Fault Tolerant Deep Neural Networks for Detection of Unrecognizable Situations

Kaoutar Rhazali, Benjamin Lussier, Walter Schön, Stéphane Géronimi

Fault Tolerant Deep Neural Networks for Detection of Unrecognizable Situations

Kaoutar Rhazali [*] **'** [**] **Benjamin Lussier** [*] **Walter Schön** [*]
Stéphane Geronimi [**]

[*] *Sorbonne Universités, Université de Technologie de Compiègne, CNRS, UMR 7253, Heudiasyc CS 60 319, 60203 Compiègne, France (e-mail: kaoutar.rhazali@hds.utc.fr, benjamin.lussier@hds.utc.fr, walter.schon@hds.utc.fr).*
[**] *Groupe PSA, Direction de la recherche et de l'innovation automobile, Route de Gisy 78943 Vélizy Villacoublay Cedex, France (e-mail: kaoutar.rhazali@mpsa.com, stephane.geronimi@mpsa.com).*

Abstract: Deep Neural Networks are achieving great success in various fields. However, their use remains limited to non critical applications because their behavior is unpredictable and unsafe. In this paper we propose some fault tolerant approaches based on diversifying learning in order to improve DNNs dependability and particularly safety. Our main goal is to increase trust in the outcome of deep learning mechanisms by recognizing the unlearned inputs and preventing misclassification.

Keywords: Safety, Fault Tolerance, Artificial Intelligence, Neural Networks, Autonomous Vehicles.

1. INTRODUCTION

Deep Learning (DL) is a Machine Learning (ML) technique that has known these past years a growing interest from technology developers and researchers. This technique involves the use of Deep Neural Networks (DNNs) to train algorithms dedicated to speech recognition, computer vision, machine translation, data processing, etc.

Vision, situation recognition and decision making are the most important functionalities that give an autonomous system the ability to interact efficiently with its environment. The complexity of vision increases exponentially when moving from a closed environment to an open environment where the number of situations that could be faced becomes almost infinite. In such cases, it is almost impossible to make a robot perceive its environment by using classical techniques of programming. However, deep learning algorithms provide robust solutions to this complexity by generalizing the outcome of a learned limited set of examples. But these algorithms provide outcomes without any guarantee, since all possible situations can not be tested and these algorithms can be easily fooled (Nguyen et al. (2015)). Thus, the use of DNNs is currently not allowed in critical applications.

In this paper our goal is to increase the confidence in the output of DL mechanisms by detecting the erroneous outcomes in order to avoid misinterpretations, that could lead to severe consequences. At terms, and with combining it with other methods, we hope to allow the use of DNN for autonomous cars vision and situation recognition.

The second section of this paper presents basic concepts on DNN and a state of the art on dependability in this domain. The third section explains our method: using diversified networks to detect ambiguous inputs. Section 4 presents models and architectures used to obtain diversified DNNs. Section 5 presents experimental results on a traffic signs recognition application. Finally, this paper ends with a conclusion and some perspectives.

2. BACKGROUNDS

In this section, we first present Artificial Neural Networks (ANN) and introduce the extended Deep Neural Network (DNN) version. Second we give an overview of related work in the field of DNNs dependability.

2.1 Deep Neural Networks

A standard Artificial Neural Network (ANN) is built by connecting layers composed of several nodes called neurons. An artificial neuron can be mathematically modeled by a transfer function that computes the image of the weighted sum of its inputs:

$$\varphi(x_1, x_2, ..., x_n) = \varphi(\sum_{i=1}^{n} w_i x_i + b) \quad (1)$$

where x_i represent the neuron inputs, w_i the weights, b the bias, and φ the transfer function. This function is called the activation function and it can take multiple forms (sigmoid, TanH, ReLu, softmax, ...). The value computed by the neuron can be real or binary.

ANNs are made of several layers : the input layer, the hidden layers and the output layer. The hidden layers are called fully connected since each of their neurons is connected to all neurons of the previous and the next

layers. Due to this complexity, ANNs usually have very few hidden layers.

The goal of an ANN is to learn a function $\widehat{f}$ approximating a function f, that maps the input X to the output $Y = f(X)$, from a set of labeled images $(X, f(X))$.

Designing a neural network involves setting a number of parameters, namely the structure, the cost function, the activation functions, the optimization algorithm, and the initial conditions (weights, biases, learning rate, ...). Most of these parameters are determined empirically which complicates the task of developing a dependable network. On the other hand, training a neural network means adjusting its weights and biases that are often initialized randomly by learning from a finite dataset. Hence, all the network parameters plus the training dataset (nature of samples, the order of training, the batch size, ...) affect significantly the learned function.

DNNs are an extended version of standard ANNs with sophisticated structures (convolutional neural networks, recurrent neural networks, ...), and with more hidden layers. The depth of a DNN (number of hidden layers) helps come up with complex problems, since more layers will allow a more complex function.

Apart of the MLP model that was described previously, there exist many other variants and combinations of DNNs models. The most known architectures are: Convolutional Neural Networks (CNNs), Deep Belief Networks (DBNs), Autoencoders (AE), and Recurrent Neural Networks (RNNs). AE and DBNs are commonly used in unsupervised learning mode. CNNs are widely used in several applications including image recognition, speech recognition, data processing, etc. RNNs are particularly designed to deal with sequenced data analysis. For more details about DNNs, we refer the interested readers to (Schmidhuber (2015) and Goodfellow et al. (2016)). In this work, we will especially focus on CNNs since they have recently attained a state-of-the-art performance on the task of image recognition.

Constructing a DNN is a complex task, since the architectures are mainly designed empirically. A network designer should make various design choices, including the number of hidden layers, the size of each layer, the learning rate, the cost function, the activation function, the initial weights and biases, the nature of layers (pooling, dropout, sub-sampling,...), etc. At the present, there is no strong defined mathematical model that helps choosing the right topology and determining the appropriate hyperparameters to develop an optimal and efficient network. Thus, designing a network is often criticized as being a black art.

The major CNNs models, that have achieved outstanding results at ILSVRC challenges, are now viewed as state-of-the-art CNNs architectures. The most common are: AlexNet, VGGNet, GoogleNet and ResNet. In our experiments, we use particularly these four architectures to evaluate our approaches.

2.2 Dependability in Deep Neural Networks

Despite their great success in various fields, DNNs lack interpretability and are mostly considered as a black boxes. For these reasons, safety standards have forbidden their use in safety critical systems, and only allow them in applications requiring the lowest Safety Integrity Level (SIL 1) (Kurd et al. (2003)).

There are few works in the literature that have studied the safety aspect of neural networks, whereas the security and robustness aspects recently received more attention. DNNs weakness to adversarial examples have particularly been highlighted: the alteration of inputs in order to force a learned DNN to misclassify them. These alterations can be due to human malicious attacks or external faults due to the environment.

Several methods to generate adversarial examples have been proposed. Kurakin et al. (2016) explore the possibility of creating adversarial modifications for machine learning systems which operate in the physical world. Papernot et al. (2017) show that it is possible to attack classifiers without knowledge of their training data or model, and introduce an attacker able to produce adversarial examples for black-box classifiers. Evtimov et al. (2017) introduce an attack algorithm called Robust Physical Perturbation that produces perturbations physically realizable, and robust to changing physical conditions. Moosavi-Dezfooli et al. (2016) showed the existence of universal image-agnostic perturbations which, when added to all data points, fool state-of-the-art DNNs. Moreover, the paper proposed an algorithm for finding such perturbations for a set of training examples. Goodfellow et al. (2014) introduce a family of fast methods for generating adversarial examples, and show that RBF (Radial Basis Function) networks are resistant to distortions. Instead of generating adversarial samples which are inputs with small modifications that are often imperceptible to the human eye, some authors (Goodfellow et al. (2014) and Nguyen et al. (2015)) have been interested in generating inputs that a human would classify as not belonging to any of the categories class, whereas the network assigns them to some class with high confidence. Those samples are called *rubbish class*. Nguyen et al. (2015) addressed the issue under the name of fooling images. It introduces a new algorithm based on evolutionary algorithms used for generating fooling images that are completely unrecognizable to humans, but that DNNs believe to be recognizable objects with high confidence. The authors showed that re-training the network by adding a new class labeled *fooling images* didn't prevent producing new fooling images.

On the other hand, various approaches exist to increase model's robustness against adversarial attacks. Zheng et al. (2016) presents a general stability training method that makes DNNs more robust for near-duplicate detection, similar image ranking and classification on noisy datasets. Metzen et al. (2017) uses a detector subnetwork trained on the binary classification task of distinguishing genuine data from data containing adversarial perturbations. The proposed approach does not necessarily allow classifying adversarial examples correctly, but it allows mitigating adversarial attacks. Uličnỳ et al. (2016) proposes a robustness method called Adversarial Committee

and shows its capacity to resist to adversarial noise for the MNIST dataset (LeCun et al. (1998)), by comparing it with other methods. The method consists of combining a set of networks in such a way that each network is trained on its own adversarial noise. Tang and Eliasmith (2010) present two strategies for improving the robustness of DBNs: by using sparse connections in the first layer of the network (sparse DBN), and by implementing a probabilistic denoising algorithm that makes networks robust to common variations such as occlusion and random noise.

In comparison, works published in the field of DNNs safety are relatively rare. Huang et al. (2017) have introduced a verification algorithm for checking safety of DNNs by exploring a region around a data point. The goal is to determine the extent to which the input can be altered without changing the classification result by propagating constraints through the layers of the networks. This method is however difficult to use in practice, as a tremendous amount of calculations are needed for each data point. Hosseini et al. (2017) propose training the classifier to reject the adversarial examples by assigning them a NULL label, while classifying the clean data as their original labels. In the same context, Shalev-Shwartz et al. (2016) tackled the safety problem in reinforcement learning, including the application of forming long term driving strategies. The authors outlined the challenge of ensuring functional safety of the driving policy that faces the autonomous driving. They decompose the driving policy into a policy for desires using machine learning, and a trajectory planning with hard constraints using classical imperative methods. The safety task was only addressed by using hard constraints in the trajectory planning phase that requires simply resolving an optimization problem and doesn't involve any learning algorithm. Kurd and Kelly (2003a) define the safety criteria that would contribute to justify the safety of neural networks. The criteria are specified as a set of safety requirements for the behavior of ANNs that should be respected during the development phase. However, the paper doesn't explain how to implement these requirements and remains theoretical. The same authors present in (Kurd et al. (2003) and Kurd and Kelly (2003b)), a development cycle for safe ANNs, which focuses on studying the safety of learned features by translating them into a symbolic level. In practice, such a method is very difficult to achieve, as it is complex and costly to understand the learned parameters of a ANN. Software development process models for ANNs used in critical applications have also been introduced in (Rodvold (1999) and Nabney et al. (1997)). The first paper split the development process into 5 steps and implement feedback loops between some steps to prevent becoming mired in an unproductive path, by moving back when a step is deficient. The second paper, presents a development lifecycle model composed of 6 steps, and studies two practical methods of assessing neural networks. The steps specified in these two references are generic and can be considered as sets of good practices that can be used complementarily to fault tolerance techniques.

2.3 Diversification in Neural Networks

The generation of diversified networks has been featured in a number of previous studies. Partridge and Yates (1996) proposes a methodology for constructing reliable neural-nets divided into three steps. First, producing diverse neural nets by changing: the net type (two architecture types have been used: MLP and Restricted Boltzman Machines (RBM)), the net structure, the training set, the number of hidden layers, and the initial weights. Second, picking up the optimal versions by using three types of optimization approaches like genetic algorithm. Third, gathering the optimal networks to constitute a multiversion system with a decision strategy such as majority vote. Another work (Lincoln and Skrzypek (1989)) suggests combining the outputs of a cluster of multiple nets into a single output. To do so, a judge determines the cluster output by computing the simple average or the weighted sum of the multiple outputs. Then, after evaluating the common error, this latter is back-propagated through all networks. The authors argue that this approach adds fault tolerance by giving the judge the ability to bias the outputs based on the past reliability of the nets. The concept of generating N-version programming with the aim of increasing the fault tolerance and improving the reliability of neural networks has been addressed in (Sharkey et al. (1995)). The authors studied the correlation between the generated networks, considering three methods of creating diversity: varying the initial conditions, varying the training sets, and using contrasting measures (alternative inputs). This last method was evaluated as the best mean of producing diverse networks, whereas the versions generated only by changing the initial conditions had highly correlated failures. However, all of these works intend to improve a network availability. To our knowledge, none of them aimed to identify unspecified inputs, that are inputs far away from each learned class.

3. FORCED DIVERSIFIED LEARNING

In this work, we seek to make up the large lack of theoretical explanations and the inability of humans to understand DNNs, by implementing fault tolerant techniques to increase their dependability, and more particularly their safety.

Most DNNs researchers are mainly interested in developing networks able to achieve high accuracy by labeling correctly and robustly their inputs. In this article our main concern is to give the network the capacity to identify and discard erroneous or unspecified inputs, without trying to classify those unspecified inputs correctly: since the system has not been specified and developed to consider them, any of its response to them can not be intended. Thus, we consider that detecting such unspecified inputs and alerting a diagnosis mechanism or ultimately the operator is the safest response for an unpredictable component such as a DNN. Indeed, in applications such as autonomous driving, recognizing traffic signs is generally assigned to DL mechanisms, and mislabeling an unlearned traffic sign or simply a signboard that has some similarities with a traffic sign (see fig. 1), could eventually cause an accident. In addition, our proposed technique can contribute to a classifier's security towards malicious adversarial attacks, since it could detect the perturbed inputs and prevent misclassification.

Our technique is based on the concept of diversified redundancy. We consider that when classifiers are developed and

Fig. 1. Examples of a perturbed input (left image) ((Evtimov et al., 2017)) and an unknown input (right image)

trained independently, they will learn a different network classification function, and consequently they may act differently in front of an unlearned input. In fact, during the training process, the network progressively learns from its training samples to set the boundaries of its classes in a multidimensional space by optimizing a cost function. What allows the robustness of neural networks is that a classifier doesn't learn to only fit the learned examples into each class, but it generally defines the boundaries that separate the set of classes. Therefore, a big part of the empty multidimensional space is distributed to the learned classes, which allows DNNs to be easily fooled in front of an unlearned input. By using forced diversified DNNs, we hope that the different DNNs will not have the same boundaries between classes, and thus that unspecified inputs will be classified differently by some networks. Since we can not precisely know the boundaries of each network without extremely complex and costly analyses, we can not guarantee that this technique alone is sufficient to detect all or even most of unspecified inputs. We are nonetheless confident that it is an efficient first step, to be used with other complementary methods, in improving significantly the safety of DNNs.

Forced diversification can be done mainly following three approaches: diversification of the training parameters (training sets, optimization function, etc.), diversification of the network architecture and structure (number of and connections between layers, value of the initial parameters, etc.), diversification of the precise function of the network (classifying between each output classes, introducing a new class "unknown" as an additional output class, tree classification, etc.). The next session will present examples for each of these approaches.

4. SAFER DEEP NEURAL NETWORKS

We have developed different approaches for neural networks diversification. We have classified these approaches in three ways: diversification of the training dataset, diversification of the network parameters and structure and finally classification diversification. Note that, although we present these different approaches separately, our technique aims at combining more than one approach in order to build fault tolerant architectures.

4.1 Approach 1: Training set diversification

As a first approach, we propose generating diversified DNNs by acting directly on the training dataset. According to some experiments we carried out, it appeared that just changing the order of training (the order in which the training samples are used to train the network), or training with different datasets can produce significantly diversified networks.

Data augmentation is a potential technique that seems to be useful and which is often used by DNNs developers in order to improve the robustness. It consists in slightly altering the images of each batch before injecting them into the network. Once chosen for training, the images undergo modifications according to some degrees selected randomly from predefined intervals. The alterations can be made by varying several parameters like brightness, contrast, rotation, cropping, sharpness, etc. Similarly, we can promote diversification by using the techniques of boosting and bagging (Bauer and Kohavi (1999) and Schapire (1999)), which both assign weights to the training examples. The main difference between these two techniques resides in how the samples are weighted. The boosting algorithm relies on the performance of its classifiers to generate the probability distribution over the dataset while the bagging algorithm does not take into account this performance.

We also propose training networks with RGB and Grayscale images separately in order to foster diversification. This technique seems to be promising, since it enables networks to learn distinct features. Note that this method is not recommended in all applications. In fact, training with RGB images is inevitable in some classification problems, where the color is a crucial element to make decision.

4.2 Approach 2: network parameters diversification

Obviously, the function learned by a network is highly affected by the network parameters. In those parameters, we count the initial conditions of the network (initial values of weights and biases), its structure (the number of hidden layers, the topology between layers, their sizes, etc.), its activation functions, the cost function, the optimization algorithm, etc. Note that some of the aforementioned structure, activation functions and optimization algorithm are generally defined in the literature's architectures (Alexnet, VGGNet, GoogleNet, etc.). Varying some or all of these parameters will undoubtedly contribute to diversifications in the networks. These techniques can be efficient, however nothing prevents two networks with different architectures and settings from developing the same or very similar functions.

4.3 Approach 3: Classification diversification

In this section, we propose to directly diversify the function of the networks: that is what to classify and how.

Classification 1 - Basic classification: The common model of networks is based on an end-to-end classification, by performing a direct input-output mapping. These networks are created to take an image as input and gives a class label as output according to a predefined set of classes.

Classification 2 - Classification with additional class "unknown": We propose adding a new class label as NoId (Not Identified) to the network, in order to recognize unlearned situations. To do so, a new data set must be added to the original training data, in such a way that the new samples must be diverse and different from the categories

to be learned. Also, it shall be ensured that the training data is balanced, that is to say the amount of images for each class should be somehow equal, in order to represent the classes equally during the training process. The idea is to narrow the boundaries of the main classes by allocating a portion of the empty multidimensional space to the new NoId class. Note that this technique does not guarantee that the trained network will place all the aberrant images in the NoId class, although it should improve the system fault tolerance.

The NoId class is similar to the rubbish class used in Goodfellow et al. (2014) and Nguyen et al. (2015). The latter paper showed that retraining the network with the negative images did not prevent their algorithm from producing new fooling images in the case of the MNIST dataset. However, the same algorithm was less able to generate new successful malicious images with the ImageNet dataset.

Classification 3 - Class reference classification: For the third model, we want to develop a network able to compare the input image with a class reference and decide if they are similar or not. A reference image must thus be selected from each class dataset, in order to be used as a class reference. At the learning phase, each training sample is concatenated with a class reference, selected randomly, to form one global input image, which is subsequently injected in the network. The considered network is designed with two outputs, labeled Similar and Different. In the case where the input image is representing the same category as the reference class, the correct output is to classify them as Similar and otherwise as Different.

This model can be exploited in two ways. First, by comparing each input image with all the class references successively: then if the model returns exactly one positive output, the input will be classified as the class reference which provided the positive result, otherwise the input will be classified as unknown. Second, the model can be alternatively used to confirm the decision of another network. The mechanism is as follows: once the network to be checked assigns the input to a given class, the model will compare that input with the reference of the detected class and validate the classification if it returns a positive result.

Classification 4 - Tree classification: In this classification, instead of using an end-to-end network to perform classification, we propose to split the recognition task into several steps. To do so, the classes must be arranged hierarchically by gathering the categories that have common visual elements (color, shape, pattern, ...) into coarse classes. The objective is to build a hierarchical label tree, in such a way that the macro-classes and their sub-classes will constitute the tree nodes, and the refined classes will represent the tree leafs. At each node, a network must be trained to classify the categories linked to that node. Therefore, the number of networks to be trained will be equal to the number of the tree nodes. Then, the classifiers might be organized in a hierarchical manner, in order to perform a top-down classification.

This approach is an alternative method of classification which must be combined with other diversified classifications in order to detect the invalid inputs. In addition, it could reduce the complexity of classification, especially when dealing with problems with large number of classes.

Fig. 2. The 10 traffic signs used for learning and their labels

Other diversifications: In the same context as the classification 3, we propose developing a network able to compare an input image with a class reference, in such a way that the class reference will represent only the object to be recognized with a white background. This method will force the classifier to give more attention to the object features, which could increase the classification performance.

Another approach is to develop a set of binary classifiers, with only two outputs, in the sense that each classifier will be specialized at recognizing a particular class. The objective is to build a number of networks equal to the number of classes considered, and to make each network learn a specific class, in order to be able to decide if the input image belongs to that class or not. Such training method could help optimizing the space assigned to each class. As the third model, this approach can be exploited in two ways: alone by injecting the image into as many reference classifiers as classes, or to validate another classifier.

Another alternative consists in dividing the number of classes into three-thirds, and to train three classifiers independently, each one with two different thirds of classes. The goal is to decide that a given input belongs to a specific class only if exactly two networks place it in that class. This strategy is intended to diversify the function of each classifier, by changing the set of classes that they will each learn.

Other classification diversifications are of course possible. In the rest of this paper we will focus on classifications 1 and 3.

5. EXPERIMENTAL RESULTS

To evaluate the performance of the proposed diversification approaches, we present in this section the results of a real-world application related to the problem of traffic signs recognition. We trained our classifiers on the 10 traffic signs illustrated in figure 2, extracted from the GTSRB (German Traffic Sign Recognition Benchmark) dataset (Stallkamp et al. (2011)). The images in this dataset vary in size and are RGB-encoded. In our application, we only used images larger than 30*30 pixels, thus we have only considered 10 classes, since the other traffic signs have only few images larger than 30*30 pixels. We resized all the images to 60*60 pixels. We used a test set of 1185 samples. For evaluation, we used various sets of images collected from several websites. For architectures, we used state-of-the-art convolutional networks, namely extents of AlexNet, GoogleNet, VGGNet, and ResNet. As previously explained each one of these architectures is defined by a depth, a succession of specific layers, predefined filters sizes, and a special interconnection between layers.

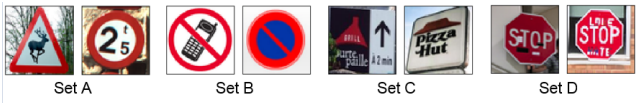

Fig. 3. Images extracted from the evaluation sets

5.1 Evaluation sets

To analyze the diversification efficiency of some of our proposed approaches, we used four different sets of images:

- *Set A*: consists of 100 traffic signs photos different from the learned classes.
- *Set B*: consists of 70 traffic signs drawings different from the learned classes.
- *Set C*: consists of 13 photos of signboards that we can find at roadsides but that are not traffic signs (such as brand signs).
- *Set D*: consists of 9 physical traffic signs with adversarial perturbations. Those images were used in (Evtimov et al. (2017)) as attack images to fool a classifier.

Our purpose is to label the maximum of the images in sets A, B, and C as unknown, and to avoid misclassifying the images in D and the test set by labeling them correctly or by classifying them at least as unknown. Indeed, for safety reasons, we prefer to classify a given class as unknown instead of labeling it incorrectly. Examples from each dataset are illustrated in figure 3.

5.2 Diversification analysis

In these experiments, we study the diversification efficiency of some of the proposed approaches. In the first two experiments we study a diversified block using two networks in parallel, while in the third experiment we study a diversified block using two sequential networks. Note that by deciding on everything that is random in a network, we always generate similar networks that provide the same outcome for any input, since network learning is a deterministic process. In order not to be captured in a local minimum, we generate several blocks in our experiments and present their average results.

Sampling order. In this experiment, we studied the impact of changing the sampling order on diversity. The sampling order is the order in which the training samples are used to train the network. To achieve this goal, we trained six times an extent version of AlexNet architecture by varying the training order and by fixing all the other network settings including the initial conditions. We performed a pairwise comparison over the six generated networks (resulting into 15 diversified blocks) and we identified inputs labeled differently as unknown. Table 1 reports the average results of the 15 blocks considering the test set and the 4 evaluation sets.

We can notice that just varying the sampling order allowed us to detect some unknown inputs in the sets A, B and C. In addition, an average of 7.07/9 of adversarial examples have been correctly classified and 1.13/9 have been classified as unknown while only 0.8/9 of samples have been misclassified in the set D. Moreover, the average misclassification rate in the test set reduced to 0.08% by classifying 0.36% samples as unknown.

Network structure. We trained extent versions of the four CNNs architectures: AlexNet, VGG-16, GoogleNet, and ResNet. Due to the lack of space, we only report the results of comparing just two architectures (GoogleNet and ResNet) in table 2. The results are provided by calculating the average of six blocks. Each block is composed of two networks (GoogleNet and ResNet) connected in parallel and trained with the same sampling order.

Changing the network structure has significantly improved diversification, to such an extent that we detected more than 50% unknown examples in the sets A, B and C. Furthermore, the number of samples misclassified in the set D was reduced to 0.17/9. However, accuracy was slightly degraded to 97.22% in the test set while the misclassification rate was successfully to 0.01%.

RGB-Grayscale training. We used two extent versions of VGG-16 architecture: one version trained with RGB images and another with Grayscale images. Both versions have similar structure except for the input layer. Table 3 reports the average of 6 blocks, each one providing the results of two networks connected in parallel: one network trained with RGB images and another with Grayscale images. The networks in each block are trained with the same sampling order.

We notice that the number of samples misclassified in the sets B and C has decreased to 18.17/70 and 2.67/13 respectively. However, this technique was less efficient in labeling the perturbed images of the set D.

Classification diversification. In this experiment, we tested the class reference classification method (approach 3) to detect the unknown inputs. We consider here a block of two networks in series: a first network used to perform

Table 1. The average classification results of 15 diversified blocks resulting from a pairing of 6 networks generated by varying the sampling order.

Sets	test set	D
Correctly classified	99.56 %	7.07/9
Classified as unknown	0.36 %	1.13/9
Misclassified	0.08 %	0.8/9

Sets	A	B	C
Correctly classified as unknown	14.06/100	13.53/70	3.13/13
Misclassified	85.93/100	56.46/70	9.86/13

Table 2. The average classification results over six blocks. Each block is composed of two networks with different structures.

Sets	test set	D
Correctly classified	97.22 %	6.83/9
Classified as unknown	2.77 %	2/9
Misclassified	0.01 %	0.17/9

Sets	A	B	C
Correctly classified as unknown	54/100	41.83/70	7.83/13
Misclassified	46/100	28.17/70	5.17/13

Table 3. The average classification results over six blocks. Each block is composed of two networks: one trained with RGB images and another with Grayscale images.

Sets	test set	D
Correctly classified	99 %	4.33/9
Classified as unknown	0.86 %	3.67/9
Misclassified	0.14 %	1/9

Sets	A	B	C
Correctly classified as unknown	51.83/100	51.33/70	10.33/13
Misclassified	48.17/100	18.17/70	2.67/13

basic classification and a second one used for validation. We trained six basic classification networks and six class reference networks. In table 4 we present the average classification results of 36 blocks obtained by testing all possible combinations.

Table 4. The average classification results over 36 blocks. Each block is a combination of two networks in series: A basic classification network and a class reference network.

Sets	test set	D
Correctly classified	98.90 %	7.06/9
Classified as unknown	1.02 %	1.94/9
Misclassified	0.08 %	0

Sets	A	B	C
Correctly classified as unknown	34.14/100	34.42/70	7.36/13
Misclassified	65.86/100	35.58/70	5.64/13

Using class reference classification was less effective in detecting unknown inputs in sets A, B and C, however it was more robust in classifying correctly the perturbed inputs.

Results analysis. All diversification techniques investigated previously have proved their capacity for detecting unknown inputs even if some of them were less effective than others. It is difficult to determine the better one because the results differ according to the evaluation sets. For instance, diversifying the network structure achieved high performance in the set A, while training with RGB and Grayscale images allowed detecting the highest number of unknown images in sets B and C. Moreover, the results are probably very different depending on the application.

5.3 Fault tolerant architecture

In this section, we combine some of our proposed approaches to validate whether they are complementary. We propose a potential fault tolerant architecture in figure 4 that takes an input image and provides a safe output. This architecture involves four diversified networks (N1, N2, N3 and N4):

- The classifiers N1 and N2 are respectively extents of GoogleNet and ResNet designed to perform a basic classification and are trained with RGB images.
- The classifier N3 is an extent of VGG-16 designed to perform basic classification and trained with Grayscale images.
- The classifier N4 is also an extent of VGG-16 designed to perform class reference classification and trained with RGB images.

As illustrated in figure 4, we carry out two tests before labeling the input image. Firstly, we feed the input into the three networks (N1, N2 and N3) and we compare their outcomes. If at least one outcome is different, the image is classified as unknown, otherwise we perform a second test to validate the label "i" assigned to the input. To this end, we concatenate the input image with the class reference "i" and we feed it into N4. If this latter returns a negative outcome, that is the input image is different from the class reference, the image will be as well classified as unknown, otherwise it will take the label "i". Table 5 reports the classification results of this architecture.

Th results presented in figure 5 demonstrate that combining various approaches was more effective in detecting

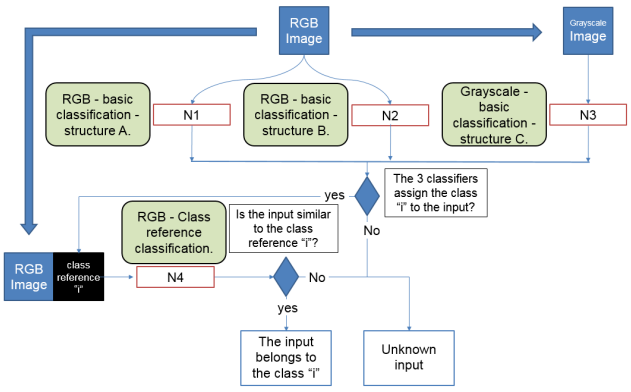

Fig. 4. A Detection mechanism for unknown images by combining the 3 approaches

the unknown inputs, at a minimal cost to accuracy. In addition, we can notice that our technique has identified the adversarial images that we failed to label correctly by classifying them as unknown.

6. CONCLUSION

Despite the great success of DL in many areas, its use remains limited to non critical applications. This is due to the lack of methodologies and tools that can justify and validate their dependability. In this context, we have developed a technique based on the concept of diversified redundancy in order to reinforce the DNNs safety. The idea is to develop diverse networks trained independently and combine them in one fault tolerant architecture in order to detect the unlearned inputs and classify them as unknown. This way, misinterpretations that could lead to severe consequences could be avoided. To achieve our goal, we proposed three approaches based on diversifying: (1)the training dataset, (2)the network parameters, and (3)classification. The first two approaches enable learning different approximations of the same function, while the third approach aims at diversifying the learned function itself by using alternative strategies of performing classification. In fact, classifiers created by modifying the training dataset or the network parameters don't learn the same way, and hence they are more likely to behave differently in front of unlearned inputs. In addition, diversifying classification allows focusing on different features which significantly influence the decision making.

The experiments conducted on an application dedicated to traffic signs recognition showed the efficiency of our approaches, particularly when combined.

Our study thus demonstrated that invalid inputs can be effectively identified by an approach of diversifying learning. In the future, we plan to test our technique on other critical applications and to develop more effective

Table 5. The classification results obtained by using the proposed fault tolerant architecture.

Sets	test set	D
Correctly classified	97.55 %	6/9
Classified as unknown	2.45 %	3/9
Misclassified	0 %	0

Sets	A	B	C
Correctly classified as unknown	81/100	69/70	13/13
Misclassified	19/100	1/70	0

and practicable approaches that could be also used in detecting adversarial inputs.

ACKNOWLEDGEMENTS

This work took place in the framework of a CIFRE thesis with PSA Group and the Heudiasyc Laboratory (Sorbonnes Universités UTC, UMR CNRS 7253).

REFERENCES

Bauer, E. and Kohavi, R. (1999). An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine learning*, 36(1), 105–139.

Evtimov, I., Eykholt, K., Fernandes, E., Kohno, T., Li, B., Prakash, A., Rahmati, A., and Song, D. (2017). Robust physical-world attacks on machine learning models. *arXiv preprint arXiv:1707.08945*.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

Goodfellow, I.J., Shlens, J., and Szegedy, C. (2014). Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, 1026–1034.

Hosseini, H., Chen, Y., Kannan, S., Zhang, B., and Poovendran, R. (2017). Blocking transferability of adversarial examples in black-box learning systems. *arXiv preprint arXiv:1703.04318*.

Huang, X., Kwiatkowska, M., Wang, S., and Wu, M. (2017). Safety verification of deep neural networks. In *International Conference on Computer Aided Verification*, 3–29. Springer.

Krizhevsky, A., Sutskever, I., and Hinton, G.E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 1097–1105.

Kurakin, A., Goodfellow, I., and Bengio, S. (2016). Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*.

Kurd, Z. and Kelly, T. (2003a). Establishing safety criteria for artificial neural networks. In *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*, 163–169. Springer.

Kurd, Z. and Kelly, T. (2003b). Safety lifecycle for developing safety critical artificial neural networks. In *International Conference on Computer Safety, Reliability, and Security*, 77–91. Springer.

Kurd, Z., Kelly, T., and Austin, J. (2003). Safety criteria and safety lifecycle for artificial neural networks. In *Proc. of Eunite*, volume 2003.

LeCun, Y., Cortes, C., and Burges, C.J. (1998). The mnist database of handwritten digits.

Lincoln, W.P. and Skrzypek, J. (1989). Synergy of clustering multiple back propagation networks. In *NIPS*, 650–657.

Metzen, J.H., Genewein, T., Fischer, V., and Bischoff, B. (2017). On detecting adversarial perturbations. *arXiv preprint arXiv:1702.04267*.

Moosavi-Dezfooli, S.M., Fawzi, A., Fawzi, O., and Frossard, P. (2016). Universal adversarial perturbations. *arXiv preprint arXiv:1610.08401*.

Nabney, I.T., Paven, M.J., Eldridge, R.C., and Lee, C. (1997). Practical assessment of neural network applications. In *Safe Comp 97*, 357–368. Springer.

Nguyen, A., Yosinski, J., and Clune, J. (2015). Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 427–436.

Papernot, N., McDaniel, P., Goodfellow, I., Jha, S., Celik, Z.B., and Swami, A. (2017). Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, 506–519. ACM.

Partridge, D. and Yates, W.B. (1996). Engineering multiversion neural-net systems. *Neural Computation*, 8(4), 869–893.

Rodvold, D.M. (1999). A software development process model for artificial neural networks in critical applications. In *Neural Networks, 1999. IJCNN'99. International Joint Conference on*, volume 5, 3317–3322. IEEE.

Schapire, R.E. (1999). A brief introduction to boosting. In *Ijcai*, volume 99, 1401–1406.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural networks*, 61, 85–117.

Shalev-Shwartz, S., Shammah, S., and Shashua, A. (2016). Safe, multi-agent, reinforcement learning for autonomous driving. *arXiv preprint arXiv:1610.03295*.

Sharkey, A., Sharkey, N., and Gopinath, O. (1995). Diversity, neural nets and safety critical applications. *Current trends in connectionism*, 165–178.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Stallkamp, J., Schlipsing, M., Salmen, J., and Igel, C. (2011). The German Traffic Sign Recognition Benchmark: A multi-class classification competition. In *IEEE International Joint Conference on Neural Networks*, 1453–1460.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 1–9.

Tang, Y. and Eliasmith, C. (2010). Deep networks for robust visual recognition. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 1055–1062.

Uličnỳ, M., Lundström, J., and Byttner, S. (2016). Robustness of deep convolutional neural networks for image recognition. In *International Symposium on Intelligent Computing Systems*, 16–30. Springer.

Zheng, S., Song, Y., Leung, T., and Goodfellow, I. (2016). Improving the robustness of deep neural networks via stability training. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 4480–4488.