**Problem statement:** Analyzing changes in evolving data fromadvanced habitation systems.

The goal of this task is to create a framework that monitors and provides change detection in a multimodal system. **Our objective** is two-fold (a) To provide the mining of patterns from older data as changes in data could reflect long term/ previous trends. (b) Mining for patterns over near data – as recent changes in data could indicate the recurrences of the previously known pattern or an upcoming event.

**Reported outcomes:** An exhaustive survey of related datasets that capture multi-modal environments/systems.

**Specific Aims:**

1. Explore some of the TensorFlow capabilities
   1. Environments
   2. Exploratory Data Analysis
   3. Create neural network architectures
   4. Create convolution neural network architectures
2. Create a CNN architecture to do a multi-feature extraction and high-level representation of the data

**Key Accomplishments:**

1. Getting familiar with machine learning on TensorFlow
2. Creating a CNN architecture on TensorFlow that is very close of one from a paper, which seems to be promising

**Red Flags:**

1. TensorFlow has some problems when running on local Windows machines
2. The final CNN that is shown last is not complete nor completely loyal to the paper yet
3. Takes a long time to go through all the 200 epochs, which was proposed by the paper, on a local Macintosh machine

**Future Work:**

1. Finish building the CNN architecture
2. Try to optimize the code to train faster, if necessary

**Timeline (tentative timeline for the upcoming week)**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Future Specific aims** | **10/09** | **10/10** | **10/11** | **10/14** | **10/15** | **10/16** |
| Building the CNN architecture |  |  |  |  |  |  |
| Analysing results |  |  |  |  |  |  |
| Optmizing code, if necessary |  |  |  |  |  |  |

**References:**

[1] Xiang Li, Wei Zhang, Qian Ding (2018). Deep learning-based remaining useful life estimation of bearings using multi-scale feature extraction

[2] Keras Documentation. https://keras.io/

**Appendix A**

**Results**

First, some environments were tested, since some problems have been noticed in the past when working with Windows machines. Environments tested are: jupyter notebook on Macintosh machine, jupyter notebook on Windows machine, PyCharm on Macintosh machine, PyCharm on Windows machine, Google Colaboratory on Macintosh machine and Google Colaboratory on Windows machine.

The results were good on all cases in the Macintosh machine. In the Windows machine there were decreasing problems as the experiments migrated from the local machine to full cloud applications, which had no problems. The source of such problems still unknown.

Second, some tutorials and examples were followed.

Since this first part is not much of interest, just two examples will be talked about.

First example is of a neural network on a regression problem. It tries to predict fuel efficiency of early automobiles using the Auto MPG Dataset. The few unknown values were dropped out of the dataset for simplification purposes. The Origin feature was converted into a one-hot vector. 80% of the data was used as training and the rest, as testing.

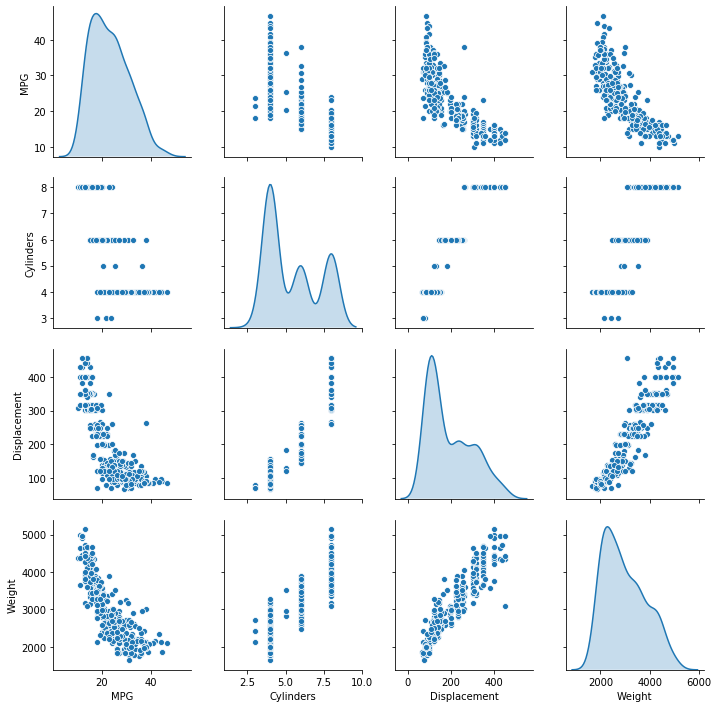


Fig 1 – Exploratory Data Analisys

On Fig 1 it is shown a feature in term of another, of all the features that should have some form of correlation. We can see that there is a correlation between the features, as expected.

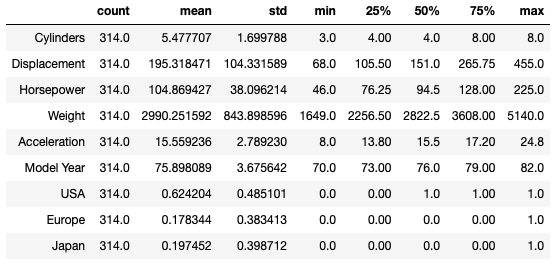


Fig 2 – Probability distribution of the features

Looking at the table, it can be observed that the data is varying from a feature to another. To obtain better results, normalization of the data was done. The normalization was done in this way: (x – mean(x))/std(x), were x is a feature.

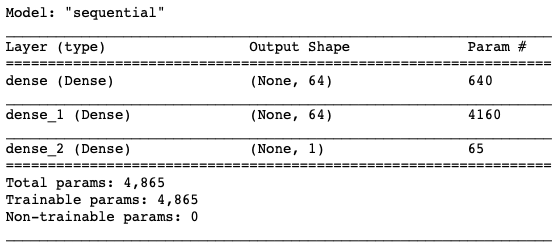


Fig 3 – Neural network architecture

Fig 3 shows how the neural network is setup. The input feeds to a fully connected layer of 64 neurons, which feeds to another fully connected layer of 64 neurons – both having a rectified linear unit as an activation function – and it all condenses as one output layer with a single, continuous value.

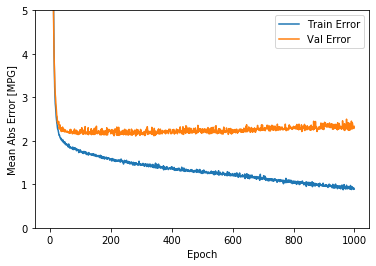


Fig 4 – Mean Absolute Error for the training data and for the validation data (1000 epochs)

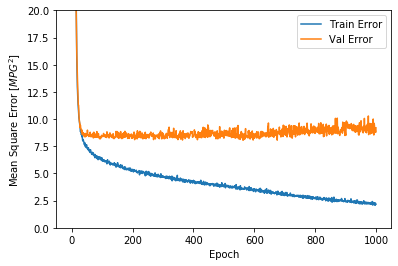


Fig 5 – Mean Square Error for the training data and for the validation data (1000 epochs)

As validation data shows, the model did not get much better after 100 epochs. It can be said that there was even some level of degradation of the model.

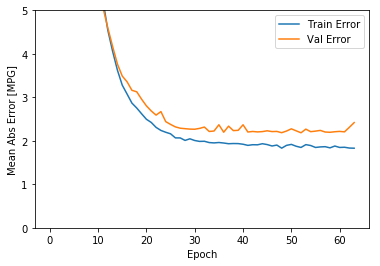


Fig 6 – Mean Absolute Error for the training data and for the validation data (1000 epochs, early\_stop activated)

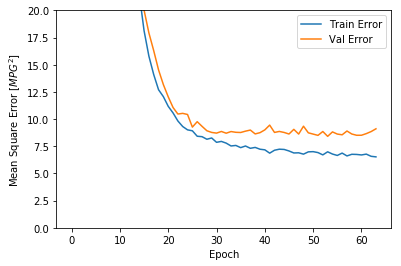


Fig 7 – Mean Square Error for the training data and for the validation data (1000 epochs, early\_stop activated)

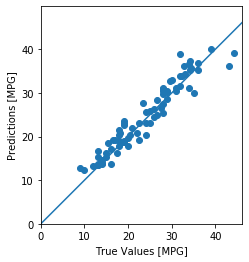


Fig 8 – Prediction of the regression model

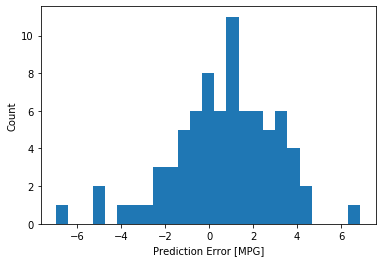


Fig 9 – Distribution of prediction error

As previously observed, the model does a good job until somewhere near the 60th epoch, when it starts to degrade, and its training is automatically stopped.

The prediction and the prediction error plots are made with the model being stopped before the 70th epoch.

It does a good job, as expected from the training and test error values and as shown in Fig 8. The distribution of prediction error is close to a Gaussian. A bigger dataset should confirm that.

Next, a convolutional neural network to classify CIFAR images. This dataset has 60,000 color images in 10 distinct classes without any ovelaping, with 6,000 images each class. It is divided into 50,000 training images and 10,000 testing images.

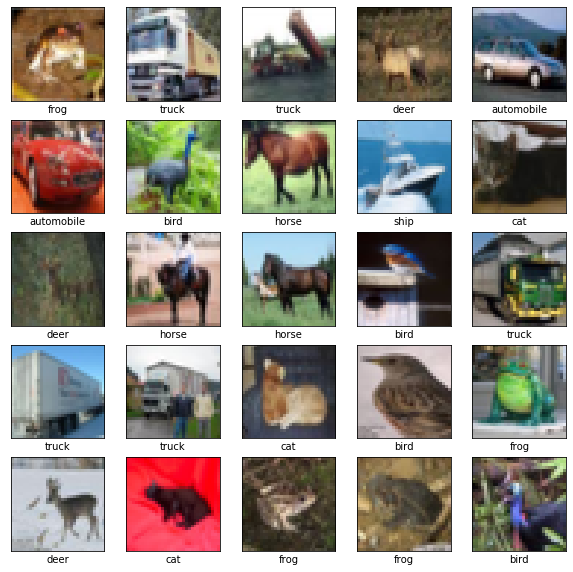


Fig 10 – Some samples from the training set, along with its respective label

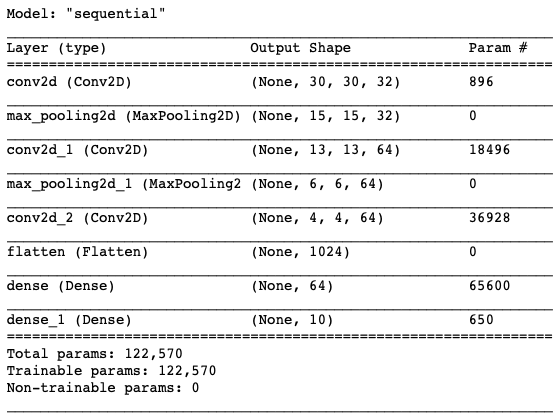


Fig 11 – The convolution neural network architecture

The CNN architecture consists of a convolution layer of 32 neurons with a rectified linear unit (ReLU) activation function, that feeds into a pooling layer of 2 by 2 (halves both the number of columns and the number of rows), repeats the same process again but the convolution having 64 neurons this time. After the second pooling, another convolution layer of 64 neurons with a ReLU activation function is performed, the results are flattened and fed to a last neural network with 64 neurons with a ReLU activation function, before being fed to an output layer of 10 neurons with a softmax activation function.

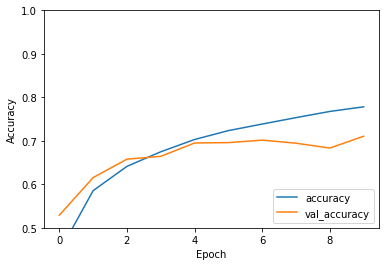


Fig 12 – Accuracy of the training and of the testing, after each epoch

The number of epochs was set to 10 and it seems to be a reasonable number, as the testing accuracy value starts to diverge from the training accuracy, indicating a good stopping point. The final test accuracy was measured as 71.05%

More analysis could be done on this model, but it was time to move on to the next architecture, which is of bigger interest.

Lastly, the CNN as proposed in the “Deep learning-based remaining useful life estimation of bearings using multi-scale feature extraction” paper.

The data used to train and test the network is the fashion\_mnist from the tensorflow dataset for simplification and testing purposes. The main goal now is to get a working CNN architecture that is as close as the one described on the paper as possible. This dataset is composed of 70,000 images, each having 28x28 grayscale pixels. From those images, 60,000 are used as training and 10,000 are used as testing. The dataset is equally distributed among 10 non-overlapping and distinct classes; each image has its label and will be normalized before going into the model training and testing.

As for now, the multi-scale feature extractor and the high-level representation are implemented, but due to some errors in the code, it is not working yet. The architecture is as follows:

1. The input image goes through a pooling layer with stride (1, 2) to reduce the dimension while keeping the significant features.
2. Three identical convolutional layers with filter size (3, 3) and filter number 5 – each followed by a dropout layer with rate of 0.5 – are adopted for feature extraction.
3. The results from each dropout layer are concatenated together and then fed to a convolution layer and a dropout layer that are the same as the ones before for the high-level representation.
4. The results of the final layer are flattened and presented as the output in order to do the training of the model, which was done with these parameters:

* Adam optimizer algorithm
* Mean absolute error as the loss function
* Each epoch is trained using mini-batches of 32 samples
* 200 epochs
* Two metrics are used to evaluate the model performance: mean absolute percentage error and root mean square error

The training of the flawed architecture was almost 3.5 hours long, with about one minute/epoch. The training time of the proposed model, as is, after the due corrections, are not expected to change too much.

It was observed that the model might not need too much epochs to reach an optimal state. Further investigation will be carried.