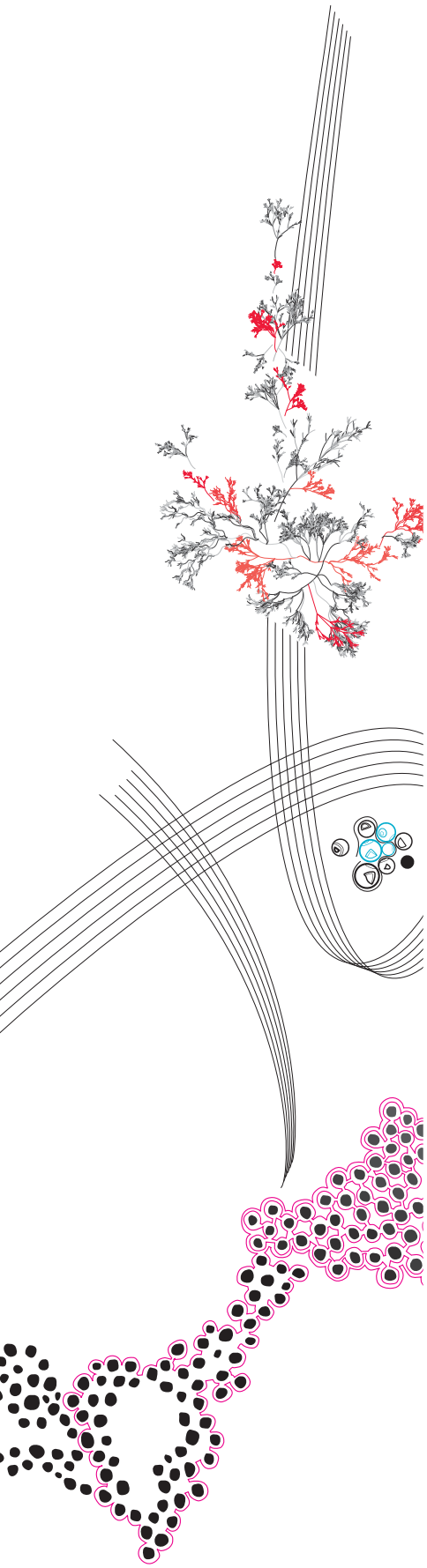

MSc Computer Science
Final Project

Self Organised Criticality and Avalanche Detection in Neural Networks

Tayfun AKIN

Supervisor: Moritz Hahn

July, 2025

Department of Computer Science
Faculty of Electrical Engineering,
Mathematics and Computer Science,
University of Twente

UNIVERSITY OF TWENTE.

Contents

Chapter 1

Introduction

In the domain of Machine Learning, which deals with algorithms that are designed to extract patterns from data that is fed to them, Neural Networks (NN) are a fundamental tool for tackling complex problems that may require higher levels of pattern recognition [22]. They require a large amount of data to be trained as compared to other methods and are relatively expensive to both train and operate. However, their ability to learn tasks that are mainly associated with human capabilities and depending on the context, being able to outperform human cognition makes them the choice of method for some domains [29].

NNs are what would be considered a Black-box system, yielding minimum information about the purpose and the structure of the neurons within them [25]. By their design, the influence that the neurons have on each other will form according to the training data and in an unpredictable fashion. This leads to neurons forming groups among themselves to achieve a particular step in the processing of its input, where their purpose in the overall system will not be immediately explainable to an outside observer.

The Black-box nature of the NNs causes them to be tough to evaluate for reliability and faults. This secretive nature means that certain components of the decision process (for cases where the decisions need multiple stages of reasoning to achieve) can not be tested individually, only the system as a whole. That means that the whole system can not be theoretically proven to be reliable and, only be assessed as far as the testing data allows. This fact raises large concerns for applications that are safety-critical, such as the automotive and the health industries [27].

One of the factors that hinder the reliability of NNs is the naturally occurring state of criticality [8]. This state increases the likelihood of the system to change its state with very minimal outside influence, making the system highly sensitive. Having the neurons set up in such a way that would result in critical behaviour, they get a chance to get the rest of the neurons to fire in a cascading manner, with their numbers of activations following a power-law. Naturally, when this kind of behaviour is not designed into the system, such disturbances can have negative effects on their performance. We aim to reduce the number of these disturbances in a system by analyzing the underlying structure which may lead to such behaviors and potentially devising a method in which the structure can be modified to minimize the effect of them.

Chapter 2

Theoretical Background

2.1 Neural Networks

Neural networks are a type of a model used in machine learning to capture information from a labeled set of data [6]. The main idea behind a NN is to mimic the working principles of organic nervous systems, having a large number of neurons, which are individual components of the system doing simple mathematical calculations. These systems are built with an underlying connections between neurons, which will transfer the output of one into another. These connections have an associated weight attribute, that is used to multiply the value being transferred. These values are modified during a training phase, where an algorithm called Backward Propagation [28] is used to break down the influence of other neurons over a select neuron, and will modify the weight values of the connections between these neurons to get closer to the desired output, given the input for that instance. In fig. 2.1, details regarding the anatomy of the implementations of a neuron can be seen.

Typically, a neuron has connections from multiple sources, all connected with their own synaptic weights. Then, these values are combined using a simple mathematical function, commonly chosen as the sum. Then, the result of this operation is taken through an activation function. This function will do a simple operation over the previously calculated value to represent the total activation value of the given neuron. A common function used for this operation is the ReLU function, that outputs a 0 if the given values is smaller than 0, otherwise outputs the given values without change. Furthermore, a threshold value can be chosen to again reduce the resulting number to 0 if its below this chosen number. This operation is done mainly to reduce the noise that may be caused by the input within the system; by not allowing very small numbers to propagate from the system.

For the setup of the system, the connections will be initialized with random values in a small range, as the algorithms that are used for the training of these systems work on the assumption that there are number in place already, that just need to be optimized. Over the training phase, these values will be shaped in a way to capture certain information from a provided input.

A typical NN is comprised of individual layers, that house neurons in parallel. These layers are referred to as either an input layer, a hidden layer or an output layer. Every system has one input and one output layer. The input layer's job is to be fed data directly. The input layer is usually connected to a series of hidden layers, which are layers that are only connect to other layers, however the inclusion of these layers are optional. At the end of it all, the final layer that is found is called an output layer. This layer is usually stretched out to represent an external concept, which would correspond to classes for which we are trying to train the network to detect.

In more advanced systems, we start to get more specialized layers, that can do extra operations and have different architectures [16]. A common practice is to have drop-out layers [31], which cancel a certain amount of connections at random on training rounds to make sure

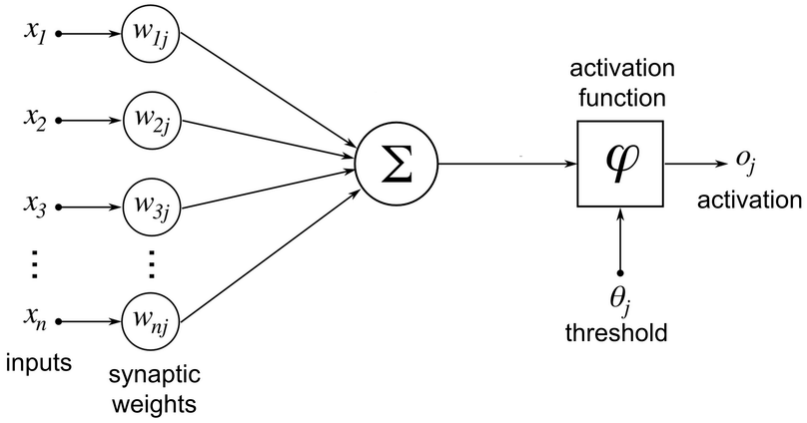

FIGURE 2.1: Anatomy of a neuron used in a digital neural network (diagram taken from [10]).

the system can develop a bit of redundancy and does not over specialize. Another one of these specialized layers is the convolution layer, which is used extensively for image processing tasks and usually makes up a large majority of a given network in such domains [9]. These layers work by having a filter move across the main input, calculating dot products over the area it goes over with itself, then recreating a new representation of the original data from the results of these. The size of the filter and how many rows the filter shifts after every calculation can be altered to fit the specific needs of the system. Neural networks who mainly employ convolution layers are referred to as Convolutional Neural Networks (CNN).

2.1.1 Convolutional Neural Networks

Convolutional neural networks are a kind of specialized version of the generic neural networks [15]. These networks are defined by having one ore more convolutional layers. The main advantage that comes from employing such systems is that they look at the input data in small groups of adjacent values [33]. These networks excel on tasks where the adjacency of the data points within your inputs are important and carry some information, also referred to as spacial data. Some of these areas include computer vision, natural language processing and signal processing.

What defines a convolutional layer is the act of compressing a group of values within the previous layer into a single value [32]. This operation is done through the utilization of a mask, most commonly referred to as the kernel. This kernel of a specific size (must be smaller than the input, however most common practical scenarios will use 3 by 3 or a 5 by 5 configuration) is then applied to a selection of points form the input, with the same size as the kernel. These values are put through a matrix multiplication, resulting in the reduction of their dimension by one. Then, this kernel is moved along and the operation is repeated until the whole of the input is scanned through. The amount of movement of the kernel, referred to as the stride, may be adjusted. These values are then put in the same order in a new space, forming the output. Essentially, the kernel acts as the connections between the input and the output. Since there are no direct connections like the standard layers, instead the values of the kernel are what will be adjusted while the system is being trained.

To help the performance of these layers, padding operations on the input data are usually performed. These operations are a simple way of increasing the size of the data, to make sure that the kernel can fit comfortably. Especially for smaller data, these assure that the kernel has enough space to generate a meaningful output size that can be worked on further by the rest of the network. One other benefit of these paddings is that in a typical setting, the kernel wont have the chance to examine the outer most values on their own; for example, a data point on the middle will be visited by the kernel multiple times, however if there is no padding, the

corner values will only be visited once. The padding ensures that these points also get the same amount of attention from the kernels passes as the rest of the data points. The values put into the padding sections are usually set as 0, in order to have the least affect on the output of the regions that include them.

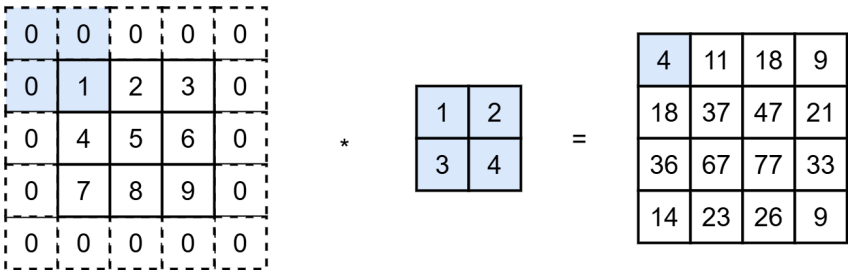

FIGURE 2.2: An example of how a convolutional layer's operations look, with a kernel size of 2x2, input size of 3x3, padding size of 1 and stirde of 1. (Diagram taken from [34])

2.2 Decision Trees

Decision trees are another type of a machine learning model, aimed at providing highly accurate results for problems that require relatively simpler logical steps, with significantly cheaper training and running costs as compared to neural networks [26]. The main idea behind decision trees is to divide the feature space into smaller regions that are more homogenized in the classes that they contain [19]. This operation is done recursively to create as large of a tree as needed, or allowed. In the end, the resulting model will go through the features in the order it deems the most efficient, and will return a result based on the regions that these features of the input lie in.

The core algorithm that the decision trees use to figure out how to divide the feature space employs a simple search strategy. In the training phase, the system will simply go through the features one by one, picking out boundaries between all the unique entries. Then, given one of these boundaries, the data will be split into 2 regions, one where the value of this given feature is below the decided boundary point (threshold), and one equal to or higher. Then, a quality metric will be calculated to asses how good the split was. After this operation has been done on ever boundary for a given feature, the best result of this quality metric will be chosen as the decision point of the given node. For the two regions that have been split, this process will be run again to create new nodes. This operation is continued until the max depth allowed for the tree has been reached or until the splits don't improve the decision making in a substantial amount.

2.2.1 Random Forests

Random forests (RF) are another machine learning method that is an improvement over the regular decision trees [17]. They are considered a type of an ensemble learning algorithm, meaning they combine multiple decision making algorithms to form their decisions [20]. Random forests, as their name suggests, contain multiple trees. When an input is supplied, the trees in the given RF all process it separately and create their own outputs. Then depending on the task, either these outputs are put through an accumulative function, such as averaging or put into a voting system to select one of the results.

One other procedure found in RFs is limitation of features. In the training phase, when the individual trees within the system are forming, the input that is fed into them is limited within the amount of features they include; in other words only a subset of the feature space. This both

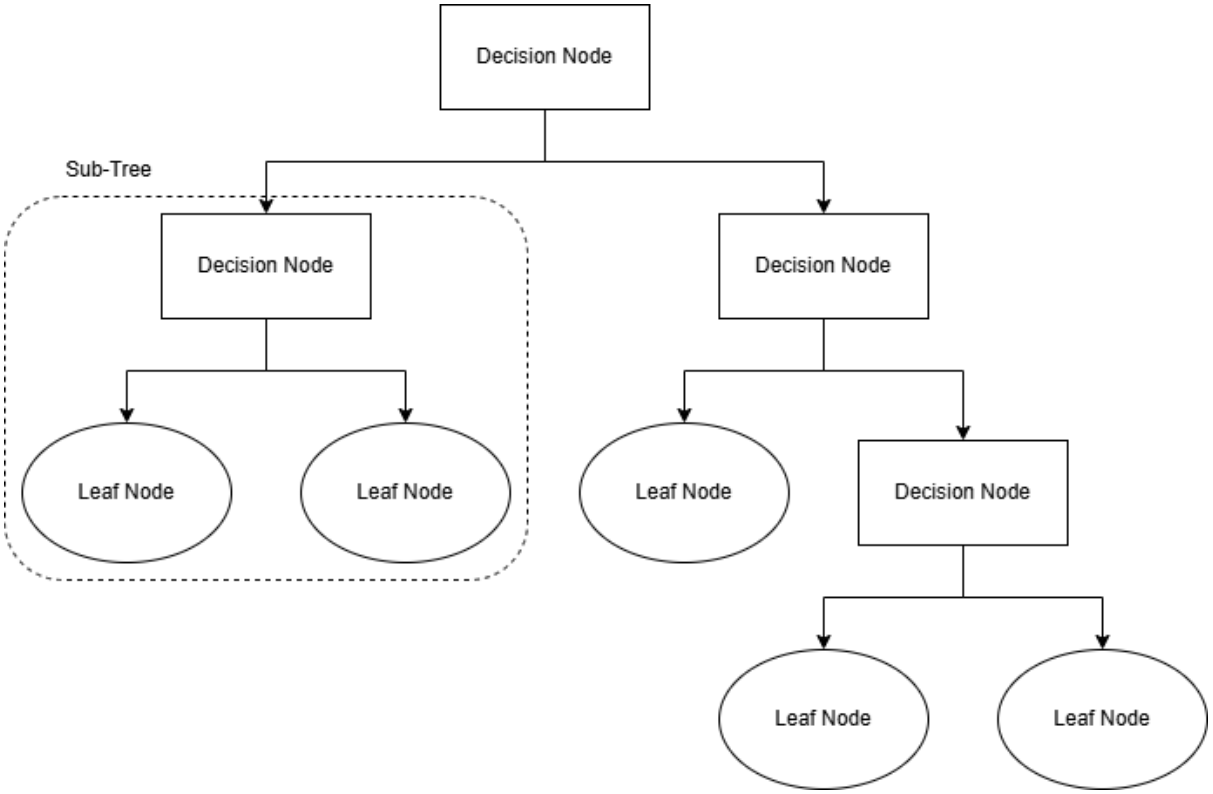

FIGURE 2.3: Anatomy of a decision tree, with 4 decision nodes, 5 leaf nodes and a depth of 3.

prevents overfitting and helps the system identify more complex relations between features, as each tree can focus on just a few features.

2.3 Critical State & Avalanches

Criticality is defined by the proximity of the system to multiple states at once, also referred to as the critical state, creating a dynamic where relatively smaller changes regarding the parameters may result in the system completely changing states [18]. The main cause of the occurrence of the critical states is for a system to be in a condition that would ease state transitions compared to how easy it would be to change states in regular operating conditions. As this critical state is reached, the systems gain the risk of crossing the boundary of two different states that would change the characteristics of its performance. When a system is at its critical point, the behavioral characteristics change from their expected patterns. At the state of criticality, the system behaves following power-law [12] and scale-invariant distributions [21], defined by some attributes of the system following the same distribution regardless of the time scale; meaning the shape of the figure that would be obtained by graphing some performance metrics of the system over a period of time, would still have the same shape if the time scale was changed. This also results in the distributions of the outputs that are coming from the system being heavy-tailed [24], meaning the tail end (the part that goes to infinity and gets smaller over time) occupies a larger sum of space in relation to the rest of the distribution, thus, increasing the range of values that may be expected. Coupled with the fact that a system may not behave this way in other states, raises concerns regarding their robustness and trustworthiness, as the chance of unexpected results is increased.

In NNs, the state of the individual neurons dictates how the system arrives at a decision and performs. These states are usually represented by an activation value, which can be turned into

a binary value, activated or non-activated, based on whether it's larger than a set threshold value (usually 0.50). In systems where a change in the state of a neuron may also trigger some other neurons' states to change as well, chains of varying sizes of such state changes being triggered by neighboring neurons will occur, referred to as a cascading chain. These cascading chains of events also referred to as avalanches, have direct links to the critical state and are the reason behind the sudden state changes in the systems [8]. As with the real world avalanches, this behavior is characterized by a small change happening to one end of a system, in our case the input, causing an effect in magnitudes in power on neighboring areas, in our case being the preceding layers. As they grow rapidly in size and have a disproportionally large effect on the state of the system compared to the relatively minor change that has caused them, they might prompt the system to behave in was that may not be what is expected. Although these avalanches can happen with varying sizes, our interests lie in relatively larger ones, where they may have significant effects on the output of the systems, as our main case, changing the output class in classification tasks. As commonly inspected on the Ising model [11], which is a mathematical model that consists of discrete variables who interact with their nearest neighbors, with the system evolving based on these interactions, the size of these avalanches follows a power law, in which smaller avalanches are more common and as the sizes increase, their rarity also increasing following a power relation.

The definition of avalanches can be formally defined by the out-liars in the statistical distributions of the firing patterns within a continuous range. As the inputs of the system in question changes, in our context being neural networks, the firing patterns of neurons, both in terms of paths taken and total number of neurons activated, will largely remain in the center of a normal distribution. The avalanches can be thought of as the instances where the firing patterns deviate from the mean when the input change is kept at a constant; as since the input change is constant, we also would expect a constant change in the output. The exact definition of what would be considered an avalanche can vary based on the context and the system at hand, so should be set up in relation to the testing environment.

2.4 Self Organized Criticality in DNNs

Many natural phenomena, such as some biological [30], neurological [23] and physical [7] systems, showcase behavioral patterns that drive them toward a critical state. Since this state is characterized by the sensitivity or fragility of the response a system may give out after a relatively small change within the input parameters, It has been found by previous studies that this state is beneficial for the traversal of information within the system [13]. As such, many systems experience a phenomenon that drives them towards learning behaviors that would establish a critical state over time. This has been observed in many natural phenomena and more importantly for our work, also in the structure of DNNs. As this is a naturally occurring phenomenon in multiple system by their nature, it is also crucial that the effects and implications of this tendency are understood.

2.5 Performance Metrics for Machine Learning Models

To showcase the performance of the models we have trained, a multitude of empirical measurements have been proposed and used over the years [35]. Within the current state of data science, there are four metrics commonly used to denote the performance and behavior of models [14]: Accuracy, Recall, Precision and F1-Score. These metrics are based on whats called a confusion matrix, which is a 2x2 matrix used to display the predictions of a decision making model based on binary classification; meaning the result can be either one of 2 possible classes. As seen on 2.4, we have 2 rows and columns; the Y axis corresponding to what our model has predicted and the X axis representing what the answer should have been according to the real data. The top

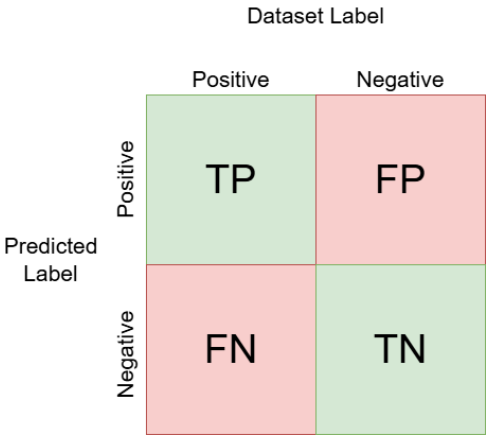

FIGURE 2.4: Visual representation of the confusion matrix.

left box is referred to as the True Positives (TP), meaning they were classified as positive by our model while they were also labeled by positive by our dataset. The top right box is referred to as the Fals Positives (FP), representing the instances where our model classified them as positive classes however they were labeled as negatives in the dataset. The bottom left box is called the False Negatives, representing the instances that were labeled as negatives by the decision making model however were meant to be positives according to the dataset. And finally the bottom right box is referred to as the True Negatives, representing instances that were classified as negatives by the decision making model while also being labeled as negatives according to the dataset. For the future of our work, we will not use this matrix directly, however the performance metrics that we will be using are directly based on these 4 boxes of the confusion matrix. Below, you can find detailed descriptions and formulas explaining how they are calculated and what they represent.

2.5.1 Accuracy

Accuracy is a simple metric representing the correctness of the guesses made by a model. It is calculated by dividing the number of correct guesses by the number of total guesses. This is a generic metric that shows us the likelihood of a guess being made by our system to be correct. It is a generic indicator of the success of a model and for most cases should not be used on its own, especially for environments where the data used has an asymmetrical distribution of the different classes present. In more formal terms, using the confusion matrix as a basis for our calculations the equation for accuracy is as shown in Equation 2.1.

$$ACCURACY = \frac{TP + TN}{TP + FP + TN + FN} \tag{2.1}$$

2.5.2 Recall

Recall is a metric that represents the classification misses on the positive class. This metric is most useful when the classes are not perfectly symmetrical, and there is an external importance put on identifying members of the positive class. To simply put, the recall metric represents how many instances of the positive class, has gotten labeled positive by the decision making model. For most cases, getting a high recall is easy of the model sacrifices its precision, and vice versa. As such, this metric is always combined with precision to make sure such sacrifices were not made. In formal terms, the recall value is calculated by diving the true positives by the total amount of positives, or in other words the sum of true positives and false negatives, as seen in Equation 2.2.

$$RECALL = \frac{TP}{TP + FN} \tag{2.2}$$

2.5.3 Precision

Precision is a score representing the correctness of the decision making model's guesses when the guess is of a positive class. This metric mainly checks the models to see if they have enough restraint to only make positive guesses when the system is relatively sure. In more formal terms, the precision value is calculated by dividing true positives by the total amount of positive guesses, or in other words true positives and false positives, as seen in Equation 2.3.

$$PRECISION = \frac{TP}{TP + FP} \tag{2.3}$$

2.5.4 F1-Score

F1-Score is the most advanced metric out of the 4 we have mentioned. To simply put, F1-Score is a metric that combines the precision and recall values, making it only yield high results when the model in question can both recall most of the positive classes and do so while giving away very minimal wrong guesses for the positive classes. As this metric covers both reach and the precision of the model's guesses, it is the best metric to represent a model's success. The formula for the F1-Score will be represented using the Precision and Recall values directly instead of using the confusion matrix for simplicity. The F1-Score is calculated by diving the multiplication of the precision and recall score by their sum, then multiplying the result by 2, as seen on Equation 2.4.

$$F1\text{-}SCORE = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{2.4}$$

Chapter 3

Avalanche Detection

One of the research gaps that we have identified within this domain is the detection of avalanches that occur within DNNs. As relatively large avalanches may cause abrupt changes in the behavior of a system, e.g. the classification of an image changing with minimal modifications to the picture, the need to be able to detect and potentially predict/avoid these avalanches rises. In order to be able to understand the behavior of a system, viewed through empirical measurements, we will first create an environment that simulates a use case scenario from a real-world application. While trying to induce an avalanche, we will monitor the system and examine the statistics we have gathered to see if an anomaly can be detected within these measurements at moments where the system is at a critical state.

To be able to induce and analyze a critical state for our network, we propose the experimental setup that is explained in section 4.

3.1 Application Domain

The main application of this process would be environments with continuous monitoring, as the avalanches that are being aimed to detect are formed through temporal changes. By modifying them, to include a second monitoring system, the aim is to increase accuracy and the trustability. Of course, extra modifications on these decision systems such as NNs come at a cost, be it computational effort, cost, or performance. This means that such protections and/or improvements are more suited in environments where the extra benefit that they provide are at a higher value.

As an example and the main source of our inspiration, is the automotive industry. Especially with the rises of the trend of autonomous driving and the improvements in technologies leading companies to be able to create products that perform at a high enough level to be adopted for public use, we see a lot of interest that goes into both creating more robust networks [4] and improving their safety against malicious users. One such case that these systems might need to be defended against would be the use of adversarial attacks [5], where a third party may manipulate an image using a mask, changing specific values in pixels, subtle enough to be impossible to detect by a human eye however changing the classification of the system completely. In another form, could include displaying images to these sensors that are sparse in the dataset to the point where the system is very inefficient at detecting them, causing the system to misclassify the objects around them. Since driving and transport activities are not completely safe, errors in these programs and/or deliberate attacks might have great consequences for human well-being. Being able to detect and possibly prevent these kinds of attacks would be a great way to improve the safety of these systems.

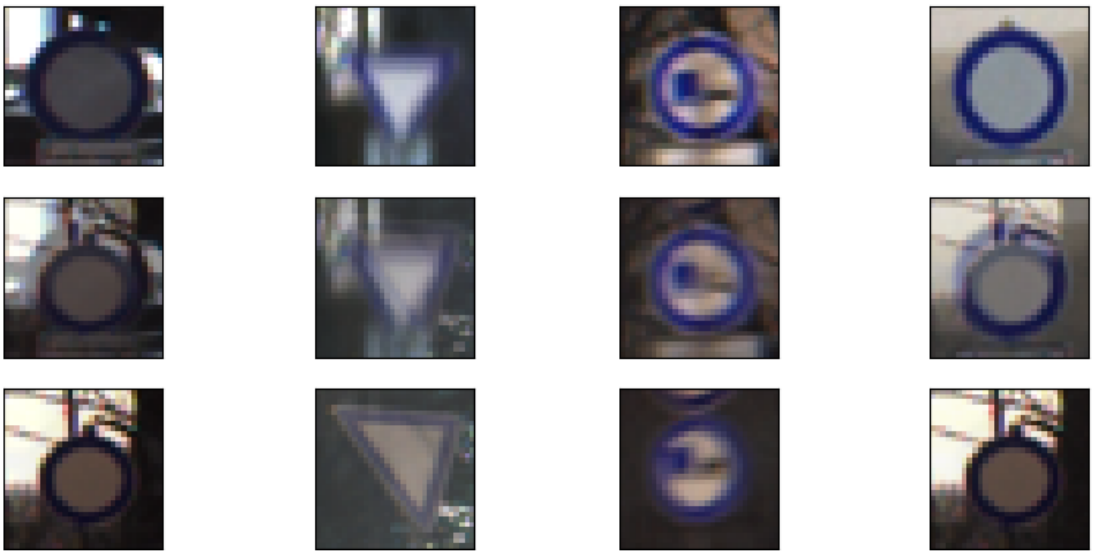

FIGURE 3.1: Example of 4 different morph cases on multiple sign types, and images created at the halfway point of the transition.

3.2 Data Collection

In order to detect these avalanches, we will first create a dataset that will house points of potential avalanches and a series of metrics collected from the system around those points. Since getting video data at a decent frame rate and labeling each frame will be above the limitations of our project, the main data collection we propose will consist of simulating a video environment.

We propose a method we will refer to as the Morphing, in which we take 2 pictures and alter the pixel values to create "in-between" pictures that are a combination of both with varying degrees of similarity to either one of the pictures. This operation will be performed on discreet steps within the theoretically continuous range between the two chosen pictures. By providing a continuous and gradual change to the inputs over strategically chosen start and end points, e.g. starting from a correctly classified picture and moving to a falsely classified one of the same kind, we will be guaranteed to have a change in the output. As this change will occur over minor changes within the input, it will, by definition, be an avalanche. The details on how this process is completed are given in chapter 4.

The main idea is to create a transition that will mimic how objects might move, how artifacts can be formed on the image and other conditions that might effect the quality of the frames received. We have gathered a number of different morphed images to check that they create visible pictures that can be used for the tests. Some examples of these can be found on fig. 3.1. The transitions over which the tests will be conducted will consist of discrete steps within this theoretically continuous change. The metrics that are going to be used will be discussed in section 3.3.

3.3 Detection Metrics

The following section contains the metrics that we have proposed to be able to detect the avalanches that may be occurring during a run time of a network. These metrics aim to capture the running behavior of the system in real-time to be able to detect outliers in activation patterns. While a neural network is running on a continues feed of images, the networks are to be measured

using these metrics to be analyzed to aid detection. These metrics are the current selection that we have come up with, as the project progresses further there might be additions to this list.

3.3.1 Neuron Activation Counts (NAC)

Neuron Activation Counts (NAC) are the most basic and initial point of examination for our system. This metric is defined by the number of neurons that are activated, or in other words have an activation value above a set threshold, which was taken as 0.5 for our experiments. This measurement is taken separately for each layer, that is we obtain a list of NACs for each individual layer. The formal definition is as follows:

$$NAC_n = \sum_{i=1}^{n_w} \begin{cases} 1, & n_i \geq \text{threshold} \\ 0, & n_i < \text{threshold} \end{cases}, \qquad n \in I : 1 < n \leq N_{Layers} \tag{3.1}$$

Where N_{layers} is the number of layers in the system, NAC_n is the NAC value of layer n, n_w is the width of the layer n and n_i is the activation value of the neuron i of layer n. Additions upon this metric could be made by calculating the slope of the resulting graph at points and examining the change in values rather than the values themselves.

3.3.2 Gradient (Grad)

A gradient, to simply put is a vector of partial derivatives of a function (in this case our NN) with respect to its input variables. By calculating the gradient of our system, we get the contributions of our inputs to our system. The implementation of the gradients are taken directly from the PyTorch library, where each backward [1] propagation operation returns us a list of grad [3] values for our output.

3.3.3 Eigen Value of Activations (EV)

Eigen Values (EV) are a set of values explaining the scale of change of the values of a matrix in a linear transformation, representing how much they were stretched. EVs provide us with a unique perspective on the change of the input through the layers as it gets transformed. The implementation is taken directly out of the PyTorch library [2]. The outputs of each 2d layer is taken through this function and the data is collected.

3.4 Monitoring Methods

To try and automate the process, we will also be exploring the idea of training additional machine learning methods to be able to detect these avalanches. Our proposed method is to integrate a second classifier tool, which will be referred to as the Monitoring Model (MM), that will be gathering activation data of the neurons from the main NN and detect the occurance of any avalanches in real time. This way, the detection of avalanches can be made more adaptable and robust. Furthermore, the MM can be modified to provide a confidence value for the given system instead of doing a binary classification, which can be used as the likelihood of the system to make a mistake, which can be crucial for safety-critical applications where any security over a given baseline is highly valuable.

The usage of a second classifier will require that we collect the data of the tests that we have proposed, into a larger dataset and train this classifier on them. This data will consist of the detection metrics that we have proposed on the section 3.3.

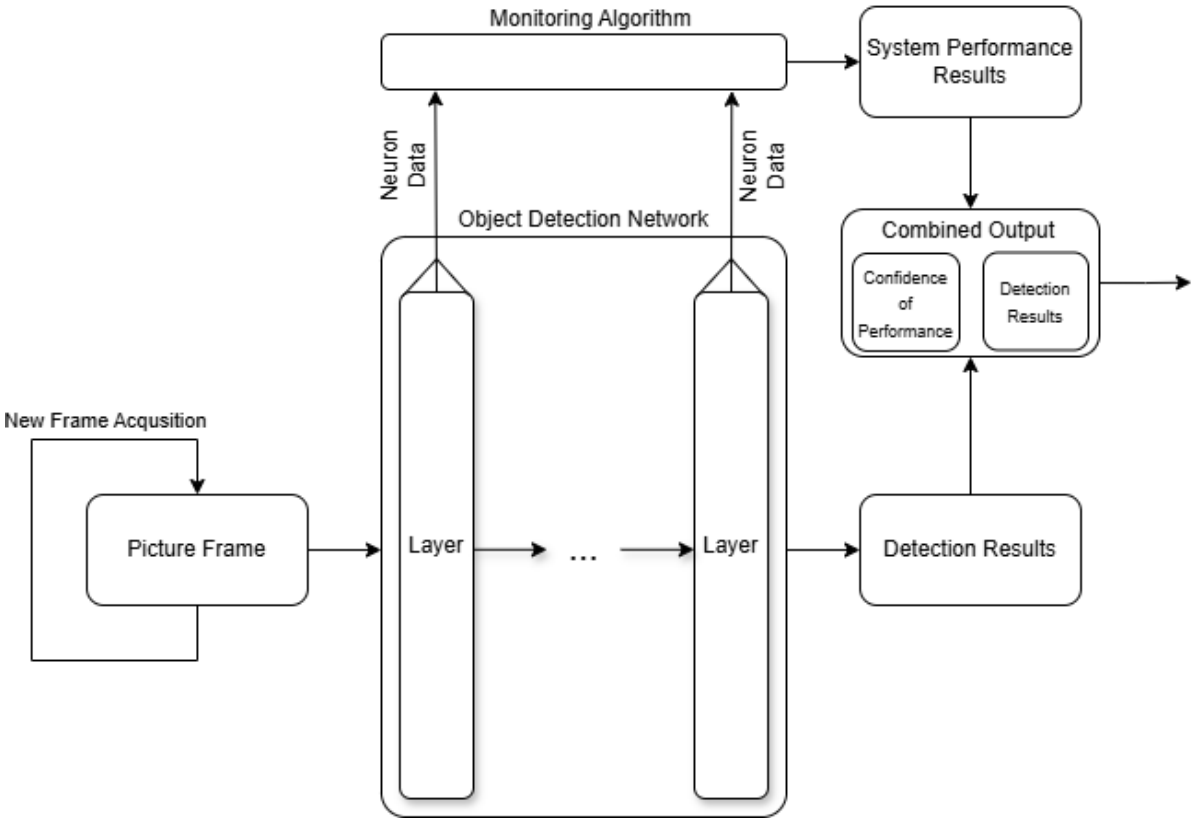

FIGURE 3.2: The architecture of the whole system with the integration of a second machine learning algorithm for real-time monitoring.

Chapter 4

Experimental Setup

To begin our efforts, we start from a basic point of creating scenarios that will mimic continues operation. For this, we propose a system where a continuous stream of pictures are fed into a neural network, with each image being visually similar to adjacent pictures on a certain transformation. We will be conducting our experiments on a data set that consists of pictures of road signs from Germany, called GTSRB. This dataset will first be used to train an image classification model to be able to determine what sign is present in given pictures. Then, we will use strategically chosen pictures to create images that will mimic real world scenarios where the quality of the images may not be optimal. The pictures obtained from this operation will be referred to as morphed images. Later, we will feed these pictures into our image classification model to cause an avalanche. As these avalanches occur, the performance metrics mentioned in 3.3 will be measured and collected to form a second dataset. Finally, we will train another machine learning method to predict avalanches only looking at these metrics. In the below sections, you can find detailed explanations of each part of this process.

4.1 Data

The data used in our work can be divided into 3 parts: the raw road sign dataset, morphed images and avalanche metrics. In the sub-sections below, each of these will be discussed in finer detail.

4.1.1 Road Sign Data

The first part, will be to directly use the GTSRB dataset as mentioned before. This dataset consists of pictures of roadsigns from Germany, and has 43 unique classes, in this case different kinds of roadsigns. The dataset is pre-processed, all the pictures are taken from a relatively direct angle (direct enough that the faces of the signs are clearly visible), has no obstructions to the signs and the images are cropped in a way where the signs are in the middle of each frame with minimal other objects around them. Each image has the dimensions 30 pixels by 30 pixels, with a total of 51.900 images.

4.1.2 Morphed Images

The second part of the data consists of pictures taken through a transformation process, referred to as morphing to generate additional, imperfect images that are designed to cause an avalanche.

The main idea behind this operation is to mimic real world scenarios, based on live video feeds. With imperfect conditions that may decrease the quality of the images received, any machine learning method might struggle to perform their tasks, varying with the severity of conditions. By using strategically selected pictures, we create these imperfect transitions that

will trigger avalanches in our system. While a direct video recording could also achieve the same result (and would be even closer to a practical, real world application scenario) it would also require a significantly higher level of effort with the data collection and pre-processing them to fit our use. Due to the resource limitations of our project, we were not able to explore this path.

The working principle is to get 2 pictures of the same kind of a road sign, one being correctly labeled by our network while the other being incorrectly labeled. We then take the correctly labeled image, and change the values of it's pixels on a gradual transformation until we eventually obtain the incorrectly labeled picture. As the pictures are of the same kind of a sign, theoretically the system should not change its prediction. As we have selected the second picture to be one that is misclassified, we guarantee that through the morphing, at a certain point the prediction will change. Since we do the morphing with incremental steps that introduce minimal change to the images, when the prediction occurs it would have occurred with minimal changes to the input while the change is not supposed to happen, fitting the definition of an avalanche.

In more formal terms, given a trained network $C_{trained}$, we take a pair of pictures $PP_n = (PP_n^1, PP_n^2)$, such that the conditions in fig. 4.1 apply.

$$\begin{aligned} &\text{(i)} && class(PP_n^1) \equiv class(PP_n^2) \\ &\text{(ii)} && C_{trained}(PP_n^1) \equiv class(PP_n^1) \\ &\text{(iii)} && C_{trained}(PP_n^2) \not\equiv class(PP_n^2) \end{aligned} \tag{4.1}$$

Then the Algorithm 1 is used to create pictures on which the experiments are to be done. Where P_{new} is the new picture that was created, H and W are the height and the width of the picture respectively, $i, j \in N$: $0 \leq i < H$ and $0 \leq j < W$, $PP_n^m[i, j]$ is the 3-dimensional value representing pixel found at the i'th row of the j'th column and the $s \in R$: $0 \leq s \leq 1$ represents a percentage of the current morphing step. Essentially, what the algorithm does is morph one image slowly into another, taking the difference of the values of each pixel in one image and the corresponding pixel on the next one, this value is then added to the first pixel with a multiplier. A multiplier of 1 would mean we get a copy of the second pixel, a multiplier of 0 would mean we get a copy of the first pixel, and any value in between will give us a hybrid value between the values of the two pixels. This operation is repeated on every single pixel of the images.

Algorithm 1 MorphImage

```
function MorphImage(PP_n, s):
for i in [0, 1, ..., N] do
    for j in [0, 1, ..., M] do
        P_new[i, j] = PP_n^1[i, j] + s * (PP_n^2[i, j] − PP_n^1[i, j])
    end for
end for
return P_new
```

The main part of the test uses the Algorithm 1 to create a range of pictures, based on the percentage steps desired, which for our experiments was decided to be increments of 0.01, starting from 0 and ending at 1. As these pictures are generated, they get run through the pre-trained CNN, while also calculating some statistics about the activation of the network and saving them. An overview of the full algorithm can be found in Algorithm 2. After the execution of the tests are done, we are left with a dataset, consisting of a list of run-time statistics of our CNN, along with the classification of the images. These results will be sorted within each run according to the percentage value of the transformations.

Algorithm 2 MorphTest

$C_{trained} \leftarrow CNN$
$PP_s \leftarrow (PP_1, PP_2, \ldots, PP_N)$
for PP_n in PP_s **do**
    for $s = 0.0, 0.01, \ldots, 1.0$ **do**
        $PP_{new} \leftarrow MorphImage(PP_n, s)$
        Classify PP_{new} with $C_{trained}$
        Calculate and save statistics
    end for
end for

4.2 Traffic Sign Detection Models

To use as our testbed, we have chosen to work with convolutional neural networks, as they have the best overall performance for image detection tasks. After looking at best practices and considering the nature of our data, we have come up with 2 different architectures that may fit our use case scenario the best. Furthermore, we have trained the larger model with different hyperparameters to test the effects of proper training on our results as well. You can find the dissection the models in fig. 4.1. Due to the framework that we have used for creating these models, supporting operations such as batch normalisation and pooling, were also considered as layers.

CNN-1 Architecture	**CNN-2 Architecture**
Convolutional Layer	Convolutional Layer
Convolutional Layer	Max Pool Layer
Max Pool Layer	Batch Norm Layer
Batch Norm Layer	
Convolutional Layer	Convolutional Layer
Convolutional Layer	Max Pool Layer
Max Pool Layer	Batch Norm Layer
Batch Norm Layer	
Flatten Layer	Flatten Layer
Linear Layer	Linear Layer
Batch Norm Layer	Linear Layer
Dropout Layer	
Linear Layer	

TABLE 4.1: Layer-by-layer comparison of the CNN architectures used for sign detection, grouped by functional clusters.

We then fed our dataset to these models for training and measured their performances, picking out the best one for use. As seen in 4.1, both of the architectures perform significantly better than the baseline of random guesses, meaning they are relatively successful at completing their tasks. However, we also see that CNN-1 outperforms CNN-2 by a significant margin. CNN-1 fits what the best practices would suggest for our dataset better, thus these result are what we had expected. Furthermore, we see that CNN-1 performs a lot better using the ADAM optimizer as opposed to the SGD optimizer. The exact implementations of these optimizers are irrelevant to our findings, we will only use these to examine what different extents of training has on our final results. Due to the higher success rate of CNN-1, we will perform the main part of our experiments on the two differently trained versions of this architecture. The results in this figure will be used as a comparison point in the final part of our study. For the rest of this paper, CNN-1 trained using ADAM will be referred to as the CNN-A and CNN-1 trained with SGD will be referred to as the CNN-S.

Model Name		Accuracy	Epochs
CNN-1	ADAM	99.60%	30
	SGD	67.07%	
CNN-2	ADAM	46.77%	
Theoretical Random Guess		02.33%	-

FIGURE 4.1: Performance metrics for sign detection models and training hyperparameters, compared with the accuracy that would be expected from trying to randomly guess the results.

4.3 Avalanche Detection Models

In order to detect avalanches, we once again resort to machine learning models. Due to the unstructured nature of our avalanche data, where the order of features does not matter, we were able to use a broader range of models to achieve our results. Due to their robust nature and low cost of operating and training, our first choice was a RF network. As we have gotten satisfactory results from them, we did not explore any additional methods. For comparison's sake, a small neural network was also tested. The performance results and hyperparameter selections of these models can be found in 5.

Chapter 5

Results

5.1 Avalanche Detection

As the result of our morph tests, we obtain the avalanche datasets for both of our networks with 47 cases, totaling 4700 pictures and 50 cases, totaling 5000 pictures for CNN-A and CNN-S respectively. We will feed these into 2 separate RFs, which will be referred to as the RF-A for the one using the CNN-A data, and RF-S for the one using the CNN-S data. For clarification, at no point in our experiments will these models ever use the data from the other CNN; they will be trained and tested within their own CNN's data exclusively. As these pictures are fed into our systems, we collect the metrics that are mentioned in the section 3.3, giving us 3 points of metrics for each layer, given there are 13 layers in our neural networks we get a total of 39 recorded metrics per test.

First point of our results will consist of feeding these data into our RF models. To ensure the quality of our results, we will test the RF with 10 different combinations of hyper-parameters. In our case, the only hyper-parameters that are altered will be the number of estimators, which controls how many individual decision trees are present in the RFs. The performance metrics of the RFs trained on the different CNN's datas can be found in fig. 5.1. Having an initial look at this figure, we can see that the performance metrics are all in very desirable ranges, meaning our models perform quite well on predicting avalanches. Furthermore, we can see that both models performed in a similar manner, with the model that uses the data of CNN-S performing marginally better. This indicates that the formation characteristics of avalanches are similar in the two different training scenarios. The slight performance increase on the model CNN-S, which is the model that has worse training, may be caused by it having a less robust internal structure, resulting in it having more and bigger avalanches, making it easier for them to be detected by our models. This explanation can further be supported by the fact that it had produced more avalanches within the data, where the CNN-S produced 50 and the CNN-A produced 6% less avalanches, with a count of 47.

Looking at the individual performance metrics, we see that both the models have very high accuracy and precision values. This is an indicator for the systems acting relatively on the safer side for their guesses. As the recall values are not as high as the accuracy and precision, we can say that the system has a bias to not label positive classes when it is not perfectly sure. Looking at the F1 score as well, we see that the tradeoffs taken by the models are justifiable as they do not sacrifice any of these values in a significant way in order to maximize others, meaning a balanced approach.

As we compare the metrics along the X axis of the plots, which represents the number of estimators used on the tests, we can see that there exists a sweet spot that yields the best results. This also serves as a sign of overfitting for higher numbers of estimators. However, not all of the graphs include the same trends, indicating that the performance of the systems change on different aspects of their decision making as this hyper-parameters is changed. One

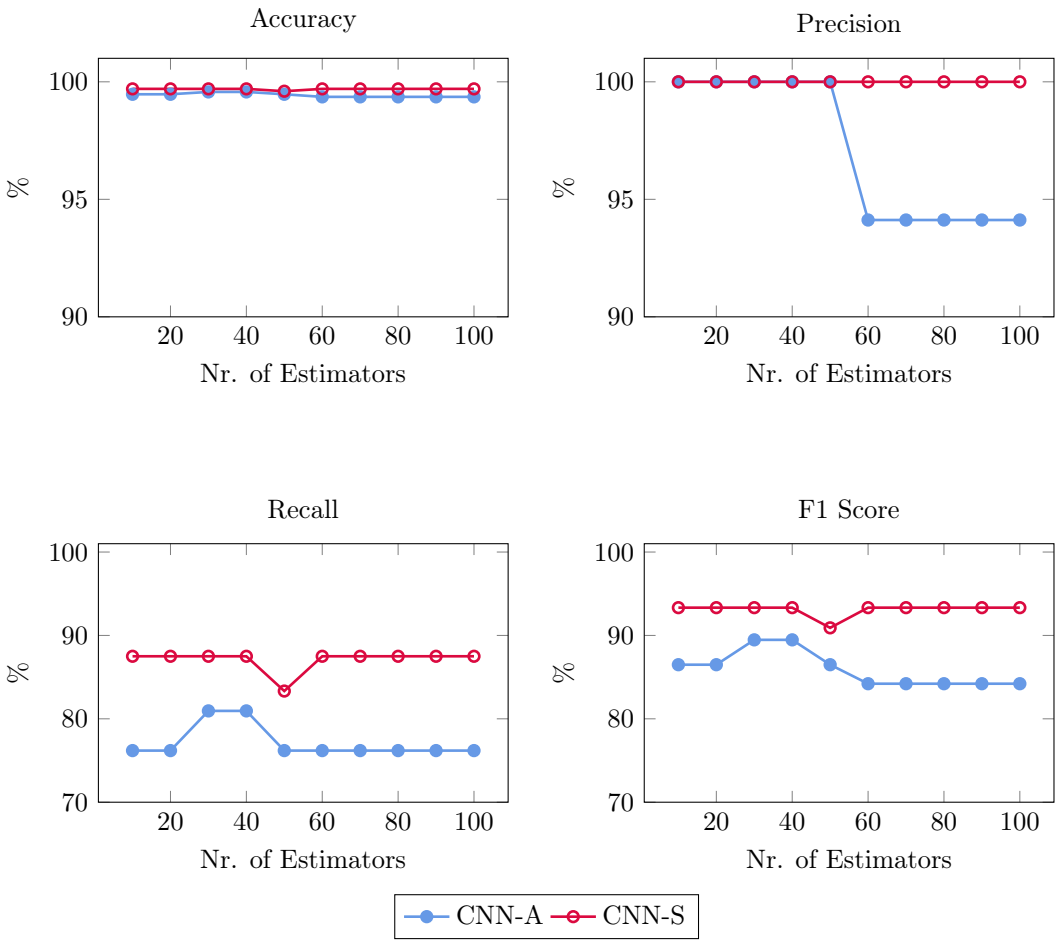

FIGURE 5.1: Performance metrics of CNN-A and CNN-S across different numbers of estimators.

positive relation between two models is that the same sweet spot produces the best results on both the models, which is in the 30-40 range of number of estimators. This also helps reduce the training and operations costs as this number is quite low in terms of practical capabilities and cost requirements of real-life scenarios.

Since our results are looking to be in very ideal ranges, we also opted in to verify them using k-fold cross validation, which is a simple algorithm used to split the dataset in to k different groups to make sure that we are not taking advantage of how we split the data into training and tests set. Using cross validation, we train our models on all the groups of the data except a single one, where the testing takes place. We do this for all the different groups and average the results to find the performance of our system. We have chosen to use a version of cross validation referred to as the stratified cross validation, which ensures that the two classes in our data are evenly split among all the groups. The results of the tests can be found in fig. 5.2. The results here show us that, indeed our systems have had very little effects of overfitting present in their training. With the CNN-A based model, RF-A having the most impact on its results, showing a 5% decrease in its F1 value. However, it is still in $> 85\%$ range, making it a good classifier.

As an added benefit of the RF models, we also get a chance to look at their feature importance's, which are metrics that tell us the usefulness of any feature on the model's decision making. These values are measured in mean difference in impurity (MDI), which is a metric that calculates what is the average effect of this feature on the impurity, or the separation of different classes on the input. Looking at the figures 5.3 and 5.4, we can see that the feature

Model Used	Testing Method	Accuracy	Recall	Precision	F1
RF-A	Cross Validation	99.38%	77.12%	96.78%	85.25%
	Regular Split	99.57%	80.95%	100.00%	89.47%
RF-S	Cross Validation	99.82%	93.56%	96.89%	95.08%
	Regular Split	99.60%	91.67%	99.80%	95.35%

FIGURE 5.2: Comparison for the performance of metrics of avalanche detection models on a single split validation and a k-fold cross validation scenarios, with k chosen as 10.

importance's are not balanced across all the features, indicating the redundancy between some of them. Another point that is immediately visible is the amount of features that have no effect on the decision process. As the features that have no effect are the same one across the two models, we can say that the formation of avalanches are heavily effected by certain layers. The overall magnitude of the importance's of the features that were used are very similar across both models, except for the increase on the data of the the later layers. This could be attributed to the fact that the later layers on the NNs are flat layers. Since the first layers are convolutional layers, they are much bigger in terms of neuron counts and are also more advanced in the sense that they utilize extra masks and operations to process their data. This would make them require more training, in which the models that haven't received proper training will get the most decrease in performance in relation to ones with better training. As such, the final layers may need to taken on heavier work loads to boost the system's performance.

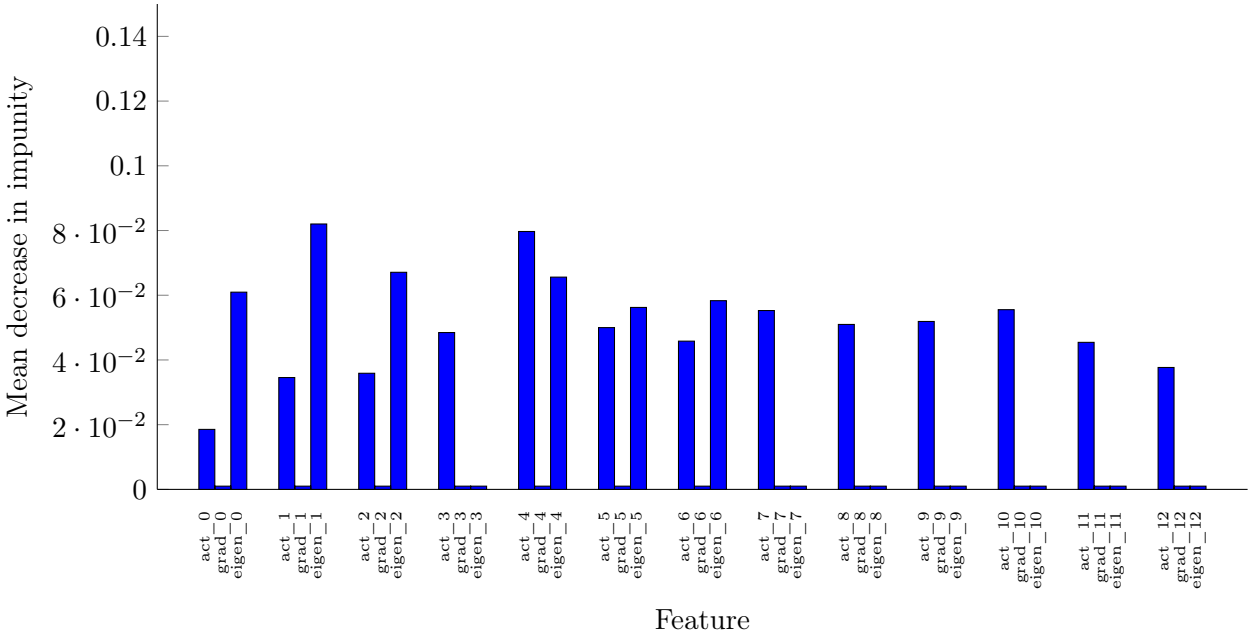

FIGURE 5.3: Feature importance of CNN-A using mean decrease in impurity (MDI).

Furthermore, we can also use the feature importance's to decrease the amount of data that would be necessary to achieve our results. As seen on figures 5.3 and 5.4, there are 20 features that are unused in the decision making process on both of the models. Having a total of 39 features originally, if these were to be removed, we would have decreased the input size by 51% without effecting the performance of the models. We can also see that no features that use the gradient values had an effect on the performance of the system. Removing the collection of these values could further decrease the operating costs of our models, as less data would be needed to

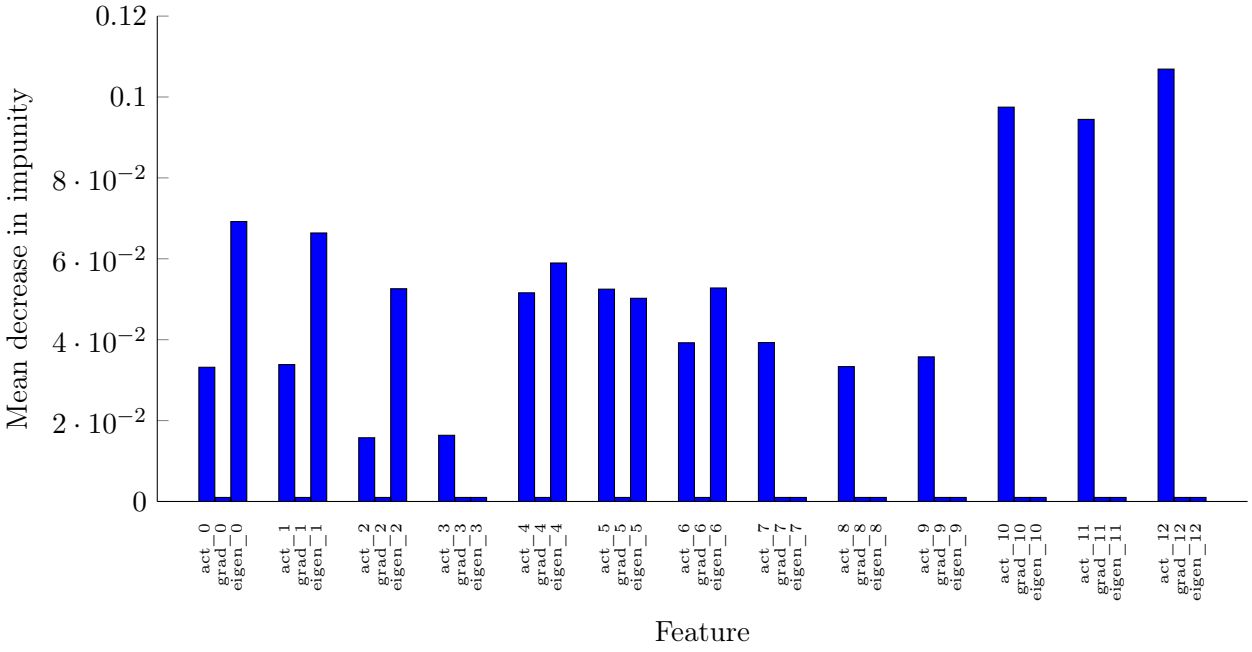

FIGURE 5.4: Feature importance of CNN-S using mean decrease in impurity (MDI).

be gathered from the running NNs. The lack of importance of gradients can be attributed to their redundancy as they represent a similar quality to eigenvalues in theory.

To have a point to compare, we also have trained a simple neural network to detect avalanches. The network that we have used is extremely basic, having only 3 layers of 39, 13 and 1 neurons; that are fully connected with no other support operations. By feeding the same data that was fed to the RFs, we obtain the results shown in 5.5. By looking at the differences in the results, we can see that NNs are significantly outperformed by RFs. Also considering their heavier training and testing costs, makes them an inferior choice to RFs. These results are what we would expect, as neural networks would require more data to be trained. With further work and more specialized architectures, such as types of neural networks that are built to remember the past inputs in limited fashions, they can perform better. However, with the high success rate we have achieved from RFs, makes NNs the worse choice, as the specialized networks would only make the operating costs higher.

Data Provider	Model	Accuracy	Recall	Precision	F1
CNN-A	RF	99.57%	80.95%	100.00%	89.47%
	NN	92.34%	46.99%	37.74%	39.40%
CNN-S	RF	99.70%	87.50%	100.00%	93.33%
	NN	60.00%	48.94%	34.84%	27.96%

FIGURE 5.5: Comparison for the performance of RFs and NNs on avalanche detection. For the representation of RFs, the hyper-parameters with the best performance were chosen.

Chapter 6

Limitations & Future Work

Due to the time and resource limitations of our project, we were not able to completely explore every aspect of our topic. We would like to mention these in this chapter for future work and to provide caution while using our findings. Below you will find, in no particular order, a short chapter for each area of our work we think can be improved further.

6.1 Avalanche Definition For NNs

The topic of avalanches in neural circuits, or other but similar domains have mainly been researched outside of the practical areas of computers science, we struggled to find work that would be directly applicable. Most of the literature has focused on working physical, chemical or biological systems that lack many of the limitations that we have in the neural networks that we use in machine learning. One of the biggest differences that we have faced is the fact that most of the systems that have been researched had a looping nature, where an output had the chance to go over the system multiple times until it exhausted itself, or the bounds of the system was bigger than the extent of the avalanches that would be formed. In our case, where our networks propagate information in a single direction and are relatively limited by their size, we were unable to use the same metrics and identification criteria for avalanches.

6.2 Video Data vs Morph Image Generation

The main motivation behind using morph image generation is to mimic the continuous inputs of a video feed. We want to analyze practical scenarios that may come across a neural network in a real world setting. Unfortunately, we were not able to find any datasets that consisted of video frames that have been labeled. In order to create our own, we would need a vehicle, a decent drive-length and a camera to record the road signs as they appear to the car and hand label them, which would have been too much for the resources that were available to us. Furthermore, the concept of Morph Image Generation can be taken a step further by using a masking process to identify the objects within the images and shift them in a way that would make more visual sense, instead of individual pixel manipulations that we have done. This could act like a middle ground between a video feed and image generation.

6.3 Tests with DNNs

Due to the definition of avalanches, their behavior becomes much more significant and pronounced when they have space to grow their sizes. Unfortunately, the models that would work for our dataset were too small to allow the avalanches to grow to their potential maximum sizes. Even though this was not a major issue as they still had the chance to form, it may be easier

to characterize them if they had bigger models to form in. A test setup where the dataset used could allow for larger networks would have the potential to shine more light on avalanches.

6.4 Examining different kinds of NNs

Similar to the previous point, we had to confine to CNNs as our testbeds due to the nature of the dataset we have used. However, this leaves some uncertainties regarding the generalization of our results to NNs in general. As we have no way of knowing if our results are specific to CNNs, we can not confidently comment on the applicability of our finding on different kinds of neural networks.

6.5 Pre-Trained Networks

For our work, we only focused on networks that we have trained ourselves on our datasets specifically. In real world, due to a variety of reasons such as generalization, costs and reliability, some individuals might prefer to use a pre-trained model. As our models are only capable of identifying certain kinds pictures, we miss out on observing how a broader knowledge might effect the performance of the avalanche detection methods.

Chapter 7

Conclusion

We have tested two convolutional neural network models, that were trained to classify images of road signs, to see if we could predict when they would have disruptive firing patterns occur within them, in the form of avalanches. By coming up with a list of metrics and training an additional machine learning model on them, we have created a pipeline to predict the formation of avalanches. Our experiments showed that by using random forests, we were able to predict avalanches with very high levels of success. Furthermore, comparing the two convolutional neural networks that we have used, we were able to see that a better trained system has less avalanches overall and makes it slightly harder to detected them. We have also conducted our experiments in a limited capacity with another set of a neural network for predicting avalanches, which did not perform up to standards of the random forests. The success we have achieved with the random forest models will enable us to improve the performance of neural networks by having this additional machine learning model that would monitor a given neural network and provide feedback on the reliability of the outputs it gives out.

Bibliography

[1] Pytorch backward, March 2024. URL: https://pytorch.org/docs/stable/generated/torch.Tensor.backward.html#torch.Tensor.backward.

[2] Pytorch eigen values, March 2024. URL: https://pytorch.org/docs/stable/generated/torch.linalg.eig.html.

[3] Pytorch grad, March 2024. URL: https://pytorch.org/docs/stable/generated/torch.Tensor.grad.html.

[4] Stephanie Abrecht, Alexander Hirsch, Shervin Raafatnia, and Matthias Woehrle. Deep learning safety concerns in automated driving perception, 2024. URL: https://arxiv.org/abs/2309.03774, arXiv:2309.03774.

[5] Barnawi A Almutairi S. Securing dnn for smart vehicles: an overview of adversarial attacks, defenses, and frameworks. *Journal of Engineering and Applied Science*, 70, March 2023. doi:10.1186/s44147-023-00184-x.

[6] Humaidi A.J. Alzubaidi L., Zhang J. Review of deep learning: concepts, cnn architectures, challenges, applications, future directions. *Big Data*, 8, March 2021. doi:10.1186/s40537-021-00444-8.

[7] Markus Aschwanden. Self-Organized Criticality in Astrophysics. *Springer Berlin Heidelberg*, January 2011. doi:10.1007/978-3-642-15001-2.

[8] John M. Beggs and Dietmar Plenz. Neuronal avalanches in neocortical circuits. *Journal of Neuroscience*, 23(35):11167–11177, 2003. URL: https://www.jneurosci.org/content/23/35/11167, arXiv:https://www.jneurosci.org/content/23/35/11167.full.pdf, doi:10.1523/JNEUROSCI.23-35-11167.2003.

[9] Matthew Browne and Saeed Ghidary. Convolutional neural networks for image processing: An application in robot vision. In *Advances in Artificial Intelligence*, pages 641–652, 12 2003. doi:10.1007/978-3-540-24581-0_55.

[10] Daniel Canedo and Alexandre Romariz. Data analysis of wireless networks using computational intelligence. *Journal of Communications*, 13:618–626, 11 2018. doi:10.12720/jcm.13.11.618-626.

[11] Barry A. Cipra. An introduction to the ising model. *American Mathematical Monthly*, 94:937–959, 1987. URL: https://api.semanticscholar.org/CorpusID:10158214.

[12] Aaron Clauset, Cosma Rohilla Shalizi, and M. E. J. Newman. Power-law distributions in empirical data. *SIAM Review*, 51(4):661–703, November 2009. URL: http://dx.doi.org/10.1137/070710111, doi:10.1137/070710111.

[13] Kreft M. et al. Cramer B., Stöckel D. Control of criticality and computation in spiking neuromorphic networks with plasticity. *Nature Communications*, 11, june 2020. doi:10.1038/s41467-020-16548-3.

[14] Cyril Goutte and Eric Gaussier. A probabilistic interpretation of precision, recall and f-score, with implication for evaluation. In David E. Losada and Juan M. Fernández-Luna, editors, *Advances in Information Retrieval*, pages 345–359, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1511.08458*, June 2016. URL: https://arxiv.org/abs/1511.08458.

[16] Evelyn Herberg. Lecture notes: Neural network architectures, 2023. URL: https://arxiv.org/abs/2304.05133, arXiv:2304.05133.

[17] A. C. Hindmarsh, P. N. Brown, K. E. Grant, S. L. Lee, R. Serban, D. E. Shumaker, and C. S. Woodward. Sundials: Suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software*, 31(3):363–396, September 2005. doi:10.1145/1089014.1089020.

[18] P. C. Hohenberg and B. I. Halperin. Theory of dynamic critical phenomena. *Rev. Mod. Phys.*, 49:435–479, Jul 1977. URL: https://link.aps.org/doi/10.1103/RevModPhys.49.435, doi:10.1103/RevModPhys.49.435.

[19] Eugene M. Izhikevich. Which model to use for cortical spiking neurons? *IEEE Transactions on Neural Networks*, 15(5):1063–1070, September 2004. doi:10.1109/TSMCC.2004.843247.

[20] Eugene M. Izhikevich. Polychronization: Computation with spikes. *IEEE Circuits and Systems Magazine*, 6(3):24–36, Third Quarter 2006. doi:10.1109/MCAS.2006.1688199.

[21] Simoens Pieter Khaluf Yara, Ferrante Eliseo and Huepe Cristián. Scale invariance in natural and artificial collective systems: a review. *J. R. Soc. Interface.*, 2017. doi:http://doi.org/10.1098/rsif.2017.0662.

[22] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Use of artificial neural network in pattern recognition. *Nature Journal*, June 2015. doi:10.1038/nature14539.

[23] K. Linkenkaer-Hansen, V. V. Nikouline, J. M. Palva, and R. J. Ilmoniemi. Long-range temporal correlations and scaling behavior in human brain oscillations. *J. Neurosci.*, 21(4), February 2001. doi:10.1523/jneurosci.21-04-01370.2001.10.

[24] Dimitrije Marković and Claudius Gros. Power laws and self-organized criticality in theory and nature. *Physics Reports*, 536(2):41–74, 2014. Power laws and Self-Organized Criticality in Theory and Nature. URL: https://www.sciencedirect.com/science/article/pii/S0370157313004298, doi:10.1016/j.physrep.2013.11.002.

[25] Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. Methods for interpreting and understanding deep neural networks. *Digital Signal Processing*, 73:1–15, February 2018. URL: http://dx.doi.org/10.1016/j.dsp.2017.10.011, doi:10.1016/j.dsp.2017.10.011.

[26] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, March 1986. doi:10.1007/BF00116251.

[27] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier, 2016. URL: https://arxiv.org/abs/1602.04938, arXiv:1602.04938.

[28] Hinton G. Williams R. Rumelhart D. Learning representations by back-propagating errors. *Nature*, 323:533 – 536, October 1986. doi:10.1038/323533a0.

[29] Saptarshi Sengupta, Sanchita Basak, Pallabi Saikia, Sayak Paul, Vasilios Tsalavoutis, Frederick Atiah, Vadlamani Ravi, and Alan Peters. A review of deep learning with special emphasis on architectures, applications and recent trends. *Knowledge-Based Systems*, 194:105596, 2020. URL: https://www.sciencedirect.com/science/article/pii/S095070512030071X, doi:10.1016/j.knosys.2020.105596.

[30] K Sneppen, P Bak, H Flyvbjerg, and M H Jensen. Evolution as a self-organized critical phenomenon. *Proc. Natl. Acad. Sci.*, 92(11), May 1995. doi:10.1073/pnas.92.11.5209.

[31] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[32] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, June 2017. URL: https://arxiv.org/abs/1706.03762.

[33] Guangyu Robert Yang, Madhura R. Joglekar, H. Francis Song, William T. Newsome, and Xiao-Jing Wang. Task representations in neural networks trained to perform many cognitive tasks. *Neural Computation*, 31(12):3415–3449, December 2019. doi:10.1162/neco_a_00990.

[34] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. *Dive into Deep Learning*. Cambridge University Press, 2021. https://d2l.ai/chapter_convolutional-neural-networks/padding-and-strides.html.

[35] Yong Zhang, Rong Jin, and Zhi-Hua Zhou. Understanding bag-of-words model: A statistical framework. *Information Processing Management*, 45(5):556–574, September 2009. doi:10.1016/j.ipm.2009.03.002.