Capstone Project

Machine Learning Engineer Nanodegree

Nicolás Alvarez

August 22nd, 2016

I. Definition

Project Overview

According to the CDC motor vehicle safety division, one in five car accidents[1] is caused by a distracted driver. Sadly, this translates to 425,000 people injured and 3,000 people killed by distracted driving every year in USA.

This project is based on a Kaggle[2] competition called “State Farm Distracted Driver Detection”[3]. Given a dataset of 2D dashboard camera images, I developed a ConvNet that is capable to classify each driver's behavior, that is, if they are driving attentively or if they are distracted, for example, taking a selfie with their friends in the backseat.

Problem Statement

The project's goal is to predict the likelihood of what the driver is doing in each picture. The main tasks involved are the following:

1. Download and preprocess the train, validation and test datasets.
2. Build and train the ConvNet classifier that can determine what the driver is doing in

1 http://www.cdc.gov/motorvehiclesafety/distracted_driving/
2 https://www.kaggle.com
3 https://www.kaggle.com/c/state-farm-distracted-driver-detection

each picture.

3. Measure performance on validation set and fine tune the model if necessary.
4. Predict images in the test dataset and submit the predictions file to the Kaggle competition.

Metrics

Two metrics are used to measure the performance of the ConvNet classifier: **Precision** and **Multi-class logarithmic loss**.

The first one is given by:

$$Precision = \frac{tp}{tp + fp}$$

where ***tp*** is the true positives and ***fp*** the false positives.

The multi-class logarithmic loss, the one used in Kaggle's competitions, is given by:

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\log\left(p_{ij}\right)$$

where ***N*** is the number of images in the test/validation set, ***M*** is the number of image class labels, log is the natural logarithm, ***y_ij*** is 1 if observation i belongs to class j and 0 otherwise, and ***p_ij*** is the predicted probability that observation i belongs to class j.

II. Analysis

Data Exploration

The data[4] provided by the competition consists of the following files:

- **imgs.zip:** zipped folder of all (train/test) images
- **sample_submission.csv:** sample submission file in the correct format
- **driver_imgs_list.csv:** list of training images, their subject (driver) id, and class id

The examples are driver images of 640px x 480px, each taken in a car with a driver doing something in the car (texting, eating, talking on the phone, makeup, reaching behind, etc).

The 10 classes to predict are:

- **c0:** safe driving

4 https://www.kaggle.com/c/state-farm-distracted-driver-detection/data

- **c1:** texting – right
- **c2:** talking on the phone – right
- **c3:** texting – left
- **c4:** talking on the phone – left
- **c5:** operating the radio
- **c6:** drinking
- **c7:** reaching behind
- **c8:** hair and makeup
- **c9:** talking to passenger

The train and test data are split on the drivers, such that one driver can only appear on either train or test set.

The train dataset has 22423 labeled images and the test dataset has 79726 unlabeled examples. The train data was split in two subsets, one of 17937 examples for training and another one of 4485 examples for validation.

Exploratory Visualization

The Figure 1 shows a bar diagram of the number of occurrences of each class. From a total of 22424 example in the training set, each class has between 2000 and 2500 examples. Despite it is a small dataset, it is balanced between classes.

On the other hand, in Figure 2 is shown the examples distribution in function of the drivers. Images of the dataset come from just 26 drivers, having approximately 80 images per every class per every driver. This is a very important thing to be considerer when splitting the dataset in training and validation sets for avoiding overfitting.

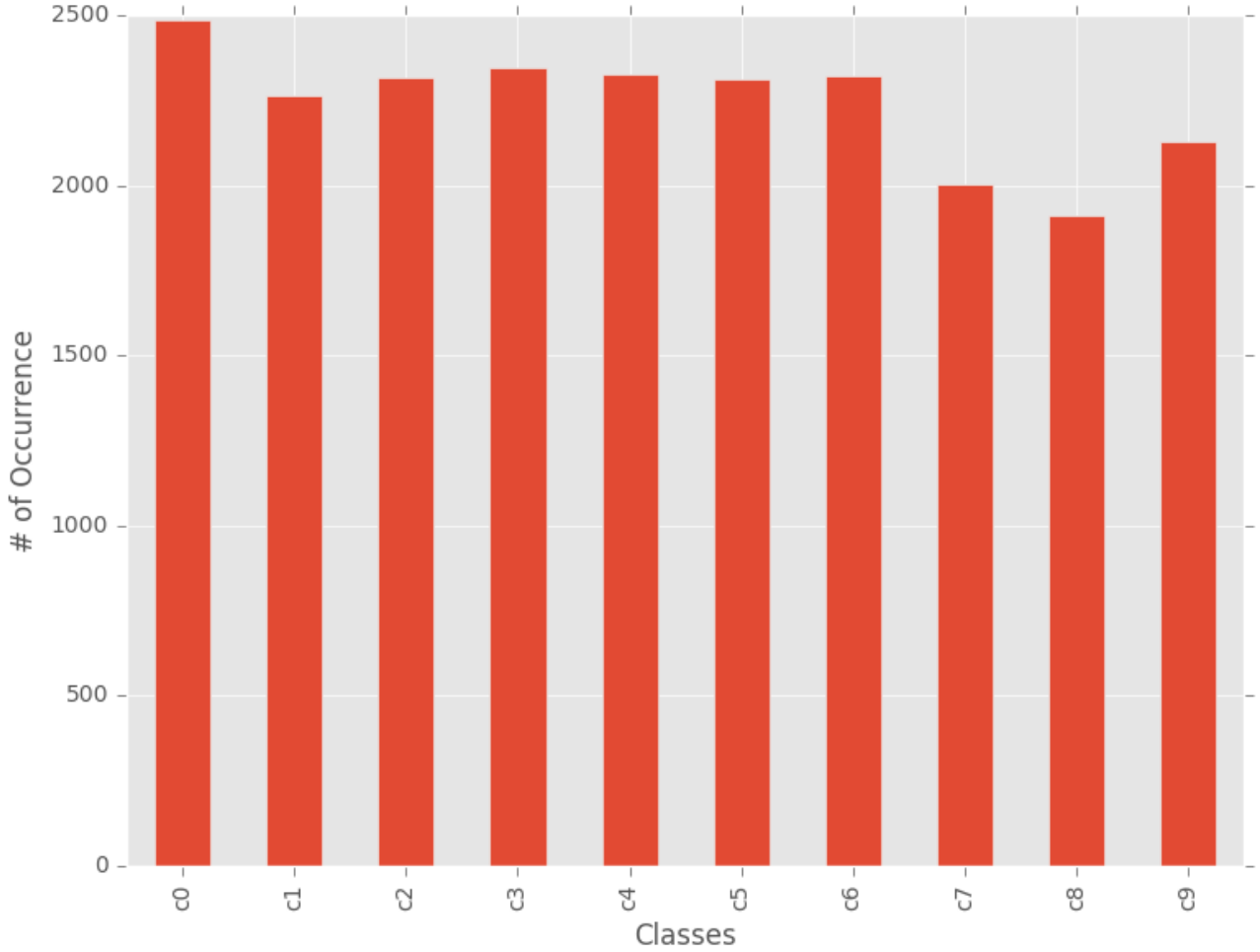

Figure 1: Occurrences per Class

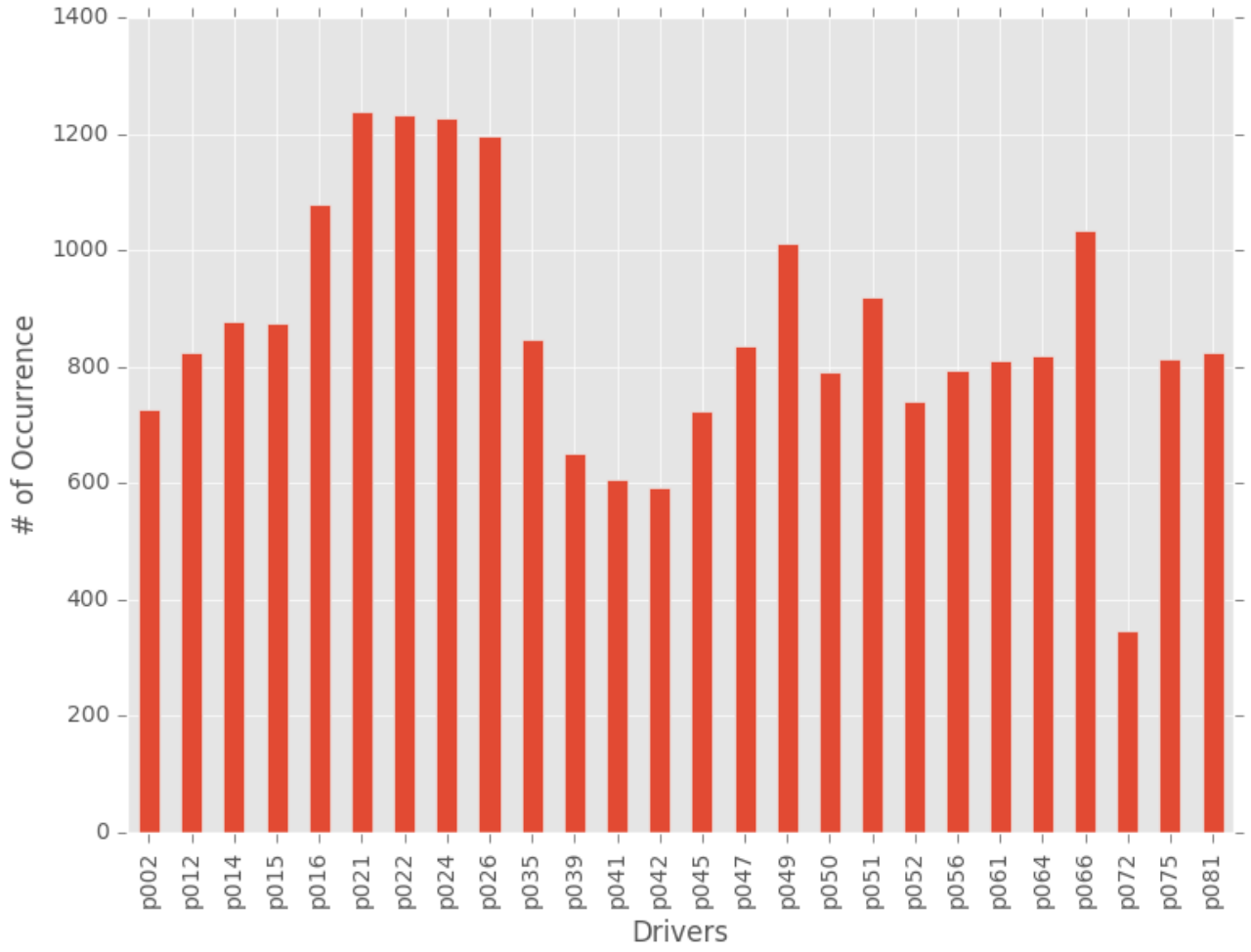

Figure 2: Occurrences per Driver

Algorithms and Techniques

Nowadays, Convolutional Networks[5][6], also known as ConvNets, or some close variant are used in most neural networks for image recognition. Making the explicit assumption that the inputs are images, these networks use a special architecture which is particularly well-adapted to classify images. This architecture vastly reduces the amount of parameters in the network, making ConvNets fast to train. Also ConvNets outputs an assigned probability for each class, as it is required by the competition rules.

The layers of a ConvNet have neurons arranged in 3 dimensions: width, height, depth. As shown in Figure 3, the neurons in a layer are only connected to a small region of the layer

5 http://cs231n.github.io/convolutional-networks/
6 http://neuralnetworksanddeeplearning.com/chap6.html

before it, instead of all of the neurons in a fully-connected manner.

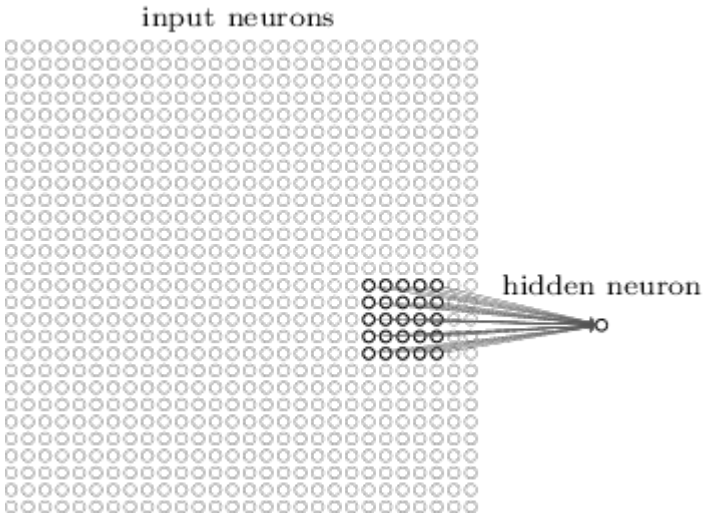

Figure 3: Hidden neuron in a ConvNet

Generally speacking, a ConvNet is a sequence of layers (Convolutional Layer, Pooling Layer and Fully-Connected Layer), and every layer of a ConvNet transforms one volume of activations to another through a differentiable function. In this way, ConvNets transform the original image layer by layer from the original pixel values to the final class scores.

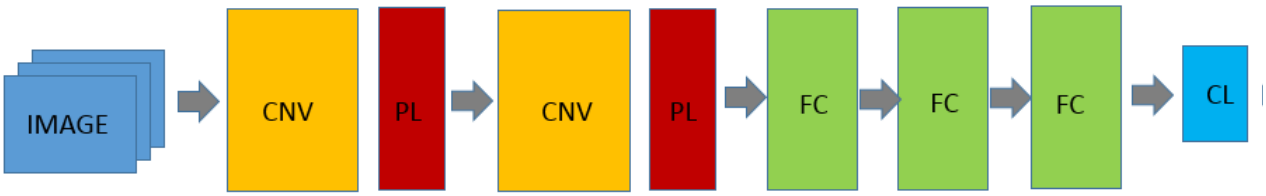

Figure 4: ConvNet Architecture Example

The following parameters can be tuned to optimize the ConvNet to be implemented:

- **Training parameters**
  - Number of epochs
  - Batch size
  - Optimization algorithms for learning, such as Gradient Descent or Adagrad
  - Learning rate (initial value and rate decay)

- Dropout probability (for avoiding overfitting)

- **Neural network architecture**
  - Number of layers
  - Layer types (convolutional, fully-connected, or pooling)
  - Layer parameters (no. of neurons, bias and weight initialization)
- **Preprocessing parameters** (see the Data Preprocessing section)

Benchmark

The Private Leaderboard[7] of the State Farm Distracted Driver Detection Competition, which is computed on 75% of the test set, will be used as benchmark for this project. The goal is to arrive to a solution that is in the top NNN of the competition.

III. Methodology

Data Preprocessing

The training dataset is not large enough for training the model, so some distortions are apply to the images. The following preprocessing is applied to the training datasets' images:

- Check that all images have size 640px by 480px-
- Resize the original images from 640px by 480px to 64px by 48px.
- Save images in binary files, in batches up to 30MB per file, formatted as follows:
  - <1 x label><height*width*depth x pixel>
- Convert images to a squared form adding zero pixels as padding to a size of 1.2 the width of the image.
- Randomly crop the image to a size of 64px by 48px.
- In a random order, apply the following distortions:
  - Randomly adjust the brightness.
  - Randomly adjust the contrast.
  - Randomly adjust the saturation.

7 https://www.kaggle.com/c/state-farm-distracted-driver-detection/leaderboard

    - Randomly adjust the hue.
- Finally, linearly scales the image to have zero mean and unit norm.

Figure 5: Processed images examples

Implementation

The model, called Distracted Driver Detection Model (DDDM), was implemented in Python using the TensorFlow[8] library. It is an open source software library for numerical computation using data flow graphs. TensorFlow was originally developed for the purposes of conducting machine learning and deep neural networks research, but the system is general enough to be applicable in a wide variety of other domains as well.

The model's code is organized as shown in the following table:

File	**Purpose**
DDDM.py	DDDM ConvNet model.
DDDM_data_exploration.py	Routine for DDDM data exploration
DDDM_input.py	Routine for convert and decode the DDDM dataset.
DDDM_train.py	Routine for training the DDDM.
DDDM_val.py	Routine for validating the DDDM.
DDDM_test.py	Routine for testing the DDDM.

8 https://www.tensorflow.org/

Model Inputs

Model Prediction

Model Training

Launching and Training the Model

Model Validation

Model Testing

Model Visualization

In this section, the process for which metrics, algorithms, and techniques that you implemented for the given data will need to be clearly documented. It should be abundantly clear how the implementation was carried out, and discussion should be made regarding any complications that occurred during this process. Questions to ask yourself when writing this section:
- _Is it made clear how the algorithms and techniques were implemented with the given datasets or input data?_
- _Were there any complications with the original metrics or techniques that required changing prior to acquiring a solution?_
- _Was there any part of the coding process (e.g., writing complicated functions) that should be documented?_

Refinement

. Make sure you're getting the loss you expect when you initialize with small parameters. It's best to first check the data loss alone (so set regularization strength to zero). For example, for CIFAR-10 with a Softmax classifier we would expect the initial loss to be 2.302, because we expect a diffuse probability of 0.1 for each class (since there are 10 classes), and Softmax loss is the negative log probability of the correct class so: -ln(0.1) = 2.302.

[(3x3x64)CONV -> RELU -> (3x3x64)CONV -> RELU -> POOL]*2 -> [(384 , 192)FC ->RELU]*2 -> FC

P=0.30

[(5x5x64)CONV -> RELU -> (5x5x64)CONV -> RELU -> POOL]*2 -> [(192 ,96)FC ->RELU]*2 -> FC

P=0.4

[(5x5x32)CONV -> RELU -> (5x5x32)CONV -> RELU -> POOL]*2 -> [(48)FC ->RELU]*1 -> FC

P=0.34

[(5x5x64)CONV -> RELU -> (5x5x64)CONV -> RELU -> POOL]*3 -> [(96, 192)FC ->RELU]*2 -> FC

P=0.37

[(3x3x128)CONV -> RELU -> (5x5x64)CONV -> RELU -> POOL]*3 -> [(96 , 48)FC ->RELU]*2 -> FC

P=0.4

[(3x3x128)CONV -> RELU -> (3x3x64)CONV -> RELU -> POOL]*3 -> [(4096, 2048)FC ->RELU]*2 -> FC

P= 0.375

Distorcionando las imagenes originales:

[(5x5x64)CONV -> RELU -> (5x5x64)CONV -> RELU -> POOL]*2 -> [(192 ,96)FC ->RELU]*2 -> FC

P=0.44 en 100 iteraciones

In this section, you will need to discuss the process of improvement you made upon the algorithms and techniques you used in your implementation. For example, adjusting parameters for certain models to acquire improved solutions would fall under the refinement category. Your initial and final solutions should be reported, as well as any significant intermediate results as necessary. Questions to ask yourself when writing this section:
- _Has an initial solution been found and clearly reported?_
- _Is the process of improvement clearly documented, such as what techniques were used?_
- _Are intermediate and final solutions clearly reported as the process is improved?_

IV. Results

(approx. 2-3 pages)

Model Evaluation and Validation

In this section, the final model and any supporting qualities should be evaluated in detail. It should be clear how the final model was derived and why this model was chosen. In addition, some type of analysis should be used to validate the robustness of this model and its solution,

such as manipulating the input data or environment to see how the model's solution is affected (this is called sensitivity analysis). Questions to ask yourself when writing this section:
- _Is the final model reasonable and aligning with solution expectations? Are the final parameters of the model appropriate?_
- _Has the final model been tested with various inputs to evaluate whether the model generalizes well to unseen data?_
- _Is the model robust enough for the problem? Do small perturbations (changes) in training data or the input space greatly affect the results?_
- _Can results found from the model be trusted?_

Justification

In this section, your model's final solution and its results should be compared to the benchmark you established earlier in the project using some type of statistical analysis. You should also justify whether these results and the solution are significant enough to have solved the problem posed in the project. Questions to ask yourself when writing this section:
- _Are the final results found stronger than the benchmark result reported earlier?_
- _Have you thoroughly analyzed and discussed the final solution?_
- _Is the final solution significant enough to have solved the problem?_

V. Conclusion

(approx. 1-2 pages)

Free-Form Visualization

In this section, you will need to provide some form of visualization that emphasizes an important quality about the project. It is much more free-form, but should reasonably support a significant result or characteristic about the problem that you want to discuss. Questions to ask yourself when writing this section:
- _Have you visualized a relevant or important quality about the problem, dataset, input data, or results?_
- _Is the visualization thoroughly analyzed and discussed?_
- _If a plot is provided, are the axes, title, and datum clearly defined?_

Reflection

In this section, you will summarize the entire end-to-end problem solution and discuss one or two particular aspects of the project you found interesting or difficult. You are expected to reflect on the project as a whole to show that you have a firm understanding of the entire process employed in your work. Questions to ask yourself when writing this section:
- _Have you thoroughly summarized the entire process you used for this project?_

- _Were there any interesting aspects of the project?_
- _Were there any difficult aspects of the project?_
- _Does the final model and solution fit your expectations for the problem, and should it be used in a general setting to solve these types of problems?_

Improvement

Explicar las soluciones ganadoras de kaggle y la diferencia en calidad-complejidad con mi solución.

Por otro lado, un objetivo secundario era crear la ConvNet y no reutilizar modelos preentrenados o implementados.

In this section, you will need to provide discussion as to how one aspect of the implementation you designed could be improved. As an example, consider ways your implementation can be made more general, and what would need to be modified. You do not need to make this improvement, but the potential solutions resulting from these changes are considered and compared/contrasted to your current solution. Questions to ask yourself when writing this section:
- _Are there further improvements that could be made on the algorithms or techniques you used in this project?_
- _Were there algorithms or techniques you researched that you did not know how to implement, but would consider using if you knew how?_
- _If you used your final solution as the new benchmark, do you think an even better solution exists?_

Before submitting, ask yourself. . .

- Does the project report you've written follow a well-organized structure similar to that of the project template?
- Is each section (particularly **Analysis** and **Methodology**) written in a clear, concise and specific fashion? Are there any ambiguous terms or phrases that need clarification?
- Would the intended audience of your project be able to understand your analysis, methods, and results?
- Have you properly proof-read your project report to assure there are minimal grammatical and spelling mistakes?
- Are all the resources used for this project correctly cited and referenced?
- Is the code that implements your solution easily readable and properly commented?
- Does the code execute without error and produce results similar to those reported?

If it's noted in your report that the training time is unrealistically long, such that it is not something that would be considered an "online" implementation, then we do not expect the reviewers to run your code in

that regard. A good way to compromise on that, would be to try pickling[4] your data after training, if possible. This would, in some sense, save the trained algorithm so that you could start the reviewer "after the training". I would personally suggest trying to avoid training times that approach more than a few minutes, if only because it then becomes a serious chore if something goes wrong and you have to re-train. The time required to do so seems unnecessary.