Repair of Convolutional Neural Networks using Convex Optimization: Preliminary Experiments

Dario Guidotti[1] and Francesco Leofante[1,2,3]

[1] University of Genoa, Genoa, Italy
[2] RWTH Aachen University, Aachen, Germany
[3] University of Sassari, Sassari, Italy
{dario.guidotti,francesco.leofante}@edu.unige.it

Abstract. Recent public calls for the development of explainable and verifiable Artificial Intelligence (AI) led to a growing interest in formal verification and repair of machine-learned models. Despite the impressive progress that the learning community has made, models such as deep neural networks remain vulnerable to adversarial attacks, and their sheer size represents a major obstacle to formal analysis and implementation. In this paper, we present our current efforts to tackle repair of deep convolutional neural networks using ideas borrowed from Transfer Learning. Using results obtained on popular MNIST and CIFAR10 datasets, we show that models of deep convolutional neural networks can be transformed into simpler ones preserving their accuracy, and we discuss how formal repair through convex programming techniques could benefit from this process.

Keywords: Transfer Learning · Network Repair · Convex Optimization.

1 Introduction

The need for the development of explainable and verifiable AI has been put forward in a number of public events, e.g., the Workshop on Explainable AI held at IJCAI 2017[4], and research programs, e.g., the DARPA program on Explainable Artificial Intelligence.[5] These "calls to arms" did not go unanswered, originating several related research streams. Among them, particularly vibrant is the one concerned with automated verification and repair of Deep Neural Networks (DNNs).

Despite the impressive progress that the learning community has made in the field, it is well known — see, e.g., [16, 3] — that DNNs can be vulnerable to *adversarial perturbations*, i.e., minimal changes to correctly classified input data that cause a network to respond in unexpected and incorrect ways. Independently from the accuracy of a network, the vulnerability to adversarial attacks calls for techniques to improve robustness and guarantee desired properties. Repair [12, 4, 5] is one such technique, whereby we seek to adjust the parameters

[4] http://home.earthlink.net/ dwaha/research/meetings/ijcai17-xai/
[5] https://www.darpa.mil/program/explainable-artificial-intelligence

of the network in order to formally guarantee that the network will respond correctly even in the presence of adversarial perturbations. In practice, the sheer size of these models still represents a major obstacle to formal analysis of any kind. Typical state of the art neural networks for tasks like image classification have more than a hundred millions of parameters [15], which makes off-the-shelf techniques hardly applicable.

In this paper we focus on the repair of Convolutional Neural Networks (CNNs), a type of DNN mainly used in computer vision — see [8] for a survey related to DNN architectures and their applications. In particular, we discuss our current efforts to repair CNNs using convex programming and ideas borrowed from Transfer Learning (TL) [18]. As posited in [14], the idea is to keep the convolutional part of the network as a learned feature extractor, and replace the final classification layer with one featuring less parameters and/or a smaller model complexity. Noticeably, the replacement may yield networks whose accuracy is comparable with the one of the original DNNs, yet more amenable to formal analysis.

More specifically, we contribute an experimental analysis based on the popular MNIST[6] and CIFAR10[7] datasets. The first step is to train CNNs on both datasets and to replace their final fully-connected layer with linear support vector machines. This step reduces by orders of magnitude the number of free parameters, e.g., from 4.7 million to 65 thousand in the case of CIFAR10, while preserving accuracy. We discover adversarial attacks on such "hybrid" models using the Fast Gradient Sign Method described in [3]. Since we replace nonlinear layers with linear ones, we are able to define a repair procedure as a convex optimization problem — as done in [4] for kernel-based learning models. The resulting problem can be solved with off-the-shelf tools — CVXOPT[8] in our case. The results we obtain are still preliminary, but show some promise as far as scaling to networks of larger size is concerned. However, the repair procedure is not yet general enough to make the networks immune to perturbations other than those considered by the repair procedure. While this might not be a strong limitation when considering real-world instead of artificial adversaries — see [2] — further investigations are needed to confirm whether our method could be effective for practical applications.

To the extent of our knowledge, this is the first time that TL is leveraged in order to repair a CNN through convex programming techniques. The idea of replacing parts of a CNN to improve its performances is not new, as it has been explored in [14] and [17], among others. However, the focus of these contributions is to improve the accuracy of the network, rather than providing models whose properties can be certified more easily than the original one. Trying to apply formal verification techniques to networks of size smaller than the original could be done following alternative paths. For instance, in [1] the authors show that it is possible to find small-sized subnetworks in CNNs which prove to be remarkably

[6] http://yann.lecun.com/exdb/mnist/
[7] https://www.cs.toronto.edu/ kriz/cifar.html
[8] https://cvxopt.org/index.html

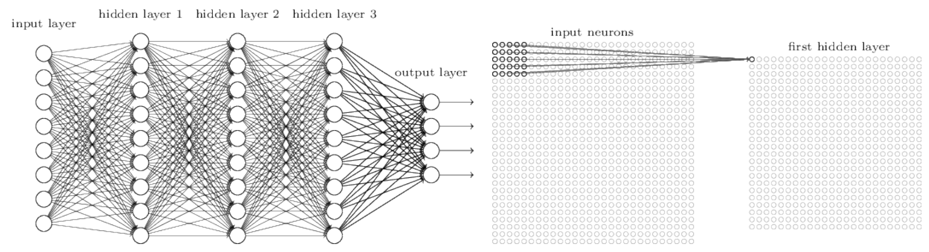

Fig. 1. Generic architecture of a fully connected DNN with 3 hidden layers (left) and graphics representation of a local receptive field (right). The images are taken from [9].

accurate on some datasets including MNIST and CIFAR10. These subnetworks could be extracted and certified considering our approach or other state-of-the-art tools like MARABOU [6]. Finally, the aim of obtaining networks robust to adversarial examples, but not necessarily smaller than the original ones, can be pursued using results in robust training: recent results — see, e.g., [19] — seem to open this possibility also for CNNs of considerable size.

2 Preliminaries

2.1 Convolutional Neural Networks

According to [7], *representation learning* "is a set of methods that allows a machine to be fed with raw data and to automatically discover the representations needed for detection of classification". DNNs are representation learning models characterized by multiple levels of representation, obtained by composing several non-linear modules. Each module transforms the representation at one level (starting with raw input) into the next, more abstract, representation. At the heart of every DNN lie the "classical" neural network modules as shown in Figure 1 (left) whose mathematical formulation can be expressed in a recursive form as:

$$\begin{aligned}\mathbf{h}^{(1)} &= \Phi^{(1)}(\mathbf{W}^{(1)} \cdot \mathbf{x} + \mathbf{b}^{(1)}) \\ \mathbf{h}^{(i)} &= \Phi^{(i)}(\mathbf{W}^{(i)} \cdot \mathbf{h}^{(i-1)} + \mathbf{b}^{(i)})\end{aligned} \tag{1}$$

where $\Phi^{(i)}$ is the activation function, $\mathbf{W}^{(i)} \in \mathbb{R}^{d_i \times d_{i-1}}$ is a matrix of weights and $\mathbf{b}^{(i)} \in \mathbb{R}^{d_i}$ is the vector of the biases of the i-th layer. $\mathbf{h}^{(i)} \in \mathbb{R}^{d_i}$ corresponds to the output of the i-th layer and the range of i depends on the number of layers. A module like this is said to be *fully connected*, because the weighted sum of the outputs of each neuron in level i is fed to every neuron in level $i+1$, creating the topology shown in Figure 1 (left). CNNs are a specifc kind of DNNs, typically adopted in computer vision applications, characterized by one or more *convolutional modules*. The distinctive element of such modules is that they feature connections for small, localized regions of the input vector, i.e., each neuron of the hidden layer is connected only to a small subset of the input neurons. This subset of the input neurons is called *local receptive field* of the hidden neuron. A graphical example of a local receptive is depicted in Figure 1 (right). Another important feature of convolutional modules is that all the local receptive fields share the same weights and bias reducing the overall number of weights substantially. In practice, each local receptive field is trained to detect a specific feature in the input image, i.e., distinctive elements of input portions. As a consequence, in a specific hidden layer, different sets of shared weights are used: each of these sets is trained to detect specific feature in the image. Usually each convolutional module is followed by a *pooling layer*, which simplifies the information received. For instance, each unit of a pooling layer could take a subset of neurons from the previous module and select their maximum activation — an operation called *max-pooling*. Since our experiments are about image classification, in the following we consider a CNN arrangement widely adopted for this task, i.e., a series of convolutional modules and pooling layers followed by fully connected modules. The first part of the network can be seen as an application of a (learned) kernel to the original input whereas the second part can be seen as the actual classifier. For a more detailed study on Convolutional Neural Networks we refer to [9].

2.2 Transfer Learning

As mentioned in [18], TL is "the improvement of learning in a new task through the transfer of knowledge from a related task that has already been learned". TL has been suggested in the context of deep learning applications — see, e.g., [14] — where pre-trained models are used as starting points for computer vision or natural language tasks. Since the training of DNNs requires substantial computational resources, it is often the case that reusing (parts of) pre-trained models enables applications which would not be feasible otherwise. For instance, in [14], a pre-trained convolutional module is extracted from a CNN and then applied as a feature extractor in the context of an object recognition task where the paucity of training samples would make training of the full CNN untenable. On the other hand, combining the pre-trained convolutional module with a newly trained classifier, makes for an effective combination, enabling to solve classification tasks that were not within the reach of the original CNN. TL and its applications suggest the possibility of replacing some modules of a DNN which are hardly analyzable with formal methods, with others that are more amenable to such analysis. As long as the accuracy of the resulting network, which we call *hybrid network* in the following, is close to the original DNN, one may (i) replace the original network with the hybrid one and (ii) fix the hybrid one instead of the original network, should adversarial examples be found also for the hybrid network. In particular, we build hybrid networks by collating the convolutional module of a CNN followed by a linear Support Vector Machine (lSVM), i.e., a classifier based on separating hyper-planes in which the distance of the hyperplane from the nearest samples of both classes is maximized. In our experiments we consider multiclass lSVMs, i.e., in order to discriminate among k classes we compute k different separating hyper-planes each one discriminating among one class and the remaining $k-1$. The input-output relation of a multiclass lSVM is defined as follows:

$$\begin{aligned}\mathbf{f}(\mathbf{x}) &= \mathbf{W}\cdot\mathbf{x}+\mathbf{b}\\ y &= argmax(\mathbf{f}(\mathbf{x}))\end{aligned} \tag{2}$$

where $\mathbf{x} \in \mathbb{R}^d$ is the vector of the inputs, $\mathbf{b} \in \mathbb{R}^k$ is the vector of the biases, $\mathbf{W} \in \mathbb{R}^{k\times d}$ is the matrix of the weights corresponding to k lSVMs, each working to detect one of the k classes. The function $\mathbf{f}(\mathbf{x})$ is the decision function corresponding to the input $\mathbf{x}$. It contains the signed distances of the input $\mathbf{x}$ from each decision hyper-plane. From the definition of the decision function we can derive the correct class y for an input $\mathbf{x}$.

3 Repair of hybrid networks

3.1 Hybrid Networks

For the sake of our experiments, we have developed two CNNs and two corresponding hybrid networks for each dataset considered. Given the preliminary nature of this work the datasets considered are CIFAR10 and MNIST, two of the most famous basic datasets for image classification. The MNIST dataset contains 60000 black-and-white images of handwritten digits whereas the CIFAR10 dataset contains 60000 color images in 10 different mutually exclusive classes: both datasets are divided in a training set of 50000 images and a test set of 10000 images.

The network considered for the MNIST dataset (MNIST-NN) is a CNN with 2 convolutional layers, 2 max-pooling layers and 2 fully connected layers. The convolutional layers have kernel size equal to 5×5 and stride length equal to 1, the max-pooling layers have kernel size equal to 2×2. The two fully connected layers have 500 and 10 hidden neurons respectively and the inputs of the first layer are the value generated from 800 neurons of the second max-pooling layer. The activation functions are all ReLU. The total number of parameters of the network is 407330 and 99.5% of them are part of the fully connected layers.

The network developed for the CIFAR10 dataset (CIFAR10-NN) is a CNN with 6 convolutional layers, 3 max-pooling layer and 3 fully connected layers. The convolutional layers have kernel size equal to 3×3, stride length equal to 1, padding equal to 1 and present respectively 32, 64, 128, 128, 256 and 256 different kernels, the max-pooling layers has kernel size equal to 2×2. There is a max pooling layer every 2 convolutional layer. The three fully connected layers have 1024, 512 and 10 hidden neurons respectively and the inputs of the first layer are the values generated from 4096 neurons of the third max-pooling layer. The activation functions are all ReLU. The total number of parameters is 4747904 and 99.5% of them are part of the fully connected layers. The network considered is similar to the Conv-6 network presented in [1], but our network features a first convolutional layer with 32 kernels whereas the corresponding layer of the Conv-6 network has 64 kernels.

In this work we have used PyTorch [10] for the implementation and training of all the networks. The hybrid networks consist of the union of the convolutional and max-pooling layers of the original networks with lSVM multiclass classifiers: in this work we have used off-of-the-shelf implementations provided by scikit-learn [11]. In particular, both for MNIST-NN and for CIFAR10-NN, we have designed corresponding *linear* and *non-linear* hybrids: the non-linear hybrids use as kernel the standard radial basis function. We identify the linear hybrids as MNIST-LH and CIFAR10-LH and the non-linear hybrids as MNIST-KH and CIFAR10-KH. All networks are trained using standard training parameters recommended respectively from PyTorch and scikit-learn documentation.

As a preliminary experiment we have analyzed the accuracy gap between the hybrid models and the corresponding neural networks: for CIFAR10 models our results are 85.4% (NN), 85.51% (KH) and 85.6% (LH). The accuracies of the MNIST models are 97.12% (NN), 98.86% (KH) and 98.72% (LH). All the accuracies were computed as the number of correctly classified images against all the images of the test sets provided by the MNIST and CIFAR10 repositories. These figures tell us that, for the MNIST dataset, hybrid models can be more accurate than the corresponding CNN, with the kernel-based hybrid being slightly more accurate than the linear one. CIFAR10 is more complex than MNIST, nevertheless the results still hold.

3.2 Repair

The main idea behind our repair approach is to circumvent the repair of CNNs and attempt to repair the corresponding hybrid networks instead. To repair hybrid networks, we generate adversarial examples for them, and then we solve an optimization problem in the space of the network's parameter, with the objective to reduce as much as possible the impact of the adversaries. In order to make the optimization problem computationally feasible, we consider the convolutional modules of our hybrid models as a fixed feature map and we do not include their parameters into the optimization problem, but we concentrate on the final layers instead. In practice, this corresponds to analyzing the network in feature space, instead of input space — as done in [4]. Owing to this, and to the fact that fully connected layers of the CNN are substituted by (l)SVMs in our hybrid networks, the number of free variables for the convex optimization problem is drastically reduced. For example, in the case of the MNIST models, we managed to reduce the number of variables from 405510 to approximately 8000.

In this first stage of our work, we decided to further simplify the problem considered by excluding KH models: in this way it is possible to limit the convex optimization problem to piece-wise linearity — because of absolute values — eliminating the need of a non-linear solver or abstraction techniques to manage the radial basis function kernels. In equation (3) we present the mathematical definition of our optimization problem: parameters c and d are the number of possible classes and the number of features of the adversarial sample in the feature space, respectively; parameters $\gamma_{i,j}$ are the modification on the weights $w_{i,j}$ of the lSVM model; the variables δ_i are slack variable necessary to keep the problem solvable at the price of some error on the prediction of the decision function for the adversarial sample of interest; finally, y_i are the correct values of the decision function of the lSVM classifier for the adversarial sample and x_j are the features of the adversarial sample of interest in the feature space. All the variable considered take real values.

$$\begin{aligned}
& min \sum_{i=1}^{c} \sum_{j=1}^{d} |\gamma_{i,j}| + \sum_{i=1}^{c} \delta_i \\
& y_i - \delta_i \leq \sum_{j=1}^{d} (w_{i,j} + \gamma_{i,j}) x_j \leq y_i + \delta_i \qquad \forall i = 1, ..., c \\
& \delta_i \geq 0 \qquad \forall i = 1, ..., c
\end{aligned} \tag{3}$$

In this case we consider only one adversarial, but the extension of the problem to the case in which more than one adversarial sample is considered is trivial. The cost function seeks to minimize the (absolute) variation of the weights of the lSVM, while satisfying the constraint of bringing the prediction of the decision function of the adversarial example *as close as possible* to the correct decision function. In the case of the CIFAR10 model we need a further simplification: even with the replacement of the fully connected layers the number of variables in the convex optimization problem is 40960 and the optimization procedure is not able to solve the problem. Therefore we decided to apply a feature-selection procedure on the output of the convolutional layers of our model. For each feature we consider two set of samples: the first one taken from the original inputs and the second one taken from the adversarial inputs. We compare the sets of samples using the Wilcoxon Signed Rank test against the null hypothesis that the two sets come from the same distribution. The procedure computes the p-values of the test for each feature and selects the ones which present a p-value below a given threshold: in our experiments we choose a threshold value of 0.1. The rationale of our procedure is to retain only the features which are affected significantly by the adversarial inputs and change only the corresponding weights in the SVM. After feature selection we manage to reduce the number of variables of the convex optimization problem below 6000, therefore making the problem manageable for the solver.

4 Experimental Results

We test our repair procedure on both the MNIST and CIFAR10 datasets, using the Fast Sign Gradient Method [3] as adversarial attack of choice. In order to generate adversarial samples for our models we utilize FoolBox [13] which provides a number of ready-made adversarial attacks. Another advantage of FoolBox is that it accepts as model to be attacked *every* valid PyTorch model, which allows us to attack also our hybrid models without complex workarounds. In our tests, firstly we analyze the loss of accuracy of our models corresponding to increasing magnitudes of the adversarial attack. We call the parameter which control this magnitude ε, and we show our results in figure 2.

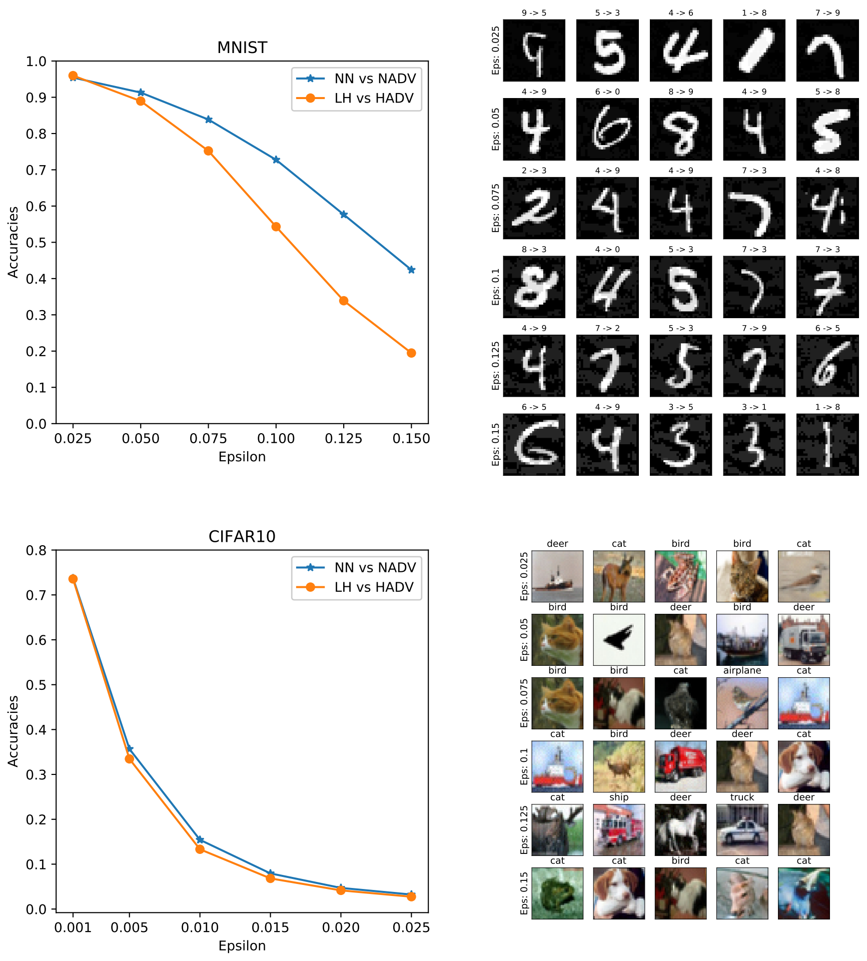

Fig. 2. Accuracies of MNIST-NN, MNIST-LH (above) and CIFAR10-NN, CIFAR10-LH (below) as ε increases (left) and graphics representation of some adversarial examples (right).

As it can be expected the accuracies of both the MNIST and CIFAR10 models drop for increasing values of ε and in general the LH models seem to be more vulnerable to this kind of adversarial attack. Given that our aim is to repair the LH models, this is not a limitation for our approach. As it can be observed in Figure 2 for $\varepsilon = 0.15$ the adversarial perturbation for the MNIST images is clearly recognizable even if it would not fool a human observer. For the CIFAR10 dataset we consider smaller adversarial perturbations: as can be seen in Figure 2 for $\varepsilon = 0.025$ the models accuracy is already below the baseline.

In our main experiment we analyzed the behavior of MNIST-LH and of its repaired version (MNIST-RLH) for $\varepsilon = [0.025, 0.05, 0.075, 0.1, 0.125, 0.15]$ and the behavior of CIFAR10-LH and of its repaired version (CIFAR10-RLH)

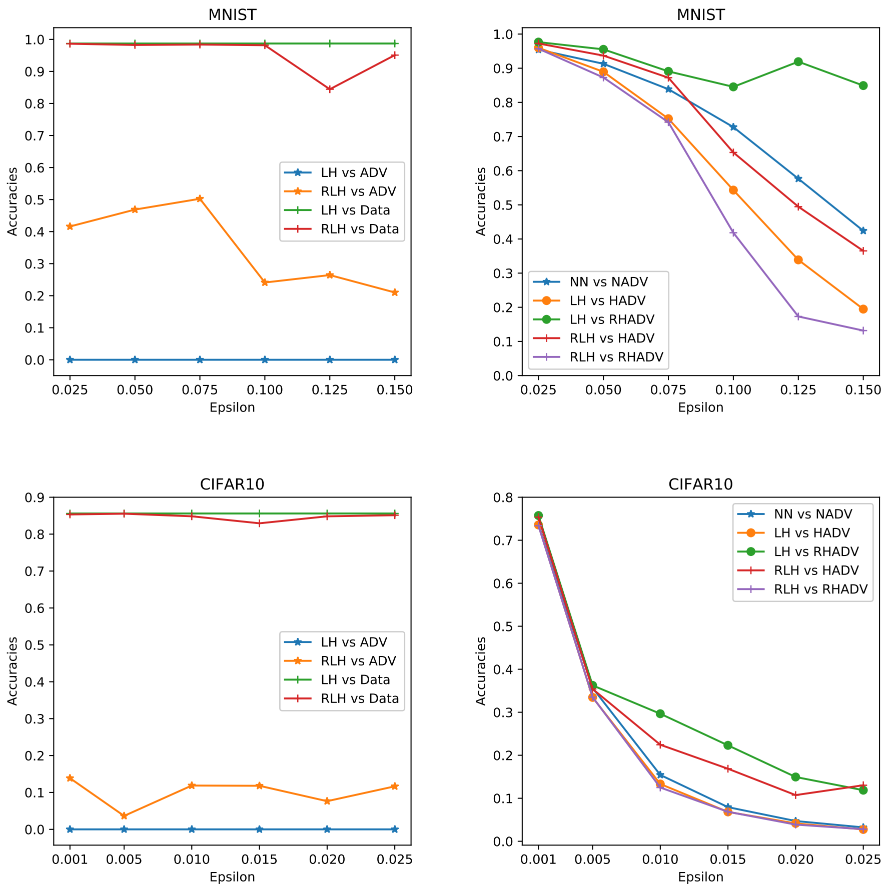

Fig. 3. Accuracies of the NN, LH and RLH models computed on different test sets of interest (MNIST above, CIFAR10 below). All the accuracies are computed for increasing values of ε.

for $\varepsilon = [.001, .005, .01, .015, 0.02, 0.025]$. More specifically, for each ε, we compute the accuracies of the LH, NN and RLH models on the following test sets: MNIST/CIFAR10 test set (Data), MNIST/CIFAR10 test sets in which all the images for which we found a corresponding adversarial example have been replaced with the adversarial example. Since the adversarial attack is model-dependent, the latter test set corresponds to three different sets computed on MNIST/CIFAR10-NN (NADV), MNIST/CIFAR10-LH (HADV) and MNIST/CIFAR10-RLH (RHADV).

In figure 3 (left) it is possible to see how repair affects the accuracy of the models both with respect to adversarial samples only (ADV) and with respect to the original test set (Data). In the case of MNIST, even considering only one adversarial in the optimization problem (3), the resulting model (MNIST-RLH) manages to generalize also on other adversarial examples, e.g. it manages to classify correctly *at least* 20% of the adversarial examples. In the case of CIFAR10, even if the repaired model is more accurate than the original one with respect to the adversarial samples the improvement is not substantial; in our opinion this is due to the fact that both the model and the dataset are more complex than the ones in MNIST. In figure 3 (right) it is also possible to see how the accuracy of the RLH models compares with the accuracies of the LH and NN ones with respect to the datasets NADV and HADV: from the images on the right it is clear that, while the repaired model is more robust to the adversarial sample computed on the non-repaired model, it has not acquired robustness against adversarial attacks in general. Moreover, it appears clear that the original model (NN) is still more robust to adversarial attacks. From the same images it is also possible to see that, as the RLH models are somewhat robust with respect to the adversarial example computed on the LH ones, so the LH models are somewhat robust with respect to the adversarial example computed on the RLH ones. This result suggests that the adversarial examples computed on the LH and RLH models belong to different categories of adversarial examples. This phenomenon requires further investigation to be confirmed.

5 Conclusions and Future Work

The main idea presented in this paper is to study the safety of DNNs in a "modular" fashion using techniques adopted from transfer learning. In particular, we consider how the properties of CNNs change if we swap the fully connected module with lSVMs obtaining hybrid networks. Our results confirm that such swap does not impact on the accuracy in a relevant manner, while making repair of hybrid networks feasible using a relatively simple encoding in a convex optimization problem. Adversarial examples can be found on hybrid networks more easily than on the original network: this result conforms to the hypothesis about the nature of adversarial examples presented in [3]. Our experimental results on MNIST show that, even using very few adversaries, the repair procedures manage to provide a model which presents an acceptable generalization on all the adversaries computed using the original hybrid model. On the other hand, our results on CIFAR10 show a more intricate picture, one in which the repaired network can be made robust against specific adversaries but generalization is still not completely achieved.

Given the results obtained from this work, our future lines of research will concentrate on understanding the properties of categories of adversarial samples in hybrid convolutional-lSVM networks and adding verification-driven kernels to our SVMs in order to obtain robust hybrid convolutional-SVM networks. Moreover, we will try to extend our work in order to repair CNNs without swapping away the fully connected modules and to explain how adversarial attacks affect the convolutional part of the networks and therefore the input in the feature space.

References

1. Frankle, J., Carbin, M.: The lottery ticket hypothesis: Finding sparse, trainable neural networks. arXiv preprint arXiv:1803.03635 (2018)
2. Gilmer, J., Adams, R.P., Goodfellow, I., Andersen, D., Dahl, G.E.: Motivating the rules of the game for adversarial example research. arXiv preprint arXiv:1807.06732 (2018)
3. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. In: ICLR (2015)
4. Guidotti, D., Leofante, F., Castellini, C., Tacchella, A.: Repairing learned controllers with convex optimization: A case study. In: CPAIOR. pp. 364–373 (2019)
5. Guidotti, D., Leofante, F., Tacchella, A., Castellini, C.: Improving reliability of myocontrol using formal verification. IEEE Transactions on Neural Systems and Rehabilitation Engineering **27**(4), 564–571 (2019)
6. Katz, G., Huang, D.A., Ibeling, D., Julian, K., Lazarus, C., Lim, R., Shah, P., Thakoor, S., Wu, H., Zeljic, A., Dill, D.L., Kochenderfer, M.J., Barrett, C.W.: The marabou framework for verification and analysis of deep neural networks. In: CAV. pp. 443–452 (2019)
7. LeCun, Y., Bengio, Y., Hinton, G.E.: Deep learning. Nature **521**(7553), 436–444 (2015)
8. Liu, W., Wang, Z., Liu, X., Zeng, N., Liu, Y., Alsaadi, F.E.: A survey of deep neural network architectures and their applications. Neurocomputing **234**, 11–26 (2017)
9. Nielsen, M.A.: Neural networks and deep learning, vol. 25. Determination press San Francisco, CA, USA: (2015)
10. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in pytorch (2017)
11. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)
12. Pulina, L., Tacchella, A.: An abstraction-refinement approach to verification of artificial neural networks. In: CAV. pp. 243–257 (2010)
13. Rauber, J., Brendel, W., Bethge, M.: Foolbox: A python toolbox to benchmark the robustness of machine learning models. arXiv preprint arXiv:1707.04131 (2017), http://arxiv.org/abs/1707.04131
14. Schwarz, M., Schulz, H., Behnke, S.: Rgb-d object recognition and pose estimation based on pre-trained convolutional neural network features. In: ICRA. pp. 1329–1335 (2015)
15. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
16. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I.J., Fergus, R.: Intriguing properties of neural networks. In: ICLR (2014)
17. Tang, Y.: Deep learning using linear support vector machines. arXiv preprint arXiv:1306.0239 (2013)
18. Torrey, L., Shavlik, J.: Transfer learning. In: Handbook of research on machine learning applications and trends: algorithms, methods, and techniques, pp. 242–264. IGI Global (2010)
19. Wong, E., Schmidt, F., Metzen, J.H., Kolter, J.Z.: Scaling provable adversarial defenses. In: NeurIPS. pp. 8400–8409 (2018)