Report of the TDA and Deep learning project

By :

Anass AL AMMIRI

Wala Bouzouita

Ruiqi Wang

Supervised by :

Bertrand Michel

bertrand.michel@ec-nantes.fr

Applied Mathematics : Data science

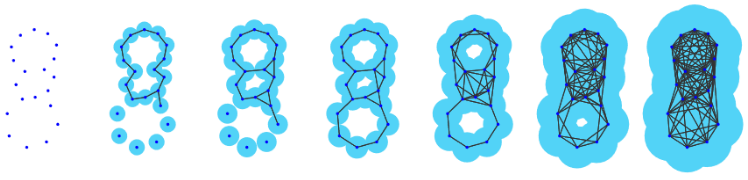
Table of Contents

Table of Contents 2
Topological data analysis: The tools 3
PersLay Layer: 3
Application: 4
Data: 4
The data set: 4
Why this data set? 4
Data preprocessing: 4
The problem and the topological descriptors: 5
Deep Learning and Topological Data analysis: 7
The results: 7
Conclusion: 12
Appendix 1: Github 13
Appendix 2: Graph-model-1 13
Appendix 3: Graph-model-2 14

Topological data analysis: The tools

In the first part of the project, we introduced some tools of persistence homology, mainly based on the article **An introduction to Topological Data Analysis: fundamental and practical aspects for data scientists** by *Frédéric Chazal* and our supervisor *Bertrand Michel.* We will use these tools with some new tools for our application. So, we briefly mention them here:

1. Rips complex and from it Rips diagrams.
2. Alpha complex and alpha diagrams.



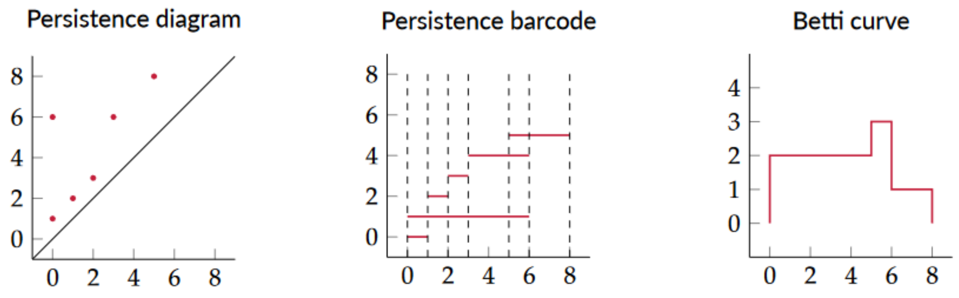
In this figure we have a growing union of balls and the 1-skeleton of the corresponding Cech complex. Here, the complexes illustrate the births and deaths of three holes. The corresponding birth-death pairs are generally used to get persistence diagrams. Since Cech complex is often computationally expensive, we use other variants like Rips and Alpha complexes.

It's also hard to work with the persistence diagrams directly because they are not defined in a metric space. So, there are numerous topological descriptors that can be used instead and that preserves the information in the persistence diagrams.

3. Persistence landscapes.

(You can go back to the first part of our project for more details about the previous three elements)

4. Persistence Images : a stable finite-dimensional vector representation of persistence diagrams (turning a diagram into a surface.
5. Betti curves is a function mapping a persistence diagram to an integer-valued curve, i.e., each Betti curve is a function $\mathbb{R} \rightarrow \mathbb{N}$. For example:



PersLay Layer:

PersLay is a neural network layer designed to handle persistence diagrams. The idea behind it is to transform persistence diagrams into vectors in a differentiable way (which allows the use of backpropagation). The first requirement is that persistence diagrams should be permutation invariant. Thus, two sets of points with different order must give the same results.

The PersLay layer achieves this by transforming each point in the persistence diagram into a vector, and then summing (or performing other operations such as maximum, average...) all these vectors to obtain a single vector. For more details on this layer, we refer to the following article : *PersLay: A Neural Network Layer for Persistence Diagrams and New Graph Topological Signatures.*

Application:

Data:

The data set:

The dataset we use is the NIST 8-Bit Gray Scale Images of Fingerprint Image Groups (FIGS) from academictorrents. The NIST Special Database 4 contains 8-bit gray scale images of randomly selected fingerprints. The database was distributed for use in developing and testing automatic fingerprint classification systems on a common set of images. The original zip file contained 4000 fingerprints stored in PNG format (2000 pairs). Each fingerprint is 512 X 512 pixels with 32 rows of white space at the bottom of the fingerprint.

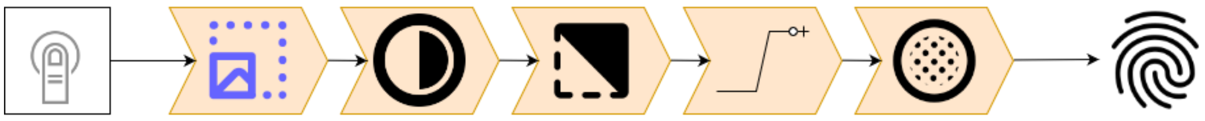
Why this data set?

Our goal was to apply TDA methods on new datasets, without any article or tutorial dealing with this dataset. So, we searched a large number of datasets on https://www.kaggle.com/datasets and https://datasetsearch.research.google.com/ and randomly chose fingerprints *(well, not quite randomly)*. The motivation is that fingerprints are composed of connected lines, so we assumed that we might be able to get some structures or features that would be extracted by persistence diagrams. After preprocessing the data, we searched for papers that would have already worked on fingerprints and found that SVM is generally the most used method (with different similarity distances), along with CNN used for gender classification and other more algorithmic methods that detect loops and singularity points.

But the only paper that interested us was Graph Representation of Fingerprint Topology, the one that shows that a graphical representation is possible for fingerprints, which reassured us to continue.

Data preprocessing:

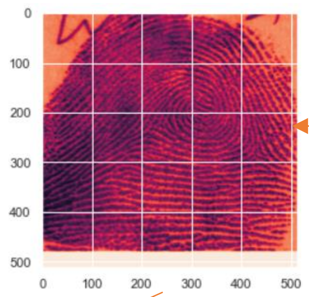
The pre-processing process of our data is described by the following flowchart:



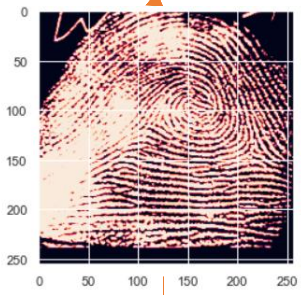
1. Resizing the original image to 256*256 pixels.
2. Enhancing the contrast.
3. Reversing the image (applying the not operator on all the bits of the image)
4. Scaling the pixels (so each cell of the array has a value between 0 and 1.
5. Noise suppression, by giving cells with a value smaller than 10^{-3} the value zero.

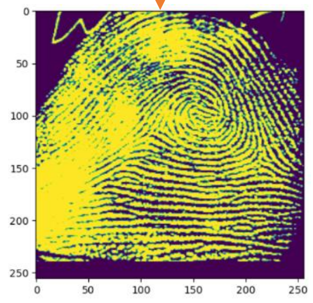
Why did we process the images this way? It's because if we took a look at the original images, we can see that the array is too noisy, all the cells have values in the same range. And we don't have a clear distinction between the lines, the cells with zeros and the cells with ones. And remember that to get the complexes from the images we want them to look like cloud points, each image has distinguished points and lines. Because that's what would (ideally) enable us to extract topological features.

Here is an example. The original image:

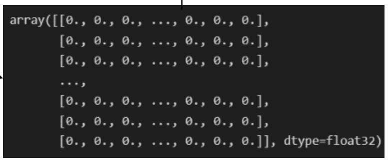


…	…	…	…	…
0.76862746	0.7607843	…	0.7647059	0.77254903
0.76862746	0.7607843	..	0.76862746	0.7607843
0.7921569	0.7529412	…	0.7490196	0.7529412
…	…	…	…	…





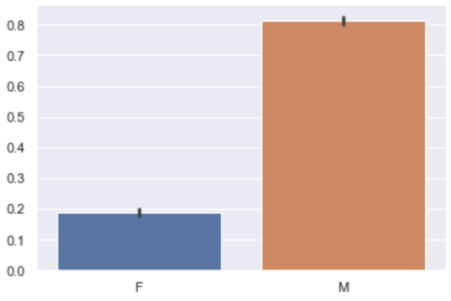
0.	0.	0.	0.	0.	0.	0.	…	0.	0.	0.	0.	0.	0.	0.
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
0.	0.40	1.	1.	1.	0.05	0.	…	0.67	0.99	1.	1.	0.40	0.	0.
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
0.	0.	0.	0.	0.	0.	0.	…	0.	0.	0.	0.	0.	0.	0.



```
array([[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.]], dtype=float32)
```

The problem and the topological descriptors:

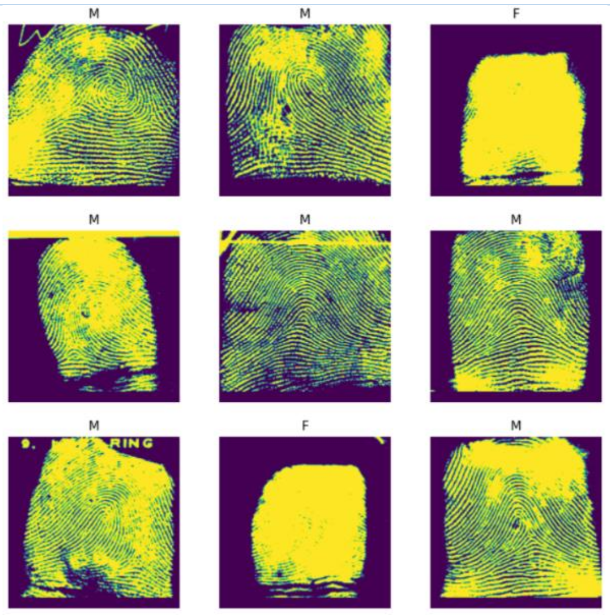
With this dataset we have two classification problems: binary classification and five-class classification. Fingerprints are classified into one of five classes (L = left loop, W = swirl, R = right loop, T = tented bow, and A = arc) with an equal number of prints from each class (400). And two gender classes M: Male and F: Female. We started with binary classification but we found that the data is not balanced, so we use the SMOTE method to balance the data (only for binary classification).



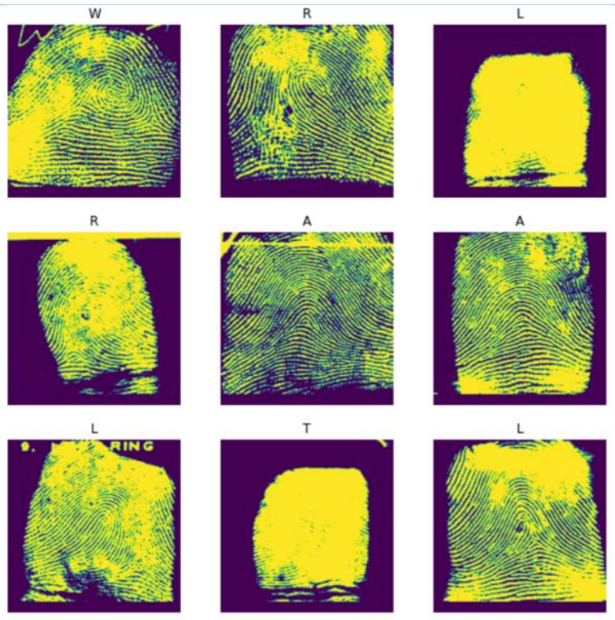
SMOTE (Synthetic Minority Oversampling Technique) is a technique to overcoming Class Imbalance, where the synthetic samples are generated for the minority class. And since we don't want to give the model repeated data (for example the same image twice) we rotate each image with a random angle θ, where $\theta \sim \mathcal{N}(0,3)$. We get 6500 images as a result.

The topological features we can extract from the images, thanks to persistence homology, are: persistence diagrams of Alpha complex and rips complex (persistence barcodes although we won't use them) and persistence images (from the persistence diagrams). We will see later that by using the PersLay model (layer) we can use other features (without creating them directly, for example Betticurves ...).

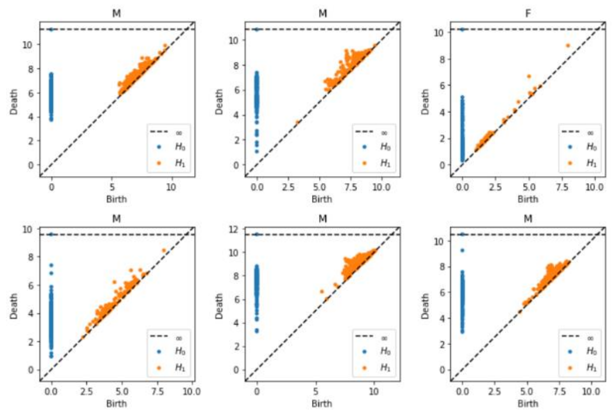
Here is an example of the final data, and topological features:



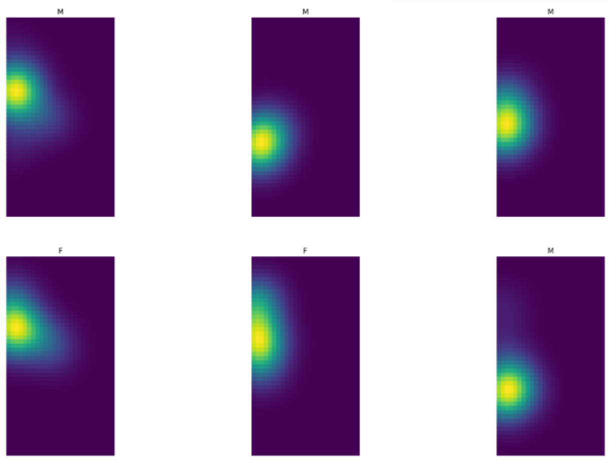
Final images with their labels for binary classification



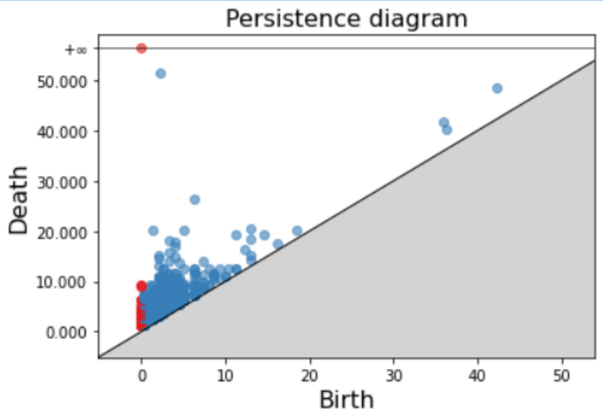
Final images with their labels 5-classes

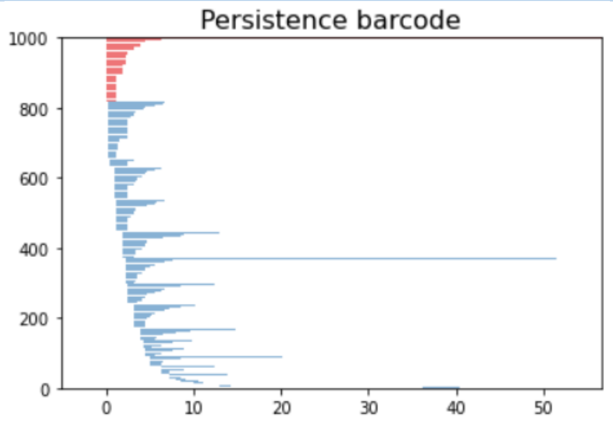


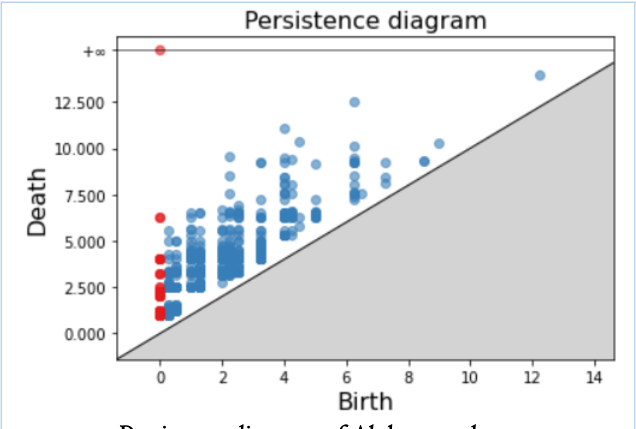
Persistence diagrams of Rips complexes



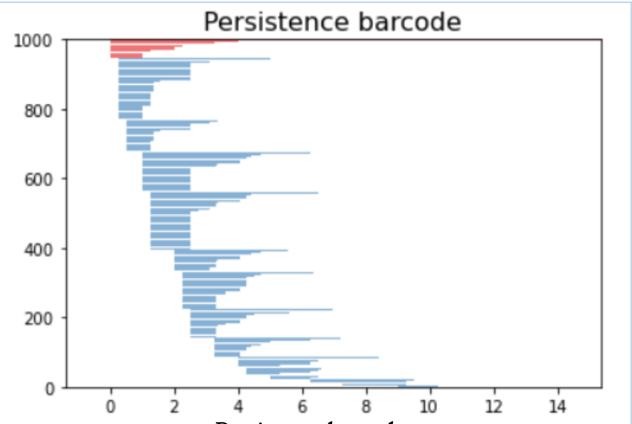
Persistence images







Persistence diagrams of Alpha complexes

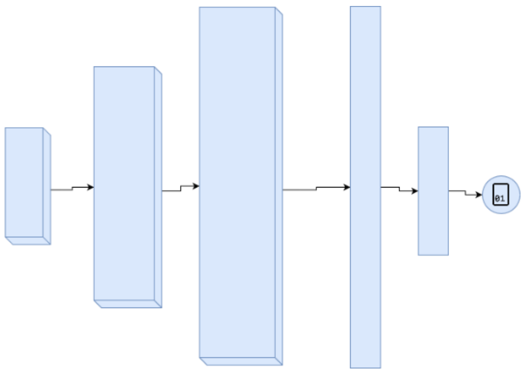


Persistence barcodes

Deep Learning and Topological Data analysis:

To establish a benchmark, we use a classical convolutional neural network for both classification problems. We tried many structures (and different hyperparameters...).

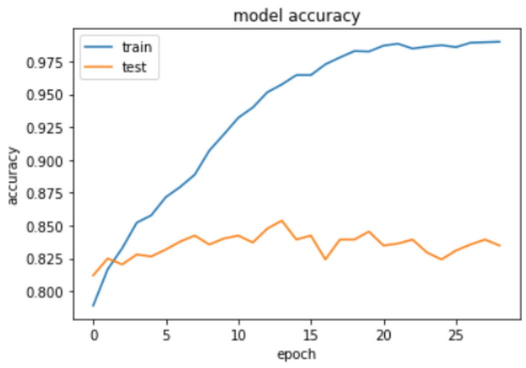
We tried a convergent CNN and a divergent CNN (by divergent and convergent we mean if the convolutional layers are getting bigger or smaller) [Conv2D(32, (2, 2)) -> Conv2D(64, (2, 2)) ->Conv2D(128, (2, 2)): divergent] and [Conv2D(128, (2, 2)) -> Conv2D(64, (2, 2)) ->Conv2D(32, (2, 2))): convergent] . The divergent method gave slightly better results (although the validation accuracy was very close), probably because most of the useful features of an image are usually local and not global. For the final model we use as a reference, the best activation function is "relu" or "LeakyRelu", and adding additional layers simply leads to faster overfitting without improving the validation accuracy. We also used early stopping to avoid overfitting. (the full graph can be found in the appendix 2).



Heuristic diagram of the CNN

The results:

We get a validation accuracy of 83.99%



Now, let’s use topological data analysis. We used persistence images (as vectors of size 84) with Logistic regression and we get an accuracy of 80.5%.

And this leads us to our first important result:

Although the neural network performs better, it should be noted that we only use vectors of size 84 + simple logistic regression, whereas the neural network, well it's a neural network, uses 256*256 images, so the persistence images have a huge time and memory advantage.

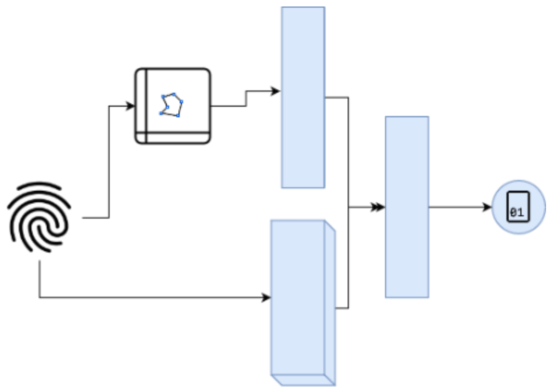
The model	Logistic regression	CNN
The input	Persistence images (84,)	Preprocessed images (256,256)
Validation accuracy	80.5%	83.99%
Strengths and weaknesses	- No overfitting - Time and memory advantage - Simple model	- Better accuracy and can be even better with a bigger dataset - But it might lead to overfitting - It takes a lot of time to train the model and optimize the hyperparameters

Now let's continue with the binary classification but after applying the SMOTE.

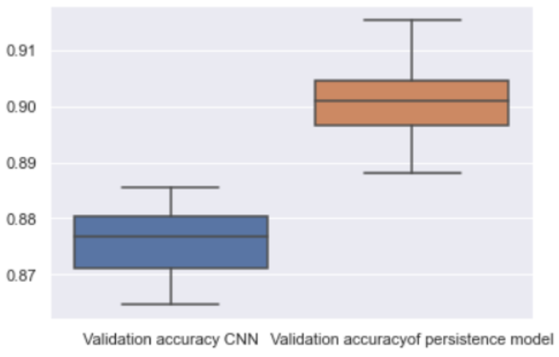
The model	Logistic regression	CNN
Validation accuracy	80.69%	88.31%

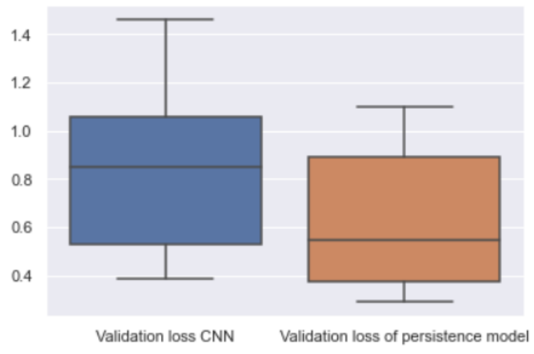
Here the CNN is far better than logistic regression but we can still use persistence homology. For feature augmentation.

The idea here is to create a model with a two-input structure as you can see in the graph below (the full graph can be found in the appendix 3). And this improves the accuracy of the validation and makes the validation loss smaller. Also, we will see that this result is true for both problems (2-class and 5-class classification).



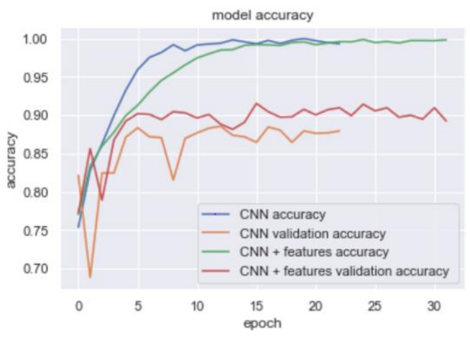
The results:

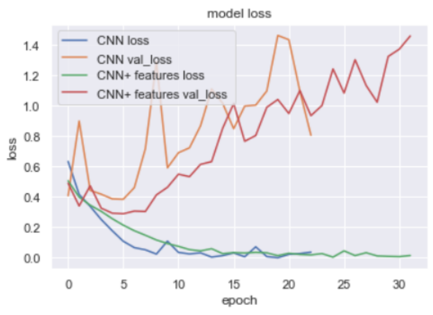




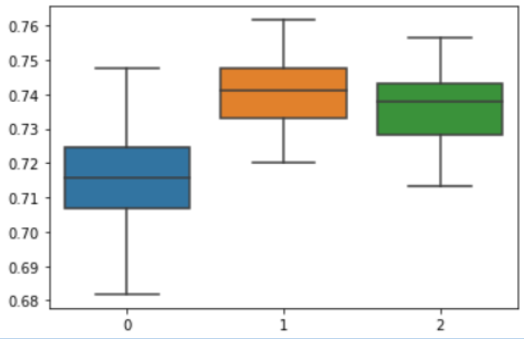
We compare then the results of the normal CNN and the CNN with persistence images as additional features. Here are the graphs of the loss and the accuracy.

The model with topological features performs clearly better than the classical CNN model, as we can see in the box plots, both in terms of accuracy (87.6% vs 90.2%) and loss.

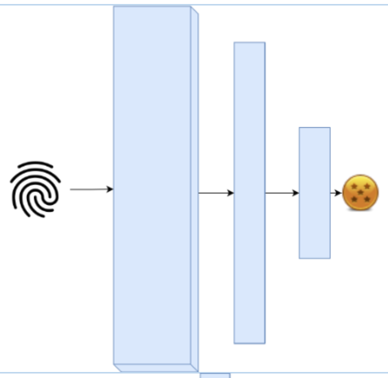
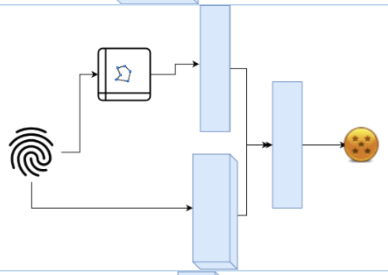
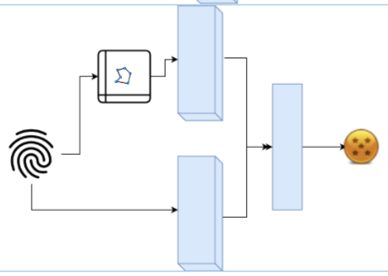




Now let's see with the five-class problem, again the model will have a similar structure but instead of using the persistence images as flat vectors we also tried to use them as images with a convolutional layer.



Label (in the boxplot)	Model	Highest validation accuracy	Heuristic diagram

0	Classic CNN	74.77	
I	CNN+PI with a dense layer	76.20	
2	CNN+PI with a convolutional layer	75.65	

The difference isn't as important here, but still there is a difference. Which means that topological features allow the network to extract information that could not be extracted without them.

Note:

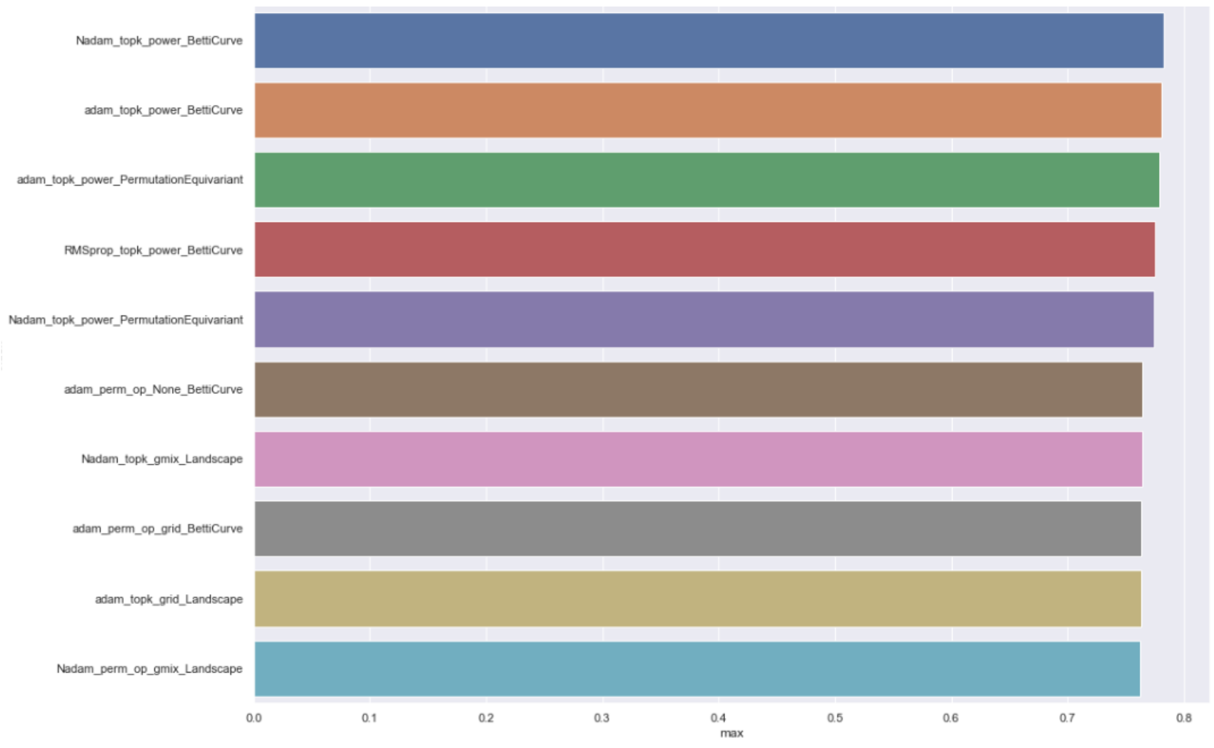
We also tried other methods like boosted trees (XGBoost), we also tried to use diagrams directly with neural networks but none of them gave an interesting result so we didn't include them in the final results.

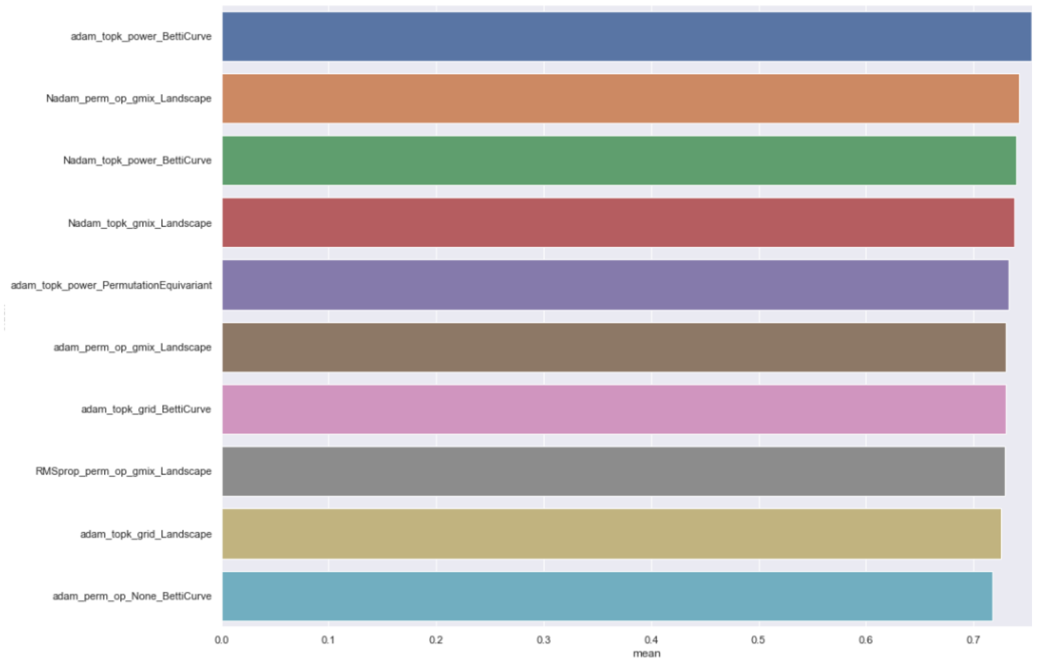
Third result:

The third part is by using PersLay model. For this we created a Rips diagram, and an Alpha diagram for each image. We tried to compare making rips and alpha diagrams by ***gudhi***, and rips diagrams by ***ripser***. Creating rips diagrams with ***gudhi*** takes a lot of time, so we used ***gudhi*** Alpha diagrams and ***ripser*** rips diagrams (in the end the diagrams will be arrays recognizable by the PersLay model).

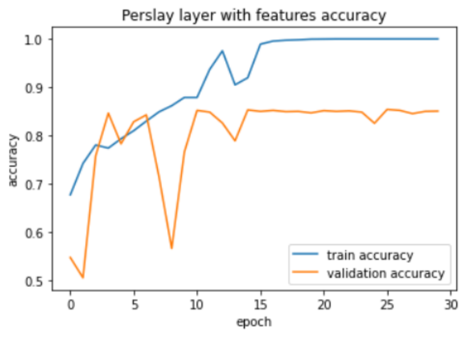
Creating these diagrams takes a lot of time (around 150 min), so we create them and store them to be used for future tests. For storage For storage, we tried three methods: ***pickle*** and ***np.load*** and ***h5py***, the first two give files of about 6 gigabytes, so it is not practical. The last one has a format problem. The best solution is ***hickle*** (combination of pickle, and h5py which avoids both format and memory problems, only 139 megabytes).

There is a large number of parameters to choose from [optimizer, layer type, permutation-invariant operation, Weight function]. So, we do a quick and easy search on the parameters, and then select the possible candidates to work on. This is the comparison of the mean and the max of validation accuracy:



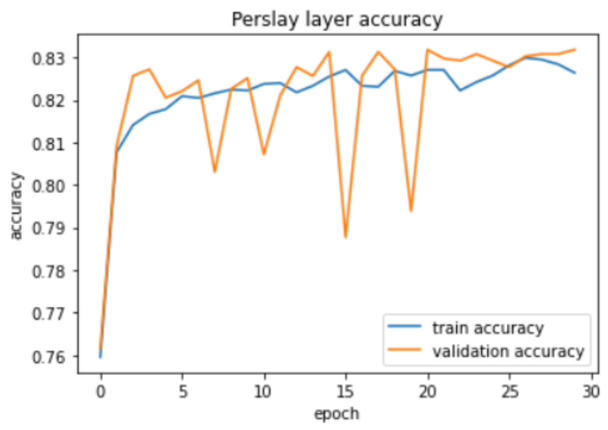


By using only, the diagrams (with a Betti curve) we get a validation accuracy of 83.17% for the binary classification problem.



While it is not better than our best model so far, the Perslay is using only 849 trainable parameters, while the CNN+ Persistence images model was using more than 1 million trainable parameters. This shows the potential of the Perslay model.

Finally, if we add features (just flat images) we get 85.38 % as a best validation accuracy. And this was without hyperparameter optimization, it was only the first test so it can be improved in the future.



Conclusion:

In conclusion, TDA tools can be used in many useful ways:

- Direct use, as a vectorization method as we have seen with logistic regression and persistence images.
- It can be used as a feature extraction or feature augmentation method.
- And finally, the persistence layer which has a lot of potential and can be used to explore other topological descriptors.

Note:

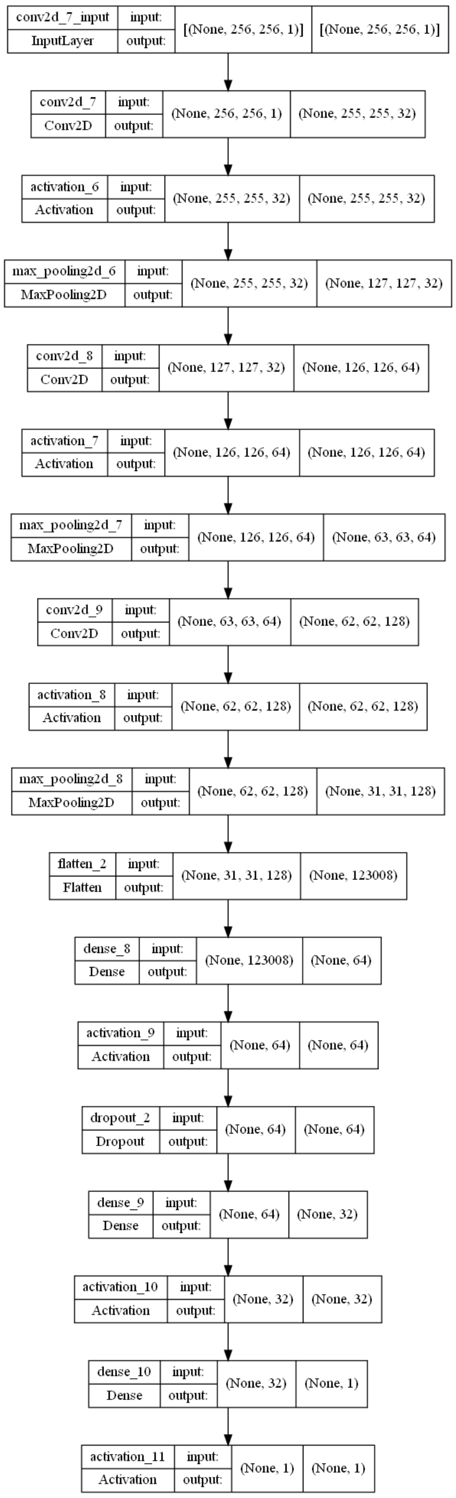
While testing the different possible uses of the perslay layer, we have got an error, related to one of the possible layer parameters. The issue is submitted here .

Appendix 1: Github

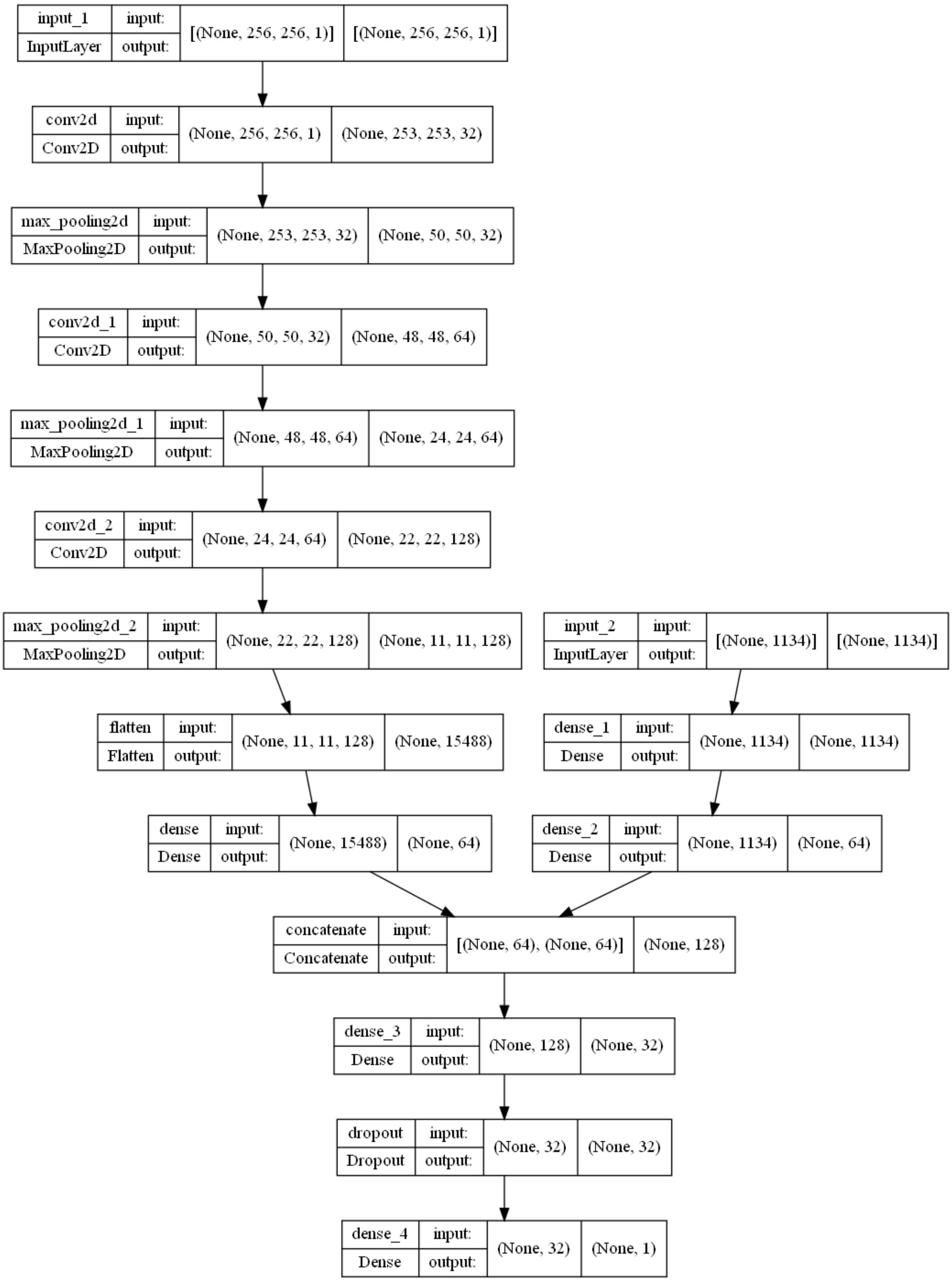
The link to the github repository: https://github.com/RandomAnass/TDA-DL

The project is reproducible, you will only need to install the required packages (for the part 1 we recommend using the Google Colab Notebook).

Appendix 2: Graph-model-1


conv2d_7_input | InputLayer | input: [(None, 256, 256, 1)] | output: [(None, 256, 256, 1)]
conv2d_7 | Conv2D | input: (None, 256, 256, 1) | output: (None, 255, 255, 32)
activation_6 | Activation | input: (None, 255, 255, 32) | output: (None, 255, 255, 32)
max_pooling2d_6 | MaxPooling2D | input: (None, 255, 255, 32) | output: (None, 127, 127, 32)
conv2d_8 | Conv2D | input: (None, 127, 127, 32) | output: (None, 126, 126, 64)
activation_7 | Activation | input: (None, 126, 126, 64) | output: (None, 126, 126, 64)
max_pooling2d_7 | MaxPooling2D | input: (None, 126, 126, 64) | output: (None, 63, 63, 64)
conv2d_9 | Conv2D | input: (None, 63, 63, 64) | output: (None, 62, 62, 128)
activation_8 | Activation | input: (None, 62, 62, 128) | output: (None, 62, 62, 128)
max_pooling2d_8 | MaxPooling2D | input: (None, 62, 62, 128) | output: (None, 31, 31, 128)
flatten_2 | Flatten | input: (None, 31, 31, 128) | output: (None, 123008)
dense_8 | Dense | input: (None, 123008) | output: (None, 64)
activation_9 | Activation | input: (None, 64) | output: (None, 64)
dropout_2 | Dropout | input: (None, 64) | output: (None, 64)
dense_9 | Dense | input: (None, 64) | output: (None, 32)
activation_10 | Activation | input: (None, 32) | output: (None, 32)
dense_10 | Dense | input: (None, 32) | output: (None, 1)
activation_11 | Activation | input: (None, 1) | output: (None, 1)

Appendix 3: Graph-model-2


input_1 | InputLayer | input: | output: | [(None, 256, 256, 1)] | [(None, 256, 256, 1)]
conv2d | Conv2D | input: | output: | (None, 256, 256, 1) | (None, 253, 253, 32)
max_pooling2d | MaxPooling2D | input: | output: | (None, 253, 253, 32) | (None, 50, 50, 32)
conv2d_1 | Conv2D | input: | output: | (None, 50, 50, 32) | (None, 48, 48, 64)
max_pooling2d_1 | MaxPooling2D | input: | output: | (None, 48, 48, 64) | (None, 24, 24, 64)
conv2d_2 | Conv2D | input: | output: | (None, 24, 24, 64) | (None, 22, 22, 128)
max_pooling2d_2 | MaxPooling2D | input: | output: | (None, 22, 22, 128) | (None, 11, 11, 128)
input_2 | InputLayer | input: | output: | [(None, 1134)] | [(None, 1134)]
flatten | Flatten | input: | output: | (None, 11, 11, 128) | (None, 15488)
dense_1 | Dense | input: | output: | (None, 1134) | (None, 1134)
dense | Dense | input: | output: | (None, 15488) | (None, 64)
dense_2 | Dense | input: | output: | (None, 1134) | (None, 64)
concatenate | Concatenate | input: | output: | [(None, 64), (None, 64)] | (None, 128)
dense_3 | Dense | input: | output: | (None, 128) | (None, 32)
dropout | Dropout | input: | output: | (None, 32) | (None, 32)
dense_4 | Dense | input: | output: | (None, 32) | (None, 1)