Udacity Machine Learning Engineer
Capstone Project

Hugo David-Mauduit
November 24th 2016

Cats vs Dogs image classifier

Definition

Project Overview

During the last years, we have seen great progress in the Artificial Intelligence (AI) field, in opposition to the desillusion we faced a few decades ago. This rebirth of AI has been mostly enabled by the explosion of computational capacities of Graphics Processing Units (GPU) built for the video-games industry and the number of public datasets like ImageNet. An approach that had been left behind when these factors were not ready suddenly revealed itself very promising : Deep Learning. With a great number of labeled examples and a lot of computation, supervised deep learning gave much more accurate results and enabled great increase in the performance of AI algorithms like image recognition, text-to-speech or speech-to-text.

If big companies like Google or Facebook are using AI to achieve their challenges, the high availability of GPUs and datasets enables anyone to create Machine Learning algorithms. The democratization of the capability to build intelligent algorithms has risen great interest in the software development and data science communities. Recurrent subjects of interest in the internet world, cats and dogs have quite naturally been chosen for a now popular image recognition Machine Learning challenge : to distinguish images of dogs from cats.

The aim of this project is to build a machine learning algorithm to take-up this challenge. We will be using the public dataset used for the dog vs cat Kaggle competition [1].

Problem Statement

The chosen dataset contains labeled images of either cats (one or many), or dogs (one or many). As each image contains at least an animal, it thus belongs either to the “Cat” category or to the “Dog” category.

Our aim is to build a binary classifier having for input an image and for output a predicted category label.

To achieve this, we will use deep learning methods and inspire ourselves from the digit recognition algorithm MNIST classifier [2]. We will thus be using a combination of

Convolutional Neural Networks (CNN) and Fully Connected Layers implemented in Python using Google's open source Machine Learning framework TensorFlow [3].

We will train and evaluate our algorithm on our labeled dataset. To make our algorithm as performant as we can, we will adjust both the algorithm architecture and its numerous parameters.

Metrics

During training, we will regularly evaluate our algorithm on a training set and a validation set. After each training session, we will evaluate it on a test set. To determine its performance on each of these sets, we will compute its accuracy. Accuracy is a good metric for our binary classifier if each class label are equally present in the dataset (50% dog photos and 50% cat photos).

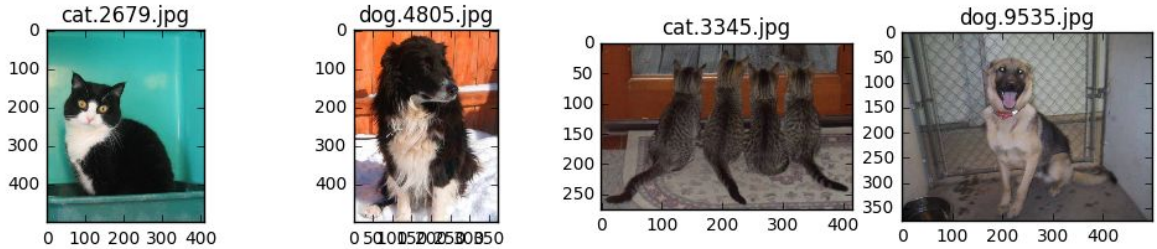
It represents the number of correct predictions among all predictions. It is defined in the following way :
accuracy = (true positives + true negatives) / dataset size
An example is positive if it is cat and negative if it is a dog.

Analysis

Data Exploration

As mentioned above, we will be using the dataset of the Dogs vs Cats Kaggle competition. As only the train folder is labeled, we will be using it for our training, validation and test sets. It is composed of 25,000 3-channels images (12,500 images of dogs and 12,500 images of cats) in the ".jpg" format, each one containing in its filename its label : "cat" or "dog".

Here are a few examples of the files, with their associated filename, width and height (in pixels number).



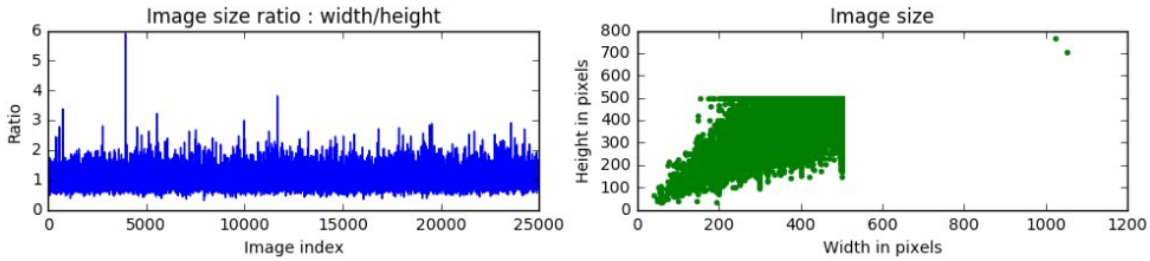
Here are a few things we can notice directly by looking at the dataset :

- As we can notice on the 3rd image above, some images contain multiple animals of the same category, which may make things harder for certain algorithms (like simple softmax regression classifiers), but better for others (CNN-based algorithms might enjoy having multiple times the feature "ear of a cat", improving the probability of the image being of cat(s)").
- Images are generally well centered on the animal.

- Images have different sizes.
- Background images seem to vary a lot and be independent from the categories. We thus avoid some misguiding correlation, which is good for generalization.

Exploratory Visualization

An important thing to notice about our data is the great variance of the image size. Indeed, our input layer needs to have a fix image input size. We thus need to know if each image can be used as input and how we need to preprocess it.
The following graphics give us for each image its size and its width/height ratio.



The average ratio is 1.157.

We can notice two outliers in the image size graph which have far too high height and width compared to the rest of the images dataset.
On the ratio graph, we can also notice an outlier which has a ratio of 6 whereas most of the other images have a ratio below 2.

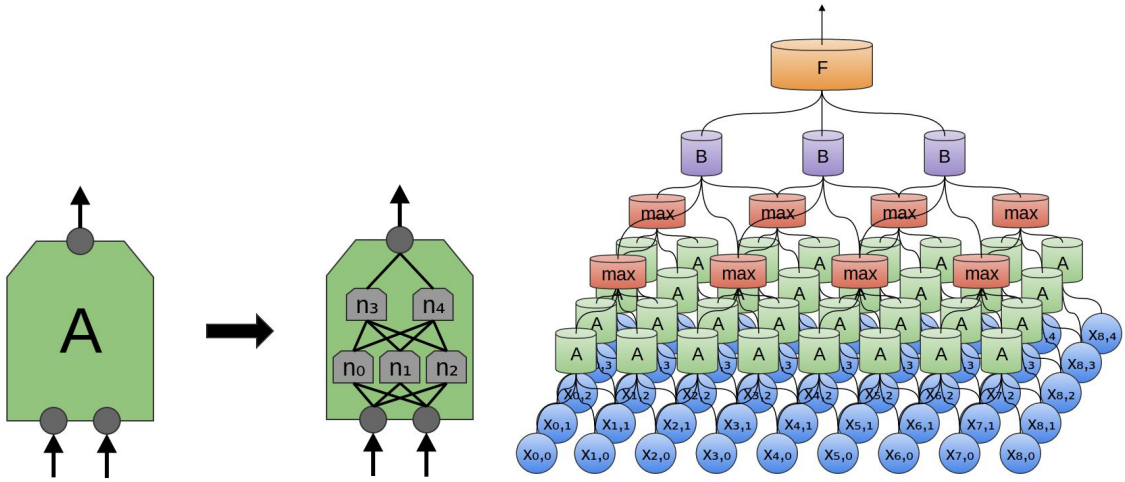
These graphs will help us determine if we need to resize or crop our images before using them as input for our algorithm.

Algorithms and Techniques

Simple softmax regression have limited efficiency when the problem it tries to solve is using data with a high variability (where many very different inputs actually have the same label), which is the case here : two pictures of cat can be extremely different. Images represent animals in different positions and images with multiple animals make it even harder for this algorithm.

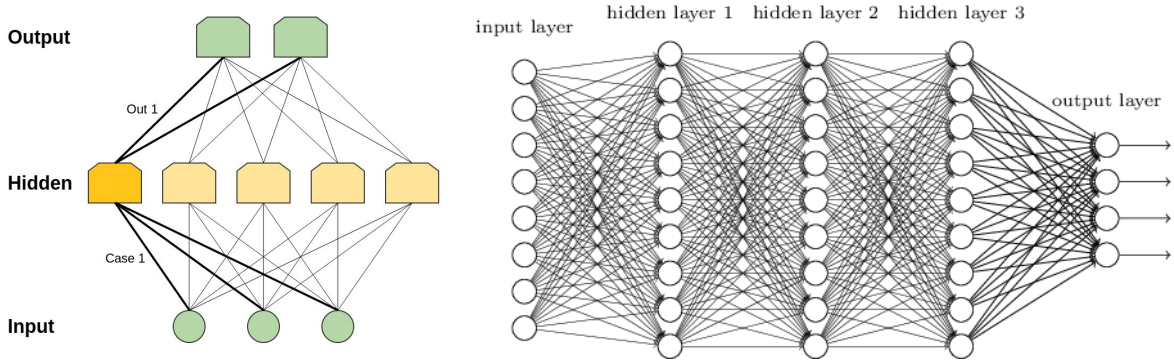
A popular tool used for image recognition is called Convolutional Neural Network (CNN). It is very useful for images where important features (like a portion of a cat ear) can be anywhere, which is the case here as animals are in different position and sometimes more than one per image.

A CNN is like a tiny neural network that is applied everywhere on square portions of an image. It takes patches (small group of pixels) as input to analyze what this portion of the image contains. Since this very same neural network is applied on all parts of the image, it won’t matter where the important feature is, it will detect it anyway.



A max-pooling layer takes the maximum of features over the outputs of a few close blocks of the previous layer. Adding a maxpool layer after a convolutional layer “zooms out” and enables the following convolutional layer to work on the image from a higher point of view, with the important selected features.

A fully connected layer is composed of a lot of neurons, each one combining all the inputs in a simple calculus, the result being its output. Using a sequence of fully connected layers, one layer’s output being another’s input, allows to create complex functions from a great number of simple operations. It is great for combining the extracted features and make reasonings like “Well, we have two pointy ears, moustaches and two green eyes. That must be a cat.”



A dropout layer is a layer that sometimes lets information pass sometimes block it. Using it for training (but letting everything pass when evaluating) can force the algorithm to write multiple times in its mind the same reasonings and less “remember” the data. It forces it to learn a methodology more than remember everything. Great to use when the algorithm have a tendency to overfit the data.

We will thus use some CNNs in our algorithm to extract important features and then multiple fully connected layers to combine these features in a useful way to determine if the image is about a cat or a dog.

Training will consist in putting images as input, make our network output a probability score for each category (ex: our model might say “I am 60% sure this is a cat and 40% this is a dog”), and since the image is labeled (let’s say it is a cat) and we know the influence of each

neuron's weights, we automatically tweak these weights to make our model closer to a 100% confidence it is a cat. This adjustment is called "backpropagation". To do this, we compute a loss function and find its lowest value with an optimizer to find the best combination of these weights.

Algorithms with convolutional layers need a lot of data to be trained on. As our dataset contains only 25000 images to be splitted into training, validation and test sets, we will be using techniques to generate more images.

We will be able to improve our algorithm by tuning the following parameters :

- Number of channels of the input image (1 for grayscale, 3 for color image).
- Height and width of the input image. High resolution images give better details of the image but require much more computation and more training images.
- Learning rate of the optimizer. A high learning rate makes the optimizer find rapidly a result to its optimization task, which also can mean it doesn't find the best result, leading to a final bad algorithm. High is good for few training data, low is better when having a lot of data.
- Initial weights and biases of neurons.
- Layers architecture.
- Layers types (convolutional, fully connected, maxpool, dropout).
- Layers parameters (ex: number of neurons per layer, CNN's input and output length, dropout's keep probability).
- Image generation algorithms and their parameters (horizontal image flip, brightness and contrast changes).
- The number of examples per batch. Higher will make less precise but faster the optimizer.
- The number of batches used for training.

Benchmark

The Kaggle leaderboard shows data scientists managing to reach a log loss of 0.042. However, our metric being an accuracy, it is more helpful to look at another competition using the same dataset, which metric is Categorization Accuracy [4]. The leader Pierre Sermanet managed to get an accuracy of 0.98914. The data it was evaluated on is not labeled so we won't be able to use the same, but it gives an idea of a potentially reachable accuracy.

Methodology

Data Preprocessing

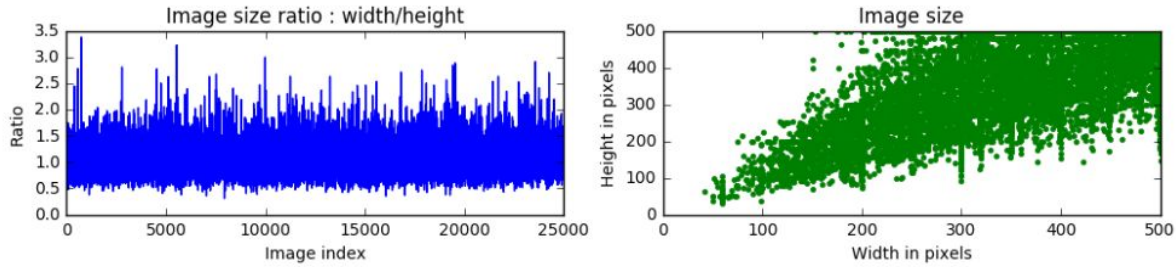
To learn effectively, the algorithm needs some good quality input but still images with not too big size for a limited information to process. Indeed, to find patterns among high volumes of information, we need large models, which cannot be the case due to our limited computation

capabilities. Additionally, it is easier to understand what matters in a image if you have less information to filter. It is thus important to have lower resolution images either by resizing them or by cropping them.

The two outliers images with largest width and height mentioned above were deleted from the dataset.

The image cropping operation to generate additional training images has been removed from the process, due to bugs in the use of the TensorFlow function : cropped image were composed of random color pixels, making it unusable for training. Resizing images was thus the chosen option and it became necessary to check that images were not too much distorted by this transformation. One particular outlier has been identified from the study of the ratio width/image. For a reason of symmetry between categories (and a bit of perfectionism), a second less obvious outlier has also been removed : it had an image size ratio of 3.8, which is higher than the other images, but is not the only one to be a bit different. It may not have been necessary to remove the second one, since a difference of one image between the dog and cat categories would not have biased too much the algorithm. The acceptability of image distortion was purely arbitrary, by plotting the resized image and keeping as many of the already too few images as possible in the dataset.

Images statistics after removing these four outliers show a more consistent new dataset :



As we only have 24996 images (12498 cats and 12498 dogs), themselves to split into the training, validation and test sets, we need to generate some more images. During training, we will be using same images multiple times but each time randomly changing its contrast, its brightness and randomly flipping. Each new image is fairly similar to its original image but still helps the algorithm to generalize.

Training, validation and test sets have been created by splitting the original labeled dataset, respecting an equal cat/dog proportion for each set.

Labeled images repartition in the sets :

	Cats Number	Dogs number
Training Set	10498	10498
Validation Set	1000	1000
Test Set	1000	1000

The images information were stored in a single Pandas DataFrame to easily fill the training algorithm with the appropriate data. The algorithm will use the file location in the computer, the image label (0 means cat, 1 means dog) and the set name (train, validation or test).

Here is the end of the dataset DataFrame :

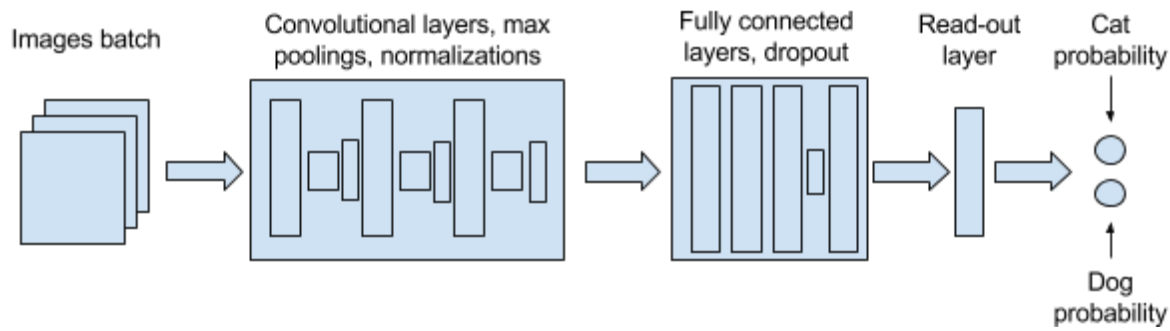
	full_path	label	width	height	ratio	set_name
24991	/home/hugo/Desktop/cat-or-dog/images/dog.661.jpg	1	240	179	1.340782	train
24992	/home/hugo/Desktop/cat-or-dog/images/dog.7429.jpg	1	399	500	0.798000	train
24993	/home/hugo/Desktop/cat-or-dog/images/dog.10948...	1	499	375	1.330667	train
24994	/home/hugo/Desktop/cat-or-dog/images/dog.2499.jpg	1	499	375	1.330667	train
24995	/home/hugo/Desktop/cat-or-dog/images/dog.8010.jpg	1	174	223	0.780269	test

Implementation

The data exploration and the machine learning algorithm were coded on two separate IPython Notebook. To be passed from the data exploration notebooks, the prepared Pandas DataFrame has been saved in a ".h5" format file.

Getting inspiration from existing image recognition algorithm on the web like the TensorFlow implementation of a MNIST, it was natural that the algorithm would consist in a sequence of :

- convolutional layers to find patterns in the image
- some max pooling and eventually normalization operations to have information of "better quality"
- followed by some fully connected layers to combine these information in a productive way
- maybe using a dropout layer to reduce overfitting by forcing the model to have multiple "copies of its logical reasoning".



However the number of each layer type was to be determined iteratively, based on the performance of each tried model architecture.

To easily iterate over different algorithms architecture and parameters combinations, the possibility to save and load trained models from checkpoints has been helpful for add training steps to already trained models. Successful training sessions' model-performance pairs were store by saving copies of IPython Notebooks to for later analysis.

The main steps of the algorithm were :

- Set the algorithm parameters
- Load the dataframe with its training, validation and test sets
- Define helpers to extract information from the dataframe
- Define helpers to create and initialize the model's components
- Create the TensorFlow model
- Define helpers to create TensorFlow operators
- Create and initialize the TensorFlow graph
- Train the model by filling it with labeled images batches from the training set and using an Adam optimizer to update neurons' weights with backpropagation.
- Regularly evaluate the algorithm on training and validation sets
- Finally evaluate the trained model on the test set

Images were randomly grouped in batches of 50. Training was made using 10000 batches (500,000 images). Each evaluation on training and validation sets used 30 batches of images of their respective sets. The final evaluation on test set used the entire and exact test set : 40 batches (2000 images).

Some problems occurred during the iterations over different models, due to the laptop used for training. When convolutional and fully connected layers had too many neurons, or when images resolution were too high, the python kernel regularly crashed, probably because of insufficient available memory, which lead to a reduction of the possible model size and thus its performances.

Additionally, as no compatible GPU were available on the laptop, the training only were using CPUs, making them very slow particularly for images with resolutions higher than 80 x 92. Training sessions were thus usually made during the night as we could only see representative performances (thanks to regular evaluations on training and validation sets) after several hours of training. The algorithm optimization was thus very time-consuming and took many weeks to finally get a learning algorithm but with not so good performances.

Refinement

Multiple combinations of parameters have been tried. The tuned parameters were :
The number of channels (1 for grayscale image, 3 for colored images).
The image size.
The learning rate of the Adam optimizer.
The initial weights and bias of convolutional and fully connected layers.
The keep probability of the dropout layer when training. It is set to 1 when evaluating.

We will present in detail three of the models that have been evaluated at differents steps of the refinement process. These parameters tunings were some of the ones that finally lead to a learning algorithm.

The fixed parameters across these models are :

The number of channels : 3.
The image size : 103 pixels width and 90 pixels height.
The learning rate of the Adam optimizer : 0.0001.

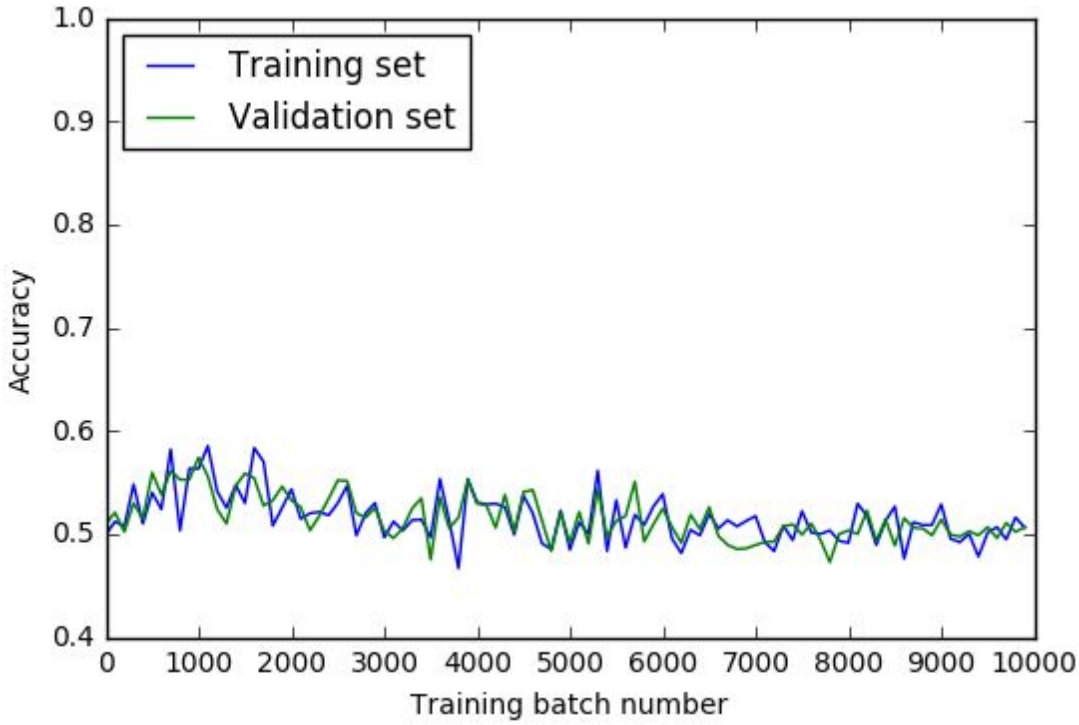
The parameters varying across these models are :
The training keep probability of the dropout layer (train_keep_prob).
The initial weights and bias of convolutional and fully connected layers. The weights are initialized using random repartition following a truncated normal distribution. The tuned parameter is its standard deviation (weights_init_stddev). The bias initial value is the value of the parameter (bias_init_val).

Changes were also made in the model architecture.

Model A
Parameters:
train_keep_prob : 0.75
weights_init_stddev = 0.1
bias_init_val = 0.1

Layers (from input to output):
Convolutional layer. Input patch: 15x15. Output length: 32.
Max pooling.
Local response normalization.
Convolutional layer. Input patch: 3x3. Output length: 32.
Local response normalization.
Max pooling.
Fully connected layer. Number of neurons : 1024.
Fully connected layer. Number of neurons : 1024.
Drop out.
Fully connected layer. Number of neurons : 1024.
Fully connected layer. Number of neurons : 1024.
Linear output.

Model evaluation:
Accuracy evolution over training steps, on training and validation sets :



Final accuracy on test set : 0.4995

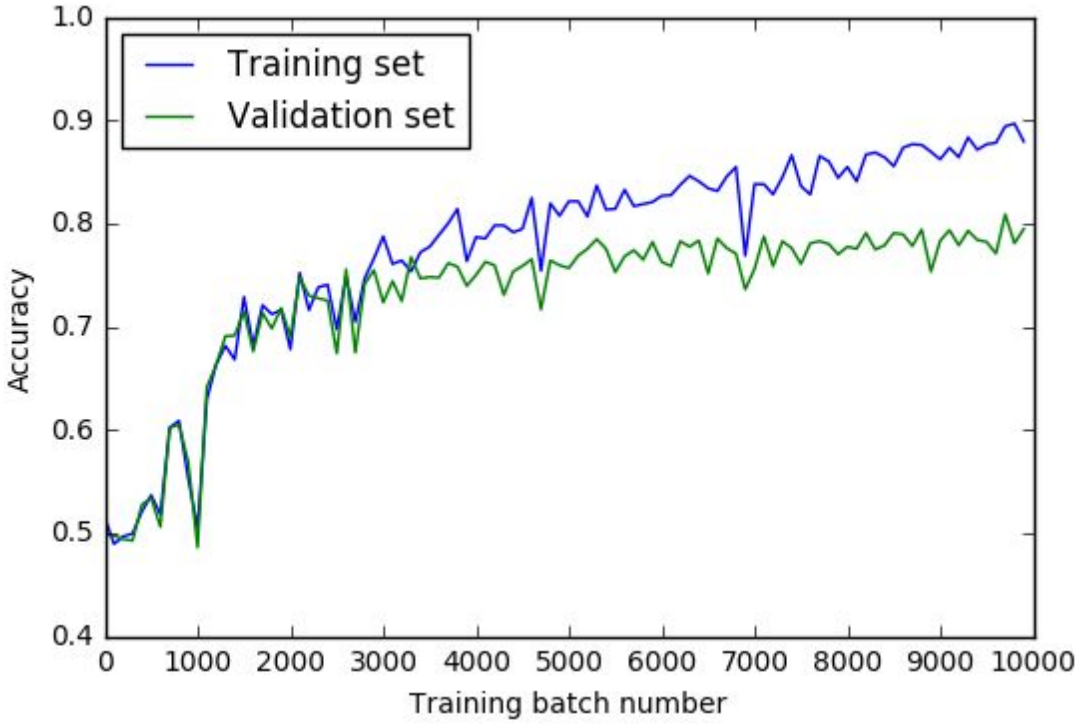
Comments :
It is clear that the algorithm is not learning at all as it is not better than a pure random binary classifier which would have a 0.5 average accuracy.

Model B
Parameters:
train_keep_prob : 0.75
weights_init_stddev = 0.01
bias_init_val = 0.01

Layers (from input to output):
Convolutional layer. Input patch: 15x15. Output length: 128.
Max pooling.
Local response normalization.
Convolutional layer. Input patch: 10x10. Output length: 128.
Max pooling.
Local response normalization.
Convolutional layer. Input patch: 5x5. Output length: 128.
Local response normalization.
Max pooling.
Fully connected layer. Number of neurons : 2048.
Fully connected layer. Number of neurons : 2048.
Drop out.
Fully connected layer. Number of neurons : 2048.
Fully connected layer. Number of neurons : 2048.

Linear output.

Model evaluation:
Accuracy evolution over training steps, on training and validation sets :



Final accuracy on test set : 0.7835

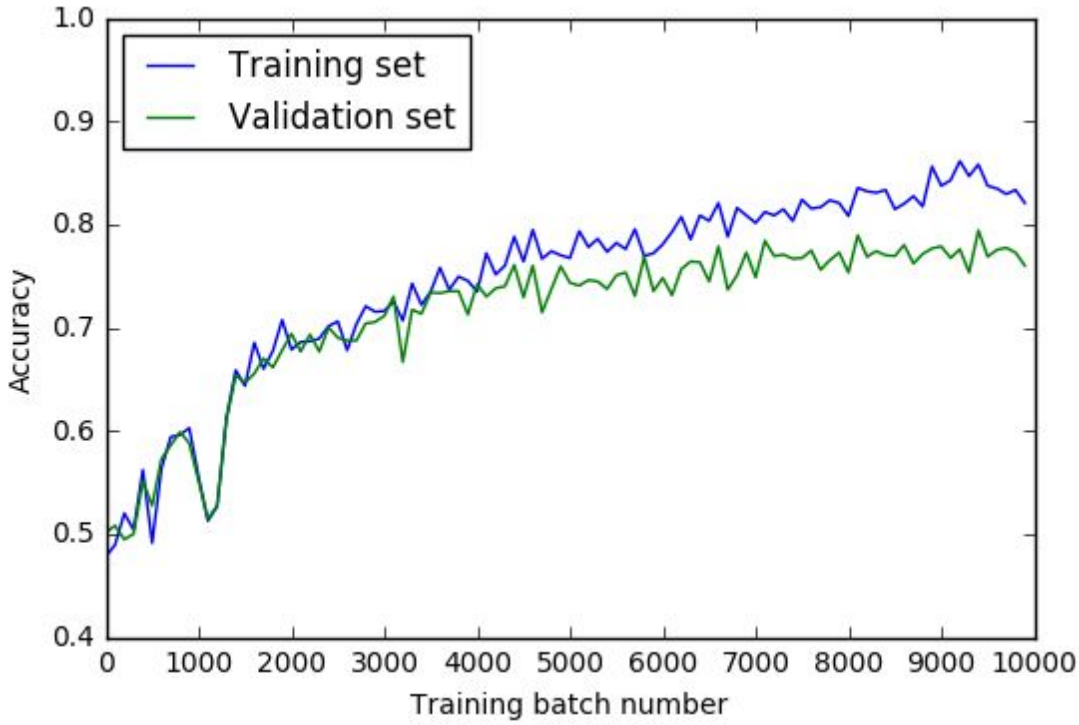
Comments :
In comparison to model A, neurons' weights and biases initial values have been reduced and the model is now deeper (more layers) and wider (more neurons).
Thanks to these changes, our algorithm has been able to learn and generalize over the training steps: the accuracy on validation and test sets is now approaching the 0.8 accuracy after being trained on 500,000 images. However accuracy on the training set keeps rising as we keep using the same images : the model is overfitting the training dataset.

Model C
Parameters:
train_keep_prob : 0.25
weights_init_stddev = 0.01
bias_init_val = 0.01

Layers (from input to output):
Convolutional layer. Input patch: 15x15. Output length: 32.
Max pooling.
Local response normalization.
Convolutional layer. Input patch: 11x11. Output length: 64.
Max pooling.

Local response normalization.
Convolutional layer. Input patch: 8x8. Output length: 128.
Max pooling.
Local response normalization.
Convolutional layer. Input patch: 5x5. Output length: 256.
Local response normalization.
Max pooling.
Fully connected layer. Number of neurons : 2048.
Fully connected layer. Number of neurons : 2048.
Drop out.
Fully connected layer. Number of neurons : 2048.
Fully connected layer. Number of neurons : 2048.
Linear output.

Model evaluation:
Accuracy evolution over training steps, on training and validation sets :



Final accuracy on test set : 0.7885

Comments :
In comparison to model B, a convolutional layer has been added, some convolutional layers' parameters adjusted and the dropout's keep probability has been reduced. Thanks to this last change, our model is less overfitting the data : the accuracy on training and validation sets are now closer to each other.
However, we did not get higher accuracies on validation and test sets.

Results

Model Evaluation and Validation

Used parameters seem appropriate, even if a 0.25 keep probability for the dropout layer seems a bit small. Other ways of reducing overfitting should be investigated to actually enable other parts of the architecture to improve the accuracy on validation and test sets.

The used images' height, width and ratio also were good enough to let the algorithm detect interesting patterns but small enough not to embarrass the algorithm with too much information or simply make it crash. This trade-off has been nicely found.

We managed to build a model that learnt to distinguish images of cats from images of dogs. Indeed, our final model's accuracy on test set is 0.7885. Its architecture is similar to others algorithms in the image recognition, even if it is wider and deeper than expected.

The original split of the labeled dataset into training, validation and test set gave us an appropriate way to evaluate our model. Using only the training model would have falsified our point of view since we would not have been able to distinguish seen images from new images. We were able to see when there was overfitting and check that our model was indeed generalizing. Additionally, using both validation and test sets enabled us not to overfit a unique test set.

Multiple training have been made using the last model each time randomly changing the dataset repartition among the training, validation and test sets. Due to the long time used for training, only three accuracy values have been made on the test set : 78.85%, 77.85% and 78.15%. Since there is not much difference between these values, we can assume that our model is quite robust to perturbations.

Justification

An accuracy of 78,85% is actually bad so we cannot say our model can be trusted but we managed to make a learning algorithm which aims in the right direction for solving the problem and shows it is able to identify patterns.

Its accuracy is way below the Kaggle competition's best score of 0.98914. We would be at the 92th place, which cannot be considered as a great achievement, but still is encouraging for our first machine learning challenge.

Conclusion

Here is the confusion matrix of our final model:

	Predicted: cat	Predicted: dog
Actual: cat	832	165
Actual: dog	258	745

It appears that even if our training, validation and test sets had strict equal repartitions between the cats and dogs categories, our model has a tendency to more often predict that an image is a cat (54.5%) than a dog (45.5%).

If we consider a cat as being “positive”, our precision and our recall are :
Precision = True Positive / (True Positive + False Positive) = 832 / (832 + 165) = 0.835
Recall = True Positive / (True Positive + False Negative) = 832 / (832 + 258) = 0.763

Our F1 score it thus :
F = 2 * Precision * Recall / (Precision + Recall) = 0.797
Let’s hope we’ll do better next time !

Our first entire project using Machine Learning techniques has been an interesting road through both theoretical and practical parts.

On the technical field, we have practiced coding with python, numpy, pandas and matplotlib through data exploration, manipulation and visualization, and by implementing the entire project. We have also greatly improved our knowledge of TensorFlow on all steps of this machine learning algorithm, from creating queues of images, putting them in batches, transforming them, define a model with multiple layers of multiple type, training it and evaluating it on different datasets.

Slowly tuning our model’s parameters and optimizing its architecture also helped us understand how each layer works and its role in the global architecture. We also faced trade-offs and saw the reality of improving a model’s accuracy thanks to metrics analysis and different techniques.

In terms of possible improvements, a better computer to train the model on, particularly with compatible GPUs, would be of good help. It would enable us to make faster trainings for more model iterations, play with better images quality and train wider and deeper models.

Some additional theoretical knowledge would also be very helpful to create models with much more performance. Let’s hope a few years of experience will enable us to get significant improvements.

In the meantime, we could use transfer learning by adjusting on our data some pre-trained models like Google's Inception to rapidly get great performance.

Globally, this has been a great first experience to learn a lot and improve our coding skills to get into the data science and machine learning world.

References

[1] https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition
[2] https://github.com/tensorflow/tensorflow/tree/r0.11/tensorflow/examples/tutorials/mnist
[3] https://www.tensorflow.org/
[4] https://www.kaggle.com/c/dogs-vs-cats/leaderboard

Many thanks to Colah for his great guides to intuitively understand the components of Deep Learning tools. Some illustration were taken from one of his articles:
http://colah.github.io/posts/2014-07-NLP-RNNs-Representations/