Journal of Machine Learning Research 24 (2023) 1-35 Submitted 1/21; Revised 2/23; Published 2/23

Topological Convolutional Layers for Deep Learning

Ephy R. Love ELOVE4@VOLS.UTK.EDU
Bredesen Center DSE
University of Tennessee
Knoxville, TN 37996, USA

Benjamin Filippenko BENFILIP@STANFORD.EDU
Department of Mathematics
Stanford University
Stanford, CA 94305, USA

Vasileios Maroulas VMAROULA@UTK.EDU
Department of Mathematics
University of Tennessee
Knoxville, TN 37996, USA

Gunnar Carlsson GUNNAR@MATH.STANFORD.EDU
Department of Mathematics
Stanford University
Stanford, CA 94305, USA

Editor: Genevera Allen

Abstract

This work introduces the Topological CNN (TCNN), which encompasses several topologically defined convolutional methods. Manifolds with important relationships to the natural image space are used to parameterize image filters which are used as convolutional weights in a TCNN. These manifolds also parameterize slices in layers of a TCNN across which the weights are localized. We show evidence that TCNNs learn faster, on less data, with fewer learned parameters, and with greater generalizability and interpretability than conventional CNNs. We introduce and explore TCNN layers for both image and video data. We propose extensions to 3D images and 3D video.

Keywords: machine learning, convolutional neural network, topology, topological data analysis, image and video classification

Contents

1. Introduction

It was observed in LeCun et al. (1998) that one motivation for the construction of Convolutional Neural Networks (CNNs) was that they permit a sparsification based on the geometry of the space of features. In the case of convolutional neural networks for images, the geometry used was that of a two-dimensional grid of features, in this case pixels. Robinson (2014) has also pointed out the importance of geometries or topologies on spaces of features, coining the term *topological signal processing* to describe this notion. In this paper, we study a space of image filters closely related to a subfamily of the Gabor filters, whose geometry is that of a well known geometric object, the *Klein bottle*[1]. These filters augment the feature set in the data, but they can also be used to construct analogues of convolutional neural networks with improved performance on a number of measures of performance. The method uses a discretization (in the form of a graph) of the Klein bottle as a template for new layers, which produces additional sparsification.

We implement the use of the Klein bottle geometry and its image filters via additional structure on convolutional layers. We call neural networks with these layers **Topological Convolutional Neural Networks (TCNNs)**. We perform experiments on image and video data. The results show significant improvement in TCNNs compared to conventional CNNs with respect to various metrics.

Deep neural network (NN) architectures are the preeminent tools for many image classification tasks since they are capable of distinguishing between a large number of classes with a high degree of precision and accuracy Guo et al. (2016). CNNs are components of the most commonly used neural network architecture for image classification, e.g. see He et al. (2016); Rawat and Wang (2017); Krizhevsky et al. (2012). The characterizing property of CNNs is their use of convolutional layers which take advantage of the 2-dimensional topology of an image to sparsify a fully connected network and employ weight sharing across slices. Each convolutional layer in a CNN assembles spatially local features, e.g. textures, lines, and edges, into complex global features, e.g. the location and classification of objects. CNNs are also used to classify videos; see e.g. Soomro et al. (November,

1. See Figure 2 for an image of a Klein Bottle immersed in $\mathbb{R}^3$.

2012), Schuldt et al. (2004), and Gorelick et al. (2007). CNNs have several major drawbacks including that the models are difficult to interpret, require large datasets, and often do not generalize well to new data Zheng et al. (2018). It has been demonstrated that as CNNs grow in size and complexity they often do not enjoy a proportional increase in utility He et al. (2016). This suggests that bigger, deeper models alone will not continue to advance image classification.

Topological data analysis (TDA) encompasses a set of methods that focuses on the shape of data. Persistent homology, the most popular TDA technique, topologically summarizes data into a persistence diagram (PD), which can be used for shape-based inference. Since the space of PDs lacks a Hilbert space structure, they may not be directly amenable to commonly-used statistical learning methods. A large body of work sought to remedy this shortcoming by inventing well-behaved Hilbert space representations of PDs such as the persistent landscapes, introduced in the work of Bubenik (2015b), and persistent images of Adams et al. (2017). Other works derive PD representations that serve as sufficient statistics thereby ensuring that PD summaries retain all statistically-pertinent information for an inference task. Some authors avoid Hilbert space representations altogether, choosing instead to work directly in a PD space. This is achieved, for example, by leveraging stability results to push statistical distributions from data space forward to PD space as in the study of Mileyko et al. (2011), or by adopting tools from point process theory as in Maroulas et al. (2020); Papamarkou et al. (2022); Adler et al. (2017). Similarly, various methods attempt to connect machine learning with TDA. Indeed, Hofer et al. (2017) and Hofer et al. (2019) examined requirements for integrating topological considerations into deep learning architectures. Moor et al. (2020) proposed a topological autoencoder and the study of Oballe et al. (2022) established topological functional units, a new trainable neural network unit with a PD dissimilarity function as its activation. More recently, there has been a trend to engage graph neural networks that include topological features of graph neighborhoods as in Zhao et al. (2020), or approximate the underlying manifold of the data using topological properties and embed those into a deep learning scheme. Indeed, Mitchell et al. (2022) decode neuronal data responsible for navigation using a novel simplicial convolutional recurrent neural network. The reader may refer to Amézquita et al. (2020), Hensel et al. (2021), and references therein for more details and various studies in TDA and machine learning.

In this paper, we introduce a new architecture where the topological structure in the layers of a TCNN is inspired from prior research on natural image data with topological data analysis (TDA). Indeed, in Carlsson et al. (2008), it was found that 3×3 patches in a database of natural images cluster around an embedded 2-manifold homeomorphic to a Klein bottle, which we denote by $\mathcal{K}$. The patches corresponding to $\mathcal{K}$ are essentially edges, lines, and interpolations between those; see the top panels of Figure 3(b)(c). This observation is a systematic geometrical formulation of the results of Hubel and Weisel (1959) which demonstrate that single neurons in the primary visual cortex of the cat are tuned to both edges and lines (or ridges), at various angles.

It has also been suggested through TDA that the filters in a CNN with 3×3 filter size arrive at filters which naturally correspond points of $\mathcal{K}$ after learning Carlsson and Gabrielsson (2020). The key idea of TCNNs is to first observe that there is a second embedded Klein bottle $\mathcal{K}^{\text{alg}}$ within the space of 3×3 patches which (a) lies close to $\mathcal{K}$ and (b) has a simple algebraic description, and then to use discretizations of $\mathcal{K}^{\text{alg}}$ directly in constructing the convolutional layers. For example, the patches in $\mathcal{K}^{\text{alg}}$ are used in TCNNs as convolutional filters that are fixed during training. Because of symmetries present in $\mathcal{K}^{\text{alg}}$, there are generalized notions of weight sharing which encode invariances under rotation and black-white reversal. Note that the filters from CNNs studied in Carlsson and Gabrielsson (2020) are not used, but rather the algebraically defined filters from $\mathcal{K}^{\text{alg}}$ are used. In other words, there is no use of pretrained filters, but rather synthetically defined filters from $\mathcal{K}^{\text{alg}}$ are used.

There is an analogue of this methodology for video data, which we introduce in Section 5. Roughly, the tangent bundle $T(\mathcal{K}^{\text{alg}})$ to $\mathcal{K}^{\text{alg}}$ parameterizes video filters which improve classification accuracy when used as weights in 3D convolutions. We demonstrate that a TCNN ResNet with these

video filters significantly outperforms a standard ResNet on the UCF-101 dataset. We also show that these video filters improve generalization accuracy from the KTH dataset to the Weizmann dataset.

Remark: We note that while the insight that $\mathcal{K}^{\text{alg}}$ should be used for static images was obtained from empirical scientific observations, the further insight that $T(\mathcal{K}^{\text{alg}})$ should be the corresponding object for video required only mathematical reasoning. One may argue that this kind of mathematical reasoning is just a leap of faith, in contrast, the reader may regard it as one of the strengths of the method that it permits such leaps of faith which can shortcut time consuming experimental work. Similar kinds of reasoning apply to extensions to 3-dimensional imaging and 3-dimensional video, with applications, for example, to fMRI. The ability to reason simultaneously in this way about signal processing, feature generation, and neural network design is, in our view, one of the most attractive aspects of our approach.

Remark: The reader might find it confusing that the statistical study of natural images shows that the patches themselves concentrate around a Klein bottle, while the neurons in the primary visual cortex and the weight vectors and the filters in CNNs are reflecting responses or functions on the space of patches. The relationship between these two alternatives arises from the fact that a 3×3 image patch, which is a 9-vector, can be regarded as a function on patches via the inner product constructions. Thus, points on $\mathcal{K}$ or $\mathcal{K}^{\text{alg}}$ can be regarded as functions on the set of all 3×3 image patches.

The method is not simply an addition of features to particular data sets, but is in fact a general methodology that can be applied to all image or video data sets. In the case of images, it builds in the notion that edges and lines are important features for any image data set and that this notion should be included and accounted for in the architecture for any kind of image data analysis. TDA contains a powerful set of methods that can be used to discover these latent manifolds on which data sit Bubenik (2015a); Chazal et al. (2017); Maroulas et al. (2019); Sgouralis et al. (2017); Marchese et al. (2017). These methods can then be used to inform the parameterization of the TCNN.

TCNNs are composed of two new types of convolutional layers which construct topological features and restrict convolutions based on embeddings of topological manifolds into the space of images (and similarly for video). TCNNs are inherently easier to interpret than CNNs since in one type of layer (Circle Filters layer and Klein Filters layer; Definition 9) the local convolutional kernels are easily interpreted as points on the Klein bottle, all of which have clear visual meaning (e.g. edges and lines). Following a Klein Filters layer, it makes sense to insert our other type of topological layer (Circle One Layer and Klein One Layer; Definitions 7 and 8) in which the 2D slices in the input and output are parameterized by a discretization of the Klein bottle and all weights between slices that are farther away than a fixed threshold distance on the Klein bottle are held at zero throughout training. This has the effect of localizing connections in this convolutional layer to slices that are nearby to each other as points on the Klein bottle. The idea of this 'pruned' convolutional layer is that it aggregates the output of Klein bottle filters from the first layer that are all nearby each other. *Both of these new types of topological convolutional layers can be viewed as a strict form of regularization, which explains the ability of the TCNN to more effectively generalize to new data.* We also provide versions of these layers for video and 3D convolutions.

The main points of the paper are as follows.

1. The method provides improved performance on measures of accuracy, speed of learning and data requirements, and generalization over standard CNNs.

2. The topological layers can be substituted for standard convolutional neural network layers that occur in other approaches, and one should expect improvements in these other situations as well. In this paper, we compare TCNNs to traditional CNNs with the same architecture except for the modified convolutional layers. The comparatively better performance of the TCNNs provides evidence that TCNNs improve the general methodology of CNNs, with the expectation that these advantages can be combined with other known methods. For example,

we use TCNN layers in a ResNet for video classification; see Section 5. As another example, we expect state-of-the-art transfer learning results to improve when some convolutional layers in the network are replaced by TCNN layers.

3. Our approach suggests a general methodology for building analogues of neural networks in contexts other than static images. We carry this out for the study of video data. In this case, the Klein bottle $\mathcal{K}$ is replaced by a different manifold, the so-called *tangent bundle* to $\mathcal{K}$, denoted by $T(\mathcal{K})$. There are also straightforward extensions to 3D imaging and video. The improvement afforded by these methods is expected to increase as the data complexity increases, and we find this to be the case in the passage from static images to video.

4. The simple geometries of the feature spaces enable reasoning about the features to use. In the video situation, we found that using the entire manifold was computationally infeasible and selected certain natural submanifolds within $T(\mathcal{K})$, which allowed for the improved performance. Even in the case of static images, there are natural submanifolds of $\mathcal{K}$ which might be sufficient for the study of specific classes of images, such as line drawings, and restricting to them would produce more efficient computation.

5. There are more general methods of obtaining geometries on feature spaces, which do not require that the feature space occur among the family of manifolds already studied by mathematicians. One such method is the Mapper construction Singh et al. (2007). A way to use such structures to produce a sparsified neural network structure has been developed (Carlsson and Gabrielsson, 2020, Section 5.3). The manifold methods described in this paper are encapsulated in the general framework described in Definition 6 which can be applied to other domains.

6. Because of the intuitive geometric nature of the approach, it permits additional transparency into the performance of the neural network.

7. There are no pretrained components in any of the models considered in this paper. This is significant in particular because the vast majority of state-of-the-art models for classification of the UCF-101 video dataset do use pretrained components Kalfaoglu et al. (2020), Qiu et al. (2019), Carreira and Zisserman (2017) .

The structure of the paper is as follows. In Section 2 we recall the basic structure of convolutional neural networks and we set up notation. TCNNs are introduced in Section 3; the version for image data is in Section 3.1, the video version is in Section 3.2, and a connection with Gabor filters is explained in Section 3.3. Then in Section 4 we describe experiments and results comparing TCNNs to standard CNNs on image data. In Section 5 we do similar experiments on video data.

2. Background: CNNs

In this section we describe in detail the components of the CNN critical to the construction of the TCNN. One of the central motivations of the structure of a CNN is the desire to balance the learning of spatially local and global features. A convolutional layer can be thought of as a sparsified version of the fully connected, feed-forward (FF) layer, with additional homogeneity enforced across subsets of weights. The traditional CNN makes use of the latent image space by creating a small perceptive field with respect to the $L^\infty-$distance in which weights can be nonzero, thus sparsifying the parameter space by enforcing locality in the image. Additionally, homogeneity of weights across different local patches is enforced, further reducing the parameter space by using the global structure of the grid.

Next, we draw parallels between CNNs and TCNNs by adopting the language found in Carlsson and Gabrielsson (2020), which we summarize as needed. First, we describe a feed forward neural network (FFNN) as a directed, acyclic graph (Definition 1).

Definition 1. *A* **Feed Forward Neural Network (FFNN)** *is a directed acyclic graph* Γ *with a vertex set* $V(\Gamma)$ *satisfying the following properties:*

1. $V(\Gamma)$ *is decomposed as the disjoint union of its* **layers**
$$V(\Gamma) = V_0(\Gamma) \sqcup V_1(\Gamma) ... \sqcup V_r(\Gamma).$$
2. *If* $v \in V_i(\Gamma)$*, then every edge* (v, w) *of* Γ *satisfies* $w \in V_{i+1}(\Gamma)$*.*
3. *For every non-initial node* $w \in V_i(\Gamma) : i > 0$*, there is at least one* $v \in V_{i-1}(\Gamma)$ *such that* (v, w) *is an edge of* Γ*.*

The vertices in $V(\Gamma)$ *are also called* **nodes***. For all* i*,* $V_i(\Gamma)$ *consists of the* **nodes in layer** $\boldsymbol{i}$*. The* 0^{th} *layer* $V_0(\Gamma)$ *consists of the* **inputs** *to the neural network (Figure 1 (a)). The last layer* $V_r(\Gamma)$ *consists of the* **outputs***.*

Notation 2. *Let* Γ *be a FFNN with vertex set* $V(\Gamma)$*.*

1. *Often we suppress* Γ *in the notation, writing* $V = V(\Gamma)$ *for the set of all nodes and* $V_i = V_i(\Gamma)$ *for the set of nodes in layer* i*.*
2. *For any* $v \in V$*, the set* $\Gamma(v)$ *consists of all* $w \in V$ *such that* (v, w) *is an edge in* Γ*, and the set* $\Gamma^{-1}(v)$ *consists of all* w *such that* (w, v) *is an edge in* Γ*.*

To describe the edges between nodes in successive layers, we use the notion of a **correspondence** between V_i and V_{i+1}, which is simply a subset of the product $C \subset V_i \times V_{i+1}$. For $v_0 \in V_i$ and $w_0 \in V_{i+1}$, we define the subsets

$$C(v_0) := \{w \in V_{i+1} \mid (v_0, w) \in C\} \subset V_{i+1},$$
$$C^{-1}(w_0) := \{v \in V_i \mid (v, w_0) \in C\} \subset V_i.$$

Note that C is determined by the subsets $C(v_0) \subset V_{i+1}$ for $v \in V_i$, and it is also determined by $C^{-1}(w_0)$ for $w_0 \in V_{i+1}$. In this way, a correspondence is a generalization of a map from V_i to V_{i+1}; Given an element in V_i the correspondence provides a subset of V_{i+1}, and we denote correspondences by $V_i \xrightarrow{C} V_{i+1}$.

We adopt the convention that given nodes $v \in V_i, w \in V_{i+1}$, the edge (v, w) is in Γ if and only if $(v, w) \in C$. Note that this implies $C(v) = \Gamma(v)$. We call C the **edge-defining correspondence of the layer**. In this way, a FFNN specifies an edge-defining correspondence between each pair of successive layers, and conversely, choices of edge-defining correspondences between successive layers specify the edges in a FFNN. The simplest type of layer in a neural network is as follows.

Definition 3. *Let* V_{i+1} *be a layer in a FFNN. We call* V_{i+1} *a* **fully connected layer** *if the edge-defining correspondence* $C \subset V_i \times V_{i+1}$ *is the entire set. In that case, we denote this correspondence* $C_c = V_i \times V_{i+1}$*.*

We proceed to describe convolutional layers. We model digital images as grids indexed by $\mathbb{Z}^2$. Modifications of our constructions to finite size images will be clear. The values of the grid specifying a grayscale image are then triples $(x, y, i) : x, y \in \mathbb{Z}$ where $i \in [0, 1]$ is the intensity at (x, y). Equivalently, an image is a map $\mathbb{Z}^2 \to [0, 1]$. Similarly, videos are modeled as grids indexed by $\mathbb{Z}^3 = \mathbb{Z}^2 \times \mathbb{Z}$, where the $\mathbb{Z}^2$ are the spacial dimensions and the third dimension is time. This generalizes to grids indexed by $\mathbb{Z}^N$ for any positive integer N.

In a CNN, the nodes in each convolutional layer form multiple grids of the same size. We model the nodes in a convolutional layer as the product of a finite index set $\mathcal{X}$ with a grid, $V_i = \mathcal{X} \times \mathbb{Z}^N$.

With this notation, the graph structure of a CNN is specified as in the following definition. A convolutional layer is a sparsification of a fully connected layer which enforces locality in the grids $\mathbb{Z}^N$. For further detail, see Carlsson and Gabrielsson (2020). We explain the homogeneity restrictions on the weights in Eq. (1).

Definition 4. *Let V_{i+1} be a layer in a FFNN. We call V_{i+1} a* **convolutional layer** *or a* **normal one layer (NOL)** *if $V_i = \mathcal{X} \times \mathbb{Z}^N$ and $V_{i+1} = \mathcal{X}' \times \mathbb{Z}^N$ for some finite sets $\mathcal{X}$ and $\mathcal{X}'$ and a positive integer N, and if for some fixed* **threshold** *$s \geq 0$ the edge-defining correspondence $C \subset V_i \times V_{i+1}$ is of the form*

$$C = C_c \times C_{d,N}(s),$$

where $C_c = \mathcal{X} \times \mathcal{X}'$ is the fully connected correspondence and $C_{d,N}(s) \subset \mathbb{Z}^N \times \mathbb{Z}^N$ is the correspondence given by

$$C_{d,N}(s)^{-1}(\underline{x}') := \{\underline{x} \in \mathbb{Z}^N \mid d_{\mathbb{Z}^N}(\underline{x}, \underline{x}') \leq s\}$$

for all $\underline{x}' \in \mathbb{Z}^N$. Here, $d_{\mathbb{Z}^N}$ is the L^∞-metric on $\mathbb{Z}^N$ defined by

$$d_{\mathbb{Z}^N}(\underline{x}, \underline{x}') = \max\{|x_1 - x_1'|, \ldots, |x_N - x_N'|\}.$$

A **Convolutional Neural Network (CNN)** *is a FFNN such that the first layers $V_0, \ldots, V_i$ are convolutional layers and the final layers $V_{i+1}, \ldots, V_r$ are fully connected.*

Remark 5. *The correspondence matrix C of the convolutional layer in Definition 4 is the analogue of the various correspondence matrices that will be introduced for the circle and Klein layers in Definition 7 and Definition 8, respectively. It is common to have pooling layers following convolutional layers in a CNN. Pooling layers downsample the size of the grids. They can be used in TCNNs in the same way and for the same purposes as traditionally used in CNNs. We do not discuss pooling in this paper for simplicity.*

The correspondence $C_{d,N}(s)^{-1}$ maps a vertex to a vertex set that is localized with respect to the threshold s. In a 2-dimensional image, the above definition gives the typical square convolutional filter. This construction results in spatially meaningful graph edges.

The graph structures given in Definitions 1, 3, and 4 yield the skeleton of a CNN. To pass data through the network, we need a system of functions on the nodes and edges of Γ that pass data from layer $i-1$ to layer i. These functions are called **activations** and **weights**. The weights are real numbers $\lambda_{v,w}$ associated to each edge,

$$\Lambda = \{\lambda_{v,w} \mid v \in V_{i-1}, w \in V_i, \text{ and } (v,w) \in \Gamma\}.$$

Let $V_{i-1} = \mathcal{X} \times \mathbb{Z}^N$ and $V_i = \mathcal{X}' \times \mathbb{Z}^N$, as in a CNN. Denote $v = (\kappa, \underline{x}) \in V_{i-1}$ and $w = (\kappa', \underline{x}') \in V_i$. Then the homogeneity of the weights, a characteristic of a CNN, is the translational invariance

$$\lambda_{(\kappa,\underline{x}),(\kappa',\underline{x}')} = \lambda_{(\kappa,\underline{x}+\underline{z}),(\kappa',\underline{x}'+\underline{z})}. \tag{1}$$

Note that there is no translational invariance in the $\kappa \in \mathcal{X}$ coordinate because κ indexes the input and output channels. The activations $\mathcal{A} = \{(u_v, f_v) \mid v \in V(\Gamma)\}$ associate to each vertex v a real number u_v and a function $f_v : \mathbb{R} \to \mathbb{R}$. To pass data from layer $i-1$ to layer i means to determine u_w for $w \in V_i$ from the values $u_v, v \in V_{i-1}$ via the formula

$$u_w := f_v\Big(\sum_{v \in \Gamma^{-1}(w)} \lambda_{v,w} \cdot u_v\Big).$$

The **activation functions** f_v in neural networks usually map $x \in \mathbb{R}$ to $0 < x < 1$ or $0 \leq x$. The output activations are a probability distribution on the output nodes of Γ, so they are non-negative real numbers. We use the activation function ReLU, defined as $f(x) = \max(0, x)$, for

all non-terminal layers, and the softmax $\sigma(x_i) = e^{x_i}(\sum_j e^{x_j})^{-1} : \quad i,j \in \{1...n\}, \quad x_i \in \mathbb{R}^+$ as the terminal layer activation function. We choose a common optimization method, precisely the adaptive moment estimation (Adam), to determine the back-propagation of our changes based on the computed gradients.

Figure 1 (a,b,c) displays an example of a weight and activation system for a convolutional layer from $V_0 = \mathcal{X}(1) \times \mathbb{Z}^2$ to $V_1 = \mathcal{X}(4) \times \mathbb{Z}^2$, where $\mathcal{X}(\eta)$ denotes a finite set of cardinality η. The correspondence is $C_c \times C_{d,2}(2) \subset V_0 \times V_1$, where $C_c = \mathcal{X}(1) \times \mathcal{X}(4)$ is fully connected and $C_{d,2}(2) \subset \mathbb{Z}^2 \times \mathbb{Z}^2$ localizes connections at $L^\infty-$distance 2. Panel (a) shows the input V_0, each weight-matrix in (b) is a vector of coefficients from $\{\lambda_{v,w}\}$, and (c) shows the resulting activations in V_1.

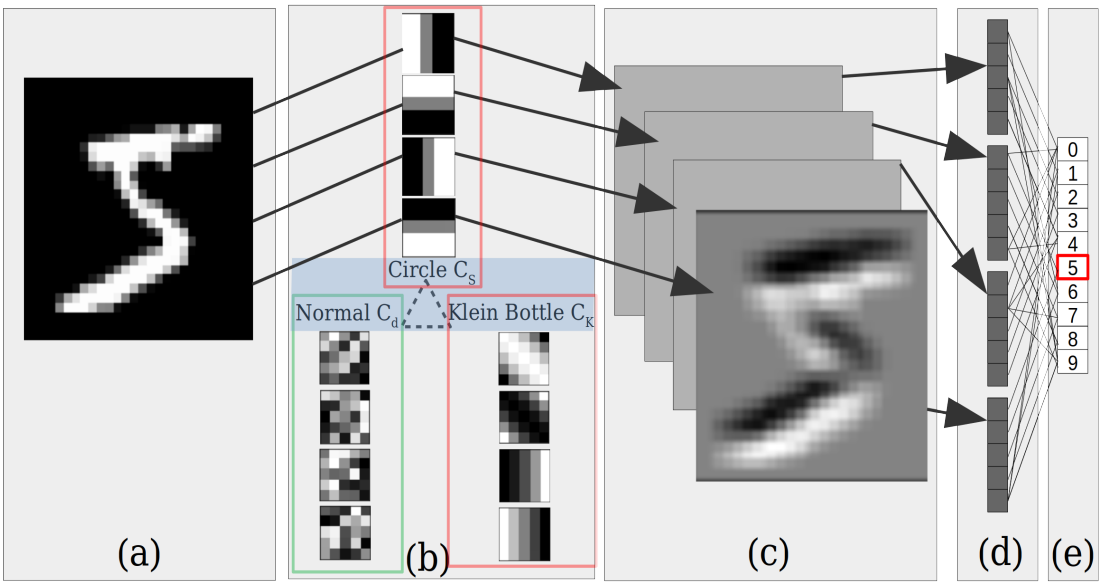


Figure 1: Visual guide to CNN (green rectangle) and TCNN (red rectangles) architectures. A typical CNN layer for image classification takes in a 2-dimensional array (a) as an input image and convolves this image through a system of spatially localized filters (b) to produce multiple slices of processed data (c) called feature maps. The feature maps in (c) are flattened to a column vector in (d) and passed through a fully-connected layer to the output nodes (e) which are of cardinality equal to the number of classes in the prediction space. The TCNN modifies the typical CNN framework by specifying the weights in (b) to lie on a topological manifold such as the circle or Klein bottle. We indicate this choice by the dashed triangle in (b) and red rectangles indicating the selection of circle or Klein weights. This shows the behavior of CF and KF type TCNN layers. The Circle One Layer (COL) and the Klein One Layer (KOL) type TCNN layers also modify the weights of a traditional CNN layer but in a different way: They localize weights with respect to the circle and Klein bottle topologies yet, as usual, instantiate them randomly and allow them to vary during training.

3. Topological Convolutional Layers

We have seen in the previous section that convolutional neural networks may be constructed directly from the graph structure on the grid of pixels, with no further information. The convolutional conditions are also derived directly from the structure of these graphs. This means that one can construct analogous layers and neural networks from other geometries or graphs. In this case, the nodes in the layers will correspond to the vertices of the graph, while the connections will be restricted to those which are local in the given geometry and therefore correspond to edges of the graph. Convolutional conditions on the set of weight vectors can also be obtained, for instance, from symmetries of the geometry or graph. In this section, we will carry this out for two choices of geometries or graph structure. The first will correspond to the product geometry $\mathbb{Z} \times S^1(n)$, where $S^1(n)$ corresponds to the discretization of the circle obtained by selecting the $n-$th roots of unity on the unit circle. The second will correspond to the geometry $\mathbb{Z} \times \mathcal{K}^\delta$, where $\mathcal{K}^\delta$ is obtained by selecting a set of points in $\mathcal{K}$ in a systematic way. The neurons will correspond to the points in the product geometries, not only to the grid points. The network therefore encodes responses not only to the raw pixel information but to derived features obtained from the pixel and its neighbors.

The interpretation of the locality condition in these augmented networks can be thought of as follows. In an ordinary CNN, the locality conditions in the connections encourage the network to learn local conditions, such as directional derivatives in various directions. The circle geometry creates these directional derivatives and allows the deep learner to proceed from there. The locality conditions encourage the network to learn features that are local both within the raw pixels and in the tangential directions encoded in the features from the circle. Therefore, one expects that what is learned in the circle version is second order local information, beginning from the well understood first order information. In the case of the Klein bottle geometries, some second order information is included directly, and the learning will therefore proceed from that information to additional second order information or even third order information. Symmetries of the circle or Klein bottle permit us to enforce rotational invariance of the features created as well as invariance under black/white reversal.

In Section 3.1 we introduce new types of neural network layers that form our TCNNs for 2D image classification: The KF (Klein Filters) layer uses predefined filters in a standard convolutional layer and the KOL (Klein One Layer) prunes connections between slices in a convolutional layer with respect to the Klein bottle's topology. There are analogous layers called CF (Circle Filters) and COL (Circle One Layer) based on the topology of a circle. See Appendix B for pseudocode implementations of TCNNs with CF and COL layers. In Section 3.2 we introduce layers for TCNNs used for video classification. In Section 3.3 we demonstrate the connection between our constructions and Gabor filters.

3.1 2D Images

Locality in a typical convolutional neural network is a function of the $L^\infty-$distance between cells, which is specified by the correspondence $C_{d,2}$ (see Definition 4) in the case of a $2-$dimensional image. We add novel, topological criteria to this notion of locality through metrics on topological manifolds. The general technique is described in Definition 6.

Definition 6. *Let M be a manifold and let $\mathcal{X}, \mathcal{X}' \subset M$ be two discretizations of M, meaning finite sets of points. Let $V_i = \mathcal{X} \times \mathbb{Z}^N$ and $V_{i+1} = \mathcal{X}' \times \mathbb{Z}^N$ be successive layers in a FFNN. Fix a threshold $s \geq 0$. Let d be a metric on M. Define a correspondence $C(s) \subset \mathcal{X} \times \mathcal{X}'$ by*

$$C(s)^{-1}(\kappa') = \{\kappa \in \mathcal{X} \mid d(\kappa, \kappa') \leq s\}, \tag{2}$$

for all $\kappa' \in \mathcal{X}'$. Together with another threshold $s' \geq 0$, this defines a correspondence $C \subset V_i \times V_{i+1}$ by

$$C = C(s) \times C_{d,N}(s'), \tag{3}$$

where $C_{d,N}(s')$ is the convolutional correspondence from Definition 4. This means that

$$\begin{aligned} C^{-1}(\kappa', \underline{x}') &= C_S(s)^{-1}(\kappa') \times C_{d,2}(s')^{-1}(\underline{x}') \\ &= \{(\kappa, x) \in \mathcal{X} \times \mathbb{Z}^N \mid d_S(\kappa, \kappa') \leq s \text{ and } d_{\mathbb{Z}^N}(\underline{x}, \underline{x}') \leq s'\} \end{aligned}$$

for all $(\kappa', \underline{x}') \in \mathcal{X} \times \mathbb{Z}^N$.

Definition 6 provides a general approach for the convolutional correspondence matrix. The layers defined in subsequent Definitions 8, 7, 11 12, are all given by an edge-defining correspondence C as in Eq. (3), however tailored to the specific example. The first example we give is a layer that localizes with respect to a position on a circle in addition to the usual $L^\infty-$locality in a convolutional layer. Let $S^1 = \{\kappa \in \mathbb{R}^2 \mid |\kappa| = 1\}$ be the unit circle in the plane $\mathbb{R}^2$. A typical discretization of S^1 is the set of n-th roots of unity $\mathcal{X} = \{e^{2\pi i k/n} \mid 0 \leq k \leq n-1\}$ for some $n \geq 1$.

Definition 7. *Let $\mathcal{X}, \mathcal{X}' \subset S^1$ be two discretizations of the circle. Let $V_i = \mathcal{X} \times \mathbb{Z}^2$ and $V_{i+1} = \mathcal{X}' \times \mathbb{Z}^2$ be successive layers in a FFNN. Fix a threshold $s \geq 0$.*

The **circle correspondence** *$C_S(s) \subset \mathcal{X} \times \mathcal{X}'$ is defined by*

$$C_S(s)^{-1}(\kappa') = \{\kappa \in \mathcal{X} \mid d_S(\kappa, \kappa') \leq s\}$$

for all $\kappa' \in \mathcal{X}'$, where the metric d_S is given by

$$d_S(\kappa, \kappa') = \cos^{-1}(\kappa \cdot \kappa') \text{ for } \kappa, \kappa' \in S^1.$$

We call V_{i+1} a **Circle One Layer (COL)** *if, for some other threshold $s' \geq 0$, the edge$-$defining correspondence $C \subset V_i \times V_{i+1}$ is of the form*

$$C = C_S(s) \times C_{d,2}(s'),$$

where $C_{d,2}(s')$ is the convolutional correspondence from Definition 4. This means that

$$\begin{aligned} C^{-1}(\kappa', x', y') &= C_S(s)^{-1}(\kappa') \times C_{d,2}(s')^{-1}(x', y') \\ &= \{(\kappa, x, y) \in \mathcal{X} \times \mathbb{Z}^2 \mid d_S(\kappa, \kappa') \leq s \text{ and } d_{\mathbb{Z}^2}((x, y), (x', y')) \leq s'\} \end{aligned}$$

for all $(\kappa', x', y') \in \mathcal{X} \times \mathbb{Z}^2$.

Next, we define a layer that localizes weights with respect to a metric on the Klein bottle $\mathcal{K}$. See Figure 2 for a visualization of the nodes and weights. Recall that $\mathcal{K}$ is the 2-dimensional manifold obtained from $\mathbb{R}^2$ as a quotient by the relations $(\theta_1, \theta_2) \sim (\theta_1 + 2k\pi, \theta_2 + 2l\pi)$ for $k, l \in \mathbb{Z}$ and $(\theta_1, \theta_2) \sim (\theta_1 + \pi, -\theta_2)$. The construction uses an embedding $F_\mathcal{K}$ of $\mathcal{K}$ into the vector space of quadratic functions on the square $[-1, 1]^2$, motivated by the embedded Klein bottle observed in Carlsson et al. (2008) and its appearance in the weights of CNNs as observed in Carlsson and Gabrielsson (2020). An image patch in the embedded Klein bottle $F_\mathcal{K}(\theta_1, \theta_2)$ has a natural 'orientation' given by the angle θ_1. Visually, one sees lines through the center of the image at angle $\theta_1 + \pi/2$; see the top right image in Figure 3. The embedding is given by

$$F_\mathcal{K}(\theta_1, \theta_2)(x, y) = \sin(\theta_2)(\cos(\theta_1)x + \sin(\theta_1)y) + \cos(\theta_2)Q(\cos(\theta_1)x + \sin(\theta_1)y), \tag{4}$$

where $Q(t) = 2t^2 - 1$. As given, $F_\mathcal{K}$ is a function on the torus, which is parameterized by the two angles θ_1 and θ_2. It actually defines a function on $\mathcal{K}$ since it satisfies $F_\mathcal{K}(\theta_1, \theta_2) = F_\mathcal{K}(\theta_1 + 2k\pi, \theta_2 + 2l\pi)$ and $F_\mathcal{K}(\theta_1 + \pi, -\theta_2) = F_\mathcal{K}(\theta_1, \theta_2)$.

Definition 8. *Let $\mathcal{X}, \mathcal{X}' \subset \mathcal{K}$ be two finite subsets of the Klein bottle. Let $V_i = \mathcal{X} \times \mathbb{Z}^2$ and $V_{i+1} = \mathcal{X}' \times \mathbb{Z}^2$ be successive layers in a FFNN. Fix a threshold $s \geq 0$.*

The **Klein correspondence** *$C_{\mathcal{K}}(s) \subset \mathcal{X} \times \mathcal{X}'$ is defined by*

$$C_{\mathcal{K}}(s)^{-1}(\kappa') = \{\kappa \in \mathcal{X} \mid d_{\mathcal{K}}(\kappa, \kappa') \leq s\}$$

for all $\kappa' \in \mathcal{X}'$, where the metric $d_{\mathcal{K}}$ is defined by

$$d_{\mathcal{K}}(\kappa, \kappa') = \left(\int_{[-1,1]^2} \left(F_{\mathcal{K}}(\kappa)(x,y) - F_{\mathcal{K}}(\kappa')(x,y)\right)^2 dxdy \right)^{\frac{1}{2}}$$

for $\kappa, \kappa' \in \mathcal{K}$.

We call V_{i+1} a **Klein One Layer (KOL)** *if, for some other threshold $s' \geq 0$, the edge−defining correspondence $C \subset V_i \times V_{i+1}$ is of the form*

$$C = C_{\mathcal{K}}(s) \times C_{d,2}(s'),$$

which means that

$$\begin{aligned} C^{-1}(\kappa', x', y') &= C_{\mathcal{K}}(s)^{-1}(\kappa') \times C_{d,2}(s')^{-1}(x', y') \\ &= \{(\kappa, x, y) \in \mathcal{X} \times \mathbb{Z}^2 \mid d_{\mathcal{K}}(\kappa, \kappa') \leq s \text{ and } d_{\mathbb{Z}^2}((x,y),(x',y')) \leq s'\} \end{aligned}$$

for all $(\kappa', x', y') \in \mathcal{X}' \times \mathbb{Z}^2$.

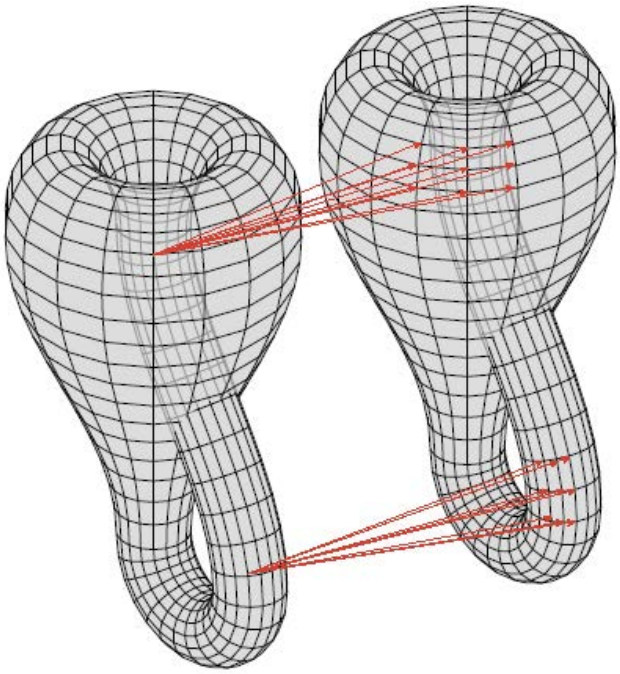


Figure 2: A visual representation of the neurons and weights in a KOL with threshold $s = 2$. Each intersection of grid lines on the two Klein bottles is a neuron. The red arrows depict the nonzero weights corresponding to the given input neuron.

We define two other layers based on the circle and the Klein bottle (Definition 9). First, we define an embedding of the circle S^1 into the space of functions on $[-1,1]^2$ by composing $F_{\mathcal{K}}$ with the embedding $S^1 \hookrightarrow \mathcal{K}$, $\theta \mapsto (\theta, \pi/2)$, i.e.

$$F_{S^1}(\theta)(x,y) := F_{\mathcal{K}}(\theta, \pi/2)(x,y) = \cos(\theta)x + \sin(\theta)y. \tag{5}$$

Now the idea is to build convolutional layers with fixed weights λ given by discretizations of $F_{S^1}(\theta)(x,y)$ and $F_{\mathcal{K}}(\theta_1,\theta_2)(x,y)$. This is motivated by Carlsson and Gabrielsson (2020) which showed that convolutional neural networks (in particular VGG16) learn the filters $F_{S^1}(\theta)$. Instead of forcing the neural networks to learn these weights, we initialize the network with these weights. Intuitively, this should cause the network to train more quickly to high accuracy. Moreover, we choose to fix these weights during training (gradient = 0) to prevent overfitting, which we conjecture contributes to our observed improvement in generalization to new data. We also use discretizations of the images $F_{\mathcal{K}}(\theta_1,\theta_2)$ given by the full Klein bottle as weights, motivated by the reasoning that the trained weights observed in Carlsson and Gabrielsson (2020) are exactly the high-density image patches found in Carlsson et al. (2008) which cluster around the Klein bottle. These layers with fixed weights can be thought of as a typical pretrained convolutional layer in a network.

Definition 9. *Let $M = S^1$ or $\mathcal{K}$ and let $\mathcal{X} \subset M$ be a finite subset. Let $V_i = \mathbb{Z}^2$ and $V_{i+1} = \mathcal{X} \times \mathbb{Z}^2$ be successive layers in a FFNN. Suppose V_{i+1} is a convolutional layer with threshold $s \geq 0$ (Definition 4). Then V_{i+1} is called a* **Circle Filters (CF) layer** *or a* **Klein Filters (KF) layer**, *respectively, if the weights $\lambda_{-,(\kappa,-,-)}$ are given for $\kappa \in \mathcal{X}$ by a convolution over V_i of the filter of size $(2s+1) \times (2s+1)$ with values*

$$Filter(\kappa)(n,m) = \int_{-1+\frac{2m}{2s+1}}^{-1+\frac{2(m+1)}{2s+1}} \int_{-1+\frac{2n}{2s+1}}^{-1+\frac{2(n+1)}{2s+1}} F_M(\kappa)(x,y)dxdy$$

for integers $0 \leq n, m \leq 2s$.

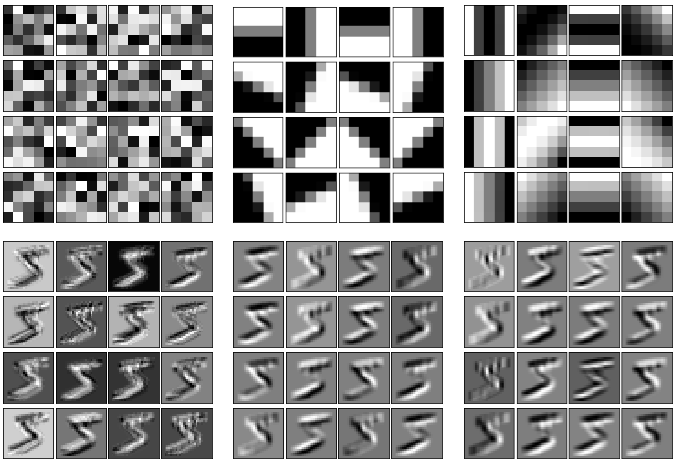
In summary, we have the following. Both $\mathcal{K}$ and S^1 can be discretized into a finite subset $\mathcal{X}$ by specifying evenly spaced values of angles. Given such a discretization $\mathcal{X}$, the convolutional layers in a TCNN have slices indexed by $\mathcal{X}$. Note that $\mathcal{X}$ has a metric induced by the embedding $F_{\mathcal{K}}$ and the L^2-metric on functions on $[-1,1]^2$. The **COL** and **KOL** layers are convolutional layers with slices indexed by $\mathcal{X}$ where all weights between slices whose distances in $\mathcal{X}$ are greater than some fixed threshold are forced to be zero. The **CF** and **KF** layers are convolutional layers with slices indexed by $\mathcal{X}$ and such that the weights are instantiated on the slice corresponding to $(\theta_1,\theta_2) \in \mathcal{K}$ to be the image $F_{\mathcal{K}}(\theta_1,\theta_2)$ discretized to the appropriate grid shape; examples of these weights are shown in Figure 3. These weights are fixed during training. See also the visual guide in Figure 1. An example of trained weights in a COL are included in Figure 17.

Remark 10. *See Algorithm 1 for an implementation of a CF convolutional layer. See Algorithm 2 for an implementation of a two layer convolutional network with first layer CF and second layer COL.*

3.2 Video

For video, the space of features of interest is parameterized by the tangent bundle of the translational Klein bottle $\mathcal{K}^t$, denoted $T(\mathcal{K}^t)$. The translational Klein bottle $\mathcal{K}^t$ is a 3-dimensional manifold and its tangent bundle $T(\mathcal{K}^t)$ is 6-dimensional. These manifolds parameterize video patches (6) in a manner related to the Klein bottle parameterization $F_{\mathcal{K}}$ of 2D images patches from (4).

Before providing the precise definitions of $\mathcal{K}^t$ and $T(\mathcal{K}^t)$ as well as the formulas for the parameterizations, we describe the idea roughly. An image patch in the embedded Klein bottle $F_{\mathcal{K}}(\theta_1,\theta_2)$ has a natural 'orientation' given by the angle θ_1. Visually, one sees lines through the center of the



(a) Normal (b) CF (c) KF

Figure 3: Table of weights (top row) and activations (bottom row) for 3 convolutional layers (a),(b),(c) evaluated on a handwritten 5 from MNIST (initial data shown in Figure 1). The first column (a) is a normal CNN layer (NOL) trained on one epoch of MNIST data followed by 2 fully connected linear layers. This network has a testing accuracy of approximately 99%. Second (b) is a Circle Filters (CF) layer. Third (c) is a Klein Filters (KF) layer.

image at angle $\theta_1 + \pi/2$. Given a real number r, there is a 2D image patch $F_{\mathcal{K}^t}(\theta_1, \theta_2, r)$ given by the translation of $F_{\mathcal{K}}(\theta_1, \theta_2)$ by r units along the line through the origin at angle θ_1, i.e. along the line perpendicular to the lines in the image. One can extend this image to a video that is constant in time. Videos that change in time are obtained by enlarging $\mathcal{K}^t$ to its tangent bundle $T(\mathcal{K}^t)$. The tangent bundle consists of pairs of a point $(\theta_1, \theta_2, r) \in \mathcal{K}^t$ and a vector (u, v, w) tangent to (θ_1, θ_2, r). The embedding $F_{T(\mathcal{K}^t)}$ sends such a pair to a video patch $F_{T(\mathcal{K}^t)}(\theta_1, \theta_2, r, u, v, w)$ that at time t is the image $F_{\mathcal{K}^t}(\theta_1 + tu, \theta_2 + tv, r + tw)$. For example, $F_{T(\mathcal{K}^t)}(\theta_1, \theta_2, 0, 0, 0, 1)$ is the video patch that translates $F_{\mathcal{K}}(\theta_1, \theta_2)$ at unit speed along the line through the origin at angle θ_1. Similarly, $F_{T(\mathcal{K}^t)}(\theta_1, \theta_2, 0, 1, 0, 0)$ is the video patch that rotates $F_{\mathcal{K}}(\theta_1, \theta_2)$ at unit speed.

Denote the coordinates on $\mathbb{R}^3 \times \mathbb{R}^3$ by the variables $(\theta_1, \theta_2, r, u, v, w)$. The variables (θ_1, θ_2, r) parameterize $\mathcal{K}^t$ and the variables (u, v, w) parameterize the tangent spaces. To be precise, $\mathcal{K}^t$ is given as the quotient of $\mathbb{R}^3$ by the relations $(\theta_1, \theta_2, r) \sim (\theta_1 + 2k\pi, \theta_2 + 2l\pi, r)$ for all $k, l \in \mathbb{Z}$ and $(\theta_1, \theta_2, r) \sim (\theta_1 + \pi, -\theta_2, -r)$, and, similarly, $T(\mathcal{K}^t)$ can be described as a quotient of $\mathbb{R}^3 \times \mathbb{R}^3$. We suppress further discussion of these relations because they are only significant to this work in that they are respected by the embeddings $F_{\mathcal{K}^t}$ and $F_{T(\mathcal{K}^t)}$.

Let $I = [-1, 1]$. Denote by $C(I^2, I)$ the space of continuous functions $I^2 \to I$, which represent image patches at infinite resolution, and similarly denote the space of video patches by $C(I^2 \times I, I)$. The embeddings

$$F_{\mathcal{K}^t} : \mathcal{K}^t \to C(I^2, I)$$

and

$$F_{T(\mathcal{K}^t)} : T(\mathcal{K}^t) \to C(I^2 \times I, I)$$

are given by

$$\begin{aligned} F_{\mathcal{K}^t}(\theta_1, \theta_2, r)(x, y) = &\sin(\theta_2)(\cos(\theta_1)(x + r\cos(\theta_1)) + \sin(\theta_1)(y + r\sin(\theta_1))) \\ &+ \cos(\theta_2)Q(\cos(\theta_1)(x + r\cos(\theta_1)) + \sin(\theta_1)(y + r\sin(\theta_1))) \end{aligned}$$

and

$$F_{T(\mathcal{K}^t)}(\theta_1,\theta_2,r,u,v,w)(x,y,t) := F_{\mathcal{K}^t}(\theta_1+tu,\theta_2+tv,r+tw), \tag{6}$$

where $Q(z) = 2z^2 - 1$.

Using the embedding $F_{T(\mathcal{K}^t)}$, we define a metric on $T(\mathcal{K}^t)$ by pulling back the L^2 metric on $C(I^2 \times I, I)$,

$$d_{T(\mathcal{K}^t)}(\kappa,\kappa') := \left(\int_{I^2\times I} \left(F_{T(\mathcal{K}^t)}(\kappa)(x,y,t) - F_{T(\mathcal{K}^t)}(\kappa')(x,y,t)\right)^2 dxdydt \right)^{\frac{1}{2}} \text{ for } \kappa,\kappa' \in T(\mathcal{K}^t). \tag{7}$$

The metric $d_{T(\mathcal{K}^t)}$ allows us to define a new type of layer in a neural network. Recall the $3D$ version of the convolutional correspondence $C_{d,3}(s)$ from Definition 4.

Definition 11. (6D Moving Klein Correspondence) *Let $\mathcal{X}, \mathcal{X}' \subset T(\mathcal{K}^t)$ be two finite subsets. Let $V_i = \mathcal{X} \times \mathbb{Z}^3$ and $V_{i+1} = \mathcal{X}' \times \mathbb{Z}^3$ be successive layers in a FFNN. Fix a threshold $s \geq 0$.*

The **6D Moving Klein correspondence** *$C_{T(\mathcal{K}^t)}(s) \subset \mathcal{X} \times \mathcal{X}'$ is defined by*

$$C_{T(\mathcal{K}^t)}(s)^{-1}(\kappa') = \{\kappa \in \mathcal{X} \mid d_{T(\mathcal{K}^t)}(\kappa,\kappa') \leq s\}$$

for all $\kappa' \in \mathcal{X}$, where the metric $d_{T(\mathcal{K}^t)}$ is defined in (7).

We call V_{i+1} a **6D Moving Klein One Layer (6MKOL)** *if, for some other threshold $s' \geq 0$, the edge−defining correspondence $C \subset V_i \times V_{i+1}$ is of the form*

$$C = C_{T(\mathcal{K}^t)}(s) \times C_{d,3}(s'),$$

which means that

$$\begin{aligned} C^{-1}(\kappa',x',y',t') &= C_{T(\mathcal{K}^t)}(s)^{-1}(\kappa') \times C_{d,3}(s')^{-1}(x',y',t') \\ &= \{(\kappa,x,y,t) \in \mathcal{X}\times\mathbb{Z}^3 \mid d_{T(\mathcal{K}^t)}(\kappa,\kappa') \leq s \text{ and } d_{\mathbb{Z}^3}((x,y,t),(x',y',t')) \leq s'\} \end{aligned}$$

for all $(\kappa',x',y',t') \in \mathcal{X} \times \mathbb{Z}^3$.

There are particular submanifolds of $T(\mathcal{K}^t)$ whose corresponding video patches we conjecture to be most relevant for video classification. In 6MKOL layers, we often choose $\mathcal{X}$ and $\mathcal{X}'$ to be subsets of these submanifolds. One reason to do this is that discretizing the 6-dimensional manifold $T(\mathcal{K}^t)$ results in a large number of filters, significantly bloating the size of the neural network. Indeed, discretizing θ_1 and θ_2 into 4 values each, as in our Klein bottle experiments, and discretizing the other dimensions into only 3 values produces $4^2 * 3^4 = 1296$ points in $\mathcal{X}$. Moreover, these are videos rather than static images, so they contain many pixels: 5^3 pixels for 5×5 video with 5 time steps. Another reason to specialize to submanifolds is one general philosophy of this paper: well-chosen features, rather than an abundance of features, provide better generalization due to less over-fitting.

The five 2-dimensional submanifolds of $T(\mathcal{K}^t)$ that we choose to work with are

$$\begin{aligned} \tilde{\mathcal{K}} &:= \{(\theta_1,\theta_2,0,0,0,0) \in T(\mathcal{K}^t)\}, \\ S_\tau^\pm &:= \{(\theta_1,\theta_2,0,0,0,\pm 1) \in T(\mathcal{K}^t)\}, \\ S_\rho^\pm &:= \{(\theta_1,\theta_2,0,\pm 1,0,0) \in T(\mathcal{K}^t)\}. \end{aligned} \tag{8}$$

Under the embedding $F_{T(\mathcal{K}^t)}$, $\tilde{\mathcal{K}}$ corresponds to the Klein bottle images held stationary in time, $S_\tau^\pm$ corresponds to the Klein bottle images translating in time perpendicular to their center line, as described above, where the sign $\pm$ controls the direction of translation, and $S_\rho^\pm$ corresponds to the Klein bottle images rotating either clockwise or counterclockwise depending on the sign $\pm$.

We now define convolutional layers with fixed weights given by discretizations of the video patches corresponding to the manifolds $T(\mathcal{K}^t), \tilde{\mathcal{K}}, S_\tau^\pm$, and $S_\rho^\pm$. These can be viewed as pretrained convolutional layers that would typically appear as the first layer in a pretrained video classifier. The

motivation is the same as for the analogous CF and KF layers defined for images in Definition 9 – faster training and better generalization to new data due to initializing the network with meaningful filters that are fixed during training.

Definition 12. *Let $M \subset T(\mathcal{K}^t)$ be any subset, for example a submanifold such as $\tilde{\mathcal{K}}, S^{\pm}_{\tau}$, or $S^{\pm}_{\rho}$ (see (8)), or unions of such submanifolds. Let $\mathcal{X} \subset M$ be a finite subset.*

Let $V_i = \mathbb{Z}^3$ and $V_{i+1} = \mathcal{X} \times \mathbb{Z}^3$ be successive layers in a FFNN. Suppose V_i is a convolutional layer with threshold $s \geq 0$ (i.e., the edge-defining correspondence is $C_c \times C_{d,3}(s)$ as in Definition 4). Then V_{i+1} is called a **M-Features (M-F) layer** *if the weights $\lambda_{-,(\kappa,-,-,-)}$ are given for $\kappa \in \mathcal{X}$ by a convolution over V_i of the filter of size $(2s+1) \times (2s+1) \times (2s+1)$ with values*

$$Filter(\kappa)(n,m,p) = \int_{-1+\frac{2p}{2s+1}}^{-1+\frac{2(p+1)}{2s+1}} \int_{-1+\frac{2m}{2s+1}}^{-1+\frac{2(m+1)}{2s+1}} \int_{-1+\frac{2n}{2s+1}}^{-1+\frac{2(n+1)}{2s+1}} F_{T(\mathcal{K}^t)}(\kappa)(x,y,t)dxdydt$$

for integers $0 \leq n, m, p \leq 2s$.

3.3 Gabor filters versus Klein bottle filters

The Klein bottle filters given by $F_{\mathcal{K}}(\theta_1, \theta_2)$ as in (4) are related to Gabor filters. In fact, besides a minor difference, they are a particular type of Gabor filters. The purpose of this section is to explain this relationship. The significance of this relationship is that, while Gabor filters are commonly used in image recognition tasks, our constructions use a particular 2-parameter family of Gabor filters that is especially important, as identified by the analysis in Carlsson et al. (2008). Restricting to this 2-parameter family provides a compact set of filters that can be effectively used as pretrained weights in a neural network. One may use other families of Gabor filters for this purpose, but then the question is on what basis does one choose a particular family. The Klein bottle filters are a topologically justified choice. In Section 4.1.6, we compare the performance of some other choices of Gabor filters with the Klein bottle filters.

Recall that the Gabor filters are functions on the square $[-1,1]^2$ given by

$$\begin{aligned} g(\lambda,\omega,\psi,\sigma,\gamma)(x,y) &= e^{-\frac{x'^2+\gamma^2 y'^2}{2\sigma^2}} \times \cos\left(\frac{2\pi x'}{\lambda} + \psi\right), \\ x' &= x\cos(\omega) + y\sin(\omega), \\ y' &= -x\sin(\omega) + y\cos(\omega). \end{aligned} \tag{9}$$

The Klein bottle filters $F_{\mathcal{K}}(\theta_1, \theta_2)$ do not taper off in intensity near the edges of the square, so we would like to remove this tapering from the Gabor filters too for a proper comparison. This is done by removing the exponential term from g, or equivalently, setting $\sigma = \infty$. This simultaneously removes the dependence on γ. So we have the restricted class of filters

$$g(\lambda,\omega,\psi,\infty,0)(x,y) = \cos\left(\frac{2\pi}{\lambda}x'(\omega) + \psi\right).$$

Given θ_1, θ_2 parameterizing the Klein bottle filter $F_{\mathcal{K}}(\theta_1, \theta_2)$, we claim that a similar Gabor filter is given by

$$\begin{aligned} \tilde{g}(\theta_1,\theta_2) &:= g\left(2 + \frac{4}{\pi}\theta_2,\ \theta_1,\ \theta_2 + \pi, \infty, 0\right) \\ &= -\cos\left(\frac{\pi \cdot x'(\theta_1)}{1 + \frac{2}{\pi}\theta_2} + \theta_2\right). \end{aligned}$$

To see the similarity between $\tilde{g}(\theta_1, \theta_2)$ and $F_{\mathcal{K}}(\theta_1, \theta_2)$, we examine the 'primary circle' $\theta_2 = \pi/2$ and the 'small circle' $\theta_2 = 0$. The formula for other values of θ_2 interpolates along the Klein bottle

between these two circles. On the primary circle, we have

$$\tilde{g}(\theta_1, \pi/2) = \sin\left(\frac{\pi}{2} \cdot x'(\theta_1)\right)$$
$$F_{\mathcal{K}}(\theta_1, \pi/2) = x'(\theta_1).$$

These are both odd functions of $x'(\theta_1)$ that are equal to ± 1 at $x'(\theta_1) = \pm 1$ and are equal to 0 at $x'(\theta_1) = 0$. Similarly, on the small circle, we have

$$\tilde{g}(\theta_1, 0) = -\cos\left(\pi \cdot x'(\theta_1)\right)$$
$$F_{\mathcal{K}}(\theta_1, 0) = 2\big(x'(\theta_1)\big)^2 - 1.$$

These are both even functions of $x'(\theta_1)$ that are equal to 1 at $x'(\theta_1) = \pm 1$ and are equal to -1 at $x'(\theta_1) = 0$.

4. 2D Images

4.1 Experiments and Results

We conduct several experiments on the image datasets described in Section 4.1.1. On individual datasets, we investigate the effect of Gaussian noise on training the TCNN (Section 4.1.2), the interpretability of TCNN activations (Section 4.1.3), and the learning rate of TCNNs in terms of testing accuracy over a number of batches in Section 4.1.4. Across different datasets, we investigate the generalization accuracy of TCNNs when trained on one dataset and tested on another in Section 4.1.5. We compare TCNNs to traditional CNNs in all of these domains. We also test other choices of Gabor filters versus Klein filters in the KF layers; see Section 4.1.6. While the improved generalizability and speed of our models are empirically evaluated, the improved interpretability is visually demonstrated. One may expect that statistics and visual tools could make empirical analysis of these TCNNs' interpretability possible with empiricism, but these fall outside of the scope of this paper. All of these experiments use one or two convolutional layers, followed by a 3-layer fully connected network terminating in 10 or 2 nodes, depending on whether the network classifies digits or cats and dogs. For additional details on the neural network models, metaparameters, and train/test splits, see Section 4.2.

4.1.1 Description of Data

We perform digit classification on 3 datasets: MNIST LeCun et al. (1998), SVHN Netzer et al. (2011) and USPS Hull (1994). These datasets are quite different from each other stylistically while each consists of images of digits 0 through 9. In particular, a human can easily identify to which dataset a particular image belongs. This makes generalization between the datasets a non-trivial task because neural networks that train on one of the datasets will in general overfit to the particular style and idiosyncracies present in the given data, which will be inconsistent with different-looking digits from the other datasets. The datasets come at significantly different resolutions: 28^2, 32^2, and 16^2, respectively. Additionally the sizes of the datasets vary widely: roughly 7e4, 5e4, and 7e3, respectively. SVHN digits are typeset whereas MNIST and USPS are handwritten. MNIST and USPS are direct 2-D impressions with significant pre-processing already done, while the SVHN numbers are natural images of typeset digits with tilts, warping, presence of secondary digits, and other irregularities.

We also use two collections of labeled images of cats and dogs: Cats vs Dogs Kaggle (2013) (which we call Kaggle), and the cat and dog labeled images from CIFAR-10, see Krizhevsky (2012). Note that the Kaggle Cats vs Dogs dataset, upon our download, included several images which were

empty/corrupt and could not be loaded, so the size reported here is the total number of loadable images. This seems to be typical for this dataset. These datasets contain 2.5e4 and 1.2e4 images, respectively. The resolutions of the images in each dataset are 50^2 and 32^2, respectively. Since we use these datasets to test the generalization accuracy of a neural network trained on one and tested on the other, we down-resolve the Kaggle data to 32^2 to be the same size as CIFAR-10. For completeness, we include Figure 16 to demonstrate that a TCNN improves the rate of learning for classification of all 10 CIFAR classes compared with a typical CNN.

Dataset	Size	Dimensions	Link
SVHN	60000	32^2	http://ufldl.stanford.edu/housenumbers
USPS	9298	16^2	https://web.stanford.edu/~hastie/StatLearnSparsity_files/DATA/zipcode.html
MNIST	70000	28^2	http://yann.lecun.com/exdb/mnist/
CIFAR-10	12000	32^2	https://www.cs.toronto.edu/~kriz/cifar.html
Kaggle	24946	64^2	https://www.kaggle.com/c/dogs-vs-cats

4.1.2 Synthetic experiments

The central hypothesis of this work is that constraining CNNs to use the Klein bottle filters (see KF, Definition 9) and to train with respect to the Klein bottle topology on slices (see KOL, Definition 8) will provide the model with highly meaningful local features right away. So, we expect a model trained on our CF, KF, COL, and KOL layers to outperform conventional CNNs when global noise is added to the images. To test this idea, we add class-correlated Gaussian noise $N(\mu_k, \sigma_k^2)$ to our images, $\mathbf{z}_k = \mathbf{x}_k + \mathcal{E}_k, \mathcal{E}_k \sim N(\mu_k, \sigma_k^2)$, where $\mathbf{x}_k$ is an image of class k and $\mathcal{E}_k$ is a vector of the same dimension as the image, drawn from a normal distribution. The hyperparameters μ_k and σ_k^2 are drawn from $N(.2, .04)$ and $\chi^2(1) \times .04$ respectively. We perform two experiments: (1) training on $\mathbf{z}$ and testing on $\mathbf{x}$ (see Column 1 of Figure 4), and (2) training on $\mathbf{x}$ and testing on $\mathbf{z}$ (see Column 2 of Figure 4). Also we offer an analysis at the end of this section while we vary the mean of μ_k and σ_k^2.

The results of this experiment on MNIST are displayed in Figure 4. A conventional two layer network's performance is greatly deteriorated by the Gaussian noise added to the images in training or testing. In the case of noise added to the training set, we expect the CNN to learn the class-correlated μ_k, σ_k^2, and then perform poorly on the test set without Gaussian noise. Note that Figure 4 includes model loss and it is clear that the 2-layer conventional CNN is classifying within the training set with high accuracy. We suspect that the TCNNs' superior performance in the noisy training set experiment is due to the fixed filters in the CF and KF layers which force a smoothing of the noise prior to the learned layers and hence generate classifiers invariant under the global noise. A similar smoothing argument explains the TCNNs' superior performance in the noisy test set experiment.

All four of the TCNNs outperform conventional CNNs on training and testing. Interestingly, while our CF based systems struggle to learn at first, after many epochs of training they are best at not fitting noise. Indeed, nothing interesting happens in the training curves after 1 epoch. All the accuracies stabilize after one epoch, and they all stabilize at approximately the same value. Also it is an interesting observation that the KOL and COL systems seem to learn slower but add robustness to avoid fitting noise in lengthy training. It is important to note that this class-correlated noise is a pathological example by design, and is meant primarily to show that the topological feature and convolution layers help to prevent fitting signals outside of the edges, angles, etc. that good models are expected to fit.

We now choose parameters τ and ω from 0 to .8 in increments of .2 and sample $\mu_k \sim N(\tau, .04)$ and $\sigma_k^2 \sim \chi^2(1) \times \omega^2$. In the first column of Figure 5, we simulate distributions $\mu_k \sim N(\tau, .04)$ and $\sigma_k^2 \sim \chi^2(1) \times .04$ testing the parameter τ from 0 to .8 in increments of .2. In the second column, we simulate distributions $\mu_k \sim N(.2, \omega^2)$ and $\sigma_k^2 \sim \chi^2(1) \times \omega^2$ testing ω from 0 to .8 in increments of .2.

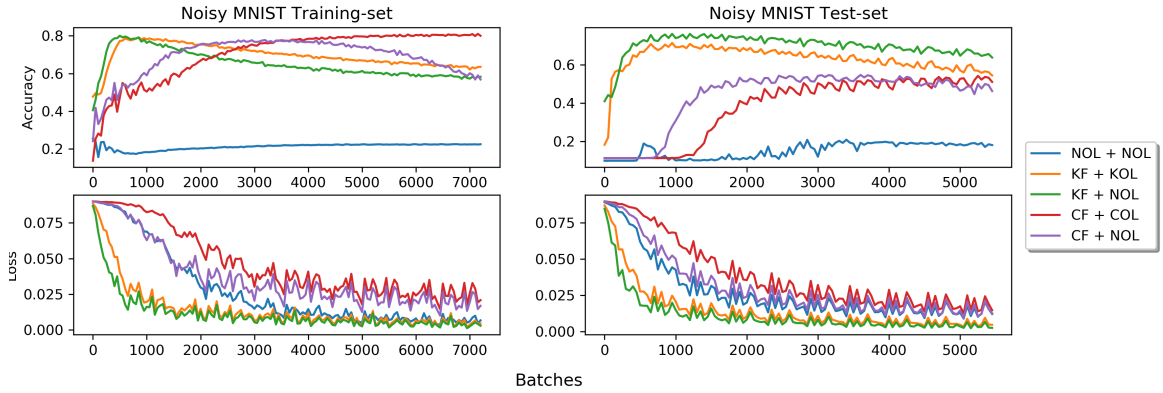


Figure 4: Two synthetic experiments on noisy MNIST data, designed to test model assumptions. The first column displays the results of the experiment where Gaussian noise is added to the training data but not the testing data. The second column displays the results of the experiment where the training data is the original MNIST data and the testing data are corrupted by Gaussian noise. The first row is testing accuracy and the second row is training loss.

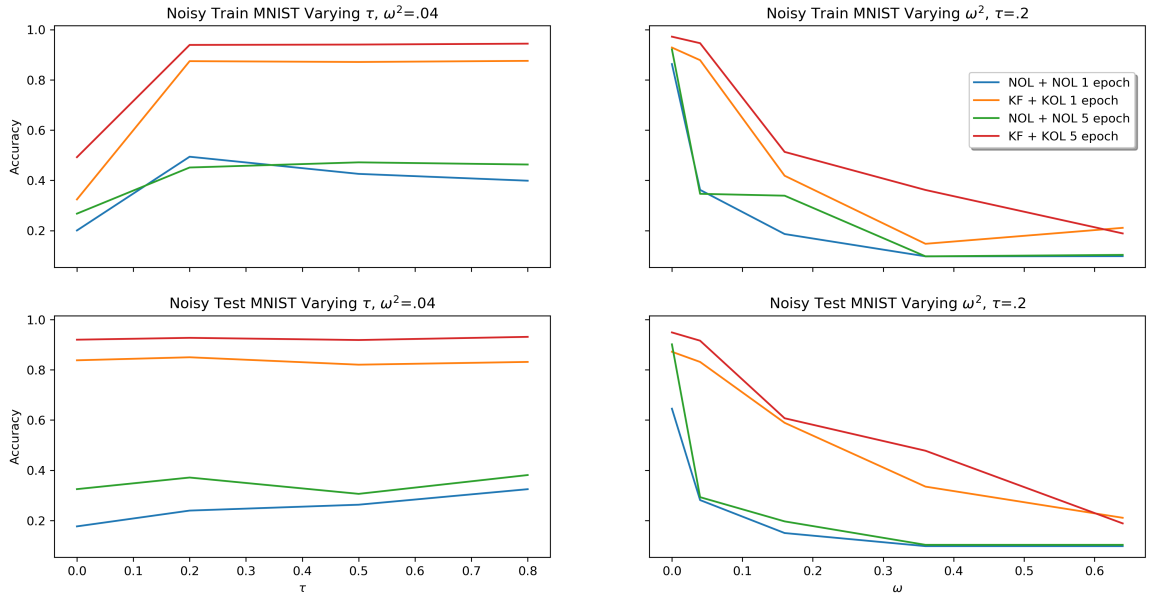


Figure 5: Sweep of distributions for synthetic MNIST data varying the mean and variance from which class-noise distributions are drawn. The first column shows the tested values of τ and the second column shows the tested values of ω^2. The first row shows accuracies when training on data with Gaussian noise and testing on unaltered test set. The second row shows accuracies when training on unaltered training data and testing on noise-added test data. We show results after 1 epoch and 5 epochs of training.

First we examine the top row of Figure 5 in which we train on data with Gaussian noise and test on the original data while varying τ and ω^2. When τ is 0, the noise added has little class−correlated signal (μ_k are small), but since ω^2 is still .04, there is an uncorrelated noise component. This noise deteriorates accuracy in all models. When τ grows, the TCNN trains significantly better than the normal CNN. When ω^2 is low, the models all perform similarly, and when ω^2 is large, all models perform poorly. However, there is a wide range of values of ω^2 for which the TCNN dramatically outperforms the CNN.

Next we examine the bottom row in which we train on the original MNIST training data and test on the Gaussian-noisy test data. Varying τ has little impact on the noisy test-set accuracy in the TCNN. As ω^2 grows in the noisy test-set experiment, it increases the random component of the noise, which degrades both classifiers as expected. Again, there is a significant portion of the range of ω^2 for which the TCNN is far superior to the CNN. These results are expected since we fix the convolutional weights in a sensible (topological) configuration in the KF layer. This forces the learned layers to classify on a smoothed version of the images which are more robust to the added noise.

4.1.3 Interpretability

We contrast the interpretability of CNNs and TCNNs by demonstrating the difference in activations between an NOL, CF, and KF (Figure 3) filtered image from MNIST. The activations shown in Figure 3 correspond to the output of the first layer of CF, KF and NOL models. The NOL filters are empirically derived by training on the MNIST training data (n = 6e10). The 16 CF filters correspond to 16 evenly spaced angles on a circle and the 16 KF filters correspond to 4 evenly spaced values for each of 2 angles on the Klein bottle.

Observing the activations in Figure 3, it is empirically clear that the activations of the CF and KF layers are easier to interpret, and can certainly be used to "probe" a specific classification. Indeed, in the case of CF, the filters are edges at various angles, and the activations show where each such orientation of an edge appears in the image. In the case of KF, filters containing interior lines are also included, and the activations reveal the presence of these interior lines in the image. The NOL learned filters and activations are, in comparison, difficult to interpret. While some of the filters seem to be finding edges, others have idiosyncratic patterns that make it difficult to assess how the combined output features contribute to a classification. The NOL model has an accuracy of approximately 99%, so further training on MNIST is unlikely to significantly alter these filters.

4.1.4 Rate of learning

We find significant jumps in accuracy in the KF + KOL and KF + COL networks compared with NOL + NOL, and, importantly, the networks with a KF layer achieve high accuracy earlier in training than those without, suggesting potential applications to smaller datasets for which less training can be done. See the bar chart in Figure 6 for a comparison of testing accuracy attained from training on only 1,000 images in each dataset. The SVHN dataset, which is much richer than MNIST, and the USPS dataset, which has much lower resolution than MNIST, received larger benefits from the use of a TCNN. This suggests that the benefit of the feature engineering in TCNNs is greatest when the local spatial priors are relatively weak or hidden.

4.1.5 Generalizability

We compare model generalizability between several models trained on either SVHN or MNIST and test on the other, and similarly across the Kaggle and CIFAR cats vs. dogs datasets. See Figures 7, and 8 for a comparison of testing accuracies throughout training. The TCNNs achieve decent generalization in both the digit and cat vs dog domains. Using the KF + KOL TCNN, 30% accuracy is achieved generalizing from MNIST to SVHN, and over 60% accuracy is achieved

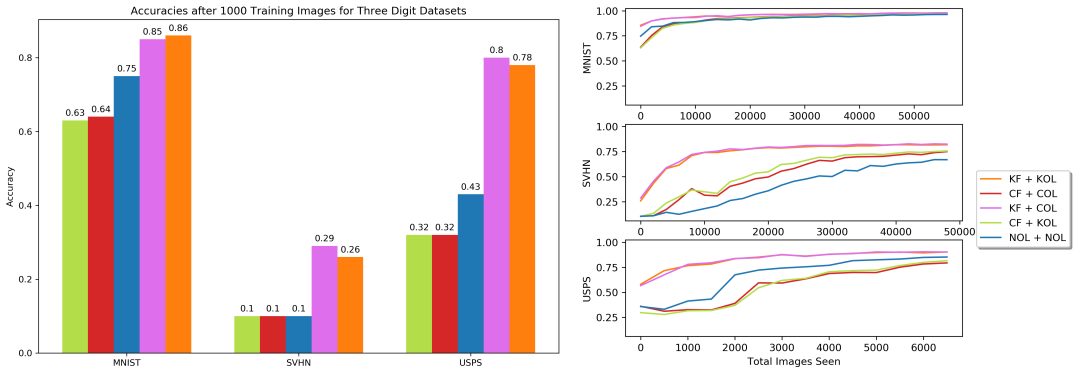


Figure 6: Left: Comparisons of testing accuracy after training on 1,000 images. Right: Full comparison of testing accuracy (y-axes) over a single epoch. MNIST was trained on 60,000, SVHN on 50,032, and USPS on 7291 images.

generalizing from SVHN to MNIST. Contrast this with the 10% generalization accuracy of the NOL + NOL conventional CNNs from MNIST to SVHN – the same as random guessing. Note that the addition of a pooling layer had negligible effect on these results. Similarly, TCNNs are better than CNNs at generalizing between the Kaggle and CIFAR cats vs. dogs datasets, however the difference is less dramatic than it is for digit classification. Here, the addition of pooling layers has no impact on the generalizability of CNNs, but provides further improvement on the generalizability of TCNNs.

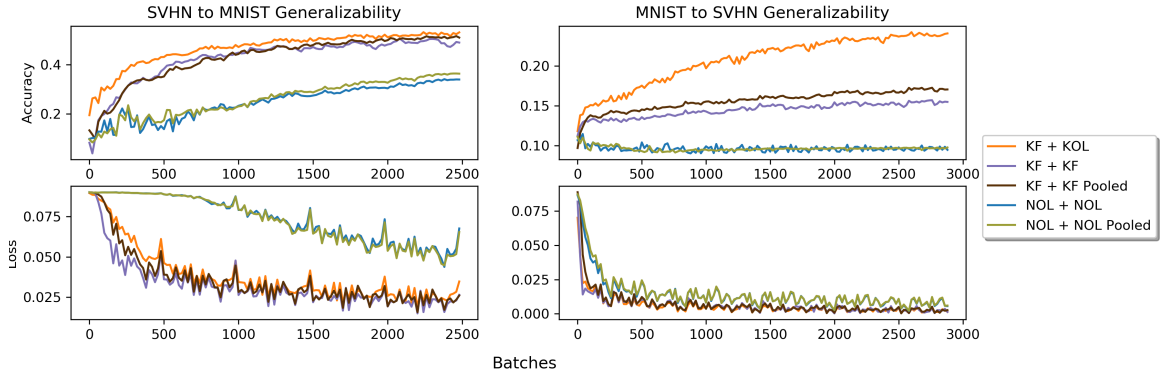


Figure 7: Comparisons of testing accuracy and validation loss when generalizing from SVHN to MNIST and vice versa.

4.1.6 Gabor filters versus Klein bottle filters

As described in Section 3.3, the Klein bottle filters given by the embedding $F_{\mathcal{K}}$ in (4) can be viewed roughly as a subset of the 5-parameter family of Gabor filters given by (9). Given any subset of Gabor filters, one can make a convolutional layer analogous to the KF layer (Definition 9) where the filters are initialized to the chosen Gabor filters and frozen throughout training. In this section, we compare a KF layer with a layer instantiated with another choice of Gabor filters, given by the parameters below in (10) and pictured in Figure 9. While the 2-parameter family of Klein bottle

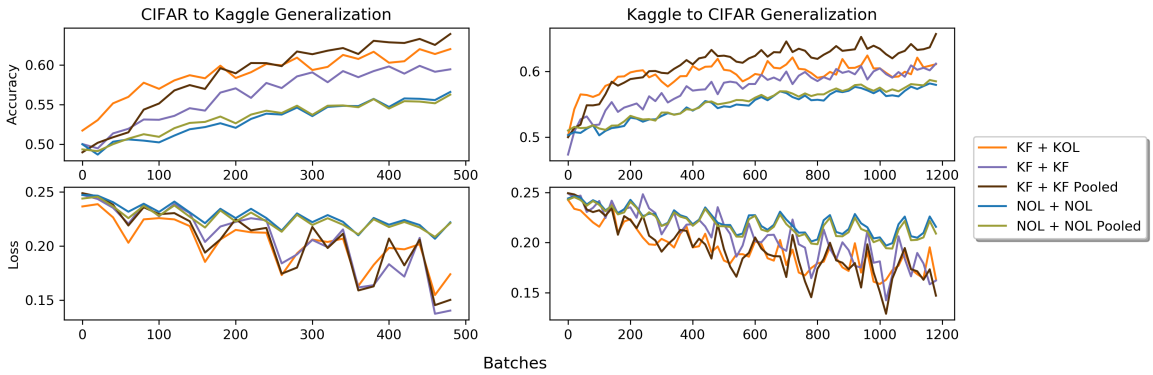


Figure 8: Comparisons of testing accuracy and validation loss when generalizing from CIFAR to Kaggle and vice versa.

filters is topologically justified as important for image analysis, we do not have any other means for selecting a different family of Gabor filters, and hence the 2-parameter family in (10) is necessarily somewhat arbitrary. Our heuristic motivation for this choice is that these are high contrast filters that are sparsely sampled from the two angles ψ and ω. The fixed parameters σ, λ, and γ are experimentally chosen so that the resulting 3×3 filters are high contrast. The variable parameter ω plays the same role of rotating the filter as the parameter θ_1 in the parameterization $F_{\mathcal{K}}$ of the Klein bottle filters.

$$\sigma = 2\pi,\ \lambda = \pi,\ \gamma = \frac{\pi}{8},\ \psi \in \left\{\frac{\pi}{4}, \frac{3\pi}{4}, \frac{5\pi}{4}, \frac{7\pi}{4}\right\},\ \omega \in \left\{\frac{\pi}{4}, \frac{3\pi}{4}, \frac{5\pi}{4}, \frac{7\pi}{4}\right\}. \tag{10}$$

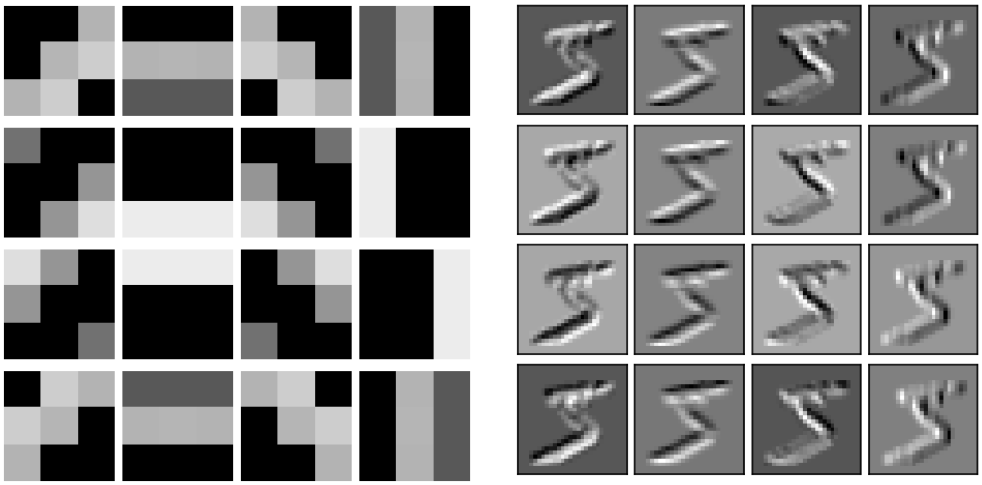


Figure 9: Example of a small set of Gabor filters and activations on a handwritten 5 from MNIST.

Using these specific Gabor filters we create a 'Gabor' convolutional layer in the same fashion as the KF topological layer except with these Gabor filters as weights instead of the Klein filters. We train a Gabor + NOL convolutional network on the datasets considered in this paper and compare

its accuracy to a KF + NOL network and a standard NOL + NOL network. See Figure 10. We find that the Klein bottle filters perform similarly or better than the Gabor filters. This suggests that the Klein bottle filters are a good choice of Gabor filters to use in the topological layers of a TCNN, as hypothesized. Note again that some highly restrictive choice of Gabor filters must be made for this type of construction since they come in a large 5-parameter family. Further, several of the parameters are difficult to interpret in the context of the typically small kernels of CNNs.

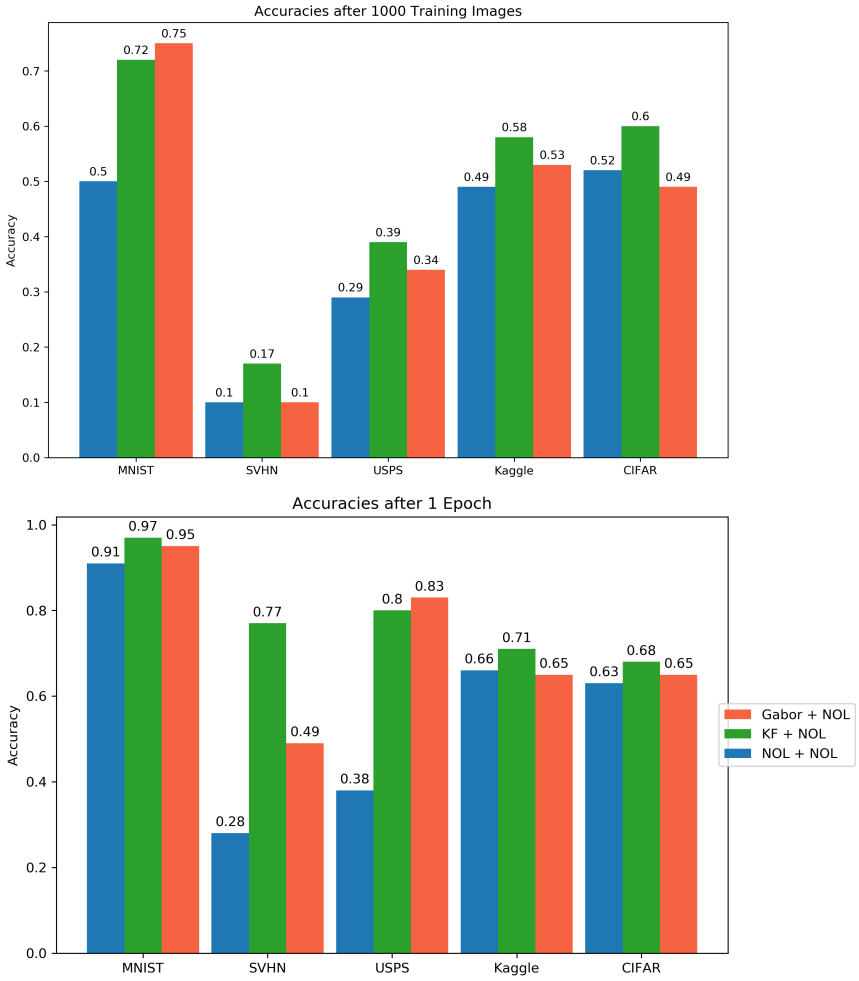


Figure 10: Top panel: testing accuracies on 3 digit datasets and 2 cats and dogs datasets after 100 training images. Bottom panel: testing accuracies after 1 full epoch of training. The KF model consistently performs similarly or better than the Gabor filters.

4.1.7 KOL sparsity compared with random sparsity

To validate the efficacy of the KOL layer, we compare a 2-layer KOL model to a randomly sparsified model. We run twenty independent trainings on all classes of CIFAR-10, testing throughout. Ten of these runs use a KF+KOL network and ten use a KF+Perm(KOL), where Perm() simply shuffles the connections of the KOL layer (i.e., the levels of sparsity are identical, but the connections in

the permuted models are random). Figure 11 shows that the permuted KOL layer is still able to learn, as expected (random sparsification of neural networks is common practice for various purposes including regularization), but the KOL models learn faster and are more reliable.

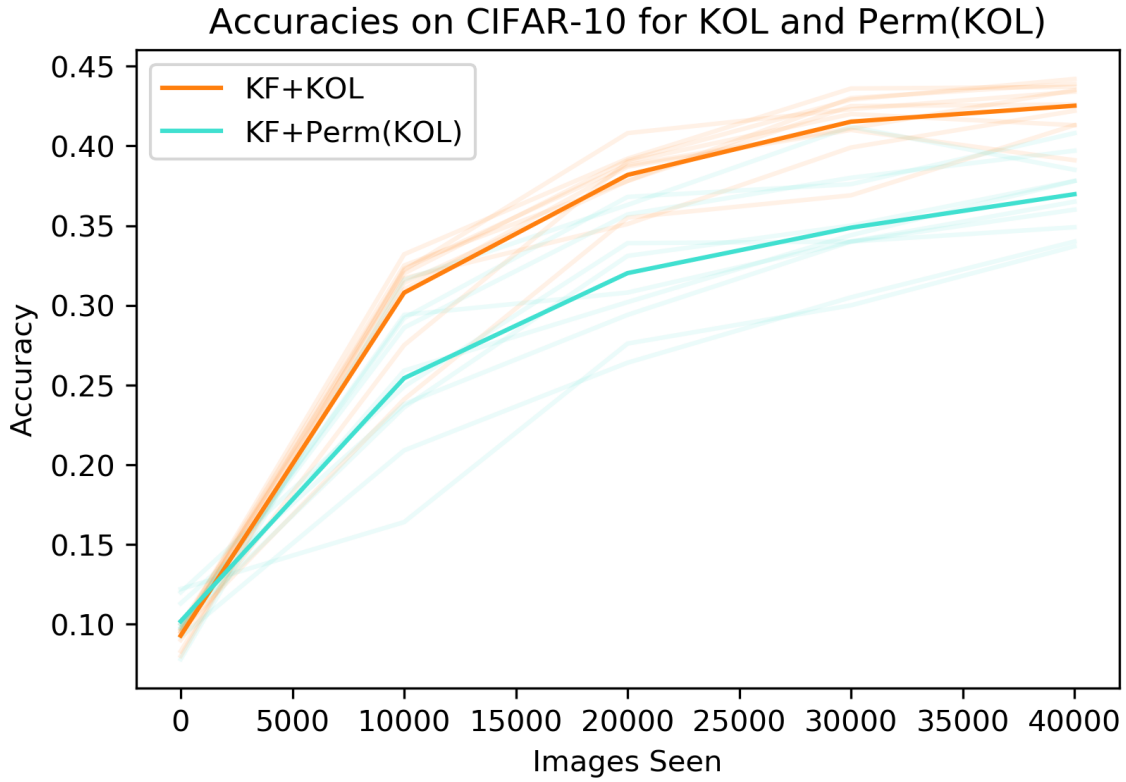


Figure 11: We provide an additional robustness test of the KOL layer, by comparing it with random permutations of the KOL with the same sparsity, i.e., we shuffle the connections of the KOL randomly in *Perm(KOL)*. We show 10 rounds of each network with the mean accuracy over training time of each in bold. Network sparsity can speed learning, but with much less reliable results than our KOL layer. The faded lines represent the accuracy of individual runs.

4.2 Details of methods

4.2.1 Train/test splits

With regard to train/test splits, we perform two different types of experiments: (1) accuracy of TCNNs and CNNs are compared on a fixed dataset, and (2) either a TCNN or a CNN is trained on one dataset and validated on a second dataset to measure the capacity of the model for generalization. For type (1), we split the dataset into training and testing sets. For type (2), we train and test on the entire datasets. For type (1), the train/test splits are as follows:

Dataset	Train	Test
MNIST	85%	15%
SVHN	80%	20%
USPS	80%	20%

For type (2), the dimension of the images is determined by the lowest resolution in the comparison, i.e., we down-resolve the higher resolution images to the lower resolution to be able to simply test generalizability.

4.2.2 Metaparameter selection

We choose metaparameters for each experiment that allow us to test TCNN performance against traditional CNN performance and do not seem to favor any particular model in the experiment with regard to the question at hand. Our strategy in comparing methods is to simply pick a relatively conventional, simple set of network specifications. We have selected an optimizer, batch size, learning rate, etc. on the criteria that all models are able to reasonably traverse the loss function on which they are optimized. We select a single configuration for all models and apply this consistently throughout the experiments. While of course the precise results fluctuate depending on these choices, our findings are generally consistent no matter the configurations we chose. We expect that, since our adaptations are within the typical CNN construction, meta-parameter selection has little effect on the relative impact of choosing a TCNN over a conventional CNN.

We provide the precise metaparameter specifications and some additional detail on our reasoning below. Within each Figure in the paper, the metaparameters are the same across all models presented. The following table lists various metaparameters by Figure number.

Figure	Conv-layers	Conv-slices	Kernel size	LR	Batch size	Epochs
3	1	16	5	1e−4	100	1
4 and 5	2	64	3	1e−5	100	5
6	2	64	3	1e−4	100	1
7 and 8	2	64	3	1e−5	100	5

Conv-layers. The number of convolutional layers is chosen to be 2 for all experiments outside of Figure 3. Using 2 convolutional layers allows us to incorporate a feature layer and a correspondence layer together, e.g. KF+KOL. We consistently use 2 convolutional layers for uniformity throughout the paper.

Conv-slices. The number of slices in each convolutional layer is chosen to be 64 for all experiments outside of Figure 3.

Kernel size. Figure 3 contains the only experiments in which we use a kernel size of 5. This is strictly for visualization purposes, as it allows the reader to see a more nuanced picture of the circle and Klein filters. All other experiments use a kernel size of 3 in each convolutional layer.

Learning rate (LR). An LR of 1e−5 or 1e−4 is used in each experiment. We simply choose the highest power of ten where none of the models exhibit pathological behavior.

Batch size. We use a batch size of 100 throughout all experiments.

Epochs. We employ 5 epochs of training for questions of generalization and 1 epoch for questions of training speed. We choose 5 epochs for questions of generalization because training over many epochs is common in applications and often results in greater potential for over-fitting. Over-fitting should reduce generalizability, hence we train long enough to allow for this possibility. 5 epochs allows us to compare the rate of over-fitting at different stages of training. In the cases regarding training speed, 1 epoch is sufficient to illustrate the comparisons since the primary interest is in the accuracy and loss over the first few batches. 1 epoch is used to create Figure 3 out of convenience.

Fully connected layers. All experiments use 2 fully connected layers following a flattening layer. The flattening layer simply flattens the outputs of the final convolutional layer into a 1D

vector. The first fully connected layer has 512 nodes, and the second has nodes of cardinality equal to the number of classes in the output.

4.2.3 Training

We processed images in 100 image batches. Computing was performed on an AMD 2990WX CPU with 128GB RAM and an NVIDIA RTX 2080TI GPU.

5. Video

5.1 Experiments and Results

We conduct several experiments on the datasets described in Section 5.1.1. On the UCF-101 dataset, we compare the learning rate of a TCNN in terms of testing accuracy over a number of batches in Section 5.1.2 and we compare it to the learning rate of a CNN. We also note that the TCNN achieves 70% accuracy after 100 epochs compared to the 55% accuracy of the CNN. We also investigate the generalization accuracy of TCNNs when trained on the KTH dataset and tested on the Weizmann data in Section 5.1.3.

5.1.1 Description of Data

We use three datasets of videos: UCF-101 Soomro et al. (November, 2012), KTH Schuldt et al. (2004), and Weizmann Gorelick et al. (2007). The UCF-101 dataset consists of 13320 videos in 101 classes of human actions, such as Baby Crawling, Playing Cello, Tennis Swing, etc. There are a large variety of camera motions, object scales and viewpoint, background, and lighting conditions. The videos are between 2 and 15 seconds long. The KTH dataset consists of 2391 videos of 6 types of human actions (walking, jogging, running, boxing, hand waving, and clapping). Each video is of one of 25 different human subjects in one of a few different settings, e.g. outdoors and indoors, for an average length of 4 seconds. This is a significantly simpler dataset than UCF-101 due to the controlled nature of the videos, e.g., backgrounds are homogeneous and the human actor is the only moving object.

The Weizmann dataset is used to test of generalization of a network trained on the KTH dataset. Only 3 Weizmann classes have a perfect analog in KTH: hand waving, running and walking. We test generalization on these 3 classes. Example frames from Weizmann and KTH, where their classes intersect, are provided in Figure 12. There are 29 Weizmann videos in the three classes. Both KTH and Weizmann have variable image lengths and KTH has half the frame rate of Weizmann. We choose to take about 1 second of each video at 25fps on which to perform classification.

Dataset	Size	Dimensions	Link
UCF-101	13320	320x240x25fps	https://www.crcv.ucf.edu/data/UCF101.php
KTH	2391	160x120x25fps	https://www.csc.kth.se/cvap/actions
Weizmann	29	180x144x50fps	http://www.wisdom.weizmann.ac.il/~vision/SpaceTimeActions.html

5.1.2 Accuracy and rate of learning on a ResNet

We train 12 layer ResNet classifiers as in He et al. (2016) on the UCF-101 dataset. One of these classifiers is a conventional CNN whose convolutional layers operate on all 3 dimensions of the video. The other classifier is a TCNN that is identical to the CNN except for the first layer in which we use a M-F layer (Definition 12) instead of a NOL. The M-F layer is of type $M = \widetilde{\mathcal{K}} \cup S^{\pm}_{\tau} \cup S^{\pm}_{\rho}$ (see (8)), that is, we use the features given by translating the Klein bottle in time, rotating the Klein bottle in time, and holding the Klein bottle still in time – see the discussion above Definition 12.

Our main result is that, when using the M-F layer as the first layer instead of NOL, we find a significant increase in final accuracy as well as a much faster learning rate in the initial epochs; see

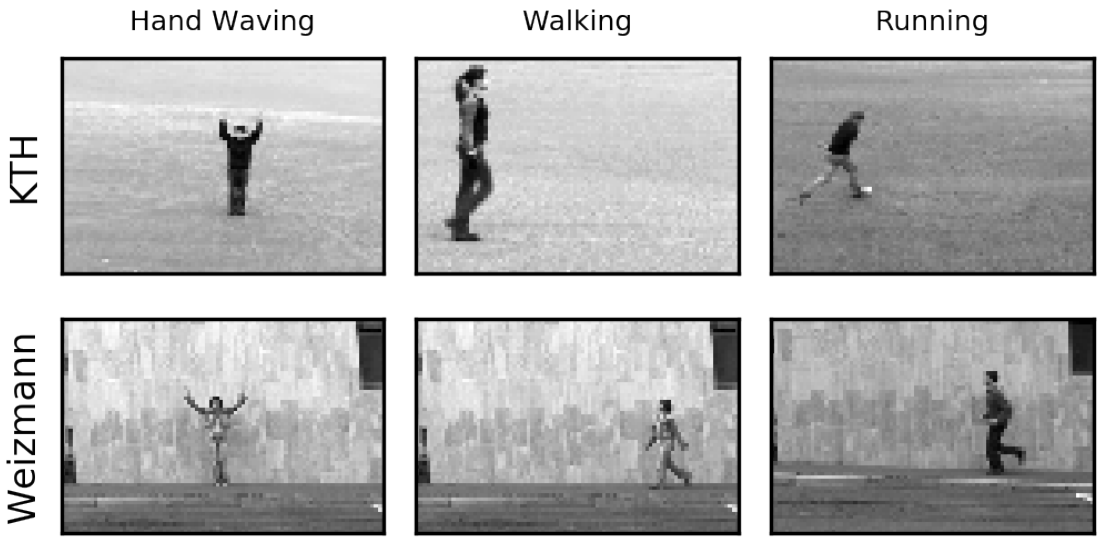


Figure 12: Examples of one frame from KTH (top row) and Weizmann (bottom row) videos from the same classes (columns).

Figure 13. In Figure 14, we plot the best class accuracy, best five, and best ten, as a function of training epochs.

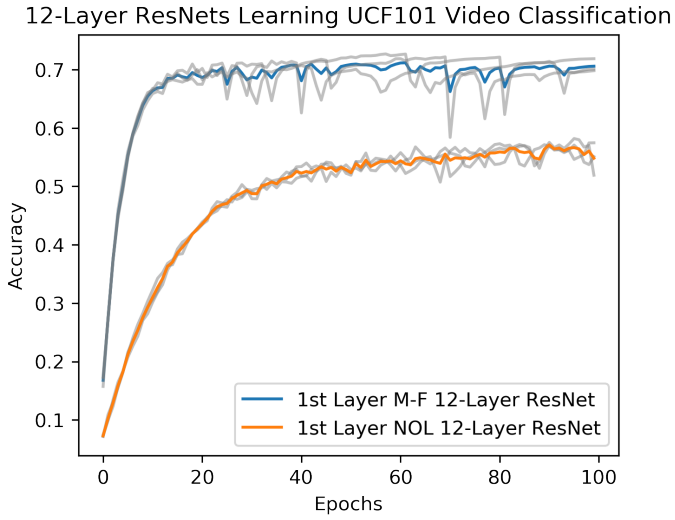


Figure 13: Testing accuracy of a 12 layer ResNet trained on UCF-101.

We also perform the same experiment on the KTH dataset. The results are given in Figure 15. Again, the TCNN achieves high accuracy more quickly and remains higher throughout all 200 epochs

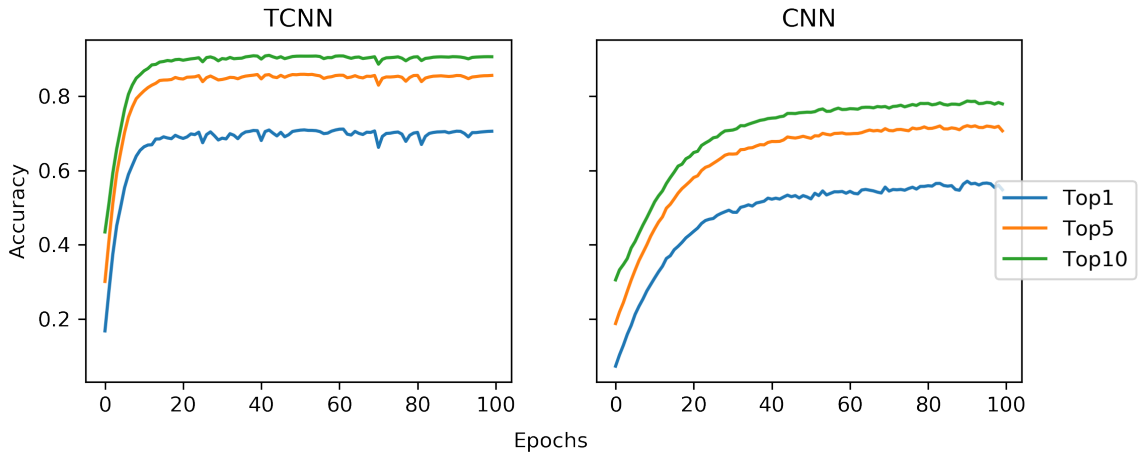


Figure 14: Top 1, 5, and 10 testing overall classification accuracies as a function of epochs for both the TCNN ResNet (left) and the CNN ResNet (right).

of training. The difference in final accuracy between the TCNN and the CNN is not as dramatic as we observe on the UCF-101 dataset, because the KTH dataset experiences less variation between videos, so it is easier for the CNN to learn meaningful features without the help of the M-F layer.

5.1.3 Generalizability

We compare generalizability of a TCNN and a CNN model trained on the KTH dataset and tested on the Weizmann dataset. Both of these models are ResNets, and the TCNN is the same as the CNN except with the first layer a M-F layer instead of a NOL layer, as described in Section 5.1.2. The TCNN generalizes from KTH to Weizmann with a testing accuracy of about 65%, whereas the CNN achieves only about 52% generalization accuracy.

The full results are shown in Figure 15. Observe that the generalization accuracy of the CNN takes a large dive at around 50 epochs, which suggests that it is initially learning features specific to the KTH dataset that do not generalize well, and then as it continues to train it learns more meaningful features that do generalize. We conjecture that the dip in accuracy on the Weizmann dataset occurs during the transition from learning KTH-specific artifacts to learning meaningful features. On the other hand, the TCNN immediately begins learning meaningful features that generalize well. This is the desired effect of the M-F layer – it biases the learner to use significant features in the video right away, instead of initially learning artifacts particular to the dataset.

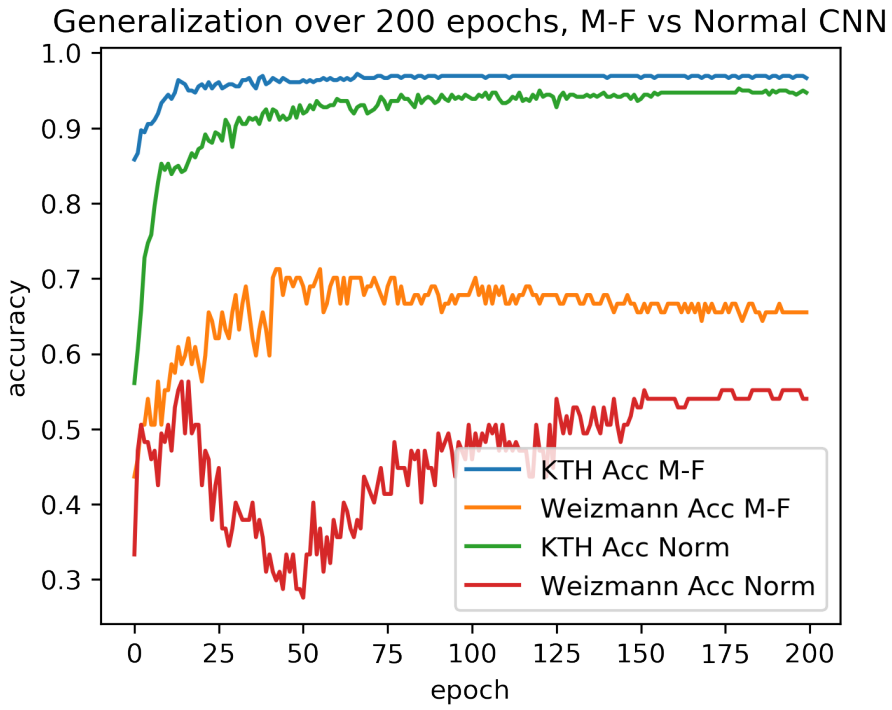


Figure 15: Testing accuracy of a TCNN and a CNN trained on the KTH dataset and tested on the Weizmann dataset.

5.2 Details of methods

5.2.1 Train/test splits

In our video experiments, we only have a train/test split for UCF-101. KTH and Weizmann videos were only used in the test of generalization, so for these datasets we use the entire set of KTH in training and the entire set of Weizmann to test (in the three common classes). The UCF-101 split is as follows:

Dataset	Train	Test
UCF-101	67%	33%

5.2.2 Metaparameter selection

Figure	Conv-layers	Conv-slices	Kernel size	LR	Batch size	Epochs
15	3	$20, 80, 160$	$(5 \times 11 \times 11)$, $(5 \times 11 \times 11)$, $(3 \times 9 \times 4)$	3e−3	100	200
13 & 14	12	$180 \times 9, 360 \times 3$	$(5, 11, 11)$, $(1, 5, 5)$, $(1, 3, 3)$, $(3, 3, 3)$	1e−5	100	5

Our video experiments differ from image experiments in that we use kernels which are not square (cubic). To indicate the size of a 3-dimensional kernel, we use the notation (t, y, x), where t is the size in the time dimension, and x, y are spatial sizes in their respective dimensions.

Notably, the ResNet12 residual network, which produces our highest accuracy classifiers for UCF-101 (Figures 13 and 14), has a significantly different form from all other models in this work, both in depth and in the use of a residual block. We choose a simple block architecture:

Block(ν, x):
$y = \text{Convolution}(\nu)(x)$
$y = \text{Convolution}(\nu)(y)$
$y = \text{Batch}(y)$
$y = \text{Upsample}(y)$
$x + y$

Table 1: Residual block for our 12-convolutional layer ResNet where x is an input tensor, data passes from top to bottom, so $x + y$ is the returned tensor.

Each block has two convolutional layers with a kernel size ν. In order to add the input to the output, we need tensors of matching dimension, so we use Upsample$(\cdot)$ which is a trilinear upsampler and Batch$(\cdot)$ which performs batch normalization. We indicate a 3-dimensional pooling layer by Pool$(\cdot)$. Our residual network then has the following structure:

ν	ResNet12(x):
	$y = M - F_1(x)$
(1, 5, 5)	y=Block(ν, y)
(1, 3, 3)	y=Block(ν, y)
(3, 3, 3)	y=Pool $(C_d(\nu)(y))$
	y=Batch(y)
(3, 3, 3)	y=$C_d(\nu)(y)$
	y=Batch(y)
(3, 3, 3)	y=Pool $(C_d(\nu)(y))$
	y=Batch(y)
	y=FC(y)
	y=Batch(y)
	$y = \text{softmax}(y)$

Table 2: Structure of our 12-convolutional layer ResNet with order of operations indicated by descending through the table. The first column of the table indicates the kernel size of a convolutional layer or layers in a residual block.

5.2.3 Training

We processed videos in 50 (UCF-101) or 100 (KTH vs Weizmann) image batches. Computing was performed on an AMD 2990WX CPU with 128GB RAM and an NVIDIA RTX 2080TI GPU.

Acknowledgments

Research has been partially supported by the Army Research Office (ARO) Grant # W911NF-17-1-0313 and Grant # W911NF2110094 (VM), the Army Research Lab Grant # W911NF212018, and the National Science Foundation (NSF) Grants # MCB-1715794 (EL), DMS-1821241, DMS-2012609 (VM), DMS-1903023 (BF). GC was supported by Altor Equity Partners AB through Unbox AI (www.unboxai.org). We would like to acknowledge Rickard Brüel Gabrielsson and Michael McCabe for helpful conversations. All authors thank the anonymous reviewers for their comments, which improved our manuscript.

Appendix A. Additional Figures

In Figure 16, we compare a TCNN to a conventional CNN on classification of all ten classes in CIFAR-10, expanding on the generalization experiment in Section 4.1.5 that only uses two classes, i.e. cats and dogs. In Figure 17, we display trained filters in a COL, complementing the displayed filters in CF and KF layers depicted in Figure 3.

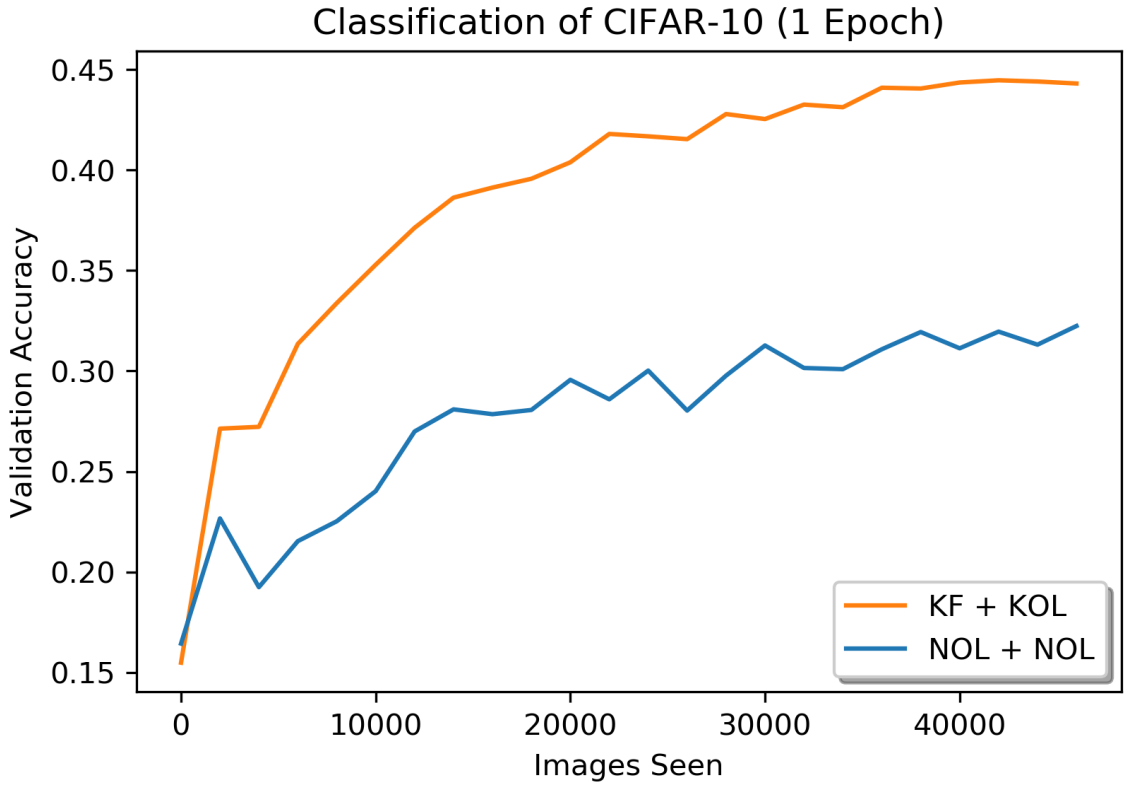


Figure 16: Comparing learning rate of 2-layer TCNN (KOL+KF) and CNN (NOL+NOL) classification of CIFAR-10 (all classes and standard train/test split). To ensure that our tests on the dogs vs. cats subset of CIFAR is not anomalous, we provide this additional experiment comparing the performance of a TCNN and CNN on the entire CIFAR-10 set.

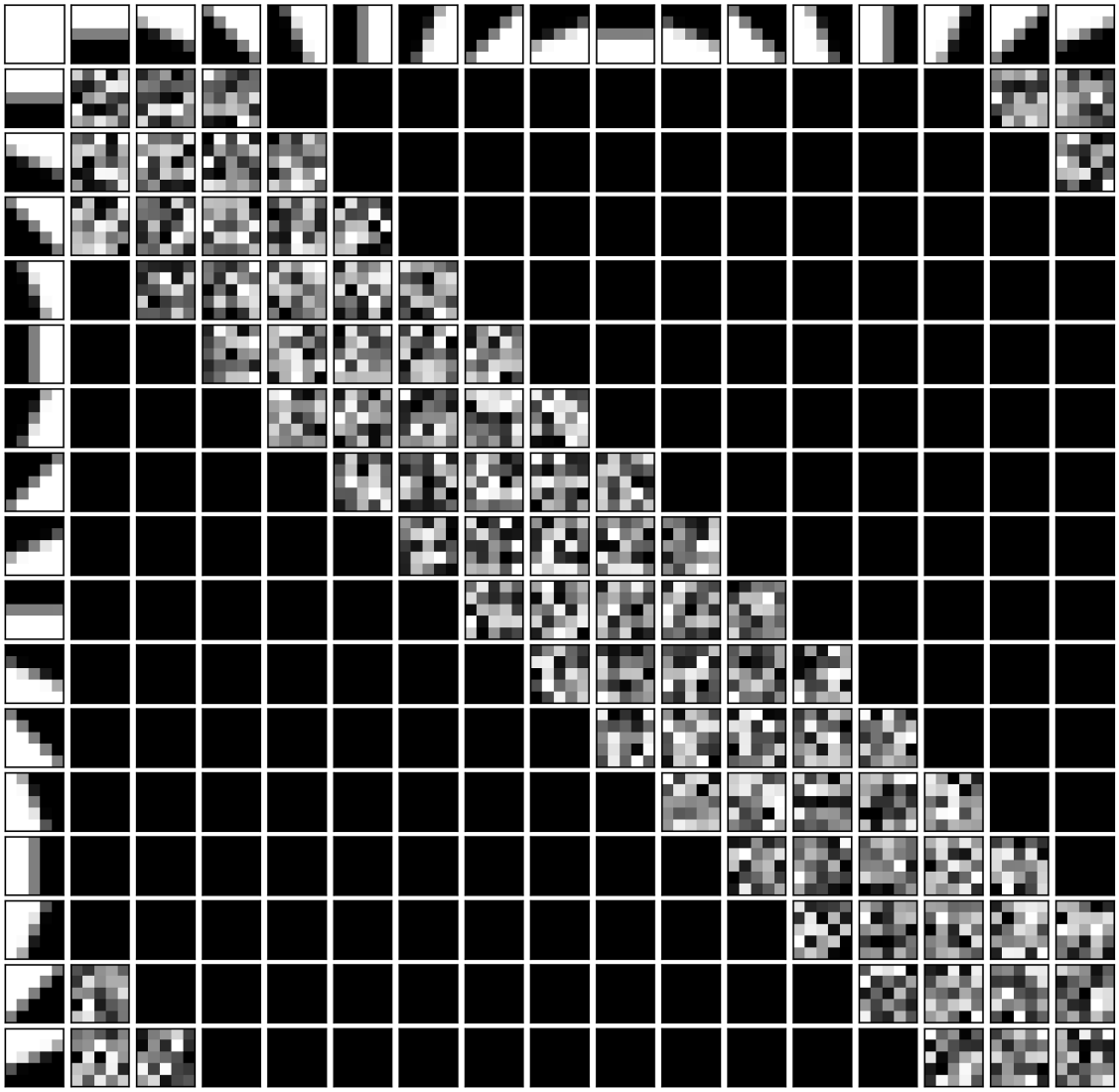


Figure 17: The trained weights in a COL. The top row indicates an input slice by the filter corresponding to a point on the circle. The left column indicates an output slice in the same way. For each input slice and output slice, the corresponding trained filter in a COL is displayed.

Appendix B. Pseudocode

Algorithm 1 summarizes how to instantiate a TCNN with a CF layer (Definition 9) as the first layer. Algorithm 2 summarizes how to instantiate a TCNN with CF as the first layer and COL (Definition 7) as the second layer.

Algorithm 1: Initialization of a CF (Circle Filters) convolutional layer with one input channel

- Choose the following fixed parameters:
  $\eta :=$ number of output channels of the convolutional layer = number of filters,
  $s :=$ distance in $\mathbb{Z}^2$ that determines the filter width (as in Definition 4).
- Initialize a single convolutional layer (Defintion 4) with convolutional correspondence

$$\mathbb{Z}^2 \xrightarrow{C_c \times C_{d,2}(s)} \mathcal{X} \times \mathbb{Z}^2,$$

  where $\mathcal{X}$ is the set of η-th roots of unity $\{e^{2\pi i \frac{j}{\eta}} \mid 0 \leq j \leq \eta - 1\}$. For each output channel indexed by $0 \leq j \leq \eta - 1$, there is a 2D convolutional filter Filter(j) of size $(2s+1) \times (2s+1)$.
- Initialize the filters to be the Circle Filters, as described in Eqs. (4) and (5) and Definition 9:
  for $0 \leq j \leq \eta - 1$ **do**
    $\theta = 2\pi \frac{j}{\eta}$
    for $0 \leq n, m \leq 2s$ **do**
      $\text{Filter}(j)(n,m) := \int_{-1+\frac{2m}{2s+1}}^{-1+\frac{2(m+1)}{2s+1}} \int_{-1+\frac{2n}{2s+1}}^{-1+\frac{2(n+1)}{2s+1}} F_{S^1}(\theta)(x,y)\, dx\, dy.$

Result: A standard convolutional layer with 1 input channel and η output channels with weights initialized to be the circle filters, giving the layer the structure of a CF layer. Note that in a CF layer, the filters are frozen during training, i.e. gradients with respect to these weights are not computed and the weights are not changed.

Algorithm 2: Construction of two consecutive convolutional layers, CF (Circle Filters) followed by COL (Circle One Layer), with one input channel

- Choose the following fixed parameters:
  $\eta :=$ number of output channels,
  $s :=$ distance in $\mathbb{Z}^2$ that determines the filter width (see Definition 4).
  $s' :=$ threshold for distance on the circle in the circle correspondence (see Definition 7).
- Initialize two consecutive convolutional layers with correspondences

$$\mathbb{Z}^2 \xrightarrow{C_c \times C_{d,2}(s)} \mathcal{X} \times \mathbb{Z}^2 \xrightarrow{C_S(s') \times C_{d,2}(s)} \mathcal{X} \times \mathbb{Z}^2,$$

  where $\mathcal{X}$ is the set of η-th roots of unity $\{e^{2\pi i \frac{j}{\eta}} \mid 0 \leq j \leq \eta - 1\}$.

  Note that the second layer is a COL as defined in Definition 7 by the nature of the correspondence $C_S(s') \times C_{d,2}(s)$.
- *Initialize the first layer to be a CF layer using Algorithm 1.*

Result: Two consecutive convolutional layers with 1 input channel and η output channels. The first layer is initialized to be a CF layer and its filters are frozen during training. In the second layer, the correspondence $C_S(s') \times C_{d,2}(s)$ makes it a COL.

References

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.

Robert J. Adler, Sarit Agami, and Pratyush Pranav. Modeling and replicating statistical topology and evidence for cmb nonhomogeneity. *Proceedings of the National Academy of Sciences*, 114(45): 11878–11883, 2017. doi: 10.1073/pnas.1706885114.

Erik J. Amézquita, Michelle Y. Quigley, Tim Ophelders, Elizabeth Munch, and Daniel H. Chitwood. The shape of things to come: Topological data analysis and biology, from molecules to organisms. *Developmental Dynamics*, 249(7):816–833, 2020.

Peter Bubenik. Statistical topological data analysis using persistence landscapes. *The Journal of Machine Learning Research*, 16(1):77–102, 2015a. ISSN 1532-4435.

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

Gunnar Carlsson and Rickard Brüel Gabrielsson. Topological approaches to deep learning. In *Topological Data Analysis*, pages 119–146, Cham, 2020. Springer International Publishing. ISBN 978-3-030-43408-3.

Gunnar Carlsson, Tigran Ishkhanov, Vin de Silva, and Afra Zomorodian. On the Local Behavior of Spaces of Natural Images. *International Journal of Computer Vision*, 76(1):1–12, January 2008. ISSN 0920-5691, 1573-1405. doi: 10.1007/s11263-007-0056-x. URL `http://link.springer.com/10.1007/s11263-007-0056-x`.

Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6299–6308, 2017.

Frédéric Chazal, Brittany Fasy, Fabrizio Lecci, Bertrand Michel, Alessandro Rinaldo, Alessandro Rinaldo, and Larry Wasserman. Robust topological inference: Distance to a measure and kernel distance. *The Journal of Machine Learning Research*, 18(1):5845–5884, 2017.

Lena Gorelick, Moshe Blank, Eli Shechtman, Michal Irani, and Ronen Basri. Actions as space-time shapes. *Transactions on Pattern Analysis and Machine Intelligence*, 29(12):2247–2253, December 2007.

Yanming Guo, Yu Liu, Ard Oerlemans, Songyang Lao, Song Wu, and Michael S Lew. Deep learning for visual understanding: A review. *Neurocomputing*, 187:27–48, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Felix Hensel, Michael Moor, and Bastian Rieck. A survey of topological machine learning methods. *Frontiers in Artificial Intelligence*, 4, 2021. doi: 10.3389/frai.2021.681108.

C. Hofer, R. Kwitt, M. Dixit, and M. Niethammer. Connectivity-optimized representation learning via persistent homology. In *ICML*, 2019.

Christoph Hofer, Roland Kwitt, Marc Niethammer, and Andreas Uhl. Deep learning with topological signatures. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/883e881bb4d22a7add958f2d6b052c9f-Paper.pdf.

D. H. Hubel and T. N. Weisel. Receptive fields of single neurons in the cat's striate cortex. *The Journal of Physiology*, 148(3):574–591, 1959.

Jonathan J. Hull. A database for handwritten text recognition research. *IEEE Transactions on pattern analysis and machine intelligence*, 16(5):550–554, 1994.

Kaggle. Dogs vs. cats. https://www.kaggle.com/c/dogs-vs-cats/overview, 2013. Accessed: 2019-10-01.

M Kalfaoglu, Sinan Kalkan, and A Aydin Alatan. Late temporal modeling in 3d cnn architectures with bert for action recognition. *arXiv preprint arXiv:2008.01232*, 2020.

Alex Krizhevsky. Learning multiple layers of features from tiny images. *University of Toronto*, 05 2012.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Andrew Marchese, Vasileios Maroulas, and Josh Mike. $k-$means clustering on the space of persistence diagrams. volume 10394, page 103940W. International Society for Optics and Photonics, 2017.

Vasileios Maroulas, Joshua L Mike, and Christopher Oballe. Nonparametric estimation of probability density functions of random persistence diagrams. *Journal of Machine Learning Research*, 20(151): 1–49, 2019. URL http://jmlr.org/papers/v20/18-618.html.

Vasileios Maroulas, Farzana Nasrin, and Christopher Oballe. A bayesian framework for persistent homology. *SIAM Journal on Mathematics of Data Science*, 2(1):48–74, 2020.

Yuriy Mileyko, Sayan Mukherjee, and John Harer. Probability measures on the space of persistence diagrams. *Inverse Problems*, 27(12):124007, nov 2011. doi: 10.1088/0266-5611/27/12/124007. URL https://doi.org/10.1088%2F0266-5611%2F27%2F12%2F124007.

Edward C. Mitchell, Brittany Story, David Boothe, Piotr J. Franaszczuk, and Vasileios Maroulas. A topological deep learning framework for neural spike decoding. 2022. URL https://arxiv.org/abs/2212.05037.

Michael Moor, Max Horn, Bastian Rieck, and Karsten Borgwardt. Topological autoencoders. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 7045–7054. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/moor20a.html.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

Christopher Oballe, David Boothe, Piotr Franaszczuk, and Vasileios Maroulas. Tofu: Topology functional units for deep learning. *Foundations of Data Science*, 4(4):641–665, 2022.

Theodore Papamarkou, Farzana Nasrin, Austin Lawson, Na Gong, Orlando Rios, and Vasileios Maroulas. A random persistence diagram generator. *Statistics and Computing*, 32(5):88, 2022. doi: 10.1007/s11222-022-10141-y. URL `https://doi.org/10.1007/s11222-022-10141-y`.

Zhaofan Qiu, Ting Yao, Chong-Wah Ngo, Xinmei Tian, and Tao Mei. Learning spatio-temporal representation with local and global diffusion. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 12056–12065, 2019.

Waseem Rawat and Zenghui Wang. Deep convolutional neural networks for image classification: A comprehensive review. *Neural computation*, 29(9):2352–2449, 2017.

M. Robinson. *Topological Signal Processing*. Springer, 2014.

Christian Schuldt, Ivan Laptev, and Barbara Caputo. Recognizing Human Actions: A Local SVM Approach. In *Proceedings of the 17th International Conference on Pattern Recognition, ICPR*, volume 3, pages 32–36, 2004.

Ioannis Sgouralis, Andreas Nebenführ, and Vasileios Maroulas. A Bayesian topological framework for the identification and reconstruction of subcellular motion. *SIAM Journal on Imaging Sciences*, 10(2):871–899, 2017. ISSN 1936-4954.

G. Singh, F. Mémoli, and G. Carlsson. Topological Methods for the Analysis of High Dimensional Data Sets and 3D Object Recognition. In *Eurographics Symposium on Point-Based Graphics*, 2007.

Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. UCF101: A Dataset of 101 Human Action Classes From Videos in The Wild. *CRCV-TR-12-01*, November, 2012.

Qi Zhao, Ze Ye, Chao Chen, and Yusu Wang. Persistence enhanced graph neural network. In Silvia Chiappa and Roberto Calandra, editors, *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pages 2896–2906. PMLR, 26–28 Aug 2020. URL `https://proceedings.mlr.press/v108/zhao20d.html`.

Qinghe Zheng, Mingqiang Yang, Jiajie Yang, Qingrui Zhang, and Xinxin Zhang. Improvement of generalization ability of deep cnn via implicit regularization in two-stage training process. *IEEE Access*, 6:15844–15869, 2018.