

Pooch Detector: Image Classification with Convolutional Neural Networks

A report by Carlos Gavidia-Calderon

Definition

Project Overview

Background: In this project, we apply a supervised learning approach to an image classification problem. That is, given a training set of labelled images, our machine learning system will learn to assign correct labels to previously unseen images. Among the multiple applications of image classification, we can mention its use for automatic diagnosing of skin cancer, outperforming professionally dermatologists¹.

Dataset: The dataset is available at the Udacity GitHub repository² corresponding to the project proposal. It contains labelled RGB dog images.

Problem Statement

We build an image classifier for dog breeds. When an image is fed to the system, it outputs the most likely breed.

Strategy: Our solution is a convolutional neural network (CNN), an approach proven effective for image classification tasks. To increase performance, we adopt a transfer learning approach, relying on a pre-trained CNN for feature extraction.

Metrics

During training, we attempt to minimize **cross-entropy loss**, as is common in multiclass classification.

As seen in later sections, classes in our dataset are balanced. Hence, we rely on **accuracy** over the test dataset to evaluate system performance. Accuracy is defined as the percentage of unseen images correctly classified after training.

¹ Esteva, Andre, Brett Kuprel, Roberto A. Novoa, Justin Ko, Susan M. Swetter, Helen M. Blau, and Sebastian Thrun. "Dermatologist-level classification of skin cancer with deep neural networks." Nature 542, no. 7639 (2017): 115-118.

² Dataset available at <https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/dogImages.zip>

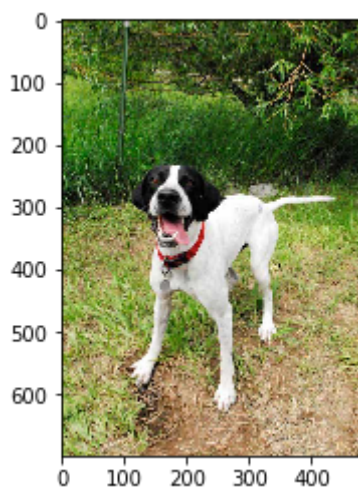
Analysis

Data Exploration

The whole dataset contains **8351 RGB images of various sizes**. Each of them have a label assigned, corresponding to a dog breed. There are **133 labels** in the dataset.

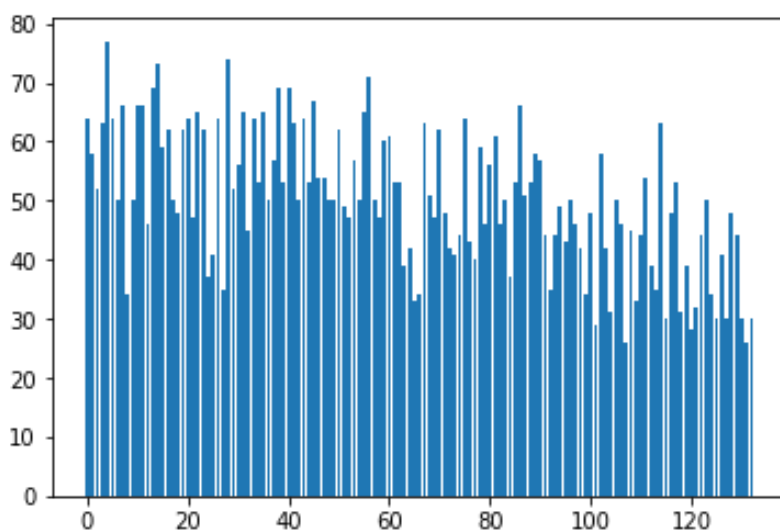
Exploratory Visualization

As reference, here's an image of a Pointer, taken from the validation dataset.



This image is 483×700 in size. However, size is not consistent over the whole dataset.

Here is a histogram of class frequency over the training dataset. We observe that **labels are evenly distributed**.



Algorithms and Techniques

Convolutional Neural Networks: Convolutional Neural Networks (CNN) take advantage of 2-dimensional input structure. This made them well suited for image processing tasks. Standard CNN architectures have the following components:

1. Convolutional layers, composed by multiple kernels. Their output is a series of feature maps.
2. Non-linear activation functions, to be applied to the feature maps.
3. Subsampling layers, using mean or max pooling.
4. Fully connected layers at the end of the network.

Transfer Learning: Training CNN effectively requires large amounts of data. When this is not possible, transfer learning approaches allow us to take advantage of pre-trained models that have already benefited from extensive training. There are two main transfer learning strategies:

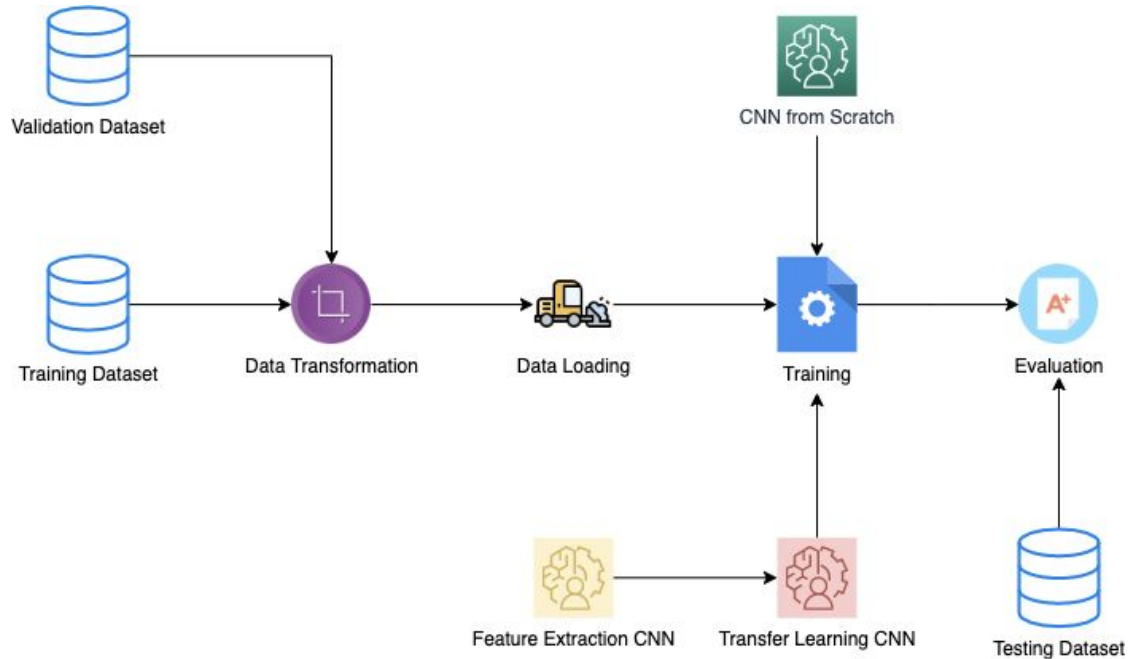
1. In a fixed feature extraction approach, we remove the last component of the base network and replace it with a custom set of layers. The training over the new dataset modifies only the weights of these new layers, without affecting the weights of the base network.
2. In a fine-tuning approach, we also replace the last component of the base network. However, training over the new dataset affects the weights of every layer of the network.

Benchmark

For comparison purposes, we also develop a convolutional neural network from scratch, instead of implementing feature extraction with transfer learning. This model will serve as reference, providing a lower-bound on the performance of our transfer learning approach.

Methodology

Our approach is illustrated in the following diagram:



Data Preprocessing

Images are fed to the training process in **batches of 4 images**. The data loading process includes the following image transformation steps:

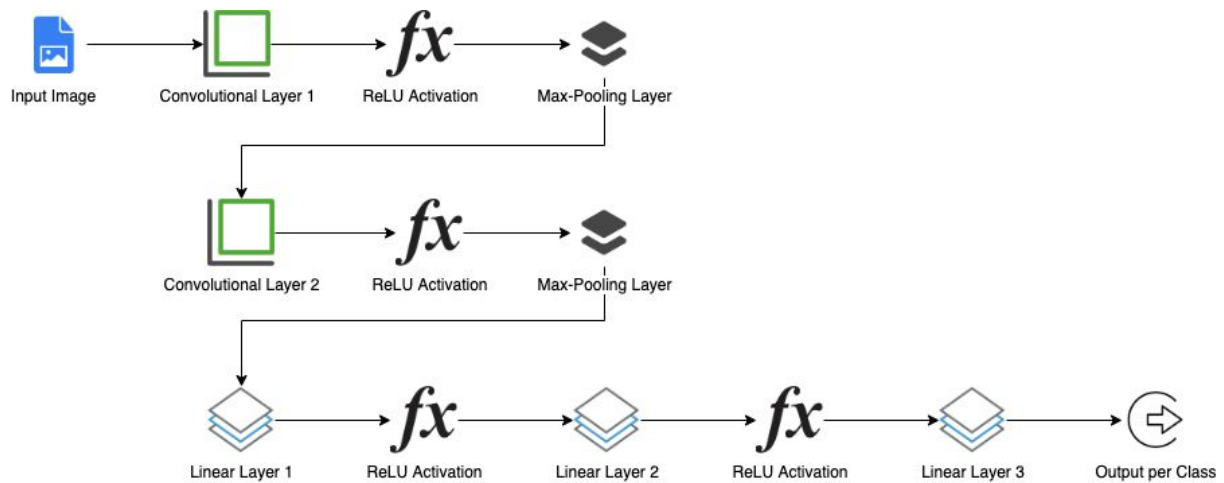
1. We extract a segment from the original image, with a random position and aspect ratio. The segments are of **size 224x224**, as required by our base network for transfer learning.
2. We randomly flip horizontally the extracted segment.
3. After converting the segment to a numerical tensor **we normalize it with means (0.485, 0.456, 0.406) and standard deviations (0.229, 0.224, 0.225)**. We have one value for each channel of the RGB images. Again, this is required by our base network for transfer learning.

For training and validation, the data loading process includes the following transformations:

4. We extract segments from the original image, from the center. The segments are of size 224x224.
5. After converting the segment to a numerical tensor we normalize it with means (0.485, 0.456, 0.406) and standard deviations (0.229, 0.224, 0.225).

Implementation

We started by implementing a convolutional neural network from scratch. Our initial design had the following architecture:



The following table contains each layer's parameters:

Component	Parameter Name	Parameter Value
Convolutional Layer 1	In channels	3
	Out channels	6
	Kernel size	5
Max-Pooling Layer	Kernel size	2
	Stride	2
Convolutional Layer 2	In channels	6
	Out channels	16
	Kernel size	5
Linear Layer 1	In features	44944
	Out features	400
Linear Layer 2	In features	400
	Out features	120
Linear Layer 3	In features	120
	Out features	133

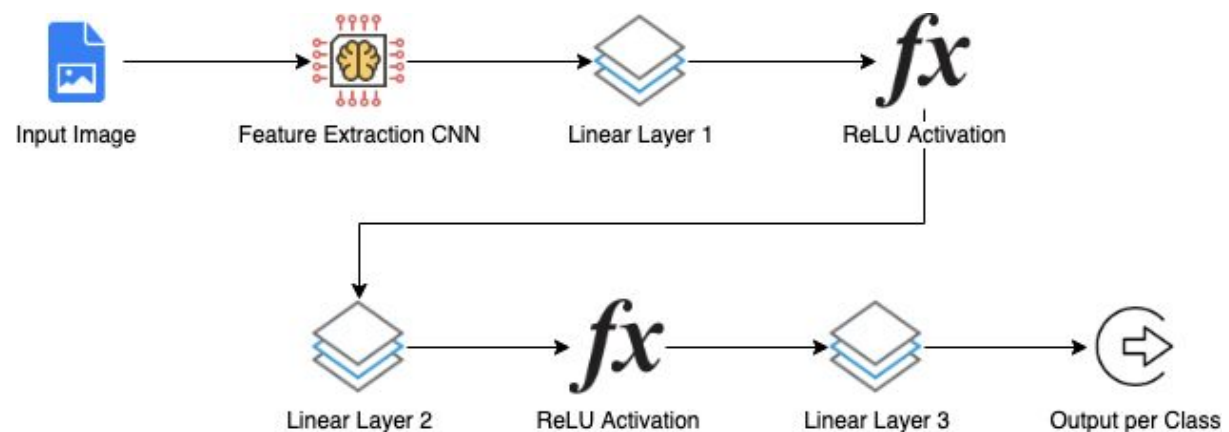
The network has two convolutional layers for feature extraction. The first convolutional layer has 6 output channels and the second one 16, following the advice of increasing the number of output channels on deeper layers of the network. Each convolutional layer is followed by a max-pooling one, to reduce dimensionality between layers. The last part of the network is composed of three fully-connected layers. The last layer outputs 133 values corresponding to each class to predict. The first two fully-connected layers are followed by ReLU activations.

For training, we adopted **cross-entropy loss** as our loss function and **stochastic-gradient-descent** as our optimisation algorithm (with a **learning rate** of 0.001 and a **momentum** of 0.9).

After training for **50 epochs**, our **loss over the training dataset** is 3.473697 and over the validation set is 3.882037. When applying our network to the test dataset, our loss is 3.896492, with an overall **accuracy** of 13%.

Refinement

In the previous section, we saw that the CNN from scratch had poor performance. Hence, we moved to a transfer learning approach. We adopt the VGG-16 model³ for feature extraction, given its effectiveness over the ImageNet dataset. Our new design has the following architecture:



The following table contains each layer's parameters:

³ Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." *arXiv preprint arXiv:1409.1556* (2014).

Component	Parameter Name	Parameter Value
Linear Layer 1	In features	25088
	Out features	4096
Linear Layer 2	In features	4096
	Out features	1000
Linear Layer 3	In features	1000
	Out features	133

The “Feature Extraction CNN” element in the diagram represents the pre-trained version of VGG16. To use it as a fixed feature extractor, we replaced its classifier component with three fully connected layers (with ReLU activations). During training, we only update the parameters of our new classifier layer.

We used VGG16 as a base model since, as seen in the Jupyter Notebook, this network proves effective in identifying dogs. We believe that using its extracted features, plus some additional training, can successfully identify dog breeds.

Results

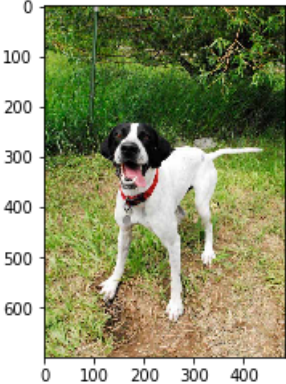

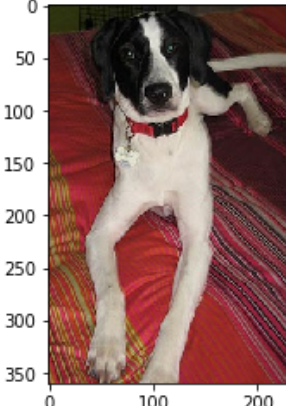

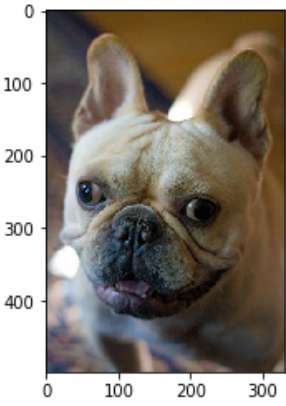

Model Evaluation and Validation

As with the CNN from scratch, for training we adopted **cross-entropy loss** as our loss function and **stochastic-gradient-descent** as our optimisation algorithm (with a **learning rate** of 0.001 and a **momentum** of 0.9).

After training for **50 epochs**, our **loss over the training dataset** is 0.794249 and over the validation set is 1.221666. When applying our network to the test dataset, our loss is 1.292455, with an overall **accuracy** of 66%.

Justification

Although the transfer learning approach performs significantly better than the CNN from scratch, the model is far from perfect. These are some examples of classification results:

Input Image	Correct Breed	Predicted Breed	Predicted Reference
	Pointer	<i>Parson russell terrier</i>	
	Pointer	<u>Pointer</u>	
	French Bulldog	<u>French Bulldog</u>	

Some of the input pictures contain dogs and humans, and that makes the algorithm behave in unexpected ways. As future work, we can pre-process the input to extract dog/human faces before feeding them to the algorithm. Also, we did not tune the parameters of the network (learning rate, number of layers, number of epochs), relying only on fixed values. Trying different configurations can potentially improve the algorithm's accuracy.