Testing Deep Learning Models

Johannes Volk, Advisor: Simon Speth

Seminar: Software Quality (SS2021)
Technical University of Munich
{johannes.volk, simon.speth}@tum.de

Abstract. Machine Learning is evolving at a rapid speed and takes up an increasing part of our daily life. The safety and the robustness of these systems have to get ensured considering the economic and political importance. This paper compares recent methodologies for testing Deep Learning models among themselves and other more traditional software testing approaches. Different properties relevant to testing these systems get explained and illustrated with examples. The presented concepts get applied by putting the robustness of an open-source image recognition model to the test. The model's performance gets improved by training it with adversarial examples, and similar attacks are less likely to cause a misclassification. Further, recently proposed neuron activation metrics are explained and implemented within the program.

Keywords: machine learning · testing · deep neural network · robustness · adversarial examples · model confidence · coverage analysis

1 Introduction

1.1 Motivation

Lately, Machine Learning (ML) gets introduced to many places all across our economy and our society as a whole. According to market analysis from the International Data Corporation, the Artificial Intelligence (AI) sector is going to grow annually by approximately 17.5 percent over the next five years [10]. AI could even replace human labor in various fields within the process of a fourth industrial revolution. This increasing rate of dependence raises first concerns whether these technologies might be able to change history in an unforeseen, in some cases even unfavorable manner. Whether development outpaces our capabilities to control these technologies is one of many ethical problems our society will face over the upcoming years. The importance of ML in safety-critical applications will grow. Therefore it has to be ensured that these systems work as intended and the usage can be deemed safe. But the algorithms behind ML are far from perfect and can be tricked. It is possible to alter a model's behavior by changing the data it is trained on or by manipulating the input to be processed. Like in traditional Software Engineering (SE), it is a challenging task to create robust systems that can withstand any failure source confronted. Ensuring safety by testing these systems is of inevitable importance. In traditional SE

activities to ensure safety and robustness are well defined and can be done more straightforwardly than in the field of ML. This comparison is made in this paper alongside the presentation of recent discoveries around ML testing.

1.2 Content

This paper focuses on the different approaches recent research has developed to achieve mentioned robustness and verify the correctness of these systems. These strategies get compared to traditional software testing methodologies. Furthermore, the vast differences inherent to the fundamentals of data-trained programs are examined and analyzed if there even is a way to ensure total safety. First, to give an introduction to ML, relevant components, the structure, and the training of these systems are explained (section 2). section 3 focuses on the role of testing in traditional SE compared to ML. Different testing properties get described in terms of ML, and differences to traditional SE get outlined. Further, it is discussed whether ML programs can be classified as "not testable"[6]. In section 4 a closer look at coverage and activation metrics is taken. The concept and effectiveness of recently proposed activation patterns get reviewed. In section 5 the proposed metrics get applied to an open-source image recognition model, and an extension for classifying real-time video frames and implementation of the activation pattern approach is presented. Further, the neuron activation patterns get used to detect adversarial examples and other unsafe and misclassified inputs. In section 6 the findings of this paper get concluded.

2 Basics of Machine Learning

This section explains the fundamentals of ML that allow us to understand the terms and inherent processes within an ML system. Many testing-related activities base on these structural properties. Therefore a closer look at how Deep Learning works is needed.

2.1 Feedforward Neural Networks (FNN's)

Deep Learning Models rely on Deep Neural Networks (DNN's), which consist of multiple Nodes, so-called neurons, and interconnecting edges. We are just going to take a look at models that only have forwarding edges (FNN's), whereas others allow signals to pass through a layer multiple times (Recurrent Neural Networks (RNN's)). FNN's are structured into K different layers, which are further divided into one input layer L_1, one output layer L_K and multiple hidden layers $L_k : k \in [1; K]$ between them. Each Layer L_k consists of s_k neurons $n_{k,i} : i \in [1; s_k]$ (Figure 1). [15] Generally speaking FNN represent a function Φ that is build on more inherent functions ϕ_k, so that $\Phi = \phi_K \circ \phi_{K-1} \circ ... \circ \phi_3 \circ \phi_2$ [7]. The set input data in L_1 is given to the first and innermost function ϕ_2, which returns its set of results to the next layer being L_2. These calculations are iterated and pass through the depth of all layers reasoning the term deep in DNN.

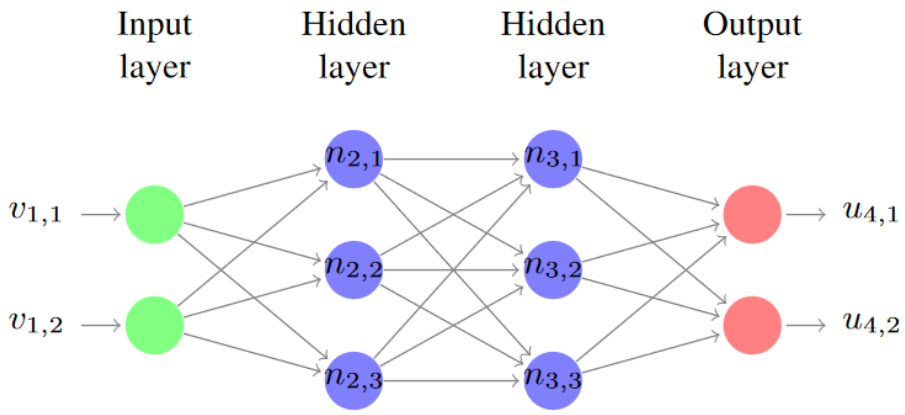


Fig. 1: Abstract structure of a simple FNN's

Finally, this leaves ϕ_K as the last function that yields the set of output values as the variables returned in the neurons of L_K. For a deeper understanding, a closer look at one of the ϕ_k is needed. ϕ_k is being given a set of input values $\{v_{k-1,h} | 1 \leq h \leq s_{k-1}\}$, being the values returned from the neurons of the previous Layer L_{k-1} or the input data in L_1 Each neuron $n_{k,l}$ of L_k has a weighted and connecting edge originating from the previous Layers L_{k-1} neurons $n_{k,h}$. These weights are labeled $w_{k-1,h,l}$ and get changed during the training process and are key to the models learning effect (subsection 2.2). The value $u_{k,l}$, processed within a neuron, is calculated as follows:

$$u_{k,l} = b_{k,l} + \sum_{1 \leq h \leq s_{k-1}} w_{k-1,h,l} * v_{k-1,h} \tag{1}$$

The value $v_{k,l}$, which should be returned to the corresponding next layer, has to be further calculated by applying the so-called activation function to the in Equation 1 calculated value $u_{k,l}$. This lets neurons "fire" only if a set level of activeness is surpassed. The so-called bias $b_{k,l}$ of $n_{k,l}$ changes a neurons threshold in a constant manner to adjust its reactiveness. There are different kinds of activation functions that each work around the same principles. For simplicity we only take a look at the Rectified Linear Unit (ReLU), which often gets used alongside others such as sigmoid or Tanh functions [12].

$$v_{k,l} = ReLU(u_{k,l}) = max\{0, u_{k,l}\} \tag{2}$$

Therefore ϕ_k is the composition of the activation function and the weighted summation of $u_{k,l}$. Input data may get transformed so that some of the network's neurons are not activated anymore, and only some get used. This reasons later discussed testing activities (see section 4). The last layer L_K does not use an activation function. Instead, the index l of the highest valued $u_{K,l}$ is being determined and resembles the so-called label linked to the input values. The possible

set of labels/classes gets defined as L. [15] With signals propagating through the model, each additional traversed layer adds some information about the input data. For example, in image recognition systems, these so-called features could be a property of the source image like a color or a geometric form. With increasing depth, each layer resolves more and more abstract features and the final function could decide if the image contains a certain object with its intrinsic characteristics and properties.

2.2 Training a Deep Learning Model

A supervised DNN gets trained with data that resembles inputs that the model should later classify according to the requirements. This so-called training data gets fed into the first layer of the DNN and gets transformed by the model. The predicted label for this data is compared to the beforehand assigned class and the model's weights are adjusted when the results deviate. By processing all training inputs, the model gets increasingly better at classifying and the loss function between the model's predictions and the reality minimizes. Further, so-called hyperparameters that tune learning algorithms get trained by validation data. Test data enables to check if the model works as intended and is robust towards new unseen inputs. This partitioning of the data set is known under the term cross validation (see subsection 3.2). [17]

3 Testing properties in Machine Learning

This section presents common properties and terms relevant for testing Deep Neural Networks. In the following, some of these testing properties, which get listed as most relevant by a literature study [17], will be summed up and explained within a further context.

3.1 Applying traditional Software Testing to ML

Software testing got created to verify more traditional systems and is relevant to Deep Learning-based systems as well. Bugs and errors get created alongside the functionality, therefore minimizing and detecting these imperfections are essential for both kinds. But ML systems can get more complex and hard to understand because of their large count of possible configurations. It is not unusual to have over a million trained weights. The training processes can last hours or even days. Therefore approaches that get used on traditional systems can not be translated directly towards ML. [5] Furthermore, a ML system is less understandable to humans due to a lack of transparency why components choose a specific result over another. There is no visible control flow, and it is not possible to use every debugging or testing strategy for classic code [14]. ML operates on probabilities, and underlying algorithms are less predictable. Some researchers even made a statement back in 2007 that ML systems are even "non-testable" [6]. Data might be insufficient to display all possible features or inputs

that carry an unforeseen property mislead the model's predictions. For example, image recognition models could get influenced by backgrounds or weather conditions displayed in a picture. Data might get mislabeled because backgrounds match the ones it got trained on and the object that should have been relevant loses influence on the prediction. Further, the mentioned perturbations like weather conditions can cause systems to produce incorrect outputs because the model never got confronted with such a situation. Therefore it isn't responding in the way it should be. This problem can get seen in section 5 where the noisy background of pictures hinders the classification process. This fact enables further test inputs by applying metamorphic relations or other transformed data that already got labeled manually. By showing the relevant information to the model in different settings and angles, the relevant features get learned, and less important ones get disregarded. The metamorphic testing approach focuses on inherent properties in the data that are very likely to interact before and after a transformation. If the relation breaks, it is likely that an error has occurred. For example, a search engine should yield more results for a single keyword, then a more specific request with this keyword and another one added to it. The specificness of a search and the scope of results should get maintained within a search engine. [6] Nonetheless, some concepts you would find in traditional SE are used in adapted ML testing approaches. Coverage criteria apply to line or branch coverage metrics but also neurons in DNN's. These coverage but also activation criteria are recently more and more developed. Their relevance and application within late research, in particular, get discussed in section 4. Further, a test oracle is relevant to both spaces of Software testing. A test oracle is defined as the mapping from inputs to the expected output the system should yield. In traditional SE this is done pretty easy, but in ML this gets challenging. Properties that should get tested are hard to specify, and labeling data is still time-consuming because it requires manual work in many cases. As indicated before, approaches for automatic test case generation with their corresponding test oracle exists, and the so-called Oracle Problem can get dealt with.[9] In the literature research, this section bases on few examples with some referenced here. The mentioned change of weather conditions alongside others get used as examples for this kind of domain-specific test input generation. Other input generations base on fuzzing or different transformation-based strategies where previously labeled data gets changed and creates new test data. Others focus on increasing coverage-metrics (subsection 4.2) or symbolic execution. Later is different from traditional symbolic execution due to the different structure and branching structure of DNN. Additional to the coverage of possible paths the execution can follow, the data is further analyzed. Testing data has to cover all possible ways a classification gets formulated, and every label gets represented through possible inputs. Statistical methods are applied to detect more possible errors when a few parts in the data are mutated. For example, symbolic execution can get used for creating attacks on image recognition DNN's by tracking the flow of data within the model and identifying paths from the output layer to the first layer. These paths allowed to manipulate only a few pixels in the input

layer that seemed essential for a decision made at the output layer and therefore resulting in a system failure [8]. Some of these mutations are even unrecognizable to the eye but can uncover corner cases and other defect relations by manipulating the model. It is possible to train the model with inputs like these and increase the performance when facing similar attacks in the future. [6] [17] The program used in 5 also leverages the generation of manipulated inputs data to increase the model's robustness against the attack that created the data in the first place.

3.2 Correctness

Correctness describes the relationship between the predicted labels and the actual labels of an input data set. The better the model is, the greater the accuracy and the smaller the loss function between correct and false classification gets [1]. The data used for testing should not be data the model gets trained on because generalization is relevant for future inputs, and new never-seen data should get tested. Correctness is hard to measure because there is no way of testing every possible input, so empirical correctness denotes the previously defined ratio of correct and incorrect labeling on a limited set of data. K-fold cross-validation allows an easy to achieve correctness measurement, where the whole labeled data set gets split into k smaller sets. All but one is used for training the model, and the remaining set gets used as the test set. This process gets iterated, and all subsets get to be the test set once. That allows for a greater testing spectrum and checks the model's correctness in slightly different settings by not increasing the needed data set. [17]

3.3 Model Relevance

ML models should not be over-dimensioned due to their size and complexity. An algorithm should only get as complicated as it needs to be for future data to be classified correctly. The problem of so-called overfitting arises from an oversized training process on the input data that hinders future, more general, and deviated inputs to be classified correctly. Model relevance describes how well the model fits the expected data that can get passed in a real-world applied online state of the system. Therefore classification boundaries are more robust against non-representative classified correctly training inputs located near or ulterior the bounds. This concept applies as well to the other side of the spectrum. Underfitting describes a model's state, which lets classifications boundaries disregard the intended and in the data displayed properties (see Figure 2). [17]

3.4 Robustness

The robustness of an ML system gets defined by its ability to withstand perturbations on its data or other components. The change of surrounding properties shouldn't affect the correctness of the system. Examples for these kinds of small

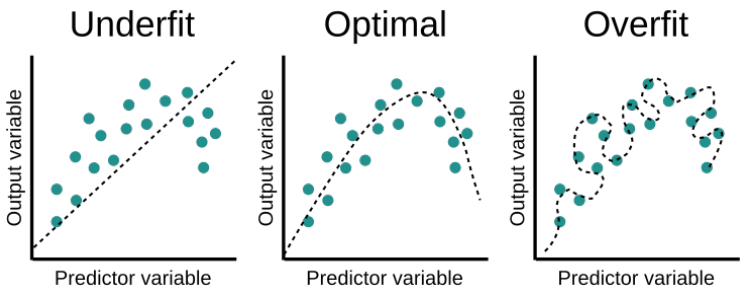


Fig. 2: Visual representation of model relevance [3]

changes range from the in subsection 3.1 outlined weather changes or other simple deviations like image resolution, coloring, or angles a picture got taken ín. All minor deviations shouldn't change the predicted label but might do due to imperfections. Adversarial robustness describes a subset of robustness but is relevant to many testing approaches leveraging so-called adversarial examples. These inputs get systematically constructed to cause a system to make a false prediction by altering critical parameters in the input data. Many of these implications are very tangent to other testing properties illustrated in subsection 3.5. A methodology for detecting these adversarial examples is described in subsection 4.3. Another strategy gets presented in section 5 where an actual model is used to generate adversarial attacks on it and even increasing its robustness by training the model with the newly created data. [17]

3.5 Security

As previously described, many safety-critical environments use ML and have to get protected against malicious intend. Political as well as economic interests depend on it. Robustness and correctness have to get ensured even when attackers specifically target weaknesses. Attacks range from the previously introduced adversarial examples over poisoned data, which is training data altered that back doors get created. Other attacks like security breaches within the database or the stealing of the model itself are scenarios you have to consider. To illustrate this: autonomous driving is a growing market and will replace many self-steered cars in the future. Therefore adversarial attacks display a hazard to public safety considering this growing digitization. In systems like these, not only the information system at hand displays a thread but every physical object around it as well. Previously described pixel attacks or other constructed mutations on the input data can result in life-threatening failure. In Figure 3 you can see a stop sign that got manipulated by some duct tape pattern that can alter a model's behavior. A human eye might disregard this intuitively as a minor thing, but ML Models can not achieve generalization to this degree. That gives the first insight into why politics and industry relatives are ought to keep a close watch

on developing trends these kinds of technologies bring with them. The military and economic implications immanent to this could cause serious harm to society. [2]

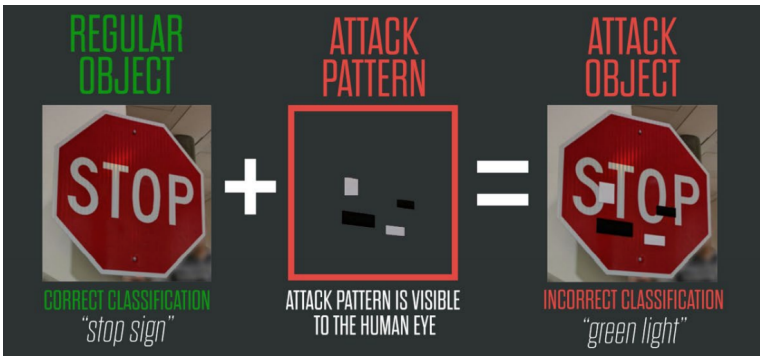


Fig. 3: Illustration of an adversarial attack on a physical object

3.6 Fairness

ML applications get used in sensitive environments that are commonly facing prejudice or other bias by society. The models trained on data should work without taking sensitive attributes unequally into account. No decision should be altered in an unwanted manner by the influence of characteristics like race, sex, religion, and many more protected attributes. The data a model gets trained on might contain these kinds of morally controvert biases. Therefore a close look during testing is needed and data preprocessed or filtered. [17] [16]

4 Coverage and activation Criteria in Deep Learning

This section presents the idea behind coverage criteria in classical SE and translates this concept to the space of ML. Native neuron coverage-based testing, as well as more recent and more developed coverage-focused metrics, are gathered. The latest research [13] presented a way to classify the confidence a model has on a specific prediction by calculating activation patterns for every labeled class. These patterns are generated during testing according to different criteria and can get compared to new ones originating from future real-world inputs. The more the patterns deviate, the less confident the prediction is. By applying this concept, malicious attacks can get detected, and misclassifications by the system can get prevented. Patterns that have their neuron activation patterns near the classification boundaries and therefore display other activation patterns are more likely to be manipulated and causing a system failure.

4.1 Coverage criterions in traditional SE

Coverage metrics often get used in software engineering. Line, branch, or functional coverage displays the degree to which the system gets tested and whether all parts get equally challenged to fail. A testing suite should test all parts of the system equally well and extensively. Therefore it should be possible to detect bugs, errors and corner cases in all areas of the software.

4.2 Neuron coverage

In Deep Learning (DL), neurons are the main components realizing the functionality of a system. Therefore Neuron Coverage was an early approach to validate the relevance and completeness of a test data set. The more neurons get activated during the processing of testing data, the more likely it is that wrong calibrated edges with their weights cause an error, and the underlying error can get identified. It got later discovered that high neuron coverage was easy to achieve and may not be an adequate and meaningful metric for testing. [11]

4.3 Activation patterns

Similar to neuron coverage metrics, activation metrics focus on the neurons in particular. As mentioned before, activation patterns can get used for the identification of possible failure-causing inputs. A level of confidence gets added to the degree to which a specific classification is predicted by a model. That increases the robustness and adaptability of the system against real-world data or never-seen data that even could be manipulated. These activation patterns can get constructed after different properties and their metrics. Furthermore, they enable testing on values taken from the last layer and even data generated from the feature space within every other layer. These patterns are generated by first dividing the neurons and layers into different sets saving the detected signature for each class/label. This signature describes the model's mean reactions to the inputs of the label. The data needed to determine these patterns gets taken from the test data set that is already labeled correctly to a high predetermined degree. The distances of newly runtime calculated activation patterns towards the previously collected trusted patterns display the degree of confidence the prediction has. Withing the following three of four different Coverage Analysis Methods (CAM's) to record these signatures presented in [13] get summed up. The collected data get structured by an aggregation algorithm, and the possible patterns get stored in a database. Further, you can visualize the process in which the patterns are created during the offline phase and later in the active online usage of the model (see Figure 4). [13]

4.4 Single Range Coverage

With Single Range Coverage (SRC) each neuron is analyzed individually by its scope of activeness, meaning that its minimal and maximum output values get

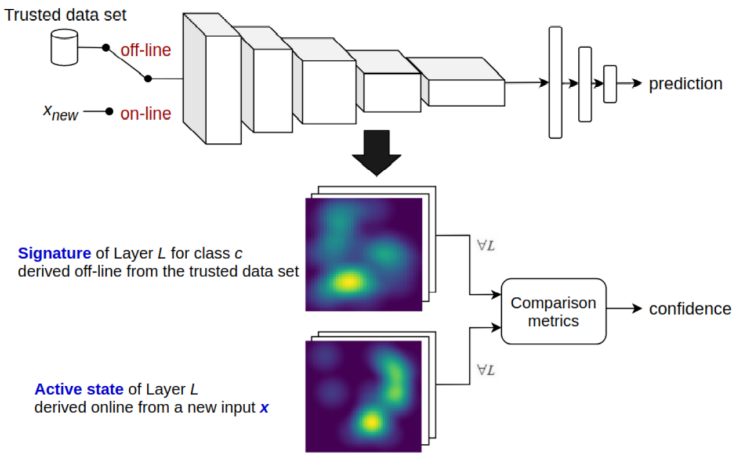


Fig. 4: Illustration of how the confidence level can get extracted from a ML model in addition to the prediction level

recorded. The trusted set used to generate the signature gets partitioned into n subsets, one for each label in the set of possible classes. Each neuron will be assigned one tuple containing its activation range for every single label the model differentiates. The confidence gets calculated by counting the activated neurons on the new unseen input data that is not within the previously determined intervals. It is possible to apply these metrics to only some of the layers due to performance and simplicity. The example in subsection 4.7 leveraged this and managed to achieve good results with less effort. [13]

4.5 Multi Range Coverage

Like with SRC the Multi Range Coverage (MRC) CAM calculates the activation intervals within each neuron for every class. These intervals get divided into equidistant subintervals. Further, it gets counted how often the neuron activation is located within a subinterval. Similar to the previous CAM, the confidence gets formulated on the number of neurons activated out of the offline-determined bounds. [13]

4.6 Neuron Rank Coverage

Neruon Rank Coverage (NRC) takes a different approach than the previous two. Each neuron's degree of activation on a specific input gets compared to all other neurons within its layer, and an activity ranking gets set up for this input. For each neuron and label, it's counted how often it appears in the top p positions within the ranking of all inputs. These counts get aggregated in the signature. A new online input x_{new} is deemed safer the fewer neurons are ranked lower than

the signature previously measured on the trusted data set. The higher neurons got ranked in the offline phase, the more impacting it is deemed to be. Therefore the rank also is taken into account when calculating the confidence. [13]

4.7 Applying the activation pattern Apporach

The researchers from [13] illustrate their approach by applying it to a small MNIST-based DNN. The MNIST data set is open-source and consists of 70,000 black and white images of handwritten numbers, 60,000 images for training the model, and 10.000 for testing it. The used model got taught to yield a correct prediction in more than 90 percent of the trusted set. It is to mention that the program used in section 5 achieves scores up to 97 percent on the testing data and therefore is of comparable quality. Adversarial examples get fed into the model, and it is reviewed how incorrectly the model's predictions are. SRC is used as the CAM with only the activity in the first convolutional block getting recorded due to efficiency. Unlike in section 5 they used ReLU as an activation function instead of sigmoid. This block consists of three layers, but their functionality, in particular, is not necessary to the observation made here. It has to get mentioned that even just a few observed neurons can prove the concept of CAM's and the usage of activation patterns for estimating a prediction's confidence level. Notice

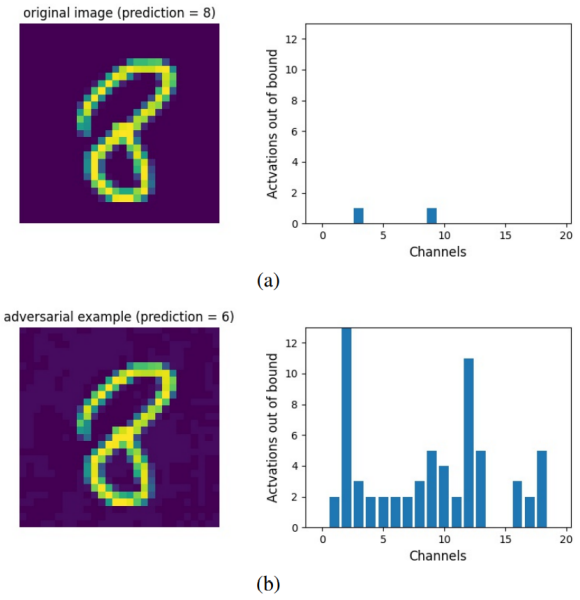


Fig. 5: (a) A correctly handwritten 8 with the associated diagram for the out-of-bounds activation pattern (b) A wrong predicted adversarial example of (a) with the associated diagram

Figure 5 wherein a conventional test image (a) displaying a six is classified by the model correctly with very little out-of-bound neuron activations and with a prediction score of 0.974. Whereas (b), a slightly altered version of image (a) is labeled falsely as an eight with a prediction score of 0.918. That could cause a big problem in another application context like previously described autonomous driving and the misclassified stop sign. The out-of-bounds neuron activations were counted in for each channel in the used convolutional neural network. For simplicity, it could also just be a usual DNN. The out-of-bounds activations describe the count of neurons that resemble an activation level that was not expected by the aggregated activation patterns and is unusual for the classified label. The mutation is so subtle you don't even notice it, but the model completely misevaluates the image. Yet, the used activation patterns recorded during training deviate significantly more with this adversarial example than in (a). The resulting low confidence score likely causes the adversarial example to be detected, and an error gets prevented. The authors of the research contemplated paper validate their approach on similar datasets and with different strategies for the generation of adversarial examples. The detection accuracy for adversarial examples reached over 90 percent and shows that with data sets like MNIST, it is possible to increase robustness and safety against manipulated data. The ability to filter inputs according to their error-causing capabilities can be deemed new and could get further explored on the path to solving the problem with adversarial examples. [13]

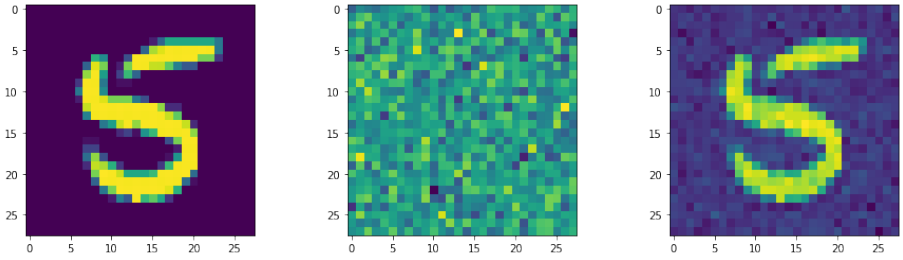
5 Application

In this section, a small application to the previously described concept gets made. The used implementation is an extended version described in [4] and is available on Github. It provides a network trained on the MNIST-Dataset with 784 neurons at the input layer for the 28x28 pixel-sized pictures, 100 neurons in the second layer, and ten at the output layer yielding the prediction. The model is claimed to reach about 97 percent accuracy on the test data set and classifies the 10,000 test images in under one second. The actual accuracy that was actually achieved only reached about 87 percent. The program got modified for this usage to work with Python version 3.9.5. The learning program was modified to utilize NVIDIA CUDA GPU acceleration with CuPy. The CuPy library translates Numpy instructions to GPU instructions as well as increases efficiency and learning speed. A newly trained model utilizes over a thousand neurons and classifies the test set in under six seconds. The training process got accelerated by a factor of around three across 30 learning iterations. The accuracy almost reached 96 percent which allows metrics to work more precisely and reliably.

5.1 Generating adversarial examples

The program allows for the generation of adversarial examples besides the capability to create and train new models. By leveraging backpropagation and

gradient descend, it is possible to generate new pictures classified as a previously determined digit. These pictures or so-called attack patterns look nothing like a digit and specifically intend to mislead the model It is now possible to generate a wrongly classified picture by modifying an image that shows another digit. This is done by taking a picture classified as a label x (6a) from the test data and merging it with an attack pattern/perturbation with different label x' (6b). In 6c you can see the resulting image after applying this process to a picture of a five that gets merged with the corresponding attack pattern and therefore gets assigned to the wrong label. The source claims to achieve high



(a) Image from the test data labeled five

(b) Plot of an attack pattern that gets classified as a two

(c) Modified image labeled two

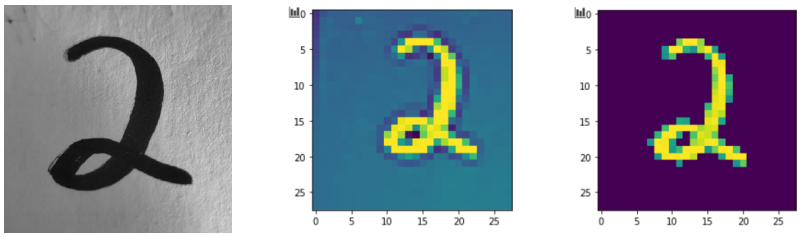
Fig. 6: Collague of pictures within the adversarial generation process

accuracy on misleading the model. Working with the project shows that some digit combinations didn't work quite as well as others. It was also evident that the used approach was not very efficient, nor could every input be manipulated as intended. Some images couldn't get manipulated as intended, and the generation of an attack took up to 5 seconds. The author noted that the given program's main goal was to provide an easy program to build on and add more features. Other more profound frameworks would be far better when it comes to the other stated terms and requirements. Another reason might be the usage of GPU instructions and the implications on efficiency losses caused by the transfer of data between CPU and accelerator. The generated adversarial examples could get used for training a new model that is more robust and safe against these kinds of attacks. Further, the accuracy of the generated attack patterns could be enhanced by increasing the number of iterations used and therefore time given to calculate them.

5.2 Captureing and classifying digits in real-time

The program gets extended by taking and processing pictures in real-time using a webcam or other video capturing device. For example, a smartphone's camera

can be connected and made available as a webcam to the computer using the Droid Cam Client and app. The open-source library OpenCV provides the necessary functionality to collect the data from the device for subsequent processing and filtering. In 7a you can see an example of how a taken raw video frame may look after extracting only the grayscale pixel information with OpenCV. These frames are further transformed by normalizing the pixel's values and reducing the resolution to the MNIST-like 28 by 28 pixels. The resulting image can be seen in 7b. Notice the color and features on the plane in the background that portrays additional unintended information that misleads the model. The MNIST data set contains pictures that do not have a background that carries misinformation, so the model cannot handle those inputs well. This causes a significant accuracy drop, and therefore further transformation is needed. Each pixel gets filtered according to an intensity threshold of 0.5, and fewer colored pixels get set to zero (see 7c). The processed data is fed into the previously presented model and gets classified as the corresponding digit.



(a) Unprocessed video frame captured by OpenCV

(b) Cropped and normalized video frame

(c) 50% threshold-filtered image

Fig. 7: Frames in different locations within the real-time video frame processing pipeline

5.3 Implementing SRC adversarial detection

In subsection 4.3 a strategy for determining the confidence on a prediction is described. This approach gets introduced to this open-source project and enables the detection of adversarial examples besides the in subsection 5.1 mentioned threshold approach. The SRC neuron activation patterns get calculated on the training data set, and the values recorded in the second layer are stored. In Figure 8 you can see a visual representation of these. They get constructed by adding the minimal and maximal values recorded at each of the 100 neurons in the intermediate layer of the small network. Please note that these patterns are quite different, but some features can be found in pairs or more digits. This property might originate from similar features that the digits share, but

this is just a suspicion. In Figure 9 the same concept is applied to the bigger network over 3 multiple layers. The ten patterns look similar to each other, but differences can get identified if looked at closely. The images that are shown here try to mimic the results visualized in Figure 4 and test the effectiveness of these activation patterns. For new inputs, the number of out-of-bounds activations η

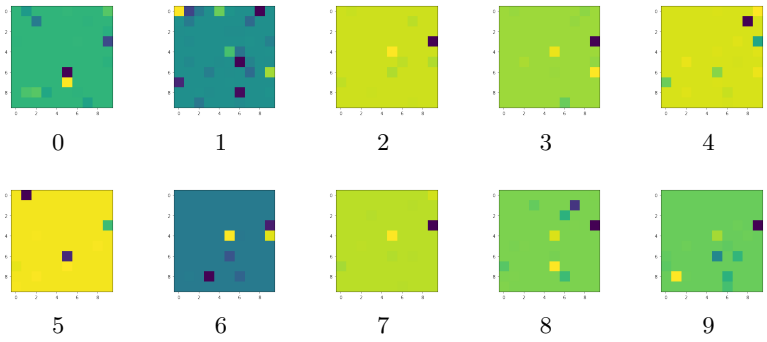


Fig. 8: Plots of SRC activation patterns with 100 recorded neurons

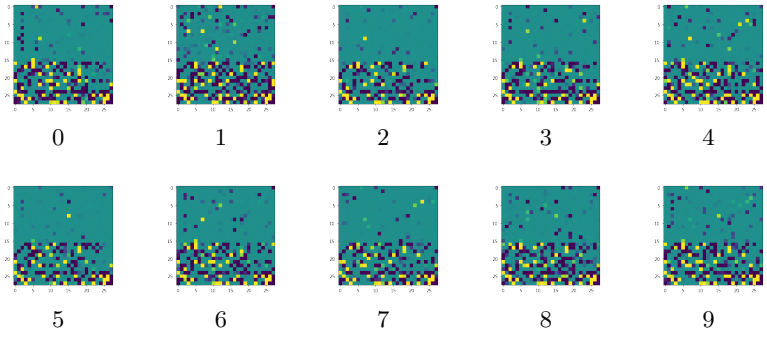


Fig. 9: Plots of SRC activation Patterns with 784 recorded neurons

gets counted. The higher this count gets, the more the input should be deemed unsafe. Further work could include an absolute metric that puts this count into context. Not every number should be as sensitive to out-of-bounds activation as others, and the larger a model gets, the more neurons can get activated in unusual patterns Therefore, a confidence level c for the classification as input from class $\hat{y}$ could get calculated after Equation 3. The confidence levels are tuned by parameters $\tau_{\hat{y}}$ that let us consider inputs with $c < 0.5$ unsafe and other

inputs as safely classified [13]. These normalized values allow for a comparison across different labels and models.

$$c = exp(\frac{-\eta * ln(2)}{\tau_{\hat{y}}}) \tag{3}$$

This would need more time to implement, but the plain number of out-of bounds-activations allows for an empirical verification that this approach can work. It was possible to detect adversarial examples generated by the procedure in subsection 5.1. Attack patterns wrongly classified with a confidence of 100 percent reached up to 200 out-of-bounds-activation within the 1000 recorded neurons. Pictures that were taken in real-time and had low to none disturbing pixel fragments in the background always stayed under 50 out-of-bound-activations. Therefore the activation pattern strategy can get implemented straightforwardly in this non-complex environment and yields results indicating its effectiveness on basic data sets like MNIST. Limitations arised and in many cases the learning program was not able to generate attack patterns reliably. This is not a problem to the concept but more so an unwanted feature of the used open-source program that is not profound enough for large models with more than 1000 neurons. You can find the code used here in the folder appended to the paper.

6 Conclusion

This seminar paper presented the most relevant and fundamental terms one should know about Deep Learning using FNN's. The structure and the functionality of these systems in terms of training and testing models got explained. Important testing properties and strategies are defined and differentiated from traditional testing. For this purpose, different literature studies and other research papers got summarized to give a brief overview of a vast topic in the field of ML. A recent approach towards increased confidence in a model's prediction was explained and implemented in an actual MNIST-based image classification program. The results seen on the detection of adversarial examples led to the estimation that the described approach is likely to work and might be a promising field for further research. Little effort enabled adversarial examples to be detected, and the model's safety against manipulation got enhanced. Even though the application in this paper and the proposed literature only managed to utilize the concept of activation patterns with small and plain MNIST-like data sets. Applying the concept to larger and more state-of-the-art networks could be the next step to take. Performance issues could surface because calculating and processing the activation patterns might not scale well. A problem that could arise is the strong connection to the training data. Therefore the performance is limited to the data, and the possible classification problem might not be covered exhaustively. So another way to go could be the development of new and more profound CAM's to extract more relevant and generalized information from the training data.

References

1. Braiek, H.B., Khomh, F.: On testing machine learning programs. CoRR **abs/1812.02257** (2018), http://arxiv.org/abs/1812.02257
2. Comiter, M.: Attacking artificial intelligence - ai's security vulnerability and what policymakers can do about it (2019), https://www.belfercenter.org/sites/default/files/2019-08/AttackingAI/AttackingAI.pdf
3. Dataman, D.: Machine learning or econometrics? (2019), https://medium.com/analytics-vidhya/machine-learning-or-econometrics-5127c1c2dc53
4. Geng, D., Veerapeneni, R.: Tricking neural networks: Create your own adversarial examples. https://github.com/dangeng/Simple_Adversarial_Examples (March 2019), https://medium.com/@ml.at.berkeley/tricking-neural-networks-create-your-own-adversarial-examples-a61eb7620fd8
5. Gerasimou, S., Eniser, H.F., Sen, A., Cakan, A.: Importance-driven deep learning system testing. CoRR **abs/2002.03433** (2020), https://arxiv.org/abs/2002.03433
6. Giray, G.: A software engineering perspective on engineering machine learning systems: State of the art and challenges. CoRR **abs/2012.07919** (2020), https://arxiv.org/abs/2012.07919
7. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016), http://www.deeplearningbook.org
8. Gopinath, D., Wang, K., Zhang, M., Pasareanu, C.S., Khurshid, S.: Symbolic execution for deep neural networks. CoRR **abs/1807.10439** (2018), http://arxiv.org/abs/1807.10439
9. Huang, S., Liu, E.H., Hui, Z.W., Tang, S.Q., Zhang, S.J.: Challenges of testing machine learning applications. International Journal of Performability Engineering **14**, 1275–1282 (06 2018). https://doi.org/10.23940/ijpe.18.06.p18.12751282
10. (IDC), I.D.C.: Idc forecasts improved growth for global ai market in 2021 (2021), https://www.idc.com/getdoc.jsp?containerId=prUS47482321
11. Mani, S., Sankaran, A., Tamilselvam, S., Sethi, A.: Coverage testing of deep learning models using dataset characterization. CoRR **abs/1911.07309** (2019), http://arxiv.org/abs/1911.07309
12. Ramachandran, P., Zoph, B., Le, Q.V.: Searching for activation functions (2017)
13. Rossolini, G., Biondi, A., Buttazzo, G.C.: Increasing the confidence of deep neural networks by coverage analysis. CoRR **abs/2101.12100** (2021), https://arxiv.org/abs/2101.12100
14. Sekhon, J., Fleming, C.: Towards improved testing for deep learning. CoRR **abs/1902.06320** (2019), http://arxiv.org/abs/1902.06320
15. Sun, Y., Huang, X., Kroening, D.: Testing deep neural networks. CoRR **abs/1803.04792** (2018), http://arxiv.org/abs/1803.04792
16. Udeshi, S., Arora, P., Chattopadhyay, S.: Automated directed fairness testing. CoRR **abs/1807.00468** (2018), http://arxiv.org/abs/1807.00468
17. Zhang, J.M., Harman, M., Ma, L., Liu, Y.: Machine learning testing: Survey, landscapes and horizons. CoRR **abs/1906.10742** (2019), http://arxiv.org/abs/1906.10742