Towards the Verification of Neural Networks for Critical Cyber-Physical Systems

Gábor Rabatin[1], András Vörös[1,2]
[1]Budapest University of Technology and Economics, Department of Measurement and Information Systems, Budapest, Hungary
[2]MTA-BME Lendület Cyber-Physical Systems Research Group, Hungary
mail: rabigabor@gmail.com, vori@mit.bme.hu

***Abstract*—Smart technologies are emerging in the field of Cyber-Physical Systems (CPS) yielding new challenges for system engineers. Rigorous techniques are necessitated to ensure the correct and accident-free behaviour of critical CPS. Neural networks gain increasing popularity in CPS, and even critical functionalities may rely on neural network based solutions. In this paper, we overview the literature and summarize our experiences of a neural network verification algorithm used in an academic case-study.**

***Index Terms*—neural network, verification, CPS**

I. Introduction

Recent advances in the field of artificial intelligence and especially neural networks led to the wide-spread application of smart techniques in Cyber-Physical Systems (CPS). However, a large set of CPSs can be regarded as critical, which necessitates their correct behaviour to be ensured. Neural networks can approximate arbitrary functions by using the so-called teaching process: input-output pairs are provided and the neural network learns an approximation by using various techniques such as back-propagation and optimization techniques to strengthen its generalization ability. Neural networks are gaining increasing importance in critical CPS, such as autonomous vehicles, smart factories and autonomous robots. Ensuring the correct behaviour of critical CPS is essential, where both design-time [1] and also run-time analysis techniques can be used. Typical design time verification techniques are testing and formal verification, both are exploited for neural networks.

A. (Deep) Neural Networks

Over the last decade, Deep Neural Networks (DNNs) achieved an impressive break-through in supervised, unsupervised and also reinforcement learning. DNNs are used in various fields such as image, video and speech recognition and they achieved better results in classification than humans [2]. The progress in the effectiveness of DNNs led to their application in critical systems.

A DNN consists of an input layer, multiple hidden layers and an output layer. Each layer contains multiple neurons as shown on Fig. 1. A *neuron* is a small unit inside a DNN which applies an *activation function* (e.g. sigmoid, hyperbolic tangent and Rectified Linear Unit (ReLU)) on its inputs and it passes the result to other neurons.

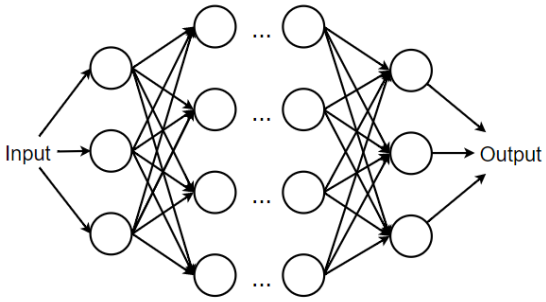

Fig. 1. Structure of a Multi-Layer Perceptron (MLP)

Each neuron has directed connections with neurons in the following layer. These connections have *weight parameters* which represent the strength of each connection. In supervised learning, these weights are learned during the training phase by minimizing a cost function over the training data. The most popular algorithm used for optimization is *gradient descent using backpropagation* [3]. The algorithm computes the gradients iteratively whereby if we changed our weights, then we could minimize the cost function over a given subset of training data.

Besides basic layers consisting only simple neurons, there are several other type of layers which have specific functions, like the convolutional layers which have been very popular recently, because it had reached significant process in image processing tasks.

An important weakness of DNNs is their sensitivity to adversarial examples. An extended definition of adversarial examples is proposed in [4] that captures all the important aspects, building on legal theory and the reasonable person test: a pair of inputs $x; x'$ is an adversarial example for a classifier, if a reasonable person would say they are of the same class but the classifier produces significantly different outputs.

Many techniques were developed to find adversarial examples for/against trained DNNs, where adversarial example synthetization algorithms may use the DNN as a white-box [5] or as a black-box [6].

B. Case-study: MoDeS3

Model-based Demonstrator for Smart and Safe Systems (MoDeS3) is used as a case-study in this paper. MoDeS3 is a demonstration system, where trains travel in a railway system controlled by the users. A camera-based monitoring system is installed to collect visual data and calculate the position information of the trains. If the monitoring system detects a dangerous situation, i.e. when the trains are too close to each other, then the safety subsystem stops the trains. A neural network is trained to detect the trains and estimate their positions. MoDeS3 demonstrates the usage of neural networks in a critical application.

C. Machine Learning Safety

The safety aspects of machine learning approaches are investigated in various papers [7], [8]. The application of AI techniques, and especially DNNs in safety application has to rely on various design time and runtime techniques [9]:

a) Redundancy and Dissimilarity: Redundant architectures are a common solution to improve the dependability of systems. Typically, some computing units (either replicated or dissimilar) calculates local results in parallel, and a voter then compares the different results and decides the final output based on the majority. Similar strategies can successfully be applied in ML.

b) Correct by Construction and Formal Verification: Correct-by-construction approaches heavily rely on formal methods to define and analyze the precise behavior of our systems.

c) Interpretability: It is highly desirable to understand the behavior of AI algorithms, and in our context, the behavior of DNNs both in design time and also at runtime.

II. Verification Techniques for Neural Networks

In the former section, we discussed the safety aspects of using AI technologies. In this section, we will only focus on various verification techniques, namely formal verification, testing and runtime verification.

A. Formal Verification

Formal verification is a technique to find a precise formal proof for the correctness of systems. Formal verification necessitates that both the property and also the system to be described in a formal (mathematically precise) way. Safety verification of DNNs is investigated in [10], where the authors propose a method to compute safe regions in a DNN by using an SMT[1] solver. They succeeded to do exhaustive search and find possible counter-examples or ensure that there cannot be one. Unfortunately, as the neural networks are getting deeper and deeper, the complexity of exhaustive search also increases.

A recent study shows an algorithm based on a modified *SMT solver* - the ReluPlex [11] - which can be used for formal verification on networks that are an order of magnitude larger than the largest networks verified using existing methods. The approach handles DNNs with a specific kind of activation function, called a Rectified Linear Unit (ReLU) and the verification algorithm searches out-of-bound behaviour of DNN-based controllers. The input of ReluPlex is a trained neural network, formally specified statements and a dataset, and it gives as output whether the statements can be satisfied.

[1]Satisfiability modulo theories

B. Testing

On the other hand, in most cases we cannot define a proper specification for the desired task (e.g. recognizing a lamp). Therefore formal verification cannot be used on them. In such cases we can have a separated validation set to determine whether the system generalizes well or overfits [12]. This works well if one can sample the distribution of the problem perfectly. If that does not occur then there is chance to have corner-cases where the model is not tested therefore can behave 'dangerously'. There are many recent studies which aim to develop new methods to ensure systematic testing with generating potential adversarial examples. Such studies are DeepTest [13], - which focuses on automated testing of autonomous cars by generating new inputs with the combination of different domain-specific image transformations - DeepSafe [14] - which tries to determine with data-guided clustering techniques whether *safe regions* within the network are robust against adversarial inputs - and DeepXplore [15] which will be discussed in this paper.

C. Runtime Verification and Monitoring

Runtime verification is a complementary technique when design time verification is not feasible. Monitoring the behaviour of neural networks relies on the traditional approaches used for runtime verification. In the MoDeS3 project, we introduced a monitoring framework to analyze the runtime behavior of the trained DNN responsible for detecting the objects. A Complex Event Processing (CEP) framework was used and a graph language helped us to specify the expected behavior of the system. The specification of the possible situations and also the simplified dynamics of the system was used to detect misclassification and other problems during runtime. The monitoring of DNNs used in real-life has to solve the challenge that typically those DNNs, which are used in CPSs, have to work in an evolving, formerly not fully known environment. This makes it difficult to define the monitored properties during design time.

III. Evaluation of DeepXplore on MoDeS3

Our main goal was to create a reliable system that can predict the positions of the trains in MoDeS3. This rather small demonstrator is used to evaluate and experiment with techniques that are used also by the industry in complex systems. The image recognition and classification task in the demonstrator would not require a complex DNN. However, we wanted to try out large networks to demonstrate the effectiveness of the verification algorithm on an illustrative example where the safety is critical. We used several image

processing techniques and neural networks to predict the positions and we used some post-processing mechanisms to filter out the incorrect predictions. In this section, our algorithm and the used techniques are explained and also DeepXplore is introduced to enhance the accuracy of our predictions.

A. Working of the Algorithm

We tried all of the state-of-the-art algorithms in object detection with neural networks, e.g. FRCNN [16], YOLO [17], YOLO9000 [18], SSD [19]. We found that we could get higher reliability and accuracy if we used an algorithm that consists of image preprocessing techniques and a neural-network based classifier and not an end-to-end object detector.

We take advantage of that our task is to predict the train positions on a video and there is only a small difference between the frames, therefore we can use background substraction [20] to reduce the complexity of the problem. After that, we use dilation and erosion [21] to get smoother blobs and from these, we filtered out the not proper train candidates by shape and size. The resulting candidates are used as input for the classifier neural network. The output of the network can be one of the train types (*Taurus, SNCF, Red*) or *Other* (e.g. the table, railway line, decorative element or a human's hand). These results are examined whether they could occur and only the possible positions are kept.

B. Trained Neural Networks

Different neural network structures are tried for this classification task. We used two pretrained neural networks (InceptionV3 [22], ResNet [23]) via transfer learning [24] and a basic neural network (target network) with convolutional layers in it, similar to LeNet-5 [25]. The test set has been sampled from the original data set, and it has not been used during the training phase at all. The results are shown in Table I.

TABLE I
EVALUATION OF USED MODELS

Model	Accuracy on test set	Training time (sec/epochs)
InceptionV3	99.17%	300
ResNet	99.38%	600
Basic network	99.72%	180

Our experience shows that the training dataset should not contain too many elements, because after about 2000 images per classes there would be many repetitions (due to the limited number of states in the railway systems). To enhance the learning, we used augmentation to artificially create new samples with shearing, rotation and brightening. The results shown above are achieved by early stopping [26].

C. Details of the Verification

Our main goal was to develop a trustworthy AI-based demonstrator, so we tried one of most prevalent techniques to verify our system: we used the DeepXplore's approach for testing the recognition capabilities and robustness of our trained DNN. For this purpose, we used three models (showed above) that are different and have been trained on the same train-classifying problem. We used a *verification dataset* consisting of 600 input images different from the training data.

a) Neuron Coverage: The proportion of the neurons that have been activated[2] on a specific input image. The bigger this value is, the bigger part of the neural network is responsible for computing the output. The optimization tries to find an attack with high neuron coverage.

b) Verification: The first step of the verification process is to find misclassification of the input images (described on Fig 2). The different networks are fed by the same input and the algorithm compares their output.

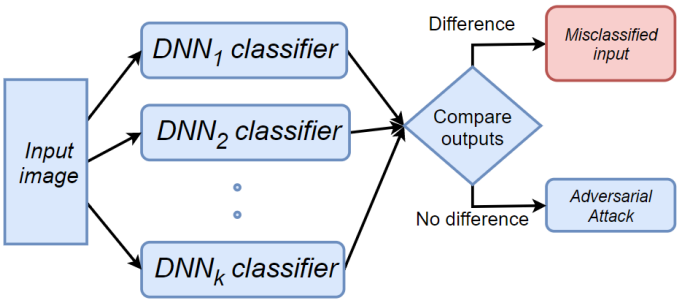

Fig. 2. Searching for misclassified images

If the outputs of the algorithms are the same, then the algorithm tries to modify the image and attack the target network.

c) Optimization Procedure: The joint objective can be described as depicted on Fig 3. the algorithm calculates the gradients of the used models on the images and this gradient value is used to alter the input image as follows. The objective for the modification of the images is to cause misclassification of (only) the target model. In addition, increasing the neuron coverage by the attack is the other optimization factor.

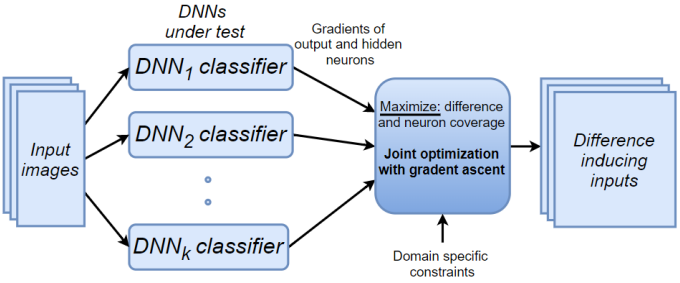

Fig. 3. Adversarial attack (DeepXplore workflow) [15]

d) Realistic Modifications: It is important to have realistic outputs, which can be ensured by generating inputs that need to satisfy several domain-specific constraints. In this case three different types of constraints are used: (1) *brightening* effects are for simulating different light conditions, (2) *occlusion* effects are for simulating that the attacker potentially is blocking some parts of the camera, and (3) *blackout* effects

[2]having an output greater than a predefined value (zero in this case)

are for simulating dirts on the camera lens. In all the three cases it is ensured that the pixel values are between 0 and 255.

This way, we generated several examples which misled our neural networks, example images are shown on Fig 4.

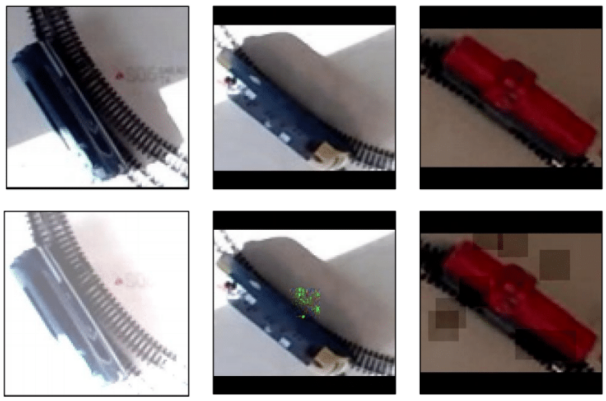

Fig. 4. First row: original images, second row: generated images by DeepXplore algorithm with *brightening, occlusion, blackout* image transformations (respectively)

D. Our Experiences

As we were able to generate a lot of adversarial images, we decided to use this technique as an augmentation technique to generate more training images. After that, we achieved *99.72%* test accuracy with the *Basic network*. We found that it is capable of not just helping to enhance the accuracy but finding the critical corner-cases for our neural network.

IV. Conclusion and Future Work

Our goal with this paper was to overview the current state of techniques for trustworthy DNNs. Various verification techniques exist for DNSs, such as testing, formal verification or monitoring. We chose the DeepXplore testing method and evaluated it on the MoDeS3 case-study.

In the future we plan to integrate the enhanced train-detector model in the MoDeS3 on a Jetson TX2 and experiment with formal verification techniques (e.g. ReluPlex) on a neural network based train controlling system. Besides, we want to develop adversarial transformations which are more similar to 'dangerous' corner-cases we experienced on MoDeS3 during open events.

Acknowledgment

This work was partially supported by the MTA-BME Lendület Cyber-Physical Systems Research Group and the ÚNKP-17-2-I. New National Excellence Program of the Ministry of Human Capacities.

References

[1] Tommaso Dreossi, Shromona Ghosh, Alberto L. Sangiovanni-Vincentelli, and Sanjit A. Seshia. Systematic testing of convolutional neural networks for autonomous driving. *CoRR*, abs/1708.03309, 2017.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015.

[3] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[4] Andrew Ilyas, Ajil Jalal, Eirini Asteri, Constantinos Daskalakis, and Alexandros G Dimakis. The robust manifold defense: Adversarial training using generative models. *arXiv preprint arXiv:1712.09196*, 2017.

[5] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[6] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against deep learning systems using adversarial examples. *arXiv preprint arXiv:1602.02697*, 2016.

[7] Rick Salay, Rodrigo Queiroz, and Krzysztof Czarnecki. An analysis of iso 26262: Using machine learning safely in automotive software. *arXiv preprint arXiv:1709.02435*, 2017.

[8] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in AI safety. *arXiv preprint arXiv:1606.06565*, 2016.

[9] CW Johnson. The increasing risks of risk assessment: On the rise of artificial intelligence and non-determinism in safety-critical systems. 2018.

[10] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. In Rupak Majumdar and Viktor Kunčak, editors, *Computer Aided Verification*, pages 3–29, Cham, 2017. Springer International Publishing.

[11] Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. *CoRR*, abs/1702.01135, 2017.

[12] Igor V. Tetko, David J. Livingstone, and Alexander I. Luik. Neural network studies. 1. comparison of overfitting and overtraining. *Journal of Chemical Information and Computer Sciences*, 35(5):826–833, 1995.

[13] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. *CoRR*, abs/1708.08559, 2017.

[14] Divya Gopinath, Guy Katz, Corina S. Pasareanu, and Clark Barrett. Deepsafe: A data-driven approach for checking adversarial robustness in neural networks. *CoRR*, abs/1710.00486, 2017.

[15] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. Deepxplore: Automated whitebox testing of deep learning systems. *CoRR*, abs/1705.06640, 2017.

[16] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149, June 2017.

[17] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[18] Joseph Redmon and Ali Farhadi. YOLO9000: better, faster, stronger. *CoRR*, abs/1612.08242, 2016.

[19] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: single shot multibox detector. *CoRR*, abs/1512.02325, 2015.

[20] T. Bouwmans, F. El Baf, and B. Vachon. Background modeling using mixture of gaussians for foreground detection a survey. In *Recent Patents on Computer Science*, pages 219–237, 2008.

[21] Jean Serra. *Image Analysis and Mathematical Morphology*. Academic Press, Inc., Orlando, FL, USA, 1983.

[22] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[24] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *CoRR*, abs/1411.1792, 2014.

[25] Yann Lecun, Lon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

[26] Yuan Yao, Lorenzo Rosasco, and Andrea Caponnetto. On early stopping in gradient descent learning. *Constructive Approximation*, 26(2):289–315, Aug 2007.