Utrecht University

Game & Media Technology

Master Thesis

# Visual exploration of the generalization of neural network projections

*Author:*
Carlo Bredius
(4132955)

*Supervisor:*
Prof. dr. Alexandru C. Telea

*Second supervisor:*
Dr. Michael Behrisch

October 28, 2021

# Abstract

As multidimensional data becomes more abundant, so does the question to understand it. Visualizations are very informative as to what this data has to say, which is why study on dimensionality reduction methods, or projections, continuously increases. Recently, Neural Network Projections has shown much promise as a good alternative to standard projection techniques, as it is able to retain almost the same quality as any other projection technique, while removing many issues that they might have. One of the issues that has not fully been explored yet, is the stability and generalization of such trained models. Being able to understand how a projection behaves due to changes to the data, can give much insight into its reliability and thus its uses. We found that while degradation is specific to the trained projection technique or dataset, the robustness of Neural Network Projections relies mostly on the type of perturbation, where robustness is stronger against the alterations to existing data than to introduction of new false information.

# Contents

# 1 Introduction

This chapter introduces this thesis, through introduction of the problem of high dimensional data in Section 1.1, what has been done to combat this through deep learning projection techniques in Section 1.2. Then we propose our research question in Section 1.3 and then the structure of this thesis is described in Section 1.4.

## 1.1 High dimensional data

Data becomes richer over the years in many application fields and with it comes the task of properly analyzing it. We start to better understand our data, with the advantage of extracting more use out of each individual sample. This does however have a negative that comes along, where it makes the data harder for humans to understand. Due to increased information gathered from samples, we create a multitude of dimensions which overloads the comprehension for humans.

Understanding such data is hard for many reasons, which are all connected to the many dimensions and how we, as humans, are not fully capable of perceiving them all at the same time. Hence why we have many techniques to better understand the many dimensions of our data. One of the techniques of choice is called Dimensionality Reduction (DR), in which we compress the high dimensional data into a lower number of dimensions, mostly being 2D projections. Within DR we also have several choices for creating such projections, like Principal Component analysis (PCA)[1] or t-Distributed Stochastic Neighbor Embedding (t-SNE)[2]. Such projection techniques create a visualized overview for humans to better understand the data. These projections scale well with the number of dimensions of data. They can more easily emphasize importance, and as a human we are well-trained to interpret visual stimulus making any such concept easier to understand. However, computing projections of such large, high-dimensional, datasets is not trivial. The algorithms that compute them do have problems, such as them being complex, hard to parametrize, non-deterministic, slow as the number of samples increase and often unstable. The latter of which will become an important subject of this thesis.

## 1.2 Deep Learning Projection

As there are several techniques to compress high-dimensional data into more comprehensible projections, there are several disadvantages in using these DR techniques. To combat these disadvantages, there has been an introduction of using deep learning projections.

Recently there has been a breakthrough in the area of DR using these so-called deep learning projections. Espadoto et al.[3] suggests Neural Network Projection (NNP), which uses a neural network to learn how the individual points are projected. It takes both a high-dimensional data vector and a projection, inferred from any DR technique, of this high-dimensional vector as input for the training of a model. This model will provide *Quality* in similar results to the learned projection, high *Scalability* as it projects up to a million observations with hundreds of dimensions in a few seconds, *Ease of use*, as it does not need any complex parameters aside from training epochs, *Genericity*, as it can handle any data represented as a high-dimensional vectors and at last *Stability and out-of-sample support*, as it is a deterministic technique allowing one to project new observations for a learned projection without having to entirely recompute it.

However, this new technique introduces the problem that most deep learning methods have, namely it introduces a *black box*. A neural network uses many, often hidden, layers in which many weights are activated and changed. This means that we do not have any knowledge of the internal workings of the technique, but we merely have its input and output (or transfer characteristics). The neural network can approximate the chosen DR function, but studying the approximated

function does not give any insights on the structure being approximated, as there is no simple link between the weights and the function being approximated. In short, the problem that is introduced is how intractable the method is, as we have no clear idea of what happens with the data within this black box.

As in any machine learning process, NNP has to infer a result based on a training set. How far one can generalize away from this training set is still quite unclear. Understanding this is extremely important if users want to use NNP in practice. They should have an idea on how similar their data should be to their training data to deliver a good projection. Espadoto et al.[4] studied the above by measuring and tuning for quality through metrics such as Trustworthiness, Continuity and Neighbourhood hit, of the NNP projections from unseen data. It explains that at some point degradation is being introduced. However useful this was, it does not fully answer the question *when* testing data is too far from the training data. We know that the degradation does happen at some point, but we do also do not know how. To further specify how the degradation takes effect, we question if this depends on the dimensionality of the dataset. Is degradation more visible when data changes in some specific dimensions rather than all? What does the projection do when degradation happens or when we change specific dimensions? Are patterns to be seen when degradation happens that could help the user in understanding what to expect when using NNP on very different data?

To answer the previously described questions, we propose a methodology to use, so that one can understand how NNP generalizes. This methodology will use a family of perturbations so that we can analyze what happens in the projections. Running the method with the chosen perturbations could give a better understanding of what causes harm or contrarily what causes no harm to the generalizations of the method. This will give insight into how the method is behaving under changing circumstances. To get a clear understanding of this behaviour, we will introduce several ways of visualizing the changes in projection. The main contribution is the ability to change the input data, and being able to measure the effect of that change on the output of the neural network.

To support this methodology, we will implement an interactive tool. This tool will yield an overview of the output projection together with a range of perturbations. The user will be able to determine which perturbation is applied and the amount of influence it has. The user will then be able to see how the samples move due to this perturbation influence. Moreover, the tool will provide several visualizations to aggregate trends of the moving samples. This method with its perturbations and visualizations will then be applied to various different real-world datasets and models with dissimilar algorithms behind them, to see if the degradation is generalizable.
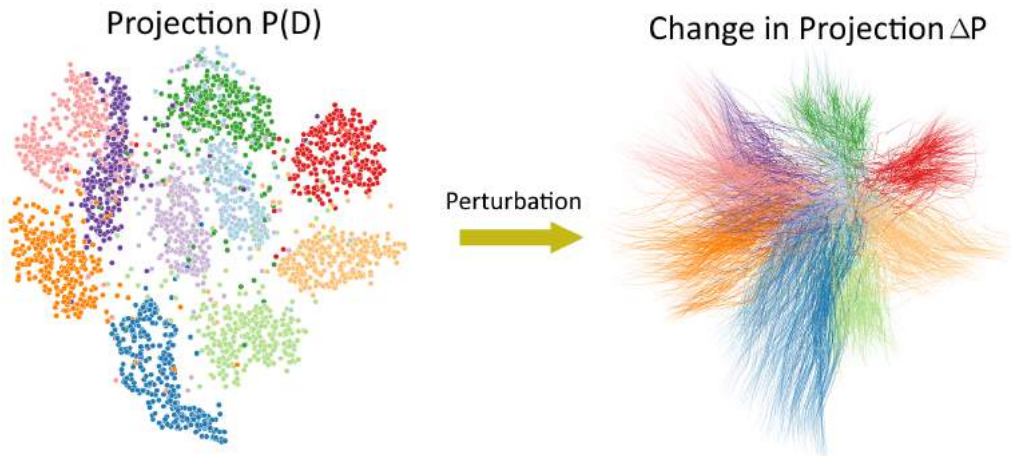


Figure 1: Optimally trained NNP inference on data $D$, where a perturbation shows the change in projection $\Delta P$.

## 1.3 Research Question

The aim of this thesis is to study the generalization of NNP. Therefore the research question states:

*When and how do the projections inferred from Neural Network Projection change based on added perturbations on the input data?*

To further investigate this question we will develop a family of perturbations towards the input data, run NNP on this perturbed data, design new visualizations to summarize how the output projections change in presence of various perturbations and validate/demonstrate these changes on various real-world datasets, as illustrated in Figure 1.

## 1.4 Structure of the thesis

This thesis is structured as follows. Chapter 2 analyzes DR techniques to be used for NNP and visualizations which increase understanding of the behaviour of such methods. Chapter 3 describes the development of our proposed tool and through what methodology we will approach our research question. Chapter 4 describes the results, which will then be discussed and concluded in Chapter 5. At last, Chapter 6 will build upon found results and propose future works.

# 2 Related Work

In this chapter, a look is given to what has already been done and accomplished in relation to the subject. First, in Section 2.1, we will talk about Dimensionality Reduction, mentioning various techniques and how one would measure them. Then, in Section 2.2, we introduce Neural Network Projection, an approach to the problem from a machine learning perspective. In Section 2.3 we will mention various ways to describe the space of change to the user. Then in Section 2.4, we describe some work that explores improvements on generalizability and in Section 2.5 we conclude what we found in this Chapter.

## 2.1 Dimensionality Reduction

This section we will look at how high dimensional data is made comprehensive by the technique called Dimensionality Reduction.

As sample information becomes richer, so does the number of dimensions. For a human to get a good understanding of how samples are different or similar to each other, DR techniques are applied to reduce the number of dimensions towards a more clear overview of what the data has to say. Dimensionality Reduction is the transformation of data from high-dimensional space to low dimensional space, while trying to reduce the loss in meaningful properties of the original data, making the low-dimensional projection ideally close to its intrinsic dimension. Its goal is to give a better understanding of the data, as for humans, the data is easier to understand when it is displayed in a low dimensional space, like 2D.

Many different DR techniques have been proposed to reach such an understanding. When data was not as complex as it is today, linear techniques such as Principal Component Analysis (PCA)[1], factor analysis[5] and classical scaling[6] were used to accomplish this. However, as today's data increases in non-linearity, more specialized techniques are introduced that perform better on this complex natural data. Van der Maaten et al.[7] investigate twelve (nonlinear) DR techniques, where they observe that most nonlinear techniques do not outperform PCA on natural datasets. They also separate the DR techniques into full spectral techniques, sparse spectral techniques and non-convex techniques. The main weaknesses found are their flaws in objective functions, the numerical problems in eigendecompositions and their susceptibility to the curse of dimensionality. In a more recent review [8], both linear and non-linear DR techniques are analyzed once more. In this review all the most recent DR techniques are enlisted as well and a selection is given for what techniques suit best to what kinds of data, together with an investigation of open issues in the field.

In a literature analysis by Sacha et al.[9] the visual analytics of visualization literature is systematically studied to investigate how analysts interact with automatic DR techniques. They claim for a DR technique to be useful in exploratory data, the techniques need to be adapted to the human needs and domain-specific problems on-the-fly, preferably including interaction. They conclude that there are seven common interaction scenarios that are amenable to interactive control such as constraints, selection of features or the option to choose from several different DR techniques. Following metrics, the state-of-the-art as of today has been achieved by work of Van der Maaten et al.[2] where they introduce a new variation of Stochastic Neighbor Embedding called *t-SNE*, which, compared to others, is easier to optimize and reduces the tendency to crowd points to the middle of the projection. However, this technique still does have some disadvantages for achieving some ideal requirements. Most traditional DR techniques do have some disadvantages in reaching ideal requirements. Espadoto et al.[3] list these requirements to be achieved as follows:

1. Quality (C1). The DR technique should provide good results as measured by several well-known metrics in DR literature. These metrics should quantify how close its projection

should reflects the structure of its intrinsic dimensions.

2. Scalability (C2): As datasets become large and samples become rich, fast convergence becomes a challenge. Ideally, we want to be able to interact with the algorithm in real time so that the speed of our understanding is not obstructed.

3. Ease of Use (C3): Many DR techniques are hard to optimize as they have many parameters. Reducing or, ideally, removing parameters in an algorithm makes for a better overview, and thus an easier time to understand. The DR method is preferably implemented in only open-source infrastructure so that it is easily replicable.

4. Genericity (C4): Being able to handle any kind of high-dimensional data makes for better understanding of the technique itself, rather than its data-dependency.

5. Stability and out-of-sample support (C5): Being able to handle high variability within the data without creating distortion in its projection. For example, having a technique that is deterministic makes for easier understanding of how the algorithm works.

There are many state of the art Dimensionality Reduction techniques, like t-SNE[2], PBC[10], IDMAP[11] or UMAP[12], which are top-quality (C1), but most of them have problems on some of the other criteria, as shown in a recent survey by Espadoto et. al[13]. For example, t-SNE and IDMAP are very slow on larger datasets (C2) and setting the parameters for t-SNE and UMAP is notoriously complicated (C3). PBC is to some extend data dependant (C4) and t-SNE is very bad at stability and has no out-of-sample support (C5).

## 2.2 Neural Network Projection

This section will introduce a deep learning approach to the dimensionality problem, called Neural Network Projection.
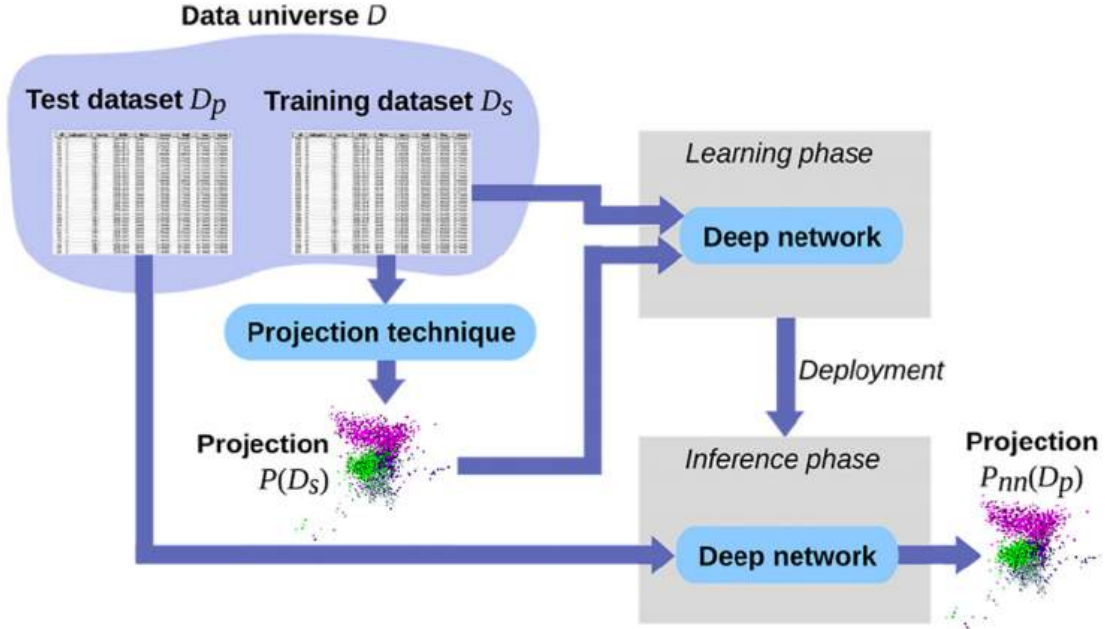


Figure 2: Pipeline of how NNP learns the structure of a projection. Image taken from Espadoto et. al [3].

Neural Network Projection[3] was developed following the observations listed in previous Section 2.1, as there is no technique that covers all five criteria reasonably well. It considers a data

universe, which is the union of all datasets created by a given application area. If some underlining structure exists within this data universe, its structure can be learned and a projection should be able to show this structure. This fits perfectly in the area of machine learning, and hence the use of a neural network to learn this underlining structure. To train the neural network, a subset of the training universe is taken as a training dataset and their respective projections computed by *any* DR technique are used as input for the network, as shown in Figure 2. The results produced by NNP can be seen in Figure 3.
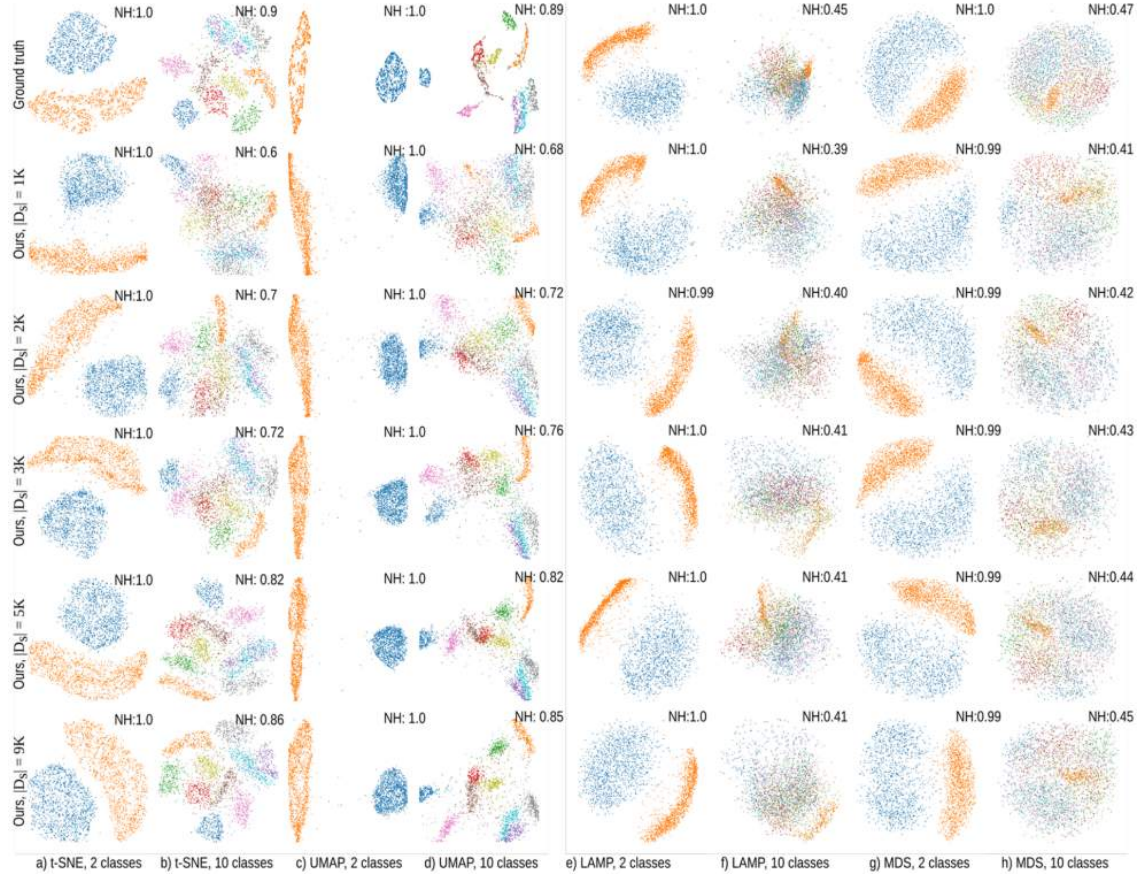


Figure 3: Top row: MNIST dataset, 10K sample projections of 2 and 10 classes created by t-SNE, UMAP, LAMP, and MDS. Next rows: projections done by NNP for varying training set sizes $|D_s|$. Image taken from Espadoto et. al[3].

NNP brings several advantages compared to using traditional DR techniques. It closely resembles the quality of the DR technique to learn as the learned projections look very close like the projections computed by the DR technique itself. The speed at which NNP can predict points is in real time as, after the model is trained, it is merely a simple function inferring data rather than an algorithm needing to do large computations on whole datasets. So especially on very large datasets, NNP becomes *multiple orders of magnitude* faster than the original DR technique. To utilize NNP, one only has to choose the number of epochs to train on, which could also be circumvented using early stopping, making the method very easy to use. The method is not restricted to specific DR techniques or number of dimensions in the data, so the projections themselves can be seen as a black box, enabling one to approach the underlining structure from many angles. Last but not least, the method is deterministic, making observations more thoroughly represent the underlining structure as, mathematically, ambiguity plays no role in our investigation.

The original authors of NNP have tested its stability by using classical train versus test sets

methodologies. And while these are fine for testing stability on classifiers, they are not sufficient for a regressor. We want to understand how a function changes upon perturbations on the samples. They have shortly touched the exploration of generalization through the use of different test samples, being very different from the training data. From this they found that degradation of the projection does happen, but without a clear idea at what point it happens precisely and how.

As NNP uses a neural network, finding the perfect structure of this network is a big task on its own. In more recent work, Espadoto et al.[4] tunes and optimizes the network on several aspects to reach better results measured by the five established metrics in DR. He explained and went into depth for C1, C2, C3 and C4. However, for C5, Stability and out-of-sample support, the conclusions were still a bit vague. The paper stated there to be a degradation point through an experiment in which a model was trained on one dataset and used to infer on another. It concluded that after some re-training iterations the model vaguely relearns to project the new unseen dataset. However, Stability and out-of-sample support can be explored more ambitiously, as they did explore the change in universe, but not the change in samples. Precisely what this paper will explore.

### 2.2.1 Optimal NNP configuration

The in-depth study[4] towards optimizing the quality and stability of NNP was explored through the design space of six directions: training-set size, network architecture, regularization, data augmentation and loss functions. These directions were all sampled using grid search for several settings. They were then compared to each other using the DR technique t-SNE. This comparison was done using four quality metrics, Trustworthiness, Continuity, Neighborhood Hit and Shepard diagram correlation, and through visual inspection. The network is trained on 9000 randomly chosen samples.

The optimal settings found to use for training NNP uses no regularization, the ADAM optimizer, *Noise after* data augmentation with a $\sigma = 0.01$, the MAE loss function, and a *Large - Bottleneck* architecture, consisted of hidden layers with respectively 600, 240 and 600 units.

## 2.3 Visualizations

This section will explore various visualizations for exploring the various aspects of machine learning and deep learning, so that users will easily understand what the data has to say.

In the process of developing a deep learning model, one can detach multiple different phases. A typical workflow is explained by a survey published by Garca et al.[14]. Within this workflow, three different ways of exploring improvements for a model are classified: Architecture Understanding (AU), like TensorFlow Graph Visualizer[15], analyzing *how* the network architecture might be updated to increase performance. Training Analysis (TA), like in BIDViz[16], used to understand *why* training did not perform as expected. And Feature Understanding (FU), like Zeiler and Fergus[17] used to understand *which* aspects of the input data affects the quality of the learning process from the highest level.

Figure 4: Workflow image taken from by Garcia et al.[14] including Data Understanding (DU), exploring the effect of changes to to the data.

Our proposed method will somewhat resemble FU, since we want to understand how an end-to-end trained model operates, but the described workflow is suited for a classifier and not for a regressor. In our method, we do not care about the effect of individual features on the network's
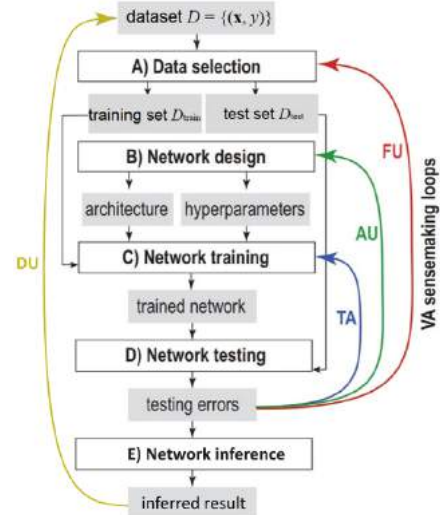
output. Our focus lies on understanding inference. Understanding how changes to dimensions of the data affect the output. Therefore, we propose an updated workflow that includes understanding of changes to the dimensions of the data through exploration of the inferred output, or in short *Data Understanding* (DU) as shown in Figure 4. In our case, this would be from perturbing the data and looking at the change in projection to get a better understanding of how robust our model is. The survey explains a lot of visualization methods for deep learning models, but does not explain a method for our cause, which is why we will provide our own methodology to explore the generalization of a neural network with the use of visualizations.

The choice to use visualizations in our method has many reasons. Humans are very good at understanding visual stimuli, hence the reason why a visualization will create better understanding of results rather than raw data values. Especially once we have lots of data to analyze at once, a visualization can give a good comprehension of what the data has to say. A whole field of exploratory tools exists to give understanding to specific scenario's. Like so, there exists quite a selection of visualizations for neural networks, like for example heat maps[18] or more specialized segmented heat maps[19], general loss landscapes[20] or specifically for generalization[21] and even showing exactly how a classifier came to its predictions[22]. Such visualization yield a faster understanding of what the network learns, providing a feeling for how certain models behave under specific conditions.

However, mere imagery of chosen scenario's may not contribute as much as making a user itself choose what is to be visualized. This is why interaction with visualizations could yield an even greater understanding of what neural networks do. Especially as users could have different goals in what they wants to learn. Therefore, many interactive tools exist to explore the black box of deep learning. The amount of interactivity can also vary depending on the goal of the tool. Some tools let the user choose on what data a trained model needs to run, like for example on what slices of the brain a trained model needs to predict Alzheimer [23]. Some tools give interactivity on the training of a model itself, exploring how the structure of a neural network[24] or autoencoder[25] changes the outcome or metrics of the model. Other visualization tools explore the output of a neural network, like for example the inner workings and activation patterns of a Convolutional Neural Network (CNN)[26, 27]. More concrete interactive tools exist to shed light on a specific algorithm like Principal Component Analysis (PCA)[28].

T. Modradowski[29] has build an interactive tool to visualize NNP. This tool has pinpointed and applied fixes to reduce diffusion of the projection. Her thesis has explored the network through visualization and improved it by applying a nearest-neighbour approach to improve the quality of deep learning projections. This already shows the potential of visualizing NNP, albeit for optimizing quality through metrics, whereas the main focus of this thesis is for exploring its generalization.

The type of visualization matters for the information that is to be communicated. The proposed tool will have a way to show trends of perturbations, thus the use of a trail interface comes up. This interface will somewhat look like the vector map created by Cantareira et al.[30] as shown in Figure 5.A. In our proposed tool, the amount of change will probably not be showing that much, but it gives an indication of what the trails could look like in extremes.

Another way to express trends is with the use of heat maps. In a single image, the amount of change is easily understood as areas with a lot of movement will be coated accordingly, as done by visualizing neuron activation while engineering deep learning architectures[14]. Then there is still the option between a number of different heat map types [18, 19, 32]. In our proposed tool we will want to create a single image showing the entire space of change at once. The main purpose of this heat map will not specifically be to pinpoint the samples that move a lot, but preferably show the area over which movement occurs through perturbations. Therefore, a heat map like shown in Figure 5.B created by Saunders et al.[31] will be used to show the space of change in a single image.
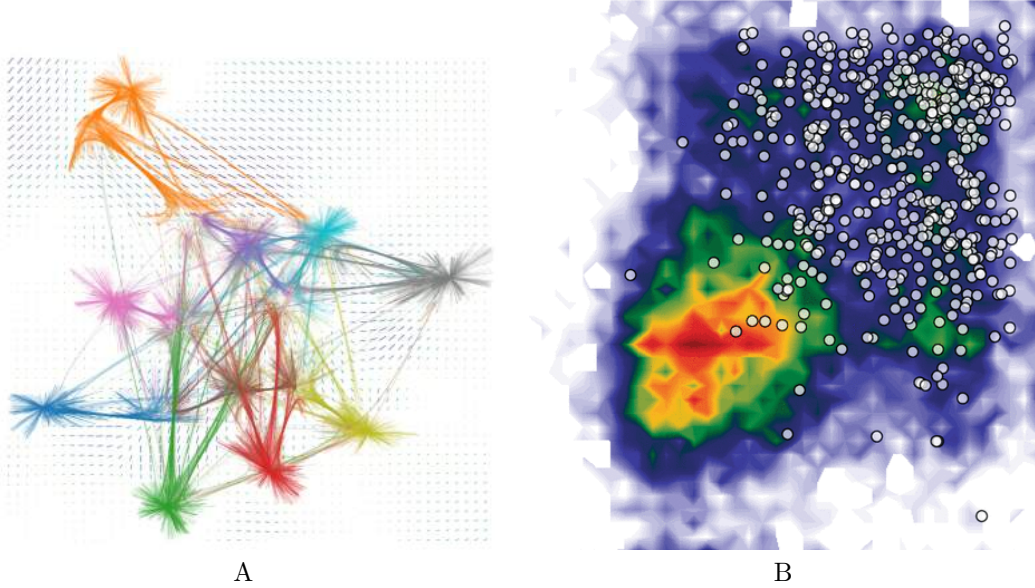
11

<center>A                          B</center>

Figure 5: An indication of what the proposed tool's interfaces will look like. The left image, taken from Cantareira et. al[30], shows a vector map yielding a displacement map and a trail map, showing the movement of each individual sample. On the right image, a heat map, image taken from Saunders et. al[31], showing the averaged movement in space.

In short, exploration through visualization, whether with or without interactivity, gives better understanding on the inner workings of a model. Most of the previously mentioned tools are built to enhance the understanding of neural networks. However, what none of these tools explore is the understanding of the generalization of neural networks. One cannot easily find out *when* or *how* degradation happens when a trained model is used on data being offset by perturbations. Thus, our proposed tool will explore generalization through interactive visualizations.

## 2.4 Improving generalizability

This section shortly mentions some research that has been working towards optimizing the generalization of neural networks.

The exploration of generalization has not been explored yet through an interactive tool, but it has been studied upon. These studies mostly focus on improving generalization through the optimization of metrics. Novak et al.[33] explores two metrics of complexity related to input perturbations. They survey thousands of models with fully connected architectures through measurement of the norm of the input-output Jacobian of the network, which correlates well with generalization. They find that full-batch training or random labeling correspond to a lower robustness, while data augmentation and ReLu non-linearities improve generalization.

Yang et al.[34] propose two ideas on improving generalization. On is to apply the network's output sensitivity to measure diversity at the inputs, so that the best can be chosen from a pool of trained networks. The other is to use a learning mechanism that combines two good networks using complementary weights. Their results show an improvement in generalization performance over the individual networks or ensembles that average any combination from the pool of networks.

Gong et al.[35] introduce *MaxUp*, where they generate data augmentations with random perturbations and transformations while minimizing the worst case loss to implicitly introduce robustness regularization against random perturbations, improving the generation performance.

<center>12</center>

These studies do find positive results, albeit by following metrics rather than exploration through visualization. And although visualizations might be open to extracting wrong conclusions, metrics might be even worse at extracting whether a model is good at predicting faraway data. To our knowledge, inferred data has also not been used to research generalization yet, even though it is much simpler to do through interactive visualization. Therefore, exploring visualizations through projection with our eyes instead of metrics could make way for better understanding, which in turn makes way for an improvement on generalization. Because our methodology is not covered by existing work, we will do precisely that in this work.

## 2.5   Conclusion

In this section we will shortly conclude what has been found in related work and what should be done in this thesis.

We found that there are many state-of-the-art techniques that have top-quality results, but there has not been any optimal method that covers all five criteria as listed: Quality, Scalability, Ease of Use, Genericity and Stability and out-of-sample support. NNP is specifically developed to find a solution that covers all of these criteria, coming very close to such an optimal method.

NNP's coverage of those criteria is well demonstrated in quality, by reaching almost the same quality as any of the trained-on techniques. Due to NNP's linear time dependency, it does very well on Scalability. Inference using NNP is easy to understand, making it Easy to Use. And due to NNP being able to work with any data that can be represented as a high-dimensional data vector, it is also generic in its use.

However, while NNP is deterministic, what has not been properly demonstrated nor investigated is the Stability and out-of sample support of NNP. Which is why this thesis will dive into this problem through the use of various visualization techniques, as they seem to be the best way to intuitively explore generalizability.

# 3   Methodology and tooling

In this chapter, we will describe the methodology that will be used to answer the research question. In Section 3.1 we will start by describing the various notations that are used throughout the thesis. To then also encapsulate what is meant with change, we will describe its taxonomy in Section 3.2. Then in Section 3.3, we will describe the model itself and how it will answer the underlining questions. Then, in Section 3.4, all the implemented perturbations are described. And lastly, in Section 3.5, all the implemented visualizations and configurations for them will be explained.

## 3.1   Notations
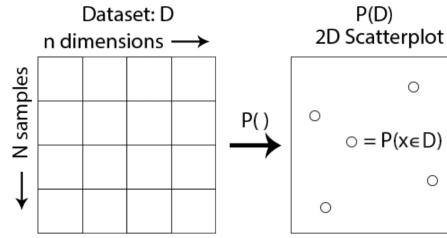


Figure 6: A Projection of $\mathbb{R}^n$ data to a $\mathbb{R}^2$ scatter-plot. Image taken from Espadoto et. al[3].

To first specify our data, let $D = \{x_i\}$ be a dataset of $N$ samples so that $1 \leq i \leq N$, where $x = \{x^1, \ldots, x^n\}$, $x^i \in \mathbb{R}$, and $1 \leq i \leq n$ is an $n$-dimensional sample. Thus $D$ has $N$ samples with $n$ dimensions or $N$ rows with $n$ columns.

$$P : \mathbb{R}^n \longrightarrow \mathbb{R}^m \tag{1}$$

Here, $P$ is the NNP function where $m << n$. The function, maps each $n$-dimensional sample to a lower dimension, often $m = 2$ to make the output $2D$, so that each sample $x \in D$ is projected as a 2-dimensional point in $P(x)$. Each point is then grouped into a projection or scatter-plot $P(D)$, as visually explained in Figure 6. We will call $\Delta D$ a perturbation of $D$, where $\Delta D = x_i + \Delta x_i$, $x_i \in D$ and where $\Delta x_i \in \mathbb{R}^n$ is a (typically small) change vector applied to the point $x_i$. Then, we will let $D' = D + \Delta D$ be the perturbed dataset $D$ by perturbation $\Delta D$. Then, let $P'$ be the result of the projection of this perturbed dataset: $P' = P(D') = P(D + \Delta D)$. With these notations we can come to the notation $\Delta P = P' - P$, which we call the perturbation of the projection.

## 3.2   Taxonomy of change

Before we want to start laying out a roadmap for our problems, we need to know what exactly we want to accomplish. Change has been studied a lot using terms like stability, out-of-sample or genericity. However, these terms are types of change. In the new area of NNP we still have very limited understanding of these different types of change. So what exactly do we mean when we use the term. To be precise in our research, a clear taxonomy of *change* is to be explained.

For change we denote the entire spectrum of data being different from its original. The same term also encompasses change in projections, where any projection being different from its original is considered changed. Such a large term can sometimes be too broad to use, so we split and use the term *perturbation* in cases where we apply a change to the test data, which may yield changes in the projection. Such a perturbation can be applied to a wide range of sample details.

Moreover, these changes could be different to every sample $x_i$, where some samples gets shifted, some others remove some dimensions and even more others are scaled down. Technically we could

even change $\Delta P$. However, that would not be helpful, because the key to interpreting $\Delta P$ is that one knows the change we bring through perturbation $\Delta D$, and we want to study this change by analyzing $\Delta P$. By construction $\Delta D$ has some coherence, as we know the change $\Delta x_i$ that we bring to the samples $x_i$, which we can consider as our ground truth. From these changes we examine $\Delta P$ to see if the coherence instead shows up in the projection. Ideally, we would see this coherence, as we applied some change to $D$. But practically there will be more or less coherence in $\Delta P$, as some samples will move a lot in $P'$ while others move only a little to the same change. Observing this change is exactly what lets one interpret how *generalizable* $P$ is.

Perturbations create a new dataset $D'$ from the original dataset $D$, creating a new projection $P'$ in the process. There are many ways to explore how $P'$ differs from $P$, through many different perturbations. But for this, we will need some way to explore the space of change without having to generate all possible perturbations to draw a higher conclusion as to when and how the data degrades. So then how do we approach this problem? If we look at the main use-case of projections we will find it is their ability to render the structure of the data. In many cases, this structure implies that there are some groups or clusters of samples in the data space, which we would want the projection to show us. The so called *cluster separation* property of projections. This will of course only show if there is some cluster structure in $nD$. Therefore we could argue that if a small influence of perturbance largely destroys the cluster separation of the projection, then that projection is very sensitive to that perturbance. Conversely, if the same small perturbance does not affect the cluster separation much, we can argue that the projection is quite robust to that perturbance. With a large influence of perturbance, the data will at some point destroy the cluster preservation. Hence why it will be useful to study at which point in which perturbations their influence will start destroying the cluster separation ability of the projection. We will call this the a *degradation point*, also referred to as *declustering*, which describes the amount of change needed at which patterns of the original projection $P$ are lost. These points will act as boundaries between NNP still being able to correctly cluster the data and where these clusters start to quickly fall apart. Do note that these points are subjective and might be different as per the purpose of the user.

So how do we find this point of degradation? We want to understand how and why projection $P(D)$ changes when we change $D$. By gradually changing the $D$ to be different from the training data, creating $D'$ which in turn creates perturbed projection $P'$. From exploring this through the use of visualizations, we can get insights by literally seeing it. This can explain where change happens, how change happens, how much change happens, whether $P$ is stable or not and more. What we want to discover is how much $P'$ differs from $P$. How does $\Delta P$ depend on the influence $\Delta D$? We want to see what perturbations cause what conversions in the projection, by comparing $P'$ with $P$.

The task of measuring $\Delta P$ as a function of $\Delta D$ is almost the same as computing and studying the derivative of the projection function $P$, as the ratio $\Delta P/\Delta D$ is conceptually speaking the same as the derivative of the projection function $P$. For example, say we have a perturbation $\Delta D = x_i^j + \delta x$, meaning it adds a small constant offset to the $j^{th}$ dimension of every point. Let us then model $P$ as a function of two output variables, say $X$ and $Y$, or the coordinates of the sample in the $2D$ scatter plot. Then, $\Delta P$ can be seen as being a $\Delta X$ function and a $\Delta Y$ function, meaning how much points scatter in the horizontal respectively vertical direction. However, $\Delta x$ is simply how much the points move horizontally, as a function of having shifted them with the value $\delta x$ along dimension $j$. This indicates $\frac{\partial X}{\partial x^j}$, meaning the partial derivative of function $x$ with respect to dimension $x^j$. This could mean that we can actually compute and study this derivative as usually done in math. However, P is a function of $n$ variables, as its input is an $n$-dimensional vector. So in the best case, we would have $n$ partial derivatives of $P$ to study. Although theoretically this can be done, physically this is way too much. It would make the task not only complicated, it would obscure any meaningful insights to the task as well.

To approach this task with more ease and insight we will define a family of perturbations $\Delta D_i$, which will be explained in Section 3.4. We let the user choose a desired perturbation $\Delta D$ and control its influence. We will then compute $P'$ from perturbed data $D'$ for this chosen perturbation. Finally we will visualize $\Delta P$ through the use of visualizations as will be explained in Section 3.5.

## 3.3 Model

Exploring the research topic could be done by changing the test data in-code, as it would provide the same results as with the proposed tool. However, exploring generalization means assessing the results is not binary but rather regressive. Conclusion and discussion will be based on interpretation of visualizations, which will largely be subjective. Therefore, one will want to create an environment that is least prone to this subjective judgement. To accomplish this, the environment should easily compare or distinguish between raw test data and perturbed test data. Being able to asses multiple visualizations in quick succession or perturb the data in real time will most likely give the strongest indication on how the inferred projections from NNP change due to perturbations of the input data.

When perturbing the test data, many different perturbations will be explored to strengthen conclusions on change to the test data in general. Many different perturbation will need to be compared with each other to achieve such strong conclusions. A tool will provide quick interaction of many perturbations, both isolated or in combination. This will quickly give a feeling for how the data behaves under the various changes applied to them without much overhead distorting this feeling. This is real valuable and a big reason why the tool is created to accompany the proposed methodology.

Visualizations will help in finding aggregated results through the various perturbations. Being able to quickly switch between various configurations, such as line thickness, color or opacity, without having to recompute whole datasets will help in not distorting this feeling through the overhead of long computation. It may also show correlations expressed through different visualizations, indicating the same internal robustness or degradation. In short, the methodology will support users in their exploration for points of degradation without having the problem of overhead distorting their search.

The following sections present many different perturbations and visualizations, but only a subset of these might be used in finding our results in Chapter 4. This does not mean some of these perturbations and visualizations are useless, but they are then proposals to be further explored by the user. As such, a study of which are best to explore which and the amount of influence of each perturbation, is part of future work.

## 3.4 Perturbations

Many perturbations are implemented to see the effect of a trained NNP model on perturbed test data. These are still a mere subset of what is possible, as we can not explore, implement and test an unbounded number of possible perturbations. These chosen set of simple perturbations shed light from different angles so that we can see how the model performs under various different scenario's instead of looking at the entire derivative. In this section we will explain various ways to perturb the data in order as shown in Figure 7. Both how and why they are implemented will be touched upon, together with its possible pro's and cons.

### 3.4.1 Translation

This perturbation will add a constant value to a randomly chosen percentage of dimensions, applied through Equation 2, where $y \in \mathbb{R}, y \in (0, 1)$, and $k_i$ is either 0 or 1, randomly chosen so that
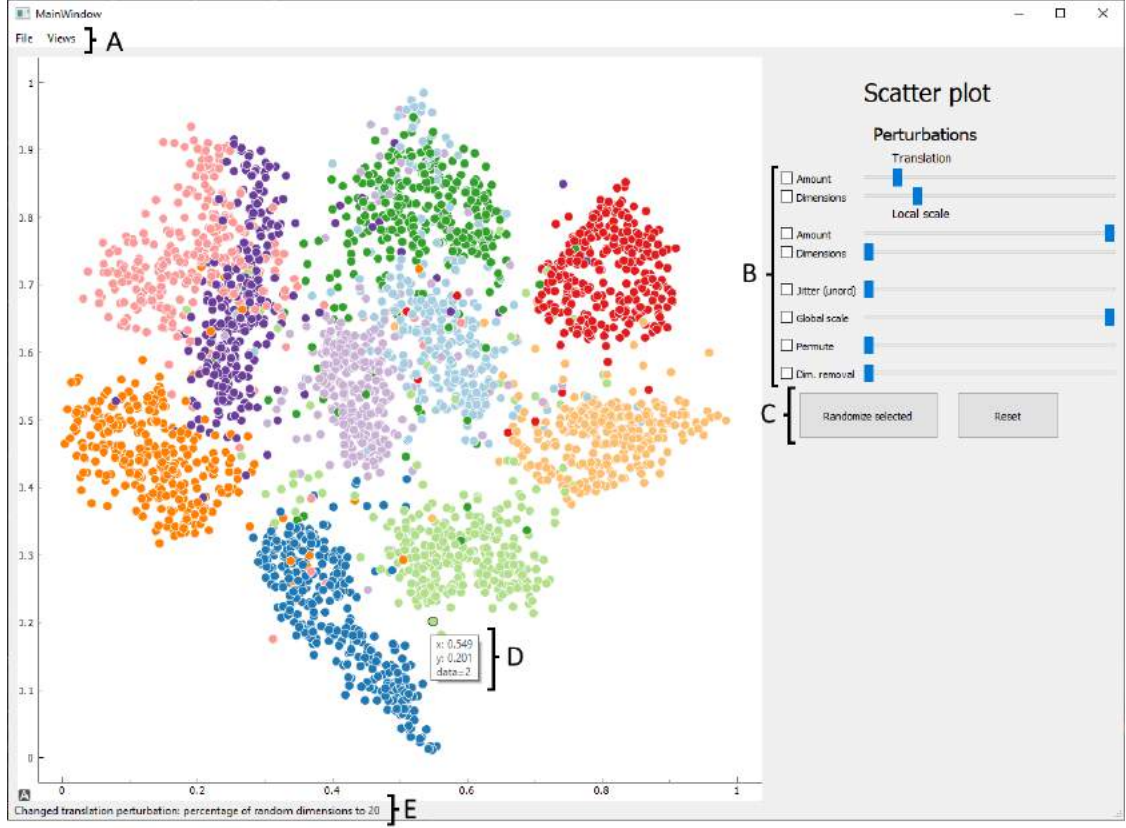
Figure 7: A snapshot of the tool, with navigation option to the different visualizations (A), a various range of perturbations (B), buttons to perturb or reset a multitude of perturbations (C), hover information (D) and a status bar (E) giving explanation or indication of the tool's inner workings.

$\sum_i \frac{k_i}{n} = F$, where F is the fraction, in the range $(0, 1)$, of dimensions influenced.

$$\Delta x_i = \{k_1 + y, k_2 + y, \dots, k_n + y\} \tag{2}$$

This will give greater importance to these influenced weights, thus revealing how the model reacts to dimensions that are increased in value. The biggest advantage of this perturbation is applying more emphasis on some weights compared to others, showing how the network behaves under these circumstances. A disadvantage of this perturbation is that the perturbation does not look at the dimensions' weights already in place, it introduced new false information. Rerunning the perturbation on the same configuration will have a slightly different result as this perturbation, and many other perturbations that choose random dimensions, is of a probabilistic nature due to the random factor being involved, so its results have to be interpreted with that in mind.

The perturbation comes with two sliders, *Amount* determines the amount of translation added and *Dimensions* determines to what percentage of randomly chosen dimensions this chosen amount is added. Thus changing only one of the sliders will not yield any difference between $D$ and $D'$ yet. When adding a constant value to 100% of the dimensions, the relative difference between the weights stays the same. So due to the normalization, projection $P$ will not be different from $P'$.

### 3.4.2 Local scaling

This perturbation will scale a randomly chosen number of dimensions down to a chosen amount, applied through Equation 3, where $y \in \mathbb{R}, y \in (0,1)$, and $k_i$ is either 0 or 1, randomly chosen so that $\sum_i \frac{k_i}{n} = F$, where F is the fraction, in the range $(0,1)$, of dimensions influenced.

$$\Delta x_i = \{k_1 * y, k_2 * y, \dots, k_n * y\} \tag{3}$$

This is similar to the translation perturbation as it gives greater importance to the untouched dimensions relative to the others, but instead of using addition it uses multiplication. However, these dimensions are influenced by weights already in place. A touched dimensions that has a high value will lessen faster than a touched dimensions with a low value if we talk in absolute terms. This brings the advantage of having perturbations that are relative to the influenced dimensions, whereas translation will blindly change them. Another advantage of this relative change to weight amount is that it only scales true information, thus not introducing new false information. Weights that are zero are not influenced, even when its dimension is chosen. A disadvantage is that the perturbation is limited to only scaling down.

This perturbation also comes with two sliders, *Amount* again determines to what percentage of the initial weight value is scaled and *Dimensions* to determine to what percentage of randomly chosen dimensions this scaling is applied. Just like the translation perturbation, when all dimensions are scaled, projection $P$ and its perturbed $P'$ will not change compared to each other, due to the relative differences staying the same after normalization.

### 3.4.3 Jitter

This perturbation will add separate random values to each dimensions, applied through Equation 4, where $r \in \mathbb{R}$ is a random variable in $(-1,1)$ or when using an aggregating visualization like the star map, a random variable in $(-0.1, 0.1)$.

$$\Delta x_i = \{k_1 + r, k_2 + r, \dots, k_n + r\} \tag{4}$$

Through this salt-and-pepper noise, we will shed light on general robustness, showing how some dimensions might react different to slight alterations. Advantage of this perturbation is that it is the simplest way to see how robust a network is by simply nudging the weights up and down and seeing the effect. Disadvantage is that, just like translation, it does not look at the dimension values in place, introducing false information in the process.

This perturbation is not ordered. There is no relation between the perturbation of one dimensions to the other, each time the data is perturbed with it. For the interactive scatter plot, described in Section 3.5.1, the slider will determine the intensity of the jitter range, and for the aggregation visualizations this slider will determine the number of times the jitter is performed using an intensity of range of $(-0.1, 0.1)$. Therefore, this is a prime subject for the star map visualization as will be further explained in Section 3.5.4 and it would not give as much insight when applying any other visualizations to it.

### 3.4.4 Global scaling

This perturbation will scale all previously explained perturbations. It computes difference between the weights of original dataset $D$ and the perturbed dataset $D'$, as in Equation 5 where $\Delta x_i$ is the change in dataset, $k_{n,o}$ are the original data values and $k_{n,p}$ are the perturbed data values. This change in dataset is then scaled and added to the original dataset through Equation 6, where $\Delta y_i$ is the perturbed dataset after global scaling and $z \in \mathbb{R}, z \in (0,1)$, being the percentage of influence of all the perturbance.

$$\Delta x_i = \{k_{1,p} - k_{1,o}, k_{2,p} - k_{2,o}, \dots, k_{n,p} - k_{n,o}\} \tag{5}$$

$$\Delta y_i = \{k_{1,o} + \Delta x_{i,1} * z, k_{2,o} + \Delta x_{i,2} * z, \ldots, k_{n,o} + \Delta x_{i,n} * z\} \tag{6}$$

Global scaling can be used to look at certain perturbance in more detail. Say one would find that some samples change by a large amount through only a small influence of $\Delta D_i$. This difference can then be scaled down even more through this perturbation, which can even be used as input for the various visualizations. As an example, one would find that the jitter perturbation causes a large difference in sample location in the chosen value range of $(-0.1, 0.1)$, this can then be down scaled using this perturbation. There is no effect of global scaling on permutation or dimensions removal as those work with dimensions instead of amounts as will be explained in their appropriate Sections 3.4.5 and 3.4.6.

### 3.4.5   Permutation

This perturbation will swap dimensions with each other. First we define a function $\sigma(i) : [1..n]->$ $[1..n]$, that swaps the value i with any other value in [1..i-1, i+1..n] in a random fashion. Then, let $x = \{x^1, x^2, ..., x^n\}$ be an $nD$ point and let $j = \sigma(i)$. We can then denote the effect of applying $\sigma(i)$ upon the vector $x$ through Equation 7, where the $i^{th}$ and $j^{th}$ component of vector x are swapped. We can then define a set of swaps start from vector x in Equation 8, where $\{i_1..i_K\}$ are a set of randomly chosen different integers in $\{1..n\}$, so that $K = n * F$, where $F \in (0,1)$ is the fraction describing the number of chosen swaps. Sample $x \in D$ then becomes samples $x' = x_K \in D'$.

$$\sigma(x,i) = \{x^1, ..., x^{i-1}, x^j, ..., x^{j-1}, x^i, ..., x^n\} \tag{7}$$

$$\begin{aligned} x_0 &= x \\ x_1 &= \sigma(x_0, i_1) \\ x_2 &= \sigma(x_1, i_2) \\ &... \\ x_K &= \sigma(x_{i-1}, i_K) \end{aligned} \tag{8}$$

A percentage of the dimensions will be chosen and swapped with other randomly chosen dimensions, excluding ones self. The dimensions to be swapped with is chosen from the whole vector, excluding itself. The big advantage of using this perturbation is seeing if a network only looks at the individual weights or whether there are correlations between the dimensions of samples. As a very simple example in the medical field, what happens if a patients hearth rate and blood pressure is swapped. It can give insight on correlation between these data fields. If there is none, then the resulting projection should not change. However should there be any correlation, where certain dimensions are influenced by other dimensions, then we will see a change in projection.

### 3.4.6   Dimension removal

Dimensions removal offers the user the option to set weights of a percentage of randomly chosen dimensions to zero, as done through Equation 9, where $z$ is either 0 or 1, randomly chosen so that $\sum_i \frac{k_i}{n} = F$, where F is the fraction, in the range $(0,1)$, of dimensions influenced, that make dimensions $(k_1..k_n)$ either active or non-active.

$$\Delta x_i = \{k_1 * z, k_2 * z, \ldots, k_n * z\} \tag{9}$$

Where other perturbation are an example of changed data, this is the equivalent of missing data. Thus, this will give an indication of how robust NNP would be in predicting with less information than it was trained on. Or equivalently, how one can determine to not track as much data as before, due to knowing that NNP can still robustly predict on the smaller amount of data per sample. The advantage of this perturbation is that it does not introduce false information, but merely alters existing data. From this we can quite safely make assumptions which will represent practical use.

### 3.4.7 Perturb all

Each slider has a checkbox added so that they can be selected. The selected sliders can then be randomized or reset using the *Randomize selected* button or *Reset* button respectively as shown in Figure 7.C. This will give the user the option to search through multiple perturbations at once. So instead of inspecting every perturbation in isolation, one can get a feeling for multiple perturbation together. A big disadvantage is that one can not precisely know what perturbation would have what effect, or some selected perturbations together could cover up the effect of both, missing out on information.

## 3.5 Visualizations

In this section we will propose various ways of visualizing $\Delta P$. We want to show the difference between Projection $P(D)$ and projection $P' = P(D + \Delta D)$ of the perturbed dataset. There are several ways to do this, each with their own pro's and cons. Every proposed visualization and its configuration options will be explained on how and why they are implemented, together with what good and bad features they have.

A task bar (Figure 7.A) is added to navigate to each visualization interface. Alternatively one can use the keybindings `Ctrl+1` for the interactive scatter plot, `Ctrl+2` for the trail map interface, `Ctrl+3` for the heat map interface and `Ctrl+4` for the star map interface. A status bar is also present to give the user an indication on what is happening behind the interface itself or any relevant information about a visualization should be shown to the user, as shown in Figure 7.E.

The individual family of perturbations $\Delta D_i$ will yield different effects in different configurations. How exactly it behaves is easiest to follow by literally seeing the effect. Animation through the use of the interactive scatter plot can give a good feeling for its behaviour, while the trail, heat and star map all aggregate results by showing the whole influence range of $\Delta D$. The user can choose which range this is by selecting it with its appropriate radio button. The range over which the results will be aggregated is from zero percent influence to a maximum influence chosen with the slider. The tool will then compute the intermediate data sets from each increment and save this for reuse, so that comparisons between other visualizations or configurations is fast. These intermediate results will only be recomputed when any change is done to any of the perturbation sliders.

As the tool is capable of any combination of perturbations at once, it also gives the option to do this for the aggregated visualizations. One can choose to set certain perturbation influences, while aggregating the range of another perturbation. For example, the user could be interested in the influence of the jitter perturbations while half the dimensions are turned off. In this case, the user sets the dimension removal perturbation to 50%, selects the radio button for the jitter perturbation and chooses the range maximum with the jitter slider before pressing the "Compute visualization" button.

### 3.5.1 Interactive scatter plot

The interactive scatter plot, as seen in Figure 8, is a very intuitive way of exploring how a projection reacts to perturbed data $D'$. The color of the samples are chosen to be far apart for up to 12 classes, where they can be changed to be more friendly towards color blind people. Animation is an easy way for a human to comprehend what information is shown in a short amount of time. Moreover, when a user can choose what precise influence $\Delta D$ will have, while immediately seeing the results in $P'$, he will be able to understand what exactly the perturbation changed. The scatter plot is computed in real time when any perturbation is changed so that instant feedback will give the user a great feeling for the difference between new projection $P(D')$ and original projection $P(D)$ due to the selected perturbation. While animation does show a quick grasp over

|      |      |      |      |      |
|------|------|------|------|------|
| 0%   | 20%  | 40%  | 60%  | 80%  |
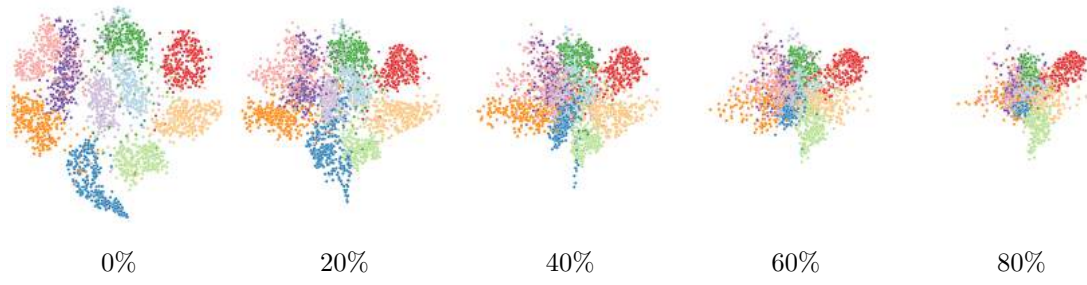
Figure 8: Animation in the interactive scatter plot with increasingly more perturbed data.
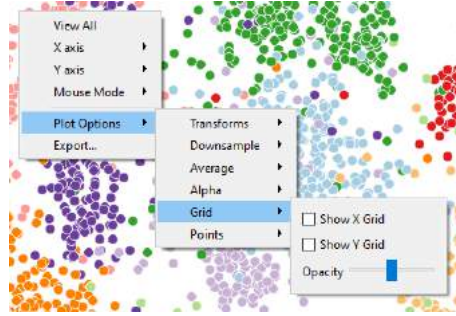


Figure 9: The interactive scatter plot with its configuration menu open.

the influence of any perturbation $\Delta D$, it will not be able to show the overall effect or trend of these perturbations.

**Configuration options**    To inspect further details, the user has some configuration options to move around the plot or zoom in/out using the mouse. When hovering over a sample, information will be given on the location and class of the sample, as shown in Figure 7.D. Furthermore, the user can right-click the plot to have even more interaction with the scatter plot (Figure 9) such as changing the axis or exporting the scatter plot in several different file formats.

### 3.5.2 Trail map
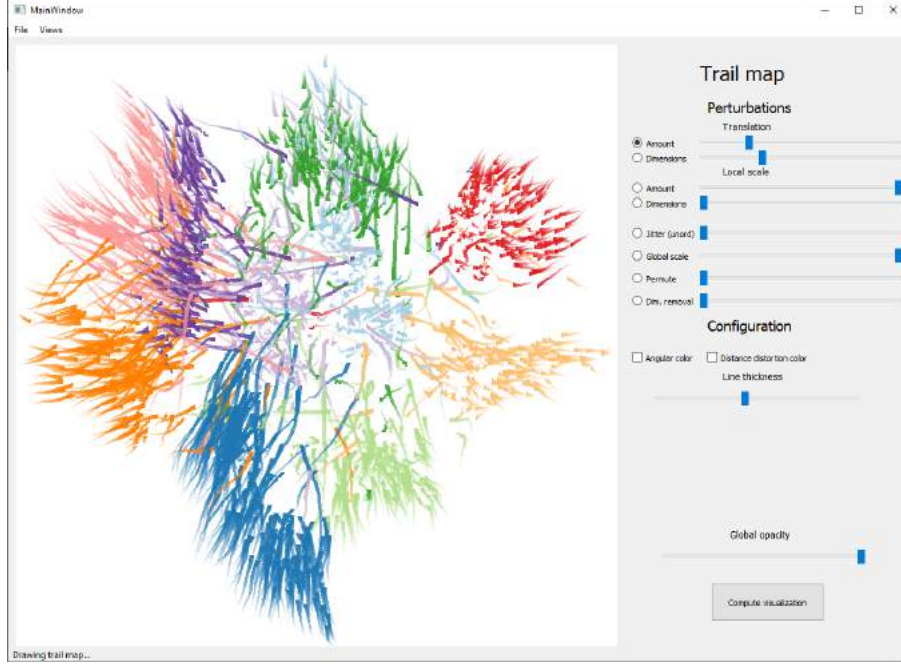


Figure 10: The trail map interface, drawing a path traversed by each samples' movement according the interval range of the selected perturbation.

The trail map (Figure 10) removes the time dimension and folds it into the space dimension, which is better for static imagery. It links the many intermediate results of $P$ and $P_i'$ together with lines, so that through increasing perturbation influence, a line shows the path that the sample has moved over in projection $P'$. This aggregated image will show all the paths as traversed by the samples' movements induced by the selected perturbation $\Delta D$. The samples' path of travel is a very clear way of seeing trends in the overall projection $P'$. It will not only be able to easily show individual movement, but also the movement of entire clusters. A disadvantage is that the screen might become very cluttered through the many lines, especially once they become rather long. This could result in not being able to see the individual lines anymore. This can however be solved by lowering the amount of test samples, decreasing line thickness or global opacity.
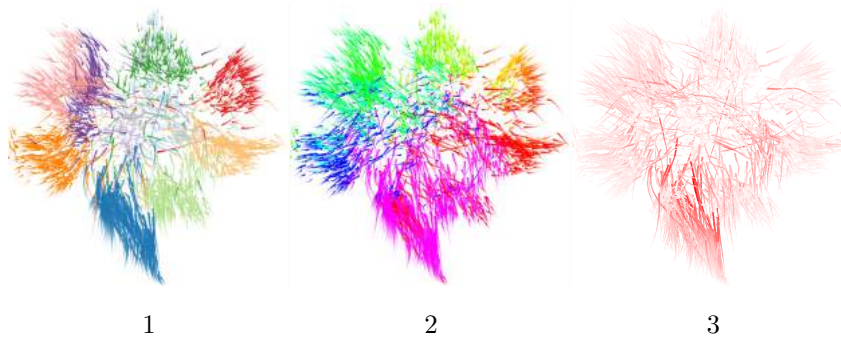


Figure 11: Class color (1), angular color (2) and distance distortion color (3).

**Configuration options** There are 3 options of coloring the trails, as can be seen in Figure 11.

1. Class color, Figure 11.1. The default color, so that the user can distinguish the movement of clusters.

2. Angular color, Figure 11.2. This option opts for coloring the direction of the trail. It computes the angle between this direction and the horizontal axis, and uses that angle to determine the hue of each line segment in an HSV color wheel. This can more easily give an idea to which direction the trails move, and thus what $\Delta D$ causes to the projection.

3. Distance distortion color, Figure 11.3. For each line segment, we compare the difference in $Nd$ and in $2D$. We do this by computing the Euclidean distance of the perturbed sample to its origin, both for the $n$-dimensional input data and for the $2D$ sample location. By then dividing the found $Nd$ value by $2D$ value, we get an indication of robustness for each line segment. For example, if a big change in $Nd$ has only a small amount of change in $2D$, it means the model is very robust. This value is then interpolated from white to red so that we can distinguish the robustness of each line segment compared to each other. In the status bar some statistical information will be shown to give more idea as to what is seen in the visualization.



Figure 12: Gradually increased line thickness.

Furthermore there is line thickness (Figure 12) and global opacity (Figure 13). To present importance of the most influenced sample location and to show the direction of the samples' movement, the lines increase in thickness and in local opacity. However the user may work with a lot of samples, so the option for maximum line thickness is provided. To further increase flexibility of working with different sizes of test data, the global opacity slider will scale the opacity if every samples' trail, so that one can still differentiate between the samples when much overlap is present.



Figure 13: Gradually increased global opacity.

### 3.5.3 Heat map

A heatmap (Figure 14) trades occlusion for aggregation. It is created using the Kernel Density Estimation algorithm[36] through Equation 10, where $n$ is the total number of dimensions, $r$ is the chosen radius and where $K$ is a density which has a max heat of which its threshold $t \in \mathbb{R} \in (0,1)$
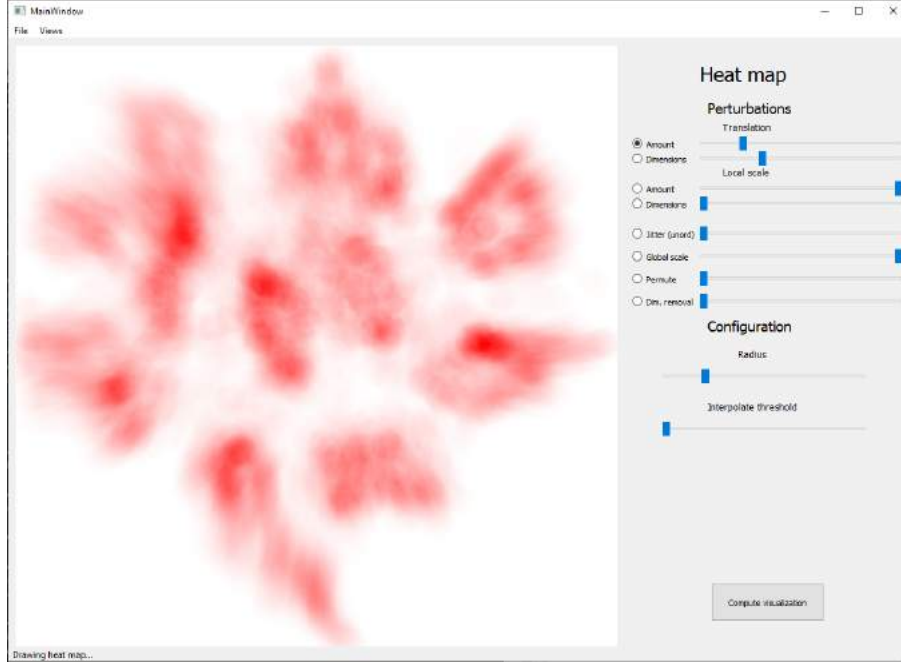
Figure 14: The heat map interface, counting all neighbouring sample points in every intermediate result. The color interpolates from white to red, meaning relatively low to a respectively high amount of movement.

is a fraction of this max, chosen by the user. Then $(0, t)$ maps white to red in a continuous color map, where any value higher than $t$ becomes red.

$$\hat{f}_r(x) = \frac{1}{nr} \sum_{i=1}^{n} t * K\left(\frac{x - x_i}{r}\right) \tag{10}$$

The algorithm walks over every pixel and is given a radius. It counts every other sample point in all intermediate results. It then determines the interpolated value between white and red, having respectively a low to high number of neighbours in its radius. This aggregated result will visualize in what areas most sample movement has occurred compared to each other. This will give the user information regarding in what areas any perturbation $\Delta D$ has the most effect in terms of their projection. Disadvantage of this aggregation is that there is no way of knowing what this movement is, but this is covered by other visualizations.

**Configuration options**    The heatmap is implemented using Equation 10, where K is a uniform kernel, where $r > 0$ is the radius and $t$ is the interpolate threshold. 2 sliders are present to configure the heat map, as seen in Figure 14. A slider to determine the radius in which the algorithm will look around each pixel. The radius can be chosen to a maximum of 50 pixels apart, with a default of 20 pixels. At more than 50 pixels, the area becomes so large that it aggregates too much, making the information not useful and the time it takes to compute very long.

Outliers may draw only some particular areas in red, while blending any other interesting observations. Therefore, an interpolate threshold slider will give the user the option to choose a max percentage of the highest found value to be the ceiling while interpolating the colors. This will keep the areas with the highest values red, but also gives bigger differences in saturation levels in other areas of the heatmap.

### 3.5.4 Star map
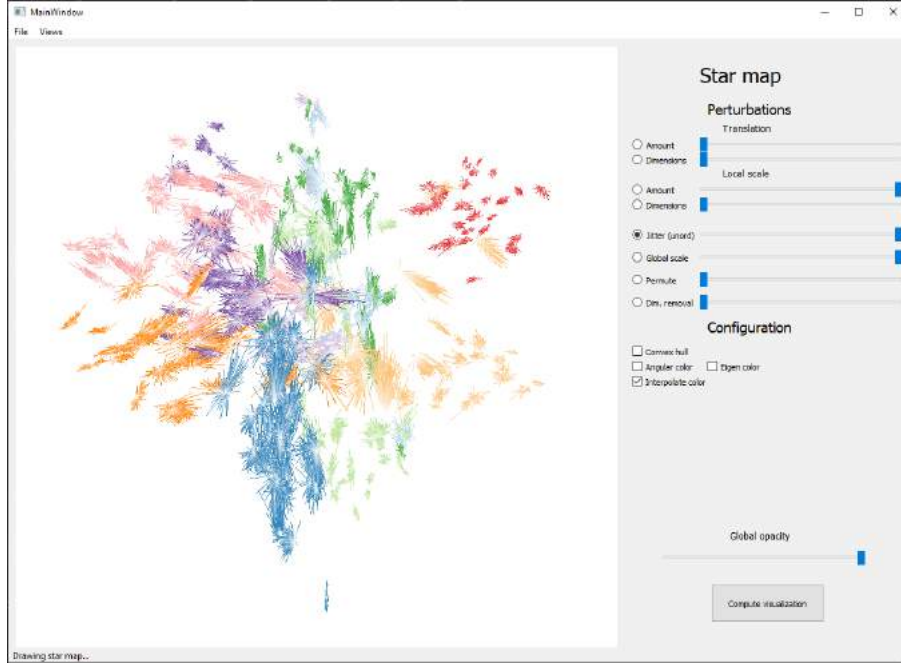


Figure 15: The star map interface, drawing rays from origin to perturbed location, indicating the area in which the point will move after applying the perturbation. A tenth of the number of test samples is taken to clarify what the stars look like.

Where a trail map implies order, a star map (Figure 15) does not. It is a visualization working with non-ordered perturbations, like the jitter perturbation, described in Section 3.4.3. Although every other perturbation does work, the information gathered from those is not as useful as working with non-ordered perturbations. The original location of a sample $P(D)$ on which no perturbation is done, is taken as the origin, or center of a star. Then for every intermediate result, a ray is cast from this origin $P(D)$ to the perturbed location. When doing this for many rays, we will get a star indicating the area in which the sample will move. Big stars will mean the perturbation will move the sample a lot, where small stars say the sample is robust as the sample did not move much. A disadvantage for using this visualization is that it creates a lot of overlap between the different created stars, so a visualization of many samples will create clutter rather fast. But for that, aside from lowering the amount of data samples, we have many configurations to extract information.

**Configuration options** There are 3 options of coloring the stars, as can be seen in Figure 16.

1. Class color, Figure 16.1. The default color, so that the user can distinguish the movement of clusters.

2. Angular color, Figure 16.2. This option opts for coloring the direction of the rays. It computes the angle between the ray and the horizontal axis, and uses that angle to determine the hue of each ray in an HSV color wheel. This can more easily give an idea to which direction the rays are the longest, and thus to which direction the sample gets perturbed the most.

3. Eigen color, Figure 16.3. For every sample and its perturbed locations, the eigenvector and eigenvalues are computed using its covariance matrix, as in Equations 11, where if $T$ is a linear transformation from vector space $V$ and $v$ is a nonzero vector in $V$, then $v$ is an *eigenvector* of $T$ if $T(v)$ is a scalar multiple of $v$. $\lambda$ here is then known as the *eigenvalue.*
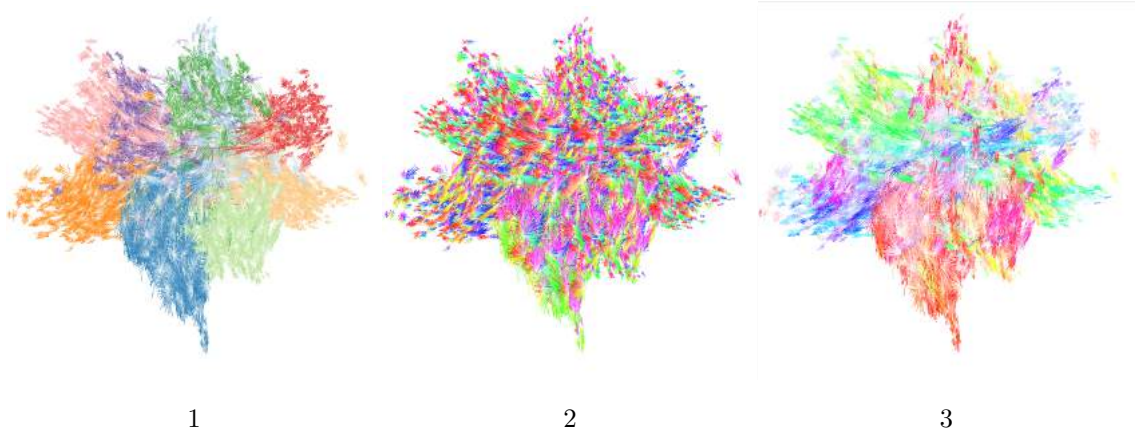
Figure 16: Class color (1), angular color (2) and eigencolor (3).

With this information, the color is determined using the hue from the direction of the longest eigenvector and its length to determine saturation. This will give indication towards which direction the sample moved the most and how much it moved in that direction compared to the other stars.

$$T(v) = \lambda v \tag{11}$$
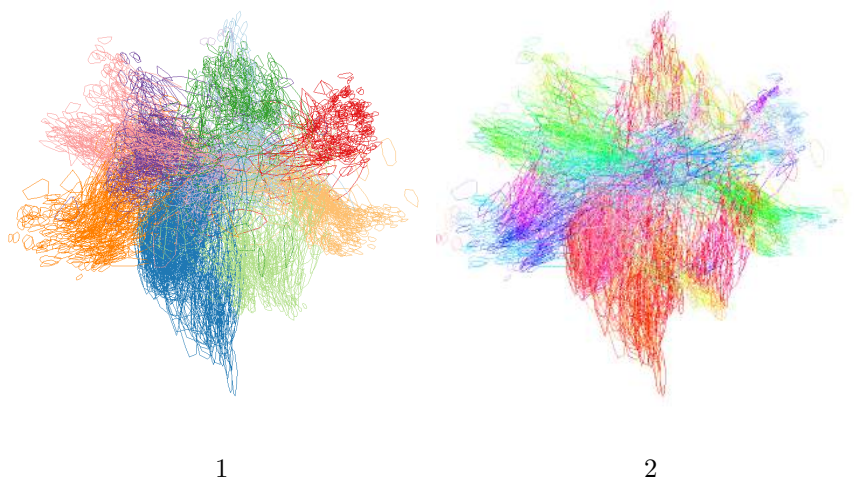


Figure 17: Class color (1) and Eigen color (2) for the convex hull.

The option is given to draw a convex hull around each star to more easily distinguish the area in which each sample moves due to the influence of the chosen perturbation. This option has the choice between drawing the convex using class colors or Eigen colors as seen in respectively Figure 17.1 and 17.2.

To create an idea on where the origin of the star is, the option to interpolate from white to the selected color is present. One can also more easily distinguish each star from each other, be it within or between sample classes.

Overdrawing becomes a big reason to use the global opacity slider so that information is more easily extracted from what is presented by the visualization, just like used for the trail map shown in Figure 13.

# 4 Results

In this chapter, we will explain how we will use our described methodology to answer our research question and what results we found. First, in Section 4.1 we will describe a precise roadmap that answered *when* degradation happens. In Section 4.2 we will explain which datasets will be used for our exploration. Section 4.3 will then shortly discuss the projection techniques that we will be using for our explorations. And in Section 4.4 we will describe how to interpret the results we find. In Section 4.5 we will explore the various perturbations and in Section 4.6 we compare and try to generalize our results.

## 4.1 Approach

For answering *when* change happens, we will want to find a point at which the projection quickly starts to decluster. This may happen at different points depending on which data set, projection technique or perturbation is used. Therefore, we should look at them separately and try to understand how the space of change affects the data in general. To do this, we first separately look at the degradation points of each perturbation in isolation to then later try to generalize what we found. All testing will be done using the optimal NNP configuration as explained in Section 2.2.1.

The following roadmap will be used to find degradation points of isolated perturbations. As illustrated in Figure 18, it describes the steps to be taken for our exploration. The time it takes to train a model is somewhat scaled by the amount of training samples, but it won't take much longer than a minute, due to early stopping.
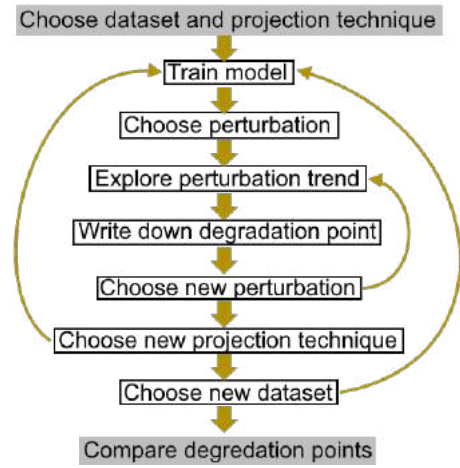


Figure 18: Roadmap to explore the generalization of NNP.

1. We start by loading in a chosen dataset, which will be explained in Section 4.2. NNP will be trained on this dataset with optimal configuration using a chosen projection technique, as explained in Section 4.3, to create projection labels for each samples' multidimensional vector.

2. The user starts to increase the influence of a perturbation so that one gets a good idea of the trend of this perturbation's influence.

3. The user then increases the influence of the perturbation to the point at which major declustering starts. The amount of influence at this degradation point is then written down for later comparisons.

4. The user chooses the next perturbation and starts from step 2 up until all perturbations are explored.

5. A new model is trained using the same dataset, but using a different projection technique so that the technique creates different projection labels. The user then starts again from step 2 up until all projection techniques are explored.

6. A new model is trained using a different dataset and the user again starts from step 2.

7. At this point, all perturbations for each technique-dataset configuration are explored and will now be compared to each other so that we can argue about generalizable change through our visual exploration.

## 4.2   Datasets

In this section we will explain which datasets were used for our exploration process. The datasets we used in our exploration process are the following:

1. MNIST[37], A database of handwritten digits, subset of the larger NIST dataset. It consists of a training set of 60,000 images with a test set of 10,000 images, which have been size-normalized and centered in a fixed-size image of 28x28. 9000 samples will be randomly chosen and used for training NNP.

2. Fashion MNIST[38], A dataset consisting of Zalando's article images. It consists of a training set of 60,000 images with a test set of 10,000 images. Each sample is a 28x28 sized greyscale image, associated with a label from 10 different classes. Just like MNIST, 9000 samples were randomly selected and chosen to train NNP on.

3. Reuters[39], A benchmark dataset for document classification. Consisting of 90 classes and a little over 10,000 train- and test documents, with a vocabulary size of 35.247. From this dataset, we chose the 10 most occurring classes so that we match the other two datasets having 10 classes. Those classes being: earn, money-fx, grain, wheat, corn, money-supply, sugar, gnp, soybean, livestock. Due to this choice we were left with around 4500 documents left to train and test on. However, exploration using this very different type of data and with a smaller amount of training available could give more and new perspectives, and thus a more complete insight into the robustness of NNP.

## 4.3   Projection techniques

To prevent conclusions based on a projection technique level, we will explore multiple projection techniques on which we will train NNP. The chosen techniques are well-known or recent with good quality, as we mentioned in Section 2.1. The chosen techniques we used for our exploration are the following:

1. Principal Component Analysis (PCA)[1]. One of the oldest techniques known. Although its quality is not up to standards these days, it is used throughout many fields and will thus be good for comparison.

2. t-Stochastic Neighbor Embedding (t-SNE)[2]. One of the most well-known techniques for Dimensionality Reduction. Although very slow for larger amounts of data and it being ambiguous in output to the same input based on its stochastic nature, its quality is very good, which is precisely what NNP uses.

3. Uniform Manifold Approximation and Projection (UMAP)[12]. A very recent addition to the Dimensionality Reduction techniques and one that is very much made for DR projections. A technique that is of high quality but very different from t-SNE projections, which is why this is good to compare robustness with.

## 4.4   Result assessment

This section shortly explains how the various results using our proposed roadmap can be interpreted. First, when assessing any isolated perturbation, we could get nine different points of degradation. This could mean that there is no generalizable assessment to be made. If degradation point are nearly the same for the different datasets or trained projection techniques, we can argue there being generalizable outcomes from the used perturbation, but it would then be projection technique respectively dataset dependent. If all nine degradation points are close together, we can assert that the effect of the perturbation has a general effect on the underlining change it has made. We could then argue that the effect is not data dependant and holds for more than only

the trained models.

After having assessed all the perturbations in isolation, we could find patterns between the perturbations. If multiple perturbations show the same degradation points, we could argue that there is some underlining change which has a degradation point of itself. This is the precise answer we are looking for to answer the first part of the research question.

## 4.5 Perturbation exploration

In this section, we will explore the various perturbation in isolation, so that we can understand how inferred projections change based on various changes to the data.

For both the translation and local scale perturbations, we have two variables to explore, increasing the exploration span in the process. Therefore, for the first technique-dataset configuration, we will use incremental and decremental steps of 10% so that we can very closely inspect the change in perturbation $\Delta P$. But in the other configurations we will show incremental and decremental steps of 20%, as the principle of the exploration is clear already. For the jitter, permutation and dimension removal perturbation there is only a single variable, so we will show a single table with the trends for each technique-dataset configuration with incremental steps of 10%.

Each perturbation will first show the trends and then show a comparison between the found degradation points. The image resolutions are kept the same for easy comparison. The point of degradation is chosen when we start do not see the patterns that the original projection had anymore. These points of degradation are highlighted in each table that has increments or decrements of 10%. As this is a subjective task, we will try to be on the modest side, rounding down so that we can draw conclusions without having a bias towards some sort of success.

We will then explore aggregated results through the trail, heat and star map to look at the broader spectrum of the perturbations influence. We will use the trail map to see what path the samples move over. Furthermore, we will use the heat map to see the general influence of a perturbation, or if small areas of the projection are more prone to the perturbation. For both the star map and heat map, we will use a perturbation influence of 50%, as the 50% to 100% influence spectrum will not yield as much interesting information besides the samples degrading into the center. As the heat map shows color in relation to the maximum heat found, we will keep track of this value, shown through the status bar, to use in our findings. These aggregated results will be performed using t-SNE on the MNIST dataset, as the resulting projection is nicely spread out, utilizing a large area of the 2D projection and yielding nicely separated clusters. This will give room for more clear analysis on the behaviour of the model, although the findings could be subject to technique- or dataset specific influences.

### 4.5.1 Translation

**Trend** Influencing 0% of the dimensions changes nothing, and having 100% of the dimensions be influenced changes nothing as well due to the relative difference between the samples staying the same. Therefore, normalization will revert those added values back.

Table 1 shows us the trend for translation with a model trained on MNIST using the t-SNE projection technique. If we look from left to right, we see that the sample clusters first start to get close to each other before starting to overlap and then decluster rapidly. This effect is increased in speed if we influence more dimensions. If we look from top to bottom we see that if only 0.1 is added to the randomly chosen dimensions, not much is happening. However if start to add more than that, we see that declustering happens rather quickly, especially if we influence 40% or more of the dimensions. Overall, the samples almost exclusively move towards the centre.
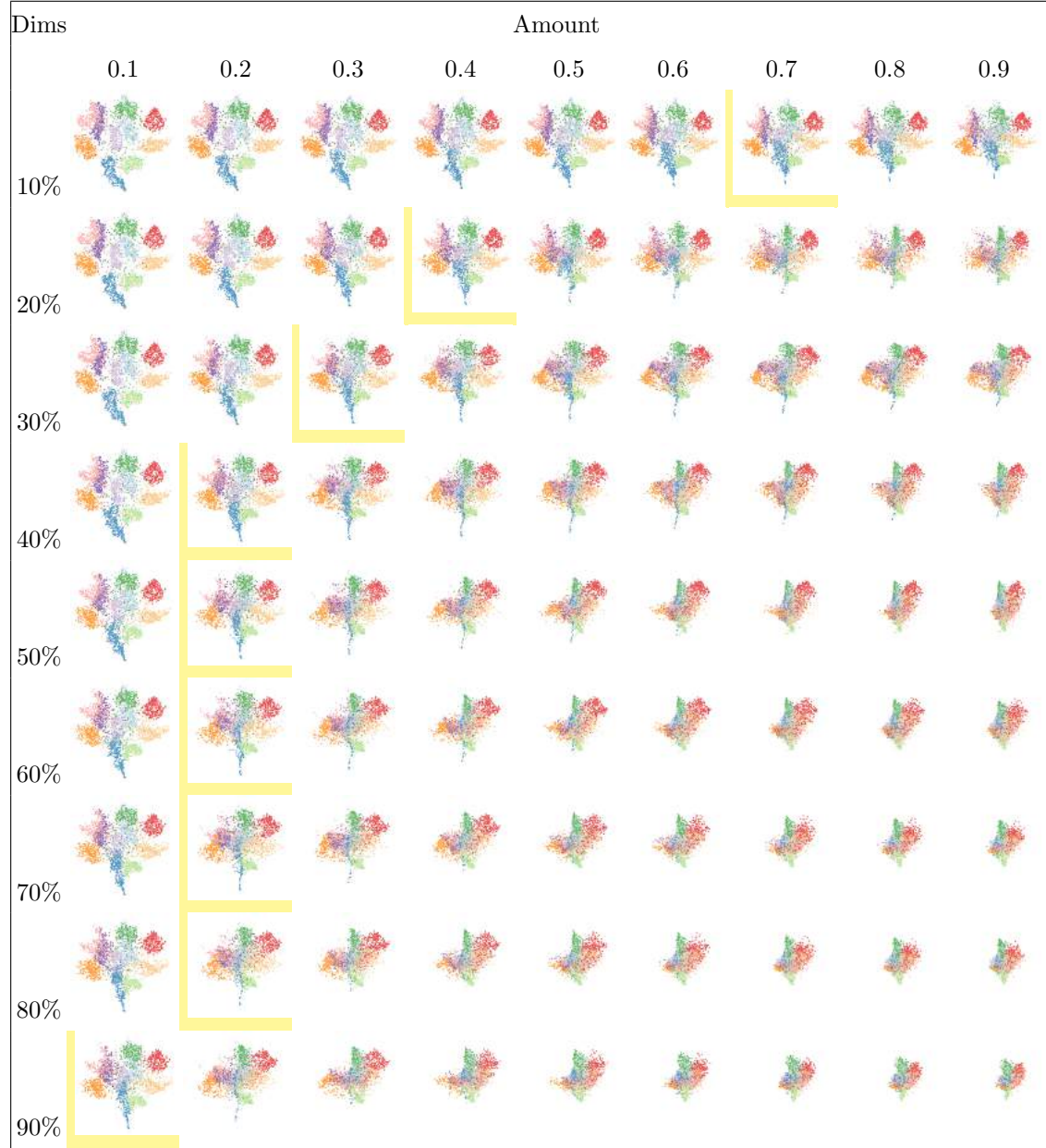
Table 1: The trend for translation. Model trained on t-SNE using the MNIST dataset. Points of degradation are highlighted and rounded to the nearest tenfold.

Table 2 shows the trend for translation of a model trained on the MNIST dataset using PCA as the trained projection technique. From looking both left to right and top to bottom we see the same thing happening, being the samples moving inwards while their relative distance to each other stays exactly the same. It also looks to be scaling at the same rate no matter how many dimensions are influenced. We can also see that if we add a large amount of translation, say 0.8 and higher, the cluster only shifts its barycenter to the right, if we increase the number of dimensions influenced.
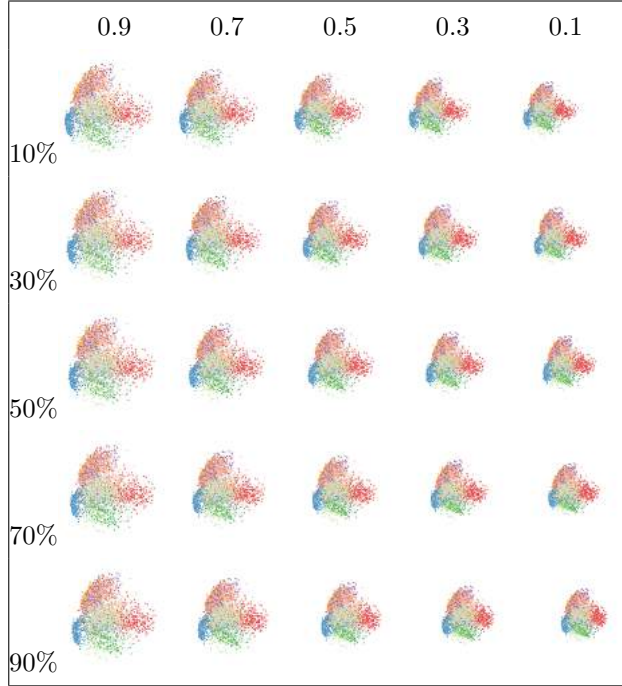


Table 2: The trend for translation. Model trained on PCA using the MNIST dataset.

In Table 3 we see the trend of translation for a model trained on the MNIST dataset using UMAP as the projection technique. Both from left to right and from top to bottom, we see the clusters scattering rather quickly. It seems like the clusters in a projection from a UMAP trained NNP model have a relatively large amount of white space between its clusters compared to when the model is trained on the PCA and t-SNE, like seen in Tables 1 and 2 respectively. However, this white space does shrink faster than with the other projection techniques.
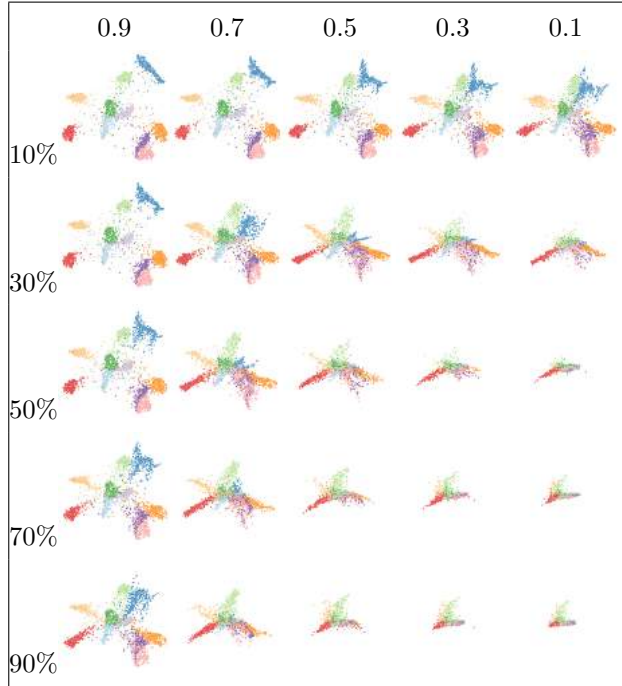


Table 3: The trend for translation. Model trained on UMAP using the MNIST dataset.

Table 4 Shows the trend for translation with a model trained on the fashion-MNIST dataset using t-SNE as the trained projection technique. We see that just like in Table 1, from left to right and top to bottom we see the same kind of degradation, where first the white space shrinks, and when the clusters start to overlap the clusters themselves also seem to scatter.
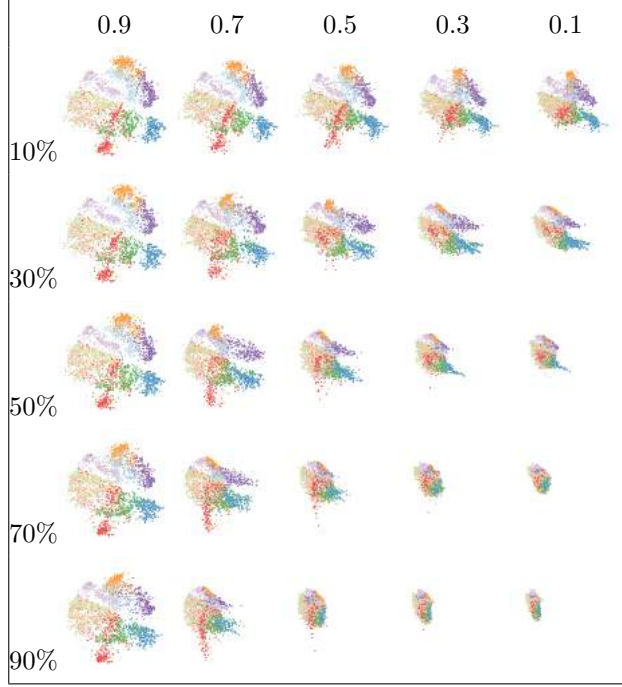


Table 4: The trend for translation. Model trained on t-SNE using the fashion-MNIST dataset.

In Figure 5 we see another trend for translation, with a model trained on the fashion-MNIST dataset with the PCA for the projection technique. And just like the model trained with PCA, but then on the MNIST dataset, like in Figure 2, we see the relative distance between each sample staying the same for however much influence of the translation perturbation. Both from left to right and from top to bottom, the structure of the projection stays the same, where only the barycenter shift if we influence more dimensions.
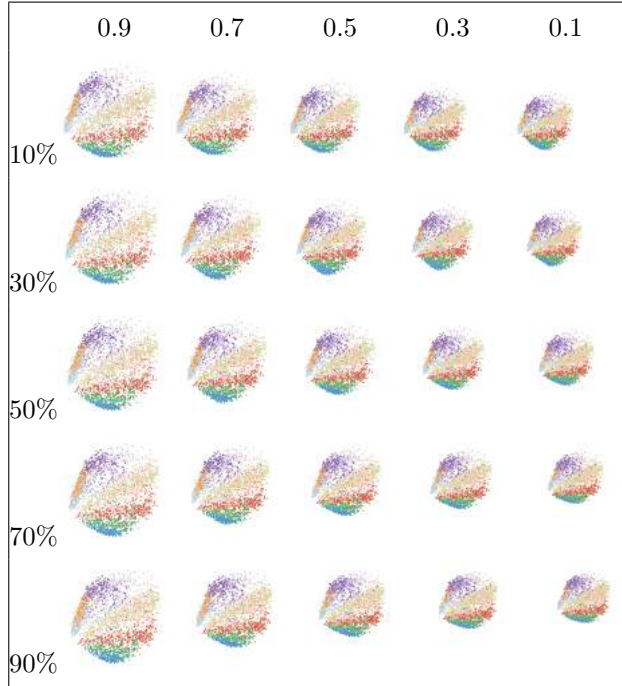


Table 5: The trend for translation. Model trained on PCA using the fashion-MNIST dataset.

Table 6 shows the trend of translation through a model trained on the fashion-MNIST dataset using UMAP as the trained projection technique. We again notice quite a lot of white space between the clusters, where influence of the translation perturbation quickly start to let that shrink. Just like in Table 3, from 50% and higher influenced dimensions an amount of 0.5 or higher, we see a sharp decline in projection quality, or this may just look like that due to the relatively large gaps between certain clusters when compared to the model trained on either PCA or t-SNE.
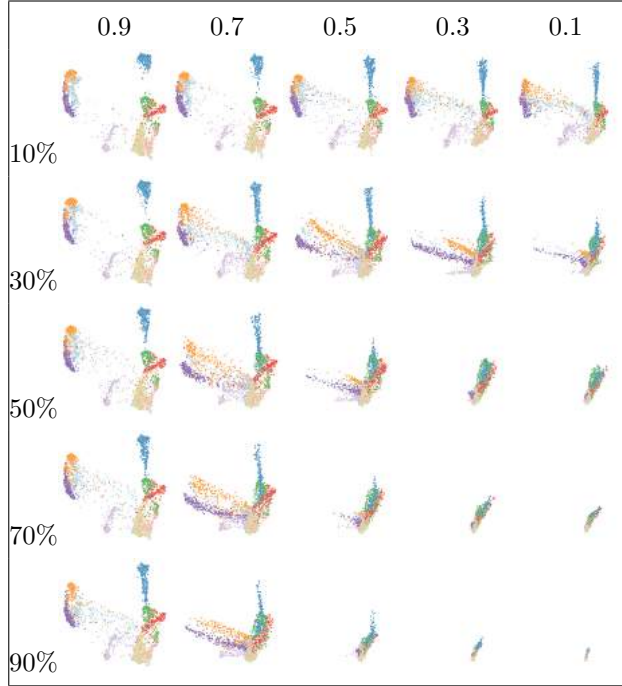


Table 6: The trend for translation. Model trained on UMAP using the fashion-MNIST dataset.

In Tables 7, 8 and 9 we see the trend for translation. This model is trained on the Reuters dataset, where the trained projection technique is t-SNE, PCA and UMAP respectively. In all 3 of these sets of images we see a very rapid decline in projection quality, where not even 10% of the randomly chosen dimensions are influenced or not even 0.1 amount of translation is done before declustering has already happened. Due to this phenomenon, we can argue that the Reuters dataset could be the cause of this rapid declustering.
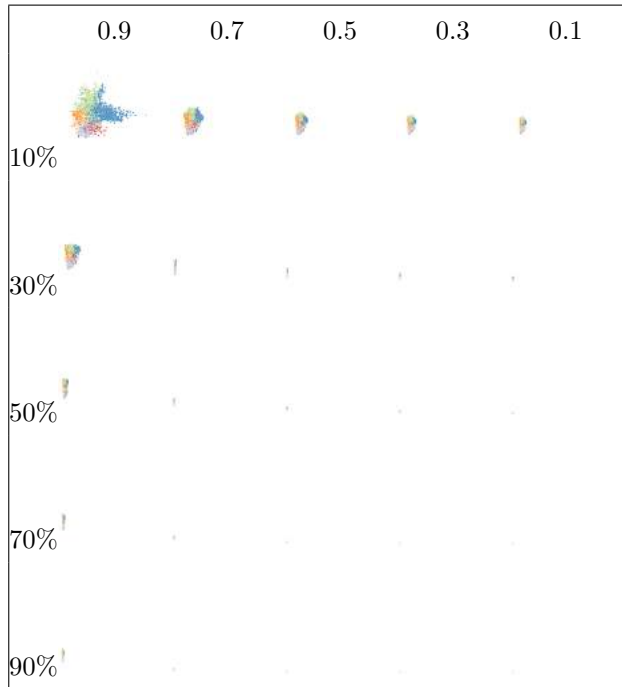


Table 7: The trend for translation. Model trained on t-SNE using the Reuters dataset.
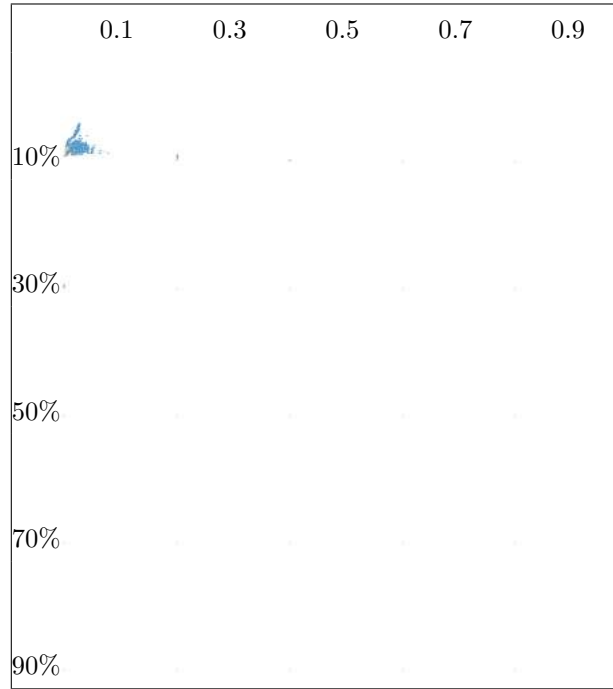
Table 8: The trend for translation. Model trained on PCA using the Reuters dataset.
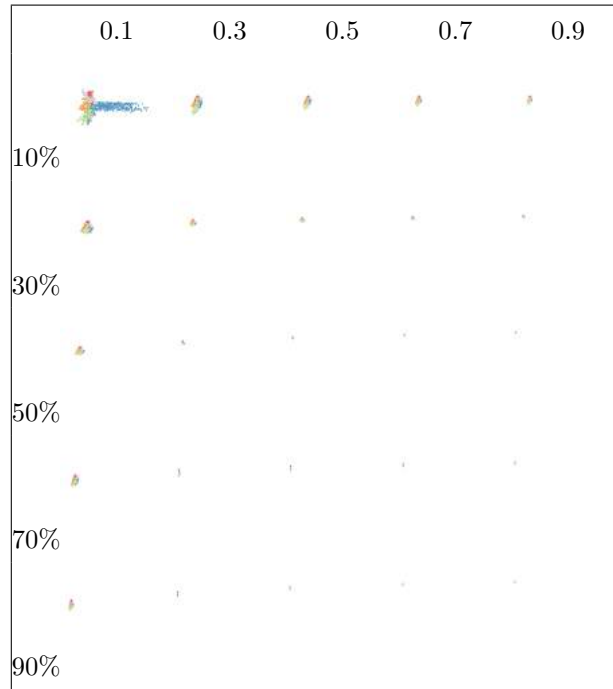


Table 9: The trend for translation. Model trained on UMAP using the Reuters dataset.

**Points of degradation**    In Table 10 we can see a summary of all the degradation points found through the translation perturbation caused by every technique-dataset configuration.

|  |  | 10% | 20% | 30% | 40% | 50% | 60% | 70% | 80% | 90% |
|---|---|---|---|---|---|---|---|---|---|---|
| MNIST | t-SNE | 0.60 | 0.35 | 0.25 | 0.20 | 0.15 | 0.12 | 0.11 | 0.10 | 0.08 |
|  | PCA | 0.70 | 0.65 | 0.60 | 0.60 | 0.60 | 0.60 | 0.60 | 0.55 | 0.50 |
|  | UMAP | 0.50 | 0.40 | 0.30 | 0.25 | 0.20 | 0.15 | 0.13 | 0.12 | 0.10 |
| fashion MNIST | t-SNE | 0.55 | 0.40 | 0.30 | 0.25 | 0.20 | 0.15 | 0.10 | 0.10 | 0.08 |
|  | PCA | 0.70 | 0.60 | 0.50 | 0.40 | 0.35 | 0.30 | 0.25 | 0.23 | 0.20 |
|  | UMAP | 0.50 | 0.40 | 0.25 | 0.20 | 0.15 | 0.12 | 0.10 | 0.09 | 0.07 |
| Reuters | t-SNE | 0.06 | 0.03 | 0.02 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 |
|  | PCA | 0.06 | 0.03 | 0.02 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 |
|  | UMAP | 0.08 | 0.04 | 0.02 | 0.02 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 |

Table 10: Degradation points found for the different technique-dataset configurations for the translation perturbation.

When we compare the degradation points between each other, the first thing we notice is that NNP falls off very fast if it was trained on the Reuters dataset, no matter what projection technique was trained on. This could be caused due to the lower number of training samples that the model was trained on, which sparks a very interesting topic as to what this influence of training sample size has on the robustness of NNP or even any neural network in general. However, the reason *why* NNP degrades is out of scope for this thesis, and will thus be discussed in future works in Chapter 6.

From looking at the degradation points, where we ignore the Reuters dataset, we see that the projections degrade relatively slower when NNP is trained on PCA. This is due to the samples' relative distance not changing when the amount of translation or the number of randomly chosen dimensions is increased. This could be caused due to NNP training the inner workings of PCA. This is out of scope again, but it could be relevant for future work, as we will see in its appropriate Chapter 6.

When we look at the full set of numbers, we do see that the degradation points of NNP trained on the same projection technique, but a different dataset, has somewhat the same behavior. Both in terms of how quickly they degrade, as in what way they degrade. When trained on t-SNE we first see the white space between the clusters shrink until clusters start to overlap. With PCA we see the relative distance between each sample stay the same, while the barycenter seems to move and for UMAP we see a lot of white space, over which the samples move relatively quick through increased amount of translation compared to NNP trained on the other projection techniques.

### 4.5.2   Local scale

**Trend**   Just like the translation perturbation, influencing 0% of the dimensions changes nothing, and having 100% of the dimensions be influenced changes nothing as well, due to the relative difference between the samples staying the same. Therefore, normalization will again revert those scaled values back.



Table 11: The trend Local scale. Model trained on t-SNE using the MNIST dataset. Points of degradation are highlighted and rounded to the nearest tenfold.

In Table 11 we see the trend for the local scale perturbation. The model is trained on the MNIST dataset, where NNP uses t-SNE for its projection technique. We quickly notice, both from left to right and from top to bottom, that the projection is very robust to scaling down its dimension values. We could randomly choose up to 50% of the dimensions and scale them down to 0 before the projection started to get to a point of degradation. The model behaves in the same

manner as the translation perturbation with NNP models trained on t-SNE, as in Tables 1, 2 and 3, where first the white space between the clusters decrease until overlap, before they start to scatter through each other. However, for the local scale perturbation, the amount of local scaling and the number of influenced dimensions are much higher before degradation is found than with the translation perturbation.

In Table 12 we see the trend for local scale, with a model trained on the MNIST dataset with PCA as its trained projection technique. From left to right and from top to bottom we see the same thing happening as in Tables 2 and 5, where the relative distance between each sample stays the same and only its barycenter shifts when the number of influenced dimensions increases. For the local scale perturbation, we also see the same speed at which the samples of the projection move towards each other.
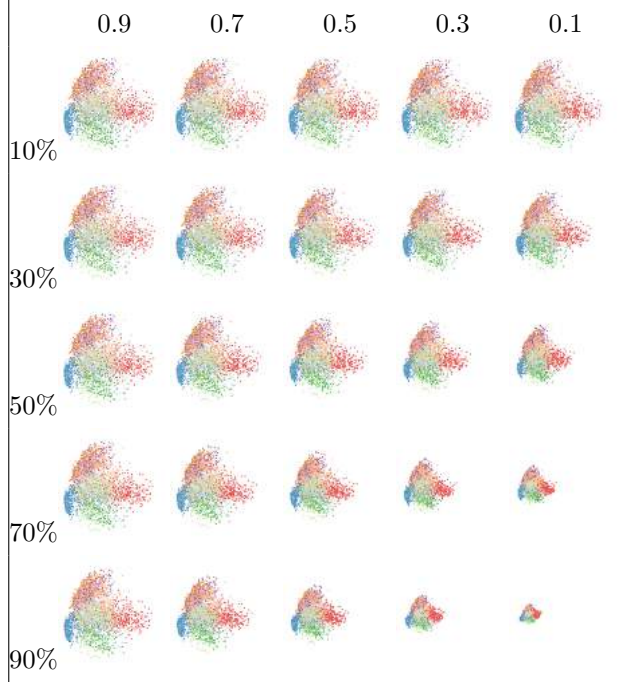


Table 12: The trend for local scale. Model trained on PCA using the MNIST dataset.

In Table 13 we, again, see the trend of local scale with a model trained on the MNIST dataset but with UMAP as its trained projection technique. We can see that, just like the model trained on t-SNE in Table 11, the projection is very robust to this local scaling. As we found through exploration of the previous perturbation, an NNP model trained on UMAP seems to lose its relatively larger amount of white space quicker. However, as NNP seems to be rather stable to local scaling, we see that this large amount of white space remains large, and also degrades slower than with the translation perturbation.
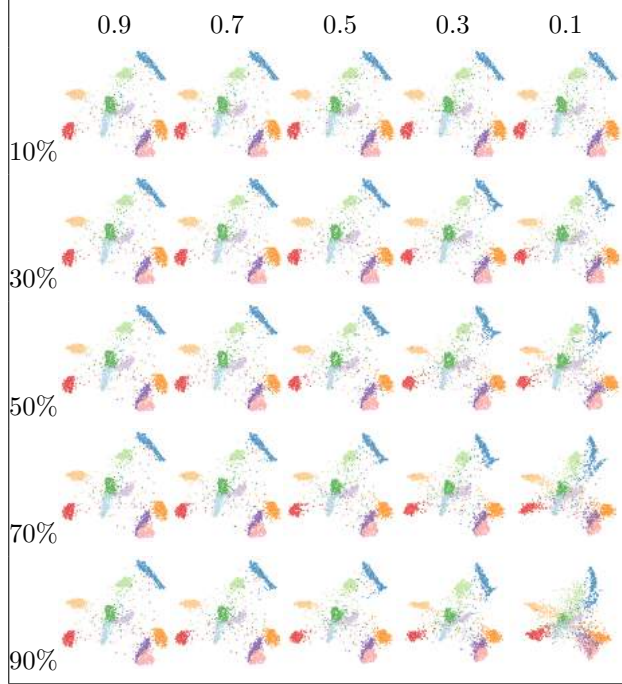


Table 13: The trend for local scale. Model trained on UMAP using the MNIST dataset.

In Table 14 we see the trend for the local scale perturbation on a model trained with the fashion-MNIST dataset and t-SNE as its trained projection technique. When comparing the trend of translation on a model trained with t-SNE as its trained projection technique in Tables 1 and 4, we saw that they were very much alike. However, if we compare this for the local scale trend in Tables 11 and 14, we see that the projection degrades slighter quicker. This might suggest that NNP is more robust when trained on MNIST than when trained on fashion-MNIST, albeit only marginally.
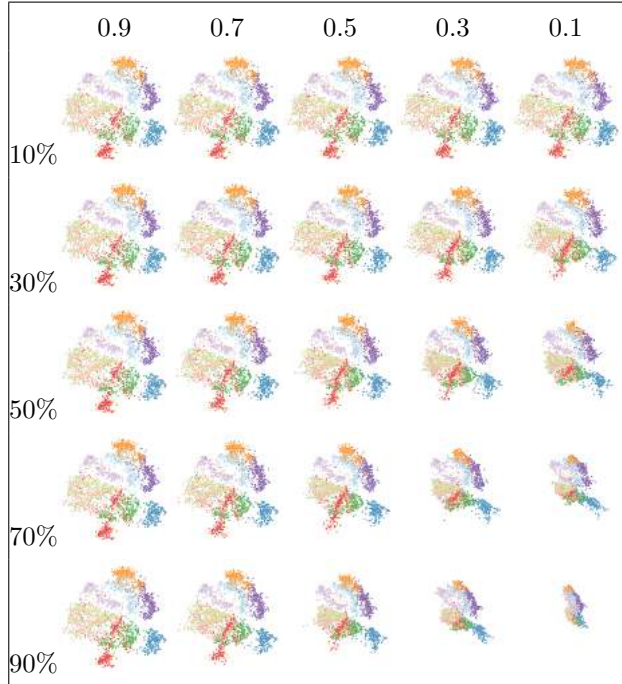


Table 14: The trend for local scale. Model trained on t-SNE using the fashion-MNIST dataset.

Table 15 shows the trend for the local scale perturbation of a model trained on the fashion-MNIST dataset with PCA for its trained projection technique. And again, just like Table 12, but also for Tables 2 and 5 with the trend for the translation perturbation, we see the relative distance between each sample staying the same, where for each increase in influenced dimensions or each decrease in the amount local scaling we see the same speed at which the projections degrade. And again, the barycenter moves when increasing the number of influenced dimensions.
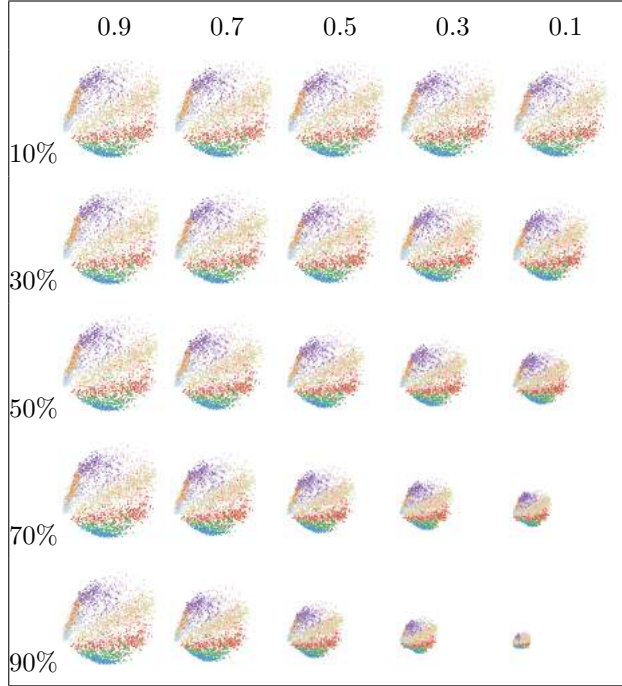


Table 15: The trend for local scale. Model trained on PCA using the fashion-MNIST dataset.

In Table 16 we see the trend for the local scale perturbation for an NNP model trained on the fashion-MNIST dataset with UMAP as its trained projection technique. Just like we saw in Table 13 where the model was trained on the MNIST dataset using the same projection technique, we see the degradation happening very slowly, where the white space does shrink and the declustering happens only with very much influence of the perturbation or a large number of influenced dimensions.
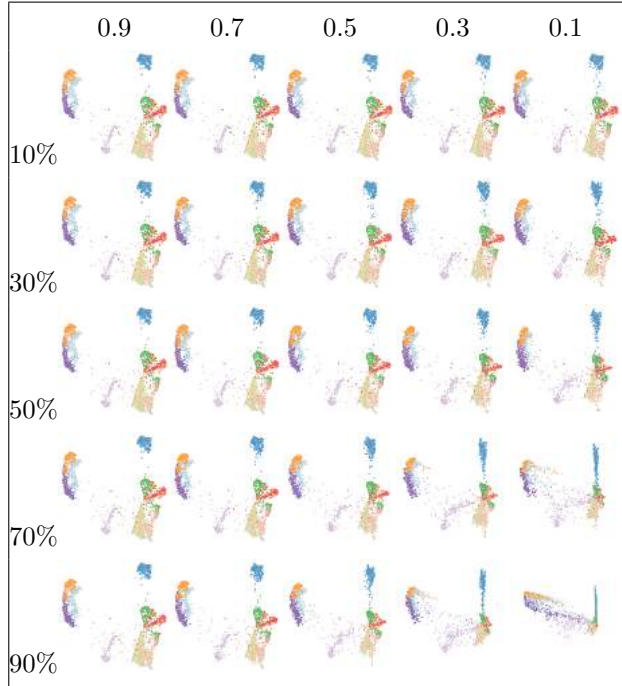


Table 16: The trend for local scale. Model trained on UMAP using the fashion-MNIST dataset.

In Table 17 we see the trend for the local scale perturbation using a model trained on the Reuters dataset using t-SNE as its projection technique. When we look left to right and top to bottom, we somewhat see the same behavior as we saw with the model trained on the MNIST or fashion-MNIST dataset with t-SNE as its trained projection technique in Tables 11 and 14. First some white space is reduced until clusters start to overlap, with scattering of the clusters happening next. But what is more interesting, is comparing this to any of the translation trends on the Reuters dataset, as this time, the projection does not degrade as absurdly fast when compared to Tables 7, 8 and 9.
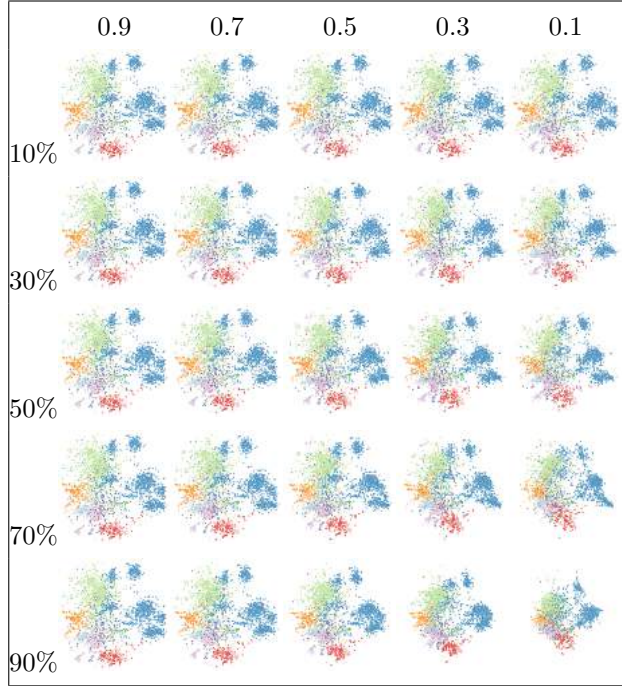


Table 17: The trend for local scale. Model trained on t-SNE using the Reuters dataset.

In Table 18 we see the trend for the local scale perturbation on a model trained on the Reuters dataset with PCA as its projection technique. We again see the slow degradation of the projection, where the relative distance between the samples somewhat stay the same. However, this time we can also see that this does not hold true until the higher amounts of local scaling and number of influenced dimensions. In the lower right projection, we see that the shape of the projection does indeed change. A change that we did not see in the comparison between any of the other PCA trained models, such as in Tables 12 and 15 or even in the translation perturbation in Tables 2 and 5. This falls in line with the earlier found results where the amount of training samples might influence the robustness of NNP. But this will be further explored in Chapter 6.
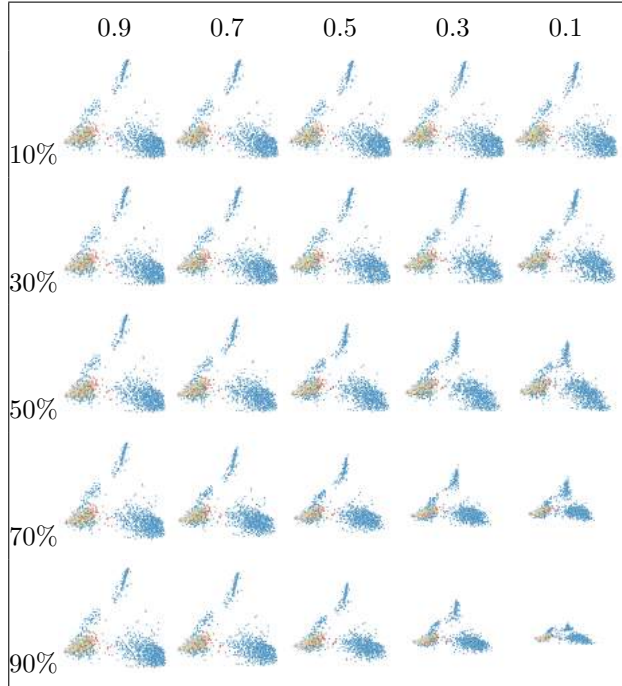


Table 18: The trend for local scale. Model trained on PCA using the Reuters dataset.

In Table 19 we see the trend for local scale on a model trained on the Reuters dataset using UMAP as its trained projection technique. Again, the increased influence of the local scale perturbation makes the projections degrade rather slowly, where the shape of the projection stays somewhat the same up until a higher amount of influence, where even still there is somewhat of a separation in clusters. Although, not as strongly as the model trained on t-SNE and the same dataset in Table 17 or the model trained on the MNIST dataset with the same projection technique.
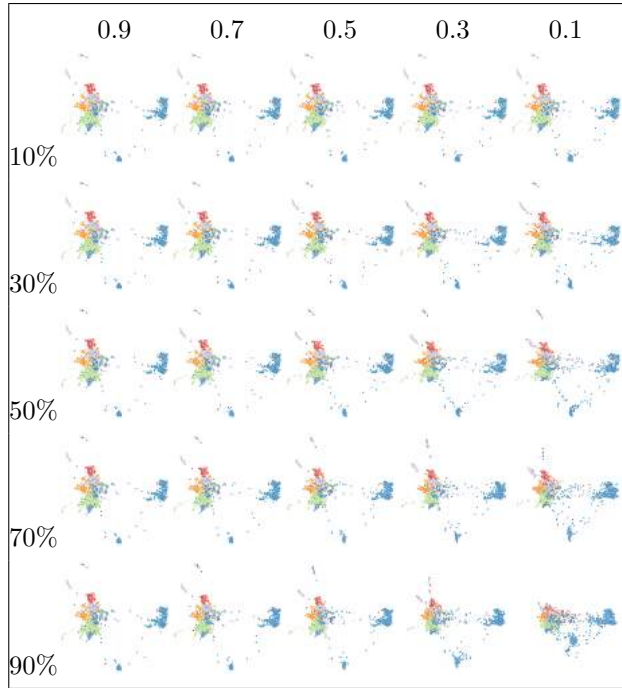


Table 19: The trend for local scale. Model trained on UMAP using the Reuters dataset.

**Points of degradation**   Table 20 shows an overview of all the degradation points found in the trends for the local scale perturbation on every technique-dataset configuration.

|  |  | 10% | 20% | 30% | 40% | 50% | 60% | 70% | 80% | 90% |
|---|---|---|---|---|---|---|---|---|---|---|
| MNIST | t-SNE | 0.00 | 0.00 | 0.00 | 0.00 | 0.10 | 0.25 | 0.40 | 0.45 | 0.45 |
|  | PCA | 0.00 | 0.00 | 0.00 | 0.10 | 0.30 | 0.40 | 0.50 | 0.55 | 0.60 |
|  | UMAP | 0.00 | 0.00 | 0.00 | 0.00 | 0.10 | 0.20 | 0.30 | 0.45 | 0.60 |
| fashion MNIST | t-SNE | 0.00 | 0.00 | 0.00 | 0.10 | 0.30 | 0.40 | 0.45 | 0.60 | 0.65 |
|  | PCA | 0.00 | 0.00 | 0.00 | 0.15 | 0.25 | 0.35 | 0.50 | 0.60 | 0.70 |
|  | UMAP | 0.00 | 0.00 | 0.00 | 0.00 | 0.15 | 0.20 | 0.30 | 0.40 | 0.45 |
| Reuters | t-SNE | 0.00 | 0.00 | 0.00 | 0.15 | 0.25 | 0.40 | 0.55 | 0.65 | 0.70 |
|  | PCA | 0.00 | 0.00 | 0.00 | 0.20 | 0.30 | 0.35 | 0.40 | 0.45 | 0.55 |
|  | UMAP | 0.00 | 0.00 | 0.00 | 0.00 | 0.15 | 0.25 | 0.35 | 0.45 | 0.55 |

Table 20: Degradation points found for the different technique-dataset configurations for the local scale perturbation.

We quickly see that NNP is relatively robust to the local scale perturbation in general, where models trained on UMAP are the strongest against its influence. This is due to the large amount of white space that the projection produces. With much white space between the clusters, only a small amount of degradation still keeps enough separation between the clusters.

When scaling down by up to 40% of the dimensions to 0, no declustering is found yet. Scaling dimensions to 0 acts in the same manner as dimensions removal would do, which will be explored in Section 4.5.5. Only from at least 40% of influenced dimensions, there will be a point at which the effect of scaling has enough influence to start declustering. Then, after influencing 80% or more of the dimensions with scaling, the declustering starts to stagnate to a factor of at maximum 0.7. From this point on influencing more dimensions with scaling will not have much effect to the point of declustering.

A big difference when comparing the trends and degradation points for the local scale perturbation with the translation perturbation is when inspecting the Reuters dataset. Where the translation perturbation would heavily degrade after only marginal influences on the models trained on the Reuters dataset, for the local scale perturbation we see somewhat the same behavior to perturbation of the other datasets. However, if we then dive deeper into details when looking only at the models that were trained on PCA, we do not see the same exact behavior as the structure does change on the models trained on the Reuters dataset. Whereas, the models trained on the MNIST or fashion MNIST dataset using PCA as the trained projection technique, we do not see any structural change through perturbance. This falls in line with the hypothesis of a lower number of training samples not being able to fully learn the behavior of PCA. This will be further discussed in Chapter 6 Future work, as it is out of scope for this thesis.

### 4.5.3 Jitter



Table 21: Trend for the jitter perturbation. All technique-dataset configurations together.

In Table 21 we see an overview of the trend for the jitter perturbation of an increased intensity applied to every samples' dimension, applied for every technique-dataset configuration. We notice that the projections inferred from each NNP model is not very robust to this type of perturbation, as the point of degradation is already reached at a jitter range with a minimum and maximum of $(-0.1, 0.1)$. If we specifically look at PCA, we see that the model is more robust than when a model is trained on the other projection techniques, albeit only marginally. However, PCA does not separate the class clusters very well from the start, thus the point of degradation is much harder to judge.

Also, just like when we explored the translation trend using the Reuters dataset, we see that the projection almost immediately degrades into disorder. Where, even at a range of $(-0.1, 0.1)$ we see all the samples in very small area. From this, and the trend we saw with the translation perturbation, we can see some kind of pattern. What both the jitter and the translation perturbation have in common, is that they introduce false information to the samples. In more detail, that means that with both translation and jitter, there are dimensions that were zero, which would then changed to be non-zero. This could explain the faster degradation, especially for the Reuters dataset, where the models were not trained by the same higher amount of training samples.

Table 22: Percentage of dimensions influenced by jitter to the point of degradation, for all technique-dataset configurations.

Table 22 shows an overview of all the points of degradation for each technique-dataset configurations, together with the projection we see at these points of degradation. We can see that through not much influence of the perturbation, each technique-dataset configuration starts to decluster very early on, confirming what we saw happening in the trend lines in Table 21. Moreover, we see that at only a max range of $(-0.01, 0.01)$ the models trained on the Reuters dataset are already degraded into disorder. Even at that small of an influence, the projections do not show much clustering any more.

A                                       B

Table 23: The jitter perturbations, with a max range of $(-0.1, 0.1)$, visualized through the convex hull option of the star map. Figure A shows that NNP trained using t-SNE as its projection technique and Figure B showing NNP trained using UMAP as its projection technique, both trained using the MNIST da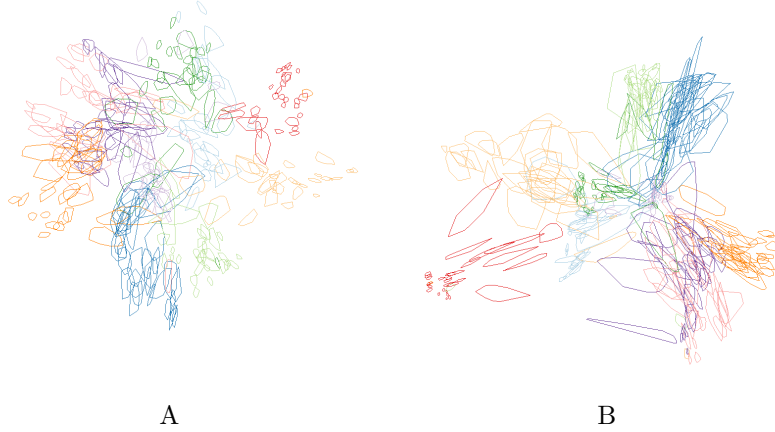taset. A tenth (300) of the normal number of samples are used to prevent over cluttering the projection, so that one can more clearly extract information from it.

Table 23 shows the jitter perturbation through the convex hull option of the star map. The models were trained using the MNIST dataset, and t-SNE and UMAP as its projection techniques respectively. We can see that the projection using the model trained on t-SNE has slightly smaller convex hulls, indicating a smaller amount of sample movement through the jitter perturbation. This could be seen as t-SNE being more robust towards the perturbation. But, one has to take into account the amount of white space in both projections, as UMAP tends to have more of it between its projected clusters.

When looking closer at the individual clusters of the t-SNE trained model compared to the UMAP trained model, we see that the convex hulls of the latter or more elongated. The samples of the t-SNE model have a more even spread around their original location than the samples in the UMAP models. As the UMAP model has more white space between its clusters, it seems to use this space more when it is less sure as to how the perturbed samples distinguish between each other. On the other hand, the t-SNE model has less space to work with, so it can use less to relocate the perturbed samples.

#### 4.5.4 Permutations



Table 24: Trend for the permutation perturbation. All technique-dataset configurations together.

Figure 24 shows an overview of the permutation trends for each technique-dataset configurations. Through increasing the amount of dimensions swapped, in the table from left to right, we see the samples jump relatively small or large amounts. With only a small influence of permutation, the samples only jump around its original place. Only with more permutation we see the samples degrade towards a central point. The jumping around in small or large distances is probably caused due to the random choice of which dimensions are swapped. When zero dimensions get swapped with non-zero dimensions, a large change of location could be seen, whereas if many zero dimensions are swapped with other zero dimension, then a small change in location could be seen. Either way, we could argue there being a correlation between neighbouring dimensions being broken, indicating the change we see. The actual point of degradation happens after around 30% of the dimensions are swapped throughout all the technique-dataset configurations.

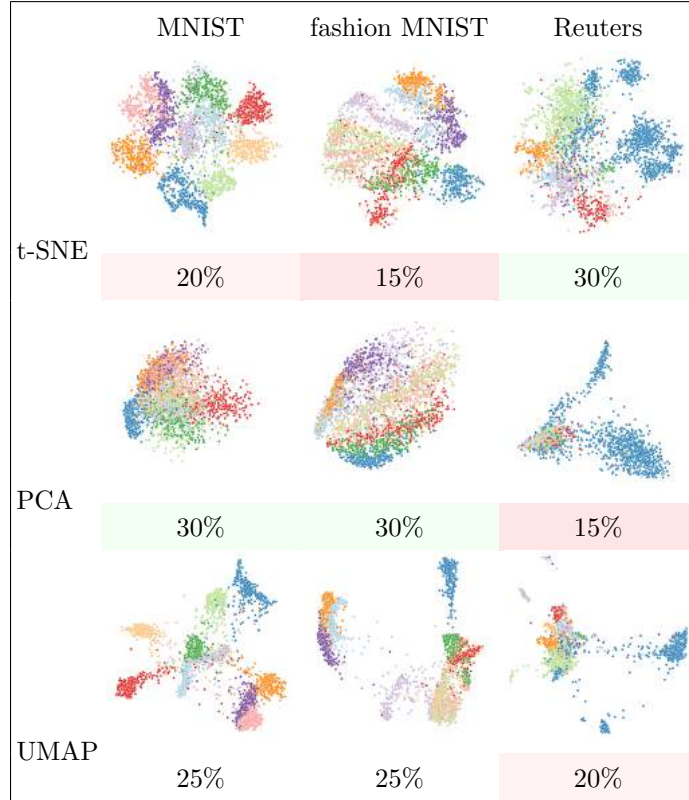|  | MNIST | fashion MNIST | Reuters |
|---|---|---|---|
| t-SNE | 20% | 15% | 30% |
| PCA | 30% | 30% | 15% |
| UMAP | 25% | 25% | 20% |

Table 25: Permutation influence in percentage to the point of degradation, for all technique-dataset configurations.

Table 25 shows an overview of all the points of degradation together for all technique-dataset configurations. At these more precise points of degradation, we do see that some samples of each cluster are already jumped relatively much compared to degradation points in other perturbations, like we have seen in for example the translation trend or the local scale trend in Figure 4 or 14, where the clusters' samples are still more close together when degradation happens.

Through permutation, we also wanted to find out whether the model looks at the individual dimensions or if there is more to be looked at. From the difference between $P$ and $P'$ we do see much change happening, so that means there must be some indirect dependencies that the NNP models learn, where the model might take neighbouring dimensions into account for the locations of the samples, as the precise values in each vector do not change.

### 4.5.5 Dimension removal

In this perturbation we kept removing dimensions until the degradation point was found as shown in Table 26. Setting 100% of the dimensions' weights to 0 would make the data void, so those projections are left out, as they yields no interesting information about the perturbation.



Table 26: Trend for the dimensions removal perturbation. All technique-dataset configurations together.

Table 27 show an overview of the trends of the dimension removal perturbation for each technique-dataset configurations. When comparing these to the other trends, we see that the point of degradation is relatively far into the perturbations' influence. When we look from left to right on each row, we see that there is only marginal change through increased perturbation influence. We do notice a synonym with the points of degradation for the local scale perturbation in Section 4.5.2, as scaling down weights to 0 acts in the same way as the dimension removal perturbation does.

Models trained using t-SNE change very gradually, whereas the models trained using UMAP changes relatively faster. Although the clusters are still distinguishable, as the original UMAP projection technique creates more white space between the clusters than t-SNE does. PCA on the other hand already has barely any white space between its clusters in the original projection, where this distance between samples decreases very gradually.

Table 27 shows the percentage of dimensions set to zero at which declustering starts, together with the projection at this percentage. An impressive average of 37.8% of the dimensions could be removed before the models started to lose the patterns that the original projection had due to

|  | MNIST | fashion MNIST | Reuters |
|---|---|---|---|
| t-SNE | 40% | 30% | 40% |
| PCA | 30% | 35% | 25% |
| UMAP | 55% | 45% | 40% |

Table 27: Dimension removal for the number of influenced dimensions to the point of degradation, for all technique-dataset configurations.

degradation. This means that one could potentially miss up to 37.8% of its sample data before the model starts to degrade in its quality of projections.

There is no real pattern to be found between the trained on projection techniques on the same dataset. However, we do see some sort of pattern for the models that keep the same projection technique, where the used dataset is changed. We see that models trained using UMAP are the most robust to dimension removal with an average of 46.7%, followed up by models trained using t-SNE with an average of 36.7% and lastly models trained using PCA with an average of 30%. It seems like the amount of white space in the original projection might be an indication as to how robust the projection is to the removal of data.

## 4.6 Comparison

This section will compare the various results we found through each perturbation.

To find a more generalizable answer as to *how* inferred projections change based on the perturbations, we will also explore patterns in aggregated results between visualizations on isolated perturbations. These can shed light on the underlining behaviour of how a model behaves when data is perturbed. As the jitter perturbation is not ordered, it will not yield much value to inspect the aggregation through trail or heat map visualizations.

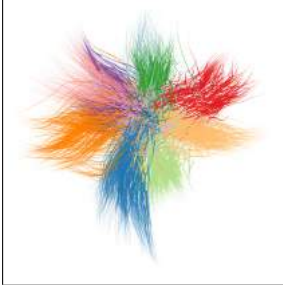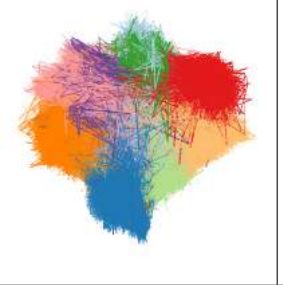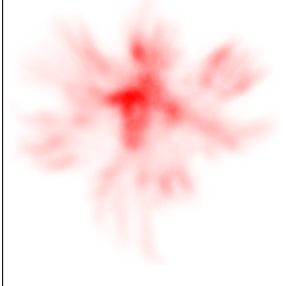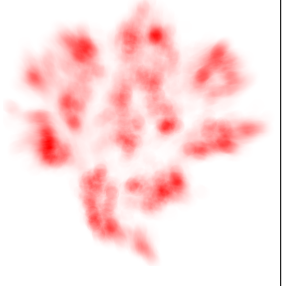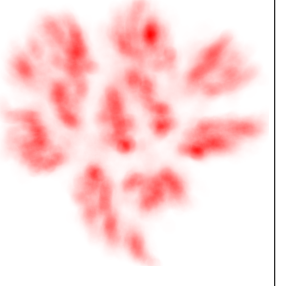| Translation | Local scale | Permutation | Dimension removal |
|:---:|:---:|:---:|:---:|
| | | | |
| | | | |
| 3299 | 2351 | 2142 | 1967 |

Table 28: Aggregation results for the ordered perturbations. Perturbation influence on 50%. Heat maps include the max heat found.

As seen through all the trail maps and as mentioned earlier, sooner or later the trails of any perturbation all degrade into each other. Through perturbation of the data, the model will have a harder time in separating which samples are alike and which samples or different from each other. Hence, why in most perturbation, we first see the white space that separates the clusters shrink before we see the clusters overlap or degrade away from the other in-class samples. We could argue that the perturbation with more influence on degradation makes its white space shrink relatively faster. This could mean that the model is less robust to that specific perturbation than to others. But like so, we could also argue the other way around, where when a perturbation has its white space shrink slowly through increased perturbation influence, the model is relatively robust towards it.

The heatmap puts more trust into the degradation towards the centre hypothesis, as can mostly been seen from Translation and Dimension removal in Table 28. Where translation immediately moves towards the centre, dimensions removal first circles around its non-perturbed locations. More so as the max heat found in translation is much higher than the other, due to the samples already being degraded to the center, whereas the samples influenced by dimension removal are still very spread out. From this we can argue that the model is more robust to for example local scaling than to translation.
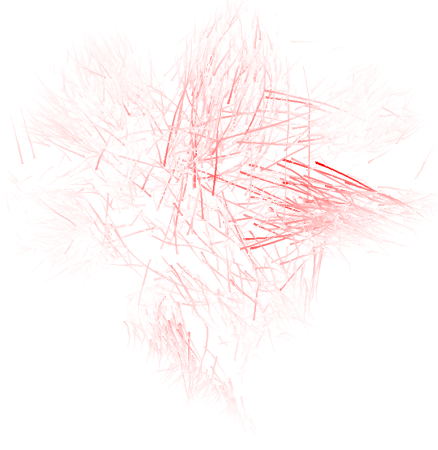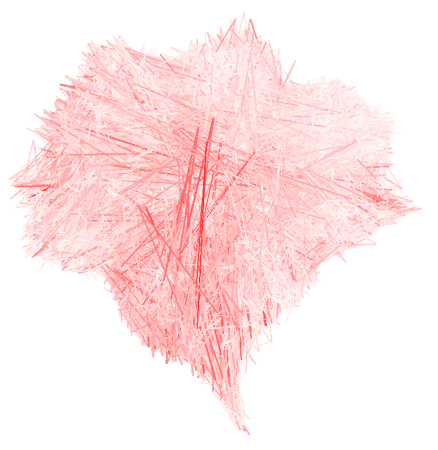
| | Local Scale | Permutation |
|---|---|---|
| (min, max) | $(1.675 * 10^{-5}, 0.2588)$ | $(4.700 * 10^{-5}, 0.1085)$ |
| Average | 0.01440 | 0.01250 |
| Variance | $2.296 * 10^{-4}$ | $0.9187 * 10^{-4}$ |

Table 29: Difference in distortion distance and its statistics for the permutation and local scaling perturbations.

However, we do have to then look at how the permutation perturbation shows its results compared to local scale, because they look very different in their trail and heat maps, but show somewhat the same amount of movement if we look at the maximum heat found for both of them. To find out more about this, we can use the distortion distance color together with statistics for this to understand more. The visuals strengthen the statistics, as shown in Table 29. It explains that the local scale perturbation has a larger range, with a higher variance than the permutation perturbation, meaning the distance that the samples move in $Nd$ relative to $2D$ is higher. This tells us that permutation is steadier in terms of robustness than local scaling is, strengthening the claim of the visualization.

# 5  Discussion and conclusion

In Chapter 1 we stated that Neural Network Projections are a solution to many problems we face in Dimensionality Reduction techniques. Among these problems, stability and generalization were not explored enough to have a good idea as to how robust the inference of NNP is to change of the data. To explore these aspects, we presented a tooling and methodology with some simple example perturbations. The results from these explorations were presented in Chapter 4.

We can confirm that, for a small amount of perturbance, NNP trained on any projection technique is stable to a certain degree for most perturbations. While we are not claiming any hard figures, as our results are just examples, we can confirm that the found robustness is dataset and projection technique dependent. This was a given, but was not confirmed yet. The amount of difference between the datasets and projection techniques however were not as large as the differences in the type of perturbations. To some type of perturbations, the stability is stronger than for others. NNP seems to be more robust towards alterations of existing data than to the introduction of false information, as the inferred projections seem to degrade faster when weights are changed from zero to non-zero than when non-zero weights are changed in value. With alteration of existing data, perturbance will first make the samples move around their original data before degrading towards a center point, whereas false information will immediately start the degradation towards the center.

The tool created to explore these findings is generic. For this thesis it was used to explore the spectrum of change within NNP, but it could also be used to explore changes to the projections of Dimensionality Reduction techniques themselves, like for example t-SNE or UMAP. For these cases we would simply infer projections through those techniques and display them in the tool. We could then change the data and infer this changed data through the technique to then analyze what has changed to the projection. The chosen family of perturbations that was used in this thesis could be used as well, making the exploration for the generalization of the projection technique, reasonably easy.

Results from the explorations using the proposed tooling and methodology can for example be used for data augmentation. Where one can change the existing data through the various perturbations. The amount of augmentation to this data can be tested by the tool to the point where it is still recognizable by the trained model. Another use could for example be to find out how much data can be wrong or missing before predictions start to be useless. For instance, if any field works with possibly incorrect or missing data, one can explore how much of it can happen before the output becomes unreliable. In short, the proposed tool and methodology give an easy solution to exploring the amount of change to data one could require to be either stable, useful or both.

# 6  Future Work

We have seen that NNP is affected differently by the choice of dataset, projection technique, perturbation and whether the perturbation introduced false information or not. But another interesting finding was the introduction of false information to the Reuters dataset. We've seen that alteration to existing data had somewhat the same effect on both the 9000 sample training set of MNIST or fashion-MNIST and the 4500 sample dataset of Reuters. However, introduction of false information had quite substantially different effects to this dataset, where the speed of degradation was much quicker on the Reuters dataset compared to the others. Comparing the points of degradation in Figure 21 shows this very clearly. This effect might also be caused through using only a subset of all the 90 classes of the Reuters dataset. In either case finding out what causes this major drop-off could give further insight into the generalization of NNP or other techniques.

Moreover, from our results we found that NNP learned the underlining linear workings of PCA as we saw on the effects of the perturbation to the projections. However, when we explored a model trained on PCA with the Reuters dataset, yielding the lower amount of training samples, we saw that this underlining behavior was less clearly visible, as seen in a comparison between Figures 12 or 15 and Figure 18. Another indication of the effect of the amount of training samples is when we compare the inferred projections between models that are trained on, for example, UMAP using 3000, 6000 and 9000 samples respectively, as seen in Table 30. Even though UMAP is a deterministic projection technique, we still see a difference in its unperturbed projections, indicating that the underlining behavior of UMAP is not fully learned yet. Therefore in future work, one could explore how the amount of training samples has effect on the points of degradation by the proposed, or any other perturbations.



Table 30: Models trained using the deterministic UMAP projection technique on the MNIST dataset, with respectively 3000, 6000 and 9000 training samples.

The scope of this thesis has been on *when* and *how* perturbations to data has effect on the inferred projections. Now that we have answers for these questions, the following that one could explore is *why* the perturbance degrades the projections like it does. Some small indications as to the reasons behind the change were given, but due to time restrains this topic is not fully explored yet. The exploration of this *why* can give reasoning to the generalization, providing a more complete overview of how one can expect the models behavior, shedding more light into our black box.

# References

1. F.R.S., K. P. LIII. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* **2,** 559–572. eprint: https://doi.org/10.1080/14786440109462720. https://doi.org/10.1080/14786440109462720 (1901).

2. Van der Maaten, L. & Hinton, G. Visualizing data using t-SNE. *Journal of machine learning research* **9** (2008).

3. Espadoto, M., Hirata, N. S. T. & Telea, A. C. Deep learning multidimensional projections. *Information Visualization* **19,** 247–269. eprint: https://doi.org/10.1177/1473871620909485. https://doi.org/10.1177/1473871620909485 (2020).

4. Espadoto, M., Falcao, A., Hirata, N. & Telea, A. *Improving Neural Network-based Multidimensional Projections* in *Proc. IVAPP* (2020).

5. Spearman, C. "General Intelligence" Objectively Determined and Measured. (1961).

6. Torgerson, W. S. Multidimensional scaling: I. Theory and method. *Psychometrika* **17,** 401–419 (1952).

7. Van Der Maaten, L., Postma, E. & Van den Herik, J. Dimensionality reduction: a comparative. *J Mach Learn Res* **10,** 13 (2009).

8. Ayesha, S., Hanif, M. K. & Talib, R. Overview and comparative study of dimensionality reduction techniques for high dimensional data. *Information Fusion* **59,** 44–58. ISSN: 1566-2535. https://www.sciencedirect.com/science/article/pii/S156625351930377X (2020).

9. Sacha, D. *et al.* Visual Interaction with Dimensionality Reduction: A Structured Literature Analysis. *IEEE Transactions on Visualization and Computer Graphics* **23,** 241–250 (2017).

10. Thrun, M. & Ultsch, A. Using Projection-Based Clustering to Find Distance- and Density-Based Clusters in High-Dimensional Data. *Journal of Classification* **38,** 280–312 (2021).

11. He, K., Zhang, X., Ren, S. & Sun, J. *Identity Mappings in Deep Residual Networks* in *Computer Vision – ECCV 2016* (eds Leibe, B., Matas, J., Sebe, N. & Welling, M.) (Springer International Publishing, Cham, 2016), 630–645. ISBN: 978-3-319-46493-0.

12. McInnes, L., Healy, J. & Melville, J. *UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction* 2020. arXiv: 1802.03426 [stat.ML].

13. Espadoto, M., Martins, R. M., Kerren, A., Hirata, N. S. T. & Telea, A. C. Toward a Quantitative Survey of Dimension Reduction Techniques. *IEEE Transactions on Visualization and Computer Graphics* **27,** 2153–2173 (2021).

14. Garcia, R., Telea, A. C., Castro da Silva, B., Tørresen, J. & Dihl Comba, J. L. A task-and-technique centered survey on visual analytics for deep learning model engineering. *Computers & Graphics* **77,** 30–49. ISSN: 0097-8493. https://www.sciencedirect.com/science/article/pii/S0097849318301535 (2018).

15. Wongsuphasawat, K. *et al.* Visualizing dataflow graphs of deep learning models in tensorflow. *IEEE transactions on visualization and computer graphics* **24,** 1–12 (2017).

16. Qi, H., Liu, J., Zou, X., Tang, A. & Canny, J. F. *Bidviz: Real-time monitoring and debugging of machine learning training processes* PhD thesis (Master's thesis, EECS Department, University of California, Berkeley, 2017).

17. Zeiler, M. D. & Fergus, R. *Visualizing and understanding convolutional networks* in *European conference on computer vision* (2014), 818–833.

18. Samek, W., Binder, A., Montavon, G., Lapuschkin, S. & Müller, K. Evaluating the Visualization of What a Deep Neural Network Has Learned. *IEEE Transactions on Neural Networks and Learning Systems* **28,** 2660–2673 (2017).

19. Rajaraman, S., Candemir, S., Kim, I., Thoma, G. & Antani, S. Visualization and Interpretation of Convolutional Neural Network Predictions in Detecting Pneumonia in Pediatric Chest Radiographs. *Applied Sciences* **8.** ISSN: 2076-3417. https://www.mdpi.com/2076-3417/8/10/1715 (2018).

20. Li, H., Xu, Z., Taylor, G. & Goldstein, T. Visualizing the Loss Landscape of Neural Nets. *CoRR* **abs/1712.09913.** arXiv: 1712.09913. http://arxiv.org/abs/1712.09913 (2017).

21. Huang, W. R. *et al. Understanding Generalization Through Visualizations* in *Proceedings on "I Can't Believe It's Not Better!" at NeurIPS Workshops* **137** (PMLR, Dec. 2020), 87–97. http://proceedings.mlr.press/v137/huang20a.html.

22. Ribeiro, M. T., Singh, S. & Guestrin, C. *" Why should i trust you?" Explaining the predictions of any classifier* in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining* (2016), 1135–1144.

23. Dyrba, M. *et al. Improving 3D convolutional neural network comprehensibility via interactive visualization of relevance maps: Evaluation in Alzheimer's disease* 2021. arXiv: 2012.10294 [eess.IV].

24. Smilkov, D., Carter, S., Sculley, D., Viégas, F. B. & Wattenberg, M. Direct-Manipulation Visualization of Deep Networks. *CoRR* **abs/1708.03788.** arXiv: 1708.03788. http://arxiv.org/abs/1708.03788 (2017).

25. Zeigermann, O. *Interactive Visualization of Autoencoders* 2020. anomagram.fastforwardlabs.com/#/.

26. Wang, Z. J. *et al.* CNN explainer: Learning convolutional neural networks with interactive visualization. *IEEE Transactions on Visualization and Computer Graphics* (2020).

27. Harley, A. W. *An Interactive Node-Link Visualization of Convolutional Neural Networks* in *Advances in Visual Computing* (eds Bebis, G. *et al.*) (Springer International Publishing, Cham, 2015), 867–877. ISBN: 978-3-319-27857-5.

28. Jeong, D. H., Ziemkiewicz, C., Fisher, B., Ribarsky, W. & Chang, R. *ipca: An interactive system for pca-based visual analytics* in *Computer Graphics Forum* **28** (2009), 767–774.

29. Modrakowski, T. *Visual analytics for improving deep learning multidimensional projections* MA thesis (2020).

30. Cantareira, G. D., Etemad, E. & Paulovich, F. V. Exploring Neural Network Hidden Layer Activity Using Vector Fields. *Information* **11.** ISSN: 2078-2489. https://www.mdpi.com/2078-2489/11/9/426 (2020).

31. Saunders, D. G. O., Win, J., Kamoun, S. & Raffaele, S. in *Plant-Pathogen Interactions: Methods and Protocols* (eds Birch, P., Jones, J. T. & Bos, J. I.) 29–51 (Humana Press, Totowa, NJ, 2014). ISBN: 978-1-62703-986-4. https://doi.org/10.1007/978-1-62703-986-4_3.

32. Babicki, S. *et al.* Heatmapper: web-enabled heat mapping for all. *Nucleic Acids Research* **44,** W147–W153. ISSN: 0305-1048. eprint: https://academic.oup.com/nar/article-pdf/44/W1/W147/7634999/gkw419.pdf. https://doi.org/10.1093/nar/gkw419 (May 2016).

33. Novak, R., Bahri, Y., Abolafia, D. A., Pennington, J. & Sohl-Dickstein, J. *Sensitivity and Generalization in Neural Networks: an Empirical Study* 2018. arXiv: 1802.08760 [stat.ML].

34. Yang, J., Zeng, X., Zhong, S. & Wu, S. Effective Neural Network Ensemble Approach for Improving Generalization Performance. *IEEE Transactions on Neural Networks and Learning Systems* **24,** 878–887 (2013).

35. Gong, C., Ren, T., Ye, M. & Liu, Q. *MaxUp: A Simple Way to Improve Generalization of Neural Network Training* 2020. arXiv: 2002.09024 [cs.LG].

36. Parzen, E. On Estimation of a Probability Density Function and Mode. *The Annals of Mathematical Statistics* **33,** 1065–1076. ISSN: 00034851. http://www.jstor.org/stable/2237880 (1962).

37. LeCun, Y. & Cortes, C. MNIST handwritten digit database. `http://yann.lecun.com/exdb/mnist/` (2010).

38. Xiao, H., Rasul, K. & Vollgraf, R. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms* cite arxiv:1708.07747Comment: Dataset is freely available at https://github.com/zalandoresearch/fashion-mnist Benchmark is available at http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/. 2017. `http://arxiv.org/abs/1708.07747`.

39. Thoma, M. *The Reuters Dataset* July 2017. `https://martin-thoma.com/nlp-reuters`.