Deep learning persistent topological features

Author:
Stijn Slot

Supervisors:
Bettina Speckmann
Kevin Verbeek
Vlado Menkovki

August 23, 2019

1 Introduction

The following report is written for a research honors project in the Geometric Algorithms group. In the project, the combination of deep learning and topology is explored.

Deep learning is a part of the machine learning field and has become very popular, with application in the areas of artificial intelligence and data science. Deep learning tools such as neural networks have achieved great success in various difficult tasks, from image recognition to playing chess. In many cases these networks are even able to outperform human experts. However, there are still many challenges and issues that need to be solved. One such issue is that of *interpretability*. A desirable feature of a neural network would be that humans can explain how and why it makes certain decisions. In general, most neural networks are seen as a black box, where data is fed into and solutions come out. Therefore, it can be difficult to determine how exactly the machine works and whether it is learning the right information. Efforts to increase interpretability try to either use techniques to understand why the trained network makes certain decisions, for example by visualizing intermediate results or by seeing which inputs a neuron or layer reacts strongly too, or to try to restrict the network to a certain shape that can be more easily interpretable, while hopefully not sacrificing too much of its predictive power.

At the same time, the developments in topology and persistent homology has led to the emerging of the field of topological data analysis. This field deals with the extraction and analysis of topological information from data. The use of topological features helps to describe the underlying information present in the data. These features can provide insights into the shape independent of properties such as the resolution and exact positioning, which could be subject to noise. Much of the data that needs to be analyzed is noisy, incomplete, or otherwise challenging to process. Thus, with topological data analysis we would like to extract features that are *stable* with respect to small changes (noise) in the data. This is where persistent homology is a useful tool, since it offers to extract topological features at different "persistent levels". Features with high persistence are deemed important, while low persistent features are seen as noise.

The combination of deep learning and topological data analysis is just now being explored. At the moment it is unclear to what effect topology can be used together with deep learning. The

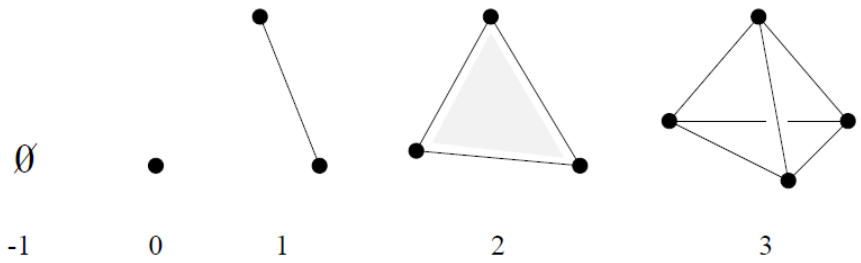

Figure 1: Examples of simplices of different dimensions. From StackExchange.

use of topological features in deep learning seem promising, for example to have networks learn topological shape instead of focusing on other artifacts of the data. Having the network either learn topology or be given the (persistent) topological information might make it more general and less susceptible to focusing on the "wrong" information (e.g. visual artifacts appearing in all pictures of a certain class). Furthermore, having the network extract topological features before classification might help with the interpretability of the network, since we know better which information it uses and what it does not use to come to a prediction, compared to a standard convolutional network that might also use geometry, etc.

The main question that will be explored in the project is whether deep learning can be used to extract topological features from an image.

1.1 Topology and persistent homology

In *persistent homology* the topological features of a space are computed at different resolutions. In a way, it captures the emergence and disappearance of topological features, such as holes, over different spatial scales. Here, a feature is more *persistent* when the feature exists for a longer period (in terms of spatial resolution). The assumption is that features that are persistent (have high persistence) are important for the general topology, while features with low persistence are regarded as noise. Persistent homology is a useful tool for describing data, since it takes into account the stability with respect to noise of topological features in the data.

Before I will explain persistent homology, I will give some background on geometry/topology, in particular on simplicial complexes. In geometry, a *simplex* is a generalization of concepts such as points, edges, triangles, and tetrahedrons to arbitrary dimensions. A k-simplex is defined by a set of $k+1$ vertices $u_0, \ldots, u_k$ in k-dimensional space, which are *affinely independent* meaning that $u_1 - u_0, \ldots, u_k - u_0$ are linearly independent (this avoids a degenerate case where points are colinear). See Figure 1 for examples of simplices at different dimensions. Any non-empty subset of the $k+1$ vertices that define the k-simplex is called a *face*. The faces of a simplex are themselves also simplices of a lower dimension. In particular, a subset of size $n+1$ will be an n-simplex and can be called an n-face of the k-simplex. For example, a triangle is a 2-simplex that consist of three 1-faces in the form of edges and three 0-faces as vertices.

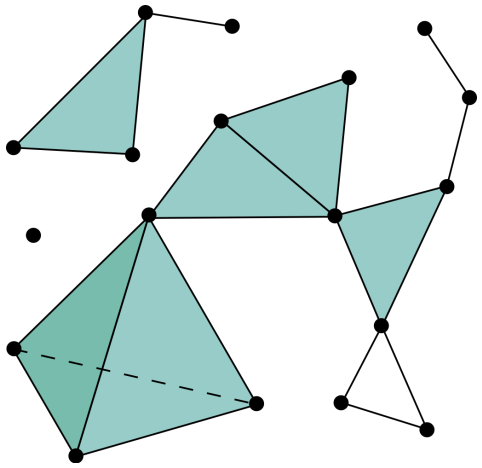

Figure 2: Example of a simplical complex. From Wikipedia.

A set of different simplices (points, edges, triangles) together is called a *simplicial complex*. It can be seen as the result of "gluing together" simplices, possibly of different dimensions. The set of simplices $\mathbb{K}$ needs to satisfy the following two conditions in order to be called a simplicial complex:

1. Every *face* of a simplex in $\mathbb{K}$ is also in $\mathbb{K}$.
2. The (non-empty) intersection of two simplices in $\mathbb{K}$ is a face of both simplices.

A simplicial complex without an associated geometry is called an *abstract simplicial complex*. See Figure 2 for an example of a simplicial complex.

For persistent homology (and topology in general) the concept of a filtration is imporant. Given some simplicial complex $\mathcal{K}$, we define a filtration of the complex:

$$\mathcal{K}_0 \subset K_1 \subset \ldots \subset K_n \subseteq$$

This filtration is defined as a nested sequence of increasing subsets of the simplicial complex, which capture the scale for the persistent homology. The filtration can be created using a distance function on the underlying space, where a radius value r is increased and vertices within distance r are connected in a simplex. This will give a finite set of complexes that capture topological information about the data at different distance scales. When the data consists of a set of points, a typical filtration uses the Vietoris–Rips complex. This complex uses a parameter value δ where a set of points is added as a simplex whenever their diameter (the maximum distance between two points in the simplex) is less or equal to δ. For the filtration this parameter is increased to create the increasing subsets of simplicial complexes. See Figure 3 for an example of the Vietoris-Rips complex.

Persistent homology gives a multiscale description of the topology of this filtration. It turns out that the persistent topological features can be well described by a finite sequence of birth-death pairs (b_i, d_i) with $b_i < d_i$. This represents a topological feature that is "born" at b_i and "dies" at d_i. For points in the plane, these features roughly represent the "holes" present

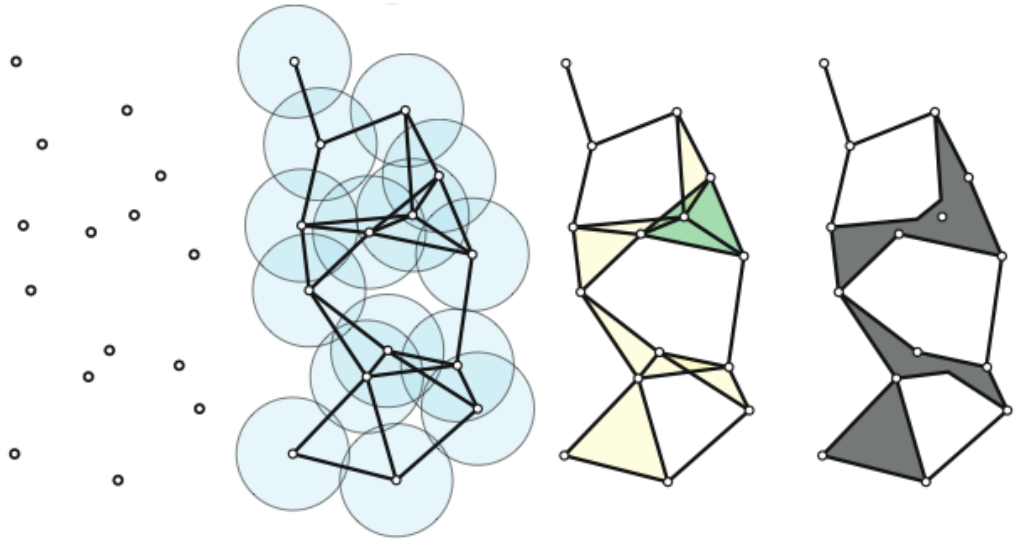

Figure 3: An example of the Vietoris-Rips complex for a (planar) set of points for a given parameter δ. From University of Illinois.

between the points. Considering these pairs as points in the plane, one obtains the *persistence diagram*. If instead, one considers them as intervals it is typically called a *barcode*. With this definition $b_i, d_i \in \{1, \ldots, n\}$, but we can generalize this to $\mathbb{R}$ by associating a increasing sequence of real numbers with the filtration. With the Vietoris-Rips complex, this is typically done by associating the δ value that was used to compute the complex in the filtration.

Persistence diagrams are stable with respect to input noise [5]. This makes the diagrams a good representation of the topology and useful for classification task, etc.

1.1.1 Metrics

There exist some standard metrics to compare two persistence diagrams with each other. First, there is the Wasserstein distance, also called the earth mover's distance, which represents the minimum "cost" to transform one diagram into the other. The p-Wasserstein distance between two PDs B and B' for $1 \leq p < \infty$ is defined as:

$$W_p(B, B') = \inf_{\gamma: B \to B'} \Big(\sum_{u \in B} ||u - \gamma(u)||_\infty^p \Big)^{1/p}$$

Another standard metric is the bottleneck distance, which is simply the ∞-Wasserstein distance:

$$W_\infty(B, B') = \inf_{\gamma: B \to B'} \sup_{u \in B} ||u - \gamma(u)||_\infty$$

One issue with these metrics is that they require to solve a minimum weight matching problem between two persistence diagrams. This can quickly become costly when the size of the diagrams grows. Another problem is that the metrics do not have a derivative which can be used for calculating gradients. Therefore, they are not very suitable cost functions for deep

learning purposes. For the neural network I will thus resort to sorting the birth-death pairs on persistence value and comparing the persistence values using mean squared error (MSE). This method is less theoretically sound, but is "good enough" for the purpose of exploring whether a machine can learn topology.

1.2 Prior works

There has already been effort to incorporate persistent homology together with deep learning. Much of the research is focused on transforming the persistence diagram into some representation, typically something in vector space, that can be more easily used for data mining and machine learning tasks. The idea is then to calculate this persistence diagram representation for each entry in the dataset and feed this information to a neural network or other machine learning tool.

One such representation is the persistent landscape [3]. The persistence landscape of the birth death pairs (b_i, d_i) is a sequence of functions $\lambda_k : \mathbb{R} \to [0, \infty]$, $k = 1, 2, 3, \ldots$. Here, each birth death pair is associated with a piece-wise linear function and $\lambda_k(x)$ is the k-th largest value of these functions for some position x. This summary obeys a strong law of large numbers and a central limit theorem. It is shown that the persistence landscape is a stable summary statistic with respect to the L. The landscape distance also gives lower bounds for the bottleneck and Wasserstein distances. The author has also developed efficient algorithms for calculating the persistent landscapes in $O(n^2)$, as well as averages over N landscape in $O(n^2 N \log N)$, similar with calculating the L^p distance [4]. Once calculated, persistence landscapes are significantly faster to use in standard tasks such as classification.

Another approach is to transform the persistence diagram into a feature map or image, that is stable with respect to noise and common distance metrics (Wasserstein, bottleneck). One way to create such an image is to use a sum of Gaussian functions centered at each persistence pair in the diagram [7] [1]. For such a representation to be stable the diagonal of the diagram, containing points with low persistence, should have a value of zero. One possible way to resolve this is to add negative Gaussian functions centered on each point reflected below the diagonal, which means the functions would cancel at the diagonal [7]. Another possible solution is to add a continuous weight function to the sum that is zero at the diagonal [1]. In the papers it is shown that persistence images can be generated efficiently, are more accurate on standard classification tasks than persistence landscapes and improve upon persistence diagrams on performance.

A different approach that has been developed is to not use a fixed representation of the persistence diagram, but instead use a novel topological layer to learn said representation using deep neural networks [6]. Such a layer uses a continuous and differentiable function on the input from the persistence diagram to generate a parametrized representation that can be used and trained by the network. It is shown that the proposed layer is stable with respect to the 1-Wasserstein distance and that a network with the new layer can even outperform other state-of-the-art networks on certain image classification tasks.

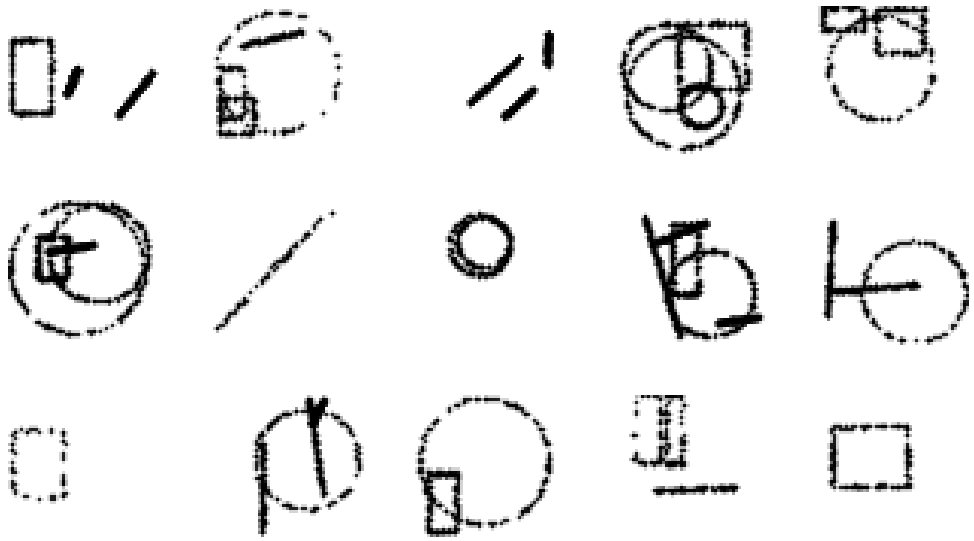

Figure 4: The first 15 input images of the validation set for experiment 1.

2 Experiments

The goal is to see if it is possible at all for neural networks to learn to extract topological features, as this is a necessary first step before more advanced work can be accomplished. To do this, a simple prediction task is designed. A simple dataset is generated of images with different topological features. Then, a convolutional neural network is trained to predict the birth-death barcode of each image.

The expected barcode of an image is calculated using a tool called *Ripser* [2]. The birth-death pairs are sorted on decreasing persistence and restricted to a certain amount of pairs. This means that low persistence pairs might be lost, but this is not a problem since these are considered noise anyway. The cost function used for comparing the prediction with the expected barcode is simply the mean squared error (MSE) between the persistence values of the pairs.

2.1 Experiment 1

In the experiment we would want to see if a neural network could accurately predict topological features. For this goal an experiment was designed which is quite simple. It consisted of different shapes in a window, possibly intersecting with each other. These intersections would create holes which would appear as a persistent feature in the output of Ripser. The neural network could then train to detect such holes in the image.

2.1.1 Data generation

For the experiment a custom dataset was generated. The dataset consists of images of a 64-by-64 window containing different shapes that possibly intersect, which would result in a difference in topology. The shapes chosen consisted of lines, rectangles and circles of various sizes, as long as they were contained inside the window. These shapes were chosen since they are simple to generate. For the circles and rectangles I used a minimum and maximum size in order to avoid very large or small rectangles that would not work well with turning it into a 64-by-64 image. The type of shape and the number of shapes were chosen uniformly at random, where the number was chosen between one and five.

From the generated shape data both the persistent barcode using the Ripser tool and input images for the network needed to be calculated. However, Ripser is defined only on point sets, not on shape information. Thus, before we can calculate the barcode we had to sample the shapes to create a point set that approximates it. This was done with a uniformly random choice of between 50 to 200 sample point at random positions on the shape. In order to give the network the same information as the Ripser tool, it was chosen to generate an image of the point set instead of the shapes. The images were generated by rasterizing the data, see Figure 4 for examples. The input images were black and white and had a size of 64-by-64, chosen such that it could capture most of the shape and hole information while not resulting in very large images (which would also require more neurons in the network, making learning slower).

The expected barcode was generated using the aforementioned Ripser tool. This calculated persistence values using the Vietoris-Rips complex. These values are equal to $b - a$ for each pair (a, b) in the PD, and are sorted decreasingly. To have a consistent input size for the network, only the first 20 values are stored, after which the values were deemed insignificant (noise).

2.1.2 Network structure

For the neural network a standard convolutional structure was used, namely a sequential structure consisting of a number of convolutional layers with max-pooling and dropout layers in between, followed by a number of dense layers with dropout layers. See Figure 5 for an example of a typical structure. The network gets its greyscale images of size 64-by-64 in the input layer. Then there follows a single convolutional layer with 64 kernels of size 3-by-3. Then a number of "convolutional blocks" follow, that consist of two sequential convolutional layers with 128 3-by-3 kernels, followed by a 2-by-2 max pooling layer and a 0.5-dropout layer. Multiple different number of convolutional blocks were experimented with. After the convolutional blocks follows a series of dense blocks which are used to make the final prediction. A dense block consists of a dense layer of size 256 followed by a 0.5-dropout layer. Finally, the output layer is another dense layer with the given output size.

Three different network settings were experimented with, namely a network with two convolutional blocks and one dense block, a network with three convolutional blocks and two dense blocks, and one with three convolutional blocks and two dense blocks. These networks are referred to as CONV2-1, CONV3-2, and CONV3-3. One thing to keep in mind is that fewer

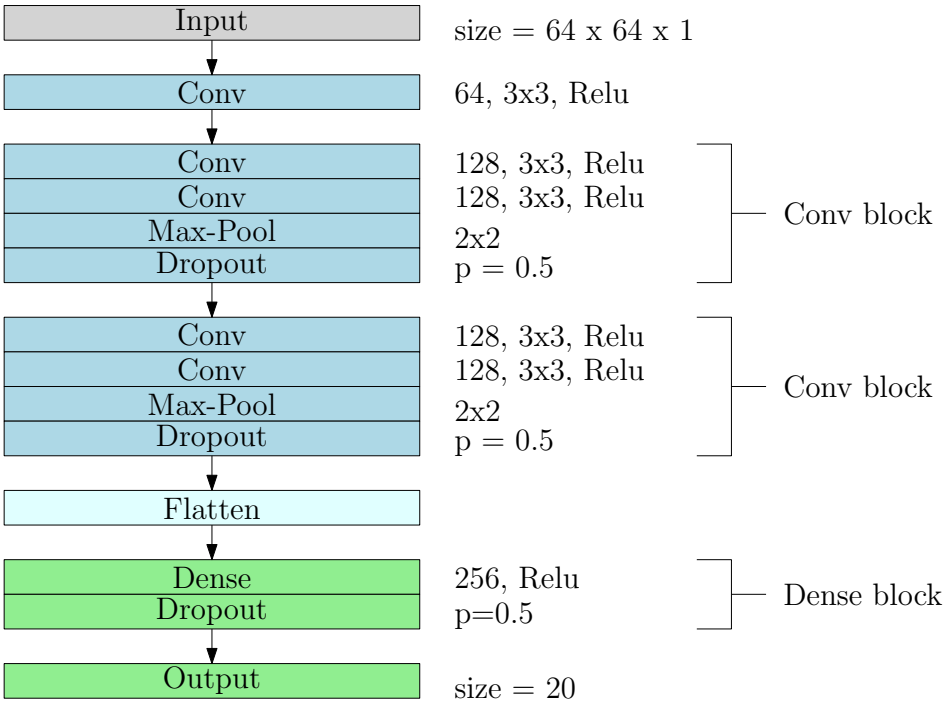

Figure 5: An example of the typical structure of the convolutional neural network. Here one network is shown with 2 convolutional blocks and 1 dense block.

convolutional blocks also results in fewer max-pooling layers, which can counter-intuitively mean that the smaller network has more weight parameters due to the first dense layer acting on a larger image.

2.1.3 Results

The network was trained for 100 epochs with a batch size of 128, see Figure 6. As can be seen, all networks end up at roughly the same loss values after training. The networks CONV2-1, CONV3-2, and CONV3-3 achieved MSE scores of 0.60, 0.55, 0.86 respectively, though the latter network had much better loss scores in previous epochs. From now on we will take the CONV3-2 network to explore further, since it achieved the best loss score. In Table 1 some statistics are given on the loss for the entire validation set, such as the mean and standard deviation.

In Appendix B.1, the network's predictions are given for the first 20 validation test cases. In general, it seems as if the network can get quite close to the correct answer. However, looking at some more examples I also observed that the network can still be off by quite a margin for some of the larger persistence pairs. Since there were at most five shapes, in almost all test cases there would be fewer than 10 persistence pairs with a value higher than 1.0 (on average a test case has 3.53 persistence values above 1.0, though the maximum is 20). This means the mean squared error might look good due to a low error on the last 10-15 values which are always close to 0 but the prediction could still be off for the few largest values. In some cases, the network seemed to have simply learned to find the mean persistence values

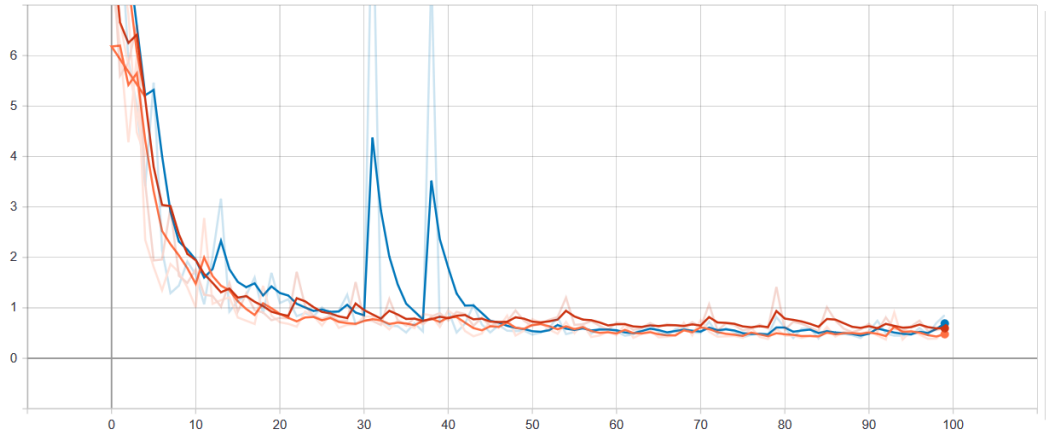

Figure 6: Graph of the MSE loss of experiment 1 for the different networks. Orange, blue, and red show CONV2-1, CONV3-2, and CONV3-3 respectively.

for certain cases. For example, the machine would typically underestimate large persistence values (indicating there is a significant hole) as well as still give some small value even when there were no holes at all. This either indicates that the task was not properly setup or that it might still be too difficult for the tools that were used.

	Loss (MSE)
Mean	0.548
Std. dev.	0.537
0%	0.0.000277
25%	0.145
50%	0.413
75%	0.771
100%	4.547

Table 1: Loss (MSE) statistics of the network CONV3-2 for experiment 1.

Another way to analyze the predictions of the network is to look at the percentage the network guesses the "correct" number of holes present according to some threshold when we consider a persistence value a hole. For this experiment a threshold value of 1.0 was used. In Appendix D.1 the entire confusion matrix for the experiment can be seen. In total, the network guesses the correct number of holes according to this threshold in 37% of the cases. That percentage does not seem very good, though we can see from the confusion matrix that the network at least typically is within one or two holes of the correct answer. The network appears to slightly overestimate the number of holes when there are few and underestimate when there are many. One interesting thing to note is that the network does seem to score well for cases with zero or one holes, with a precision and recall of 0.99 and 0.92 respectively for zero holes and 0.91 and 0.74 respectively for one hole. That means that at the very least the network is actually reliably finding whether or not there are holes present at all. In total, this confusion matrix shows that the network is not very good at guessing the exact number of holes, but that it does not completely output an average of the dataset.

Figure 7: The first 15 input images of the validation set for experiment 2.

2.2 Experiment 2

Since the first experiment did not provide us with quite the result that would show that a neural network could learn persistent topological features, a second experiment was done. One possible reason for this might be that the dataset was still too complex, since it contained multiple different shapes which could potentially result in many intersections and holes. The sampling rate of images could also be quite small, which resulted in shapes with many small holes in them. Therefore, it was decided to perform a second experiment, one which was even more simple than the first.

2.2.1 Data generation

The dataset that was generated consisted of four lines in arbitrary configurations. Compared to the previous experiment, there was now only one shape instead of three and a fixed amount (four). This meant we knew exactly how many intersections there could possibly be and therefore how many holes there could be. With four lines, there can be at most three holes created. See Figure 7 for examples of the dataset.

Since we knew that there could be at most three holes created, only the highest three persistence values from the Ripser tool were important for defining the topological features. The tool would still output many more values due to the sampling, but those would all be close to zero. Therefore, only the highest three persistence values would be used for the output that the network needed to guess. This also avoids any issue we had that the mean value is brought down by many output values always being extremely small.

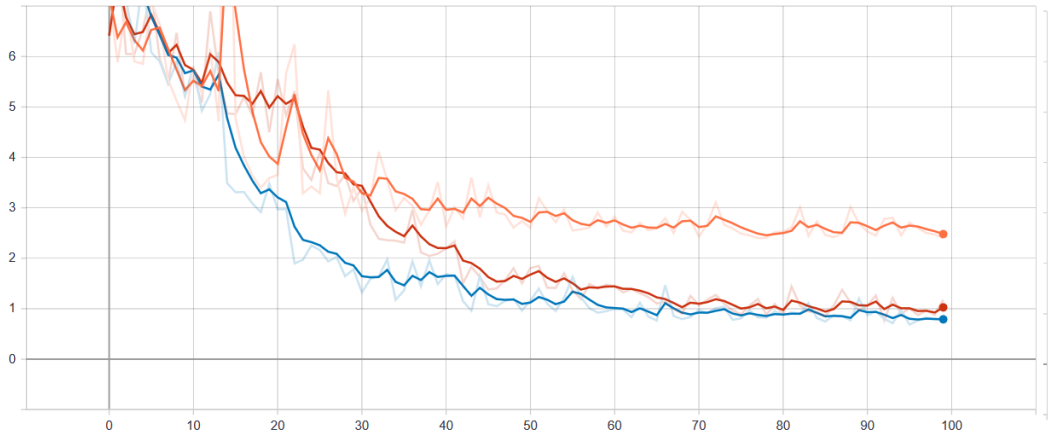

Figure 8: Graph of the MSE loss of experiment 2 for the different networks. Orange, blue, and red show CONV2-1, CONV3-2, and CONV3-3 respectiveyl.

2.2.2 Network structure

The network structure was similar to that of the first experiment, shown in Figure 5. That is, it consists of a number of convolutional layers (with 2D max-pool and dropout) followed by a number of dense layers (with dropout). The only difference is that now the output layer has just a size of three, since we only want three values as output instead of 20. Again, three different settings were experimented with, the same settings as for the previous experiment. These are referred to as CONV2-1, CONV3-2, CONV3-3.

2.2.3 Results

The network was trained for 100 epochs on the dataset with a batch size of 128. In Figure 8 we can see the loss function plotted against the number of epochs trained. The settings with three convolutional blocks seem to outperform the network with two, with CONV2-1 having a loss of 2.40. There is a small difference between the two networks CONV3-2 and CONV3-3, with CONV3-2 performing slightly better than CONV3-3 with 0.78 against 1.18 respectively. In Table 2 we can see some additional statistics on the loss for all 2000 validation test cases. The table shows that the loss distribution is quite skewed due to some outliers with a very high cost (due to MSE), which also causes the standard deviation to be quite large. Looking at the median we can see that for more than half the cases it has a very low mean squared error, indicating that if we disregard the outliers the network can find the topological features accurately in many cases.

In Appendix B.2 20 predictions of the network are shown against the correct solution. We can observe that there are still some cases where the network is off by quite a bit, but on average it seems to be able to find how many holes there are present and roughly how large these are.

Once again, we can use a threshold value of 1.0 to see the percentage of cases where the network predicted the correct number of holes. In Table 3 the confusion matrix for experiment 2 is given. As we can see, the average prediction is correct in roughly 69% of cases, with the

	Loss (MSE)
Mean	1.477
Std. dev.	3.346
0%	0.0001030
25%	0.2040
50%	0.5296
75%	1.432
100%	55.64

Table 2: Loss (MSE) statistics of the network CONV3-2 for experiment 2.

network being the most precise when predicting zero holes (79.5%) and having the best recall for cases with two holes (75.7%). What we can also notice is that the network rarely predicts three holes, it only predicts this in $\frac{19}{2000} = .95\%$ of the cases while it occurs 4.4% of the cases, resulting in a recall of just 11.7%. A reason for this low recall might be that these cases do not occur often in the dataset, thus the network has too little incentive to be accurate in these cases.

		Correct				
		0 holes	**1 holes**	**2 holes**	**3 holes**	**Precision**
Prediction	**0 holes**	276	69	2	0	0.7954
	1 holes	108	632	138	6	0.7149
	2 holes	4	213	457	77	0.6085
	3 holes	0	0	7	11	0.6111
	Recall	0.7113	0.6915	0.7566	0.1170	0.6880

Table 3: Confusion matrix of network CONV3-2 for experiment 2 using a threshold value of 1.0 for a hole.

Next, we wanted to test if the network is robust against slight changes in the image, as this is one of the main goals of topology. For this purpose, the neural network was run on the input images with various transformations applied to them to see what the effect on the prediction would be. In Appendix C.1 four tables are shown with different metrics to compare the prediction before and after transformation. One problem with the transformations is that holes disappear when parts are outside the 64-by-64 window, which can happen for both translation and scaling. Still, we can see that the mean absolute difference before and after rotation and translation is close to zero and the ratio is close to one, indicating that the network predicted values do not change in any particular direction. This is to be expected, as the topological features should not change after rotation or translation and the network is trained on a dataset that is randomly generated, thus it should not have any biases.

For scaling the absolute difference and ratio are not zero and one respectively, as we would expect that the holes scaled up would give larger persistence values. The strange negative mean difference when scaling with a factor 2 is likely due to the issue with the holes disappearing when outside the window. We can also observe that the MSE for rotation before and after rotation is around 0.45 and that for just a (+1,+1)-translation the MSE is only 0.09, which seems to indicate that the network is not dependent on particular pixel arrangements.

3 Discussion and conclusion

In this report I have discussed the application of topological features with deep learning with two simple experiments. For these experiments custom datasets were created with the goal of showing that neural networks are capable of learning persistent features of an image. These features roughly correspond to the size of the holes in the images.

The first experiment showed decent results in guessing the persistent values of the images. The mean squared error was quite low and the confusion matrix (see Appendix D.1) showed the network could get relatively close to the correct number of holes, especially if there were few holes present. However, the large size of the output meant that many values were close to zero, which resulted in the mean error being smaller on the surface. Furthermore, the different shapes at arbitrary sizes meant the images could get quite complicated with many different intersections and holes, which could not always be captured in a 64-by-64 images. This all resulted in the network not always being able to detect the correct number of holes and giving more "conservative" that are closer to the dataset's average. These issues were reason to create a second experiment.

In the second experiment better result were achieved. The network seemed to actually learn to accurately predict when there were holes present and what their rough size was. Overall, it could predict the correct number of holes in 69% of the cases. However, there were still some issues with the dataset, as there were too few cases with three holes present. A possible improvement would be to have an equal amount of test cases for each number of holes present. One problem with the current network is that it rarely predicts three holes, likely due to there not being enough of these cases in the training set.

For the second experiment the robustness with respect to certain transformations was also tested. The results here seem to indicate that the network was not particular sensitive to such translations. For example, the MSE after rotation was only 0.45 and very small for a small translation. However, the values here are hard to correctly analyze due to a large issue with the method, namely that the holes can clip outside the 64-by-64 image making them disappear for the network. This makes the analysis of large translation and scaling practically useless.

Overall, the project shows that there is some promise that neural networks can accurately predict topological features of images. In a very simple dataset with just four intersecting lines it was able to predict persistent topological features quite decently, though it is hard to judge just how well it performs. Of course, in a real dataset with actual images there are many more additional challenges for the network to overcome, so it is not immediately clear if this approach works in those cases. As previously said, it also remains to be seen whether learning topological features has any benefits for machine learning.

A next question is whether or not training the network to explicitly conserve topological features of an image has any effect on the performance on different machine learning tasks, such as classifying images. Given more time, I would have liked to experiment with a network similar to an auto-encoder, that learns to extract a code (typically a vector of a given, small size that captures the input) from the input images. This code is then used to predict the topological features and perform the classification simultaneously, meaning the network tries to optimize both. See Figure 9 for a sketch of how such a network might look like. The

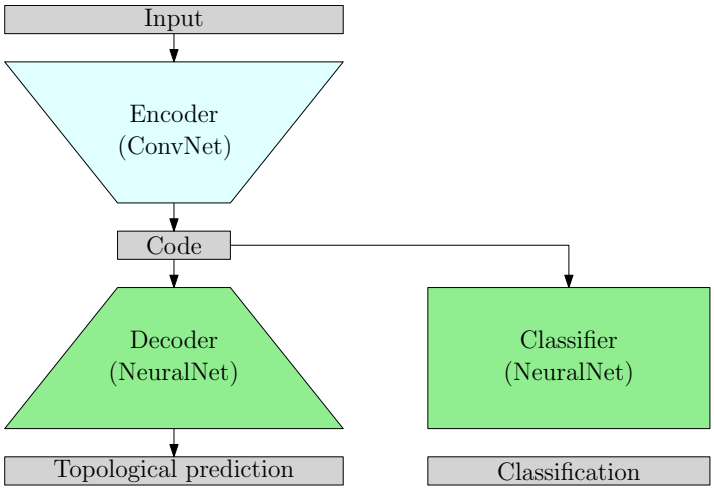

Figure 9: An idea for the structure of a network that performs classification with a code that also captures topological features.

prediction of topological features could be done similarly to shown here. The advantage would be that we know the network performs classification on a code which represents the topological information. It would be interesting to see if explicitly training the network to capture this topological information benefits it, and if so on what tasks, or if the inflexibility only hinders learning.

References

[1] Henry Adams, Tegan Emerson, Michael Kirby, Rachel Neville, Chris Peterson, Patrick Shipman, Sofya Chepushtanova, Eric Hanson, Francis Motta, and Lori Ziegelmeier. Persistence images: A stable vector representation of persistent homology. *The Journal of Machine Learning Research*, 18(1):218–252, 2017.

[2] Ulrich Bauer. Ripser 1.0.1. `http://ripser.org`, 2016.

[3] Peter Bubenik. Statistical topological data analysis using persistence landscapes. *The Journal of Machine Learning Research*, 16(1):77–102, 2015.

[4] Peter Bubenik and Paweł Dłotko. A persistence landscapes toolbox for topological statistics. *Journal of Symbolic Computation*, 78:91–114, 2017.

[5] David Cohen-Steiner, Herbert Edelsbrunner, and John Harer. Stability of persistence diagrams. *Discrete & Computational Geometry*, 37(1):103–120, 2007.

[6] Christoph Hofer, Roland Kwitt, Marc Niethammer, and Andreas Uhl. Deep learning with topological signatures. In *Advances in Neural Information Processing Systems*, pages 1634–1644, 2017.

[7] Jan Reininghaus, Stefan Huber, Ulrich Bauer, and Roland Kwitt. A stable multi-scale kernel for topological machine learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4741–4748, 2015.

Files and data

All relevant files and the two datasets can be foud in the following git repository:
`https://github.com/StijnSlot/PersistentDeepLearning`

A Self-reflection

The project was very much a learning experience for me. Previously, I had done some research into (convolutional) neural networks, but I did not have much practical experience with the entire process of training a network. However, the bigger challenge initially was that I had little knowledge of topology. This meant that the first few weeks were mostly spent on catching up on topological data analysis, reading through the book "Computational Topology: An introduction" from Edelsbrunner and Harer. The material in this book can be quite difficult to wrap your head around reading it for the first time, so it took me a couple of tries in order to get through the most important parts for the project. It has given me some insights into a field I was previously somewhat unfamiliar with.

The big challenge of the project was that it was something new for everyone involved. This was exciting as there did not seem to be much research into this area out there. However, it also meant that at times the direction to take was unclear for me. The exact goal of the project did not completely materialize until late into the project. For me it could be difficult as I sometimes did not have a clear idea of what to do and there were not any online resources to fall back on. However, this was also what made the project interesting in the first place.

Overall, I am satisfied with what I have achieved, though I would have liked to have gone on further to find more exciting results. Due to the limited time of an honors project (e.g. in comparison to a master's project) and the fact that the area was new to me, I think it is understandable that we did not get much further with the project. We did still achieve some smaller results and I learned a great deal on topology and practical skills of the entire process of training a network, so I would still call the project a success.

B Predictions

Here the correct answer versus the prediction by the neural network are shown of the first 20 cases in the validation set.

B.1 Experiment 1

```
Correct  : 13.2 5.3 4.0 3.5 0.7 0.5 0.3 0.2 0.2 0.1 0.1 0.1 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Predicted: 11.6 7.2 5.2 3.1 1.8 1.1 0.8 0.5 0.4 0.3 0.3 0.2 0.2 0.2 0.2 0.1 0.1 0.1 0.1 0.1

Correct  : 11.5 6.4 4.7 4.2 1.0 0.6 0.5 0.4 0.3 0.3 0.3 0.3 0.3 0.2 0.2 0.1 0.1 0.1 0.1 0.1
Predicted: 10.3 6.9 5.3 3.6 2.3 1.6 1.1 0.7 0.5 0.4 0.3 0.3 0.2 0.2 0.2 0.2 0.1 0.1 0.1 0.1

Correct  : 16.3 6.8 0.4 0.3 0.3 0.2 0.2 0.2 0.2 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1
Predicted: 16.4 7.0 2.1 0.9 0.5 0.3 0.3 0.2 0.2 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.0 0.0

Correct  : 17.8 9.6 9.1 5.9 5.4 0.6 0.5 0.4 0.4 0.4 0.3 0.3 0.3 0.3 0.3 0.2 0.2 0.2 0.2 0.2
Predicted: 15.6 9.8 7.6 6.2 4.7 3.5 2.6 1.7 1.2 0.9 0.7 0.6 0.5 0.4 0.4 0.4 0.3 0.3 0.2 0.2

Correct  : 0.3 0.1 0.1 0.1 0.1 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Predicted: 0.5 0.3 0.2 0.1 0.1 0.1 0.1 0.1 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0

Correct  : 23.4 10.2 1.8 1.4 0.6 0.4 0.4 0.3 0.3 0.2 0.2 0.2 0.2 0.2 0.2 0.1 0.1 0.1 0.1 0.1
Predicted: 22.7 7.4 3.7 1.9 1.0 0.6 0.5 0.3 0.3 0.2 0.2 0.2 0.2 0.1 0.1 0.1 0.1 0.1 0.1 0.1

Correct  : 0.3 0.2 0.2 0.1 0.1 0.1 0.1 0.1 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Predicted: 2.0 0.6 0.3 0.2 0.2 0.1 0.1 0.1 0.1 0.1 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0

Correct  : 6.9 4.5 0.3 0.2 0.1 0.1 0.1 0.1 0.1 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Predicted: 6.1 3.6 1.5 0.8 0.5 0.3 0.3 0.2 0.2 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.0 0.0

Correct  : 13.1 9.2 7.6 4.8 3.5 1.9 1.1 1.1 0.9 0.6 0.6 0.6 0.5 0.5 0.5 0.4 0.3 0.3 0.2 0.2
Predicted: 10.9 8.9 7.7 6.3 4.8 3.5 2.5 1.7 1.2 0.9 0.7 0.6 0.5 0.4 0.4 0.3 0.3 0.3 0.2 0.2

Correct  : 10.9 4.4 2.7 1.9 1.2 1.1 0.4 0.4 0.4 0.3 0.3 0.3 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2
Predicted: 8.9 5.6 3.8 2.2 1.2 0.8 0.6 0.4 0.3 0.3 0.2 0.2 0.2 0.1 0.1 0.1 0.1 0.1 0.1 0.1

Correct  : 17.9 13.2 7.6 6.6 4.9 1.9 0.4 0.4 0.3 0.3 0.2 0.2 0.2 0.2 0.2 0.1 0.1 0.1 0.1 0.1
Predicted: 17.1 10.0 6.9 4.8 3.2 2.2 1.6 1.1 0.8 0.6 0.5 0.4 0.3 0.3 0.3 0.3 0.2 0.2 0.2 0.1

Correct  : 19.8 10.9 10.6 4.4 3.2 2.6 1.9 1.7 1.1 1.0 0.9 0.5 0.5 0.4 0.4 0.3 0.3 0.3 0.3 0.3
Predicted: 18.4 8.7 6.0 4.7 3.5 2.6 2.0 1.3 1.0 0.7 0.6 0.5 0.4 0.4 0.3 0.3 0.2 0.2 0.2 0.2

Correct  : 11.2 9.3 0.5 0.3 0.2 0.2 0.2 0.2 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1
Predicted: 10.1 7.7 2.0 0.7 0.3 0.3 0.2 0.2 0.2 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.0 0.0 0.0 0.0

Correct  : 20.2 10.0 0.8 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Predicted: 18.8 8.0 2.2 0.9 0.4 0.3 0.3 0.2 0.2 0.2 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.0 0.0

Correct  : 17.4 13.2 8.8 6.3 3.4 1.0 0.7 0.4 0.3 0.3 0.3 0.3 0.3 0.2 0.2 0.2 0.2 0.2 0.2 0.1
Predicted: 16.9 11.7 7.7 5.4 3.6 2.5 1.8 1.2 0.9 0.6 0.5 0.4 0.4 0.3 0.3 0.3 0.2 0.2 0.2 0.2

Correct  : 10.6 6.3 0.4 0.3 0.3 0.3 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.1 0.1 0.1 0.1 0.1 0.1 0.1
Predicted: 9.2 5.1 1.5 0.6 0.3 0.2 0.2 0.2 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.0 0.0 0.0 0.0

Correct  : 8.0 1.5 0.4 0.4 0.4 0.2 0.2 0.1 0.1 0.1 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Predicted: 7.1 3.3 1.1 0.6 0.4 0.3 0.2 0.2 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.0 0.0 0.0 0.0

Correct  : 40.5 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Predicted: 35.1 -0.2 0.1 0.3 0.1 0.1 0.2 0.1 0.1 0.1 0.1 0.1 0.1 0.0 0.1 0.1 0.0 0.0 -0.0 -0.0

Correct  : 17.9 15.8 0.6 0.5 0.1 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Predicted: 17.8 11.6 1.9 0.7 0.4 0.4 0.3 0.2 0.2 0.2 0.1 0.1 0.1 0.1 0.1 0.1 0.0 0.0 0.0 0.0

Correct  : 9.8 9.8 3.4 1.3 0.8 0.3 0.3 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2
Predicted: 8.9 7.1 5.4 3.4 2.0 1.4 0.9 0.6 0.5 0.3 0.3 0.2 0.2 0.2 0.2 0.1 0.1 0.1 0.1 0.1
```

B.2 Experiment 2

```
Correct  : 4.04927  2.45165  0.547832
Predicted: 3.710838 1.684036 0.524424
```

```
Correct  : 1.9412  0.51353 0.46096
Predicted: 5.028775 0.901758 0.405372

Correct  : 0.807706 0.379835 0.299589
Predicted: 0.834467 0.487155 0.384819

Correct  : 6.81527  1.409567 0.501454
Predicted: 7.142719 1.024133 0.404103

Correct  : 0.458439 0.39173  0.321437
Predicted: 1.029503 0.538804 0.416943

Correct  : 0.45305  0.302403 0.295773
Predicted: 0.883609 0.499666 0.390405

Correct  : 0.39369  0.35372  0.340605
Predicted: 0.737548 0.441797 0.348353

Correct  : 2.4645   0.310998 0.261927
Predicted: 1.858838 0.638941 0.431218

Correct  : 0.85469  0.57582  0.526818
Predicted: 1.049743 0.549666 0.421996

Correct  : 5.96634 0.52678 0.37914
Predicted: 6.698636 1.573268 0.431028

Correct  : 4.29115 1.48777 0.58437
Predicted: 3.882682 2.203536 0.590345

Correct  : 14.74203  2.30622  0.30078
Predicted: 11.484598  1.685348  0.39064

Correct  : 1.1708   0.32775  0.274413
Predicted: 3.017384 0.458616 0.382765

Correct  : 2.76448  0.901068 0.56723
Predicted: 1.282648 0.572911 0.418961

Correct  : 1.11692  0.53677  0.463406
Predicted: 2.818947 0.927566 0.468931

Correct  : 6.1794   0.373061 0.322395
Predicted: 7.047706 0.448348 0.353643

Correct  : 2.57822  0.411067 0.308878
Predicted: 3.87229  0.683087 0.402156

Correct  : 18.40229   0.360517  0.267253
Predicted: 20.332745  0.542745  0.119596

Correct  : 1.73669 0.6375  0.55439
Predicted: 3.110613 1.339332 0.51291

Correct  : 1.55686 1.418   0.67134
Predicted: 1.274821 0.588513 0.427763
```

C Transformations

Here the robustness of the neural network with respect to transformations is shown. Three different transformations are tested: rotation (in 90 degrees interval), translation, and scaling. Different metrics of comparing the correct answer, original prediction, and new prediction after transformation are shown in Tables 4, 5, 6, and 7.

C.1 Experiment 2

	rot90	rot180	rot270	trans1_1	trans5_5	trans-5_-5	scale1.2	scale1.5	scale2
mean	0.003	-0.019	-0.040	-0.023	-0.302	-0.174	0.701	0.527	-0.782
std. dev.	1.187	1.196	1.184	0.537	1.263	1.107	1.372	2.883	4.291
0%	-7.526	-6.957	-6.145	-3.810	-10.091	-7.154	-8.673	-21.521	-20.508
25%	-0.559	-0.552	-0.589	-0.244	-0.677	-0.515	0.044	-0.259	-2.661
50%	-0.005	0.004	-0.026	0.004	-0.018	-0.027	0.671	0.873	0.154
75%	0.579	0.536	0.544	0.234	0.289	0.316	1.463	2.170	1.932
100%	5.332	5.660	4.878	2.315	6.271	4.918	5.881	9.224	11.971

Table 4: Mean absolute difference between prediction values before and after transformation.

	rot90	rot180	rot270	trans1_1	trans5_5	trans-5_-5	scale1.2	scale1.5	scale2
mean	1.022	1.020	1.014	1.002	0.981	0.995	1.142	1.205	1.147
std. dev.	0.239	0.228	0.223	0.098	0.175	0.175	0.274	0.510	0.769
0%	0.351	0.293	0.229	0.542	0.219	0.242	0.216	0.088	0.088
25%	0.908	0.909	0.903	0.963	0.909	0.923	1.011	0.947	0.634
50%	0.999	1.002	0.994	1.001	0.995	0.994	1.109	1.166	1.048
75%	1.099	1.098	1.100	1.040	1.071	1.065	1.218	1.389	1.420
100%	3.625	3.366	2.868	2.106	2.524	3.628	3.142	4.890	7.421

Table 5: Ratio between the sum of prediction values before and after transformation.

	rot90	rot180	rot270	trans1_1	trans5_5	trans-5_-5	scale1.2	scale1.5	scale2
mean	0.449	0.463	0.455	0.090	0.498	0.376	0.727	2.654	5.584
std. dev.	1.065	1.104	1.009	0.239	1.823	1.092	1.542	7.515	12.760
0%	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
25%	0.019	0.018	0.017	0.003	0.011	0.009	0.046	0.191	0.319
50%	0.129	0.123	0.122	0.021	0.076	0.072	0.267	0.899	1.530
75%	0.455	0.440	0.459	0.083	0.309	0.287	0.815	2.509	4.792
100%	21.614	19.227	16.243	5.446	41.592	17.767	25.738	174.606	161.432

Table 6: MSE between prediction values before and after transformation

D Confusion Matrix

D.1 Experiment 1

See Figure 8 for the confusion matrix for experiment 1.

	rot90	rot180	rot270	trans1_1	trans5_5	trans-5_-5	scale1.2	scale1.5	scale2
mean	0.785	0.775	0.747	0.771	1.034	0.958	1.645	3.444	5.879
std. dev.	1.687	1.368	1.389	1.456	2.431	1.948	2.826	8.290	12.678
0%	0.000	0.001	0.001	0.001	0.001	0.000	0.001	0.003	0.001
25%	0.083	0.092	0.085	0.090	0.096	0.091	0.167	0.368	0.472
50%	0.295	0.307	0.307	0.311	0.345	0.326	0.743	1.514	1.975
75%	0.862	0.854	0.842	0.838	0.978	1.000	1.966	3.824	5.630
100%	32.106	19.222	24.728	21.033	41.426	33.405	37.783	174.424	161.184

Table 7: MSE between correct values and prediction values after transformation

D.2 Experiment 2

See Figure 9 for the confusion matrix for experiment 2.

									Correct													
Pred.	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**Precision**
0	177	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.99
1	14	338	18	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.91
2	1	95	20	4	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.16
3	0	20	155	35	7	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.16
4	0	2	82	63	29	4	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.16
5	0	0	11	89	50	28	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.15
6	0	0	1	19	71	48	16	10	2	1	0	0	0	0	0	0	0	0	0	0	0	0.10
7	0	0	0	4	25	65	50	27	12	8	3	0	0	0	0	0	0	0	0	0	0	0.14
8	0	0	0	0	2	21	37	35	18	17	8	0	0	0	0	1	0	0	0	0	0	0.13
9	0	0	0	0	0	5	21	29	30	40	20	10	5	1	1	0	1	0	0	0	0	0.25
10	0	0	0	0	0	0	2	2	9	14	11	13	9	3	2	2	0	0	0	0	0	0.16
11	0	0	0	0	0	0	0	0	0	1	2	1	2	1	3	1	0	0	0	0	0	0.09
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Recall	0.92	0.74	0.07	0.16	0.16	0.16	0.12	0.26	0.25	0.49	0.25	0.04	0	0	0	0	0	0	0	0	0	0.37

Table 8: Confusion matrix for experiment 1 using a threshold value of 1.0 for a hole.

		Correct				
		0 holes	**1 holes**	**2 holes**	**3 holes**	**Precision**
Prediction	**0 holes**	276	69	2	0	0.7954
	1 holes	108	632	138	6	0.7149
	2 holes	4	213	457	77	0.6085
	3 holes	0	0	7	11	0.6111
	Recall	0.7113	0.6915	0.7566	0.1170	0.6880

Table 9: Confusion matrix of network CONV3-2 for experiment 2 using a threshold value of 1.0 for a hole.