuals below suggests that the most frequent labels are largely the same across all three subsets.

We can therefore move forward in confidence that the train, test, and validation samples are not heavily skewed.

### Data Visualisation




# Section 3: Methodology

### Data Pre-processing

The different models built as part of the project required different pre-processing steps.

The first model built as part of the guidance provided by Udacity was using the cv2 Haar Cascade classifier to identify human faces. This required first converting images to grayscale before inputting them into the model.

The second model was using the ResNet50 model to detect dogs in the images. This CNN required the input image path to be converted to a 4d array containing the number of images in the sample, the dimensions of the picture, and the number of channels (I.e. colour scales) represented in the image. For that purpose, the notebook contained the pre-defined path_to_tensor and paths_to_tensor functions. Path_to_tensor loads and resizes the image to 244x244 pixels, and the

converts it to first an error and then into a 4D tensor. Paths_to_tensor then uses this function to assemble the input tensor. The final steps, specific to ResNet50, then included re-ordering the channels from RGB to BGR and normalising the pixel values by subtracting the mean pixel values across the sample from each channel value.

For the third model, which was a CNN built from scratch, the pre-processing was once more already provided in the notebook and used the same paths_to_tensor function as explained above, while also dividing all pixels by 255.

The final model used pre-trained bottleneck features from ReNet which had to be downloaded and extracted as training inputs. Any inputs for making predictions were put through the above-described steps of the paths_to_tensor function and a pre-defined extract_ResNet50 function.

## Implementation

For this section I will focus on the 2 models I constructed myself: the CNN from scratch and the final model which used transfer learning.

**Model from Scratch**

For the first of these, the aim was to correctly classify dog breeds with a CNN built from scratch, and to achieve an accuracy score of at least 1%. The first step was to set up the model architecture. Consulting Aurelien Geron's book Hands-on Machine Learning with Scikit-Learn, Keras, and Tensorflow (2019), I followed some key principles in setting up the CNN. Firstly, I used relu activation function for all but the final layer. I started 32 filters but doubled that number after each Pooling layer. My original intention was to use a MaxPooling2D followed by a flattening layer before adding some dense layers at the end. This however, kept the number of trainable parameters too large, so I replaced the flattening layer with a Global Average Pooling layer. I finally added a final softmax dense layer to give the desired classification output.

This model was then compiled, trained, and evaluated using pre-coded cells.

**Model for final algorithm**

The second major task was to create an algorithm that would classify user-input images with an accuracy of at least 60%. I decided that the algorithm should give one of three outputs:

'The dog in the picture is a ...!' , if a dog is recognised in the image.

"That's not a dog! Though it does look a bit like a {lookalike}.", if a human face was recognised in the image.

"Sorry, we could find neither dog nor human in that picture. Try a different one!" if neither could be detected.

I therefore used the functions provided at the start of the notebook to detect whether the user input image is a dog or a human. In either case I would then apply a function which transforms the input image and uses a transfer learning model to make a classification.

The first step to setting up that model was to select one of several networks and extract the pre-trained bottleneck features from that network. I chose ResNet-50, for the simple reason that the provided dog-detection algorithm at the start of the project was based on ResNet-50 and performed perfectly as correctly identifying dogs with no false positives from the human images.

As the bottleneck features are essentially the output of the convolutional network, all that remained to do for the model was to add a Global pooling layer or flattening layer as the input layer, as well as one or more dense layers with dropout layers in-between. As it worked really well in the model which was built from scratch, I chose a Global Average pooling layer as the input, and then started with a single dense relu layer before the output layer which was a softmax dense layer.

I set up the model compilation and training following the examples provided earlier on in the project, starting with 20 epochs of training time.

### Refinement

**Model from Scratch**

After having set up the initial architecture, I tried multiple alternative architectures: First I added another dense layer, but as the resulting model was overtraining, I added a dropout layer between the two tense layers. Training the model for 10 rather than just 5 epochs brought a further improvement. Other improvement attempts such as increasing the number of filters across the convolutional layers, increasing the number of convolutional layers, and increasing the training periods even further were unsuccessful.

**Model for final algorithm**

As a first step to further improve the original model, was reducing training time by reducing it to 10 epochs, which was achieved at no loss of test accuracy. I also attempted adding more dense layers and different numbers of nodes. None of these brought significant improvements, with slight gains being made by having one dense layer with a dropout layer in between the input and output layers.

## Section 4: Results

### Model Evaluation and Validation

**Model from Scratch**

Though the test accuracy varied quite strongly with each time training the model, in peak test accuracy scores of 8.73% were achieved, vastly outperforming the threshold of 1% test accuracy required for this model. This is also a big improvement on the 4% test accuracy achieved with the initial model architecture. This suggests, that adding more dense layers made a big difference in terms of utilising the information output by the convolutional layers. It is noteworthy that the model was not over, but rather underfitting with the high validation accuracy in training being around 7%. The addition of dropout layers to counter over training was evidently successful.

**Model for final algorithm**

The test accuracy of the final model was consistently above 80%, reaching over 81% in peak. This is again significantly outperforming the 60% accuracy threshold required for this model. The improvement from the initial architecture however was negligible at only around 2% test accuracy. Unlike in the CNN built from scratch, the addition of extra dense layers did not bring about much improvement. It is evident that the main driver of model performance in this case are the bottleneck features used as input.

### Justification

| Model | Training Acc | Val Acc | Test Acc | Test Acc initial | Reequired Test Acc |
|---|---|---|---|---|---|
| From Scratch | 8.1% | 7.1% | 8.7% | 4% | 1% |
| Final | 85.6% | 82.5% | 81.25% | 79% | 60% |

As can be seen from the summary of the results from above, using the pre-trained features through transfer learning made a huge difference to the model performance. The impact of these features is so big that the set-up of the dense part of the has barely any influence on the model performance. The key therefore appears to be in the convolutional part of the model. With such well-developed convolutional networks available, using pre-trained features is definitely more time efficient and also more effective than constructing a model from scratch.

Of course, the actual task of the project was not only to build a model, but also to construct an algorithm around it which can take user images and provide their respective dog breed or lookalike as an output. To test this algorithm, I gave it 10 images which were either provided in the notebook or I accessed from iStock photos. Below is a summary of how the algorithm performed.

| Image | Image Content | Algorithm response |
|---|---|---|
| Brittany_02625 | Brittany Dog | Dog = Brittany |
| Husky | 2 Siberian Huskys | Dog = Alaskan Malamute |
| Corgi | Corgi Dog | Dog = Pembroke Welsh Corgi |
| Human | Human | Human = American Water Spaniel |
| Human 2 | Human | Human = Chihuahua |
| Human with Jack | Human + Dog | Human = Cavalier King Charles Spaniel |
| Human 3 | Human | Human = Bull Terrier |
| Landscape | Landscape | Neither |
| Cat | Cat | Neither |
| Whippet | Whippet | Italian Greyhound |

The algorithm classified 3 of the 5 dogs correctly. Though the image of the Huskies was returned for a search for Husky, they might very well be Alaskan Malamutes. Similarly, I could not rule out that the image of the Whippet is in fact an Italian greyhound. The algorithm also correctly identified all humans, though the lookalike it assigned changed each time the underlying model was retrained. It correctly threw an error when shown the picture of a landscape and a cat. In the image of the Human with the Jack Russel, the algorithm identified the human, but not the dog. This result is slightly surprising, as the human's face is partly hidden behind the dog's face. The set-up of this picture may have been an issue, as human and dog face are so close to each other that they might merge through the pooling and field sizes in the convolutional model. Giving the algorithm an image where human and dog are slightly further apart, correctly returned that the dog is a German Shepherd.

## Section 5: Conclusion

## Reflection

To summarise, this project was a particularly interesting experience as it dealt with a new area of data that I didn't have prior experience in. It was also great to not only built the model but implement an entire algorithm to use the model to make a prediction based on user input. I started by checking the training, testing, and validation data are well balanced. Then built a CNN from scratch and built a second CNN using transfer learning. This was then wrapped in an algorithm to process the user input, detect whether human or dog can be seen in the picture, and put out an appropriate response. The final algorithm performed very well, though there are some edge cases where it is difficult to judge whether the algorithm performed well or not.

All in all, it was a very instructive experience, and I learned a lot completing this project.

## Improvement

There are a number of ways in which both the algorithm and the model can be improved. For a start, the algorithm can be made more user friendly, for example by classifying both dog and human when they are in the picture, or by changing the way the breed is returned to separate, capitalised words.

For the model, different networks could be tried to see which one provides the best results. It could even be attempted to combine the results from two separate sets of bottleneck features.

The next logical step after these optimisations would be to integrate this solution into a web or mobile app.