Capstone Project
Machine Learning Nanodegree

Fernando Favoretti Vital do Prado
March, 2017

Introduction

Recognition of images are easily made by our brains, we are fast and certain to differentiate a car from a animal from another person. But for computers these are actually hard problems to solve[1].

In the last few years the field of machine learning has made tremendous progress on addressing these kind of image recognition problems. The new techniques that arrives are a great innovation like TensorFlow, Convolutional networks, GPU(Graphics Processing Unit) clusters and others.

This project has the objective to apply a Convolutional Neural Network (CNN) for the field of image recognition, basically our algorithm takes a image as an input and generate probabilities for the classes that we are trying to observe.

Definition

Project Overview

A famous competition in Kaggle has come back and as the computers are getting better to understanding images due high advances in Machine Learning techniques like deep learning and tensor flow, tasks like these are becoming more possible and more precise.

This work will make an approach to the Dogs Vs Cat classification problem hosted by Kaggle (https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition) and have the unique objetive of distinguish images of dogs from cats.

Problem Statement

The goal is to create a classificator that can correctly separate images that have a Dog in it from images that have a Cat in it, the task involved are the following

1. Download and preprocess the Data for the problem.
2. Training a Classifier using Deep Learning Techniques.
3. Test the classifier and see the probabilities that the our CNN have generated.

The algorithm should work like that:

We will input an image with a Dog or a Cat for it and we should receive an answer like:
“I’m 96% sure that this image is of a Dog!”.
The operation of this algorithm will be explained in the sections below.

For the final part of the project, after we have tested the classifier with our metrics, we also will make a submission on the Kaggle Platform just to look how good (or not) our CNN is.

Metrics

The goal of the training phase for a image recognition algorithm is learn the network weights, for that we need two states[11]:

Number One: A good train set, already normalized with the corresponding labels, we will do this in the preprocessing step

Number Two: A metric used to learn the weights and a metric to measure the accuracy of the predictions, and this is what we gonna see in this section.

For each step in our training phase we will use the Categorial Cross-Entropy Loss function to accomplish the adjustment of the weights. This function Computes cross entropy between y_pred (logits) and y_true (labels) and is used to measures the probability error in discrete classification tasks in which the classes are mutually exclusive (each entry is in exactly one class) and try to minimize it. The Cross Entropy indicates the distance between what the network believes this distribution should be and what the real result says it should be[3].

For measure the accuracy of the model we will use the simple accuracy function, which is a common and good metric for binary classifiers, the accuracy for our model can be computed like that:

$$Accuracy = \frac{Num.\ of\ dogs\ correctly\ predicted + Num.\ of\ cats\ correctly\ predicted}{Complete\ size\ of\ the\ dataset}$$

Using that, in the test phase we will have some result that are easy to understand like: “The network correctly predict 90% of the images”

Analysis

Data Exploration

The data that kaggle is making available contains 25,000 images of dogs and cats, and each image in the folder has the correct label as part of the the filename. The test folder contains 12,500 images named by a numeric Id. The images are all colored but all the images in the Train and Test dataset come with different sizes, shapes, resolutions and lighting as we can see bellow:

Fig 1: Images from the Kaggle Dog vs Cat Dataset

The dataset is not fully cleaned at all, images that are neither Dogs nor cats exists, so when I found something like that I just remove it from the set, because if the sample do not belong to any cases that I'm trying to classify there is no reason for keep it, the wrong image will only impair my model's ability to classify. Some example are:

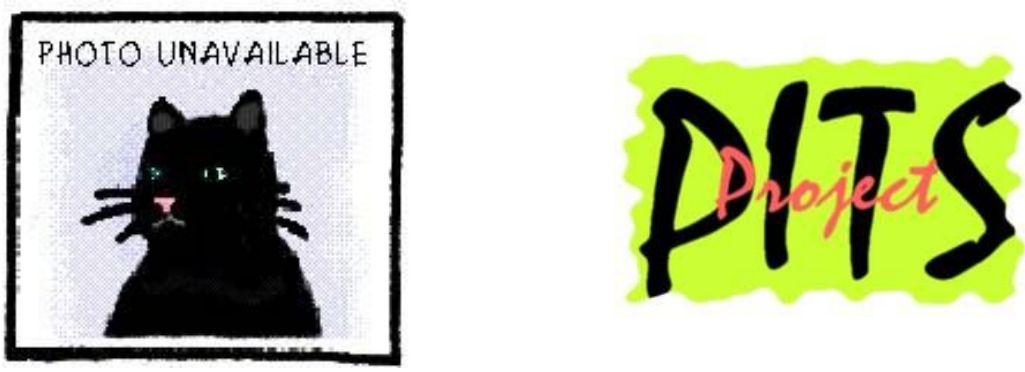


Fig 2: Images from the Kaggle Dog vs Cat Dataset that which are neither a Dog nor a Cat. (Or maybe the first can be a cat)

On the figures above, the two figures have the 'dog' label, so they will obviously skew up my dataset, when I found this kind of images I just delete them, but if for some reason I let some of them still on dataset, the proportion of these totally wrong images is so small that I believe this will not make a huge difference for the final results.

Exploratory Visualization

Looking for visualizing some important features of the images in dataset. The table below shows how the dimensions of the images are distributed among the dataset. This will be helpful for some insights of pre process images after.

	Widths
count	25000.00000
mean	404.09904
std	109.03793
min	42.00000
25%	323.00000
50%	447.00000
75%	499.00000
max	1050.00000

	Heights
count	25000.000000
mean	360.478080
std	97.019959
min	32.000000
25%	301.000000
50%	374.000000
75%	421.000000
max	768.000000

The Tables above shows the describe command used in a sample of 25.000 images (looking for the Widths and the Heights only) we can see that the image size can change a lot, and that is not great for the method that I'll use, so later, a work that normalize the resolution and other aspects of the images will be necessary.

Algorithms And Techniques

Tensors and Matrix

Everybody is familiar with matrices, in this work we will need to represent tensors, what are high order matrix like three dimensional matrices or other.

Day-a-day we have interacting with tensors, the images that we are using in this project is an order 3 tensor with the channel RGB,i.e, an image with **R** rows and **C** columns is a tensor with the size R x C x 3 (if the image is stored in the RGB format).In other words a RGB image has 3 channels and each channel has a number of rows R and number of columns C what is a RxC matrix (or a second order tensor) that contains the R or G or B value of all pixel.[4]

In early computer vision paradigms working with images were so hard, so the images are usually converted to a gray scale color, transforming this into a simple matrix, and all the color information was lost in this process. But color is very important in some recognition tasks and we want to keep it.

TensorFlow

TensorFlow is an open source software library for numerical computation using data flow graphs and high parallelism computation. TensorFlow allows for really fast iteration of custom machine learning models. It also comes with a ton of useful functions that I'm going to use in this project. One of these features is the TFLearn Library: "TFlearn is a modular and transparent deep learning library built on top of Tensorflow. It was designed to provide a higher-level API to TensorFlow in order to facilitate and speed-up experimentations, while remaining fully transparent and compatible with it." The Tflearn features are easy to use, fast and high level[4].

Convolutional Neural Networks

Convolutional networks (CNN or ConvNets) have recently enjoyed a great success in large-scale image recognition, which has become possible due the easy access to high-performance computing systems such as GPUs or other types of distributed clusters.

A CNN generally takes a order 3 tensor as a input (e.g., an image with H rows, W columns, and 3 channels (R, G, B color channels), but high order tensor also can be inputted. After, the input go sequentially in a serie of processing steps (Layers). There are different types of layers, each one with a different function, but in simple words, a input is given to the first layers, that process and give the output to the next layer… and this goes until the end of the entire network, when the training is made we just need to run our network forward, input something into it and see the result.

Using the Dog vs Cat problem as an example, basically, for training a CNN we input the image, it goes for all the layers, generate a result, we compare it to the expected result and use a function to compute the loss and we have to minimize this loss adjusting the weights looking in a backward execution of the networks.

For predicting we input an never saw image of a dog/cat into the network, the first layers process and pass the result for the next layer and so on, final for each possible output (two for this problem) we will receive the probability of the inputted image belong to each class, so what we need to do is just take the high probability predicted.

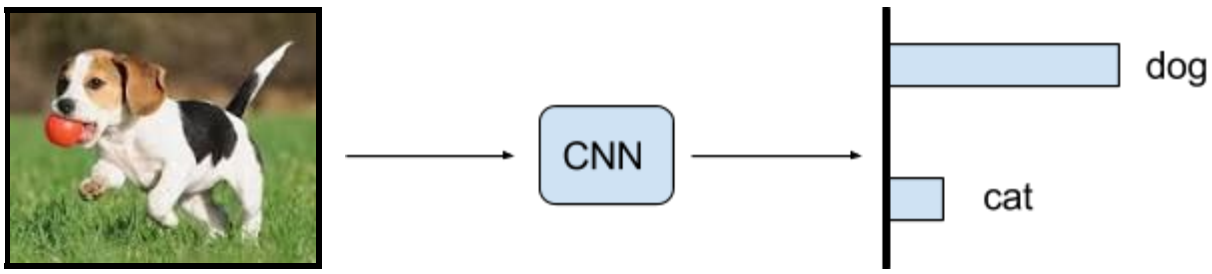


Fig 5: Simplified image recognition path

As said above, there are several types of CNN layers, some of them will be explained below:

ReLU layer

The Rectified Linear Unit, this layer does not affect the size of the input and there's no parameter need inside this layer.

The ReLU is a nonlinear function, simpler than the sigmoid function and is used to get a more fast training for the network, ReLU is the max function(x,0) with input x e.g. matrix from a convolved image. ReLU then sets all negative values in the matrix x to zero and all other values are kept constant.[6]

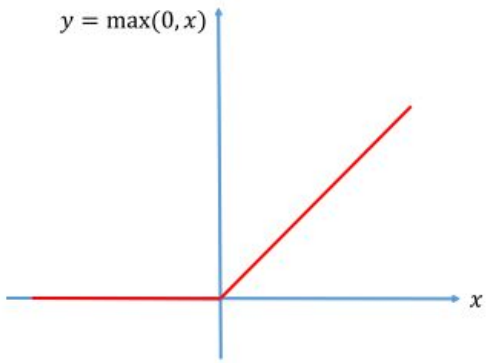


Fig 6: The ReLU function

Convolution Layer

This layer consists of a set of learnable filters that we slide over the image spatially, computing dot products between the entries of the filter and the input image. These filters are predefined and the coefficients are computing during the training phase. Convolve can make that the deepest layers can learn to active for different complex patterns in images, e.g., groups of edges in a particular shape, and this particularly objects activation can be linked to semantically meaningful object parts or even a particular type of object[6].

The pre process taking on the steps above in this same document like the histogram equalization (on figure 4) and so other techniques like blur, horizontal edge and vertical edge are all convolutions.

Pooling layer

The goal of the pooling layer is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting. There are several functions that can be implemented for pooling but the most common is the max pooling one that partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, outputs the maximum[5]

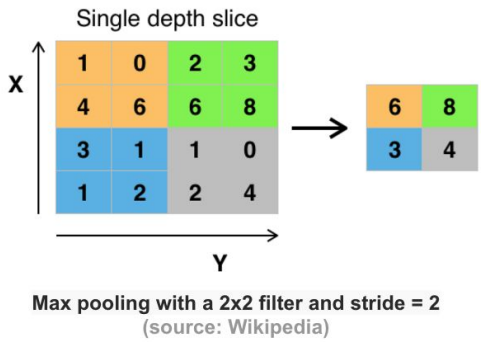


Fig 7: Max pooling function in action

Also it is common to periodically insert a pooling layer between successive convolutional layers in a CNN architecture. The pooling operation provides a form of translation invariance.

Fully Connected Layer

The term “Fully Connected” implies that every neuron in the previous layer is connected to every neuron on the next layer. It will reduce the results outputted by the previous layers to the size of classes that the CNN are trained to predict.

Dropout Layer

Dropout is a technique that prevents overfitting and provides a way of approximately combining exponentially many different neural network architectures efficiently. The term “dropout” refers to dropping out units (hidden and visible) in a neural network. By dropping a unit out, we mean temporarily removing it from the network, along with all its incoming and outgoing connections. The choice of which units to drop is random, in the simplest case each unit has a probability p of being dropped. Dropout is used only in the Train phase[12].

Classification Layer

The final layer that outputs the result of the FC layer to the probabilities of each object being a certain class.

Methodology

As sad in the Exploratory Visualization section, our data need to be preprocessed. In this section, we will see all the steps to perform the appropriate preprocessing.

Adjusting images contrast

An image histogram is a graphical representation of the intensity distribution of an image, basically it quantifies the number of pixels for each intensity value considered. Below we can see an example of that:

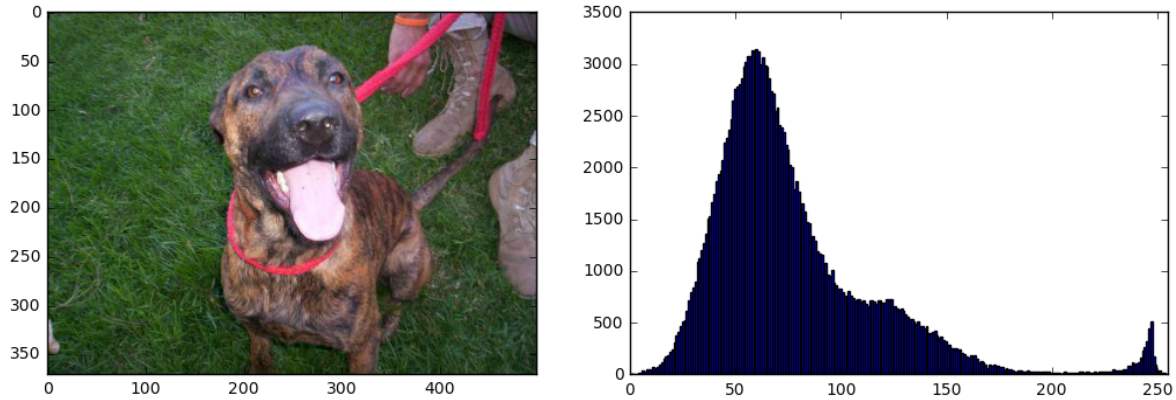


Fig 3: In the left a random image of the dataset, in the right the image histogram.

For adjusting the image contrast we're gonna use the Histogram equalization method[7]. This method is used to make a better use of the available brightness range, thus, increase the global contrast of an image and make that the intensities are better distributed in an histogram.

Equalization implies mapping one distribution (the given histogram) to another distribution (a wider and more uniform distribution of intensity values) so the intensity values are spreaded over the whole range.

To accomplish the equalization effect, the remapping should be the cumulative distribution function:

$$H'(i) = \sum_{0 \le j < i} H(j)$$

To use this function we also need to normalize H'(i) such that the maximum value is 255 or the maximum value of the intensity on the image. Finally, for remapping:

$$equalized(x,y) = H'(src(x,y))$$

And the result:

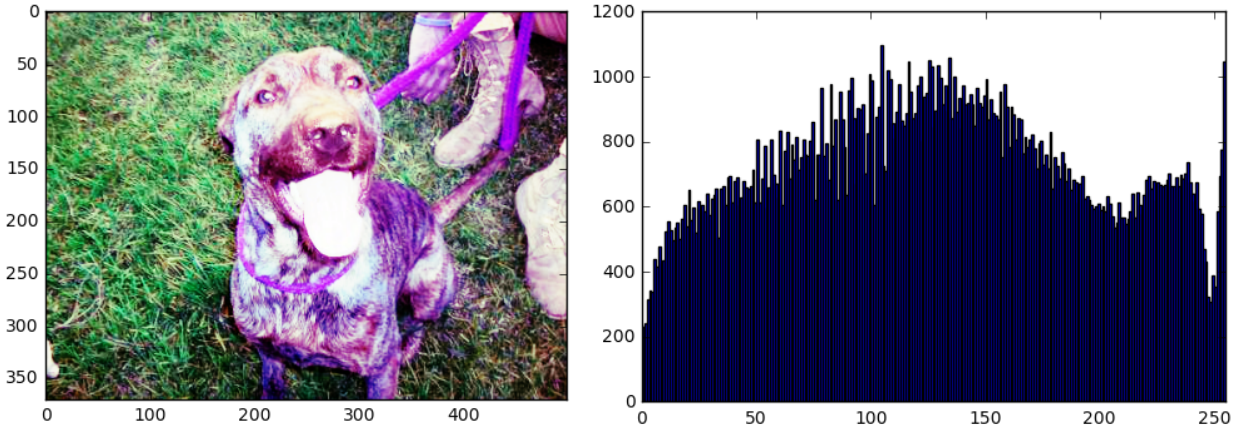


Fig 4: In the left, the same image with the histogram equalization method, on the right the image histogram.

Adjusting images sizes

Now that our images are all equalized the next step that we need is resize the images, rescaling the images is necessary because they vary in size a lot, as we can see in the first two tables, the smallest image have 42x32 pixels and the largest image have 1050x768 pixels, so, we'll resize all the picture to 64x64 pixels to reduce computational demands. But downsizing sacrifices information; larger images have more detail.and afford larger filters.

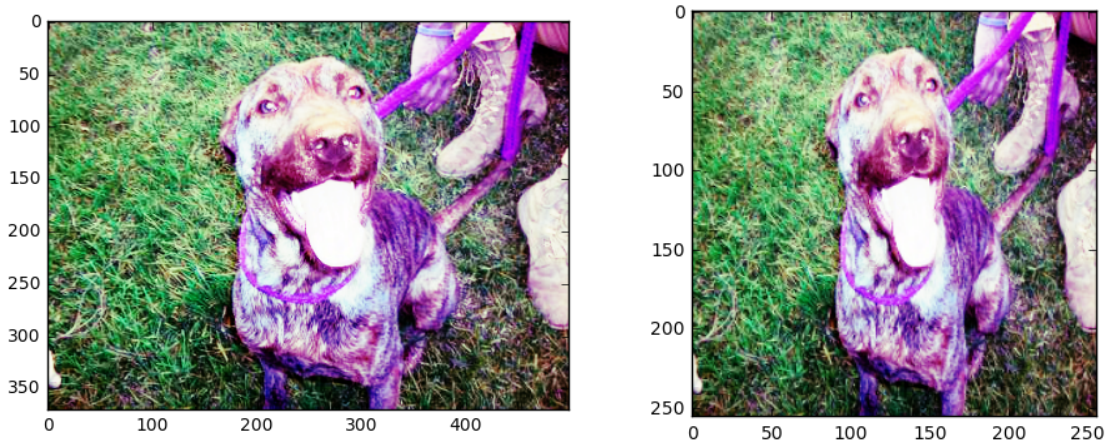


Fig 4: Image on the left before the resizing, image on the right after the resizing

We also normalized the mean and the standard deviation (using the Preprocessing function of TFLearn) and after all these transformation now we can create the train dataset,

for this we will divide the set in two parts: 90% of the images will be used for the training and the 10% will be used for the test set.

Benchmark

For Benchmarking I will made real submission to kaggle system. As Deep Learning is a topic that that I have started learning to this project my first objective is get on the top 1000 users. After that I'll keep trying to increase my score.

Implementation

CNN Architecture

The simplest architecture for a CNN start with the input layer (an image in our case) followed by a sequence of convolutional layer and ReLU layers.

The most common form of a ConvNet architecture stacks a few Conv-ReLu layers, follows them with Pool layers, and repeats this pattern until the image has been merged spatially to a small size. At some point, it is common to transition to fully-connected layers. The last fully-connected layer holds the output, such as the class scores. In other words, the most common ConvNet architecture follows the pattern:

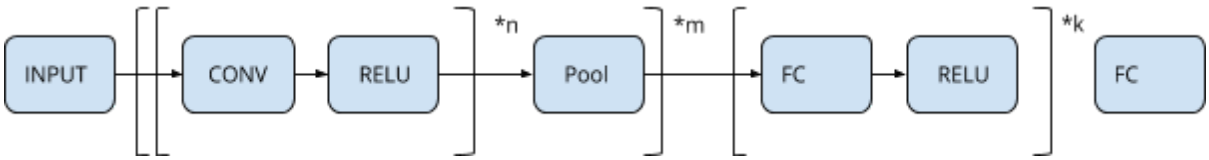


Fig 5: Common ConvNet architecture

All the classes say one thing about the CNN architecture: "Don't reinvent the wheel, use whatever works best on ImageNet". We should look what architecture work best for each type of problem, and use it as a model only making small tunes according to your data.[8][9]

Our CNN is based on the tflearn CIFAR-10 CNN tutorial code[10], in our case the input will be a image of 64x64. with 3 RGB channels, the images already have been pre processing as seen in the above sections.

After the input we use the first Convolutional Layer ("conv_1") with 32 filters each 3x3 using the ReLu activation, followed by a Max Pooling layer. In the next section we use another two convolutional layers with 64 filters each 3x3, also using ReLu activation. In the Ending of the network we are using a fully connected layer with 512 nodes, a dropout layer to avoid overfitting and the last fully connected layer with two outputs. Despite that 9 layers is pretty small for deep learning, the computation required to train this CNN is significant. Across all layers, there's 8,446,466 weight and bias parameters to be learned. In the Attachment A we can see the generated graph for the last working CNN.

Refinement

For testing the results that I get I made submissions to Kaggle, the kaggle team are using the log loss to score the results:

$$\text{LogLoss} = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right]$$

Where ‘n’ is the number of images in the test set ‘yi’ is the predicted probability of the image being a dog yi is 1 if the image is a dog, 0 if cat and log() is the natural (base e) logarithm[13].

To get the initial results I’ve used a simple CNN model, the model was like a simple version of the Common ConvNet architecture (as seen in fig. 5), with some Convolutional and ReLU layers using MSE. My first Kaggle result was not so great:

Submission and Description	Private Score	Public Score
submission_file.csv 6 hours ago by Fernando Favoretti		1.12156

The log loss of 1.12156 was very bad, in the kaggle platform such score give me about the position of approximately 1200 in the public Leaderboard.

After that result I look for the CIFAR-10 example so I made these improvements to the original net:

- Added a dropout layer [12]
- Changed the loss function which was the Mean Squared Error and now is the Cross-Entropy.
- Add more preprocessing (now use tf learn lib)

And now with 10 epoch of training and my score receive a really nice boost:

Submission and Description	Private Score	Public Score
submission_file.csv 2 hours ago by Fernando Favoretti		0.50573

The score of 0.50573 for me it’s a nice score, I get about the 900th position in the public leaderboard and finally got where I wanted in my benchmark

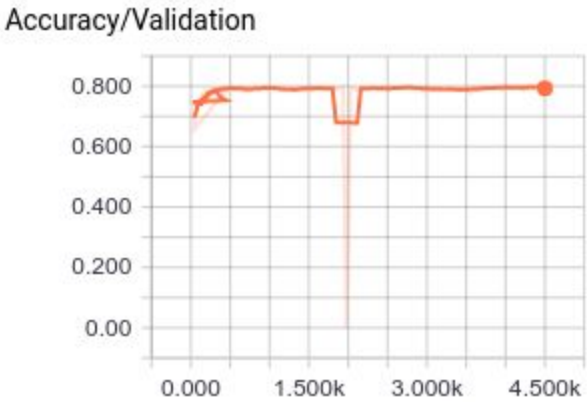
Just for testing one more thing, in the Cifar-10 Example they’ve used a option of TFlearn to generate synthetic data making simple rotations on the images to give more examples to train. As I had about 25.000 images to train the neural net (Actually 22500 to train and 2500 to test) I thought that this technique was not necessary, but I’ve tried it and for my surprise (and after some searches) I founded that 25.000 images are not such a large

amount of images to train a CNN, and using this I achieved my final score on kaggle that give me about the ~800th position:

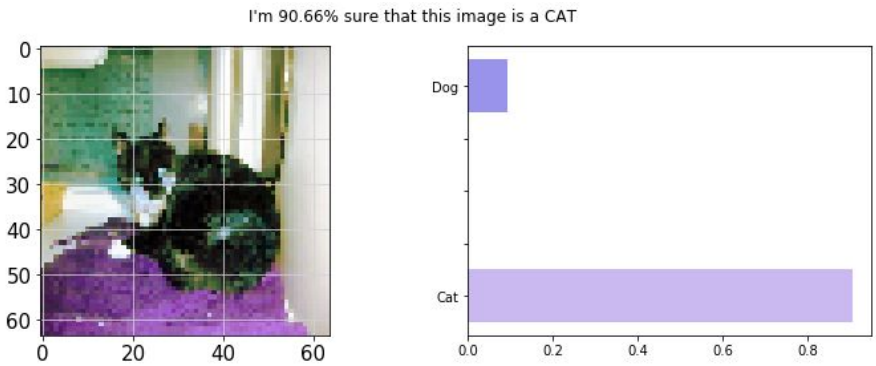
Submission and Description	Private Score	Public Score
submission_file.csv 18 minutes ago by Fernando Favoretti		0.43491

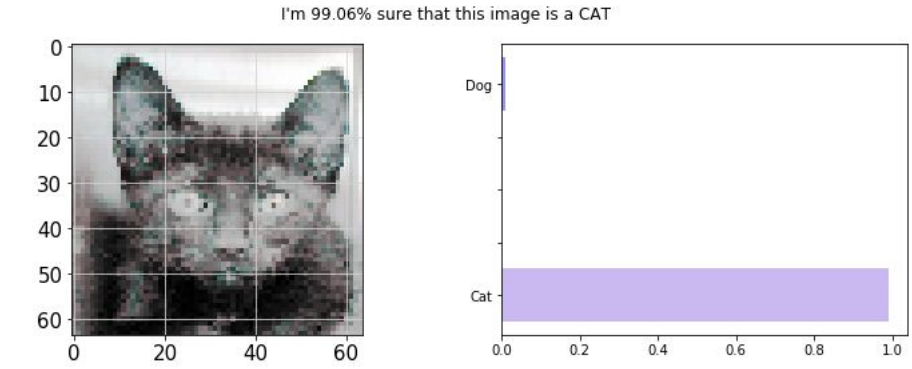
Results

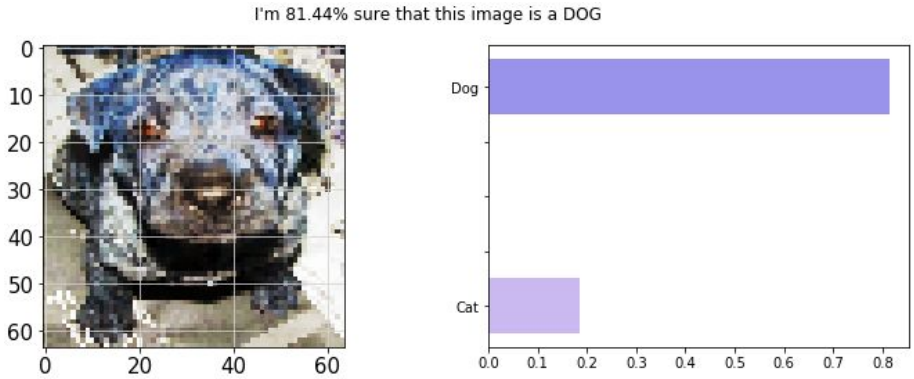
Based on the benchmark written above and as I said above, this is my first contact with Deep Learning and CNN, so I'm really very happy with the result. I get above my expectation on the benchmarking section using a CNN based on the CIFAR-10 example. The final accuracy of the model was about 80%:

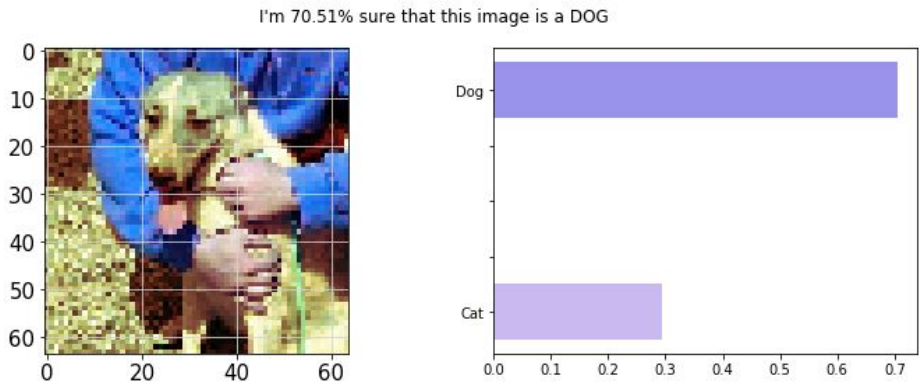


And Here are some example of predictions:

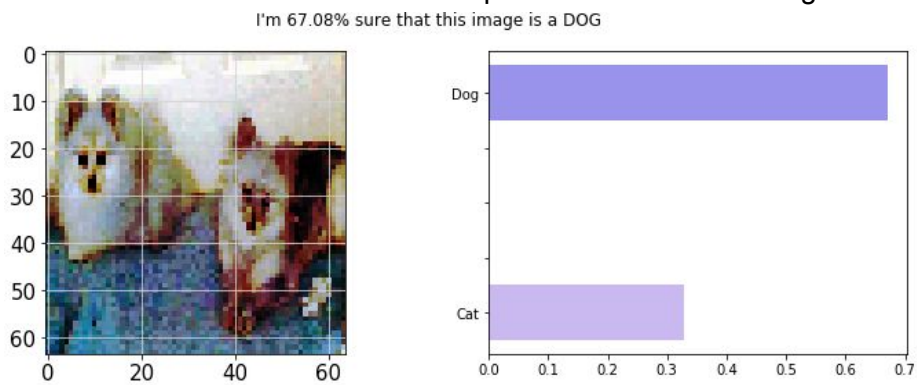






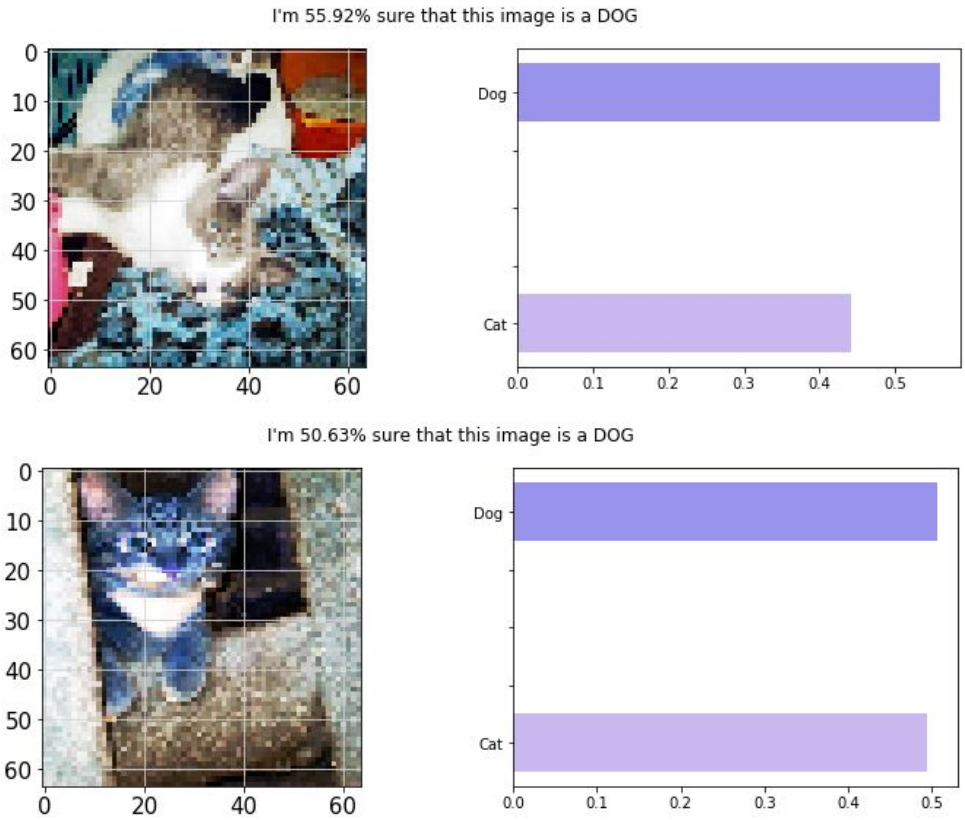


(Even with two or more animals in the picture the CNN show a good result)



Conclusion

Some examples when the CNN failed to recognize when the photo is of a cat or a dog:



From the Figures Above, and other tests some failure case can be identified:

- When the animal is in front of very colorful and contrasting objects
- When only a few parts of the animal are in the image (some faces are easy to recognize but when the picture is in some angles the task becomes hard)
- When the animal is too shrunken or lying down in a different angle like in the first picture

Reflection

The entire process for this work can be summarized with these steps:

1. An initial problem with a public dataset was found
2. The data has been downloaded , observed and preprocessed
3. The classifier was training using the data, we start with a simple one and get some insights from the CIFAR-10 example
4. The tests were made with images never seen before
5. We made some submissions to kaggle platform to achieve our benchmark

For me the steps 2 and 3 are the most difficult, as I had to study a lot of image recognition techniques to know how to preprocess the images properly for a deep learning

application (what is different to preprocess to a simple neural network as an example). After, in the step 3 The most difficult thing is to thought how to start, for this I needed to study a lot of deep learning, specially CNN, with the udacity courses, some papers and some books to get it, and after that improve the algorithm.

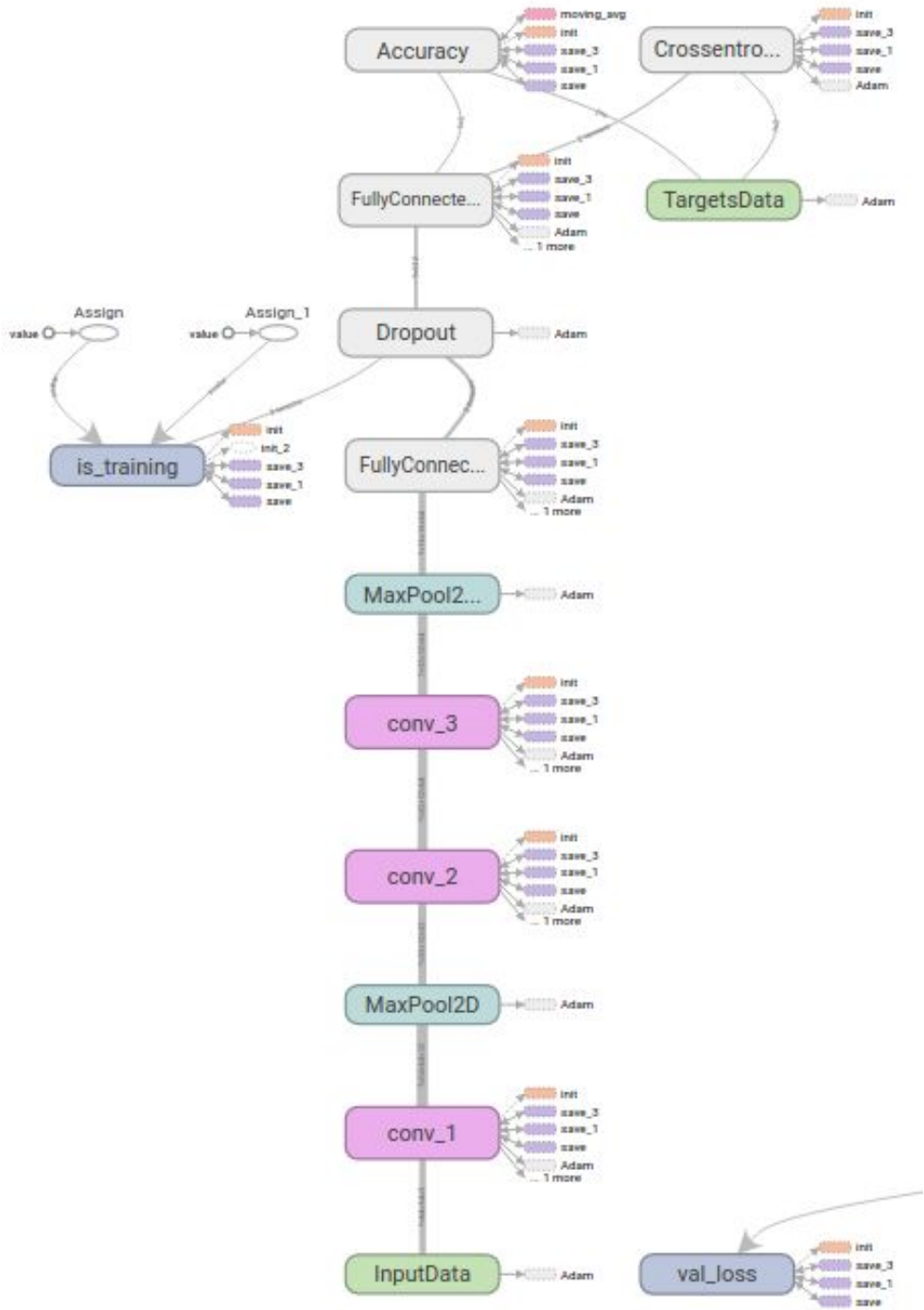
Improvements

Some nice improvements can be taken in the speed of the training and test, I've trained my CNN using a simple machine only with a 4 cores CPU (that take hours with a small quantity of epochs). Using a GPU cluster we can optimize the time and make it run with more epochs and also more data what I believe that will make our score greater.

Some users in Kaggle used pre-trained CNN to achieve better scores, but that is not the focus of this work.

Attachments

A. Graph of the last working CNN:



References:

[1]https://www.tensorflow.org/tutorials/image_recognition/
[2]https://www.toptal.com/machine-learning/an-introduction-to-deep-learning-from-perceptrons-to-deep-networks
[3]http://www.cse.unsw.edu.au/~billw/cs9444/crossentropy.html
[4]https://www.tensorflow.org/get_started/get_started
[5]https://arxiv.org/pdf/1409.1556.pdf
[6]http://cs.nju.edu.cn/wujx/paper/CNN.pdf
[7]https://www.ismll.uni-hildesheim.de/lehre/ip-14s/script/imageanalysis-02-restoration-contrast-2up.pdf
[8]https://benanne.github.io/2015/03/17/plankton.html#prepro-augmentationcnn
[9]http://cs231n.github.io/convolutional-networks/
[10]https://github.com/tflearn/tflearn/blob/master/examples/images/convnet_cifar10.py
[11]http://adilmoujahid.com/posts/2016/06/introduction-deep-learning-python-caffe/
[12]http://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf
[13]https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition#evaluation