# MDI341 - Machine Learning Avancé

## Challenge Report

**Sami Bargaoui**

April 23, 2017

## 1 Context of the challenge

For face recognition, researchers often build 'templates' (a vector of 128 float values) from high resolution images. These templates are useful in certain applications where only low resolution images can be acquired and/or there is only low processing power (cpu/gpu) available.

The goal of this challenge is to learn a predictive system which predicts templates from low resolution images. The predicted templates should be as close as possible to original templates that were built from high resolution images.

The properties of the dataset are as follows:

- Training data:
  - $\mathbf{X}_{train}$: size 100000 x 2304. Each row of this matrix contains a low-resolution image. There are 100000 images where each image is of size 48x48=2304.
  - $\mathbf{Y}_{train}$: size 100000 x 128. Each row of this matrix contains a template of size 128, which is previously learned from a high-resolution image
- Validation data:
  - $\mathbf{X}_{valid}$: size 10000 x 2304. There are 10000 images in this dataset.
  - $\mathbf{Y}_{valid}$: size 10000 x 128.
- Test data:
  - $\mathbf{X}_{test}$: size 10000 x 2304. There are 10000 images in this dataset similar to the validation set.
  - $\mathbf{Y}_{test}$: size 10000 x 128. (You do not have access to this matrix)

The goal is to build a model, which would produce templates for the test data, given $\mathbf{X}_{train}$, $\mathbf{Y}_{train}$, $\mathbf{X}_{valid}$, $\mathbf{Y}_{valid}$, and $\mathbf{X}_{test}$.

Let us call the prediction of the model: $\hat{\mathbf{Y}}_{test}$. The performance criterion is given as follows:

$$\text{score} = \frac{1}{N} \sum_{i=1}^{10000} \sum_{j=1}^{128} \Big( \mathbf{Y}_{test}(i,j) - \hat{\mathbf{Y}}_{test}(i,j) \Big)^2$$

where $N$ denotes the total number of elements in $\mathbf{Y}_{test}$, such that $N = 128 \times 10000$.

The lower the score, the better the performance.

## 2 Preprocessing

The first step of preprocessing was to display the picture to see what we are dealing with :

**Figure 1:** Train Data in raw format

After a thorough investigation and research, I narrowed the preprocessing methods to the following :

### 2.1 Standard Scaling

The principle is to simply scale are out data along : Center to the mean and component wise scale to unit variance.

### 2.2 Gaussian Filtering or Denoising

Gaussian filtering is done by convolving each point in the input array with a Gaussian kernel and then summing them all to produce the output array. It is used to denoise pictures by applying the filter.

Here is our data sample after applying the filer :

**Figure 2:** Train Data after denoising

### 2.3 Histogram Equalization or Contrast Enhancement

It is a method that improves the contrast in an image, in order to stretch out the intensity range. 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 spread over the whole range.

**Figure 3:** Train Data after contrasting

### 2.4 Morphological Operations

The principle of this set of transformations is to process images based on shapes. Morphological operations apply a structuring element to an input image and generate an output image.

The most basic morphological operations are two: Erosion and Dilation. They have a wide array of uses, i.e. :

- Removing noise
- Isolation of individual elements and joining disparate elements in an image.
- Finding of intensity bumps or holes in an image

Here are some examples :

**Figure 4:** Train Data after erosion

**Figure 5:** Train Data after dilation

In the next section, depending on the score, we will expose the retained preprocessing solutions.

## 3 Neural Networks

### 3.1 Fully connected one pixel network

Now after processing the data , it is time to look for the best fitting model to our problem. Since it is a challenge dealing with images as data, I chose to work with neural networks, on both Tensorflow and Keras :

The idea was to establish a benchmark to compare to. In fact, this test is computationally expensive based on the whole train set. At another point, one idea I considered was that each algorithm depends on a set of hyperparameters which made the task harder.

The first benchmark established was using a relatively easy neural network (no convolutions, layers of one pixel out filters, ReLu activation layers). This solution provided my first bettered initial submit.

Next, I moved to a more complex and better performing neural network architecture.

### 3.2 Convolutional Neural Network

While building a more complex architecture, I based myself of many articles, and after many tries, I settled on using this architecture :

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 48, 48, 16)        160
_________________________________________________________________
p_re_lu_1 (PReLU)            (None, 48, 48, 16)        16
_________________________________________________________________
dropout_1 (Dropout)          (None, 48, 48, 16)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 24, 24, 16)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 24, 24, 32)        4640
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 24, 24, 32)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 24, 24, 32)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 12, 12, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 12, 12, 40)        11560
_________________________________________________________________
p_re_lu_2 (PReLU)            (None, 12, 12, 40)        40
_________________________________________________________________
dropout_3 (Dropout)          (None, 12, 12, 40)        0
```

```
---------------------------------------------------------------
max_pooling2d_3 (MaxPooling2 (None, 6, 6, 40)          0
---------------------------------------------------------------
conv2d_4 (Conv2D)            (None, 3, 3, 64)          23104
---------------------------------------------------------------
leaky_re_lu_2 (LeakyReLU)    (None, 3, 3, 64)          0
---------------------------------------------------------------
dropout_4 (Dropout)          (None, 3, 3, 64)          0
---------------------------------------------------------------
max_pooling2d_4 (MaxPooling2 (None, 1, 1, 64)          0
---------------------------------------------------------------
dense_1 (Dense)              (None, 1, 1, 128)         8320
---------------------------------------------------------------
flatten_1 (Flatten)          (None, 128)               0
===============================================================
Total params: 47,840.0
Trainable params: 47,840.0
Non-trainable params: 0.0
---------------------------------------------------------------
```

A detailed architecture diagram in available in Appendix A.

## 3.3 Impact of the preprocessing treatment

In the previously exposed methods above, the best performing one, as this following graph exposes, was a mixture between standard scaling and grey dilation :

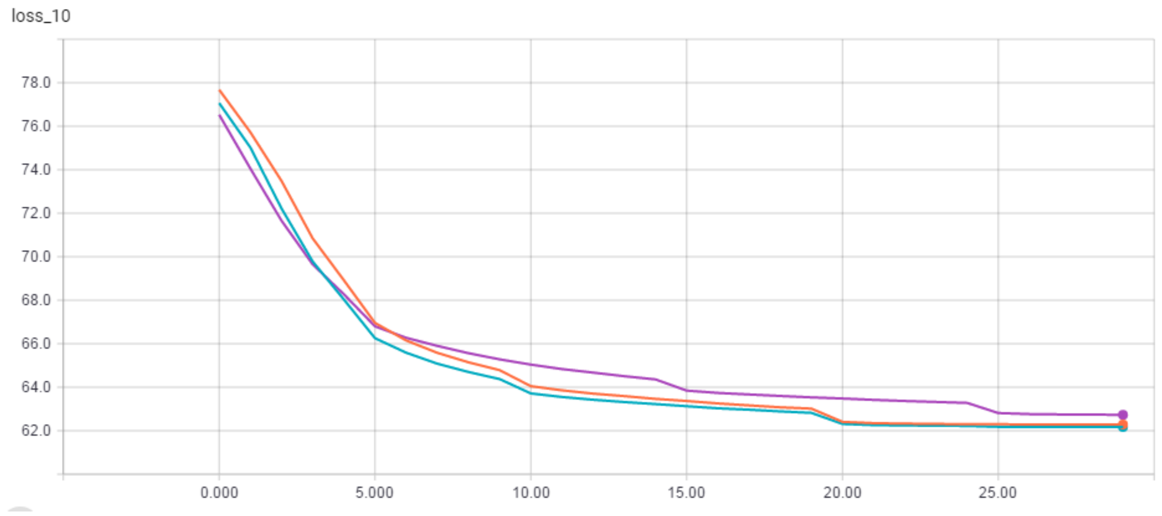


**Figure 6:** Impact of preprocessing methods on the train set

The best performing (from a loss on the validation set perspective) is the blue one corresponding to the dilation and standard scaling routines, compared to other explored solutions (denoising, constrast enhancement, etc). I kept to work with this method for the following of the challenge.

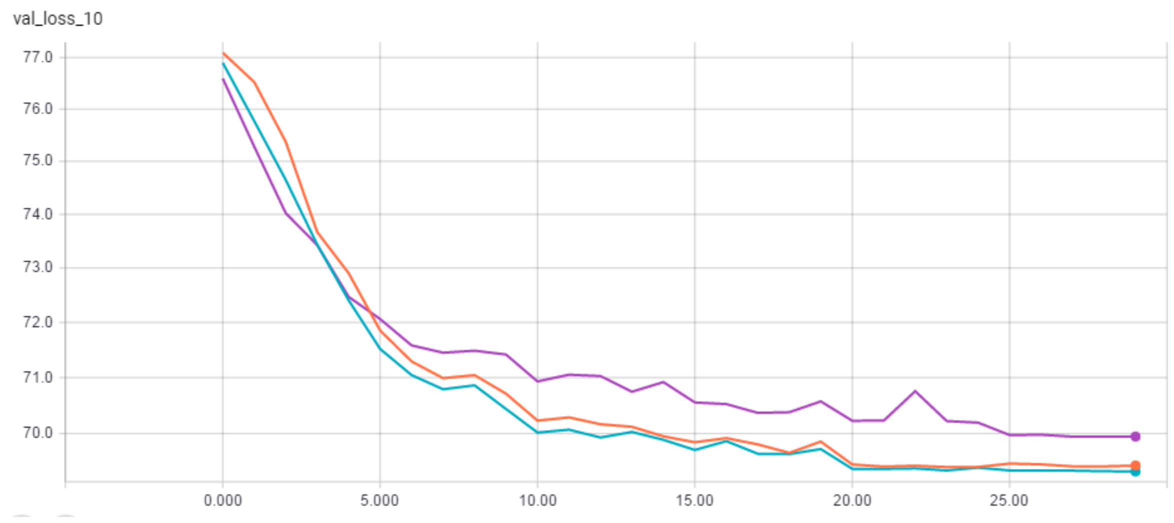


**Figure 7:** Impact of preprocessing methods on the validation set

**Figure 8:** Preprocessed data as fed to the network

### 3.4 Impact of the optimization algorithm

From an algorithmic standpoint, every neural network has to go by an optimization step, whether the algorithm is or the architecture. That is why I chose to explore different algorithms : mainly SGD (Stochastic Gradient Descent) here in purple, ADAM Solver in orange and DeltaGrad in cyan.

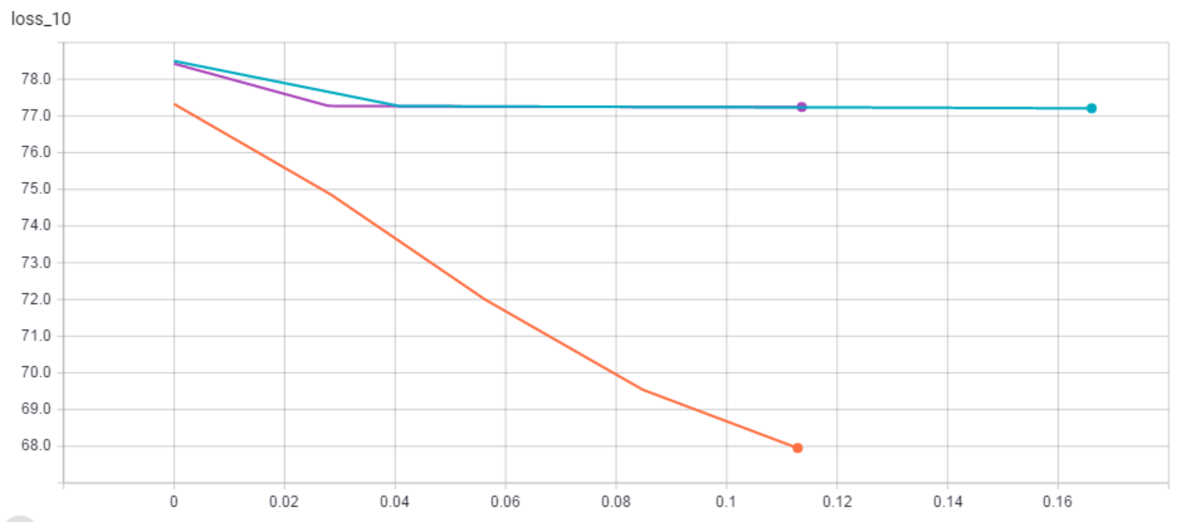


**Figure 9:** Impact of optimizers on the train set

From a loss and time standpoint, ADAM outperformed the other optimizers, that is why I chose to go further with this solution.

### 3.5 Other explored and improvement possibilities

Besides the exposed solutions, some of the other possibilities were different architectures, mainly adding a fully connected layer in the middle of the current architecture. Somehow, it seemed to slow down the whole network without giving better results, so I discarded this solution.

One of two solutions that I really tried to explore was adding BatchNormalization layers. These layers normalize the output each time before feeding it to the next layers. This move somehow got my score worse and gave a bigger error on both the train and validation datasets. The other solution was to try different activations : Sigmoid, Tanh, Softmax. The litterature recommended some activation function for specific tasks (muti-output classification, soft voting, etc. I found that ReLu and advanced ReLu functions performed the best with the chosen architecture.

For improving the solution, I really wanted to find optimal hyperparameters for the network. I found wrappers to use bayesian hyperparameter tuning for neural networks, but sadly I didn't had enough time to use these packages. I also wanted to try to run my solution for longer Epochs and try data augmentation techniques besides of preprocessing, since some articles suggest that data augmentation is a real helplful method to use when using image data in deep learning problems. I also thought at trying to use a ResNet insted of a ConvNet, looking for the available solutions to use such network.

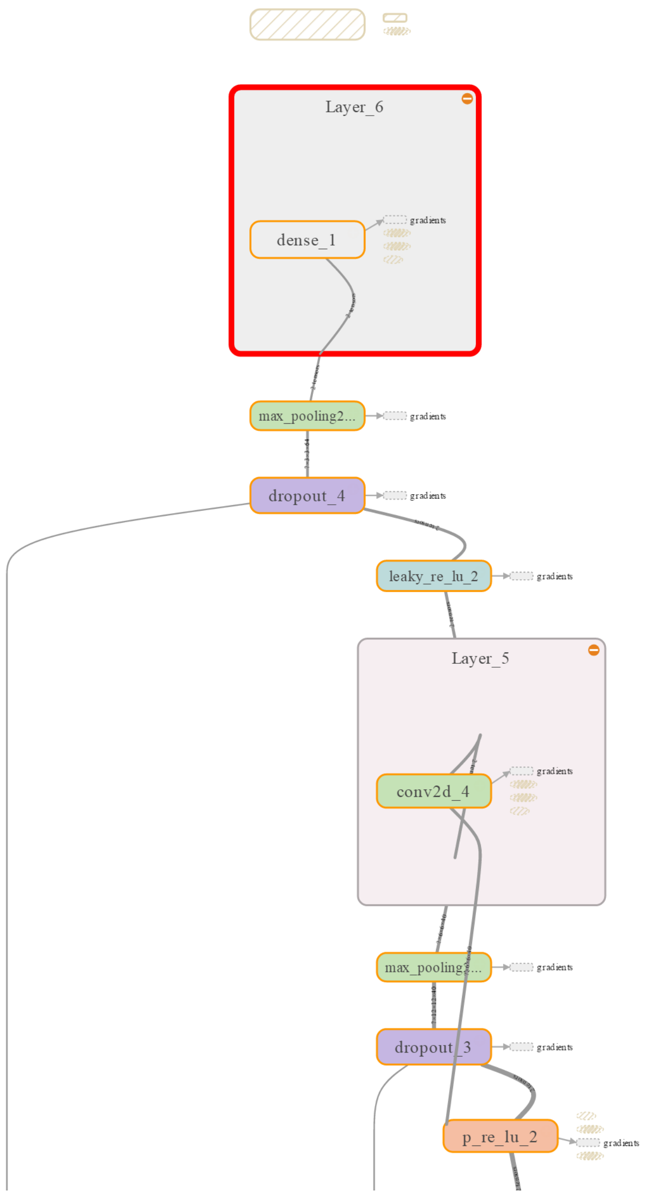
## 4 Conclusion

The challenge was really a chance to get a real glimpse and get hands on on a problem of deep learning and trying to extract useful information and patterns with the training data. This was a great opportunity to get accustomed to deep learning infrastructures like Tensorflow, Keras, TfLearn, etc.

The project allowed me to comprehend the methodology and the process behind selecting the algorithm based on the theoretical knowledge that we did receive in class and lab sessions, trying to establish a benchmark to compare to each time, and most certainly develop my skills in machine learning and especially have a better understanding about neural network, convolutions and deep learning algorithms.

# Appendices

## A CNN Architecture Graph



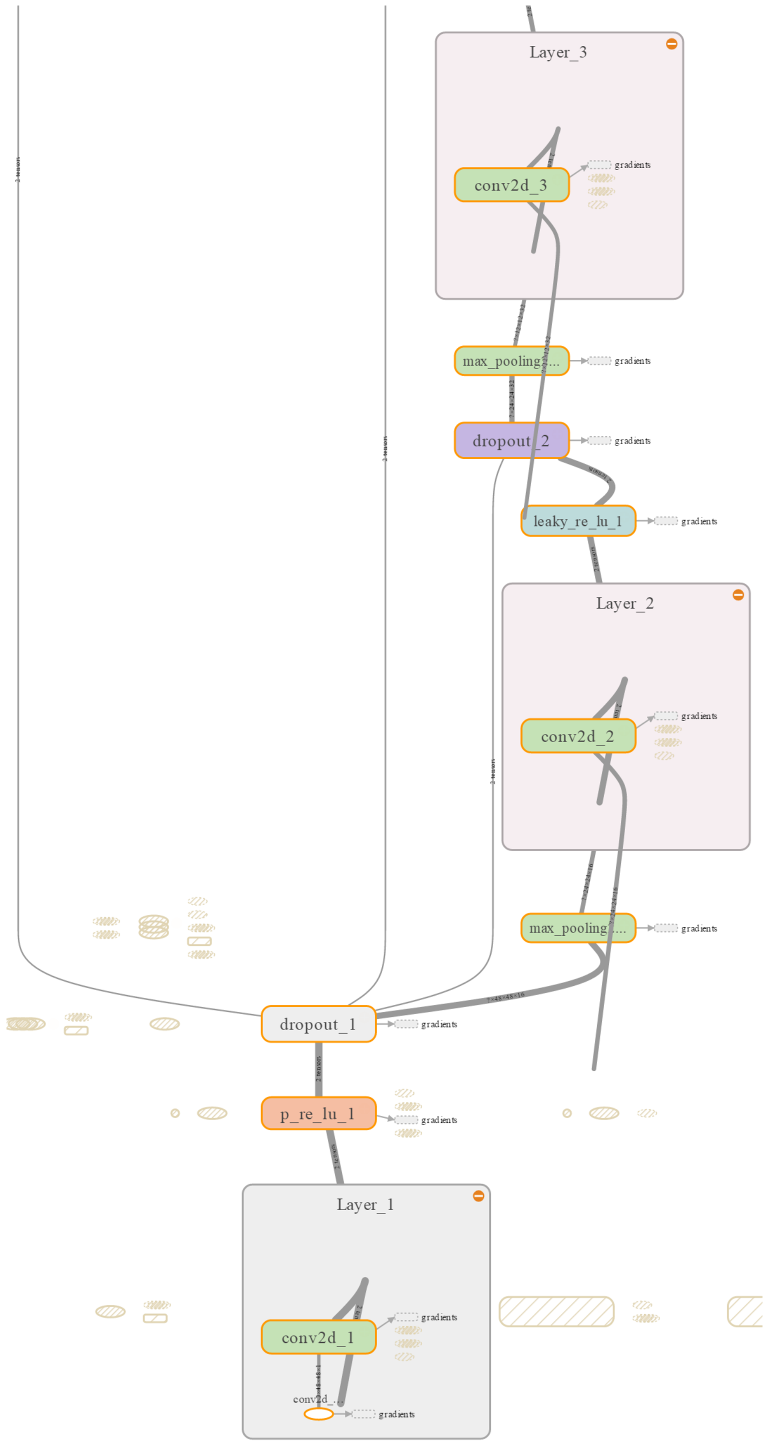


**Figure 10:** Architecture Diagram