CLASSIFICATION BASED ON MISSING FEATURES IN DEEP CONVOLUTIONAL NEURAL NETWORKS

N. Milošević[*], *M. Racković*[†]

Abstract: Artificial Neural Networks, notably Convolutional Neural Networks (CNN) are widely used for classification purposes in different fields such as image classification, text classification and others. It is not uncommon therefore that these models are used in critical systems (e.g. self-driving cars), where robustness is a very important attribute. All Convolutional Neural Networks used for classification, classify based on the extracted features found in the input sample. In this paper, we present a novel approach of doing the opposite – classification based on features not present in the input sample. Obtained results show not only that this way of learning is indeed possible but also that the trained models become more robust in certain scenarios. The presented approach can be applied to any existing Convolutional Neural Network model and does not require any additional training data.

Key words: *neural networks, convolutional neural networks, neural network robustness, classification*

Received: December 13, 2018 **DOI:** 10.14311/NNW.2019.29.015
Revised and accepted: August 30, 2019

1. Introduction and motivation

The widely known neural network models for classification always use present features to figure out what the output class is. In other words, even though for many problems there is a finite set of features that are possible only the features that are present are used for classification.

Our approach is guided with intuition that neural networks can and should also take into consideration the features that are missing. For example for humans, when classifying images, it is beneficial to also look what is not present in a given image, and if we know all the possibilities, then we can deduce what the given image actually represents. Our modification to the training process and models tries to mimic this ability.

[*]Nemanja Milošević – Coresponding author; University of Novi Sad, Faculty of Sciences, Department of Mathematics and Informatics, Trg Dositeja Obradovića 3, 21000 Novi Sad, Serbia, E-mail: nmilosev@dmi.uns.ac.rs

[†]Miloš Racković; University of Novi Sad, Faculty of Sciences, Department of Mathematics and Informatics, Trg Dositeja Obradovića 3, 21000 Novi Sad, Serbia, E-mail: rackovic@dmi.uns.ac.rs

Fig. 1 *A motivational example where classification based on missing features would work in our dataset. Digit "5" from the MNIST dataset and its missing features named here: Feature 1 (on the left, circle-like feature) and Feature 2 (on the right, corner-line feature).*

In Fig. 1 we can see an example. Consider the given image of digit 5 (on the left) and two illustrative, very high-level features from our network model. Digit 5 can be defined in many ways, one of which is as "a digit missing these two features" (features are smaller squares on the right). To clarify, features are kernels (filters) from the convolutional layers in our model.

Circle-like Feature 1 given here is present in digits 0, 6, 8, 9 while a sharp corner-line Feature 2 is present in digits 1, 2, 3 (e.g. top-right corner, or the middle part), 4, and 7. Digit 5 does not have these features, therefore we can check the input image and see if these features are missing. If they are, we can safely assume that we are looking at digit 5.

This is not the only example where this is possible and this example is only given to clarify our way of thinking about "missing feature classification". In this paper we will demonstrate not only that this way of training neural networks is possible but also that it comes with the added benefit of increased robustness in trained models.

For a wide-spread adoption of systems that rely on neural networks it is needed to improve the current standard ways for training so the networks can be better prepared for intentional attacks and uncertain situations. It is very easy to describe this problem on the now standard task where neural networks are used – image classification [7]. Neural networks are widely used for image classification, especially ones with convolutional layers. However, new research is taking place to investigate how these networks can handle real-world situations where there is noise in the image, the image is of low quality or where image is not given in full [1, 3].

Our approach of classification based on missing features, as we will show, certainly can help with the last part – it can improve image classification accuracy with convolutional neural networks when they are faced with a task to classify an image by only seeing one part of it (partial input samples).

This proof-of-concept paper is just the first step towards larger research where the ultimate goal for us is to see where missing feature classification models are applicable firstly for robustness in image classification and then also for other problems.

One example which we hope to explore in our future research is traffic signs. In self-driving cars – a critical system that uses neural networks [2], traffic signs are processed as inputs from many cameras on a vehicle. These cameras are not perfect, but they produce very high quality images and usually the model used can easily detect and classify all traffic signs present in any given image. But what

happens when a traffic sign is obstructed by another object, for example a tree or another car? A person in a similar situation can deduce what sort of a sign it is just by looking at one part of it and it is reasonable to require from the CNN models to be able to do the same.

1.1 Related work

Classification based on missing features, as presented here, is a new area of research in the neural network research field. We believe this new family of neural network models can be used in many different scenarios. The main benefit of these models described in this paper is increasing robustness in partial input classification which is related to the neural network robustness, a growing topic in neural network research [3].

Another topic which should be mentioned here is adversarial attacks on neural networks research [1].

There have been some research papers similarly exploring how to increase robustness of neural networks when the inputs have been tampered with. However, our approach is not directly comparable with their approaches for many reasons i.e. different network architectures, usage of partial inputs in training, usage of adversarial examples in training, etc.

In [5] the MNIST [8] dataset used in our work was also used to investigate robustness of neural networks. In our paper parts of the input image were removed as will be explained later, while the authors in [5] describe a way to combine two images as an adversarial example input.

[18] and [6] described also on the MNIST dataset different modifications to the input image which affect the model greatly. In [18] MNIST dataset was used to investigate how different elements in input images maximize some network activations. The second part of the paper describes an adversarial attack on the network using previously gathered information. Our paper differs from this paper in that we are investigating how missing features are affecting classification instead of what features affect it the most, and that we are also using a convolutional neural network, while in this paper a traditional multi-layer fully connected network is used. Also in the mentioned paper, authors suggest that training with a mixture of adversarial and clean examples is a way to achieve better performance. In our case, we did not want to train on our generated adversarial (partial) examples, as will be explained later. Similarly in [6] adversarial examples are being used to increase neural network robustness, but they are being used during training.

2. Dataset

For this proof-of-concept research we decided to use widely known MNIST [8] dataset of handwritten digits. In addition to MNIST we also validated our work with the Extended MNIST dataset also known as EMNIST [4].

MNIST dataset consists of 60000 training examples (pairs of images and labels) and 10000 testing or validation examples. To try and mimic a real life scenario where we wanted to test out our neural network model, we decided to make a few other validation sets which also contain 10000 examples. The way we did it is

that we took the testing examples and removed some parts of every image, while keeping the label intact. It is important to clarify that we did not modify the training dataset. It is crucial to be able to train the network on the complete images, because in a real-world scenario we are unlikely to have partial inputs available for training. Also, we wanted to check if our network modification affects the standard, unmodified inputs.

We did not want to limit ourselves to only one validation set because it is difficult to decide what data should be removed from the images. So we created multiple validation sets:

- Horizontal cut dataset (top half removed)
- Vertical cut dataset (left half removed)
- Diagonal cut dataset (two diagonal quarters of the image removed – top-right and bottom-left)
- Triple cut dataset (three (9×9) pixel squares removed from coordinates (5, 5), (17, 10), and (7, 16) – this is roughly 30 % of the input image removed, but the locations were chosen so that they cover vital parts of the digits

Fig. 2 *Example of digit 3 in our validation set; From left to right: unmodified – original version, horizontally cut image – top half removed, vertically cut image – left half removed, diagonally cut image – first and third quadrants removed, "triple cut" image – three squares removed as described before*

We will refer to this dataset as PMNIST or partial MNIST dataset. The final created dataset then consisted of:

- 60000 training examples (unmodified)
- 10000 test examples (unmodified)
- 10000 horizontally cut validation examples
- 10000 vertically cut validation examples
- 10000 diagonally cut validation examples
- 10000 "triple-cut" validation examples

For the EMNIST dataset, we did exactly the same for two of its subsets. We first used the EMNIST-MNIST dataset whose structure is similar to MNIST dataset to validate our results, and then we tried our networks on the EMNIST-Balanced dataset which contains 131600 characters of digits and letters with 47 different classes for classification.

The EMNIST-MNIST dataset, contains 60000 images and labels in the training set and 10000 images and labels in the test set – exactly the same number of samples as in MNIST dataset. We used the process described above to generate four new validation sets, same as with PMNIST dataset. As for the EMNIST-Balanced dataset the same 85/15 % training and testing split was used to get a training set of 112800 images and labels and a test set of 18800 images and labels. Then, we generated four new test sets of size 18800, with different partial inputs for validation, same as before.

As both EMNIST and MNIST datasets have images of the same size (28x28 pixels) we used the exact same model architecture except for the last layer in the neural network which had to be changed to accommodate different number of classes in different datasets. The model is described in great detail in the next section.

It is important to note that during training the test sets and the newly introduced validation sets are not used. We want to completely avoid "peeking" at our validation data. That is why we split our dataset into three subsets: for training, testing and validation. All the models are trained and tested on complete input and output images and then validated on all validation data sets.

Another important remark is that approach described in this paper can work on any dataset, not just digits, letters etc. We chose these datasets because we can very clearly describe the features and their presence in a sample. With other datasets which have color or larger images this approach would still work but the features would be less interpretable.

3. Model architecture

For the purpose of testing our theory a fairly standard model was used. The model is a default example model used in machine learning frameworks (e.g. PyTorch [14]), so it was a good starting point for us to experiment with.

The model consists of five layers ordered in a common way – a number of convolutional layers followed by a number of fully connected layers:

1. Input layer – 2D grayscale image of size 28×28 pixels
2. 2D Convolutional layer 1 – 20 kernels of size 5×5, with max-pooling of stride 2, and ReLU activation function
3. 2D Convolutional layer 2 – 50 kernels of size 5×5, with max-pooling of stride 2, and ReLU activation function
4. Fully-connected layer 1 – 500 neurons, ReLU activation function
5. Fully-connected layer 2 – 10 output neurons for MNIST and EMNIST-MNIST cases, 47 output neurons for EMNIST-Balanced case, Softmax activation function

We used SGD (Stochastic Gradient Descent) with learning rate of 0.01, and momentum of 0.5. These values were also not changed from the given model. Again, the given model was not modified in any way apart from introducing the conditional

negation of the output vector of the last convolutional layer, as described in the following section.

3.1 Model upgrades

In the first section we showed an example where it is easy to see how missing features can be used to classify a digit. While this example shows what we want to do it requires some additional knowledge about the data used. We want to use standard datasets without any additional knowledge so our approach can be used in any scenario. The main question was how can we obtain the missing features in the input sample. In this and the following section we will explain how we can use existing knowledge from convolutional layers obtained with normal training in our modified approach.

We also realize that the features in these features vectors are not binary. For a person it is very easy to decide whether a feature is either present or not present in an image. Neural networks are more flexible and can also say "how much" a feature is in a given sample. If the features were binary it would be trivial to find all the missing features in an image by replacing values in the feature vector with their opposites. Also, at this stage, it is very hard for us to say which missing features are more important than others, we simply try to classify based on all missing features.

For our approach of classification on missing features the model had to be modified slightly. We modified the forward pass in the network to negate the vector which represents what features are present in an image. When negated, this vector will represent what features are not present in an image. To demonstrate on an example, imagine a feature vector where 1 denotes a present feature and 0 denotes a missing feature. By simply replacing zeros with ones and vice-versa we can obtain a vector with all the missing features in the feature vector.

The negation process takes place between the exit of the last convolutional layer in a network and the entry to the first fully connected layer. It can be applied after the activation function application in the last convolutional layer, or by modifying the activation function as we will cover later. At that point the signal passed through the neurons is simply a feature vector describing what features were detected by convolutional layers.

The negation operation largely depends on the activation function of the last convolutional layer. We have to negate the vector in a way that it represents the complete opposite of what would be the output of an unmodified network. The term "negate" probably can be replaced with "invert" in cases of some activation functions.

For example, for hyperbolic tangent function tanh which is used as an activation function in neural networks a simple negation is enough. The tanh function always returns a value between -1 and 1. If we agree that a present feature is represented by a value close to 1 and an absent feature is represented by a value of close to -1, it is easy to see that negating a whole vector of tanh outputs would provide us a vector of features that are missing.

Rectified linear unit ($ReLU$) [10] functions are also widely used in neural networks. The $ReLU$ function is different than the tanh function in that it returns

values between zero and positive infinity. Here, simply negating the vector would not work, but calculating a new vector is not complicated. The output with a positive value represents a present feature and the value of zero represents a missing feature. If we apply a simple function as such:

$$f(x) = 1 - x$$

we will get a vector representing what features are missing from the input image.

In our model there were a total of 800 values (a vector of length 800) which is the output of the last convolutional layer and the input to the first fully connected layer. This vector represents all the present features and their positions in the input sample. As we are using ($ReLU$) activation function, we can negate the vector using the formula above.

The implementation of this modification is very simple in PyTorch library. We only need to modify the "forward" function of the neural net Python module to negate the vector at a specific stage. The backwards pass is calculated automatically with autograd [15].

4. Training processes

The training processes are also modified from the standard process. The unmodified network is trained for 10 epochs in the provided model. After that, no significant increase in accuracy is noted, as the network already gets around 99 % accuracy on the test set. At this stage in our research we have experimented with several training processes.

The first one is to simply apply the model upgrade we described in the previous chapter and train the network normally. We called this model "ONN" (only negate network). Although this approach gives us some improvements, it does not represent fully what we wanted to do. Because the network is modified to negate the vector representing the features in the input image, we observed that our new model adapted to our layer inversion modification. The change affected the backwards pass in the network so the convolutional filters in convolutional layers were completely different opposed to the standard network. In other words, the features found in input images were not the same – the network learned them in a different way. As we wanted the features to remain the same and only to modify the later stages of the network, we thought of the second training process which allows this. Our goal is to classify on the same features but to emphasize those which are missing.

The second process requires a few extra steps and is as follows. The training process begins for a number of epochs where the "negation layer" is inactive. This is so that the convolutional layers inside the network learn all the features in the training data. In this step the filters inside the convolutional layers will learn both the high-level and low-level features of digits given the digit images from the dataset. We do this training step for 10 epochs, which is enough for the model to learn the features well enough.

The next step consists of freezing the convolutional layers and resetting the fully connected layers. The freezing of convolutional layers is necessary so that the further training does not affect them. The features represented in the convolutional kernels are learned already and we do not wish to modify them. The convolutional layers are simply going to be used for feature extraction at this and future points.

Resetting the fully connected layers is also necessary as we want the network to start over the learning process but to classify based on the missing features in an image. Resetting the layers simply means re-initializing them with the same initializer used in the model setup.

With the features learned, convolutional layers frozen and fully connected layers ready for new training, we can activate the modification in the model which will negate or invert the output of the convolutional layer. This is made possible by dynamic nature of execution which is available in PyTorch neural network library. This is the main reason why it was chosen to be used for this work.

The training is then continued on for another 10 epochs, making it 20 epochs in total which is more than normal training. It is important to clarify that while our modified network is in total trained for 10 epochs more than the standard network its fully connected layers are reset after the tenth epoch making it so the final models are equal in quantity of training received. Convolutional layers only receive the 10 epochs of training also, before being frozen. This approach is a hybrid between normal and our new way of training so we called it "HN" ("hybrid network").

In the next chapter, the result section also contains the results for two more experimental training processes very similar to this one.

The first modification to our described process is to skip resetting the fully connected layers after the features were learned (referred in the results tables as "NR" – "no reset"). In a way, this means that the network continues training after this step but in a different way. The reasoning for this approach is that there may still be useful weights in our fully connected layer which can improve the model accuracy even after we have trained with our inversion modification in place. We wanted to try to combine standard training with our modified way to see if synergy between standard and our training process has any effect.

Another modification we tried is to alternate between normal training and training with inversion modification. For a number of epochs, we train the network so that one epoch the network is unmodified and another epoch is with the inversion modification in place. This is an extension of the previous modification because we wanted to make sure that the order of training is not important. This method we called "ALT" method as it alternates between ways of training. We also noticed that the "ALT" training model works best with smaller learning rates. When using large learning rates, the model would change the weights too much when switching from one way of propagation to another. This is something to be aware of, if using this approach.

For all processes and models, because random initialization is used we made sure to test several times to avoid any coincidental results. In development stages a constant random seed method was used for reproducible results.

5. Results and observations

Since we are introducing a new neural network model in this work, we decided that the best baseline in comparing the results would be a traditional neural network with the same architecture ("SN"). This way we can be sure that our modification in the model is what we are benchmarking.

We are aware that a model architecture which would give even better results compared to our own model probably exists. It is a very difficult task in finding such a model, while making sure that our modification actually does affect the accuracy increase. This is why we decided to test our approach on a very well-known model to see how it behaves. Since our modification is simple and can be applied to any CNN model, we will definitely experiment with other models (network architectures) on our newly introduced validation sets to see how they perform.

After testing the models on the mentioned datasets we obtained the following results. First, we present the results on unmodified testing sets.

Dataset/Model	SN	ONN	HN	NR	ALT
Unmodified MNIST	99.13	98.90	99.18	99.21	99.05
Unmodified EMNIST-MNIST	99.18	99.07	99.16	99.15	99.00
Unmodified EMNIST-Balanced	87.14	87.62	87.38	86.78	87.92

Tab. I *Results with accuracy for all models and unmodified testing datasets. Here, SN denotes the standard, unmodified network, ONN denotes the network only trained with layer negation and HN denotes Hybrid network which was trained normally for a number of epochs but was then switched to negate the output of the last convolutional layer. The NR and ALT models are trained as explained in previous section. NR model is the model which is not reset (NR) after the inversion modification and the ALT model is extension of the NR model where the normal and inversed training takes place in alternating (ALT) epochs. All the values are percents which depict accuracy of a network on a given dataset.*

As seen in Tab. I the modified network models performed better in almost all of the standard unmodified test sets showing that classification on missing features does slightly improve accuracy when the input sample is given in full. We want to emphasize that this method of training while longer and slower does not negatively affect the network performance when the input is given in full. This is something we were hoping for to be achieved. These results also show that our initial assumption was correct – it is possible to train a neural network to classify based on missing features.

In Tabs. II, III and IV we present the accuracy percentages on the newly introduced validation sets. In Tab. II we show the results on the four PMNIST validation sets while in Tabs. III and IV we present the result on the four validation sets generated for EMNIST-MNIST and EMNIST-Balanced datasets, respectively.

Dataset/Model	SN	ONN	HN	NR	ALT
Horizontal cut	44.71	48.96	52.33	56.07	41.60
Vertical cut	57.46	64.64	60.45	66.07	69.66
Diagonal cut	52.97	59.59	55.40	56.01	62.49
Triple cut	40.68	34.62	41.19	41.73	46.40

Tab. II *Results with accuracy for all models used on newly introduced PMNIST validation sets.*

Dataset/Model	SN	ONN	HN	NR	ALT
Horizontal cut	49.07	51.34	54.76	48.70	48.73
Vertical cut	31.10	28.10	32.91	28.62	31.12
Diagonal cut	58.43	61.22	59.37	58.18	61.50
Triple cut	46.78	49.63	53.90	48.99	47.44

Tab. III *Results with accuracy for all models used on newly introduced EMNIST-MNIST validation sets.*

Dataset/Model	SN	ONN	HN	NR	ALT
Horizontal cut	20.95	26.97	26.34	19.32	26.53
Vertical cut	22.23	20.02	22.19	19.50	24.36
Diagonal cut	27.91	30.14	30.79	25.80	26.83
Triple cut	20.39	22.88	21.07	21.81	19.83

Tab. IV *Results with accuracy for all models used on newly introduced EMNIST-Balanced validation sets.*

6. Discussion and summary

When comparing our training processes or models (Tab. V), it is clear to see that some of them perform better in certain scenarios. However, apart from the 0.02 % accuracy loss on the unmodified EMNIST-MNIST test set, it is uniform that the newly introduced models featuring some shape of classification based on missing features outperform traditional neural network models, and in some cases by large margins. This is the most important finding in this experiment with our new models.

We also notice that models which use convolutional layer freezing outperform the model which just negates the convolutional feature vector (ONN). Also, strong performance of ALT network suggests there is some benefit of combining traditional neural network models with our newly introduced ones. We will discuss this further in the Future Work section. As for choosing what model would work best in a certain scenario, it is difficult to say with certainty. We suggest trying different models and deciding by testing them.

The different datasets we used all behave similarly. We see the largest accuracy increase of 12.2 % with the vertical cut validation set in the PMNIST set.

Dataset	Best model	Accuracy	Delta
Unmodified – PMNIST	NR	99.21	0.08
Horizontal cut – PMNIST	NR	56.07	11.36
Vertical cut – PMNIST	ALT	69.66	12.20
Diagonal cut – PMNIST	ALT	62.49	9.52
Triple cut – PMNIST	ALT	46.40	5.72
Unmodified – EMNIST-MNIST	HN	99.16	−0.02
Horizontal cut – EMNIST-MNIST	HN	54.76	5.69
Vertical cut – EMNIST-MNIST	HN	32.91	1.81
Diagonal cut – EMNIST-MNIST	ALT	61.50	3.07
Triple cut – EMNIST-MNIST	HN	53.90	7.12
Unmodified – EMNIST-Balanced	ALT	87.92	0.78
Horizontal cut – EMNIST-Balanced	ONN	26.97	6.02
Vertical cut – EMNIST-Balanced	ALT	24.36	2.13
Diagonal cut – EMNIST-Balanced	HN	30.79	2.88
Triple cut – EMNIST-Balanced	ONN	22.88	2.49

Tab. V *Results with showing what models worked best with different test and validation sets. The "Accuracy" column shows final, highest accuracy achieved while the "Delta" column shows accuracy gain over the standard unmodified network. Both "Accuracy" and "Delta" columns are given in percentages.*

To summarize, in this paper we decided to test out the idea that neural networks can be taught to classify based on missing features. Consequently we discovered that performance of neural networks can be improved for classification when partial inputs are given using this approach. We described our approach of classification based on missing features and tested it in well-known environments.

This paper draws these conclusions and observations:

- It is possible to train a convolutional neural network to classify based on missing features in the input sample.
- Our approach of "negating" feature vectors before passing them to fully connected layers implements this idea and shows that this simple modification can help in a scenario where a partial input is given.
- The performance of convolutional neural networks, as expected, degrades greatly when we use partial inputs.
- The PMNIST dataset and other partial datasets based on EMNIST dataset, can be used for checking how a network behaves when given a partial input to classify.
- We also showed four similar but different training techniques to maximize the usefulness of our modification. These training techniques can be used to experiment with different datasets.

The results show that classification based on missing features is possible and that these new models we introduced help in partial input scenarios. Our approach,

albeit much simpler than some other approaches (e.g. training with adversarial examples) can also help with a very difficult real-world problem of having partial inputs to classify in a critical environment. The partial input example is only one of many use cases for our models that we hope to discover.

6.1 Future work

During our experiments, especially with "ALT" and "NR" networks we realized that it would be of great benefit if we could somehow combine the traditionally trained network with our new "negatively" trained network. A synergy or ensemble of two or more networks trained in this way would, we believe, work even better than the models described in this paper. Some initial testing is already being done and we can see that this approach is promising.

Another logical step in this research is to test our training processes and the new model in a different scenario. We already mentioned that one possible good use case would be in traffic sign detection. This is something that we wish to pursue very soon. There are already some traffic sign datasets (e.g. [12]) but none with partially visible inputs so manual labelling will be required.

Another idea is to see what happens when the inputs are somehow differently modified opposed to simply removing their parts. As some of the related work would suggest [1,13], attacks with image noise and other methods are possible, and we wish to test out our model against those attacks. It stands to reason that this way of classification can help in some of those scenarios. Another addition to this idea is to replace removed parts of the inputs with other parts taken from different inputs with a different label similar to [5]. This would probably be a very strong attack to our model and we would like to see how it affects accuracy.

We also believe that these models and methods of training neural networks can be applied to regression problems, not just classification. Instead of training on missing features we want to introduce a new idea of negative learning or telling the network what surely is not the desired output. One example that is familiar to us is the basketball referee problem [16, 17] where the position of basketball referees needs to be decided based on the position of the ball and the players. There, we sometimes do not know the best positions for referees but we are certain what are the bad positions (where referees cannot see the ball), and we believe that making a model that can take this information into consideration will perform better than traditional model. This idea is somewhat related to our "negation" or "inversion" processes described here.

Another different model where a similar technique could be used is in reinforcement learning. For example in games [11] it is very easy to see how a missing feature would affect the model output. Therefore, it stands to reason that classification based on missing features would yield good results in this scenario.

Lastly, in generative models our approach is relevant for reconstructing images from their parts similar to [9]. Since we are trying to create a model where missing features are defining a class, we could also provide what features are missing and also where. This approach, we believe, would work well with a structures like GANs (Generative Adversarial Network) or VAEs (Variable Auto Encoder).

7. Source code

Work described in this paper is free open-source software distributed on Github under GNU Public License (v3). The results are therefore reproducible with the correct environment.

The source code is available at the following link:
https://github.com/nmilosev/negative-learning/

Acknowledgement

The work is partially supported by Ministry of Education and Science of the Republic of Serbia, through project no. OI174023: "Intelligent techniques and their integration into wide-spectrum decision support".

We would also like to thank NNW's anonymous reviewers for their comments and suggestions which greatly improved this work.

References

[1] BASTANI O., IOANNOU Y., LAMPROPOULOS L., VYTINIOTIS D., NORI A., CRIMINISI A. Measuring neural net robustness with constraints. In: *Advances in neural information processing systems*, 2016, pp. 2613–2621.

[2] BOJARSKI M., DEL TESTA D., DWORAKOWSKI D., FIRNER B., FLEPP B., GOYAL P., JACKEL L.D., MONFORT M., MULLER U., ZHANG J., ET AL. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016, doi: 10.1109/ivs.2017.7995975.

[3] CARLINI N., WAGNER D. Towards evaluating the robustness of neural networks. In: *2017 IEEE Symposium on Security and Privacy (SP)*, 2017, IEEE, pp. 39–57, doi: 10.1109/sp.2017.49.

[4] COHEN G., AFSHAR S., TAPSON J., VAN SCHAIK A. Emnist: an extension of mnist to handwritten letters. *arXiv preprint arXiv:1702.05373*, 2017.

[5] GLOBERSON A., ROWEIS S. Nightmare at test time: robust learning by feature deletion. In: *Proceedings of the 23rd international conference on Machine learning*, ACM, 2006, pp. 353–360, doi: 10.1145/1143844.1143889.

[6] GOODFELLOW I.J., SHLENS J., SZEGEDY C. Explaining and harnessing adversarial examples. corr., 2015.

[7] KRIZHEVSKY A., SUTSKEVER I., HINTON G.E. Imagenet classification with deep convolutional neural networks. In: *Advances in neural information processing systems*, 2012, pp. 1097–1105, doi: 10.1145/3065386.

[8] LECUN, Y. The mnist database of handwritten digits. http://yann.lecun.com/exdb/mnist/, 1998.

[9] LI F., ZURADAY J., WU W. Sparse representation learning of data by autoencoders with l^ sub 1/2^ regularization. *Neural Network World*, 2018, 28(2), pp. 133–147, doi: 10.14311/NNW.2018.28.008.

[10] MAAS A.L., HANNUN A.Y., NG A.Y. Rectifier nonlinearities improve neural network acoustic models. In: *Proc. icml.*, 2013, 30, p. 3.

[11] MNIH V., KAVUKCUOGLU K., SILVER D., GRAVES A., ANTONOGLOU I., WIERSTRA D., RIEDMILLER M. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, https://arxiv.org/abs/1312.5602, 2013.

[12] MØGELMOSE A., TRIVEDI M.M., MOESLUND T.B. Vision-based traffic sign detection and analysis for intelligent driver assistance systems: Perspectives and survey. *IEEE Trans. Intelligent Transportation Systems*, 2012, 13(4), pp. 1484–1497. doi: 10.1109/tits.2012.2209421.

[13] MOOSAVI-DEZFOOLI S.-M., FAWZI A., FROSSARD P. Deepfool: a simple and accurate method to fool deep neural networks. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2574–2582. doi: 10.1109/cvpr.2016.282.

[14] PASZKE A., GROSS S., CHINTALA S., CHANAN G. Pytorch. *Computer software. Vers. 1.2.0*, https://pytorch.org. 2019.

[15] PASZKE A., GROSS S., CHINTALA S., CHANAN G., YANG E., DEVITO Z., LIN Z., DESMAISON A., ANTIGA L., LERER A. Automatic differentiation in pytorch. *NIPS 2017 Workshop Autodiff Submission*, https://openreview.net/forum?id=BJJsrmfCZ, 2017.

[16] PECEV P., RACKOVIC M. Ltr–mdts structure–a structure for multiple dependent time series prediction. *Computer Science and Information Systems* 2017, 14(2), pp. 467–490, doi: 10.2298/CSIS150815004P.

[17] PECEV P., RACKOVIĆ M., IVKOVIĆ M. A system for deductive prediction and analysis of movement of basketball referees. *Multimedia Tools and Applications*, 2016, 75(23), pp. 16389–16416, doi: 10.1007/s11042-015-2938-1.

[18] SZEGEDY C., ZAREMBA W., SUTSKEVER I., BRUNA J., ERHAN D., GOODFELLOW I., FERGUS R. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, https://arxiv.org/abs/1312.6199, 2013.