UCL Université catholique de Louvain

École polytechnique de Louvain (EPL)

epl

Study of label errors on a convolutional neural network

Dissertation presented by
Eric LEBEAU

for obtaining the Master's degree in
Electrical Engineering

Supervisor(s)
Christophe DE VLEESCHOUWER

Reader(s)
Benoît MACQ, Simon CARBONNELLE , Dimitri VAN ASSCHE

Academic year 2016-2017

Acknowledgement

First, I would like to thank my supervisor, Prof. Christophe De Vleeschouwer who was available when needed for guidance all along this project. I would also like to thank him for all the advice he gave me during the meetings we had all over the year.

Secondly, I would like to thank very much the PhD student Simon Carbonnelle for his support, his availability and the time he spent to help and guide me.

Lastly, this master thesis would not have been possible without the technical support of Dimitri Van Assche and the company he represents.

Abstract

Label errors can have a negative impact on the training of a convolutional neural network for image classification. Consequently, the learning of these label errors can lead to a decrease in overall performance with lower than expected image detection rates. Even with a well trained convolution neural network, a decrease in performance can be observed.

The goal of this paper is first to study the impact of the label errors. Next to provide a tool to identify the label errors in order to purify the database. Finally, the paper studies a method that allows the training of a convolutional neural network that is more robust to label errors.

The purification technique presented in this paper defines multiple criteria based on the neural network output to distinguish label errors from classification errors. These criteria give a probability to each image to contain a label error. The training of the convolutional neural network is based on these label error probabilities. Images that are suspected to have a label error will have a reduced impact on the training.

The paper presents different encouraging results obtained on a well-know database, MNIST. However, the usage of a more complex database, like CIFAR, displays the importance of an efficient network in order to have the maximum gain of the purification method.

A deeper analysis of the purification methods in different cases is also provided. Indeed, deeper tests show the negative impact of overfitting in the presence of label errors. Small networks seem to be less affected by this overfitting but their results are unfortunately not as good as the ones obtained with complex networks. Some alternative solutions like removing all errors (label errors and classification errors) during training by using 0 weights are tested. Results show that these solutions allow a certain stability but do not seem optimal. More creative solutions with iterative weights give promising results.

Finally, a real test on a more complex database finalized this paper. This test confirms the previously observed results but the overall performance and efficiency is not as good as for simpler databases like MNIST. The results seem to suffer from the not random noise between similar characters. However, they still confirm the interest of the method and suggest to deepen the research.

CONTENTS

CHAPTER 1

INTRODUCTION

To begin this master thesis, the following sections provide an introduction to the subject. First, the context of the study will be described. Then, the objectives of the thesis will be stated. Lastly, the structure of the report will be briefly detailed.

1.1 Context

Reliable and accurate image recognition is of great interest in more recent research domains like artificial vision systems and more generally machine learning. It is often referred to as image classification as it aims to classify an input image based on its visual content.[2] The field has made recent progress going from the detection of a few objects in controlled setups towards hundreds of objects classes in more arbitrary environments.[3] This progress was made possible by the development of robust image descriptors such as such SIFT[4] and HOG[5] but also increasingly large and realistic image databases for training and testing.[3]

Neural networks have been less common in the past years although they have a long history in visual recognition[3] with Rosenblatt's Mark I Perceptron being arguably one of the first computer vision systems.[6] Convolutional neural networks were introduced in the early 1990's by LeCun et al[7] as a special type of multilayer perceptron.[8] They have demonstrated excellent performance in image classification tasks such as hand-written digit classification and face detection.[9] However, computational speed is a limiting factor for convolutional neural network architectures.[10] This explains why they were less common in early 2000's but hardware progress has increased their usage for image classification.

Increasing computational speed and excellent performance in image classification are in favor of the usage of convolutional neural networks for license plate recognition. The latter can lead to many applications such as payment of parking fee, highway toll fee, traffic data collection (vehicle classification and counting), etc. This type of applications offers many innovative business opportunities. As a Belgian-based company, Macq offers products such as license plate recognition using convolutional neural networks. Their system was recently installed in several large Belgian cities.[11] Indeed, local authorities were looking at a solution as the local police could no longer control traffic to identify stolen vehicles or vehicles involved in crimes. Macq solved their problem with its modern, automated, reliable and integrated iCAR system for license plate recognition.[11]

This master thesis works on convolutional neural networks for license plate recognition and more generally for image classification. It was made in collaboration with Macq as the subject is linked to their current product offering, iCAR.

1.2 Objectives

The main goal of this master thesis is to improve the detection rate of license plates from the Macq database. More specifically, the characters of license plates on images from the Macq database should be detected and properly identified by machine learning methods based on convolutional neural networks. Generally, a license plate recognition (LPR) system contains three major components: license plate localization, character segmentation and character recognition. This paper focuses on the character recognition.

During this research, it has been observed that the initial database could potentially contain label errors. Therefore, the objectives of this work have been directed towards label error detection. Indeed, label errors tend to negatively impact the character detection rate. Therefore, the first goal of this work is to develop a label error detection algorithm that will ultimately improve character recognition on database that could potentially contain label errors. A second goal of this master thesis, is to explore optimization possibilities of the label error detection algorithm as well as deepen the understanding of the algorithm by evaluating the impact of its parameters.

1.3 Structure

To begin a literature review on topics relevant to the subject of this master thesis is provided. This literature review contains information on image classification methods but also more generally on convolutional neural networks.

Next, the materials and methods used during this work are detailed. Regarding the materials, the used software is described whereas for the method, the label error detection algorithm is detailed step by step.

Then, the main obtained results are presented and discussed. This starts with the results obtained on the MNIST and CIFAR databases. Next, the different methods and associated results to optimize the algorithm are given. Lastly, results obtained on the database of license plate images are presented.

Finally, this work is ended with a conclusion and some future perspectives.

CHAPTER 2

LITERATURE REVIEW

In this chapter, a literature review is provided on different subjects relevant to this master thesis. First, a review on image classification and recognition with label noise, i.e. label errors, is given. Then, a review of the basics of convolutional neural networks is presented.

2.1 Label noise in training databases

A way to improve recognition results is to fine tune the convolutional neural network network with the right parameters but, given the impact of the training database, it is crucial to focus on the training database.

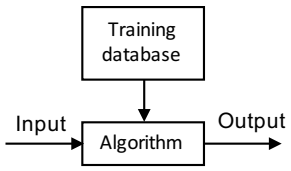

Figure 2.1: Illustration of the impact of the training database.

Machine learning algorithms rely critically on the training database not only in terms of quantity (bigger is better) but, even more important, in terms of quality. The lack of quality is called noise. There are two types of noise: feature noise (e.g. low definition picture) and class noise (e.g. incorrect label). Class noise is by far a bigger concern than feature noise as there a one class but many features. The presence of inaccurate labels is known to deteriorate the performance of a classifier. The labels used to train are expected to be unambiguous and accurate. But these labels, usually provided by human judgments, are subjective. Indeed, it is costly to train accurate observers or to pool a large number of observations. In addition, some training databases are coming from social network applications.

Based on Schafer and Graham,[12] Frenay and Verleysen[13] have defined three categories of label noise schematically represented on figure 2.2 that we illustrate hereafter with examples in plate recognition:

- Noise Completely At Random (NCAR)
  Imagine that, end of the day, all observers are tired and they do more mistakes

across all samples. This is an example of NCAR mislabels randomly distributed across all classes.

- Noise At Random (NAR)
  Imagine now that German observers are more disciplined than French observers: we may see more mislabels with F than with G. This noise is not randomly distributed around all classes but still independent of the classifier. Let's call it NAR.

- Noise Not At Random (NNAR)
  When observers mislabel O with 0 and U with V, this is NNAR as the proximity of the classes interfere with the mislabeling exercise.

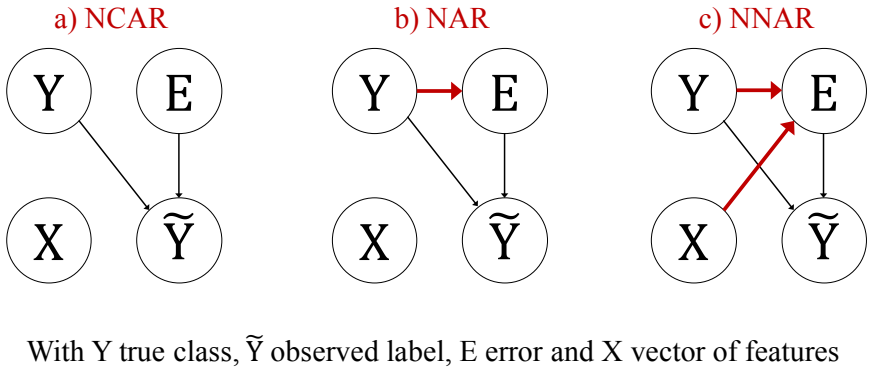

Figure 2.2: Statistical taxonomy of label noise: a) noisy completely at random (NCAR), b) noisy at random (NAR) and c) noisy not at random (NNAR). Arrows report statistical dependencies. Notice the increasing complexity of statistical dependencies in the label noise generation models, from left to right. The statistical link between X and Y is not shown for clarity.[13]

In real life, NNAR is more frequently observed and its impact is more devastating.

Note: Given the time frame, we have corrupted the labels of training database with additional NCAR noise to calculate the performance of our method. Nearly all the research observed are performing similarly. A further work should consider NNAR noise to improve the validity of the model.

Algorithms which minimize the impact of noises have been studied for 15 years or more. There are hundreds of scientific papers which deal with the subject. In 2014, Frenay and Verleyen[13] have published a classification of the different methodologies that are described in Figure 2.3 and detailed in Table 2.1.

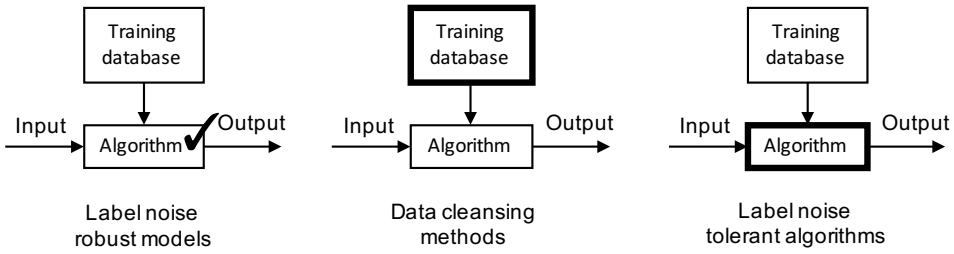

Figure 2.3: Illustration of three different classes of methodologies to minimize the impact of noise.

Articles related to "Label noise robust models" discuss and validate the different algorithms. Articles related to "Data Cleansing methods" discuss ways to remove or relabel data in the training database. Finally, articles related to "Label noise tolerant algorithms" propose improved algorithms to handle the noise.

Category	**Sub-Category**
Label Noise Robust Models	Bagging & Boosting
	Decision trees
Data Cleansing Methods	Measures and Methods
	Model Predictions-based Filtering: • Classification filtering • Voting filtering • Partition filtering
	Model influence & introspection
	kNN & graph based methods
	Ensemble and Boosting-based methods
Label Noise-Tolerant Algorithms	Probabilistic methods: • Bayesian Approaches • Frequentist Methods • Clustering-based Methods • Belief functions
	Model-based Methods

Table 2.1: Classification of the different algorithms dealing with noisy labels[13]

2.1.1 Label Noise Robust Models

The articles of the first category “Label Noise Robust Models” study the robustness of the different models with respect to different levels of label noise. In this category, the key principle is that overfitting is sufficient to deal with label noise. Obviously, models based on overfitting give better results. Bagging is also better than boosting. AdaBoost is probably the worst results as mislabels receive after few iterations very large weights.

2.1.2 Data Cleansing Methods

In the second category “Data Cleansing Methods”, mislabeled instances are detected, removed or relabeled. In this category, the art of the different heuristic is to distinguish mislabel from misclassified. Nearly all these data cleansing methods modify the training database by discarding or correcting mislabeled instances.

Note: our approach differs as we keep the training database intact and only modify the weight of some instances.

This category has different sub-categories: measure and methods, Model Predictions-based filtering, Model influence & introspection, kNN & graph based methods, Ensemble and Bosting-based methods. We focus on the first two.

In the sub-category “Measures and Methods”, we point out the work of Sun & al.[14] who use a Bayesian approach to evaluate the probability of the instance to belong to all class labels and relabel instances which have a high confidence degree to belong to the

predicted label.

In the sub-category "Model Predictions-based filtering", methods run the prediction of the classifiers on the training database to identify suspicious instances where the labeled class differs from the predicted class. Simplistic method (Thongkam et al[15]) remove all suspicious instances. Such methods are dangerous as suspicious instances include not-only mislabeled data that we want to remove but also miss-classified data that we want to keep. The miss-classified data are crucial to improve the quality of the classifier as explained in Guyon's article[16] who introduced the concept of "point of optimal cleaning".

Note: our approach uses "Model Prediction-based Filtering" but we develop an original method to allocate a small weight factor to mislabeled instances and a large weight factor to misclassified instances.

2.1.3 Label Noise-Tolerant Learning Algorithms

In the third category "Label Noise-Tolerant Learning Algorithms", models are designed taking label-noise into account. There are two sub-categories "Probabilistic models" and "Model Based Methods". The first makes assumptions on the noise distribution while the second modifies popular algorithms to make them noise tolerant.

2.1.4 Positioning of our method

We point out the article "Class Noise Mitigation through Instance Weighting" of Rebbapragada and Brodley.[17] While, as mentioned before, a large proportion of the research focus on discarding or relabeling a part of the training database, this paper provides a similar approach to our thesis by using the confidence as a weight for each instance during the training process. They called PWEM (Pair-Wise Expectation Maximization) their probabilistic approach to calculate that confidence.

Note: our approach differs as we only allocate a weight factor (different from 1) to a subset of suspicious instances of the training database.

Although this taxonomy seems clear, it is not obvious to allocate a specific method to these approaches and sub-categories given the large overlap between them. As an example (see Figure 2.4), a method which does not remove or relabel the training instances but changes their weights can be seen as a "Data Cleansing Method" by some and as a "Label Noise-Tolerant Learning Algorithm" by others.

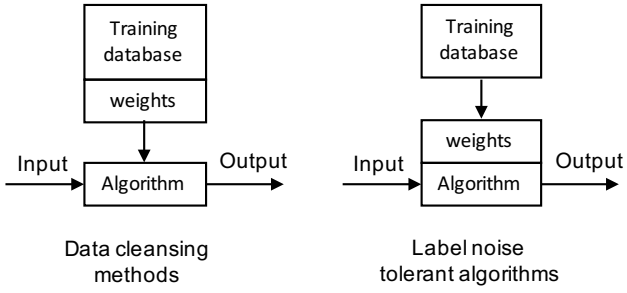

Figure 2.4: Illustration of the positioning of our method.

Our approach is thus a mix of different approaches:

1. Data Cleansing Method (subcategory: model predictions-based filtering) because we clearly apply the classifier on the training database to identify the suspicious instances where the predicted label differs from the allocated label.

2. Label Noise-Tolerant Learning Algorithm (subcategory: model-based methods) because we use probabilistic method to allocate a confidence level to the suspicious instances.

2.2 Convolutional neural network[1]

Convolutional neural networks are used in machine learning for image detection and identification. A convolutional neural network is made of an input and output layer as well as multiple hidden layers. The latter can be convolutional, pooling or fully connected. However, in this case, the focus is on convolutional layers.

2.2.1 Convolutional layer

Whereas a fully-connected layer of neurons receives each single pixel as input, the neurons of a convolutional layer work with a small, localized region of pixels from the input image. This is illustrated in Figure 2.5 that shows all the pixels of an image (being the input neurons) with an example of a small, localized region of pixels connected to one hidden neuron.

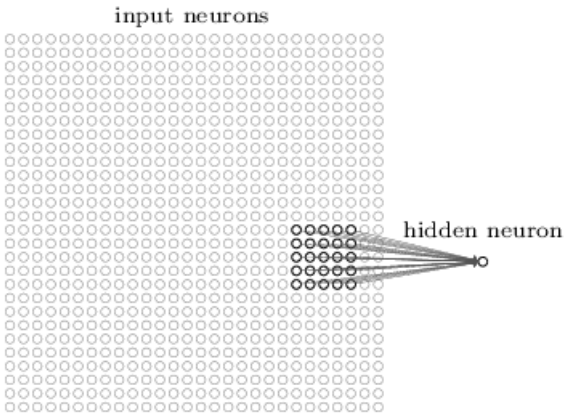

Figure 2.5: Illustration of a hidden neuron connected to a small, localized region.[?]

That region in the input image is called the local receptive field for the hidden neuron. For each connection in this region of 5×5 pixels, a weight is learned. The output is represented by a hidden neuron. Furthermore, the number of hidden neurons is normally lower than the number of pixels in the image as shown in Figure 2.6.

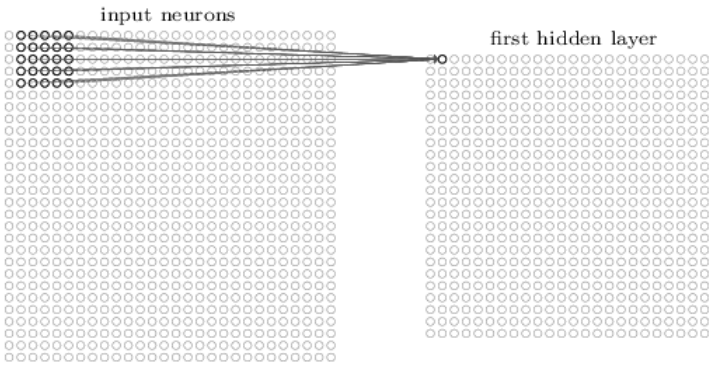

Figure 2.6: Illustration of the completion of the first hidden layer of neurons by sliding a 5×5 receptive field.[?]

Each hidden neuron represents a local receptive field of 25 different pixels. However, the weights associated to the local receptive fields are the same for every hidden neuron. This means that neurons from the same hidden layer detect an identical feature at different locations in the input image. Moreover, the result is an image of approximately the same size[1]. It can thus be seen as filtering an image with a filter having a 5×5 size.

To successfully recognize images, multiple filters are necessary. Therefore, a convolutional layer is made of multiple layers of neurons. Each of these represent the image with a different filter. Moreover, multiple layers can be connected together. A connected layer receives a number of images equal to the number of neurons in the previous layer. Each image is then filtered again with a new filter. Thus, each connected layer acts as a filter applied to the input image in order to identify a specific feature in the image. The main advantage of this type of network is the low number of required parameters. In the example illustrated on Figure 2.6, 25 parameters are necessary to determine the weight and 1 to find the bias parameter. Consequently, the image can be treated by using 26 parameters in total to obtain a good result. Lastly, the convolutional neural network should return an output allowing to classify the input images. More precisely, the classification of an image consists of assigning a class to each image. This is done by first calculating the probability that an image belongs to a certain class, this probability must be calculated for each class. The set of calculated probabilities is the output of the convolutional neural network. Then, the predicted class is set as the one with the highest probability.

2.2.2 Pooling layers

Next to the convolutional layers, convolutional neural networks also consist of pooling layers. Those are usually inserted between convolutional layers in order to simplify the information in the output of a convolutional layer. The most common procedure for pooling is known as maxpooling. Other pooling methods are average pooling or minimum pooling.

[1]The hidden neuron layer will usually be slightly smaller than the original image. For example with the region of 5×5 pixels associated to each hidden neuron, 4 pixels will be lost at the end of each line and column of the image.

Max pooling

The high number of parameters required by convolutional neural networks hampers high speed calculation. Max pooling can be used to reduce this number of parameters and increase calculation speed without dropping to much information. The method takes an area of pixels, often 2×2 pixels are used, and converts this area in one pixel only. The value of this single pixel corresponds to the maximum value of converted area. By using max pooling with a pooling layer after a convolutional layer, the size of the output of the convolutional layer is reduced as the information is condensed.

2.2.3 Training and softmax

Training is done using a method called backpropagation. It calculates the error contribution of each neuron after a certain batch of data containing several images is processed. In this paper, the batch of data consists of 16 images. The heart of the backpropagation method is the gradient of the cost function:

$$\frac{\delta C}{\delta W}$$

where C is the cost function and W is the weight or the bias. This gradient calculates the impact of the weight or the bias on the cost function. The latter computes an error as the difference between the estimated result of the convolutional neural network and the true value. This estimated result is obtained by using the batch of data. More precisely, every image is called by batch to train the network. The data training can be performed several times to improve the convolutional neural network. Each time is called an epoch. Finally, the backpropagation method allows to optimize the weight and bias parameters by minimizing the error obtained on a batch of data.

Weight

During training, parameters are modified to minimize the error as described in the previous section. However, if a training image is considered as an outlier or wrongly classified, it is possible for the user to diminish its impact on training by reducing the weight associated to this image.

2.2.4 Dropout

Convolutional neural networks require an important number of parameters, which can easily result in overfitting. One method to reduce overfitting is called dropout. The method temporarily turns off some randomly chosen neurons. Indeed, each neuron receives a probability to be activated or not at each training iteration. It results in a reduced network for each iteration. By avoiding to train all neurons on all training data, overfitting decreases. In addition, the training speed will be increased. The final result can be seen as a kind of average of multiple small neural networks.

CHAPTER 3

MATERIALS AND METHODS

In this chapter, materials and methods used during this master thesis are detailed starting with the different methods to detect label errors in the training set of the neural network. Next, the used materials being in this case hardware and software will be detailed.

3.1 Methods

During training, some methods are employed to detect label errors that might exist in the dataset. To begin, the intuition behind the method will be explained. Next, different tools to detect label errors based on the general intuition are described. Then, the method used to purify the training database (purification method) and limit the impact of label errors is presented. Lastly, the methods are extended to detect uncertainties in image predictions resulting in an error detection method.

3.1.1 Intuition

Once the 'softmax'[a] layer of the neural network is applied to the database of images of license plates, the generated output corresponds to the probability that the input (image of license plate) represents a certain class (set of characters). As this probability is calculated for all classes, the dimension of the output is equal to the number of different classes. In other words, for each class the probability that the input belongs to it is given. Subsequently, the highest probability is chosen as the predicted class because the input is most likely belonging to that class. If a convolutional neural network works properly, the larger the probability, the lower the risk of prediction mistake. This means, the more the network is uncertain, the lower the output probabilities will be.

Distinguishing the errors made by the neural network from the labeling errors can be done in an intuitive way. Figure 3.1 represents an example of output from a neural network with two different classes. In this example, every image is represented with a point. The color of the point represents the label. Most of the points with the same label are grouped together as it is expected that the network works properly.

[a]Last layer of the convolutional neural network that makes sure that the sum of all the ouputs (probabilities of belonging to each class) is equal to 1.

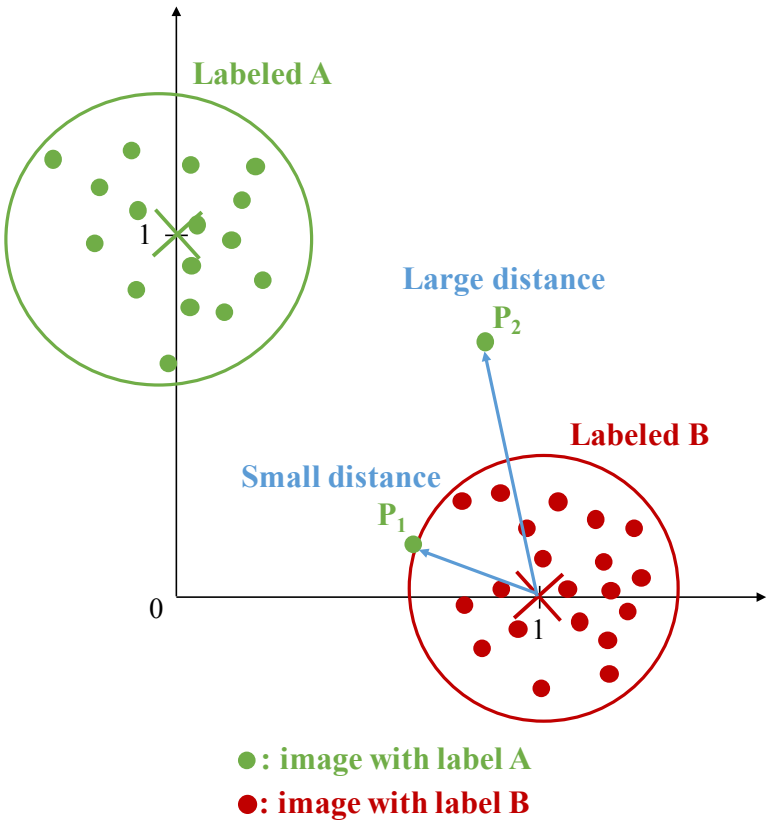

Figure 3.1: Output of a neural network with two different classes, A (green) and B (red).

However, two points, P_1 and P_2, labeled A (green) are locate outside the cluster of points labeled A and are thus isolated from them. The question of label error detection is the following :

Imagine that both of these points are predicted as B (red) by the neural network. These points are now seen as misclassified since their label is A (green) but their prediction is B (red). Nevertheless, a reliable source says that one of these points belongs to class A (green) and was indeed misclassified whereas the other point is actually belonging to class B (red) and was mislabeled. In that case, which point belongs to class A (green) and is misclassified and which point belongs to class B (red) and is mislabeled?

The best response to this question would be:

- The point P_1 belongs to class B (red) because it is really close to the other points belonging to that class. Thus, P_1 was mislabeled.
- The point P_2 belongs to class A (green) and was misclassified.

This response is based on the fact that one of the two points, P_1 or P_2, is mislabeled. Out of the two points, there is a higher chance that P_1 is mislabeled because it is closer to the cluster of class B (red) than P_2.

Of course there is no answer that can be 100% certain as this is only based on likelihood and probabilities. However, this answer looks like the most reasonable and just like image classification, label error detection is based on probabilities and not on certain outputs. The farther away a misclassified point is from its label cluster, the

higher the chance of label error. This is the intuition behind the methods described in the following sections.

3.1.2 Tools to detect label errors

In this section, all the tools and associated methods to detect label errors are described. The first tool is based on the distance between a point and the predicted class. The second tool adds the distance to other classes. The third tool takes into account the distance to the labeled class. The fourth and last tool includes the variance. Each tool is based on the previous one but is more complex as it always contains an additional feature or measure.

Distance to the predicted class

The first way to evaluate how far away a point is from the predicted label, is to calculate the Euclidean distance between the point and the center of the predicted label as explained in the previous section called "Intuition". A misclassified point with a small Euclidean distance has a higher probability to be wrongly labeled than one with a larger distance.

This distance is intuitive and gives a solid basis to separate the wrongly labeled points from the really misclassified ones corresponding to prediction errors of the network. This distance can be formulated as :

$$D_i^{basic} = Euclidean\_distance(P_i, C_{label\ i})$$

With:

- P_i: The output of the image i.
- $C_{label\ i}$: The center of all the outputs that belong to class i, in other words, center of the label class.

Distance to other classes

So far, the distances are based on the prediction of the image, i.e the highest probability of the output of 'softmax'. To go one step further, the other label classes should be taken into account by using multiple outputs of the 'softmax' network. Indeed, up to now, only the most expected, thus the predicted, class has been used.

But, what if the network classifies an image with multiple high probabilities. For example, an image having two probabilities that are significantly higher than the other probabilities and that are also close to each other in value. In numbers, this could translate in all probabilities being close to zero except two, one being equal to 0.44 and the other to 0.45. Our network, will choose the class with the highest probability, 0.45, as the predicted class. If that class has a high probability of label error there is an issue. The network will predict the class corresponding to the label but this label has a high chance of being wrong (label error). Therefore, it could be interesting to also evaluate the second best class with a probability of 0.44. What happens if that class is chosen as the expected class, will the result be better?

To answer that question, the Euclidean distance between the output of image i and the center of all other classes should be taken into account. This way, the impact of images with a high chance of having a label error can be limited and the second best

output could be chosen as predicted class.

To keep a logic result, the distance between an image i and the center of class j is multiplied by the probability that the image i belongs to the class j. Going back to the last example were two classes have high and close probabilities of 0.45 and 0.44, this may allow to predict the class corresponding to a probability of 0.44 instead of the one corresponding to a probability of 0.45 assuming that the latter has a label error. In practice, the distance to the expected class is used first but this result is now weighted with the distance to the center of the second expected class. This may be done by using P_i as the probability output of the image i. P_i is thus a vector of size equal to the number of classes.

Let's note P_i^j the probability of image i to belong to class j. $P_i^1 D_i^1$ represents the distance between image i and the center of class 1 weighted with the probability of this image belonging to class 1. The result noted Γ with the three most likely classes follows the formula:

$$\Gamma = \frac{P_i^1 D_i^1}{P_i^1 + P_i^2 + P_i^3} + \frac{P_i^2 D_i^2}{P_i^1 + P_i^2 + P_i^3} + \frac{P_i^3 D_i^3}{P_i^1 + P_i^2 + P_i^3}$$

With:

- D_i^j: The distance between image i and the center of class j.
- P_i^j: The probability that image i belongs to class j.

One must note that the denominator contains the sum of all the used probabilities in order to normalize Γ. This is useful if not all classes are used. Otherwise, if all classes are used, the denominator is equal to 1. In the example above, the formula is expressed with three classes but it can be generalized to any number of classes as shown below.

$$\Gamma_i = \frac{\sum_{j=1}^{j=n} P_i^j D_i^j}{\sum_{j=1}^{j=n} P_i^j}$$

Label distance

So far, the distance between an image and its predicted label class was used but also the distances between an image and the other classes. One more step can consist of using the distance between the image and the center of its label. This may be useful to improve the detection of label errors when two images have their output at a similar distance of their expected label but one of them has a label error. This case is represented by figure 3.2. Indeed, let's take the intuition's example :

Both points are predicted as class B (red) by the neural network despite their label A (green)! But the truth is that one of them is indeed a green point as its label says but the other is in fact a red point and its label is mistaken (label error). How can the label error be distinguished from the classification error?

The first step could be using the distance with the expected cluster, d_1 or d_2 to discern these two point but in this case, they are equal.

The best answer will be to use the distance with the label cluster, d_3 and d_4. As shown on the figure, the point P_1 is closer of its green label and therefore have the higher

probability to belong to the green. On the contrary, the point P_2 is far from its label cluster and have higher probability two be the red point.

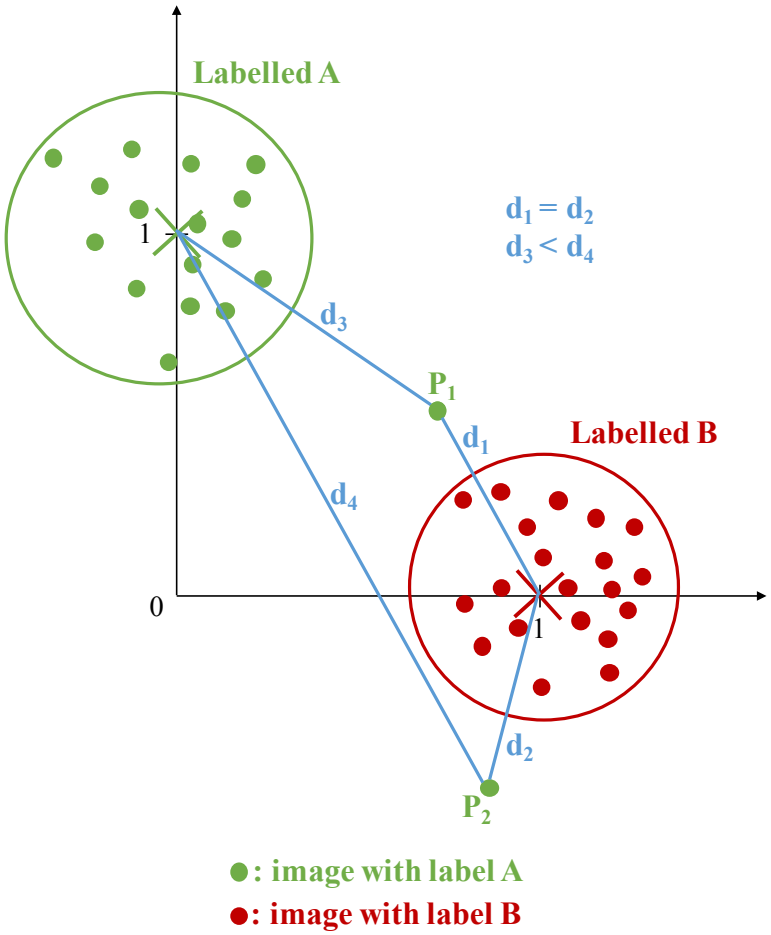

Figure 3.2: Output of a neural network with two different classes.

This case shows that their distances alone is not enough to distinguishes both case since it does not take into account the position of the different center of classes. This case can be extended to the value Γ that have similar intuition with the previous euclidean distance.

As result, the first calculated distance, Γ is divided by the Euclidean distance between the point and the center of its label. We note the result as Δ, it becomes:

$$\Delta = \frac{\sum_{j=1}^{j=n} P_i^j D_i^j}{D_i^1 \sum_{j=1}^{j=n} P_i^j}$$

where D_i^1 represents the distance between the image i and the center of points of its label.

Variance

A second and improved way to evaluate if a point is a labeled error or not, is to take into account the variance of the cluster. Indeed, the variance gives an idea of how dispersed a cluster is. A large variance implies that the center of the cluster is less

representative than a lower one. Without taking account of the variance, two image close to their center of predicted class have the same probabilities to be a label error. Nevertheless, higher is the variance of a cluster, lower us the probability for a point to be close of a cluster. From the two image with the same distance, the image where the variance of all the predicted image is higher have more probabilities to be a label error if the image is misclassified. Since the probabilities to be a label error is inversely proportional to Δ, we divide it by the variance. The result obtain, Ω is use to determine if a image is a label error or not. Ω is obtain with the following formula :

$$\Omega = \frac{V_i^1 \sum_{j=1}^{j=n} P_i^j D_i^j}{D_i^1 \sum_{j=1}^{j=n} P_i^j V^j \sum_{j=1}^{j=n} P_i^j}$$

with:

- V_i^1 : the variance of the most expected label cluster of the image i
- V^j : the variance of the label cluster j

3.1.3 Purification method

In this section the used purification method is described. This method works with the training set of the database of images. The various steps are detailed below.

1. The first step is to train the neural network with the training set. Each image train the neural network. The weight associate to a image for training is set initially to 1.
2. Once train, the training set is given as input of the neural network that return the output prediction for all training image.
3. For each output of the training set, the distance Ω is calculated as explained in the previous section.
4. The calculated distances Ω is then sorted in ascending order.
5. For the misclassified point, a new weight of training, proportional to Ω, is assigned to the training point.
6. Lastly, the network must be trained again using the new weight.

As mentioned above, the assigned weights are proportional to Ω. Indeed, a low Ω might represent a wrong label image leading to a weight close to 0 whereas a large Ω might represent a random error and should thus be normally used to train the neural network. A linear formula was chosen for the weight:

$$W_i = \frac{\omega_i}{O_{max}}$$

Where:

- W_i : The weight of image i
- ω_i : The ω of image i
- ω_{max} : The maximum ω of all the misclassified images

Thanks to this formula, all the misclassified images will have a weight between 0 and 1 according to their ω. It attributes a weight at the misclassified images depending on their contribution to the training.

3.1.4 Error detection method

The previous sections showed how to distinguish a label error from a classification error. In this section, the same methods are used to isolate images with high risks of misclassification. In other words, it tries to distinguish the correct classification from the errors.
Figure 3.3 represents a case where two points of interest, P_1 and P_2 are predicted as red. The user does not know the labels in this case. A reliable source assures that one point is labelled as red but the other is labelled as green.

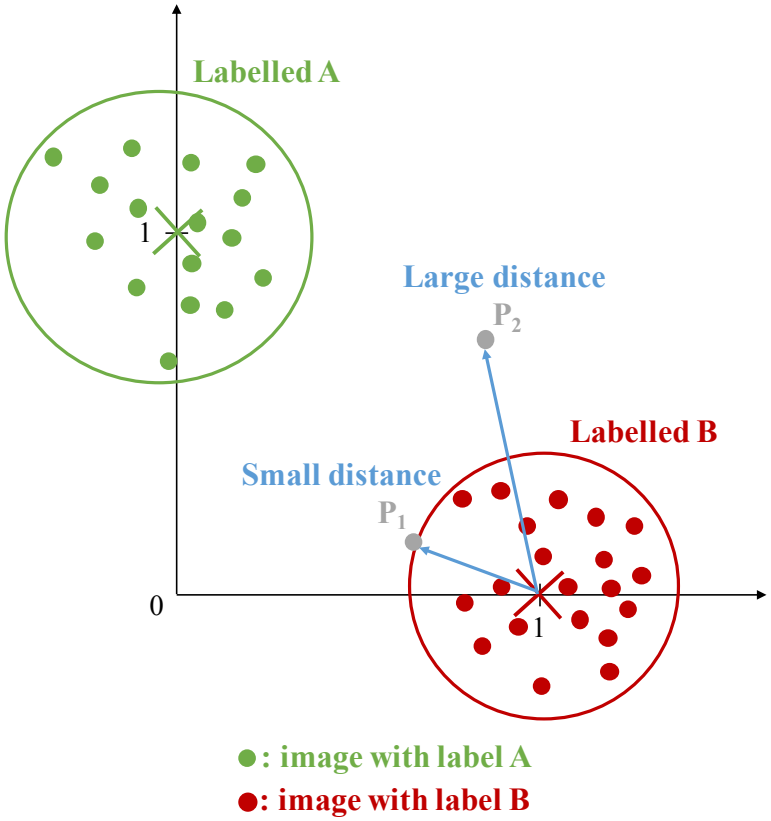

Figure 3.3: Output of a neural network with two different classes.

The basic distances are reused and allow to determine which point is the most likely to be misclassified. In other words, the tools from the purification method can help to determine which prediction is the most reliable and which one is prone to be misclassified. Indeed, P_1 is really close to the points labeled as red and therefore appears to be more likely to belong to the red class. For this reason, the point P_2 has the highest probability to belong to the green class.
The same reasoning can be applied to the different tools described in the previous section going from the basic distance to the computation of Ω. Consequently, these tools can be used to determine how reliable a prediction is and to identify images that are likely to be misclassified.

3.2 Materials: software and hardware

The license plate database are provide by the companies Macq Traffic& Automation. In total about 3 To of image of cars are available. These data are readable thanks to

a xml file. This format allows a ordered structure of the data. For every image the coordinate to localize the license plate are provide as well as the coordinate for each character. Of course the character of a plate are easily recoverable.

The license plate use in this this paper contains 6 or 7 character. The plate is decomposed into separate image with 1 character for each of one thanks coordinate provide by Macq. Once separate, each character are use to train a neural network that can detect a image of 1 character. In this case the output of such a network contains a vector of 36 value, each of one associate to a class i.e. the 26 for the letters and 10 for the digits. The value itself correspond to the probabilities for the image to belong to the class. The sum of the 36 probabilities is 1 since a softmax distribution is use.

With the data, a powerful NVIDIA TITANT X, a gpu from nvidia, is provide by Macq for the computation. This was possible thanks to a VPN connection that allows any user to access to this machine. All experiments in this paper are made using python thanks to theano, a numerical computation library for Python that allows fast calculation and keras, a open source library for neural network that provide fast tool to create complex and efficiency neural network.

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Database

In order to assess our algorithms, we use two databases: MNIST and CIFAR. These databases have a reputation of superior quality. This is required to measure the performance of our methodology.

4.1.1 MNIST Database

The first database that was used is MNIST. That well-known database in machine learning provides 60.000 images of digits for training and 10.000 for testing. These 70.000 handwritten images have a 28×28 format. Figure 4.1 shows some digits from the database.

Figure 4.1: Image of one digit from the MNIST database.

4.1.2 CIFAR database

After assessing the algorithm on the MNIST database, the same tests are performed on the CIFAR database. Just like the MNIST database, CIFAR is also well-known in machine learning. It is a more complicated one than MNIST. Images from CIFAR do not represent 10 simple digits but objects from daily life such as airplanes, automobiles,

birds or ships. See figure 4.2. There are 10 different classes in total. The database contains 60 000 images in total with 50 000 images for training and 10 000 for testing.

Figure 4.2: Images from the CIFAR database.

4.2 Validation methods

This section describes the methods used to validate the results.

When the predicted class differs from the labeled class, an image is suspicious. The goal of the validation method is to differentiate, for these suspicious instances, label errors from classification errors. As explained in section 3.1, suspicious images are sorted according to our discrimination treshold Ω. Small Ω corresponds to mislabel; large Ω corresponds to misclassification.

To verify this hypothesis, we plot a Receiver Operating Characteristic (ROC) curve. The ROC curve is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied. The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings.

The images are sorted by ascending order of Ω and browsed one by one. This allows to verify if a misclassified image with a small Ω has more chance to be wrongly labeled than an image with a higher Ω. Figure 4.3 shows the percentage of detected label errors against the percentage of detected classification errors when browsing the sorted images from the smallest to the largest Ω. In other words, the graph represents the two different proportions of errors (label errors and classification errors) discovered when browsing the images from the smallest to the largest Ω.

The graph was created from the MNIST database with an addition of 4% of label error randomly distributed.

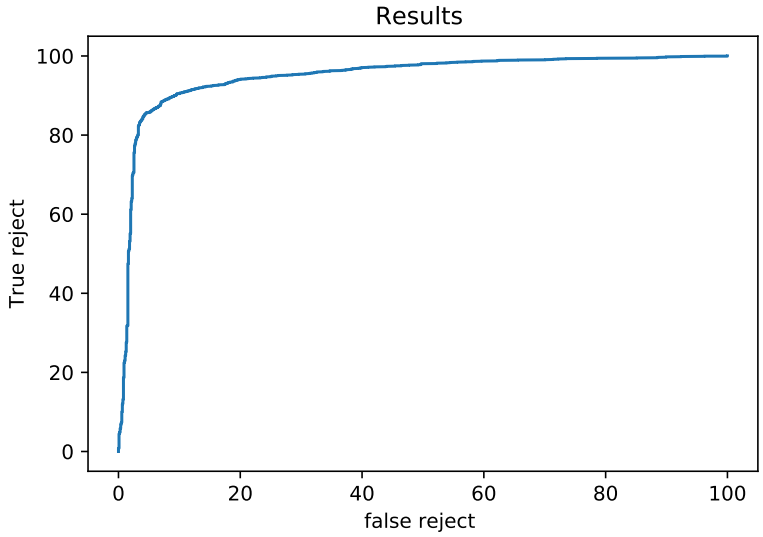

Figure 4.3: ROC curve representing the proportion of label errors (true reject) against the proportion of classification errors (false reject) detected when browsing the images sorted in ascending order of distance.

This graph shows that the proportion of label errors detected at low distance, i.e. in the beginning of the browsing of the images, is higher than the proportion of classification errors detected at the same low distance. In addition, from this graph, it can be observed that it will be possible to isolate a number of images containing about 90 percent of label errors and 20 percent of classification errors. This can be interesting if the user has a large number of training images.

Looking to these curves is nice but there is a drawback. Figure 4.4 shows two curves very close one from the other. It is indeed very difficult to select the best method from these two curves. Both curves use the MNIST database with 4% of error label added. The blue curve uses Δ (without variance) while the orange curve uses Ω (with variance).

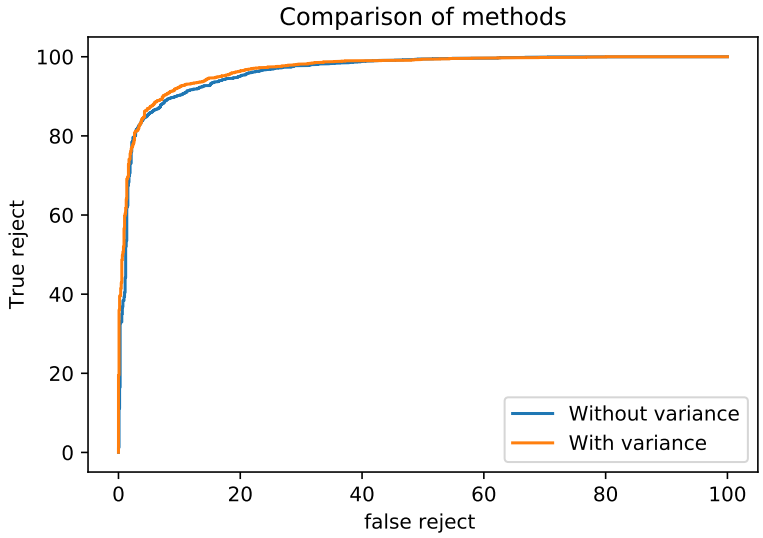

Figure 4.4: Comparison of two validation methods being the validation mixture and division.

Let's call AUR (Area Under Roc) the surface between the ROC curve and the diagonal which represents the non-discrimination line. Bigger is the surface above the diagonal, better is our discrimination algorithm. The machine learning community most often uses the ROC AUC statistic for model comparison.

Comparing these two areas will lead to a more accurate conclusion than visually comparing the two curves. A large area implies a fast increase of the curve. Indeed, the larger the area gets, the more the curve tends towards a step function, which correspond to an immediate increase of the curve i.e. immediate detection of label errors. Therefore, the larger the area, the faster the detection of label errors instead of classification errors. The best result will thus be obtained with the largest area under the curve.

Distance computation used on MNIST

As already explained previously, the output of the neural network is a set of probabilities. The number of output probabilities is equal to the number of different classes. Thus, when using the MNIST database, the output of the neural network has a dimension of 10 as they are 10 possible classes being the 10 different digits. Once the neural network trains on the database, each image in it is represented by a vector of dimensions 10. Moreover, by working in 10 dimensions, this vector can actually be called a point. Therefore, it is considered that every image of the MNIST database is represented by a point in 10 dimensions.
During this work, different methods have been tested to optimize a unit of measure that could differentiate error and label error. Therefore using the center of the class cluster as reference for different distance was not always the choose of the methods. Some other option have been studied as the k-nearest neighbor or choosing a number of random point. As a example, the average Euclidean distance between a point, i.e. representing an image of the database, and the k nearest neighbors of the expected label is calculated. Figure 4.5 shows the obtained result according to the variable k.

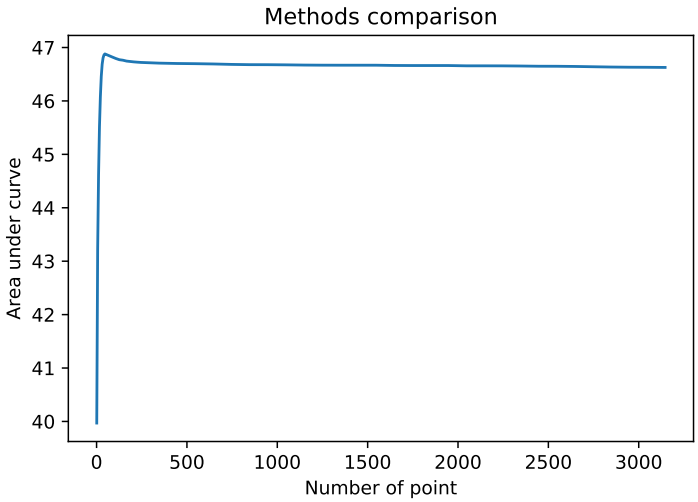

Figure 4.5: Result obtained on the MNIST database using the Euclidean distance of the k nearest neighbors of the expected label.

As one can see on the graph, increasing the number of nearest neighbors taken into account for the calculation of the Euclidean distance has a positive impact. But this is only the case at low values of k. Indeed, after reaching a maximum, the curve slightly decreases. Using the k value corresponding to the maximum seems legit but it is not

relevant. Choosing a pertinent number of nearest neighbors is a difficult task and can depends on the application. For this reason, the average of all label point i.e. the center of the label point are choose to computer the distance despite the lower result. This choice has an additional advantage of fast computation since only one distance must be calculated.

The area under the curve was also used to differentiate the multiple improvement between the basic distance D_i^{basic} in the beginning of the section 3.1.2 and Ω explained at the end. The table below show the result on the MNIST database with 4% of label error added.

	Area comparison on MNIST			
Method used	Distance D^{basic}	Γ	Δ	Ω
Area under curve	94.93	94.96	96.41	96.96

4.3 Results MNIST

The results obtained on the MNIST database are described in the following sections. For every result that is presented, the convolutional neural network described in section 4.3 was used. In addition, distances were calculated according to the method explained in section 4.2.

Convolutional neural network used on MNIST

The training of the MNIST database was done with a classical convolutional network. It consists of five different layers as detailed below.

1. One convolutional layer of 24 neurons.
2. One convolutional layer of 48 neurons.
3. One convolutional layer of 48 neurons.
4. One convolutional layer of 64 neurons.
5. The dense layer of 10 neurons in order to have an output for each class.

Each convolutional layer takes a receptive field (region of pixels as explained in chapter 2 section 2.2.1) of 5×5 pixels leading to a number of 26 parameters per neuron. Indeed, 25 parameters for the weight, i.e. one for each pixel of the 5×5 region, and 1 parameter for the bias.

Label error detection

To begin, the label error detection algorithm was applied to the MNIST database. However, as the database does not contain any label errors, they had to be created. This was done by randomly choosing images in the training set and by subsequently changing their label so as to create a label error. In addition, the number of label errors created this way is chosen by the user. In this case, it was chosen to make four training sets containing respectively 1/3, 1/10, 1/20 and 1/25 of images with label errors. As the training set has a size of 60 000 images, it means that for a fraction 1/10 of images with label errors, 54 000 images will be correctly labeled whereas 6000 images will have a wrong label. Furthermore, the training is done during one epoch. Indeed the long training of a neural network with label error can lead to overfitting. As result, the methods is less efficiently because image with label error are not always misclassified. Therefore, the methods is unable to detect them as label error. This is shown in figure 4.6. This graph shows the evolution of the area during the training of the network when no weight

are change. It show that the methods would have higher performance if it is use during the first epoch.

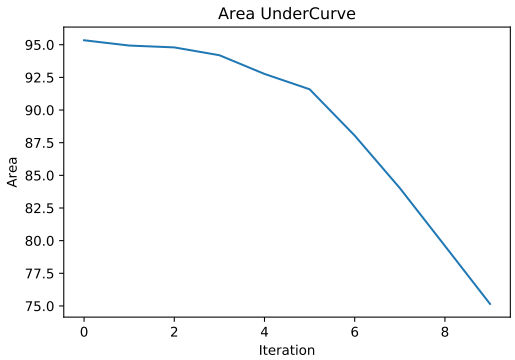

Figure 4.6: Evolution of the area during each training of the network without modification on the weight.

Next, the label error detection algorithm was applied to each of the four created training sets with label errors. One must note that the testing set was not modified and stayed identical for the different tests to make results comparable. Also, the testing set contains 10 000 images.

Table 4.1 below contains the obtained results expressed in number of correctly detected images from the testing set.

	Fraction of images with label errors			
Method used	1/25	1/20	1/10	1/3
Without label error detection	9858	9867	9822	9740
With label error detection	9902	9894	9850	9830

Table 4.1: Number of correctly detected images from the MNIST database, depending on the method and the fraction of images with label errors.

The results illustrate that applying the label error detection algorithm on the training set before training improves the number of correctly detected images in the testing set.

To analyze the results more deeply, the case 1/25 (i.e. out of the 60 000 images in the training set, 2400 have a label error) is described in detail. Figure 4.7 shows the result obtained after applying the label error detection algorithm on the training set containing 1/25 images with label errors. In this graph, the number of true rejects, i.e. label errors, are plotted against the number of false reject, i.e. classification errors. Moreover, both are expressed in percentages.

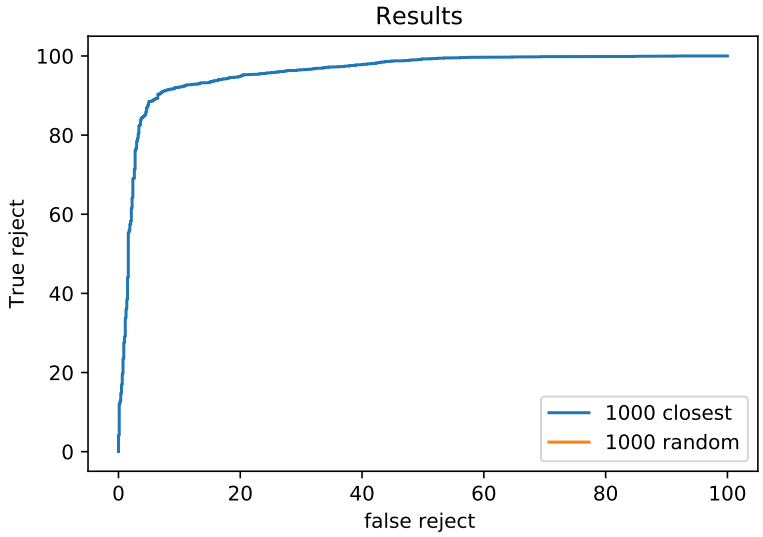

Figure 4.7: Graph of true rejects (label errors) against false rejects (classification errors) for the case 1/25 of training set images have a label error.

As one can see on Figure 4.7, for 10% of false rejects, the algorithm could remove more than 85% of the true rejects. However, in reality, databases are not known nor controlled. Therefore, it is impossible to know how much images must be removed based on their calculated distance in order to be sure to remove these 85% of images with label errors. The only hypothesis that is assumed to be true is that the lower the calculated distance, the higher the chance that the image has a label error. In practice, several methods can be employed to limit the impact of label errors on the detection rate by using the calculated distance:

Table 4.1 was made using the calculated distance to train the model. A weight between 0 and 1 depending on the calculated distance was attributed to each image. The weights are thus normalized according to the formula described in section 3.1.3. Figure 4.8 shows the weights attributed to images with label errors or true rejects. As can be seen on the graph, the images with label errors have very small weights. This again illustrates and confirms that images with label errors have a small calculated distance resulting in a small weight. The distance computation is thus effective and representative of a certain probability of images having a label error.

On the other hand, Figure 4.9 shows the weight attributed to images with classification errors or false rejects. Those weights were attributed quite uniformly over the distances, however, extreme values of weights seem to occur more frequently than other values. Ideally, all weights of images with a classification error should be equal to 1 as these images should all be taken equally and fully into account during training.

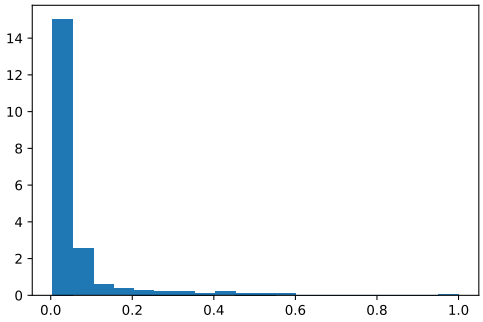

Figure 4.8: Normalized histogram of weights attributed to images with label errors.

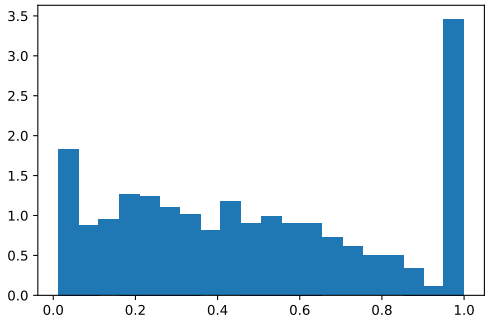

Figure 4.9: Normalized histogram of weights attributed to images with classification errors.

Uncertainty detection

Figure 4.10 shows the comparison between the correct classified image and the misclassified image when they are ordered according to Ω. The graphs show that by rejecting the high distance, it is possible to reject approximatively 90 % of the error for 10 % of the correct classified image.

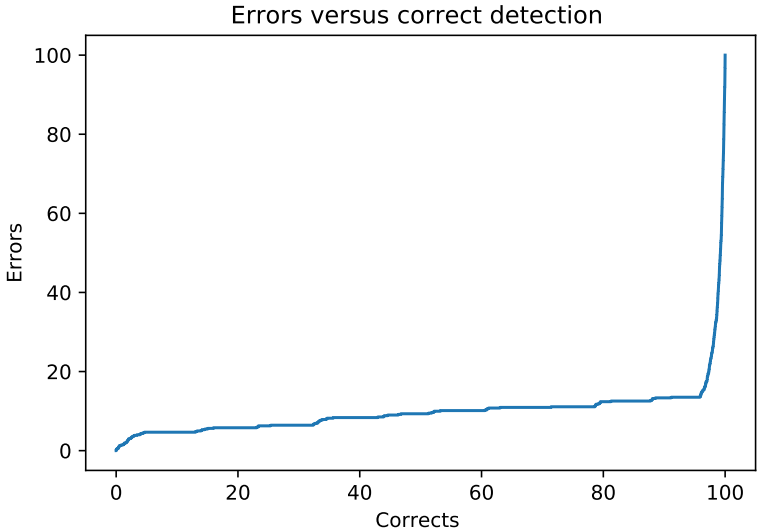

Figure 4.10: Graph of correct detection against error (misclassification errors).

4.3.1 Distance computation used on CIFAR

The distance computation used on the CIFAR database is the same as the one used on the MNIST database. Details are given in previous section 4.2.

4.4 Results CIFAR

The results obtained on the CIFAR database are described in the following sections. For every result that is presented, the convolutional neural network described in the pre-

vious section 4.4 was used. In addition, as already mentioned, distances were calculated according to the method explained in section 4.2.

Convolutional neural network used on CIFAR

The training of the CIFAR database was done with a classical convolutional network. Due to the higher complexity of the database, a more complicated network was used in comparison with the network used on the MNIST database. It consists of several different layers but also uses dropout and max pooling as described below.

1. Two convolutional layers of 32 neurons.
2. Three convolutional layers of 64 neurons.
3. Two convolutional layers of 128 neurons.
4. One dense layer of 250 neurons.
5. Use of dropout for the last four layers.
6. Maxpooling between the convolutional layers that change in size.
7. The dense layer of 10 neurons in order to have an output for each class.

Each convolutional layer takes a receptive field of 5×5 pixels leading to a number of 26 parameters per neuron.

Label error detection

First, the label error detection algorithm was applied to the CIFAR database. Just as in the MNIST database, there are no label errors in the CIFAR database. The label errors had to be created in the same way they were for MNIST. For more details on this, the reader can refer to section 4.3.

Table 4.2 below contains the obtained results expressed in number of correctly detected images from the testing set. One must note that the testing set contains 10 000 images.

	Fraction of images with label errors				
Method used	0	1/25	1/20	1/10	1/3
Without label error detection	7924	7857	7741	7581	7154
With label error detection	7994	7910	7941	7912	7408

Table 4.2: Number of correctly detected images from the CIFAR database, depending on the method and the fraction of images with label errors.

Without label error creation, the model is able to correctly detect 7924 images. It is interesting to remark that even without label error creation, the result is better with the algorithm than without. The reason might be the dropout, which creates randomness in the test. Next, just as for the MNIST database, the results illustrate that applying the label error detection algorithm on the training set before training improves the number of correctly detected image in the testing set.

In order to examine the results in depth, the case 1/25 is detailed. Figure 4.11 shows the results obtained for that case.

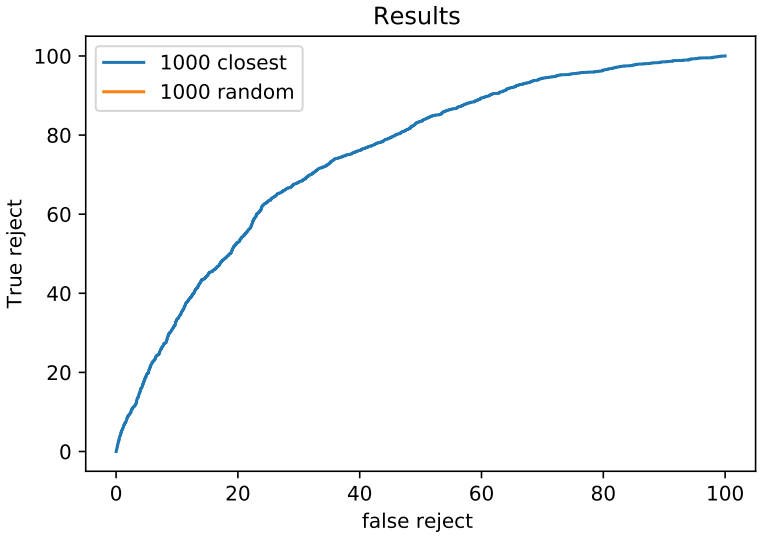

Figure 4.11: Graph of true rejects (label errors) against false rejects (classification errors for the case 1/25 of training set images have a label error.

As one can see on Figure 4.11 by suppressing less than 20% of false rejects, it is possible to eliminate 50% of the true rejects. This result shows that with a more complicated database, i.e. CIFAR is more complicated than MNIST, the network is less efficient for label error detection. However, the method still works but the performance is lower. Indeed figure 4.12 and 4.13 show the weight associated to the label error and the error of misclassification from the network. These histogram shows that the weight for the label error are lower than for a error of misclassification. The impact of the image associate to these label error will be reduced during the training.

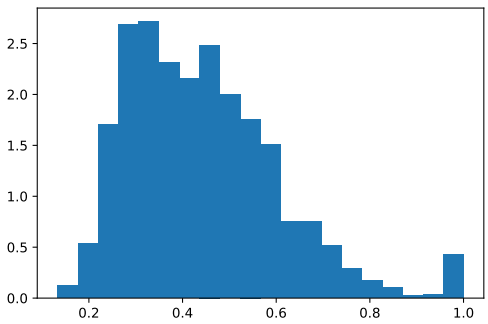

Figure 4.12: Normalized histogram of weights attributed to images with label errors.

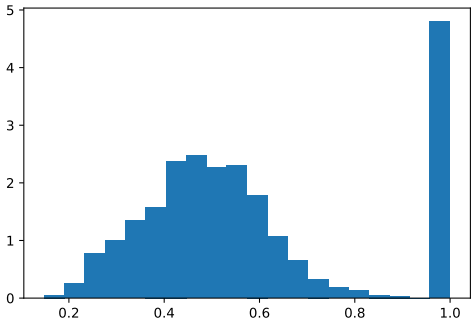

Figure 4.13: Normalized histogram of weights attributed to images with classification errors.

Uncertainty detection

Figure 4.14 shows the comparison between the correct classified image and the misclassified image according to the distance. The graphs show that by rejecting the high distance, it is possible to reject approximatively 90 % of the error for 40 % of the correct classified image. The gain is lesser than previous case since the network works less good.

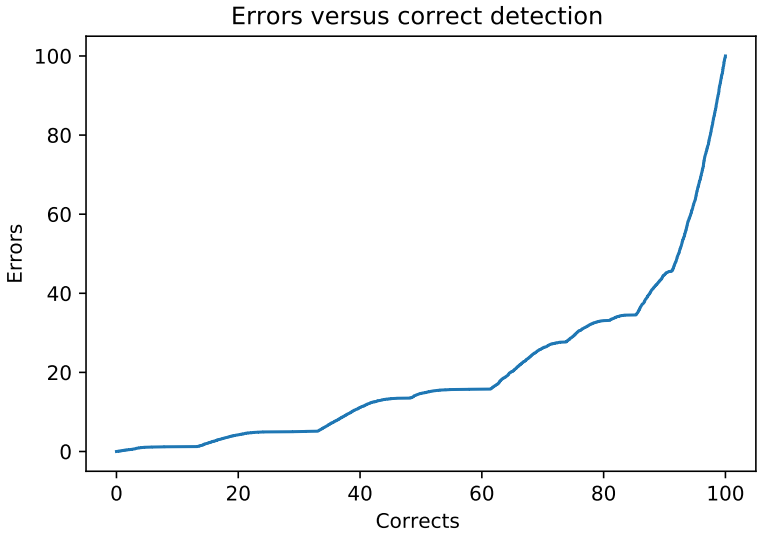

Figure 4.14: Graph of correct detection against error (misclassification errors).

4.5 Discussion of the convolutional neural network for label error detection

This section aims at studying the convolutional neural network for label error detection in depth. The previous results on the MNIST and CIFAR databases showed that the method did improve the final detection rate. Next, the goal is to further optimize the method by adapting the convolutional neural network. Therefore, three main questions were studied:

1. Does using the label error detection algorithm during more than one training iteration improves the results?
2. Does the number of errors impacts the methods?
3. What is the best network to use?

To answer these questions, the label error detection algorithm will be applied to the MNIST database only. This choice is driven by the fact that the MNIST database is less complicated than the CIFAR database. Consequently, the convolutional neural network used on the MNIST database is less complex and allows a better understanding of it making primary optimization easier. In addition, computation time is shorter for the MNIST database. Lastly, the used convolutional neural network is the same as previously used on MNIST, for more details the reader can refer to section 4.3.

4.5.1 Impact of label errors on CNN performance

To begin, it is interesting to verify whether label errors have an impact on the performance of the neural network measured by the image detection rate or the number of correct image detections. In addition, it can be assessed whether this impact is significant of not. Therefore, the convolutional neural network was applied to the MNIST database with and without label errors. Figure 4.15 shows the CNN performance when the MNIST database does not contain any label errors. Whereas Figure 4.16 shows the CNN performance when 1/25 of images from the training set contain a label error. As

already mentioned, the training set contains 60 000 images, meaning that 2400 images have a label error. The testing set always contains 10 000 images.

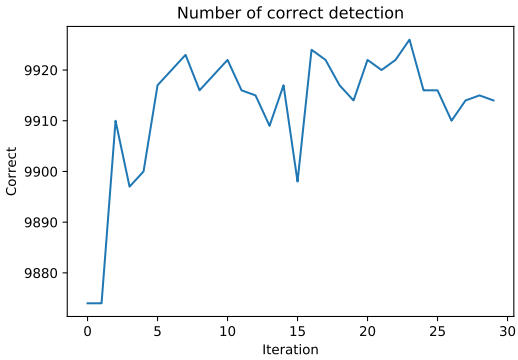

Figure 4.15: Evolution of the performance in function of the number of training iterations for the MNIST database without label errors.

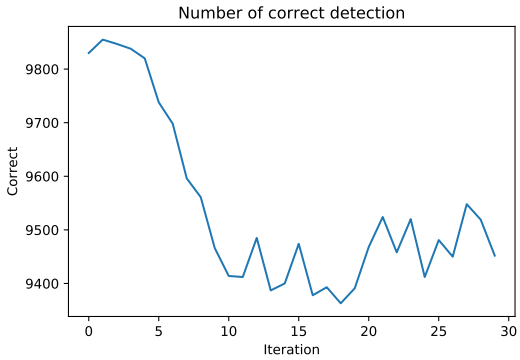

Figure 4.16: Evolution of the performance in function of the number of training iterations for the MNIST database with 1/25 of training set images having a label error.

Without label errors, the CNN has a standard behavior in the sense that adding more epoch or training iterations improves the performance. More specifically, training improves the performance during the first epoch. Then the performance is more consistent and oscillates between 9910 and 9920 correctly detected images. In conclusion, results show that training has a positive impact on CNN performance in the beginning. At high number of iterations, the impact does not seem to be significant.

On the other hand, with label errors, the CNN performance shows an opposite behavior. During the first epochs, the performance is rather steady with a small peak. In this case training does not significantly improves CNN performance during the first iterations. Moreover, as training iterations increase the CNN performance drops until reaching a steadier performance. However, even at high number of iterations, the performance still oscillates with a large amplitude compared to the steady performance at high number of iterations without label errors. In conclusion, this result highlights the particularity of training with label errors where CNN performance decreases with the number of training iterations until reaching a steady performance. In addition, overall performance is weaker with label errors compared to obtained performances without label errors.

4.5.2 Impact of label errors detection with proportional weights on CNN performance

Next, it has been shown in the previous section that label errors negatively affect the CNN's overall performance but also change the behavior of the performance in function of training iterations. To improve CNN performance on databases with label errors, the label error detection algorithm with weights is used on the training set. The weights are attributed to misclassified images after one epoch of training. They vary between 0 and 1 and are proportional to the calculated distance. Moreover, the weights are only calculated once after the first training and do not change for the 29 remaining epochs of training. The obtained CNN performance is shown in Figure 4.17.

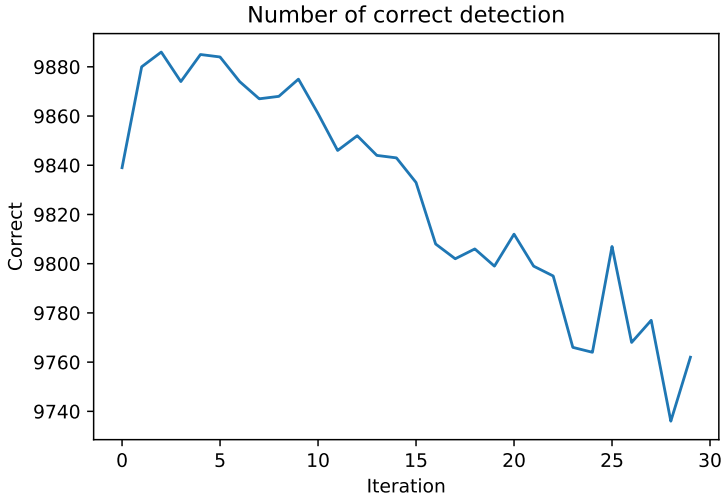

Figure 4.17: Evolution of the performance in function of the number of training iterations with 1/25 of training set images having a label error. The label error detection algorithm with proportional weights was applied once after the first epoch.

From this result, two observations can be made.

First, by comparing Figure 4.17 and 4.16, one can see that the overall performance of the CNN on a database with label errors is better when the label error detection algorithm is applied. More specifically, the latter does not significantly affect the CNN performance in the first iterations. Indeed, in Figure 4.17, the number of correct image detections varies between 9840 and 9885 during the five first iterations, whereas in Figure 4.16 it varies between 9820 and 9850. The difference between both is thus rather small in the first five iterations. However, looking at the performance during the five last iterations, one can see in Figure 4.17 that it oscillates between circa 9800 and 9740. Whereas in Figure 4.16 it varies between circa 9450 and 9550. The CNN performance during the five last iterations is thus significantly improved when using the label error detection. However, even with the label error detection algorithm, the CNN performance in Figure 4.17 does not reach the performance obtained on a database without label errors as shown in Figure 4.15. This indicates that there is room for improvement of the label error detection method.

Secondly, the behavior of the CNN performance remains the same, i.e. the performance decreases as the number of iterations increases until reaching a steadier performance. Although performance is enhanced, it does not show the same behavior as in a database without label error where performance increases with the number of iterations.

To better understand this particular behavior, Figure 4.18 shows the percentage of correct labels and label errors that were matched by the CNN in function of the training iterations. The blue curve thus shows the percentage of correctly labeled images that were matched by the CNN. This curve is always very close to 1 meaning that almost 100% of images with a correct label were identified by the CNN. On the other hand, the orange curve shows the percentage of images with label errors that were matched by the CNN or in other word the percentage of images for which the CNN prediction fit the label error. If the label error detection algorithm would perfectly work, the orange curve should always be at 0. However, as can be seen on the graph, another behavior is observed. During the first iterations, the method does not fit label errors, i.e. the

orange curve is close to 0. But as iterations increases, the CNN seems to try to fit the label errors resulting in an increasing orange curve.

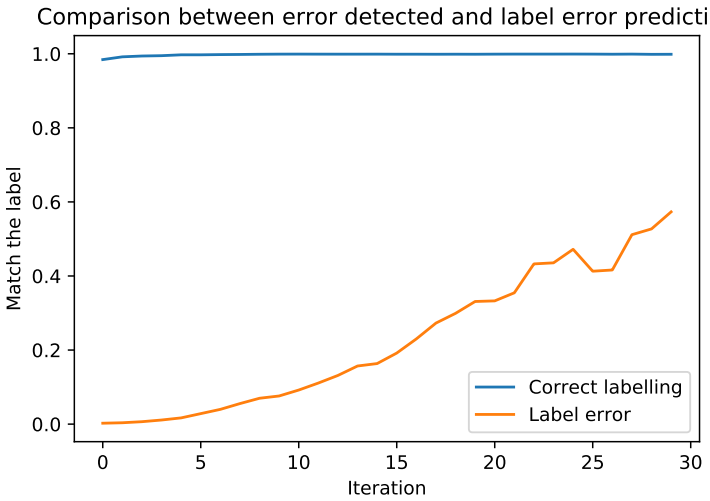

Figure 4.18: Percentage of matched labels in function of the number of training iterations. Blue curve: percentage of correctly labeled images that were matched by the CNN. Orange curve: percentage of badly labeled images (label errors) that were matched by the CNN.

The result indicates that the label error detection algorithm works in the first iterations but is no longer efficient after an increasing number of iterations. This might be due to the fact that the CNN tries to fit the label errors at any cost. It is expected that the CNN first fit the label errors with the higher weight before starting to fit the label errors with lower weight. This means that the CNN overfits. As label errors are matched as if they were correct labels, final image detection is not as good as in the database without label errors. In the following sections, several possibilities are explored to improve results at increasing iterations.

4.5.3 Impact of label errors detection with zero weights on CNN performance

A first possibility that is analyzed to improve image detection rates is to put the weights of the label error detection at zero. In the previous section, those weights were proportional to the calculated distance, which correspond to the risk of being a label error. By using a purification method with weights at zero, all the misclassified images are removed from the training set. Consequently, images with label error are likely to be removed from the training set. However, this comes at a cost as misclassified images without label errors will also be removed by this process. This was already explained in section 4.2. The main advantage of this zero weight method is that it is likely that a high proportion of images with label errors will be removed from the training set. The results obtained with this method are given in Figure 4.19.

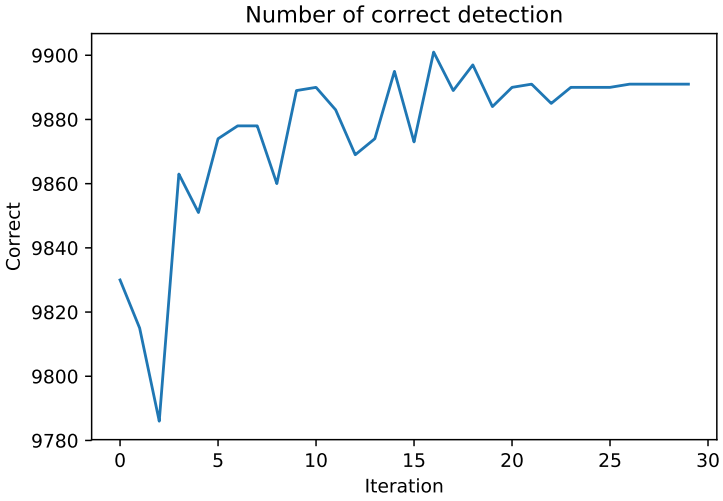

Figure 4.19: Evolution of the performance in function of the number of training iterations with 1/25 of training set images having a label error. The label error detection algorithm with zero weights was applied once after the first epoch.

The result shows a way more stable performance that is actually comparable with the results obtained on a database without label errors as shown in Figure 4.15.

Regarding the overall values of performance, the result shows that the method improves image detection significantly at high number of iterations in comparison with the result obtained in Figure 4.17. The results in the last five iterations are not as good as with a database without errors were the performance reaches 9915 correctly detected images (see Figure 4.15) but are close to them with 9885 correctly detected images.

Next, regarding the behavior of the performance in function of the number of iteration, one can see that it is the same as with a database without label errors. Meaning that the performance increases with the number of iterations. The main difference though is in the first three iterations, where performance decreases, this was not observed in a database without label errors. It might be due on one hand to the fact that a significant number of images are removed from the training set after the first iteration. Less images to train on means a less efficient training. On the other hand, the main reason might be the fact that the training is based on the learning from the errors. As all misclassified images, i.e. with errors, were removed, the training did not actually help the CNN to improve its detection based on its mistakes. This could explain the drop in performance during the first iterations. After that, the CNN trains on the same training set without any further image removal resulting in an improved performance.

To conclude, this result is very good as the performance is the best yet obtained on a database with label errors. In addition, the classical behavior of the performance, i.e. the same as in a database without label errors, is more or less obtained. A hypothesis to explain this "classical" behavior, is that all images with label errors were removed. Indeed, if this hypothesis was true, it could explain why the same behavior as in a database without label errors is observed. Furthermore, as the method puts a weight zero at all the misclassified images, all of them are removed from the training set. In addition, as already described in section 4.2, the smaller the computed distance the higher the chance of being an image with a label error. But more importantly looking at Figure 4.7, all images with a label error have an attributed distance as the curve reached

100% on the Y-axis. Therefore, it is likely that all images with a label error were removed by the zero weight method. For future improvements, it could be interesting to run the same test with the CIFAR database and see whether results are consistent with the ones obtained on the MNIST database.

4.5.4 Impact of iterative label errors detection with proportional weights on CNN performance

In the previous section, the label error detection algorithm was applied once with zero weights. In this section, the label error detection algorithm is applied with proportional weights as in section 4.5.2, but the difference is that the weights will be updated after each epoch of training instead of only being computed once after the first epoch. The goal of this test is to assess whether recalculating the weights after each epoch of training allows to obtain better CNN performances.

The results obtained with this method are given in Figures 4.20 and 4.21.

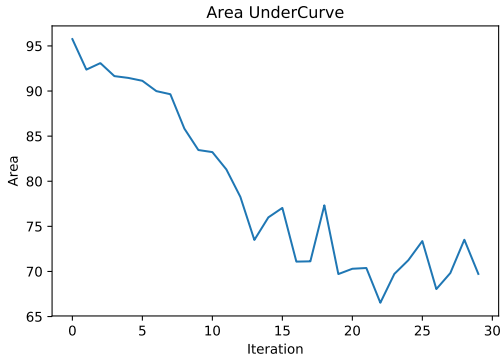

Figure 4.20: Evolution of the area under the curve of true rejects against false rejects in function of the number of training iterations with 1/25 of training set images having a label error.

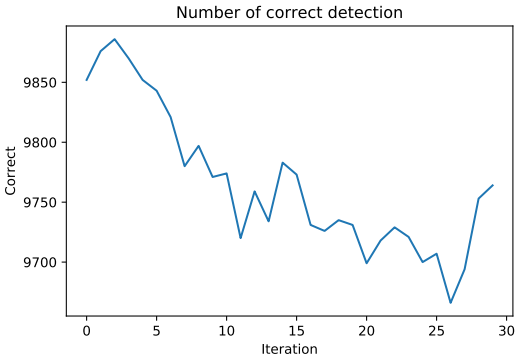

Figure 4.21: Evolution of the performance in function of the number of training iterations with 1/25 of training set images having a label error.

Figure 4.20 illustrates the evolution of the area under the curve of true rejects against false rejects as defined in section 4.2. The more this area decreases, the less likely it is that low distance images have a label error. Consequently, as the computed distance is less and less representative of the likelihood of having a label error, the algorithm is no longer efficient. From the figure it can be seen that the area under the curve decreases by about 30% between the first and the last iteration. This decrease is significant and has a negative impact on the algorithms efficiency.

This is further confirmed by the results shown in Figure 4.21. The iterative weights give a better CNN performance during the first epoch but then it drops. Indeed, as the number of iterations increases, the number of correctly detected images decreases. However, there is a significant increase during the four last iterations but this might be a coincidence. Furthermore, the CNN performances obtained with iterative weights (adapted after each iteration) are very similar to the ones obtained with the weights that are only calculated once (see Figure 4.17). This similarity is related to the overall CNN performance values and to the behavior of the performance in function of the number of iterations.

In order to better understand why updating the weights after every iteration does not improve the CNN performance, the weights after 13 iterations were plotted. The 13^{th} iteration was chosen because from then on, the performance becomes more stable. The weights attributed to images with label and classification errors are shown in Figure 4.22 and 4.23 respectively.

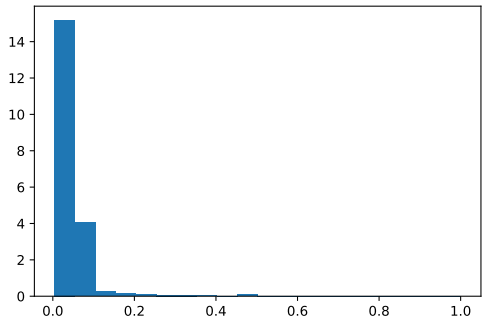

Figure 4.22: Normalized histogram of weights attributed to images with label errors.

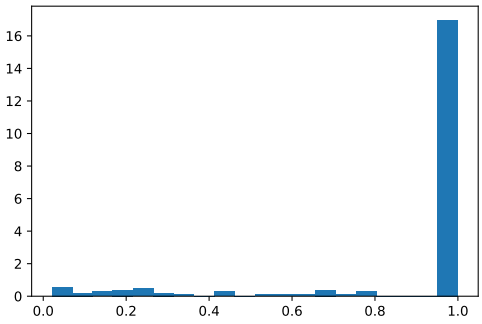

Figure 4.23: Normalized histogram of weights attributed to images with classification errors.

It appears that after 13 iterations, images with label errors have very low weights whereas images with classification errors have high weights. This means that the weights are still attributed in the expected way. Consequently, the computed distance must still be representative of the likelihood of an image having a label error. In conclusion, the weights show that the label error detection algorithm is still efficient. Thus, the bad CNN performance cannot be attributed to it.

To go one step further in understanding the bad CNN performance, Figure 4.24 shows the percentage of correct labels and label errors that were matched by the CNN in function of the training iterations. In addition, Figure 4.25 represents the maximum computed distance in function of the training iterations.

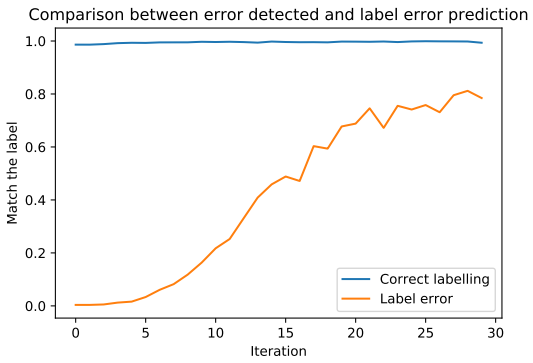

Figure 4.24: Percentage of matched labels in function of the number of training iterations for 1/25 of images containing label errors.

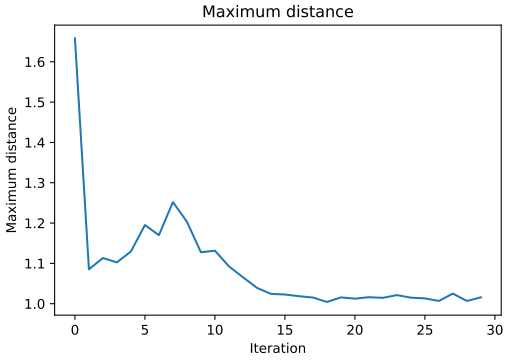

Figure 4.25: Maximum computed distance in function of the number of training iterations.

In Figure 4.24, the blue curve represents the images where the prediction fit the correct label. This curve is always close to 1 as it should. The orange curve represents

the images where the prediction fit the label error. The curve should be at zero as in the first iterations. However, the curve actually increases as the number of iteration increases. It shows that the network overfits the label error despite them having a very low weight. The network is fitting the label errors step by step to finally have a fit on circa 80% of the label errors.

In Figure 4.25, one can see that the maximum computed distance reduces significantly after one iteration. High distances should correspond to images with a higher likelihood to be misclassified. After one iteration, the network has trained and classification errors should decrease (this is illustrated by the performance increase after one iteration in Figure 4.21). Consequently, the maximum distance could decrease because the likelihood of misclassification decreases, but this is just a hypothesis. As of iteration 7, the maximum distance is reduced at each iteration, which could explain the step by step evolution of observed in Figure 4.24. Indeed, as the maximum distance decreases, the weights attributed to low distances increase due to the fact that the weights are normally proportional to the distance.

This behavior stays identical when the number of label errors is increased as shown in Figure 4.26 where 1/3 of training images have a label error. The CNN kept learning and fitting on the label errors. This case is even worse as a lower percentage of correctly labelled images are identified. The resulting CNN performance is thus bad as shown in Figure 4.27.

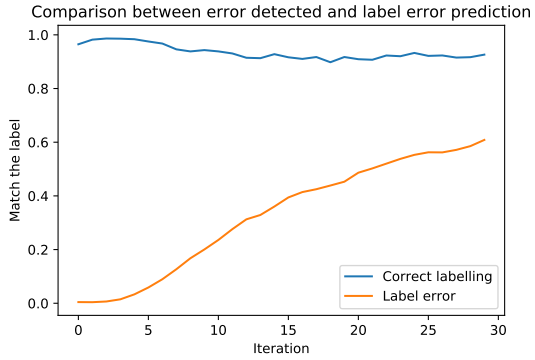

Figure 4.26: Percentage of matched labels in function of the number of training iterations with 1/3 of training images having a label error.

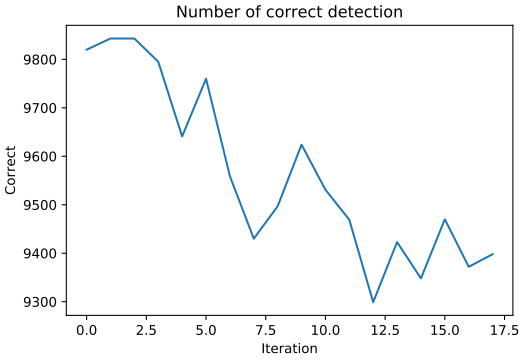

Figure 4.27: Evolution of the performance in function of the number of training iterations with 1/3 of training images having a label error.

4.5.5 Iterative weight

In the previous section, the label error detection algorithm was applied with proportional weights after each epoch of training. In this section, the weights will also be calculated again after each iteration but previously calculated weights will also be taken into account. The weights will thus be modified in an iterative way and not only be completely computed again each time. This allows to take into account the previous values of distances instead of erasing them. Therefore, an image that had a high distance after the first iteration will keep this high distance for the next iterations. The weight update method consists of five steps:

1. Calculation of every distance and associated weight based on Ω. Computed weight for image i with $W_i = \frac{\Omega_i}{\Omega_{max}}$. This is only done after the first epoch of training.

2. At the end of each iteration, the distances and associated weights are saved as the previous ones.
3. The CNN is trained again for one more epoch, new Ω and associated weights are calculated.
4. The weight of every image is updated following the formula: $W_i = W_{i-1} + \frac{\Omega_i - \Omega_{i-1}}{\Omega_i^{max}}$ with W_{i-i} and Ω_{i-1} respectively the weight and the Ω at the previous iteration, $i-1$.
5. Reiteration starting from step 2.

The result obtained with this method is given in Figure 4.28.

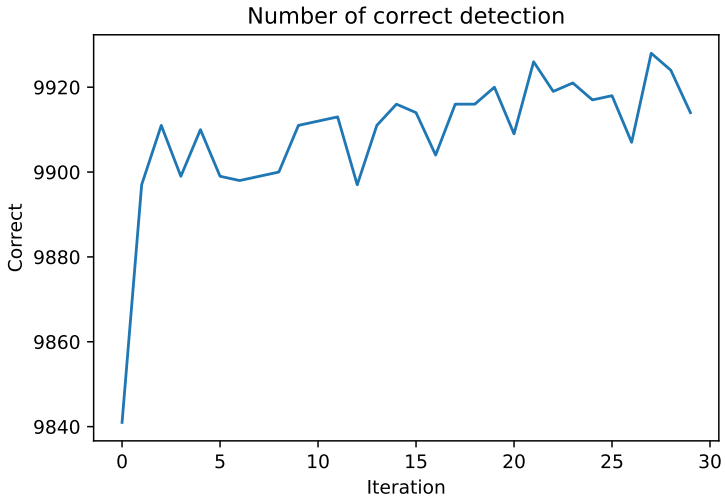

Figure 4.28: Evolution of the performance in function of the number of training iterations with 1/25 of training set images having a label error.

This graph looks like there is no label error add. To compare with figure,[?] it almost give the same performance.

4.5.6 Size of the network

The results can be slightly different according to the network. Indeed, the complexity of a network can interfere on how well the network will learn. To begin, we study a complex neural network composed of:

1. Two convolutional layers of 64 neurons.
2. Two convolutional layers of 128 neurons.
3. Two convolutional layers of 256 neurons.
4. One dense layer of 100 neurons.
5. One dense layer of 250 neurons.
6. The dense layer of 10 neurons to have an output for each class.

Figure 4.29 shows how the training is affected by label errors. In blue, the graph shows the images with correct labels that are properly classified by the network. The orange curve represents the images with a label error that were fit by the network. In other words, the network predicted the label but the label is actually an error. The figure shows that a neural network first tends to generalize the image that is why the

correct labels are first well detected whereas the label errors are not matched. It is only after a certain amount of epoch that the network tends to overfit and learns from the label errors.

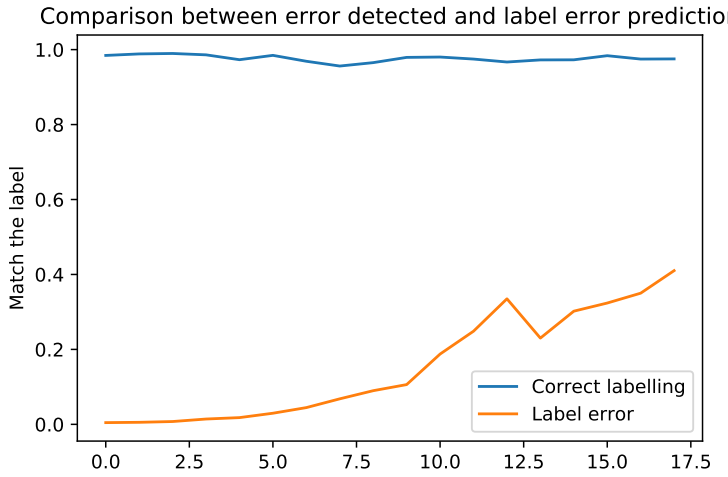

Figure 4.29: Percentage of matched labels in function of the number of training iterations with 4% of training images having a label error in a complex network.

The same simulation was made with a really small network to compare results. The used network is made of:

1. One convolutional layer of 6 neurons.
2. One convolutional layer of 12 neurons.
3. The dense layer of 10 neurons to have an output for each class.

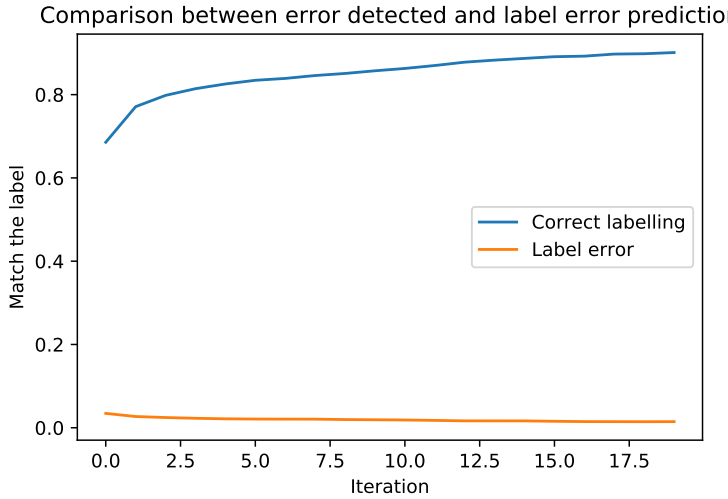

Figure 4.30: Percentage of matched labels in function of the number of training iterations with 4% of training images having a label error in a small network.

In a small network (Figure 4.30), the label errors have less impact on the training. The network is not complex enough to take into account every single image. Consequently, the label errors are not matched. The number of parameters is not high enough

to fit every label error and those are seen as outliers. On the other hand, the network is not complex enough to have good results either as it cannot outperform the previous complex network. This is shown with Figure 4.31. Indeed, the number of correctly classified images stays below 9700 unlike the more complex network. However, the performance does not drop to much and stays above 9550 after 30 epoch showing a good robustness to label errors.

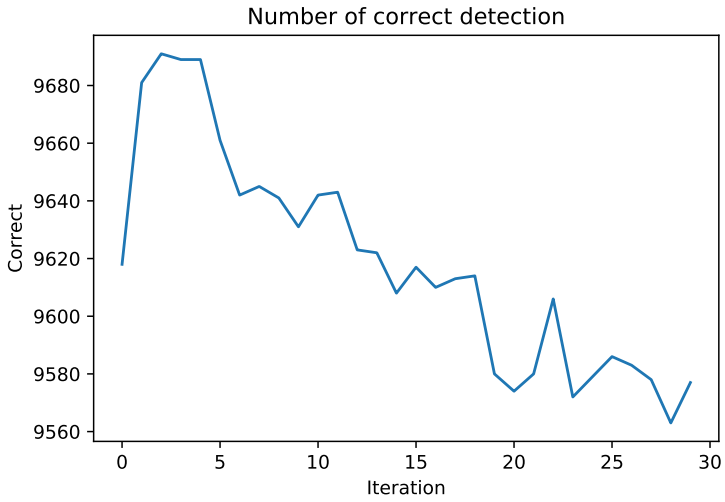

Figure 4.31: Evolution of the performance in function of the number of training iteration for the MNIST database with a small network and 4% of label errors.

4.6 Results on license plate

The final test is made for detecting the license plate from a image. 233736 image of charactere are use for training the network and 6850 image are provide for the test.

4.6.1 Convolutional neural network used on MNIST

The training of the license plate was done with a convolutional network. This was composed of :

1. 2 convolutionals layers of 64 neurons.
2. One layer of max pooling with 2X2 pixels as input shape.
3. 2 convolutionals layers of 128 neurons with dropout of probability 0.5.
4. One dense layer of 100 neurons.
5. The dense layer of 10 neurons in order to have an output for each class.

As for the previous neural network, each convolutional layer takes a receptive field (region of pixels as explained in chapter 2 section 2.2.1) of 5×5 pixels leading to a number of 26 parameters per neuron. The network detect 95% of the image correctly. Figure 4.33 show the performance of the network during multiple epoch.

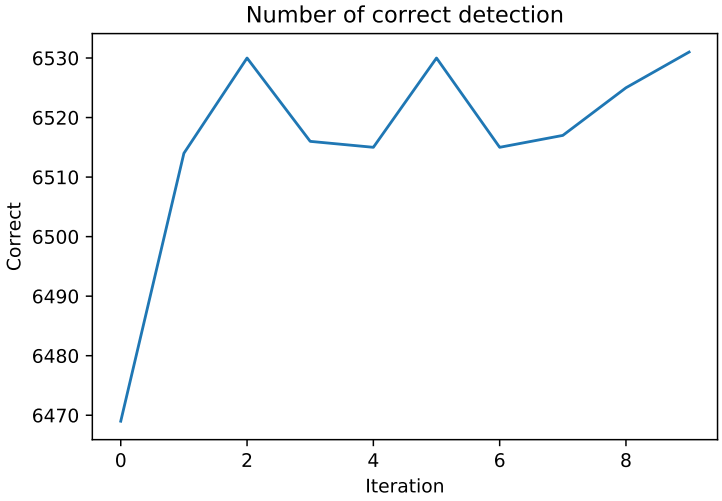

Figure 4.32: Evolution of the performance in function of the number of training iterations. The number of detection is 6850.

4.6.2 Label error analyze

To measure the algorithm, 240 label errors are added to the database.

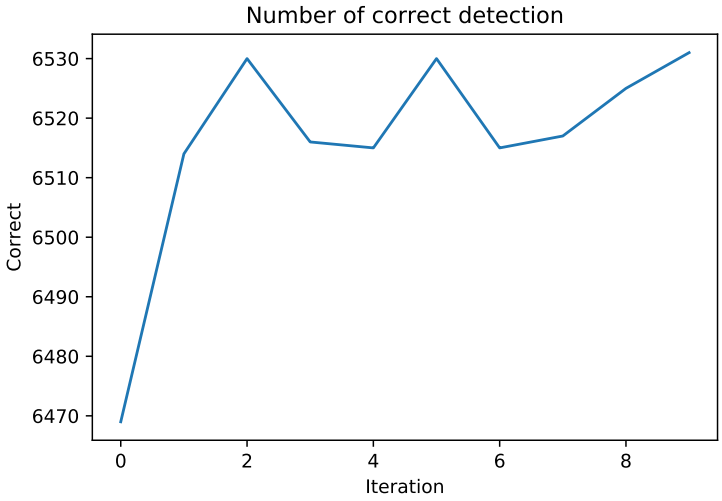

Figure 4.33: Graph of correct detection against error (misclassification errors).

Other results will be provide with an addendum. These results will be presents in the private defense.

4.7 Future improvements

In this section, future possible improvements for label error detection methods are described.

Iterative weight adaptation

Once the label error detection is done, a lot of different methods can be applied to modify the weight associated to the training images. The iterative weight method used

in this paper is linear. However it is possible to imagine other formula as this following one :

$$W_i = \frac{1 - e^{D_i}}{1 - e^{D_{max}}}$$

This formula keeps some identical properties :

1. If the distance is equal to 0, the probability to be a label error is maximal and the weight must be put at 0.
2. Since the maximum weight in this case is 1. The weight 1 must be associated to the image with the highest distance. In this case, the weight is equal to 1 when the distance reaches D_{max} i.e. the maximum distance

Several formula can be studied in order to obtain a better solution than the one suggested in this paper.

Different weight ranges

In this paper, the weights are normalized between 0 and 1. A weight of 0 is ideally attributed to an image that has a label error so that the image has a 0 or limited impact on training. The other images are considered as useful for the training and their weights are expected to be close to 1.

However, a misclassified image is extremely useful to train the network because its information has not been acquired. Therefore, we should investigate method boosting misclassified images by allocating them weights well above 1.

With the same mindset, the weights associated to an image that is well classified can be decreased if this image is similar to one other. In other way, the weights can be used to reduce the impact of images that look like each other to reduce the duplications and allow a better training of images where the information is not acquired.

In other words, the weights can be used not only to reduce the impact of mislabeled images but also to increase the impact of images rich for the training.

CHAPTER 5

CONCLUSION

Label errors can have a negative impact on the training of a convolutional neural network for image classification. Consequently, the learning of these label errors can lead to a decrease in overall performance with lower than expected image detection rates. Even with a well trained convolution neural network, a decrease in performance can be observed.

Different solutions exist to deal with label errors in order to diminish their impact on overall performance. As a first solution, some classification methods, like bagging, are not significantly affected by label errors and can thus be an alternative method to use on databases with label errors. Another solution is to use filtering techniques, which allow the user to modify the database in order to suppress images with label errors. The last option is to used techniques that train the classificator differently so as to take into account label errors.

The method presented in this paper defines multiple criteria to distinguish label errors from classification errors. These criteria give a probability to each image to contain a label error. The training of the convolutional neural network is based on the label error probability. Images that are suspected to have a label error will have a reduced impact on the training.

Results obtained on a well-know database like MNIST show that this method allows to separate quite efficiently label errors from classification errors. More precisely, it shows that 90% of the label errors could be suppressed at the expense of 20% of the classification errors. However, the usage of a more complex database, like CIFAR, displays the importance of an efficient network in order to have the maximum gain of the purification method.

This last result indicates that a deeper analysis of the purification methods in different cases is necessary. Indeed, a deeper test shows the negative impact of overfitting in the presence of label errors. This might have a significant impact on the number of epoch used in a neural network. The more epoch, the more the network overfits. Small networks seem to be less affected by this overfitting but their results are unfortunately not as good as the ones obtained with complex networks. This is due to the simplicity of the network. Solutions like removing all errors (label errors and classification errors) during training by using 0 weights allow a certain stability. Nevertheless, it cannot be considered as optimal since all misclassified images are also removed. More creative

solutions with iterative weights give promising results.

A real test on a more complex database finalized this paper. This test confirms the previously observed results but the overall performance and efficiency is not as good as for simpler databases like MNIST. The results seem to suffer from the not random noise between similar characters. However, the results still confirm the interest of the method and suggest to deepen the research.

Future improvements could for instance focus on a better weight computation. Furthermore, the methods presented in this paper only focus on images that are not correctly classified. The methods did not explore the modification of correctly classified images.

BIBLIOGRAPHY

[1] Michael Nielsen. Deep learning. In *Neural Networks and Deep Learning*, chapter 6. Determination Press, 2015.

[2] Junho Yim, Jeongwoo Ju, Heechul Jung, and Junmo Kim. Image classification using convolutional neural networks with multi-stage feature. In *Robot Intelligence Technology and Applications 3*, pages 587–594. Springer, 2015.

[3] Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1717–1724, 2014.

[4] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[5] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[6] Frank Rosenblatt. *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957.

[7] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[8] Syafeeza Ahmad Radzi and Mohamed Khalil-Hani. Character recognition of license plate number using convolutional neural network. In *International Visual Informatics Conference*, pages 45–55. Springer, 2011.

[9] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[10] Dan C Ciresan, Ueli Meier, Jonathan Masci, Luca Maria Gambardella, and Jürgen Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, page 1237. Barcelona, Spain, 2011.

[11] La technologie au service des villes. `http://www.macq.eu/la-technologie-au-service-des-villes/?lang=fr`. Accessed: 2017-10-08.

[12] Joseph L Schafer and John W Graham. Missing data: our view of the state of the art. *Psychological methods*, 7(2):147, 2002.

[13] Benoît Frénay and Michel Verleysen. Classification in the presence of label noise: a survey. *IEEE transactions on neural networks and learning systems*, 25(5):845–869, 2014.

[14] Jiang-wen Sun, Feng-ying Zhao, Chong-jun Wang, and Shi-fu Chen. Identifying and correcting mislabeled training instances. In *Proceedings of the Future Generation Communication and Networking - Volume 01*, FGCN '07, pages 244–250, Washington, DC, USA, 2007. IEEE Computer Society.

[15] Jaree Thongkam, Guandong Xu, Yanchun Zhang, and Fuchun Huang. Support vector machine for outlier detection in breast cancer survivability prediction. *Advanced Web and NetworkTechnologies, and Applications*, pages 99–109, 2008.

[16] Isabelle Guyon, Nada Matic, and Vladimir Vapnik. Advances in knowledge discovery and data mining. chapter Discovering Informative Patterns and Data Cleaning, pages 181–203. American Association for Artificial Intelligence, Menlo Park, CA, USA, 1996.

[17] Umaa Rebbapragada and Carla E Brodley. Class noise mitigation through instance weighting. In *European Conference on Machine Learning*, pages 708–715. Springer, 2007.

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve **www.uclouvain.be/epl**