Safety assertions in neural network classification

https://github.com/ericbill21/siemens

Eric Tillmann Bill[*]
RWTH Aachen University
Aachen, Germany
eric.bill@rwth-aachen.de

Robert Maximilian Giesler[*]
RWTH Aachen University
Aachen, Germany
robert.giesler@rwth-aachen.de

Felix Maximilian Knispel[*]
RWTH Aachen University
Aachen, Germany
felix.knispel@rwth-aachen.de

***Abstract*—Dependability and reliability of machine learning classification systems are getting increasingly important as machine learning techniques are being applied in more and more domains, including safety-critical areas. The task of the *Siemens AI Dependability Assessment Student's Challenge* is to provide a classifier for a given classification task as well as to determine an as-accurate-as-possible misclassification probability estimate.**

In this paper, we describe a machine learning model that has been optimized for the safest classification possible. A multilayer feed-forward neural network was trained on the three given datasets. We introduce a custom loss function to reduce the number of safety-critical misclassifications. Additionally, we detect predictions that are less certain than a determined threshold value and decide to resort to a safer classification in these instances.

Moreover, we describe the approach taken to determine a probability estimate for misclassifications: As training data quality is crucial to model performance and dependability, we develop several interpretations of training data and use these as indicators to estimate model accuracy.

For the three given datasets, our approach provides reliable (though not guaranteed) misclassification rates no larger than 15.23%, 8.63%, and 5.45%, respectively.

I. Introduction

A. Overview

Siemens Mobility's *AI Dependability Assessment Student's Challenge* consists of two main tasks:

1) developing a machine learning algorithm for a simple binary two-dimensional classification problem and, more importantly,
2) providing justifiable safety assertions for the decisions made by the algorithm and ideally presenting a reliable upper bound for the misclassification probability.

The motivation for this challenge stems from the controversy surrounding the deployment of machine learning algorithms in areas such as autonomous driving or unmanned aerial vehicles, where decisions critical to human safety must be made. The calculations made by machine learning algorithms are usually highly complex and unintuitive to human reasoning, which often leads to hesitance when applying machine learning solutions in safety-critical fields. It is important that we can provide justifiable safety assertions for the decisions made by machine learning algorithms and that we can sufficiently limit the probability of misclassification before implementing them in high-risk applications.

Potential safety-critical applications for machine learning algorithms which come to mind are often highly complicated and require the processing of multi-dimensional data. Although the complexity of such problems goes far beyond that of our two-dimensional classification problem, it is still meaningful to view the tasks at hand in this very basic context. For one, we cannot hope to solve these tasks for high-dimensional, safety-critical real-world examples if we cannot solve them for simple problems such as the one given here. Furthermore, solutions found here may be scaled up to prove effective for more complicated applications.

B. The Challenge

The challenge is defined by the following game:

"Player A (the Assessor) chooses a subset S of the unit square $I = [0, 1]^2$ and a probability distribution P on I.

Then A chooses a sample size n and generates n random variates x_i from P. Additionally labels I_i are assigned: $I_i = 1$ (red), if x_i is an element of S, and 0 (green), else – thus, the Assessor uses an indicator function I_S.

Player E (the Safety Expert) now has the task to provide a machine learning algorithm and safety arguments for this classification problem, including all assumptions and, possibly, safety-related application rules. In particular, she must provide a (non-trivial) upper bound for the probability, that the next random variate x_{n+1} is misclassified.

If a red data point is labeled green, then this decision is safety-critical, and an accident may happen. If a green data point is labeled red, then this may cause or some cost, but not directly a safety problem. Take traffic lights as an example…"

We as the challenge participants take the role of Player E while Siemens Mobility takes the role of Player A.

C. The Datasets

Siemens Mobility provides three datasets (A, B, and C) of varying sample size n, probability distribution P, and complexity of subset S, which each represent an independent instance of the proposed game. Going forward, we will refer to the attributes n, P, and S of a specific dataset X as n_X, P_X, and S_X. The datasets are provided as xls files containing the coordinates and the color labels (0 or 1) of all points, as can

[*] Equal contribution

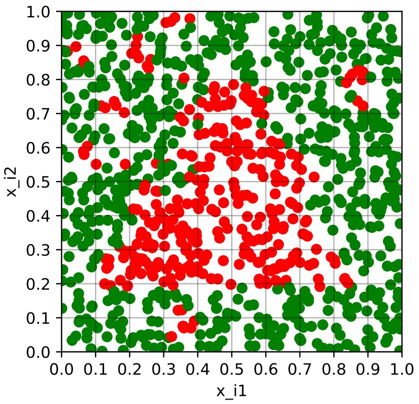


Fig. 1: Dataset A.

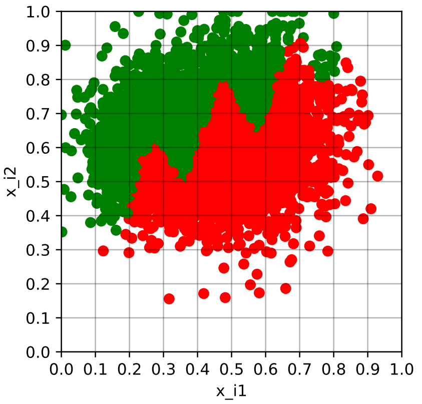


Fig. 2: Dataset B.

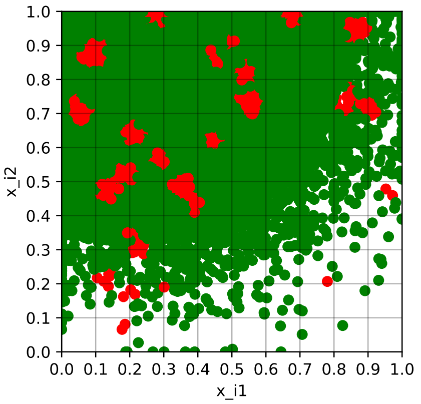
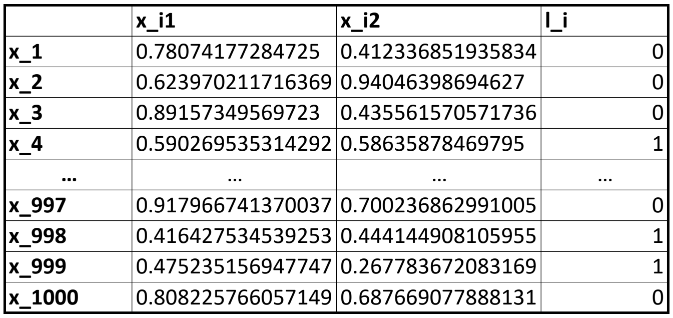


Fig. 3: Dataset C.



	x_i1	x_i2	l_i
x_1	0.78074177284725	0.412336851935834	0
x_2	0.623970211716369	0.94046398694627	0
x_3	0.89157349569723	0.435561570571736	0
x_4	0.590269535314292	0.58635878469795	1
...	...	...	...
x_997	0.917966741370037	0.700236862991005	0
x_998	0.416427534539253	0.444144908105955	1
x_999	0.475235156947747	0.267783672083169	1
x_1000	0.808225766057149	0.687669077888131	0

TABLE I: Extract of dataset A.

be seen for the example of dataset A in Table I. As the points of each dataset were randomly generated from a probability distribution P on the unit square $I = [0, 1]^2$, the coordinates of the given points take real number values on a continuous scale between 0 and 1. There is a significant discrepancy between the sizes of the datasets: dataset A consists of 1000 points, dataset B of 5000 points, and dataset C of 50,000 points.

A substantial benefit of working with two-dimensional data is that it can be easily visualized and that it is easy to grasp and interpret for human readers, which is not usually the case when working with higher dimensional data. Visualizations of the three given datasets are found in Figure 1, Figure 2, and Figure 3.

D. Paper structure

For the most part, the structure of this paper chronologically follows the line of approach we took in tackling this challenge. Section II focuses on the architecture of the machine learning model we designed, which acts as the base of our solution. We begin with an overview of the structure of our model and the process of finding suitable hyperparameters. A series of methods to specifically reduce the misclassification of red points are introduced later in section II.

In section III, we concentrate on the task of providing an estimate for the misclassification probability of our machine learning algorithm. We address the question of whether it is possible to prove that misclassification rates can be limited, and we explain the process of reaching reliable misclassification estimates on all datasets. Section III closes with suggestions on how statistical analysis of misclassification patterns may be used to further reduce the misclassification upper bounds in practice.

In section IV, we explain how our methods and approaches may be scaled to higher dimensions in order to accommodate complex real-world applications. We conclude in section V.

II. Classification model

In this section, we go into detail about the machine learning algorithm we designed for the classification of the points in the given datasets. We especially focus on the methods applied to reduce the misclassification probability of red points, which, according to the challenge description, is substantially more critical that the misclassification probability of green points.

Motivated by the widespread success of artificial neural networks (ANNs) in classification tasks over the past decade (e.g. [1]–[3]) and previous experience of our team with feedforward ANNs, we decided to employ a feedforward artificial neural network as the classifier for this challenge.

We used GitHub and Google Colaboratory to share and jointly develop the code for our machine learning solution. All code for data analysis, data operations, visualizations, and the neural network itself was written in Python 3.7.10 in a Jupyter Notebook. The TensorFlow and Keras libraries were used for the implementation of the ANN. The complete code for our solution can be found on GitHub [4]. Due to the limited computational power of the personal computers available to us, we additionally used Google Colaboratory Pro's and Datalore Pro's cloud computing services for many of our computationally intensive calculations.

A. Model Architecture

When constructing an artificial neural network for a classification task, several factors must be taken into consideration. The depth and width (number of layers and number of neurons per layer) of the network must be chosen such that the classification function separating the different classes of the given dataset may be adequately estimated by the network.

Generally speaking, increasing the depth and the width of a neural network is a straightforward way of increasing its classification performance. Larger networks, however, tend to face a higher risk of overfitting, especially when working with limited amounts of labeled training data [2]. This was especially important to keep in mind, as we were developing an ANN tasked with delivering promising classification results on three different datasets with vastly dissimilar amounts of labeled training points.

The neural network we designed for the proposed classification task consists of an input layer, four densely connected hidden layers, and a densely connected output layer. The number of neurons per layer, in order of input layer to output layer, are as follows:

$$2, 100, 70, 50, 10, 2$$

We use ReLU as the nonlinear activation function in all hidden layers. ReLU neurons are non-saturating and therefore substantially increase training speeds compared to saturating nonlinearities such as the tanh function [1], [5]. Our network receives the x_{i1} and x_{i2} coordinates of a datapoint as input and produces two output values. As is common practice in neural network classifiers, a softmax function is applied to the output to ensure that the output values sum up to 1. The two output values of the network represent the network's calculated probability that the input point is red, and the probability that the input point is green, respectively, as values between 0 and 1.

We found that our neural network architecture delivers impressive classification accuracy on all three datasets without running into noticeable issues with overfitting.

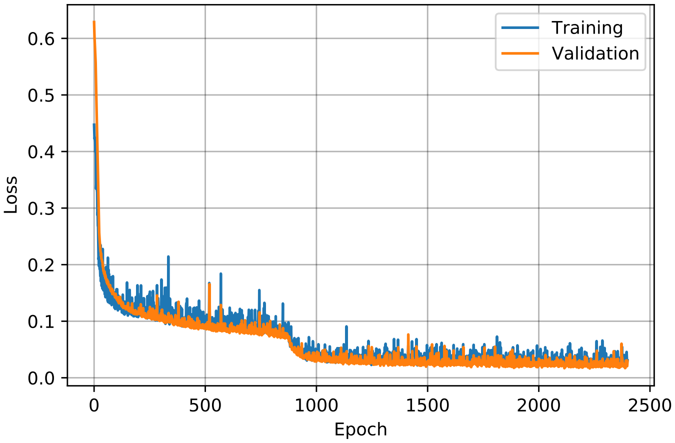


Fig. 4: Training and validation loss during training on dataset C.

Since no validation sets are provided for this challenge, we randomly extract 15-16% of the training points from each dataset before training (15% for A, 16% for B and C). Thereby we can train our model using 84-85% of the provided data and validate its accuracy using the remaining 15-16% of unseen points. Figure 4 shows the validation and training loss of an example training (fitting) run of our network on dataset C with respect to training epoch. It is evident that both training and validation loss follow a downward trend as the training process advances. Overfitting, which would be indicated by an upward trend in validation loss at some stage during training, clearly does not occur. Analogous plots for example training runs on datasets A and B can be found in appendix B.

It is important to note that one trained instance of the network cannot possibly be used to accurately classify all three datasets. Instead, three instances of the model must be created and trained on one of the datasets each, leaving us with three separate classifiers.

Given that the main task of the challenge is not to provide a neural network architecture which is as accurate or efficient as possible, we spent comparatively little time optimizing and fine tuning the architecture of our model. Rigorously testing and optimizing both the performance and the efficiency of the neural network architecture may therefore further improve the results presented here, but lies beyond the scope of this project.

B. *Training-related hyperparameters*

After having defined the architecture of our neural network, we will now look at the training-related hyperparameters used for fitting our model.

Instead of using a classical stochastic gradient descent approach, we decided to use the Adaptive Moment Estimation (Adam) optimizer [6], which combines the two stochastic gradient descent algorithms of Adaptive Gradients (AdaGrad) and Root Mean Square Propagation (RMSProp). Due to its computational efficiency and often favorable performance compared to other stochastic optimization methods, Adam is currently recommended as the default stochastic gradient descent algorithm [5]. We found that using Adam's default parameters of $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$ delivers good results when training our network on each of the three datasets.

The batch size (number of training samples propagated through in one forward pass) and the number of epochs (number of times the whole training set is shown to the network) have a strong influence on the classification performance of the network. To reduce the misclassification probability of our model as far as possible, we have to determine the optimal combination of batch size and number of epochs for training on each dataset. The optimum values of these parameters will vary from dataset to dataset due to the differences in the number of training samples and the complexity of the subsets S_A, S_B, and S_C.

We implemented a function named *averageEpochsBatchSize* to calculate the optimum batch size and number of epochs for each dataset. For a given dataset, *averageEpochsBatchSize* receives a list of possible batch sizes B, a list of possible epoch numbers E, and a number of iterations $n \in \mathbb{N}$. For every iteration $0 \leq i < n$, we extract a random balanced validation set from the given dataset and train our model once for each possible combination of batch size and epoch number $\{(b, e) \mid b \in B, e \in E\}$ using the remaining training points. Before every training run, the weights are reset to an initial value defined by the Glorot initialization algorithm. We then let our model classify the validation points to calculate the percentage of red points, the percentage of green points, and the total percentage of points from the validation set misclassified. After n iterations of extracting a random balanced validation set, training the model using all possible batch size and epoch number combinations, and classifying the validation points, the average red, green, and total misclassification percentages for every combination (b, e) are calculated and plotted in a three-dimensional space. Figure 5 shows the percentage of red points misclassified with respect to batch size and epoch number on dataset B, averaged over 50 training and validation runs for each (b, e) combination.

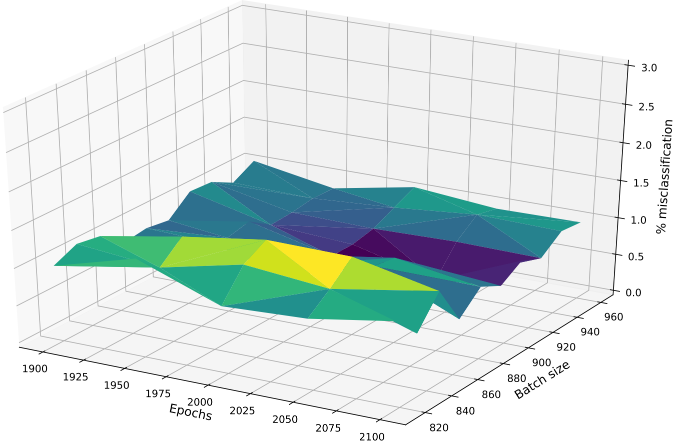


Fig. 5: Average red % misclassification with respect to batch size and epoch number on dataset B.
Parameters: $n = 50$, $B = \{816, \ldots, 960\}$ in increments of 16, $E = \{1900, \ldots, 2100\}$ in increments of 50.

Note that for this optimization procedure, we chose *balanced* validation sets, meaning that the points for the validation sets were randomly selected under the condition that each color must make up at least 40% of the validation points. In doing so, we avoid the risk of one color being underrepresented by chance, which could have resulted in a bias in our optimum batch sizes and epoch numbers.

Given the information that the misclassification of red points is severely more critical than the misclassification of green points, the optimum combination of batch size and epoch number is one which results in minimum average red percentage misclassification while maintaining acceptable green percentage misclassification.

Following this method, we conclude the following optimum batch sizes and epoch numbers for training our model on the three given datasets:

	batch size	epoch number
Dataset A	16	1450
Dataset B	912	2050
Dataset C	2000	2400

Due to our limited computational resources, we had to run *averageEpochBatchSize* multiple times for each dataset, slowly narrowing down the optimal points by first using large ranges of batch sizes and epoch numbers with large increments and later using smaller ranges with smaller increments.

Plots analogous to Figure 5 showing average total, red, and green percentage misclassification with respect to batch size and epoch number for all datasets can be found in appendix C.

C. Dataset balancing

Since the most critical part of machine learning is the learning part itself, we need to optimize training data in cases where safety-critical classes are greatly underrepresented. In the given datasets, the relations between green and red points are as follows:

Dataset A	32.60% red	67.40% green
Dataset B	52.52% red	47.48% green
Dataset C	9.59% red	90.41% green

A balanced distribution of both classes, as present in dataset B, is desirable as both classes are equally important - therefore, the network should see both classes appropriately often during training. In dataset C, for example, the safety-critical class red is heavily underrepresented. To account for such discrepancies in red or green samples, we duplicate data points of the less frequent color until both colors make up a minimum proportion of the training set defined by a threshold value $t_{Bal} \in [0, 0.5]$. During testing, we found that some threshold values lead to increased misclassification rates. Especially $t_{Bal} = 0.5$ led to a significant decrease in accuracy. As the goal of the balancing threshold is to increase accuracy, we only considered threshold values which reduce misclassification of the underrepresented class.

For each value $t_{Bal} \in \{0,\ 0.1,\ 0.2,\ 0.3,\ 0.4\}$, 50 different validation sets are extracted from dataset C. All validation sets are randomly chosen and consist of 40-60% green points. The remainder of dataset C is then balanced according to the threshold value t_{Bal} and used to train the model. Figure 6 shows the averaged misclassification on these different validation sets.

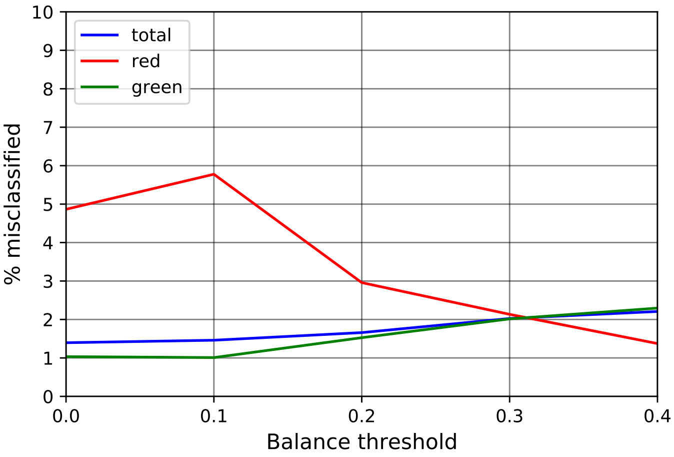


Fig. 6: Dataset balancing effect on dataset C.
Parameters: $n = 50$, $pen = 0.2$, epochs $= 1800$, batch size $= 912$.

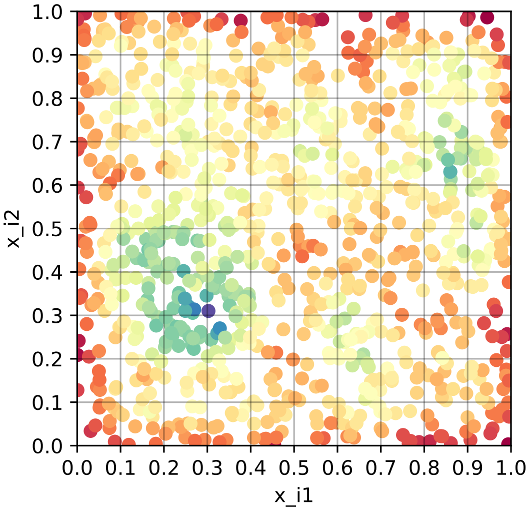


Fig. 7: Density plot showing the points of dataset A, color-coded according to the number of other points in a radius of 0.1.

We found that for dataset C, the optimal threshold value is 0.3, as red misclassification drops by 2.73% while green misclassification is increased by only 1%.

Finding such a threshold value for dataset A turned out to be more difficult: We observed that balancing the dataset with any threshold would result in either no difference at all or in a significant drop in accuracy. We assume that the small size of the dataset causes this effect. To understand this effect, we analyse dataset A further: For each data point, we calculate the number of "neighboring data points" in a radius of 0.1. We find that, on average, every point has 29.68 other points in its proximity and that the majority of points are equally spread over the unit square I, as can be seen in Figure 7. Therefore, duplicating only one point in a square of the grid in Figure 7, where red and green are equally represented, will almost certainly have a significant impact on the model's behavior. Considering these observations, we decided not to balance dataset A before training.

We conclude that balancing a given dataset can lead to an improvement in the training of the model. However, it should only be applied if the dataset is large enough and a safety-critical class is heavily underrepresented. Even then, extensive testing is necessary in order to avoid major distortions and to find the optimum threshold value t_{Bal}.

D. Penalty effect

Scenarios in which some mistakes are profoundly more impactful than others can be found in our everyday lives: Misclassifying a red traffic light as green can result in a serious traffic accident, while incorrectly classifying a green light as red usually only leads to furious honking of other drivers.

The setting of this challenge is quite similar. Misclassifying a red point as green is assumed to be more critical than misclassifying a green point as red. To integrate this information into our model's behaviour, we introduced the *custom_penalty_loss* function which artificially reduces the impact of green misclassifications, consequently making red misclassification seem more critical. This function receives the prediction output of the model as well as the ground truth values of the training samples. The model's output for a data point is represented by a tuple (p_G, p_R), where p_G is the probability that the point is green and p_R is the probability that it is red. Due to the use of the softmax function, $p_G + p_R = 1$ always holds.

The *custom_penalty_loss* function manipulates the predictions (p_G, p_R) of the model before calculating the loss. If the model misclassifies a green point as red, a constant penalty value $pen \in [0, 1]$ is added to p_G and subtracted from p_R. In cases where $p_G + pen > 1$ and $p_R - pen < 0$ applies, the model output is set to $(1, 0)$. The result of this manipulation is the artificial reduction of the loss when misclassifying a green point. In case of correct classification or the misclassifcation of red points, no changes are made to the prediction. The custom loss function then uses an underlying standard loss function to calculate and return the loss of the (possibly) manipulated predictions. Throughout all of our calculations, we used Sparse Categorical Crossentropy as the underlying loss function. We call the impact of this adjustment the *penalty effect*.

1) Approach to identify the optimal penalty: Picking a penalty value consists mainly of two considerations: 1) how many green misclassifications are we willing to accept and 2) how low do we want the red misclassification rate to be.

A penalty value of 0 results in no penalty effect at all, while a value of 1 results in all green misclassifications being fully ignored. A model which uses a penalty value of 1 rarely classifies any data points as green, making 1 an obviously bad choice for a penalty value. To determine a better penalty value, we repeatedly train our model using different penalty values and observe the changes in the misclassification rates on unseen validation sets.

We start by selecting n different validations sets, each 15-16% the size of the dataset. All sets are randomly chosen and consist of 40-60% green points. For each penalty value, we remove the validation set from the dataset, initialize the model weights, and train the model using the remaining training points. This process is repeated for each of the n validation sets. The total, red, and green misclassification values are then averaged for each penalty value.

Such a penalty value analysis for dataset B can be found in Figure 8, using 20 different penalty values and $n = 50$. As hypothesized earlier, penalty values near 1 lead to near-zero red misclassification at the cost of skyrocketing green misclassification. The downward trend in red misclassification rates as the penalty value increases suggests an inverse relationship between the penalty value and the number of red misclassifications. This shows that the use of the custom penalty loss function leads to the desired decrease in red misclassifications.

While red misclassification rates show a steady decline, green misclassification rates show a steady increase on the interval $[0, 0.5]$. Once we increase the penalty value above 0.5, however, we see a rapid increase in green misclassification of at least 4.7%.

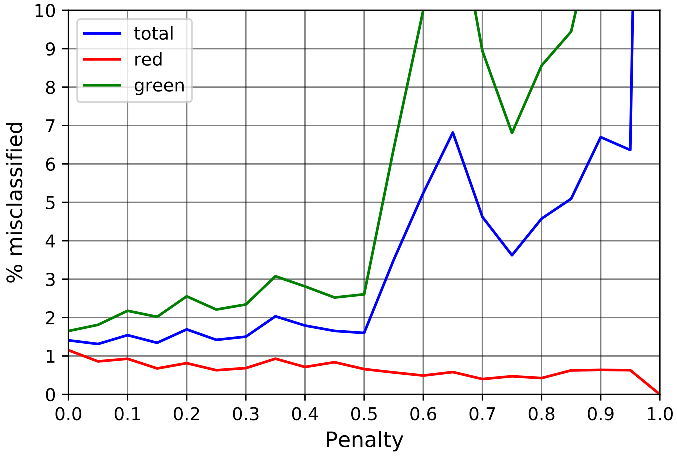


Fig. 8: Penalty effect on dataset B.
Parameters: $n = 50$, epochs $= 1800$, batch size $= 912$.

We are not willing to accept such a steep increase in green misclassification in return for very little further reduction of red misclassification. However, we find that these misclassification rates do not necessarily follow a particular pattern, making cost estimation difficult without extensive testing. Based on our testing, we recommend a penalty value between 0 and 0.5.

It is to be noted that a similar penalty effect was observed on all three datasets, further underlining the effectiveness of the custom penalty loss function.

After performing extensive testing on each dataset, we agreed on the following penalty values:

Dataset A	pen = 0.4
Dataset B	pen = 0.25
Dataset C	pen = 0.2

Visualizations of the penalty effect on datasets A and C analogous to Figure 8 can be found in appendix D.

2) Visualization: To understand how our model classifies data, we can visualize its predictions: By generating predictions for a grid of 1000×1000 data points, we obtain a predicted color as well as the model's certainty in its prediction. Those results are then color-coded and plotted, as can be seen in Figure 9 for the color red. In this specific figure, the model was trained without the penalty function. In comparison to the exact same model trained with a penalty value of $pen = 0.25$, we can easily see how the area of greater uncertainties is significantly reduced due to the penalty effect (Figure 10). Moreover, we can observe that areas where we have fewer data points (like in the top right corner), the model is highly certain that these data points are red. In contrast, the model without the penalty effect considered these points to be green, likely due to 4 green points vs. 2 red points being present in this area of dataset B. However, 6 data points hardly provide reliable information about this area. The penalty effect causes these points to be classified as red, thereby increasing the safety of the classification.

E. Alternative Penalty Approaches

To further optimize the penalty effect, we conducted two different experiments during the training stage:

- Increasing-penalty training
- Decreasing-penalty training

The *increasing-penalty training* method starts training the model with a penalty value of 0 on the first epochs and then periodically increases the penalty value until the final, previously-determined penalty value is reached. Over the last epochs, we train the model with this final penalty value. *Decreasing-penalty training* changes the penalty effect in the opposite direction during training.

For example, in Figure 12 on dataset B, we increased the penalty value every 50 epochs until we reach 1300 epochs. From there on, we train with the full penalty of 0.25 for the last 500 epochs, resulting in 1800 epochs in total. Furthermore, we tested the rate of misclassification on 50 different validation sets and averaged them, similar to the penalty effect approach. Figure 12 clearly indicates that *increasing-penalty training* as well as *decreasing-penalty training* perform worse. Hence, we will not further investigate these approaches.

F. Threshold effect

As we've seen previously, our model is highly certain of its predictions in most areas. Along the "borders" between red and green areas, however, predictions are less confident, as can be seen in Figure 11. Since guessing that a data point is red is safer in such uncertain scenarios, we introduce a threshold function that acts as a simple linear threshold neuron at the end of the model for predictions only. The model will only predict a data point as green if the certainty in its decision is equal to or above the certainty threshold. Otherwise, it swaps the predictions.

After extensive testing, similar to the findings of the penalty values, we agreed on the following certainty threshold values for each dataset:

Dataset A	t_{cert} = 0.9
Dataset B	t_{cert} = 0.9
Dataset C	t_{cert} = 0.95

Even though certainties below 0.9 seem to be far from "purely guessing", they are not sufficient to warrant green classification.

Analysis of optimal certainty threshold values on dataset C can be found in Figure 13: an increasing t_{cert} value leads to slowly rising green and falling red misclassification rates.

$t_{cert} = 0.95$ was deemed the optimal value for dataset C, since red misclassification rates were reduced by $\approx 80.5\%$ (2.15% to 0.42%), while green misclassification rates only increased by $\approx 44.7\%$ (1.83% to 3.31%).

Analogous visualizations for datasets A and B can be found in appendix E.

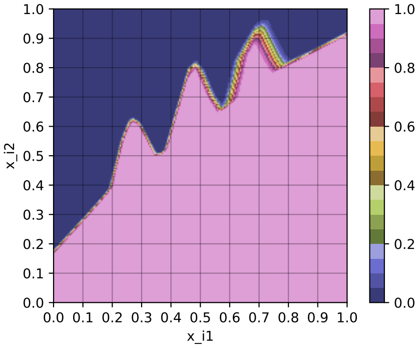


Fig. 9: Certainty for red with $pen = 0$ and $t_{cert} = 0$ on dataset B.

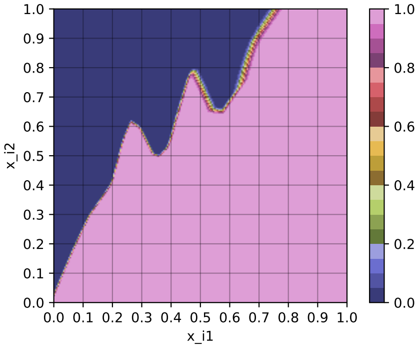


Fig. 10: Certainty for red with $pen = 0.25$ and $t_{cert} = 0$ on dataset B.

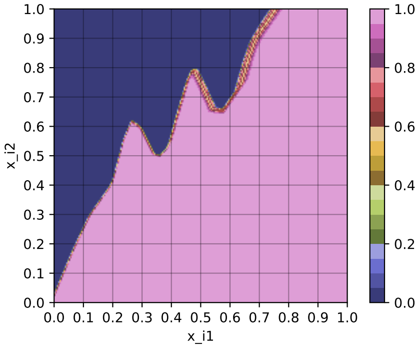


Fig. 11: Certainty for red with $pen = 0.25$ and $t_{cert} = 0.9$ on dataset B.

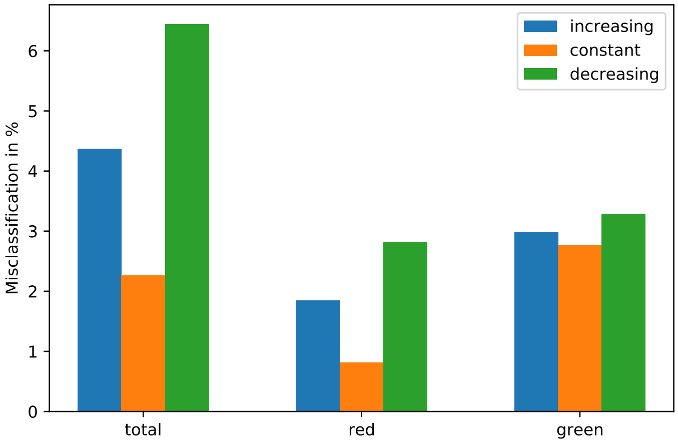


Fig. 12: Average misclassification with different training approaches on dataset B.
Parameters: $n = 50$, $pen = 0.25$, epochs $= 1800$, batch size $= 912$.

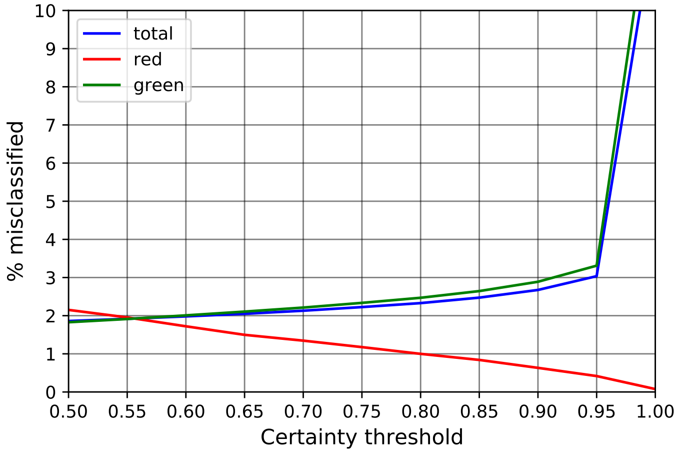


Fig. 13: Average % misclassification with respect to the certainty threshold value on dataset C.
Parameters: $n = 50$, $t_{bal} = 0.3$, $pen = 0.2$, epochs $= 2000$, batch size $= 1800$

III. Misclassification upper bound

In this section, we describe how we assess the safety and dependability of our model's predictions.

We've implemented methods such as the penalty effect and the certainty threshold which reduce the number of red points being misclassified and therefore increase the safety of our predictions. On these grounds, we now try to determine a *reliable* misclassification probability. Such an assessment is crucial, as we operate in a safety-critical area and want to provide the user with safety assertions.

The question arises, whether it is possible to determine a *guaranteed* maximum misclassification probability or to even prove that misclassification is impossible. We believe that, when given only the information provided in this challenge, such a guaranteed boundary does not exist. The primary reason for this lies in the continuous nature of the assumed two-dimensional space I.

It would be possible to provide a guaranteed misclassification upper bound if we were able identify areas of I which *definitely* only contain red points or which *definitely* only contain green points. When observing the three datasets, it may seem clear that certain areas contain only red points or only green points, and that we should be able to guarantee 0% misclassification here if our certainty maps show that our network indeed only predicts the "correct" color in those areas. Although this may be true for the few points which we were given as a training set, we simply have no reliable information about the vast majority of points in I, as I is a continuous space which contains infinitely many points.

Assume we have two points $p, q \in S$ with $p \neq q$, meaning that p and q are red data points at different locations. No matter how small the distance between the two points, we can never assume that for an unknown point r, located in the middle of p and q, $r \in S$ is valid - it is well possible that Player A has chosen the subset S in such a way that it contains the arbitrarily close together points p and q, but not the arbitrarily small space between p and q in which r lies.

In practice, we can only work with the data points that we have to make *assumptions* about the data points that we don't have. This is precisely what our neural network does as it learns the patterns of the training set, estimates what the subset S probably looks like, and then uses this information to make assumptions about the validation set during prediction.

Although it is impossible to guarantee the accuracy of our model, we can provide reliable misclassification probabilities by analysing the density of the datasets and the misclassification patterns of our network.

A. *Probability distribution*

When evaluating how certain we can be in the predictions of our model, the density of points in different areas of the provided datasets is an important factor to take into consideration. Areas of a dataset X which have a relatively high density of points provide us with much more information about the subset S_X than areas which have a low density of points - if we have no or very few given data points in a region, we cannot make reliable predictions about the color of new data points placed in that area. Higher trust can therefore be placed in the predictions of our network in high density areas than in low density areas. We have to incorporate this information when calculating the misclassification probability of our network.

For this sake, we split our datasets into grids where every square represents a subset of I and has a density value given by the number of points present in that square. The number of squares per grid varies from dataset to dataset due to the differences in the total number of points per dataset. The grid "resolution" was chosen so that for each dataset, every grid square contains at least 10 points on average. Thereby, we receive a meaningful distribution of densities across the grid squares of each dataset. The grid sizes of the three datasets were chosen as follows:

	grid size
Dataset A	10×10
Dataset B	20×20
Dataset C	50×50

Note that the choice of grid sizes does not affect our final misclassification probability upper bounds, but is more of relevance for the readability and interpretability of the misclassification patterns presented in the next section.

The density distribution of the training points in a dataset X is directly proportional to the probability distribution P_X of that dataset by a constant factor of $\frac{1}{n_X}$. We know that data points are generated by drawing variates from the probability distribution P_X. As the already known training points were drawn from the same probability distribution, the density of training points can be seen as a "likelihood-of-appearance-map" for the next data points to be generated.

Drawing a density grid of a dataset X and then dividing all grid values by n_X therefore serves us as an estimation for the probability distribution P_X and as a measure for the density distribution of the training points in dataset X. Figure 14 shows the probability distribution map of dataset B, calculated as explained above. The color coding of each square shows the probability that the next randomly generated variate from P_B will land in that square, as a value between 0 and 1. The probability distribution maps of datasets A and C can be found in appendix F.

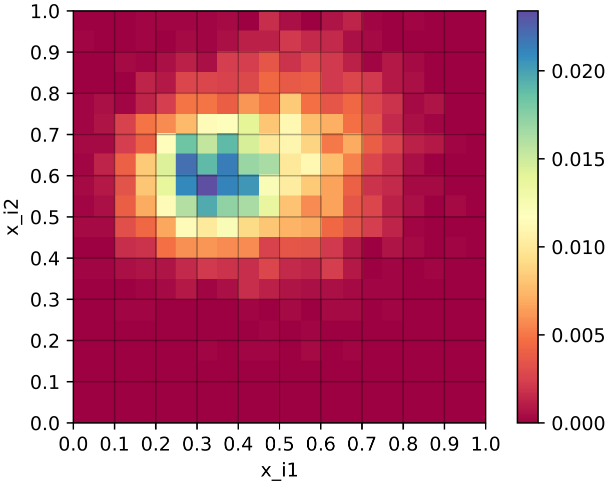


Fig. 14: The probability distribution map of dataset B.

B. *Misclassification distribution*

When classifying validation sets randomly generated from a probability distribution P_X for a dataset X, the misclassified points are generally not evenly distributed across the unit square I. Rather, there are areas where very few misclassifications occur and areas where misclassifications occur more frequently. The regions of high misclassification probability tend to be "border regions" between S and $I \backslash S$, especially when the borders are "blurry", i.e. there are few data points in the border region. This issue is especially prominent in dataset A, as can be seen in Figure 1, as dataset A has very few data points in total.

Determining the misclassification probability of validation points in different areas of a datasets can help us calculate a reliable total misclassification upper bound for that dataset. As in section III-A, we split our datasets into grids. To calculate representative misclassification probabilities per grid square, we performed multiple training and validation runs of our network on each dataset.

In each run, we first extract a random validation set from the given dataset and then train our (Glorot weight initialized) network on the remaining training set using the optimum training hyperparameters and penalty effect values calculated in section II. We then classify the validation set and for each square of the dataset grid save the proportion of validation points chosen from that square which were misclassified. We repeat this process n times and then calculate the average misclassification probability per square over the n runs. For every grid square, this leaves us with the probability that a newly generated point will be misclassified if it is located in that square. Additionally, we calculate the misclassification probability per grid square specifically for red points, as the misclassification probability of red points is of supreme interest.

The following number of training and validation runs were completed for each dataset to calculate the misclassification probability per grid square:

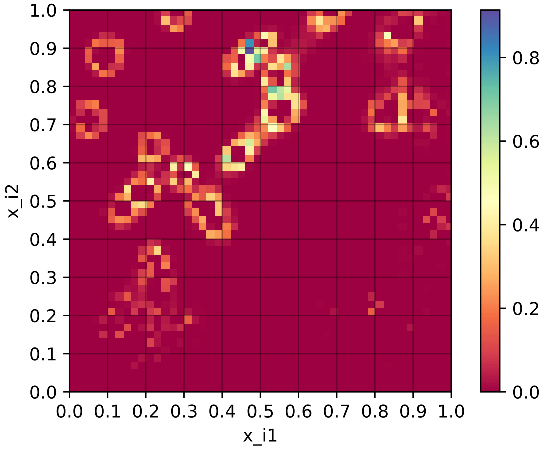


Fig. 15: Total misclassification probability per square in dataset C.

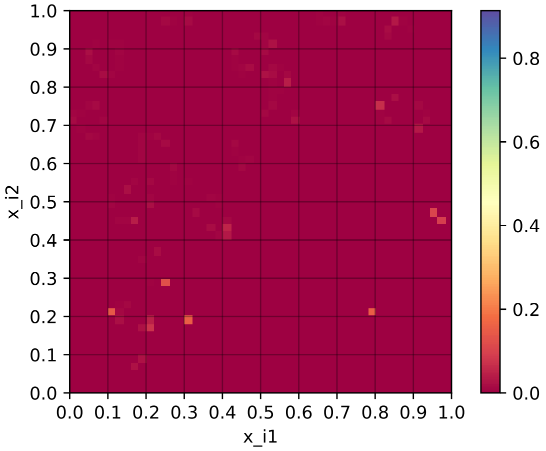


Fig. 16: Red misclassification probability per square in dataset C.

	n
Dataset A	2000
Dataset B	2000
Dataset C	1000

As we selected a random validation set from the given datasets in every run, and the datasets were randomly generated from the probability distributions P_X, we may assume that our results are representative of any validation set randomly generated from P_X.

Figure 15 and Figure 16 show the total and red misclassification probabilities per grid square in dataset C, as values between 0 and 1, respectively. Analogous plots can be found for datasets A and B in appendix G.

Note that for the training and validation runs on dataset C, we used the training hyperparameters batch size $= 1800$ and epoch number $= 2000$ instead of the calculated optimum parameters of batch size $= 2000$ and epoch number $= 2400$ due to time and computational power constraints. Despite not being optimal, our tests showed that the values used also produced very good results.

C. Misclassification probability upper bound

We can now calculate realistic and reliable misclassification probability upper bounds for both total and red misclassification by combining the probability that a new variate generated from the probability distribution P_X for a dataset X falls into a certain grid square and the probability that a point located in that square is misclassified. The prior information is encoded in the probability distribution maps computed in section III-A, and the latter information is encoded in the misclassification probability per grid square maps computed in section III-B.

We multiply the values of all grid squares in the probability distribution map of a dataset with the values of all corresponding grid squares in the misclassification probability map of that dataset. The result is the so-called *weighted misclassification probability map* where each grid square contains information about how likely it is that the next randomly generated data point lands in this square and is misclassified. In the case of using a red misclassification probability map, each grid square of the resulting weighted misclassification probability map contains information about how likely it is that the next randomly generated red data point lands in this square and is misclassified. The total and red weighted misclassification probability maps for all datasets can be found in appendix H.

By summing up the values of all grid squares, we arrive at a single average misclassification probability percentage. Doing this for total and red weighted misclassification probability maps for all datasets results in the following average total and red misclassification probability percentages:

	total	red
Dataset A	6.89%	2.16%
Dataset B	2.34%	0.19%
Dataset C	2.62%	0.10%

Note that these values are the average misclassification probability values over the 2000 (1000 for dataset C) training and validation runs for each dataset, meaning that about half of our runs fulfill these misclassification probabilities and about half of our runs do not fulfill them.

Individually looking at each of the 2000 (1000 for dataset C) training and validation runs lets us conclude that the following misclassification probability upper bounds are fulfilled in **99.9%** of all cases:

	total	red
Dataset A	15.23%	7.40%
Dataset B	8.63%	5.67%
Dataset C	5.45%	2.00%

D. Further improving misclassification upper bounds

The results presented in Section III-C clearly show that there is an inverse correlation between the size of a dataset and the misclassification probability when classifying new points in that dataset. These findings support our suggestion that a higher density of known data points results in greater classification confidence, as more information about the classification function is available to the network and the network can therefore predict the subset S_X of a dataset X more accurately.

Substantially expanding the sizes of the training datasets would most likely result in a great performance increase of the neural network across all datasets. This is a crucial step in lowering the given realistic upper bounds to a level which makes the model suitable for classification tasks in safety-critical areas.

The *misclassification probability per square* maps introduced here can further be employed to identify weak points in the training data. Areas which show high misclassification probabilities probably suffer from a lack of sufficient training data required to allow the model to learn the complex classification function at that point. The operators of the safety-critical system can use this information to selectively collect more training data in these particularly “dangerous” areas and thereby increase the model's accuracy and lower the total misclassification probability upper bounds.

IV. Scalability

Although the problem set by Siemens Mobility is certainly a simple and abstract one, the methods and approaches presented in this paper can be scaled in order to be applicable to complex real-world problems.

For one, neural networks can work well with higher-dimensional data and can be extended by dimensionality-reducing components such as autoencoders to improve performance when working with more than two dimensions.

Our idea of the penalty effect is scalable as well: manipulating loss functions to penalize different forms of misclassification to differing extents on higher-dimensional data poses no big problem with backpropagation-based neural network training.

Introducing certainty thresholds which only allow specific classifications to be made if a defined certainty threshold is reached can easily be implemented in neural networks which work with more than two output neurons. Different certainty thresholds for different classes and certainty thresholds which require specific value combinations from multiple output neurons are also feasible. A prerequisite for implementing a certainty threshold is that there must always be a form of safest classification which can be resorted to if none of the required thresholds are fulfilled. In this challenge, the safest classification which we default to if the certainty threshold is not fulfilled is red. In more complex real-world applications, the safest classification to resort to may dynamically adapt to the current state of the system. Take the example of a self-driving car. When the car is approaching a pedestrian crossing and the main machine learning algorithm of the car is not sure whether a pedestrian is currently crossing the street or not, you would want the car to default to a full emergency break, just to be on the safe side. However, when the car is casually swimming along in traffic and the machine learning algorithm is not sure about the road markings, you would want the car to default to keeping a steady course and speed rather than activating a full emergency break.

Finally, the process of dividing the possible range of values into subareas and calculating the misclassification probability per subarea is scalable to higher-dimensional data. The same goes for calculating the probability distribution of a given dataset. Visualization of these measurements will be impossible when working beyond three dimensions, but the mathematical operations are feasible beyond any dimensional limit.

V. Conclusion

We propose a fully-connected feedforward neural network architecture as a classifier for the provided datasets. Our model achieves an average total accuracy of 93.11%, 97.66%, and 97.38% and an average red accuracy of 97.84%, 99.81%, and 99.90% on datasets A, B, and C, respectively. Additional optimization of the network architecture could be undertaken to further increase classification performance and reduce training times. Our contribution to the Siemens Mobility AI Dependability Assessment Challenge is two-fold:

For one, we customize the classification model to increase the safety of its predictions. Under the assumption that red misclassifications have significantly worse consequences than green misclassifications, we tune the model to minimize red misclassifications. This makes our model inherently more safe than other classification models. Using a highly optimized neural network architecture with lower general misclassification rates would certainly further improve the safety of the model.

While misclassification rates determined during model validation do provide some idea of the safety of a model, they are highly dependent on the data that the model was *tested* on. To reach more dependable misclassification probability estimates, we focus on the knowledge that we have about the datasets. We determine areas in which our model struggles to correctly classify data points and calculate the probabilities that the next data point generated will fall into these areas. This allows us to provide reliable (though not guaranteed) upper bounds for total and red misclassification probability. In 99.9% of cases, our network fulfills the upper bounds of 15.23%, 8.63%, and 5.45% for total misclassification probability and the upper bounds of 7.40%, 5.67%, and 2.00% for red misclassification probability on datasets A, B, and C, respectively.

References

[1] A. Krizhevsky, I. Sutskever, and G. Hinton, “Imagenet classification with deep convolutional neural networks,” *Neural Information Processing Systems*, vol. 25, 01 2012.

[2] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” 2014.

[3] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, Łukasz Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean, “Google's neural machine translation system: Bridging the gap between human and machine translation,” 2016.

[4] E. Bill, R. Giesler, and F. Knispel, “Siemens Mobility AI-DA Challenge,” 05 2021. [Online]. Available: https://github.com/ericbill21/siemens

[5] F.-F. Li, R. Krishna, and D. Xu, “CS231n: Convolutional Neural Networks for Visual Recognition,” http://cs231n.stanford.edu/, 2020.

[6] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2017.

Appendix A
Hyperparameters

Dataset Parameter	A	B	C
epochs	1450	2050	2400
batch size	16	912	2000
t_{bal}	-	-	0.3
pen	0.4	0.25	0.2
t_{cert}	0.9	0.9	0.95

TABLE II: Overview of the optimal hyperparameters for training.

Appendix B
Loss plots

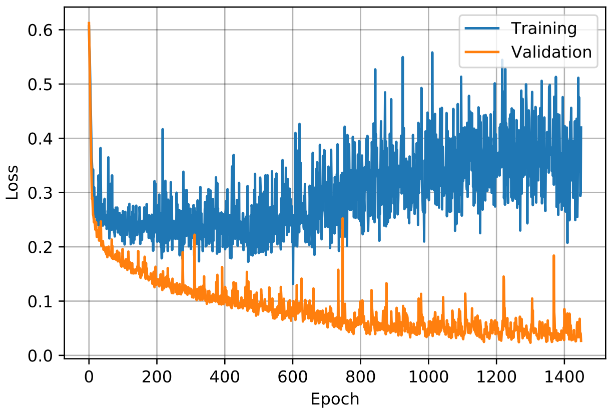


Fig. 17: Loss plot for a training and validation run on dataset A.

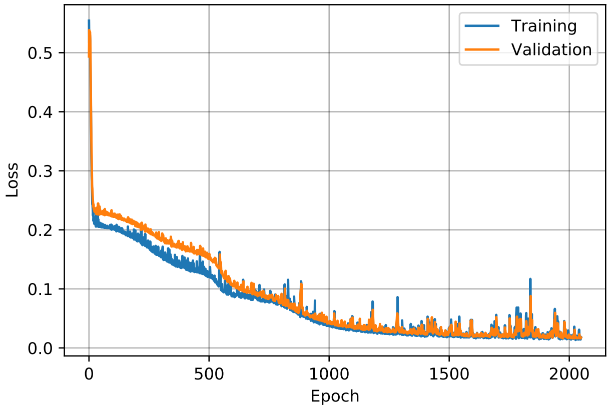
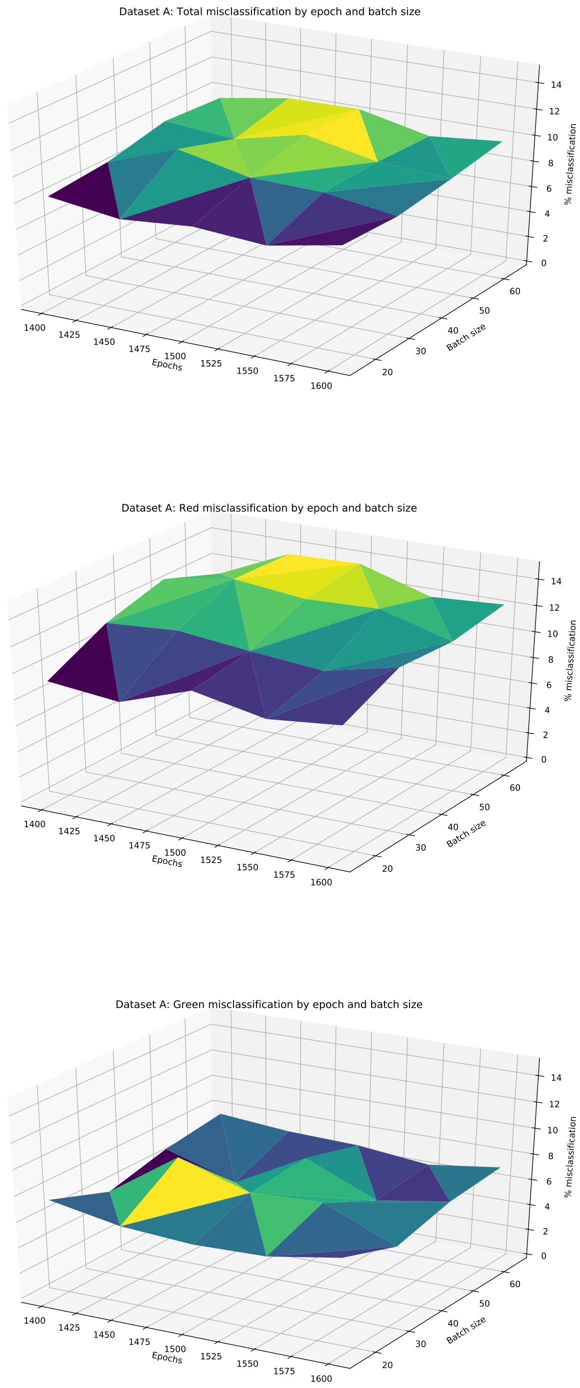


Fig. 18: Loss plot for a training and validation run on dataset B.

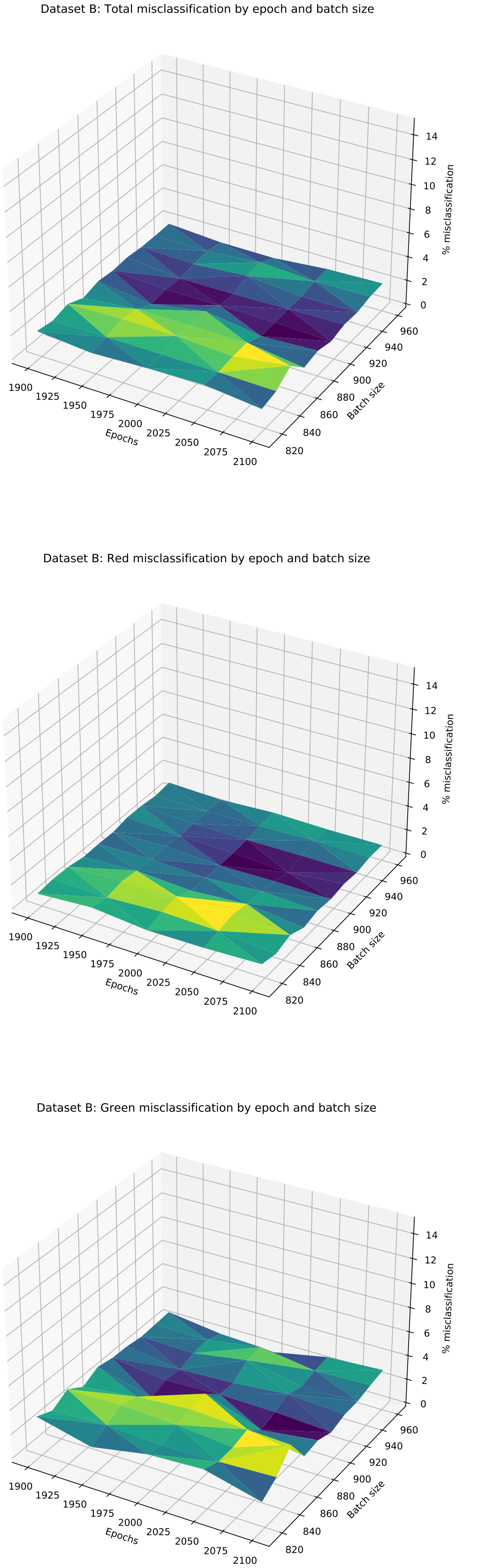
Appendix C
Misclassification w.r.t epochs & batch size


Dataset A: Total misclassification by epoch and batch size

Dataset A: Red misclassification by epoch and batch size

Dataset A: Green misclassification by epoch and batch size

Fig. 19: Average % misclassification with respect to batch size and epoch number on dataset A.
Parameters: $n = 10$, $B = \{16, \dots, 64\}$ in increments of 16, $E = \{1400, \dots, 1600\}$ in increments of 50.

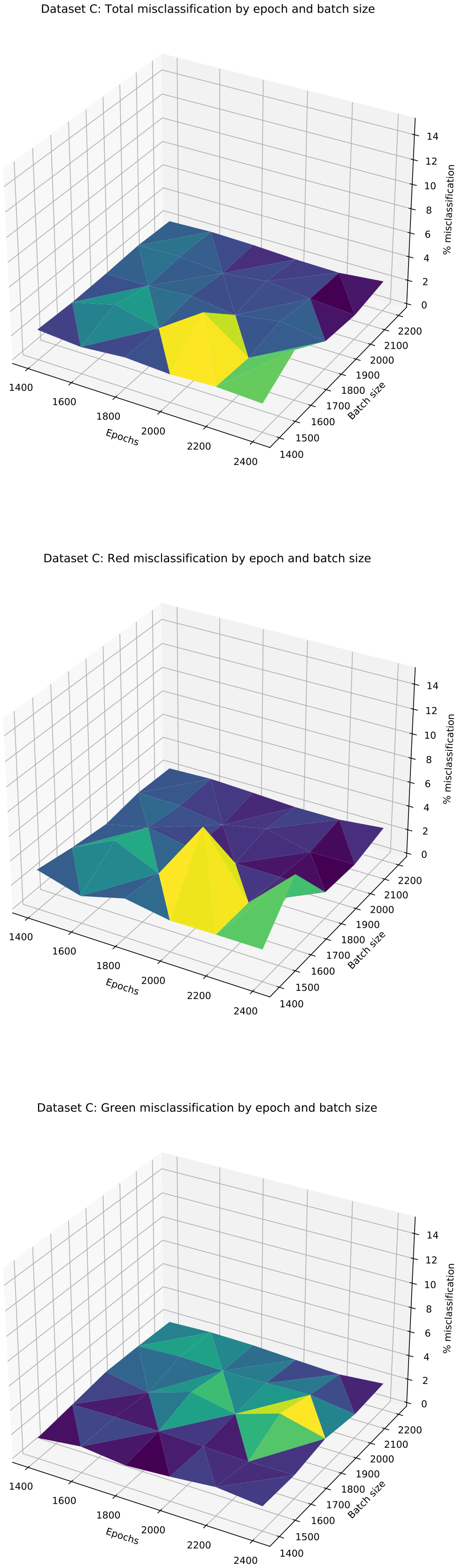


Dataset B: Total misclassification by epoch and batch size

Dataset B: Red misclassification by epoch and batch size

Dataset B: Green misclassification by epoch and batch size

Fig. 20: Average % misclassification with respect to batch size and epoch number on dataset B.
Parameters: $n = 50$, $B = \{816, \ldots, 960\}$ in increments of 16, $E = \{1900, \ldots, 2100\}$ in increments of 50.



Dataset C: Total misclassification by epoch and batch size

Dataset C: Red misclassification by epoch and batch size

Dataset C: Green misclassification by epoch and batch size

Fig. 21: Average % misclassification with respect to batch size and epoch number on dataset C.
Parameters: $n = 10$, $B = \{1400, \ldots, 2200\}$ in increments of 200, $E = \{1400, \ldots, 2400\}$ in increments of 200.

Appendix D
Penalty effect

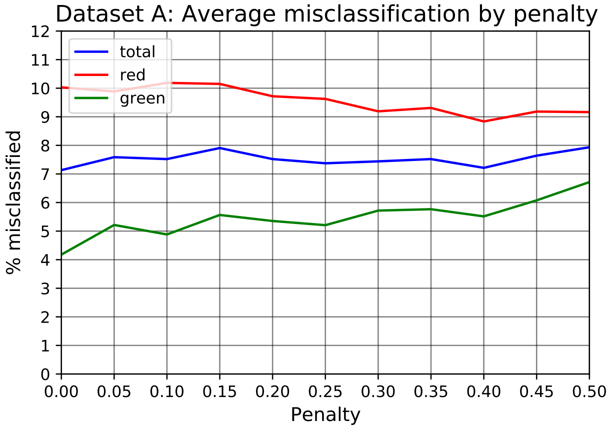


Fig. 22: Average % misclassification with respect to the penalty value on dataset A.

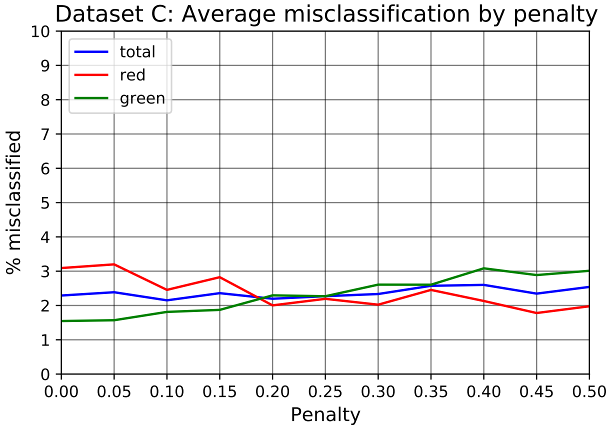


Fig. 23: Average % misclassification with respect to the penalty value on dataset C.

Appendix E
Certainty threshold effect

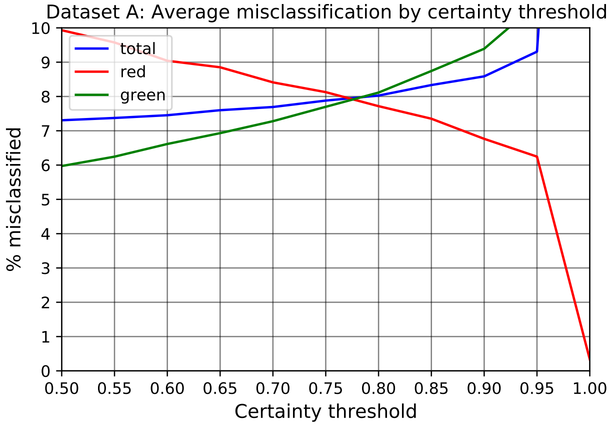


Fig. 24: Average % misclassification with respect to the certainty threshold value on dataset A.

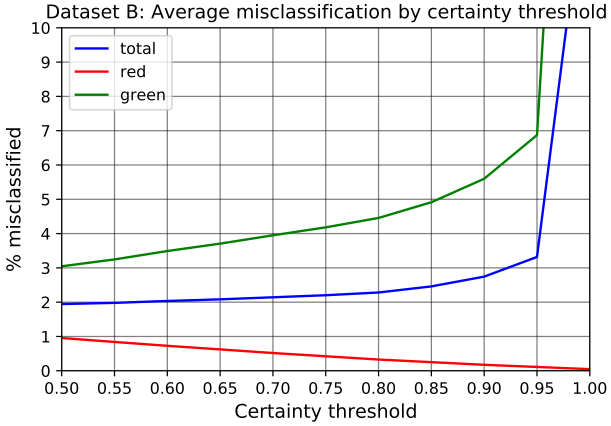


Fig. 25: Average % misclassification with respect to the certainty threshold value on dataset B.

APPENDIX F
PROBABILITY DISTRIBUTIONS

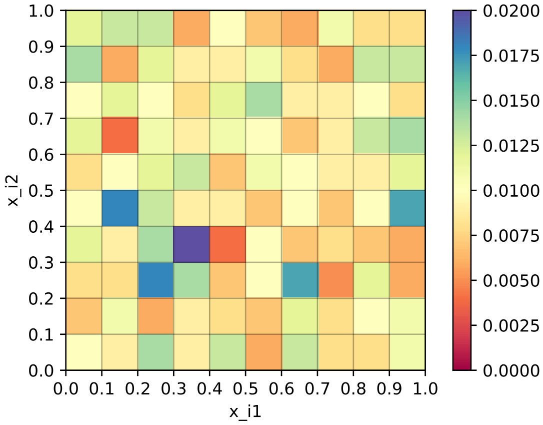


Fig. 26: The probability distribution map of dataset A.

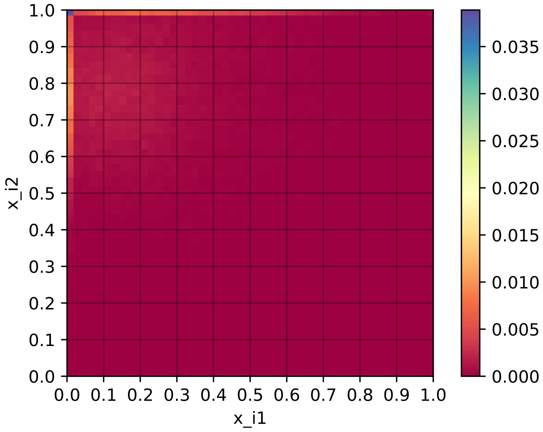


Fig. 27: The probability distribution map of dataset C.

APPENDIX G
MISCLASSIFICATIONS PER SQUARE

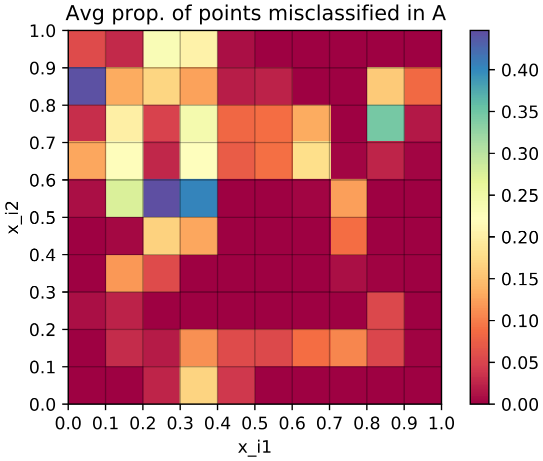

Avg prop. of points misclassified in A

Fig. 28: Total misclassification probability per square in dataset A.

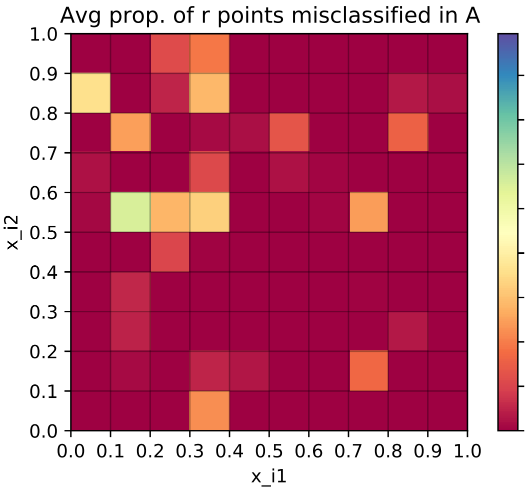

Avg prop. of r points misclassified in A

Fig. 29: Red misclassification probability per square in dataset A.

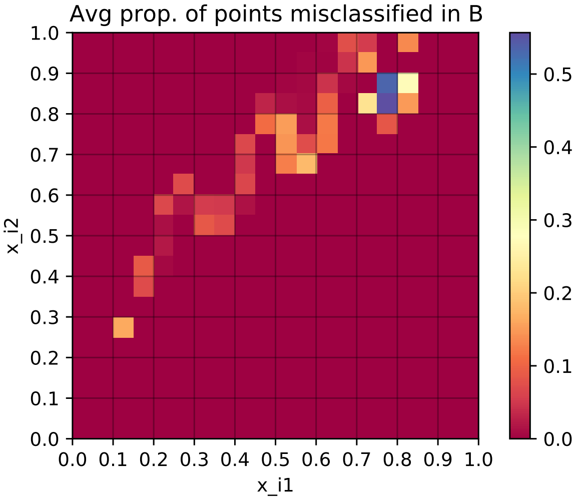

Avg prop. of points misclassified in B

Fig. 30: Total misclassification probability per square in dataset B.

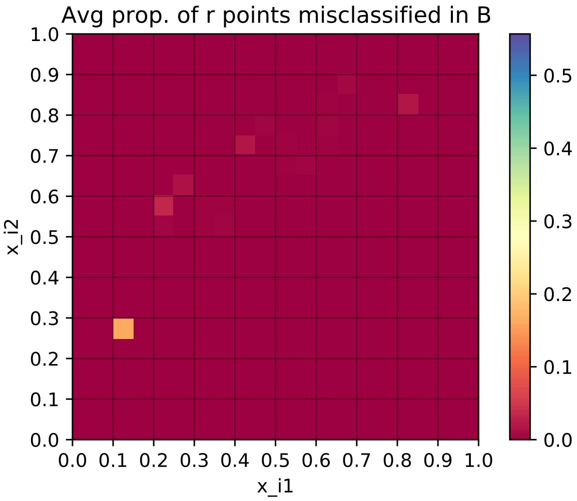

Avg prop. of r points misclassified in B

Fig. 31: Red misclassification probability per square in dataset B.

Appendix H
Weighted misclassifications per square

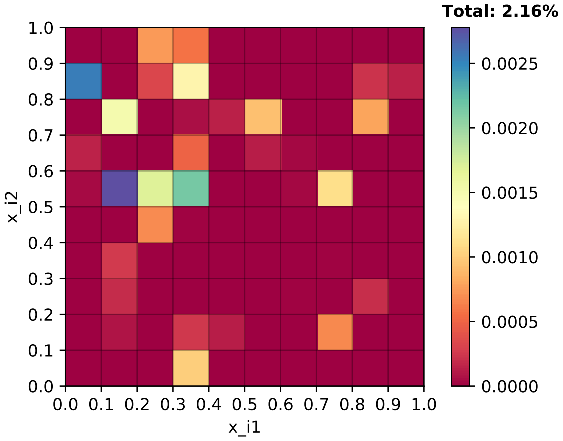


Fig. 32: Red weighted misclassification probability for dataset A.

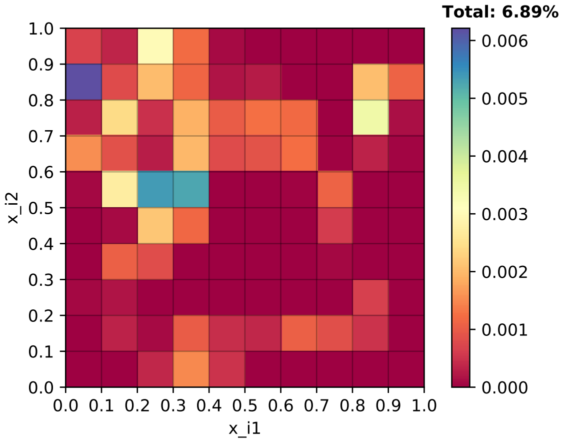


Fig. 33: Total weighted misclassification probability for dataset A.

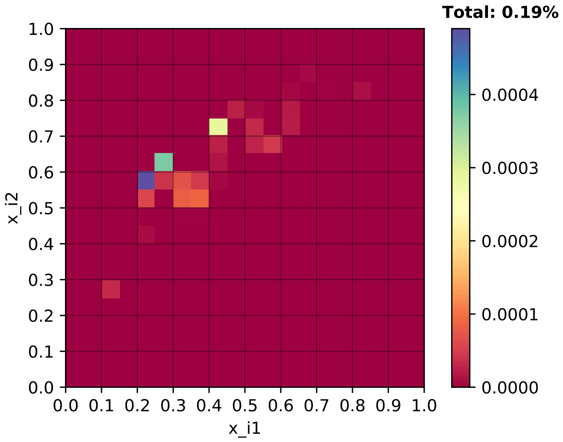


Fig. 34: Red weighted misclassification probability for dataset B.

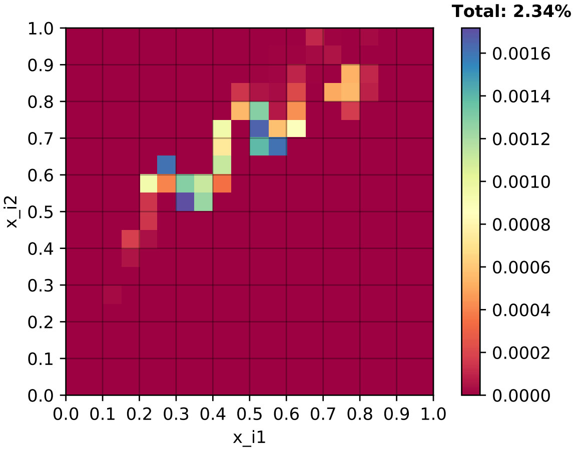


Fig. 35: Total weighted misclassification probability for dataset B.

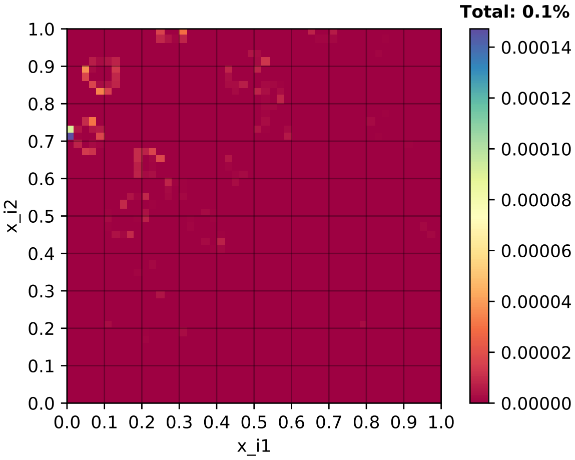


Fig. 36: Red weighted misclassification probability for dataset C.

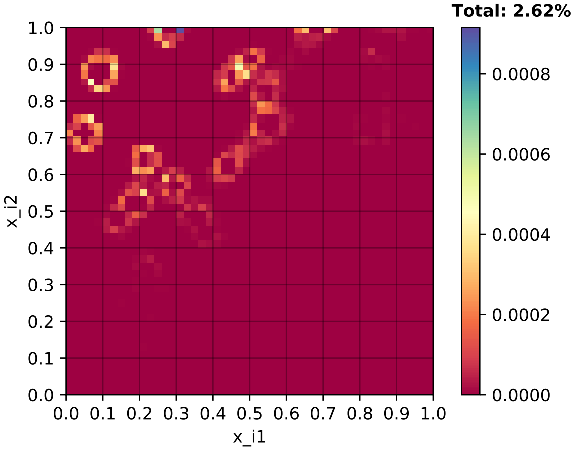


Fig. 37: Total weighted misclassification probability for dataset C.