Towards a Data Science and Engineering Club at UAB

Diego Velazquez Dorta

Resumen La contratación de profesionales especializados en Machine Learning es una cecesidad que se hace cada vez más presente para las empresas. Por lo tanto, alentar a los jóvenes estudiantes a aprender este conjunto de habilidades en particular debería ser una prioridad en todos los campus universitarios. El objetivo de este Trabajo de Fin de Grado es la creación de una plataforma donde los estudiantes de la UAB puedan participar en competiciones de Machine Learning usando datos reales y resolviendo, potencialmente, problemas que las empresas publiquen en dicha plataforma. A lo largo del desarrollo del trabajo se han ido agregando blogs a la plataforma que sirven como introducción a métodos punteros de análisis de datos y Machine Learning.

Palabras clave Machine Learning, Deep Learning, Análisis de Datos.

Abstract The hiring of specialized professionals in Machine Learning has been and still is an increasing necessity for companies. Therefore encouraging young students to learn this particular set of skills should be a priority in every university campus. The goal of this Project is the creation of a platform where UAB students can participate in Machine Learning competitions using real data and potentially solving problems that companies publish on said platform. Throughout the development of the project a blog has been created that serves as an introduction to leading methods of data analysis and Machine Learning.

Keywords Machine Learning, Deep Learning, Data Analysis

1 INTRODUCTION

MACHINE Learning becomes more relevant in every aspect of our lives each passing day. This results in an exponentially growing need for companies (especially in Europe) to hire specialized professionals in this field. The problem is that there are not many qualified professionals available to cover this need.
The first goal of this work is the development of competitions in data engineering using CodaLab[1]. Different engineers can participate in the competitions, using their models to solve a given problem using Machine Learning. Companies with the need to hire Engineers specialized in Machine Learning can propose their own competitions and reward the winners. The topics of the competitions will be restricted to the sectors of companies in the UAB environment, HUB30 [2]. In addition, a blog will be developed as a learning tool, presenting popular Machine Learning methods and techniques, evidencing their limitations and advantages when applied to specific datasets. With the creation of the competition platform and the blog, we expect to propel forward the creation of a Data Science and Engineering Club at UAB.

• E-mail de contacte: diegoalejandro.velazquez@e-campus.uab.cat
• Menció realitzada: Computació
• Treball tutoritzat per: Jordi Gonzalez (Ciencias de la Computación)
• Curs 2018/19

[1]https://competitions.codalab.org/
[2]https://www.uab.cat/web/hub-b30-1345754064093.html

2 SELECTING THE DATASETS

The databases that will be analyzed in each of the blogs have been selected. They act as an introduction to each of the main Machine Learning concepts needed to address almost any type of problem that can be solved with these methods. We have selected the following datasets:

1. Cryptocurrency Historical Prices[1]: Predicting the Bitcoin Market Price. This dataset contains information regarding the state of the cryptocurency market over the course of the last 5 years. The problem to be solved using this dataset is the prediction of Bitcoin value in USD, given the current state of the market. This dataset is composed of 2921 data points. We selected $10\,\%$ of the samples for both the validation and test set and the rest for training.

2. Game of Thrones[2]: Predicting Deaths in Game of Thrones. The goal of this dataset is to predict whether a Game of Thrones character will die. The dataset is composed of 1946 samples, each of with comprises 33 features, containing information of multiple characters from the show. We selected 20 % of the data points to evaluate our models.

3. MNIST[3, 4]: Handwritten digit recognition. The goal of this dataset is to identify handwritten digits ranging from 0 to 9, presented in binary images. This dataset consists (as we have modified it) of 47995 in the training set, 12005 in the validation set and 10000 in the test set. The images are stored in numpy serializables files (.npy).

4. Lesion Diagnosis[5]: Classifying skin lesions using dermatoscopic images. The goal of this dataset is to automate the diagnosis of different skin lesions using fine-tuning[6]. This dataset contains 10015 images belonging to one of 7 categories. We selected 20 % of the images present in each class for validation and used the rest for trai ning.

We chose the first 2 datasets to show how to perform a thorough analysis of temporal series, cross-sectional data and model performance, using techniques like *correlation matrices*, *feature extraction*, *confusion matrices*, *cross validation* and many more. The MNIST dataset was chosen because it is a well-known dataset used as a *baseline* for Deep Learning models. In the blog, we use it to provide an introduction to Convolutional Neural Networks. Finally, we chose the Lesion Diagnosis dataset in order to introduce the concept of fine-tuning in a fine-grained classification problem. Which is currently a very common approach used in Deep Learning.

3 Methodology of the Posts

Blog Posts have been created[7], detailing how to perform a deep analysis over a dataset, whether it is for classification or regression. The posts also contain an introduction to key Machine Learning concepts, serving as a guide for the reader to tackle Machine Learning problems.
What follows is a list of some of the currently existing blogs as well as an overview of their content. All of the blogs are written using Python 3.6[8] inside Jupyter-Notebook[9].

3.1. Cryptocurrency Historical Prices

In this blog post[10] we use dataset[1] to predict the price of Bitcoins using regression and data analysis techniques. This post explains how to build and evaluate a regressor and how to analyse your data.

3.1.1. Data Analysis and Preprocessing

We start by introducing the concept of feature correlation, explaining how it can help us discard useless or redundant features and keep the ones that contribute the most to the learning process of our model. We show how to visualize the correlations between your features plotting a confusion matrix as a heat map. There are multiple ways to calculate the correlation between 2 variables. We decided to use the Pearson Correlation Coefficient. Analysing the resulting correlation matrix, seen in Figure 1b, we get rid of features whose correlation with our target variable is close to 0, since these do not influence our target variable much. We also remove redundant features. These are the ones whose correlation coefficient with all other variables is very similar, meaning that if we find a pair of redundant features, we can remove one them. Thus, reducing the redundancy present in our data. We also plot the evolution of some important features over time. Looking for some interesting insight into the workings of the Bitcoin Market, see Figure 1a. We can see that the more valuable Bitcoin becomes, the harder it is to mine it and the more daily transactions are performed. Notice the early spike in Daily Confirmed Transactions, halfway into 2010, when the Mining Difficulty was low because their Market Price and Market Cap were basically non-existent. The people who bought Bitcoins in these early transactions and sold them around the spike present halfway through 2011 and on 2014 are the ones that probably got rich with Bitcoin.

3.1.2. Applying Machine Learning

In this section we show how to build and fit a Linear Regressor and we also enumerate two metrics you can use to evaluate the performance of said model.

We start by building and fitting the regressor to our training data and proceed to evaluate it's performance on the validation set using two metrics: RMSE (Root Mean Squared Error) and coefficient of determination (R2 Score). We provide a brief insight into how these metrics work and what they represent.

3.1.3. Results

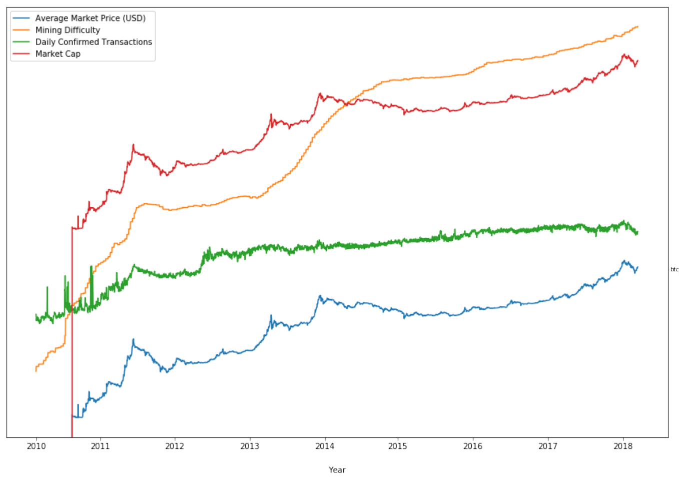
Finally we proceed to show the results obtained with our simple regressor. Looking at Figure 2a we can see that our regressor performs considerable well, specially on the predictions made for data appearing after the year 2014. Performing a deeper analysis using a residual plot, shown in Figure 2b, it becomes obvious that our model can be improved as indicated by the non-random pattern present in the projected data points on the left side of the plot.

3.2. Game of Thrones

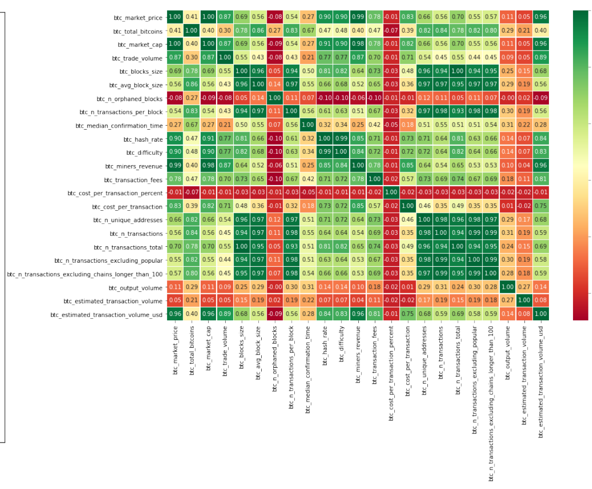
In this blog post[11] we set out to predict the fate of Game Of Thrones characters using Machine Learning. This post contains a deep analysis on the dataset[2], along with an analysis of the performance of many different Machine Learning models.

3.2.1. Data Analysis and Preprocessing

In this section we show that there are multiple ways of dealing with NAN values. Explaining that if you only have a couple of samples which contain a NAN value in a certain attribute, you might just be better off with dropping those samples entirely. But if there are too many samples that contain NANs you can't just drop them all. This is the case for this dataset. So we proceed to fill the NANs that

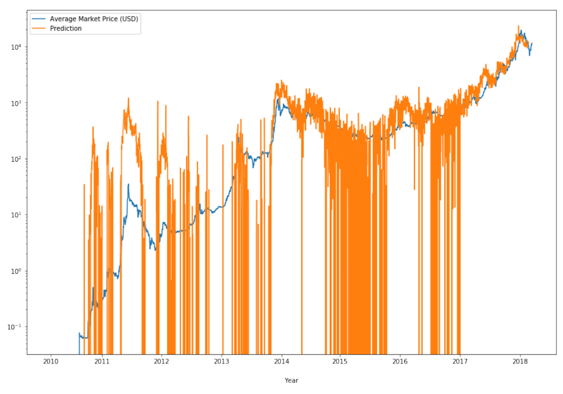


(a) Bitcoin Data over time

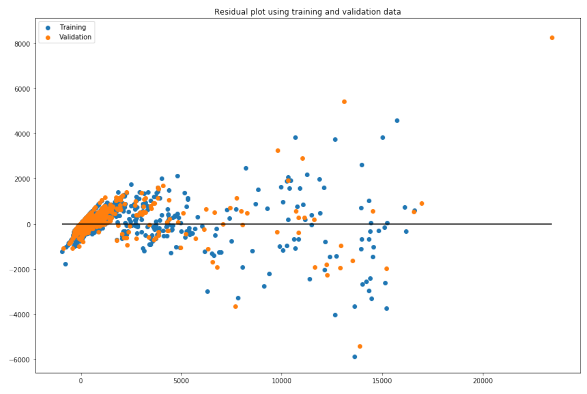


(b) Correlation Matrix

Fig. 1: Bitcoin Data Visualization



(a) Regressor prediction over entire dataset



(b) Residual Plot

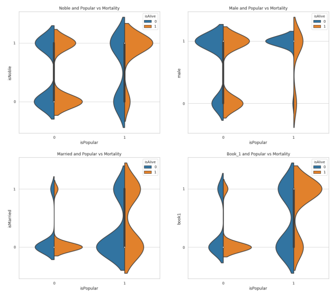
Fig. 2: Regression results

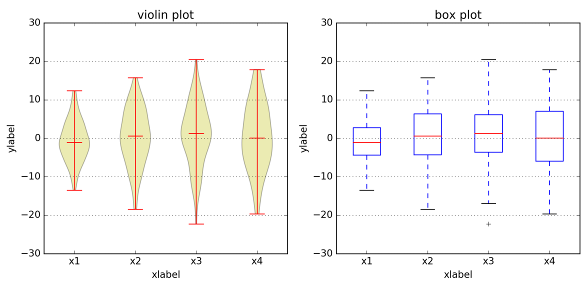
we can with the mean value of their columns and replace the rest with a -1 or empty string. We also show the usefulness of Violin plots and compare them with box plots. Showing how, since violin plots show the probability density of the data at different values, they are more informative than a plain box plot. In fact, as it can be seen in Figure 3, while a box plot only shows summary statistics such as mean/median and interquartile ranges, the violin plot shows the full distribution of the data. Afterwards, we show how to get rid of useless features in the data, like **S.No** which is basically an incremental identifier for each sample, and **name**, which is the name of the character. Since these features don't contribute at all to the prediction of a character's fate, we drop them entirely. Finally, we introduce the concept of one-hot encoding, using it to transform the categorical features in our dataset into vectors, making it possible for our models to use them in the learning process. The concept of factorizing it's also explained but not applied, since it is not the correct approach for this dataset.

3.2.2. Applying Machine Learning

The following section in the post explains different ways in which you can measure the performance of a classifier, along with the different metrics used in Machine Learning and what each of them represents.

We start by introducing the concept of splitting the data into train and test sets. The way in which you divide your data in Machine Learning is crucial when it comes to determining how well a model performs on it. It is imperative to make sure that our model performs well no matter how the data is partitioned. For example: suppose we have a model with one or more unknown parameters, and a dataset to which the model can be it (the training data set). The fitting process optimizes the model parameters to make the model fit the training data as well as possible. If we then take an independent sample of validation data from the same population as the training data, it will generally turn out that the model does not fit the validation data as well as it fits the training data. The size of this difference is likely to be large especially when the size of the training data set is small, or when the number of parameters in the model is large. Cross-validation is a way to estimate the size of this effect.
We split the data using cross validation and run many different models using 5 data splits. As it can be seen in Figure 4b we can select the most promising models by selecting the best accuracy distribution. To show the effects of hyper-parameter tuning we chose the best and the worst performing models, showing that with some tuning both models can perform almost equally well. We continue by introducing the concept of grid search and how it is used to algorithmically select the best hyper-parameters for a given


(a) Class Distribution around different features



(b) Violin Plot and Box Plot comparison

Fig. 3: Data Visualisation for the GOT dataset

model. We apply this algorithm to our selected models even though it is a brute force approach since if you have k hyper-parameters, and each one of them have c possible values. Then, performing grid search is basically taking a Cartesian product of these: $\prod_{c=1}^{k} ci$.

3.2.3. Results

In this section we visualize the performance of the classifiers and give an overview of the evaluation metrics used in Machine Learning. We plot a normalized confusion matrix for each of the models showing how well each classifier performs on each of the classes. As shown in Figure 4a, both of our classifiers perform much better on positive examples (alive characters) than on negative examples (dead characters). This is due to the fact that here are many more positive examples than negatives in the dataset. This class imbalance makes the model biased towards the positive class.
We analyze how well our models can distinguish between classes by plotting a ROC (Receiver Operating Characteristics) curve. As it can be seen in Figure 4c, both models have a fairly decent AUC (Area Under the Curve) at every threshold. Nonetheless, the Random Forest Classifier performs consistently better at lower thresholds. Lastly, after explaining the importance of different metrics such as: precision, recall and f1-score; we proceed to visualize the precision-recall trade-off every model suffers from, by plotting a PRC (precision recall curve). As seen in Figure 4d, our model behaves as expected, maintaining about 80 % precision and recall values at almost every threshold.

3.3. MNIST

In this post[12] we solve the famous MNIST dataset[4], describing widely use kind of neural network widely used in computer vision known as Convolutional Neural Networks (CNN). We introduce the reader to the concept of CNN teaching them how to implement one from scratch using the PyTorch library[13].

3.3.1. Data Analysis and Preprocessing

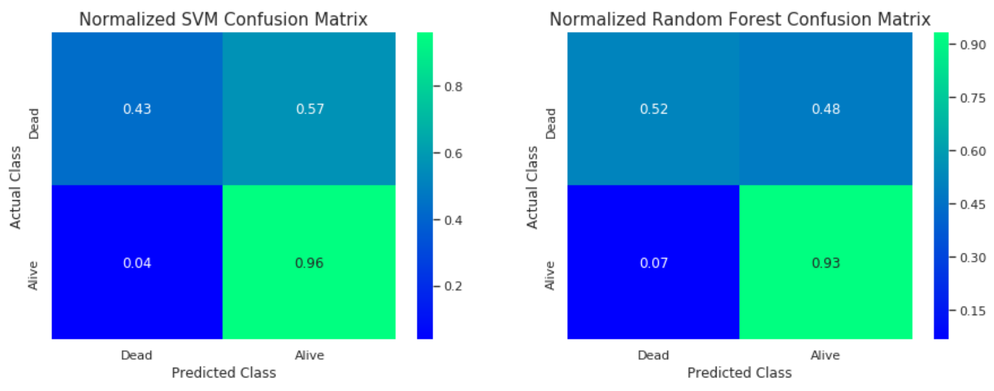
We start by introducing the reader to the concept of a Data Loader, as well as to the concept of Tensors and how images can be seen as such. Explaining that before feeding the images to our model it's always a good idea to preprocess them a bit. To do this, PyTorch[13] provides the *data.transforms* module.
Images are usually represented as a 3D array with a shape of $H \times W \times C$, containing RGB values ranging from 0 to 255. In this case since we are using binary images they have only 2 possible values: 0 or 255.
We apply one transformation to every image, changing their shape to $C \times H \times W$ and make the range of values between 0 and 1, or in our this particular case: 0 or 1. This transformation will also convert the images to tensors which is an array that can be stored in GPU. Once we have our data on disk we use PyTorch[13] *DataLoaders* to load them into memory using batches and shuffling them if needed, using several workers loading the data in parallel preventing this process from becoming a bottleneck.

3.3.2. Applying Machine Learning

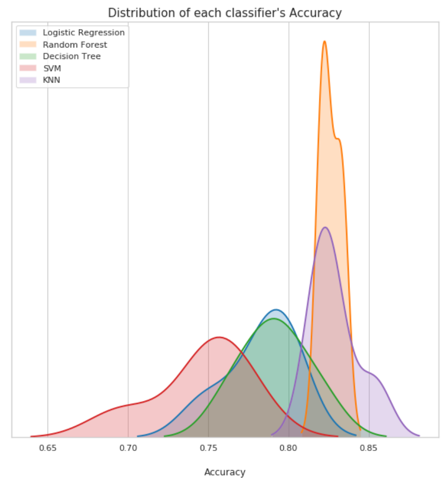
This blog has a more theoretical and introductory approach, so in the Machine Learning section we introduce the reader to some basic concepts, that are necessary to follow the code and explanations presented in the post.
Firtly we explain that Neural Networks are based on a collection of connected units or nodes called "neurons", which try to simulate, in a very simplistic manner, the way neurons interact in an animal's brain. Each connection, like the synapses in a biological brain, can transmit a signal from one artificial neuron to another. An artificial neuron that receives a signal can process it and then signal additional artificial neurons connected to it. Secondly we explain that CNNs are a kind of neural network that attempts to look at images the way the humans do, extract information from them and pass it to the artificial neurons tasked with handling this information in the fully connected layers. Lastly we show how the convolution filters work. Explaining how they look at the image progressively, using a sliding window and how this allows the network to focus on one piece of the image at a time, learning to extract the most important information from each portion of the image. This is similar to how the human brain looks at images. The human eye is mostly very low resolution, except for a tiny patch called the fovea. Though we feel as if we can see an entire scene in high resolution, this is an illusion created by our brain who stitches together several glimpses of small areas captured by the fovea.

3.3.3. Results

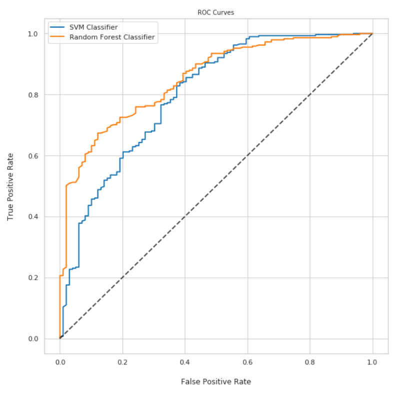
Lastly, we visualize each one of the activation maps generated by one of the convolutional layers of the trained



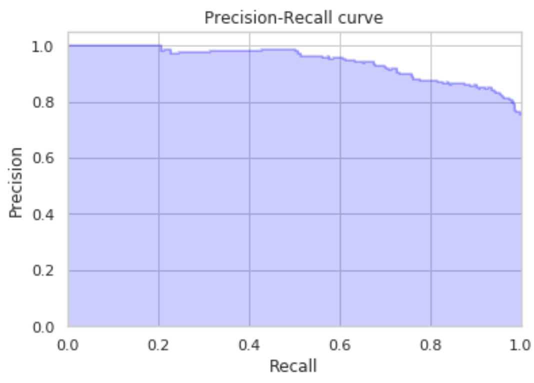
(a) Confusion Matrices of 2 of the evaluated models



(b) Accuracy Distribution obtained by all of the models evaluated using Cross-Validation



(c) ROC Curve for 2 of the evaluated classifiers



(d) Precision Recall Curve for Random Forest Classifier

Fig. 4: Different ways of evaluating a classifier

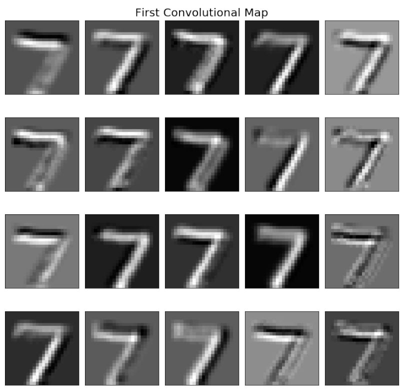
architecture. As it can be seen in Figure 5a the first convolutional layer clearly is specialized in detecting edges and curves in the numbers.
We also show the results obtained by our small and simple architecture using loss and accuracy values. As shown in Figure 5b, we achieve around 99 % accuracy on the validation set, which comes close to the state-of-the-art results[14] for the MNIST dataset.

3.4. Lesion Diagnosis

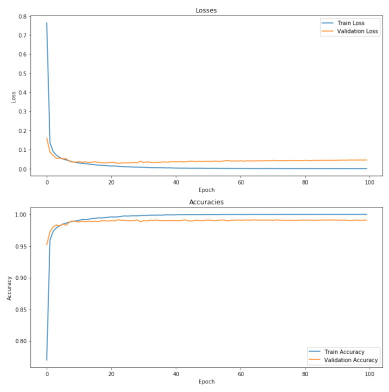
In this blog post[15] we introduce one of the many widely used techniques in Deep Learning and Computer Vision, known as fine-tuning.
We explain what fine-tuning is and how to fine-tune any CNN, applying this technique to the Skin Lesion Challenge[5] using a resnet18 architecture, shown in Figure 6). The training is done by using multiple GPUs using the *DataParalell* module provided by PyTorch[13].

3.4.1. Data Analysis and Preprocessing

In this blog post we introduce the concepts of Data Augmentation and Normalization applying these techniques to the images in the dataset. We flip images horizontally with a 50 % chance. Since on our dataset no information is lost by performing this operation, we effectively augment the amount of training images at our disposal.
We also crop a random section of the image and feed that to the network, this is a good technique to effectively aug-



(a) Activation Map of a CNN convolutional layer: The white specs in the images are the regions that the CNN is paying attention to.



(b) Results as loss and accuracy values

Fig. 5: Results on the MNIST dataset

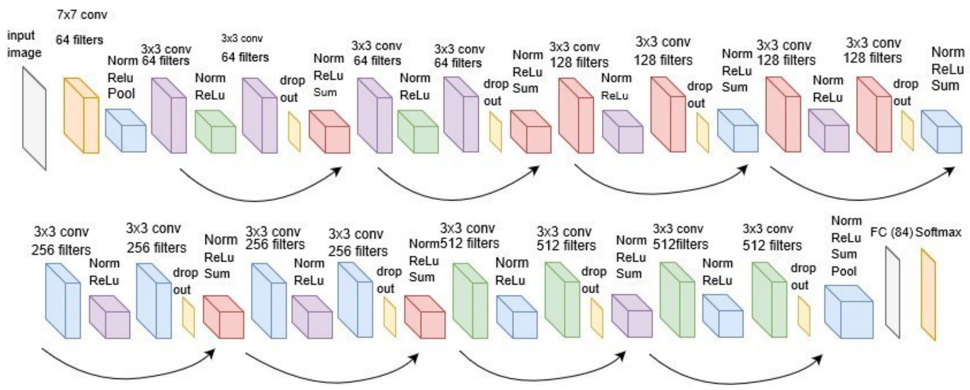


Fig. 6: Resnet18 Architecture

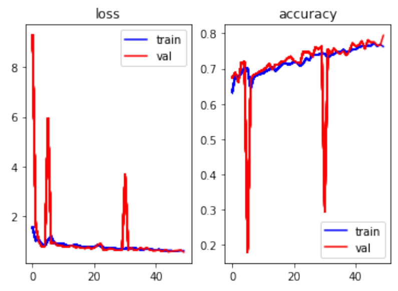


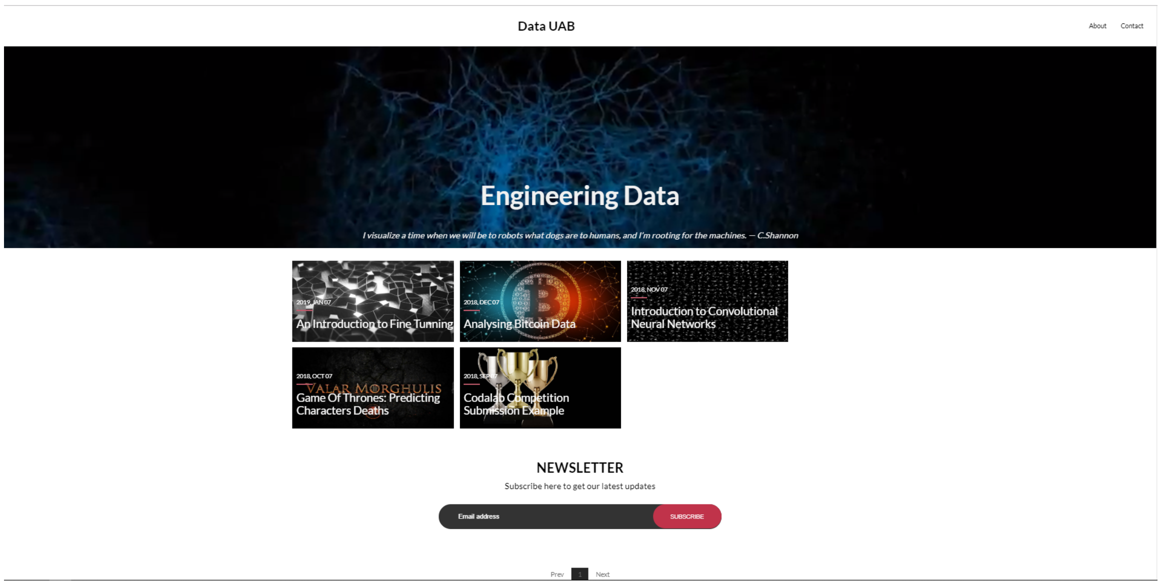
Fig. 7: Results obtained on the Resnet18 architecture

ment the amount of images in the training set. It also helps the network to generalize better since performing this operation adds more variance to the data.
Finally we re-scale and normalize our images due to the fact that the network must be fed 224×244 images and that the images with which the network was trained were normalized using the Imagenet[16] mean and standard deviation.

3.4.2. Applying Machine Learning

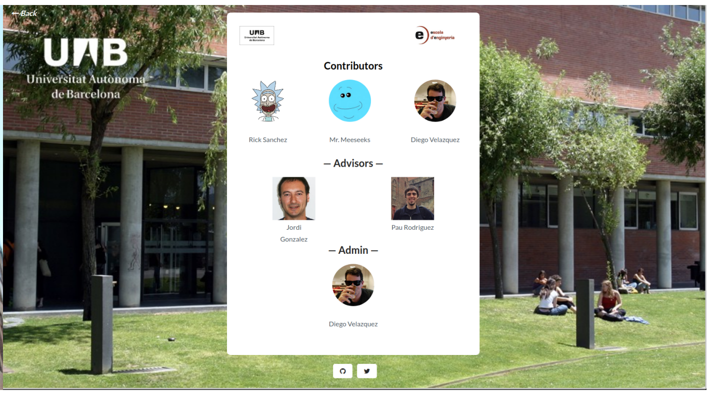
As the latter post (described above), this post is elaborated using a more theoretical approach so, in the Machine Learning section we walk the reader through some key concepts that are necessary to follow the rest of the blog post. We explain how since CNNs usually have a large number of parameters, often in the range of millions, training one of these architectures with a dataset that is significantly smaller than the number of parameters it has, will prevent it from generalizing, making it overfit the data. On top of that, training a network with millions of parameters from scratch takes a large amount of time. Fine-tuning bypass both of these problems by taking an already trained network, modifying it slightly and re-training it on new data. Avoiding overfitting and making the training significantly faster.
To fine-tune a CNN, the first thing you have to do is change only the last fully-connected layer of the network, replacing it with your own, having as many output units as your task requires. For example: a network trained with Imagenet will have 1000 neurons on the last layer, since Imagenet has 1000 classes. If our dataset has only 10 classes we just replace the last layer with one that has 10 neurons.
Once this is done Fine-tuning can be performed in mainly in 3 different ways:

- Freeze (no backprop) the entire network and train only the newly added layer with a normal learning rate.


Data UAB
About
Contact
Engineering Data
I visualize a time when we will be to robots what dogs are to humans, and I'm rooting for the machines. – C.Shannon
2019, JAN 07
An Introduction to Fine Tunning
2018, DEC 07
Analysing Bitcoin Data
2018, NOV 07
Introduction to Convolutional Neural Networks
2018, OCT 07
Game Of Thrones: Predicting Characters Deaths
Codalab Competition Submission Example
NEWSLETTER
Subscribe here to get our latest updates
Email address
SUBSCRIBE
Prev
Next

(a) Home page

(b) Contact forum



(c) About section

Fig. 8: Data UAB Website

- Freeze the first layers of the network and train the rest of the network with a smaller learning rate. Since the first convolutional layers always focus on edges and corners it is very likely that we don't have to change their weights because no matter what your image has in it, it will always have edges and corners. The newly added layer should be trained with a normal learning rate.

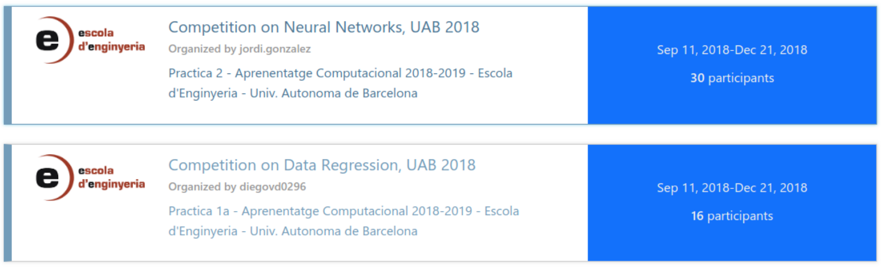
- Re-train the entire network with high learning rate. The previous approaches can work when the domain of the new task is similar to the one with which the network was pre-trained. But when the domain of the new task differs a lot from the original one, the network must re-learn a lot, thus requiring more aggressive parameter updates.

3.4.3. Results

We show the obtained results (Figure 7, reaching a 79,4 % accuracy. This result is not close to the state-of-the-art in the lesion diagnosis task for this dataset[17], which uses a model ensemble and meta learning techniques to classify each image. Lastly we conclude the post by introducing the concepts of learning rate annealing, random search[18] and early stopping, as well as by suggesting the use of a more powerful residual architecture[19]. Thus inviting the user to improve the obtained results.

4 The Platform *Data UAB*

As shown in the Figure 8a, posts are published on the blog that has been created[7] and which will serve as an archive for all publications made by students of the *UAB's Data Science and Engineering Club*. We used *mailchimp* to create a newsletter with the purpose of sending all the publications and/or news about the blog or the Data Science and Engineering Club to any subscriber. A contact forum has also been set up (see Figure 8b), where users can submit any proposal or question. Finally we added a section where all of the contributors to the blog or the platform will be credited, shown in Figure 8c.
Alongside the website, a *Github* repository[20] has been created. In it, you can find all of the code that has been used in each blog post. Readers can reuse this code in order to expand the concepts and reproduce the experiments present in each one of the posts. With the website and repository in place, a Data Science Club can be created in the UAB. The club members will be encouraged to participate in competitions within platforms such as *Kaggle* and *CodaLab*, representing the University. All members will be able to make


Competition on Neural Networks, UAB 2018
Organized by jordi.gonzalez
Practica 2 - Aprenentatge Computacional 2018-2019 - Escola d'Enginyeria - Univ. Autonoma de Barcelona
Sep 11, 2018-Dec 21, 2018
30 participants

Competition on Data Regression, UAB 2018
Organized by diegovd0296
Practica 1a - Aprenentatge Computacional 2018-2019 - Escola d'Enginyeria - Univ. Autonoma de Barcelona
Sep 11, 2018-Dec 21, 2018
16 participants

Fig. 9: Active Competitions in the CodaLab platform

posts and upload them to the blog[7], explaining the results obtained in the competitions or introducing any other concept/technique related to Machine Learning.
Two of the previously mentioned datasets, MNIST and Cryptocurrency Historical Prices, have been launched on the CodaLab platform. These competitions are open to the public and fully functional, as seen in Figure 9. Once a file is submitted the relevant metrics will be computed using CodaLab API[21]. The goal metrics for the competitions are R2 Score and Accuracy, respectively. We provide the participants with a labeled training set and an unlabeled training set. Participants must store their predictions in a *pickle* serializable file (.pkl) and upload them to the server, where the ground truth for the test set is stored. We provide a small sample code showing how to build and evaluate an extremely simple classifier to use in each competition. This sample code will also generate the submission file in the correct format for each competition. These code samples can be found at the Data UAB repository[20].

5 Conclusions

With the completion of this Bachelor's Project we have established a platform where UAB students have at their disposal a blog that will serve as an introduction, to cutting-edge methods of data analysis and Machine Learning. We have also managed, through the creation of the *UAB's Data Science and Engineering Club*, to give students the possibility of participating in competitions proposed by different companies and be rewarded. As a continuation of this work, using the *newsletter* and the platforms facilitated by the UAB we will continue to promote the participation of multiple teams in the *Kaggle* and *CodaLab* competitions and the subsequent publication of the obtained results and used methods in the created platform. Populating and enriching the blog with posts from all Club members.

Acknowledgements

I'd like to thank my colleagues at the Computer Vision Center (CVC), especially Pau Rodriguez, who has advised me a lot and taught me even more, throughout the development of this Bachelor's Project. I'd also like to thank my fellow students for their support and friendship. Above all I thank my tutor, Jordi Gonzalez, for guiding me during each step of this project, for all the knowledge he has conveyed to me and for having placed complete trust in me throughout these months.

References

[1] https://www.kaggle.com/sudalairajkumar/cryptocurrencypricehistory.

[2] https://www.kaggle.com/mylesoneill/game-of-thrones/data.

[3] http://yann.lecun.com/exdb/mnist/.

[4] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, pp. 2278–2324, Nov 1998.

[5] https://challenge2018.isic-archive.com/task3/.

[6] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?," *ArXiv e-prints*, Nov. 2014.

[7] https://datauab.github.io/.

[8] https://www.python.org/.

[9] https://jupyter.org/.

[10] https://datauab.github.io/bitcoin_price_analysis/.

[11] https://datauab.github.io/got-deaths-predictions/.

[12] https://datauab.github.io/mnist_example/.

[13] A. Paszke, S. Gross, S. Chintala, and G. Chanan, "Pytorch," 2017.

[14] http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html.

[15] https://datauab.github.io/fine-tunning/.

[16] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[17] https://challenge2018.isic-archive.com/leaderboards/.

[18] J. Bergstra and Y. Bengio, “Random search for hyper-parameter optimization,” *Journal of Machine Learning Research*, vol. 13, no. Feb, pp. 281–305, 2012.

[19] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[20] `https://github.com/DataUAB`.

[21] `https://github.com/codalab/codalab-competitions/wiki/Organizer_Leaderboard`.