Traffic Sign Classification Using Deep Learning

Axel Rubio

***Abstract*— Traffic sign classification is essential in the development of autonomous cars, requiring the use of Deep Networks to process a vast amount of data. The goal is to test different Deep network models and architecture, together with regularization techniques to obtain the best possible final classification. First we trained two models of Multi Layer Perceptrons, the first one using a Sigmoid activation function, and the second using Rectified Linear Units. Both performed well without overfitting, leading to the conclusion that the data set is easy to learn. Though, the use of sigmoid activation function highlights the vanishing gradient problem, that we are able to prevent using Rectified Linear Units function to converge faster, and to a higher accuracy. Secondly we trained a Convolutional Neural Network on the same data set. The results were not satisfactory as we observed over fitting. To fix this issue, we used a powerful regularisation method called dropout. Results were clear and beyond expectations, as we were able to reach an accuracy of 99% when using this model on the test set for the final classification. Besides, we also discuss the normalization of our data set, which is essential in our case because of the input format, and in particular the RGB color encoding. Finally we identified some trends among the misclassified images.**

I. INTRODUCTION

This paper addresses traffic sign classification using two different types of deep networks : Multilayer Perceptron (MLP) and Convolutional Neural Network (CNN). Traffic Sign classification, as a part of image classification, has become a staple meaning in the car industry the past few years. It is mainly used in the development of autonomous vehicles. In this paper, we try to understand and analyse recent methods, architectures used to optimize a Deep Networks for image classification. The Data set used was issued by The German Traffic Sign Recognition Benchmarks [1]. For reasons of computation time, we will focus on the "light version" made of 129430 samples. This light version includes data augmentation, such as image rotation, lighting modification and contrast. The Data Set contains 43 classes, each representing a specific type of sign, such as speed limit, priority, stop and so forth. The size of the images is 48x48x3. Each image has a width and a length of 48 pixels on a stack of 3 layers for color encoding done in RGB. Each value ranges from 0 to 255. To build our networks and do our simulations, we will use keras and tensorflow on python. At first glance we will focus on training a Multilayer Perceptron on this data set, comparing the different activation functions Rectified Linear Unit (ReLU) and Logistic function (Sigmoid), especially based on the convergence rate, and the accuracy. We will discuss the influence of data normalisation in our case. We will also look at different Convolutionnal Network architecture to analyse the performance to avoid over fitting by using regularisation methods like Dropout. Comparing all types of network architecture we have encountered so far, we will finally use the selected model to do our final classification, and analyse the results to get a better insight into the data set.

II. BACKGROUND

A. Multilayer Perceptron

MLP is a type of feed forward network. It is a fully connected network, meaning that each node in a hidden layer is connected to every other nodes in the next and previous layers. Each node in the network has a non-linear and differentiable activation function. There exist a lot of different activation functions, but we will use two different one in different architectures.

Rectified Linear Unit (ReLU) : $y = max\{0, x\}$

Logistic function (Sigmoid): $y = \frac{1}{1+\exp(-\lambda * x)}$

Moreover, when using MLP to classify elements from a data set, the last layer (output layer) needs to have the same number of nodes as the number of classes. Usually, it is better to use a sigmoid activation function for the output layer [2].

B. Early Stopping

During the training phase, the network tries to learn the underlying function to describe the data. The training loss is decreased because the network learns from the data and improve itself. But there is a risk of learning by heart, storing the training data instead of finding the underlying function to generalize on another data set.

Early Stopping is a regularization method used to avoid over fitting while training a networks. To do so, the method consists in adding a new set when splitting the original data set, called the validation set. It is used to simulate the network's ability to generalize [3]. As a result, when the loss on the validation test starts increasing, the generalization ability decreases, meaning that the network overfits. By retaining the corresponding number of epochs, and retrain the network until this number, the networks is less likely to overfit. This method will be used in our different analysis to ensure relevant results.

III. MULTILAYER PERCEPTRON

A. Architecture

To begin with, the architecture of the network needs to be chosen carefully. MLP are fully connected networks, that can be prone to over fitting. There only exist some rules-of-thumb to choose the number of neurons in each layers, depending on

input or output size mainly [4]. However, an interesting idea is dimension reduction in the number of nodes in successive layers. Results from our simulations seems to get along with the idea of a better feature extraction. The model seems to be more likely to perform better on such architecture where the number of nodes is decreased over the layers. We tried different empirical combinations for (instance 64*32*43 or 128*64*43). But it appeared that we get higher performance using at least the same amount of nodes in the last but one layer and the last one. We finally selected the architecture *Fig.1* where the 43 nodes in the output layer correspond to the 43 different classes in our problem.

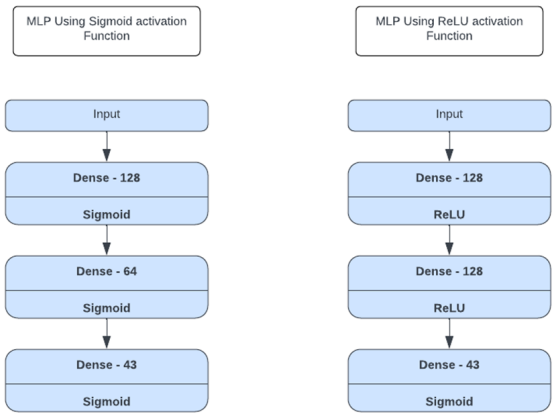

Fig. 1: Two MLP architecture that we used to train our dataset. One using a sigmoid function, the other using ReLU

B. Normalization

Gradient descent in MLP can be very sensitive to the magnitude of derivatives while computing the chain rule. We tried to train our networks on raw data. But the training accuracy was very low (about 5%), and the network was not learning from the data. In our case, normalizing the data was necessary to train our MLPs. To do so, as the colors are encoded using RGB, inputs are encoded as values from 0 to 255. To simply normalize these values, we divided each value by 255, in order to have all our input in range $[0, 1]$. Furthermore, most of the traffic sign only display one or two colors among red, blue, and white. Consequently, in one of the 3 color layers, in most of the input, data is likely to be 0. Hence normalization seems to be wise to smooth scale gaps. The results were immediately better as we were able to train the network, and get the following results. *Fig.2*

C. Simulation

We trained our networks on 100 epochs first, but we observed that sigmoid hasn't really converged. So we tried to double the number of epochs to observe convergence.

The first graph *Fig.2* illustrates the loss (categorical cross entropy) for both training and validation set using ReLU and Sigmoid. From these curve, we can emphasize that using both ReLU and Sigmoid, our networks converges, as the loss steadily decreases until reaching a plateau. The first conclusion we can draw is that our networks do not over fit, as the validation loss is not increasing after a certain number of epochs for the two activation function.

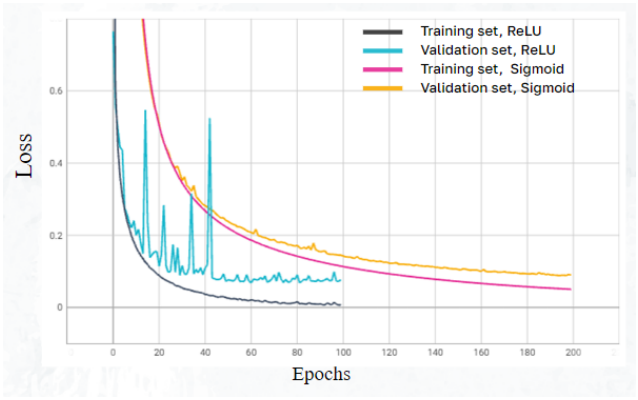

Fig. 2: Loss using ReLU and Sigmoid activation function on MLPs

Furthermore, based on the second graph *Fig.3*, we can say that both Sigmoid and ReLU function reach a high accuracy for the validation set. For sigmoid function, we reach an accuracy of 98.5 after 200 epochs, whereas using ReLU function we get an accuracy of 98.65 after 100 epochs. Also, the accuracy we reach highlights that our data set does not seem to be hard to learn.

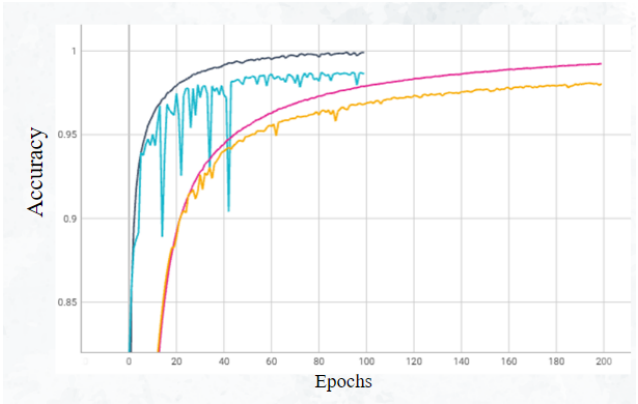

Fig. 3: Accuracy using ReLU and Sigmoid activation function on MLPs

By analysing the respective curves of ReLU and Sigmoid, we can highlight two main behaviours. On the one hand, Relu converges faster than sigmoid. We see that the validation loss for ReLU reaches a plateau after 40 epochs whereas for Sigmoid, we can't really see a plateau as if it hasn't converged yet after 200 epochs. On the other hand, we can observe that for ReLU, even if we reach a better accuracy, there are some peak in the loss and accuracy curves of the validation test during the first 40 epochs. Relu seems to be less stable at the beginning of the training on our data set.

D. Convergence Speed

One of the most interesting observation we can notice is the faster convergence speed of ReLU. In this part, we will go into details to explain this behaviour [5]. When computing the chain rule, we must compute the derivative of the activation function.
Sigmoid : $y'(x) = \lambda y(x)(1 - y(x))$
Relu : $y'(x) = sign(y(x))$ Let's take a closer look at the behavior of these two derivatives. The following plot *Fig.4* shows the magnitude of ReLU's and Sigmoid's derivatives. (lambda = 1 for plot)

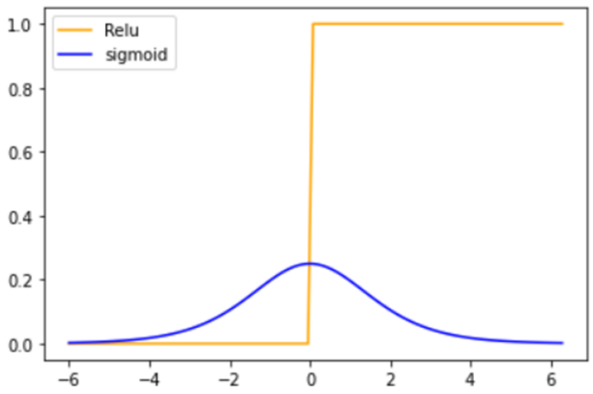

Fig. 4: Magnitude of derivatives : ReLU and Sigmoid

ReLU derivative is either 0 if the weight is negative, or 1. When computed in the chain rule, it will be seen as a series of multiplications by 1 or 0, and the magnitude won't be decreased because of the size of the chain. However, using sigmoid function, the magnitude of the derivative reaches a maximum in 0 which is smaller than 1 (in our case because lambda = 1). If the chain is big, the multiplication of these derivatives will lead to a very small value approaching 0. This will decrease considerably the magnitude of the gradient, leading to very small weight's changes after each epoch. Therefore, the network will converge more slowly. Well know as "Vanishing Gradient", this problem explains the shape of the curves for sigmoid activation function in Fig2, which tend to converge in a big number of epochs.

E. Further Observations

As mentioned previously, we can observe on the curve related to the validation set using ReLU, that we have 4 big peaks in the beginning, until 40 epochs. It seems that ReLU has some stability problems here. At first sight, we thought that the learning rate could be responsible for these peaks. However, if we look closely to the curve corresponding to the training set using ReLU, we don't see any of these peak. The learning rate affects directly the gradient descent and so the training loss [6]. Therefore the learning rate is not responsible for this behaviour. Another possible reason we took into consideration was possible "bad batches" in the validation set, leading to these peaks only in the validation set. We tried different splits of the data, randomizing the order of the batches presentation. But we still had some similar peaks during the first phase of the training. However, it does not seem to have a bad impact, as the network converges after 40 epochs, reaching a high accuracy.

IV. Convolutional Neural Networks (CNN)

Convolutional Neural Networks are networks composed of multiple layers including Convolution Layers, Pooling Layers, and Fully connected (or Dense) Layers. A lot of different parameters have to be tuned while building a CNN, especially on keras, but we will not mention them in this section. Research and influence of these parameters will be exposed in the Related Work section.

A. Architecture

Building a CNN can be very difficult if we do not have any example of existing structures. So we compared different architectures in order to get some intuition about elements that would fit in our problem [7][8].
Though, as mentioned during the conclusion about MLP results, our data set seems easy to learn. Therefore, we carefully chose a relatively simplistic architecture which is shown below *Fig.5* to reduce the risk of non convergence or overfitting.

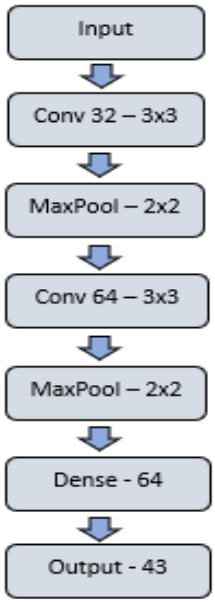

Fig. 5: First CNN architecture. Both the convolution layers, and the last but one dense layer have ReLU activation function. The last dense layer, output, has a sigmoid activation function.

B. Normalization

Convolution layers use receptive fields that look for pattern in the input data, simultaneously in different areas of the input. It allows them to find spatial relationship, and then being less sensitive to data distribution. So we could reasonably make the assessment that CNN are less sensitive to input normalisation than MLP.
However for our dataset, we were not able to train the CNN

without normalizing the data before. This problem is mainly due to the color distribution in the signs that we mentioned in the section about normalisation for MLP. We printed a lot of input data, and were able to observe these 0 layers in the RGB encoding in most of the images. Therefore, we also normalized the data in order to avoid this problem, and being able to train our network.

C. Simulation

Fig. 6: Training and validation Loss using the CNN

The first relevant feature from our simulation is the training time. In our configuration, it took us 36min 20sec to train the network on 25 epochs. This training time is very high compared to the one using MLP (about 12 min for 100 epochs). But in terms of epochs, a first conclusion we can draw is that the convergence is extremely faster than using an MLP, as we reach 98% of accuracy after only 2 to 3 epochs.

After training the networks for 25 epochs using the first architecture, we plotted the loss in *Fig.6*. We can observe a very low training error after a few epochs (we used the same loss function as for MLP). However, the validation curve shows that our network overfits almost immediately after 6 epochs, as our validation loss starts increasing. This problem leads to a deterioration in the accuracy over the epochs, leading to a bad generalization of our network. (94% of accuracy for the validation test after 25 epochs). To avoid this problem, we will use a recent regularization method called dropout.

D. Dropout

Dropout technique is one of the most recent and efficient regularization method for Deep Networks. It relies on sparse activation of nodes each time we present a new input vector to the network. Nodes are switched at random with a certain probability, leading to weights scaled by the probability. Being forced to learn more robust features, the generalisation performance of the model is improved. Some works [9] about the way to choose the layers on which we drop, and the percentage of nodes dropped led us to the following architecture in *Fig.7*

The plot *Fig.8* allows us to compare the previous loss curves of the network, with the new ones corresponding

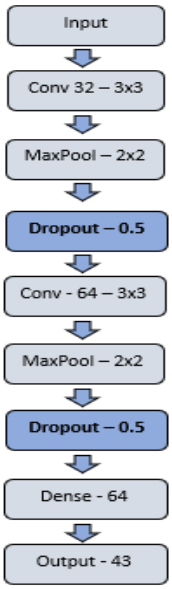

Fig. 7: Second CNN Architecture using Dropout Layers in Tensorflow

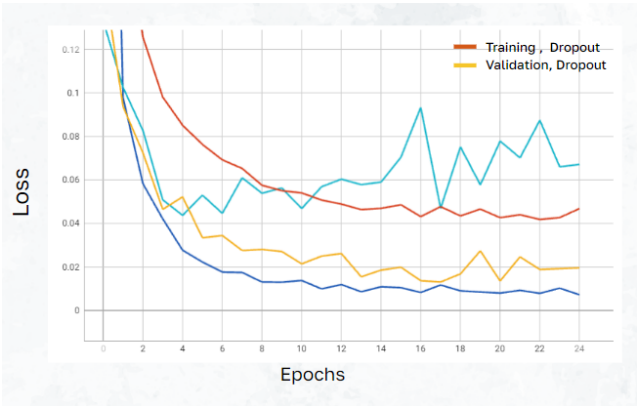

Fig. 8: Influence of Dropout on the loss on both training and validation sets.

to the use of Dropout. It depicts the benefit of this regularization method. First, the validation loss is not increasing anymore after a few epochs, and converges after 15 epochs. So we can conclude from our simulation, and observations, that we succeed in limiting over fitting on our network thanks to this method. If we plot the accuracy, we can observe that it is very high compared with the previous one obtained using MLPs, reaching almost 99.5% for the validation set.

However, this simulation using Dropout highlights a typical behaviour of the loss and accuracy of networks using dropout. Referring to these plots, we can see that the training loss is higher than the validation loss. This can be explained by the high rate of nodes dropped at each input presentation (50% in our case). Dropout allows the networks to improve

the final overall generalisation on the validation set. Nodes are dropped during the training phase only, in order to force the networks to find the underlying function and not overfit by storing previous results. So a high percentage of nodes dropped has a direct consequence on the training loss, as we prevent the network from using some prior knowledge at a given moment. This result is also reverberated on the test accuracy *Fig.9*, which is lower than the validation one. Though, there is almost no difference in term of number of epoch to converge (around 20).

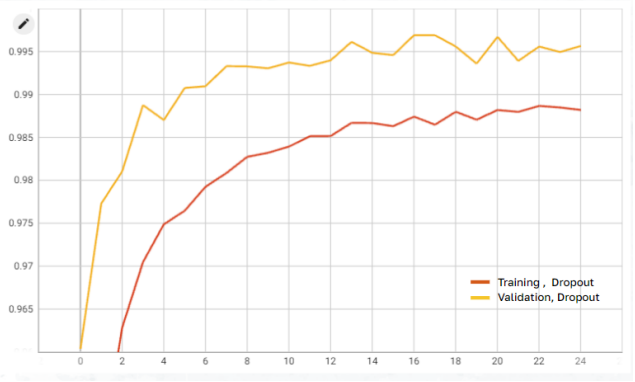

Fig. 9: Accuracy using Dropout in the CNN

Finally, we made another observation while looking closer at the training time for each batch. The training time increases from 36min 20s to 44min 42s while using dropout layers in our model. In other words this means a 24% increase in training time. Dropout is considered as a bagging method [9]. The network is sampled in smaller networks that randomly drop some weights. Every randomly defined network will be trained on its own. The network will then summarise the overall information by scaling down weights in the final network. But the successive training do not provide gradual improvement to the networks. Everything is done during the last phase, while simulating on the test set. Therefore, it will increase the training time.

E. MaxPooling

Pooling layers are used to reduce the dimensions of the data. Because of the zero padding technique used in the convolution layers to avoid loosing information on the edges, dimensions remains the same after a convolution phase [CF Related work]. To see the impact of dimension reduction between convolution layers, we tried to train our network without pooling layers. We did not succeed, as the training time per batches was more than 6 times higher than previously, leading to an error for the computer. The previous network using 2 Maxpooling layers took 1min 32 sec on average to train on 1 epoch, without the Maxpooling is takes around 8min 45sec. Also, because we do not reduce dimensions anymore, we can reasonably assess that the network would have probably overfitted very quickly.

V. Model Choice and Classification Test

In this final section, we will choose our final model, and analyse the classification results, based on the classes and the images in order to validate our model.

A. Final Model Selected

To begin with, let's recall the previous results from MLP and CNN in the following table *Fig.10*.

	Training Epochs	Epochs to Converge	Time	Accuracy
MLP with **ReLu**	100	80	12min 10s	98,4
MLP with **Sigmoïd**	200	>200	23min 55s	97,8
CNN	25	Overfitting	36min 20s	94,23
CNN with **Dropout**	25	18-20	47 min 15s	99,55

Fig. 10: Accuracy using Dropout in the CNN

We are looking for the best model allowing us to well-classify as many pictures as possible. Because the number of pictures is limited in our case, we do not want to restrict our choice looking at the training time, as long as it is feasible. We will then focus on the convergence and the accuracy that are the main features responsible for the classification. As mentioned above, our data set looks easy to train, reaching very high accuracy's using MLP structures. It could be amply sufficient to use an MLP. But as we are craving for the best classification possible, our final choice will be the CNN with Dropout.

B. Results

We reach a final accuracy of 99.01% on the test set. This result is very satisfactory, as we only misclassify 1% of the pictures; 77 over 7766 in our test set. To better identify misclassified classes, and possible links between them, we'll use a confusion matrix to represent the results *Fig.11*.

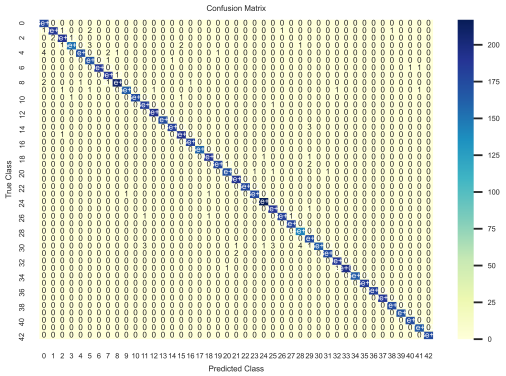

Fig. 11: Confusion Matrix

We can identify 3 main classes in which images are more likely to be misclassified :
- Class 4 (90km/h speed limit), 7 images misclassified of which 3 misclassified in Class 6 (70km/h speed limit)
- Class 5 (80km/h speed limit), 8 images misclassified of which 3 misclassified in Class 1 (30km/h speed limit)
- Class 31 (Animals crossing), 13 images misclassified of

which 4 misclassified in Class 29 (Bicycle crossing)

On the contrary, a lot of classes are perfectly classified without any mistake: 1, 13, 14, 17, 18, 22, 23, 25, 28, 29, 30 and 35 to 43

These observations lead us to the conclusion that the data set have 3 classes more likely to be misclassified, responsible for almost 40% of all misclassifications.

C. Image Analysis

We wanted to analyze the misclassified images to get an idea of the reasons for these misclassifications.

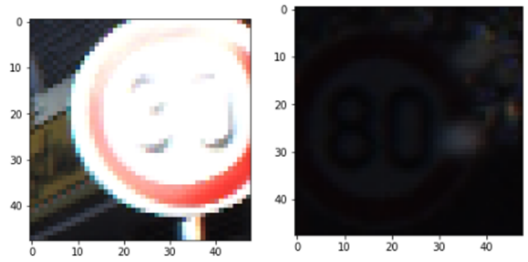

Fig. 12: Light Exposure

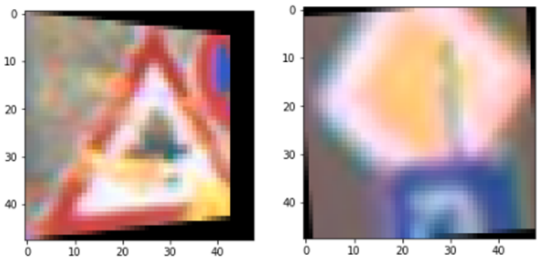

Fig. 13: Multiple Signs

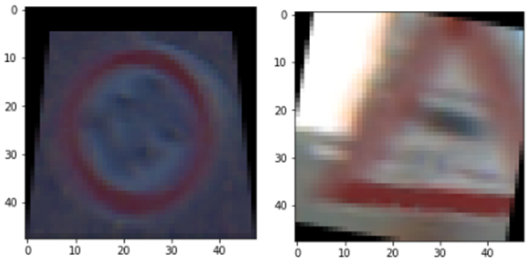

Fig. 14: Very Blurry

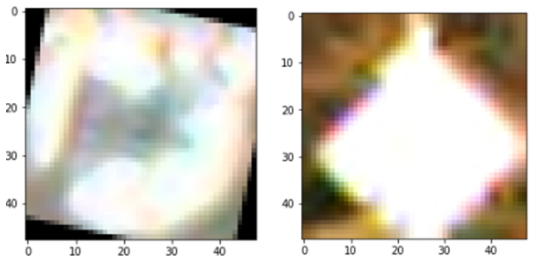

Fig. 15: Unidentifiable

To do so, we looked at some of them, and were able to identify a few trends. We grouped them into 4 groups according to their characteristics. These results can be used to qualify the figures and percentages obtained by the confusion matrix. For instance the two unidentifiable images *Fig.15* are not really misclassified, but a consequence of invalid input data. As a result, we could refine the real accuracy and performance of our model.

VI. Related Work

In this section, we present some research we made to get a better understanding of CNN, and especially how to tune different parameters when building CNN in tensorflow.

A. Convolution Layer

Convolution layers are the core layers of a CNN. They allow to search for local connectivity in the images, by using filters that will span the input space. A first parameter is the number of filters [10]. For a given input a*b*c, all 2-dimensional filters in the same c share the same weights. This is very helpful, as it allows to use a lot of filter, to add levels of abstraction to extract the data, without increasing the number of node that could lead to long training time or over fitting.

We also have to tune the size of the filters, and the Stride, which quantifies how much we move a filter at each step while searching in the input. There is a relation between these parameters [10], as the goal is to cover all the pixels in the image. Usually the stride is set to 1, to avoid missing any information. And the size of a filter is relatively small compared to the input, to be able to catch local information. Last but no least, zero-padding [10] is a method used to avoid missing significant information that could be kept on the edges of an input. Sometimes, depending on the stride, the filter size and the input dimension, some elements in the input can't be reached by the filter. By creating an extra border of zeros around the image, we allow the filters to be moved one more step, and catch information on the edges. This method makes the convolution lead to an output of the same size (if the stride is 1): so it preserves the input size.

B. Pooling Layer

Once we have extracted interesting information thanks to the convolution, we want to reduce dimension in order to reduce the number of parameters to learn, and the amount of computation needed. This non-linear transformation called pooling is performed by specific layers, the pooling layers. There exist different types of pooling layers chosen for different strategies [11]. Max Pooling and Average Pooling are the most frequently used. The first one selects the maximum elements of a neighbourhood, while the other averages the information. From a graphical point of view, Average Pooling makes pictures more smooth whereas Max Pooling can highlights some small details.

In our case, we will use Max Pooling. This choice is also justified by some observation we mentioned in our work

about the RGB encoding and the low variety of color in the traffic sign images. Because of a lot of 0 values in some color layers, keeping the maximum lets the network spot details in the image, which could help to classify them.

VII. Conclusions

In conclusion, we can say that despite a relatively simple data set to learn, we were able to deepen and analyze various parameters and techniques used in Deep Networks in order to reach a high accuracy in our final classification. MLP models that we trained on our data set performed well and could have been sufficient in our case, but we wanted to compare them to CNN which are the basis of every Big deep networks nowadays. Through sigmoid activation functions, we identified the vanishing gradient problem, and used existing solutions, like ReLU, to prevent it. Moreover, the structure of the inputs in our data set highlighted the necessity of normalisation to train both MLP and CNN. This analysis could also be used in further works by trying to train the same dataset after a black and white conversion for example. In addition, regularization methods like dropout have proved to be very efficient to prevent overfitting on our data set. The final classification is very successful as we reach 99% of accuracy, with only 77 images misclassified over 7766. We were able to identify trends in the misclassified images, that could be use to improve our model in the future.

References

[1] A. Minai, "The german traffic sign recognition benchmarks," *International Joint Conference on Neural Networks*, 2011.

[2] P. Hosseini, M. Liwicki *et al.*, "Task 11 : A comparison of activation functions for soft and hard label prediction," *Lon-eå at Semantic-Evaluation-2023*, 2023.

[3] L. Prechelt, "Early stopping but when?" *Lecture Notes in Computer Science*, vol. 1524, 2000.

[4] J. Heaton, "Introduction to neural networks," vol. 1, Chapter 5, p. 129, 2008.

[5] Y. Bai, "Relu function and derived function review," *International Conference on Science and Technology Ethics and Human Future (STEHF)*, vol. 144, 2022.

[6] A. Bilogour, "Tuning your learning rate," *Kaggle Notebook*, 2018.

[7] P. Varshney, "Alexnet complete architecture," *Kaggle Notebook*, 2020.

[8] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *International Conference on Learning Representations (ICLR)*, 2015.

[9] Srivastava, Hinton *et al.*, "Dropout : A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[10] A. Buyn and R. Gao, "Convolutional neural networks for visual recognition," *Standford: cs231n.github.io/convolutional-networks*, 2023.

[11] S. Sharma and R. Mehra, "Implication of pooling strategies in cnn," *Foundations of Computing and Decision Sciences (FCDS)*, vol. 44, no. 3, 2019.