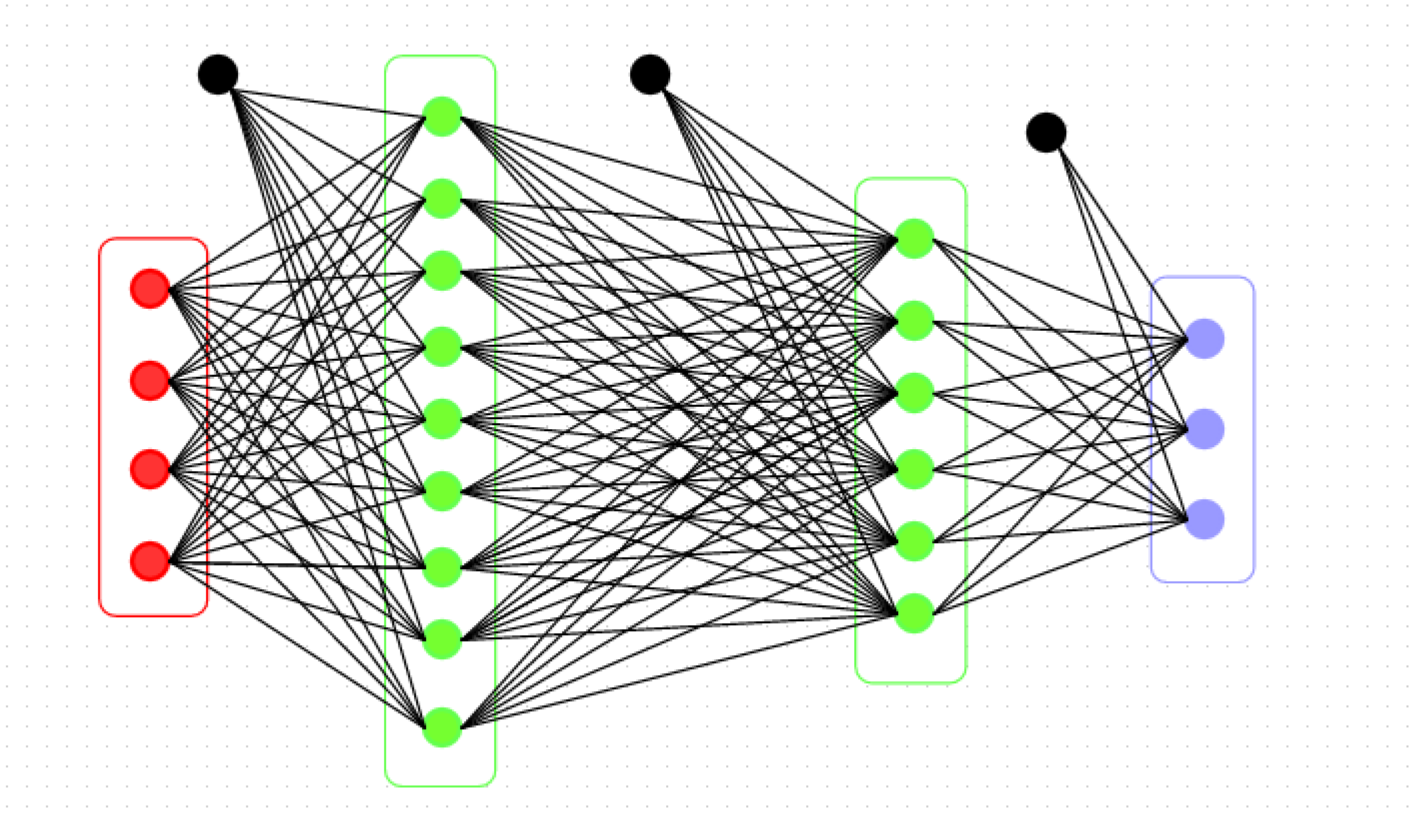
Built My Deep Learning Classifier using TensorFlow: Dog Breed Example



Convoluted Neural Networks (like the one pictured above) are powerful tools for Image Classification

In this article, I will present several techniques for you to make your first steps towards developing an algorithm that could be used for a classic **image classification** problem: detecting dog breed from an image.

By the end of this article, we'll have developed code that will accept any user-supplied image as input and return an estimate of the dog's breed. Also, if a human is detected, the algorithm will provide an estimate of the dog breed that is most resembling.

N.B. This project was completed as part of **Udacity**'s Machine Learning Nanodegree (GitHub repo).

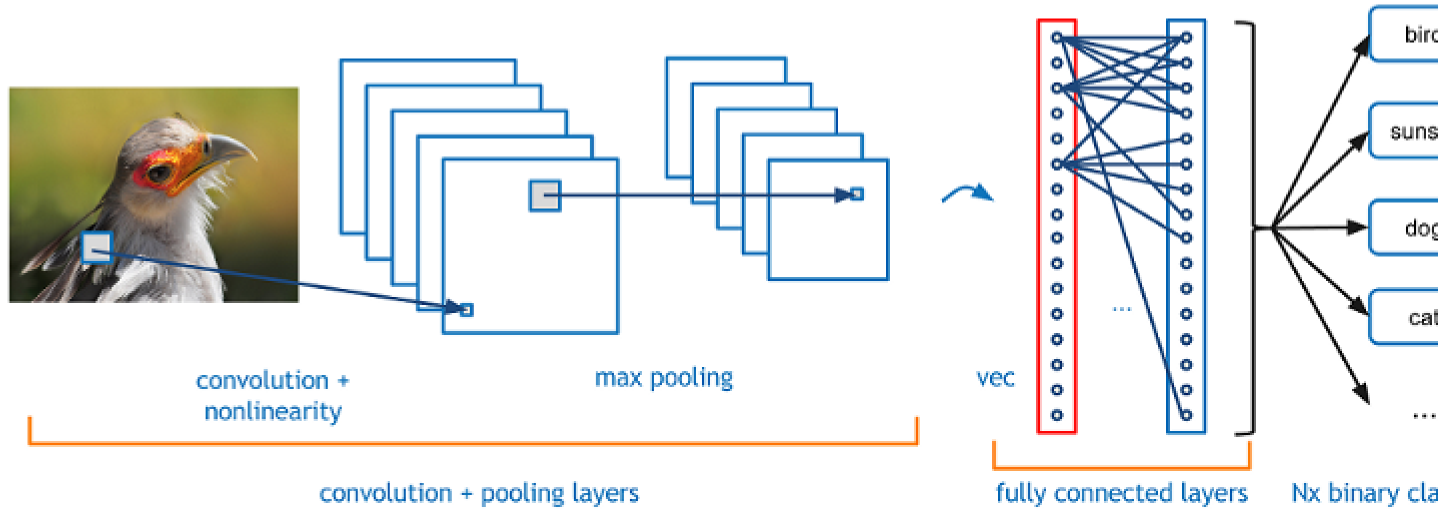
1. What are Convolutional Neural Networks?

Convolutional neural networks (also refered to as CNN or ConvNet) are a class of **deep neural networks** that have seen widespread adoption in a number of computer vision and visual imagery applications.

A famous case of CNN application was detailed in this research paper by a Stanford research team in which they demonstrated classification of skin lesions using a single CNN. The Neural Network was trained from images using only pixels and disease labels as inputs.

Convolutional Neural Networks consist of multiple layers designed to require relatively little pre-processing compared to other image classification algorithms.

They learn by using filters and applying them to the images. The algorithm takes a small square (or 'window') and starts applying it over the image. Each filter allows the CNN to identify certain patterns in the image. The CNN looks for parts of the image where a filter matches the contents of the image.

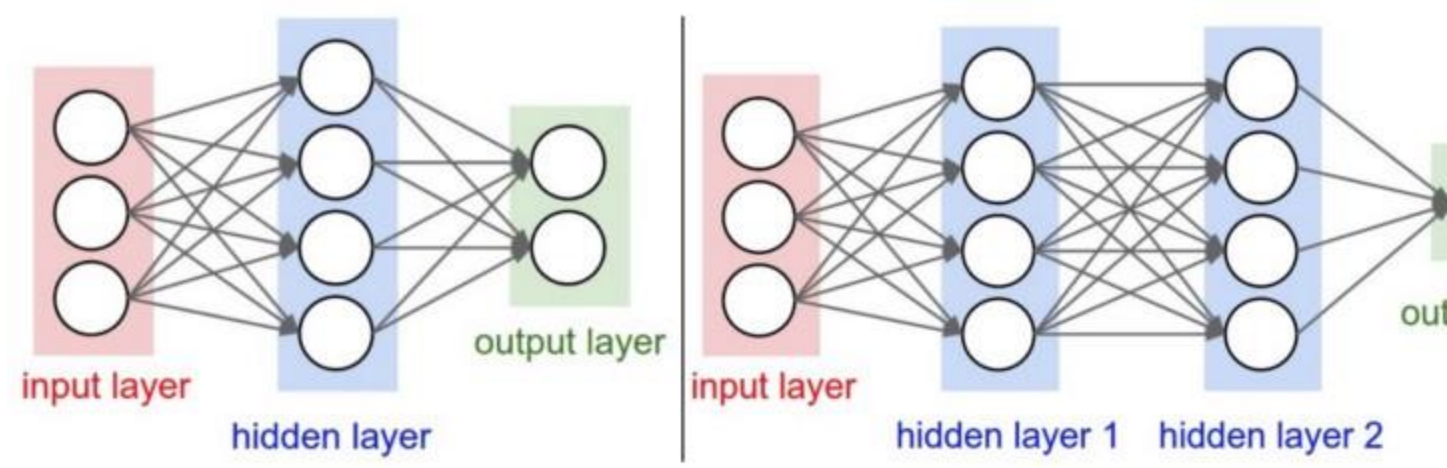


An example of a CNN Layer Architecture for Image Classification (source: https://bit.ly/2vwlegO)

The first few layers of the network may detect simple features like lines, circles, edges. In each layer, the network is able to combine these findings and continually learn more complex concepts as we go deeper and deeper into the layers of the Neural Network.

1.1 What kinds of layers are there?

The overall architecture of a CNN consists of an input layer, hidden layer(s), and an output layer. They are several types of layers, for e.g. Convolutional, Activation, Pooling, Dropout, Dense, and SoftMax layer.

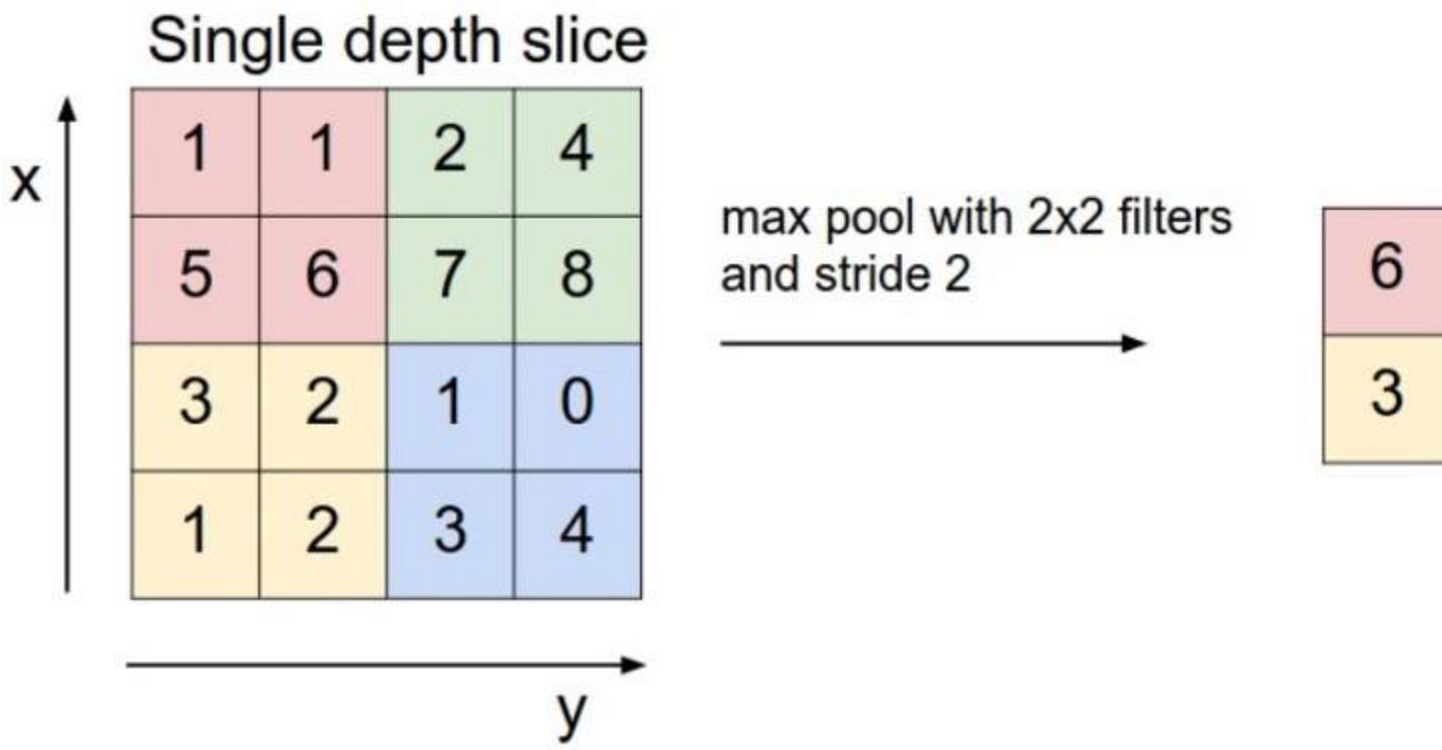


Neural Networks consist of an input layer, hidden layers, and an output layer (source: https://bit.ly/2Hxhjaw)

The Convolutional Layer (or Conv layer) is at the core of what makes a Convolutional Neural Network. The Conv layer consists of a set of filters. Every filter can be considered as a small square (with a fixed width and height) which extends through the full depth of the input volume.

During each pass, the filter 'convolves 'across the width and height of the input volume. This process results in a 2-dimensional activation map that gives the responses of that filter at every spatial position.

To avoid over-fitting, Pooling layers are used to apply non-linear downsampling on activation maps. In other words, Pooling Layers are aggressive at discarding information but can be useful if used appropriately. A Pooling layer would often follow one or two Conv Layers in CNN architecture.



Pooling layers are used to apply non-linear downsampling on activation maps (source: https://bit.ly/2Hxhjaw)

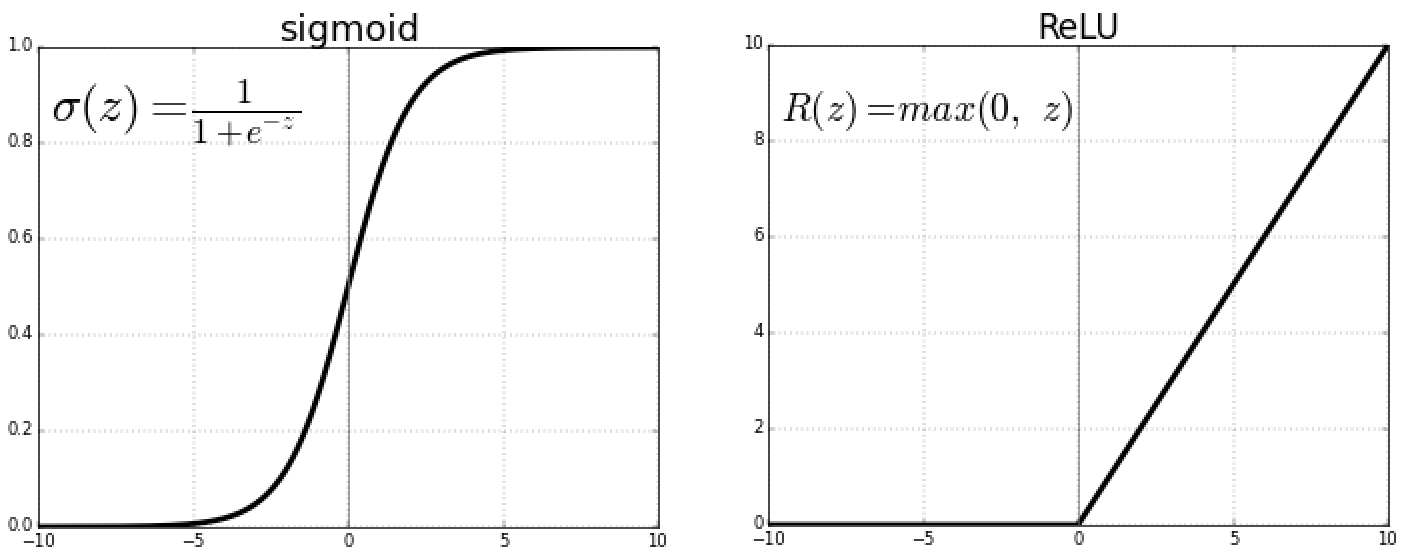
Dropout Layers are also used to reduce over-fitting, by randomly ignore certain activations functions, while Dense Layers are fully connected layers and often come at the end of the Neural Network.

1.2 What are Activation Functions?

The output of the layers and of the neural network are processed using an activation function, which is a node that is added to the hideen layers and to the output layer.

You'll often find that the ReLu activation function is used in hidden layers, while the final layer typically consists of a SoftMax activation function. The idea is that by stacking layers of linear and non-linear functions, we can detect a large range of patters and accurately predict a label for a given image.

SoftMax is often found in the final layer which acts as basically a normalizer and produces a discrete probability distribution vector, which is great for us as the CNN's output we want is a probability that an image corresponds to a particular class.



The most common activation functions include the ReLU and Sigmoid activation functions

When it comes to model evaluation and performance assessment, a loss function is chosen. In CNNs for image classification, the categorial cross-entropy is often chosen (in a nutshell: it corresponds to -log(error)). There are several methods to minimise the error using Gradient Descent—in this article, we'll rely on "rmsprop", which adaptive learning rate method, as an optimizer with accuracy as a metric.

2. Setting up the algorithm's building blocks

To build our algorithm, we'll be using TensorFlow, Keras (neural networks API running on top of TensorFlow), and OpenCV (computer vision library).

Training and testing datasets were also available on-hand when completing this project (see GitHub repo).

2.1 Detecting if Image Contains a Human Face

To detect whether the image supplied is a human face, we'll use one of OpenCV's Face Detection algorithm. Before using any of the face detectors, it is standard procedure to convert the images to grayscale. Below, the `detectMultiScale` function executes the classifier stored in `face_cascade` and takes the grayscale image as a parameter.

2.2 Detecting if Image Contains a Dog

To detect whether the image supplied contains a face of a dog, we'll use a pre-trained ResNet-50 model using the ImageNet dataset which can classify an object from one of 1000 categories. Given an image, this pre-trained ResNet-50 model returns a prediction for the object that is contained in the image.

When using TensorFlow as backend, Keras CNNs require a 4D array as input. The `path_to_tensor` function below takes a string-valued file path to a color image as input, resizes it to a square image that is 224x224 pixels, and returns a 4D array (referred to as a ‘tensor’) suitable for supplying to a Keras CNN.

Also, all pre-trained models have the additional normalization step that the mean pixel must be subtracted from every pixel in each image. This is implemented in the imported function `preprocess_input`.

As shown in the code above, for the final prediction we obtain an integer corresponding to the model’s predicted object class by taking the *argmax* of the predicted probability vector, which we can identify with an object category through the use of the ImageNet labels dictionary.

3. Build your CNN Classifier using Transfer Learning

Now that we have functions for detecting humans and dogs in images, we need a way to predict breed from images. In this section, we will create a CNN that classifies dog breeds.

To reduce training time without sacrificing accuracy, we’ll be training a CNN using Transfer Learning—which is a method that allows us to use Networks that have been pre-trained on a large dataset. By keeping the early layers and only training newly added layers, we are able to tap into the knowledge gained by the pre-trained algorithm and use it for our application.

Keras includes several pre-trained deep learning models that can be used for prediction, feature extraction, and fine-tuning.

3.1 Model Architecture

As previously mentioned, the ResNet-50 model output is going to be our input layer—called the *bottleneck* features. In the code block below, we extract the bottleneck features corresponding to the train, test, and validation sets by running the following.

We’ll set up our model architecture such that the last convolutional output of ResNet-50 is fed as input to our model. We only add a Global Average Pooling layer and a Fully Connected layer, where the latter contains one node for each dog category and has a Softmax activation function.

```
_________________________________________________________________ Layer
(type) Output Shape Param #
=================================================================
global_average_pooling2d_3 ( (None, 2048) 0
_________________________________________________________________ dense_3
(Dense) (None, 133) 272517
================================================================= Total
params: 272,517 Trainable params: 272,517 Non-trainable params: 0
_________________________________________________________________
```

As we can see in the above code’s output, we end up with a Neural Network with 272,517 parameters!

3.2 Compile & Test the Model

Now, we can use the CNN to test how well it identifies breed within our test dataset of dog images. To fine-tune the model, we go through 20 iterations (or 'epochs') in which the model's hyper-parameters are fine-tuned to reduce the loss function (categorial cross-entropy) which is optimised using RMS Prop.

```
Test accuracy: 80.0239%
```

Provided with a testing set, the algorithm scored a testing accuracy of **80%**. Not bad at all!

3.3 Predict Dog Breed with the Model

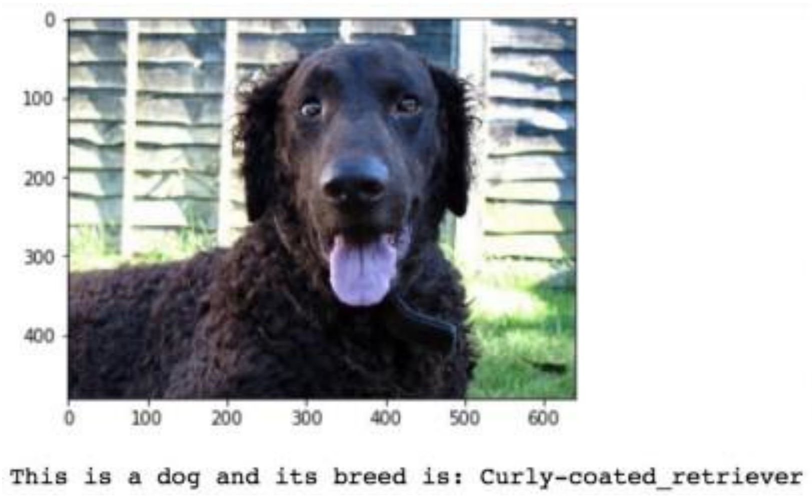
Now that we have the algorithm, let's write a function that takes an image path as input and returns the dog breed that is predicted by our model.

4. Testing our CNN Classifier

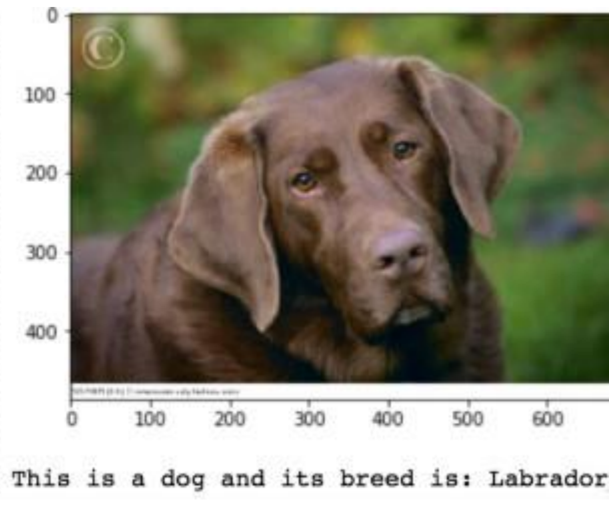
Now, we can write a function that takes accepts a file path to an image and first determines whether the image contains a human, dog, or neither.

If a **dog** is detected in the image, return the predicted breed. If a **human** is detected in the image, return the resembling dog breed. If **neither** is detected in the image, provide output that indicates an error.

We are ready to take the algorithm for a spin! Let's test the algorithm on a few sample images:



```
This is a dog and its breed is: Curly-coated_retriever
```



```
This is a dog and its breed is: Labrador
```

These predictions look accurate to me!

On a final note, I noted that that the algorithm is prone to errors unless it's a clear facing shot with minimal noise on the image. Hence, we need to make the algorithm more robust to noise.

Also, a method we can use to improve our classifier is image augmentation which allows you to "augment" your data by providing variations of the images supplied in the training set.