CycleCNN- Final Project Report for Deep Learning Class

Maestrati Louis
ENS Paris Saclay
4 Avenue des Sciences, 91190 Gif-sur-Yvette
maestratilouis@gmail.com

Abstract

Convolutional neural networks (CNNs) are known to produce the state of the art in image recognition. Their behavior during training and the theoretical reasons behind their success remain partially understood matters. In order to deepen these issues, it is natural to study the geometry and topology of images and kernel spaces, which are the neurons of CNNs. To this extent, the field of Topological Data Analysis (TDA) turns out to be relevant, since it allows precisely to perform calculations to characterize and distinguish shapes and spaces, using the concepts of Algebraic Topology. Indeed, recent research at the interplay between TDA and Deep Learning provide evidence that spatial filters of CNNs' kernels tend to form circular shapes during training. Our work exploits such empirical observations by building a new way of parameterizing CNN kernels. The resulting new architecture, CycleCNN, forces spatial filters to remain on topological cycles during training. This results in a drastic reduction of parameters, which strongly regularises the model.

1. Introduction and Motivation

As is well-known, Convolutional Neural Networks (CNN) achieve state-of-the-art performance in image classification. The building blocks of this network are the *kernels*; which are the neurons of CNNs; and that are modified during training to achieve image classification.

The optimisation of kernels, e.g. using *gradient descent*, is a well-known subject of investigation, for which we have (partial) convergence results. However, the properties of the equilibrium state, i.e. of the kernels at convergence, remain unclear. Such properties are desirable, because systematic structure can be used to derive new architectures and models with stronger guarantees.

The study of the equilibrium state is an emergent field of Deep Learning. Indeed, many researchers seek to explore the space of kernels and try to get an overall understanding of its structure. One popular approach is to analyse and interpret kernels in a relevant low-dimensional space, as in low ranked kernels [Tai et al., 2015] or sparse CNNs [Gale et al., 2019], which helps in analyzing local and global structure of kernels. Such optimal structures can then be exploited and pre-encoded in the architecture, see for instance the Wavelet Transform CNNs [Mallat, 1999].

The recent emergent field of Topological Data Analysis (TDA) aims to uncover hidden structures and shapes in datasets of high dimensionality. Indeed, TDA allows precisely to perform calculations to characterize and distinguish shapes and spaces, using the concepts of Algebraic Topology. These are the calculations of *homology groups*, which intuitively correspond to the numbers of connected components, circles and spheres of any dimension that can be inscribed in a space.

It thus seems natural to apply the recent progress and tools offered by TDA to study the geometry and topology of image and kernel spaces.

The authors of *Topological Approaches to Deep Learning* [Gunnar Carlsson, 2018] adopt such an approach to analyze deep neural networks. They focus on studying the topology of the space of elementary patches of the kernels, called *spatial filters*, obtained after training a convolutional network. Notably, viewing the set of all spatial filters as a point cloud in a fixed Euclidean space, the resulting shape seems to evolove towards a circular configuration during training.

In this work, we take advantage of the spatial filters' tendency to form circles during training. Namely, we force the spatial filters and kernels to evolve from scratch on circular shapes, and we directly learn the way these circles are embedded in the space of kernels. Below is the summary of our contribution:

- We build a new architecture called CycleCNN, which instead of parameterizing each kernel individually, parameterizes directly the embedding of the topological shape on which they live;

- In this way, we considerably reduce the number of parameters/weights needed for training. The intuitive reason for this goes as follows. The only degree of freedom needed to characterise the embedding of a circle are its orientation, center and radius. In a traditional convolutional architecture, the dimension of the space of parameters grows linearly with the input space, output space and kernel size; by contrast our model imposes strong regularization on the kernels space dimension. Our model can therefore also be seen as a method of kernels regularization;

- We demonstrate the ability of our model on various experiments. On the MNIST dataset, we show that our models are sufficiently expressive. However, on more complex classification tasks, e.g. CIFAR, the drastic reduction of degrees of freedom makes it hard for our model to compete with classic parameterizations. To tackle this, we device hybrid models, i.e. we show how to couple the layers of CycleCNN with standard layers;
- For hybrid models, it is crucial that the parameters of the CycleCNN layers and those of the standard layer have similar magnitude and pace of evolution. For this, we derive consistent choices of initialisation and learning rates for the CycleCNN parameters.

We note that our model may be viewed as an alternative approach for dimensionality reduction of kernels. Besides, although we mainly focus on embeddings of the circle in this paper, it is apparent that the method can be used with any topological shape, e.g. the Klein bottle observed by [Gunnar Carlsson, 2018]. Finally, we believe that the inherent need of less computational memory of our model is a great potential for industrial applications.

2. Problem Definition

As is usual in Deep Learning litterature, a *spatial filter* is a vector in $\mathbb{R}^{3\times 3}$, whereas a *kernel* is a vector $\mathbb{R}^{3\times 3\times \#\text{InChannels}}$, where $\#\text{InChannels}$ is the number of input channels in the layer of interest.[1] Intuitively, a spatial filter is thought of as an elementary patch that is convoluted with an image's channel. The kernel is then obtained by aggregating spatial filters accross channels.

2.1. A brief Introduction to Topological Cycles

A substantial prerequisite for this work is to understand the analysis of [Gunnar Carlsson, 2018]. This requires being used with concepts borrowed from Algebraic Topology and their computational counterparts in data analysis. In this section, we provide a high-level overview of the key notions, but refer the reader to [Hatcher, 2005] and [Oudot, 2015] for full introductions to the main concepts in Algebraic Topology and TDA respectively.

[1]Note that working in $\mathbb{R}^{3\times 3}$ is not restrictive. Our analysis is the same in $\mathbb{R}^{k\times k}$ for any k.

Homology groups One of the main goal in Algebraic Topology is to distinguish between topological spaces, that is shapes of any kind. The notion that captures the idea of two spaces being the same is that of *homotopy*, which is a weaker form of homeomorphism. Roughly, two spaces X and Y are homotopic if there are maps f, g going back and forth between X and Y such that the composition $f \circ g$ is a continuous deformation of the identity map, see Fig. 1 for an example.[2]

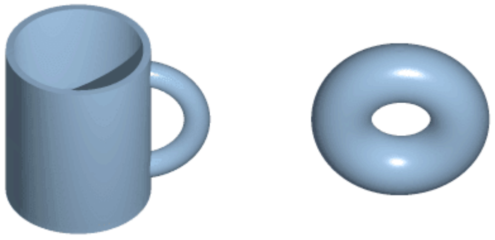

Figure 1. The cup of coffee and the plain torus are homotopic as one can continuously deform one into the other, see link

It is intuitive that in practice, deciding whether X and Y are homotopic is very challenging. The *homology groups* $\mathrm{H}_p(X)$ of the space X are algebraic and computable invariants that precisely help resolving this issue. Namely, given an integer p, the vector space $\mathrm{H}_p(X)$ has a dimension $\beta_p(X)$ -the so-called p-th *betti number* of X- that counts the number of independant p-spheres inscribed in X. So $\beta_0(X)$ counts the number of (path) connected components and $\beta_1(X)$ that of circles, as exemplified in Fig 2. Being an invariant means that if two spaces are homotopic, then they yield the same homology groups. So by contraposition, one can tell X and Y apart whenever $\beta_p(X) \neq \beta_p(Y)$. That $\mathrm{H}_p(X)$ is algebraic means that it can be computed by means of elementary tools of Commutative Algebra, which means that it is computationally tractable. The only requirement is that X must be equipped with the extra structure of a triangulation. However, most spaces encountered in practice, e.g. graphs and smooth manifolds, meet this requirement.

Persistence barcodes In Machine Learning and Data analysis, the space/shape at stake is the so-called *dataset*. In most scenarios, a dataset can simply be viewed as a point cloud in Euclidean space. The inherent discrete nature of point clouds makes the homological calculations of the previous paragraph obsolete, as the betti numbers would merely count the number of points.

In *Persistence theory*, one method to address this issue is to grow balls around each point of the dataset as in Fig. 3,

[2]Note that for the spaces X and Y to be homeomorphic one would actually require that the composition $f \circ g$ is the identity map on the nose.

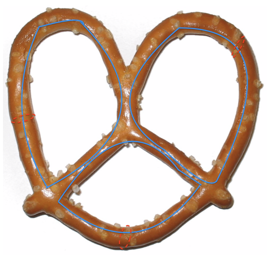

Figure 2. This bretzel has one connected component and three distinct independent circles. Hence the betti numbers $\beta_0 = 1, \beta_1 = 3$ and $\beta_p = 0$ for $p > 1$.

and compute the betti numbers of the union of balls at different scales. The resulting descriptor is the *barcode/ persistence diagram* of the dataset, and is a set of intervals. Each interval $[r, R]$ is the range of radii for which a specific connected component, loop, sphere or higher dimensional hole, persists: the hole appears at radius r, and persists until it is filled at radius R. In Fig 4, we computed the barcode of two intersecting circles using the Gudhi software [Maria et al., 2014].

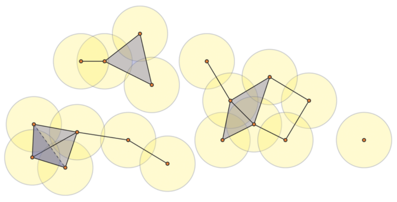

Figure 3. Union of balls with a fixed radius around each point of a point cloud in the plane. We draw an edge (resp. triangle and tetrahedron) between 2 (resp. 3 and 4) points whenever the corresponding balls intersect. The resulting combinatorial structure is the so-called *Cech Complex*, for which it is easy to compute betti numbers.

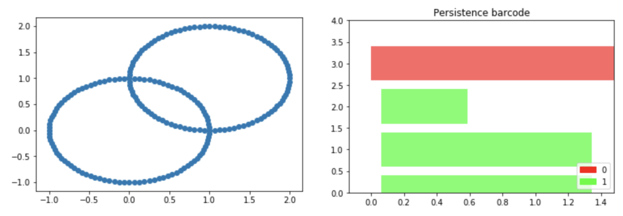

Figure 4. The point cloud aranged around two intersecting circles (left) yields the barcode with 1 interval in degree 0 and 3 intervals in degree 1 (right). The smaller green interval corresponds to the small loop formed by the intersection of the circles.

2.2. Topology of the space of spatial filters

Barcodes can be computed for small patches of natural images, as done in [A.B. Lee and Mumford, 2003, G. Carlsson and Zomorodian, 2008]. It is then observed that the barcodes have prominent intervals in homology degree 1, from which it is deduced that the shape formed by the patches tends to be circular. In [Gunnar Carlsson, 2018], the authors rather investigate the topological shape formed by the spatial filters of size 3×3 throughout training of a CNN.

To obtain this shape, the spatial filters are first centered, reduced and thresholded by variance. Then, they are filtered by codensity, the codensity being the average distance of a point to its k nearest neighbors. They thus only keep a percentage ρ of the best spatial filters (those with the lowest codensity). This was done in order to avoid considering insignificant spatial filters -one can see as *outliers*- and which would confuse the overall topological appearance of the point cloud. The obtained spatial filters were plotted in 2D *Mapper*, another standard tool of TDA for vizualizing high dimensional data in a topologically meaningfull way [Singh et al., 2007]. See Fig 5 for the Mapper representation and barcode of the spatial filters of a 1 layer network, after training on the MNIST dataset, and similarly Fig. 6 for the spatial filters of various layer in a deeper network.

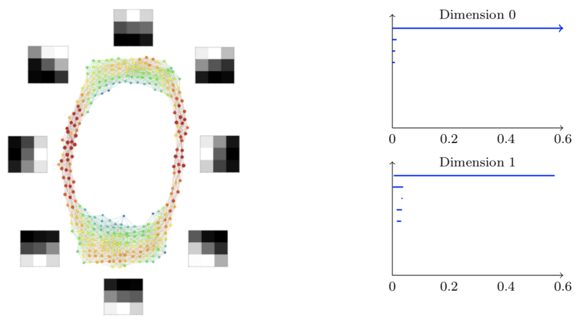

Figure 5.

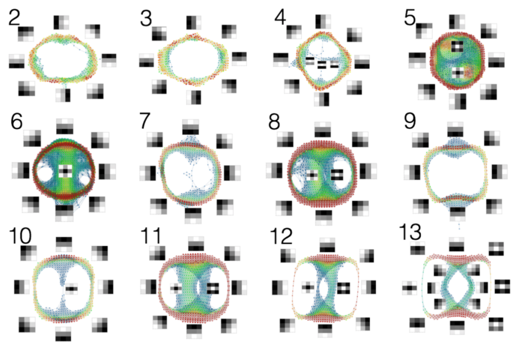

Figure 6. Mapper Model, with $k = 15$, $\rho = 30\%$ applied to VGG-16 layers trained on Imagenet. Figure from [Gunnar Carlsson, 2018].

One can clearly see that each layer tends to form 1D topological cycles (1 for layers 2 and 3; 2 for the others).

This demonstrates the tendancy of spatial filters to systematically recover circular configurations throughout training. Besides, experiments of Data Augmentation of the input images using spatial filters found in a primary circles resulted in higher accuracies of the network and further generalization abilities [Gunnar Carlsson, 2019]. This is an example of how the circular structure can be used to improve existing models.

3. Related Work

As quoted in the introduction, our work investigates the subject of kernels regularization. The most drastic regularization approach consists in completely removing the set of weights of a network. This is the approach taken in networks based on Wavelet Transforms [Mallat, 1999]. In their simplest form, such networks replace kernels with meaningful, fixed wavelets, and the training merely reduces to finding the right combination of the wavelets' outputs. This results in highly interpretable and computationally very attractive models. In spirit, our approach is similar in that it exploits the systematic structure found in complicated networks after training in order to simplify the architecture/parameterization of the model.

Another approach to kernels' regularization is sparsification. Recently, [Frankle and Carbin, 2018] formulated the *Lottery Ticket Hypothesis* which stipulates that all networks "hide" a sub-network of equivalent performance, meaning that an important amount of neurons can be prunned without damaging the network performances. A popular example of sparsification technic is Magnitude Prunning, which consists in prunning the weights of least amplitude of the network [Gale et al., 2019] until we reach the desired sparsity. This is in general done progressively during training, to ensure the network stability.

Finally, we believe that Low Rank regularization is the closest type of regularization to our model. It consists in constraining the kernels of CNNs to live in smaller dimension spaces by encoding a maximum dimensionality of the kernels in the architecture. This approach is thus by essence very related to our work, except that our model not only enforces lower dimensionality of the space of kernels, it also enforces the kernels, by architecture, to be on a fixed topological shape. As an example, [Tai et al., 2015] constrained the rank of the spatial filters living in $\mathbb{R}^{k\times k}$ by replacing them by a sum of product of "vertical" spatial filters in $\mathbb{R}^{k\times 1}$ with "horizontal" spatial filters in $\mathbb{R}^{1\times k}$. We compared the results of this approach with our method in section 5.

4. Methodology

In the following section, we describe the CycleCNN model.

4.1. CycleCNN model

Formally, we call *topological cycle* in an Euclidean space $\mathbb{R}^n$ any loop without self-intersection. Inside the huge set of topological cycles, we have the set of *ellipsoids*, which we abusively also call *circles* in the following, defined geometrically from their center, radius, supporting 2D plane and eccentricity. Although being a quite stringent class of shapes, ellipsoids are easy to parameterize. Namely, any ellipsoid can be obtained by applying the right affine transformation $T : \mathbb{R}^2 \to \mathbb{R}^n$ to the standard, unit circle in $\mathbb{S}^1 \subseteq \mathbb{R}^2$. We call such a map T a *transformer* hereafter.

Recall from [Gunnar Carlsson, 2018] that spatial filters of size 3×3 tend to form a topological cycle in $\mathbb{R}^{3\times 3}$. Our model is based on the following slightly stronger hypothesis:

The kernels tend to form a circle in $\mathbb{R}^{3\times 3\times \mathrm{InChannels}}$.

Note that this statement is equivalent to asking that the spatial filters *of each channel* tend to form a circle. Indeed, an ellipsoid in $\mathbb{R}^{3\times 3\times \#\mathrm{InChannels}}$ determines $\#\mathrm{InChannels}$ ellispoids in $\mathbb{R}^{3\times 3}$ and conversely.

To parameterize a circle in kernel space, we consider a transformer

$$T : \mathbb{R}^2 \to \mathbb{R}^{3\times 3\times \#\mathrm{InChannels}}.$$

In practice, a transformer is implemented as a fully-connected layer (FC), which we apply to a regular discretization of the unit circle $\mathbb{S}^1$. Each point of $\mathbb{S}^1$, through the transformer, gives rise to a kernel.

We can in fact use several transformers $T_1, \cdots, T_m$ in order to produce m circles in kernel space. This improves the expressiveness of the model. One example of a model using a single layer of CycleCNNs with 4 circles of kernels is detailed in figure 7.

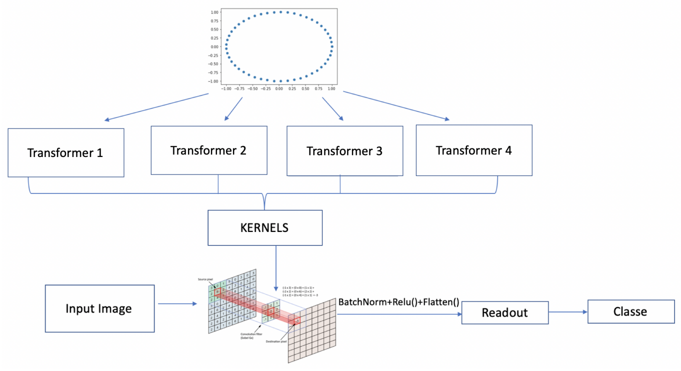

Figure 7. Diagram of a CycleCNN with one layer using 4 circles.

The parameters of the transformer become the only parameters of the entire layer, and are learnt using the backpropagation algorithm. The transformer is a FC layer with size $(2, 9 \times \#\text{InChannels})$, leading to a number of parameters of $18 \times \#\text{InChannels}$ encoding the entire layer of convolution. Notice that this number of parameters in the CycleCNN layer becomes independent of the number $\#\text{OutChannels}$ of kernels used for convolution, and one can compare this amount of parameters to $9 \times \#\text{OutChannels} \times \#\text{InChannels}$ used in a classic layer of convolution. The ratio of gain in parameters to encode a layer being $\frac{2}{\#\text{OutChannels}}$; a layer using 100 CycleCNN kernels will for example use 98% less parameters than 100 classicaly parameterized kernels.

4.2. Deep transformers

One strong assumption we have made is that the topological cycles that spatial filters tend to form are actually circles. However, the geometric rigidity of circles is a quite stringent requirement, and was not empirically observed in [Gunnar Carlsson, 2018].

In order to capture a wider class of topological cycles structures, we consider transformers that may deform the unit circle non-linearly. This leads us to add activation functions in our transformers and increase its depth. In order for the resulting transformer T to produce an actual topological cycle, it is sufficient to ensure that T is injective, so that $T : \mathbb{S}^1 \to \mathbb{R}^{3\times3\times\#\text{InChannels}}$ is indeed a loop without self-intersection.

To obtain an injective tranformer, we construct multi-layer FC network as follows. Recall that in any matrix space, the subspace of full-rank matrices is generic. Therefore, it is natural to consider a composition of layers in increasing order of dimension, that is of the form:

$$T : \mathbb{S}^1 \subseteq \mathbb{R}^2 \to \mathbb{R}^{p_1} \to \cdots \to \mathbb{R}^{p_k} = \mathbb{R}^{3\times3\times\#\text{InChannels}},$$

where $2 \leqslant p_1 \leqslant \cdots \leqslant p_k = 3 \times 3 \times \#\text{InChannels}$. In-between each layer, we add a LeakyRelu activation, which is bijective over the real line. This way, the resulting transformer is (generically) an injective map.

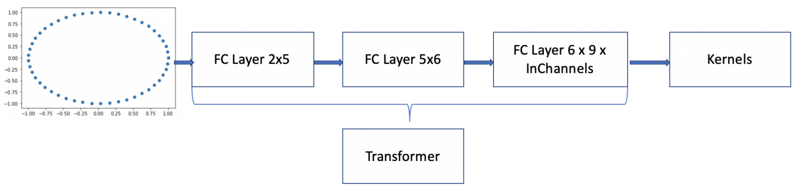

Figure 8. An instance of deep transformer used in CycleCNN.

4.3. Hyperparameters scheduling

Initialization of weights To get a benchmark comparision between classical CNN and our implementation, one challenge is to initialize the weights. We need to find an initialization of the FCs used in our transformers that comes up with comparable initial kernel values w.r.t the classical kernels parameterization. To this extent, we studied how initialization of kernels and Fully Connected Layers are generally done and came up with the following method of initialization:

Default initializations of weights in CNN's kernels and in fully connected layers are indeed different. The default initialization of CNN's kernels sample each individual weight (a real value) uniformly in $[\frac{1}{-\sqrt{\text{std}_{\text{CNN}}}}, \frac{1}{\sqrt{\text{std}_{\text{CNN}}}}]$, where

$$\text{std}_{\text{CNN}} := \#\text{InChannels} \times \#\text{WidthKernel} \times \#\text{HeightKernel}.$$

This is to ensure that a kernel has $\frac{1}{3}$ variance (squared norm). See this link for the details.

By contrast, as we sample our kernels with a Fully-Connected Layer applied to the discretized unit circle with $\text{std}_{\text{FC}} = \dim x$, the resulting initial squared norm of a kernel is far from the one corresponding to the classic kernels initialization.

In fully connected layers, each individual weight is sampled uniformly in $[\frac{1}{-\sqrt{\text{std}_{\text{FC}}}}, \frac{1}{\sqrt{\text{std}_{\text{FC}}}}]$, where

$$\text{std}_{\text{FC}} := \dim x, \ x \text{ being an input vector.}$$

This ensures that the output of each neuron (a real value of the form $\sum w_i x_i$) has the same variance, which is proportional to the squared norm of the input vector $x = (x_1, \cdots, x_{\dim x})$. Indeed, we have

$$\text{Var}(\sum w_i x_i) = \sum \text{Var}(w_i) x_i^2 = \frac{1}{3\text{std}_{\text{FC}}} \sum x_i^2 = \frac{\|x\|_2^2}{3\text{std}_{\text{FC}}}.$$

Let us see the impact of parameterizing kernels of a CNN layer via a fully connected layer applied on a fixed input x. For instance, in our model, the elements x are taken from the unit circle in $\mathbb{R}^2$, leading to $\|x\|_2^2 = 1$. For simplicity, let us first assume that we deal with one fully connnected layer, i.e. a matrix, in order to generate a kernel from x. We then have a matrix M of size $(\dim x, \#\text{WidthKernel} \times \#\text{HeightKernel} \times \#\text{InChannels})$. Each entry of the outputted kernel Mx has thus a variance (i.e. squared norm) $\frac{\|x\|_2^2}{3\text{std}_{\text{FC}}} = \frac{1}{3\text{std}_{\text{FC}}}$, hence as a whole, the kernel has squared norm

$$\#\text{WidthKernel} \times \#\text{HeightKernel} \times \#\text{InChanels} \times \frac{1}{3\#\text{std}_{\text{FC}}}.$$

In order to recover the $\frac{1}{3}$ variance of the kernels, we simply change the definition of std_{FC} to

$$\text{std}_{\text{FC}} := \#\text{InChannels} \times \#\text{WidthKernel} \times \#\text{HeightKernel}.$$

Learning Rate Schedule In all our experiments we used Stochastic Gradient Descent (SGD). This was done to stick with the optimiser used in [Gunnar Carlsson, 2018]. However, we noticed that giving to the Transformer a learning rate similar to the other layers of the model makes the overall model diverging. We can explain this behavior by the fact that the transformers encode a very high amount of information. Indeed, the transformer is responsible for embedding of all the kernels, so a small update of the transformer's weights result in important variations of the set of kernels. In practice, this resulted in ellipsoid with diverging radius.

To solve this issue, we provide our model with two different initial values of learning rates: $\mathrm{lr}_{\mathrm{transf}}$ and lr used respectively for the tranformers and the rest of the architecture. We tried various hyperparameters for SGD and noticed that a factor of decrease $\gamma = 0.9$ and a ratio $\frac{\mathrm{lr}_{\mathrm{transf}}}{\mathrm{lr}} = 0.1$ achieve optimal performances.

5. Evaluation

Our experiments have been conducted on MNIST and on CIFAR-10 as a more challenging task for our model. Those are the datasets used in [Gunnar Carlsson, 2018] which is the article that motivated our work. All our models have been trained with GoogleColab on 12GB NVIDIA Tesla K80 GPU.

5.1. Baseline Evaluations

5.1.1 Experiments on MNIST

We first looked at the performances of our model on the MNIST dataset as a sanitary check. We applied a one-layer-CycleCNN to the images, followed by a 'readout' (Feed-Forward Layer) to obtain logits for classification. While keeping the number of kernels in the layer constant (100 kernels), we changed the number of circles used for parameterization.

This results in 3 different models, using:

- a CycleCNN layer parameterized with 1 circle of 100 kernels
- a CycleCNN layer parameterized with 5 circles of 20 kernels
- a CycleCNN layer parameterized with 10 circles of 10 kernels

These models performances has been compared to a classic CNN layer, also using 100 kernels.

We can see in figure 9 the accuracies and losses of our models. All models using our parameterization perform reasonably well. We observe that using only 5 circles lead to comparable accuracies on training and test set w.r.t. the baseline model. Our model using 5 circles achieves 100 % accuracy on the trainset and 97.5% on the testset, whereas the baseline achieves 100% / 98.2%. Their needs in parameters for the convolution layer are however drastically different; the baseline model using $100 \times 3 \times 3 \times 1 = 900$ parameters whereas our model uses only $5 \times 2 \times 3 \times 3 = 90$ parameters. Interestingly, reducing the number of circles to 1 worsen the performances (leading to 99.2% / 97.2% accuracies) and increasing the number of circles to 10 does not change the accuracies . The kernels in the layer thus seem to stabilise around 5 circles.

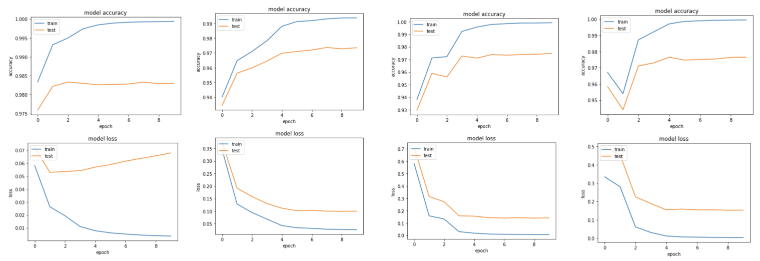

Figure 9. Results on MNIST:**1**: Single layer of a classic CNN with 100 kernels; **2**: CycleCNN layer, parameterized by 1 circle of 100 points; **3**: CycleCNN layer, parameterized by 5 circles of 20 points;**4**: CycleCNN layer, parameterized by 10 circles of 10 points

5.1.2 Experiments on CIFAR-10

We then looked at the performances of our model on a more challenging dataset: CIFAR-10. We trained the same 4 models, described in the MNIST experiments and the results can be seen in Figure 10. Different remarks can be done on the results:

- looking at the curves of the test-loss during training, it seems the models using CycleCNN are less susceptible to overfit. Indeed, the baseline model test-loss changes of convexity, while our models test-loss curves remain stable;
- as was the case on MNIST, a CycleCNN using only one circle seems too much constrained, and the model loses 20% of accuracy on trainset and 15% of accuracy on testset compared to the baseline;
- with a reasonable amount of circles (≥ 5), the CycleCNN reaches the trainset accuracy of the baseline (91%), and get close to the baseline testset accuracy (60% with 5 circles, 62% with 10 circles versus 67% for the baseline). Once again, the gain in parameter is important: the baseline model using $100 \times 3 \times 3 \times 3 = 2700$ parameters for convolution whereas $5 \times 2 \times 3 \times 3 \times 3 = 270$ parameters for the model using 5 circles.

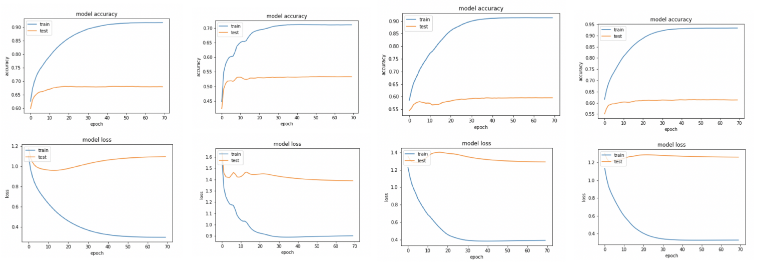

Figure 10. Results on CIFAR-10:**1**: Single layer of a classic CNN with 100 kernels; **2**: CycleCNN layer, parameterized by 1 circle of 100 points; **3**: CycleCNN layer, parameterized by 5 circles of 20 points;**4**: CycleCNN layer, parameterized by 10 circles of 10 points

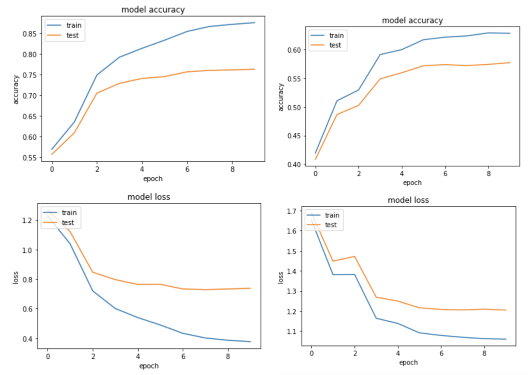

Figure 11. Results on CIFAR-10: **Left**: Accuracy and loss of the Model with classic CNN ; **Right**: Accuracy and loss of the model with CycleCNN.

5.2. Experiments on a Deep Network

To challenge even further our CycleCNN parameterization, we did some experiments with a deep CNN network of 4 layers on CIFAR-10 dataset. The network architecture can be found in figure 12.

We put in competition the network parameterized with classic CNNs and the network parameterized with our CycleCNNs. The network using CycleCNNs has been parameterized as follow:

- 1st Layer: 1 circle of kernels with 16 points (resulting in 16 kernels)
- 2nd Layer: 2 circles of kernels with 16 points (resulting in 32 kernels)
- 3rd cLayer: 4 circles of kernels with 16 points (resulting in 64 kernels)
- 4th layer: 2 cercles of kernels with 32 points (resulting in 64 kernels)

Our model uses $2\times3\times9+2\times2\times16\times9+2\times4\times32\times9+2\times2\times64\times9=5238$ parameters for convolution, whereas $16\times3\times9+32\times16\times9+64\times32\times9+64\times64\times9=60336$ parameters for the classic CNN.

The obtained results can be be seen in figure 11. Overall, our model loses 20% on both training set and testset. This loss is quite significant, and can be explained by the very low number of degrees of freedom in CycleCNN compared to the baseline. This suggests that parameterizing all convolutional layers with transformers is not expressive enough. Rather, it is natural to have both traditional kernels and kernels generated by transformers in a single layer; we developp this approach in the next section.

5.3. Experiments on an hybrid model

We introduce *hybrid* architectures that combine both implementations: classical CNN kernels in pair with CycleCNN kernels. Testing different ratios between the two types of kernels allows to quantify the influence of the CycleCNN layer.

A model of two layers of convolution is parameterized as follow:

- A first Layer of convolution of 32 kernels classicaly parameterized.
- A second Layer of convolution of 64 kernels. A certain ratio h of the kernels being parameterized by our CycleCNN models (we call them cyclic kernels in figure 14) and a ratio $1-h$ classicaly parameterized.
- A Readout Layer (FC layer) used to predict logits for classification.

The cyclic kernels have been systematically parameterized as being on 4 circles. A diagram of the architecture is presented in Figure 13.

In Figure 14 we give the results of the model for different amounts h of ”classic” and ”cyclic” kernels in the second layer, with a varying depth of the transformer used for parameterizing the cyclic kernels. We also write in the last column the initial learning rates used for the transformer and the rest of the model (as explained in section 4.3). Looking at the relative performances w.r.t. the need in parameters in the second layer of convolution, we make the following observations.

- Comparing the 1st line of the table with the two last lines of the table: if the layer only uses cyclic kernels, the drop in accuracy is not too strong (85/75 going to 72/60).
- Adding cyclic kernels systematically results in a gain of performances; and more importantly this gain is obtained with a very low cost in parameters.

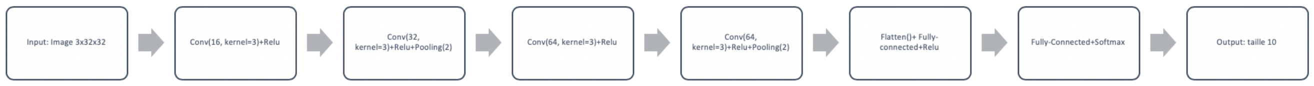

Figure 12. Diagram of the model used for comparisions.

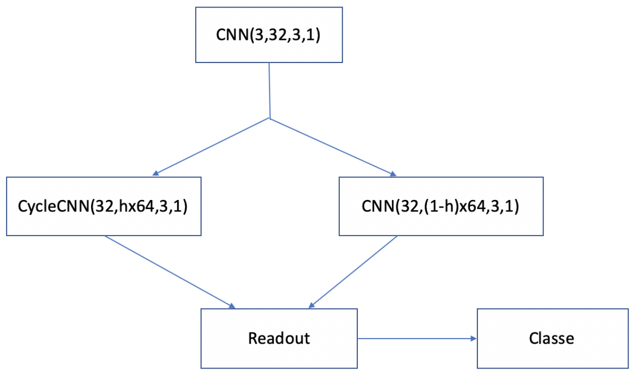

Figure 13. Diagram of the hybrid model.

6. Conclusion

We believe this work is a further step in understanding and exploiting the global topological structure observed in deep neural networks. With the CycleCNN architecture, we think of the shape formed by the kernels instead of focusing on each kernel individually. Parameterising the circle directly, the CycleCNN model is strongly regularised because (i) the number of free parameters is considerably reduced, and (ii) the kernels are forced by design to live on the circle. For some challenging classification tasks, e.g. CIFAR, the model eventually lacks expressiveness. However, considering hybrid models, our experiments provide evidence that CycleCNN layers can be viewed as a cheap boost for existing models. We believe that there are various interesting directions to explore, which we leave for future work:

- Since a CycleCNN layer is essentially described by a circle, the resulting architecture is highly interpretable. Namely, it would be interesting to analyze the evolution of the ellipsoids in kernel space during training, accross various classification tasks.

- As observed in [Gunnar Carlsson, 2018], the spatial filters in deep networks, e.g. VGG16, can form more complex topological structures than cycles, e.g. the Klein bottle. We note that, as long as we are given a fixed embedding of a given topological shape in Euclidean space, together with a regular sampling of this shape, our approach to transferring this shape in kernel space can be adapted. So we may for instance consider the embedding of the Klein Bottle in $\mathbb{R}^4$ considered in [Gunnar Carlsson, 2018], the canonical embeddings of the 2-torus, or in fact any graph or cell complex. It would be nice to compare the resulting architectures. Once a zoo of shapes is established, an ambitious task is to derive a procedure which, given a classification task, decide automatically which topological shape induces the most relevant architecture.

- It may be desirable to provide kernels in the CycleCNN architecture with more degrees of freedoms. For this, a natural idea is to associate an angle parameter to each kernel, allowing the kernel to move inside the circle.

References

[A.B. Lee and Mumford, 2003] A.B. Lee, K. P. and Mumford, D. (2003). The non-linear statistics of high-contrast patches in natural images. In *Intl. Jour.of Computer Vision*.

[Frankle and Carbin, 2018] Frankle, J. and Carbin, M. (2018). The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*.

[G. Carlsson and Zomorodian, 2008] G. Carlsson, T. Ishkhanov, V. d. S. and Zomorodian, A. (2008). On the local behavior of spaces of natural images. In *Intl. Jour. Computer Vision*.

[Gale et al., 2019] Gale, T., Elsen, E., and Hooker, S. (2019). The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*.

[Gunnar Carlsson, 2018] Gunnar Carlsson, R. B. G. (2018). Topological approaches to deep learning.

[Gunnar Carlsson, 2019] Gunnar Carlsson, R. B. G. (2019). Exposition and interpretation of the topology of neural networks.

[Hatcher, 2005] Hatcher, A. (2005). *Algebraic topology*. .

[Mallat, 1999] Mallat, S. (1999). *A wavelet tour of signal processing*. Elsevier.

[Maria et al., 2014] Maria, C., Boissonnat, J.-D., Glisse, M., and Yvinec, M. (2014). The gudhi library: Simplicial complexes and persistent homology. In *International Congress on Mathematical Software*, pages 167–174. Springer.

[Oudot, 2015] Oudot, S. Y. (2015). *Persistence theory: from quiver representations to data analysis*, volume 209. American Mathematical Society Providence.

[Singh et al., 2007] Singh, G., Mémoli, F., and Carlsson, G. (2007). Topological methods for the analysis of high dimensional data sets and 3d object recognition. In *PBG@Eurographics*.

[Tai et al., 2015] Tai, C., Xiao, T., Zhang, Y., Wang, X., et al. (2015). Convolutional neural networks with low-rank regularization. *arXiv preprint arXiv:1511.06067*.

#Classic kernels	#Cyclic kernels	Train	Test	#parameters CNN	transformer size	lr
64	0	85.2 %	75 %	18432	x	0.05
64	0	85.5 %	74.5 %	18432	x	0.01
48	16	**84.8** %	**74** %	13918	3	0.01/0.001
48	16	84.5 %	73.8 %	13851	1	0.01/0.001
48	0	84 %	73.2 %	13824	x	0.01
32	32	**82.3** %	**73.5** %	9404	3	0.01/0.001
32	32	81.5 %	72.8 %	9270	1	0.01/0.001
32	0	80.8 %	72.2 %	9216	x	0.01
16	48	75 %	70.6 %	4890	3	0.01/0.001
16	48	**79** %	**71.2** %	4689	1	0.01/0.001
16	0	73.5 %	69.3 %	4608	x	0.01
0	64	54 %	52 %	376	3	0.01/0.001
0	64	72 %	60 %	108	1	0.01/0.001

Figure 14. Results for a layer parameterized with different amounts Classic Kernels and Cyclic kernels. The model can be found in Annexe. The Cycle kernels are constructed with 4 circles.