Topological Convolutional Neural Networks

Ephy R. Love
Bredesen Center DSE
University of Tennessee
Knoxville, TN 37996, USA
elove4@vols.utk.edu

Benjamin Filippenko
Department of Mathematics
Stanford University
Stanford, CA 94305, USA
benfilip@stanford.edu

Vasileios Maroulas
Department of Mathematics
University of Tennessee
Knoxville, TN 37996, USA
vmaroula@utk.edu

Gunnar Carlsson
Department of Mathematics
Stanford University
Stanford, CA 94305, USA
gunnar@math.stanford.edu

Abstract

There is considerable interest in making convolutional neural networks (CNNs) that learn on less data, are better at generalizing, and are more easily interpreted. 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.

1 Introduction

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 [3]. A crucial NN component used in image classification is the convolutional neural network (CNN) [4, 12, 9].

Despite the substantial parameter space of a deep CNN, it has been shown through topological clustering that, in aggregate, CNNs arrive at spatially meaningful configurations in the weights of their convolutional layers [1]. This is explained in part by [2] which showed that small patches in natural images cluster around a Klein bottle embedded in the space of patches.

This work demonstrates a framework to leverage the empirically found topology of trained convolutional filters and theoretical knowledge of the topology of natural images. We introduce two new types of convolutional layers which construct topological features and restrict convolutions based on embeddings of topological manifolds into the space of images. These new convolutional layers form a new CNN framework which we call the Topological CNN (TCNN). TCNNs are inherently easier to interpret than CNNs, as the local image convolutions have well-defined spatial interpretations. We show synthetic and empirical results which indicate that TCNNs have better generalizability and are faster to train than traditional CNNs. These new topological layers can be directly used with other techniques common to convolutional neural networks, such as pooling and holdouts. Generally, all traditional tools used with CNN models are still available and can be useful when applied to TCNNs.

In future work, we will extend our methods to new convolutional layers for video data. The methods are also extensible to other data modalities, such as 3D imaging, 3D video, and 1D signals such as time series.

Topological Data Analysis and Beyond Workshop at the 34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

2 Methodology

In a CNN operating on 2D data, every layer consists of a set of 2D grids called slices. In a standard convolutional layer, which we call a **Normal One Layer (NOL)**, weights are localized with respect to the L^∞ metric by fixing all weights connecting distant pixels to be zero throughout training. Two pixels are 'distant' if their L^∞ distance is larger than an a priori fixed threshold. All other weights are randomly initialized and then trained using batched gradient descent.

The TCNN introduces two new types of convolutional layers. The first type, called the **Circle One Layer (COL)** or **Klein One Layer (KOL)**, parameterizes the sets of input and output slices by a discretization of either the circle or Klein bottle and localizes weights with respect to a metric on the Klein bottle by fixing all weights between distant slices to zero. All other weights are randomly initialized and trained as usual. This significantly reduces the number of trained weights compared to a standard NOL layer. The second type of layer, called the **Circle Filters Layer (CF)** or **Klein Filters Layer (KF)**, comes instantiated with fixed weights that do not change during training. These weights are given by the embeddings of the primary circle (2) and Klein bottle (1) into the space of image patches discovered in [2]. All of these layers (COL, KOL, CF, and KF) can be viewed as a form of regularization applied to a NOL. The image patches parameterized by the Klein bottle can be viewed as a special 2-parameter family of Gabor filters. Gabor filters have been shown to be useful in decoding imagery processed in the brain [7].

More precisely, these new convolutional layers are described as follows. The Klein bottle $\mathcal{K}$ 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 $(\theta_1, \theta_2) \sim (\theta_1 + \pi, -\theta_2)$. There is an embedding of $\mathcal{K}$ into the space of functions on the square $[-1, 1]^2$ given by

$$F_{\mathcal{K}}(\theta_1, \theta_2)(x, y) = \sin(\theta_2)(\cos(\theta_1)x + \sin(\theta_1)y) + \cos(\theta_2)Q(\cos(\theta_1)x + \sin(\theta_1)y), \quad (1)$$

where $Q(t) = 2t^2 - 1$. This restricts to an embedding on the 'primary circle' $S^1 = \{(\theta, \pi/2) \in \mathcal{K})\}$ given by

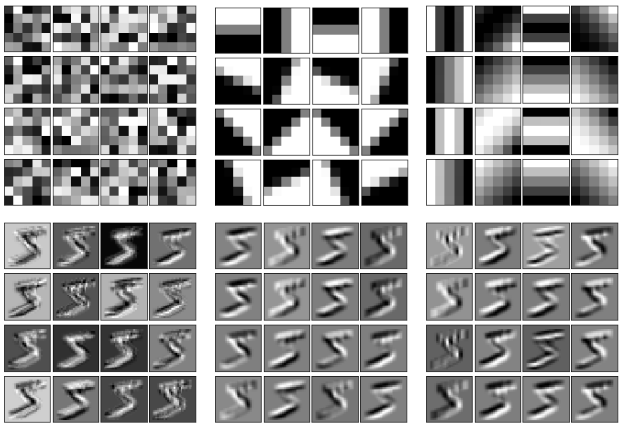
$$F_{S^1}(\theta)(x, y) := F_{\mathcal{K}}(\theta, \pi/2)(x, y) = \cos(\theta)x + \sin(\theta)y. \quad (2)$$

Both $\mathcal{K}$ and S^1 can be discretized into a finite subset $\mathcal{X}$ by specifying evenly spaced values of the 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 throughout training. 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 1. These weights are fixed during training. See Figure 2 for a visual guide.

To extend these techniques to other types of data besides 2D, one can parameterize patches of data (analogously to CF and KF) and layer slices (analogously to COL and KOL) by other appropriate manifolds analogously to the circle and Klein bottle parameterizations used here. We explore this in detail for video data in an extension of this work. The video patches we use are parameterized by a 6-dimensional space, and they consist of videos of movements around the Klein bottle and translations thereof. This extends further to 3D images (e.g. FMRI) and 3D video. In the case of 1D signals such as time series, relevant 1D patches are gradients (black to white from start to end, or the reverse) and sinusoids (a gradient from black at the start to white in the middle and then a gradient to black and the end, or the reverse coloration).

3 Results

We conduct several experiments on commonly used datasets. We perform digit classification on 3 datasets: MNIST [10], SVHN [11] and USPS [5]. On this data, we investigate the effect of Gaussian white noise on training the TCNN, the interpretability of TCNN activations, and the learning rate of TCNNs in terms of testing accuracy over a number of batches. We also test TCNNs on two collections of labeled images of cats and dogs: Kaggle Cats vs. Dogs [6], and the cats and dogs images in CIFAR-10 [8]. With both the digits and cats and dogs datasets, we investigate the generalization accuracy of TCNNs when trained on one dataset and tested on the other. For additional details on these datasets and train/test splits see Supplemental Sections 4.1 and 4.2.



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

Figure 1: 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 2). 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.

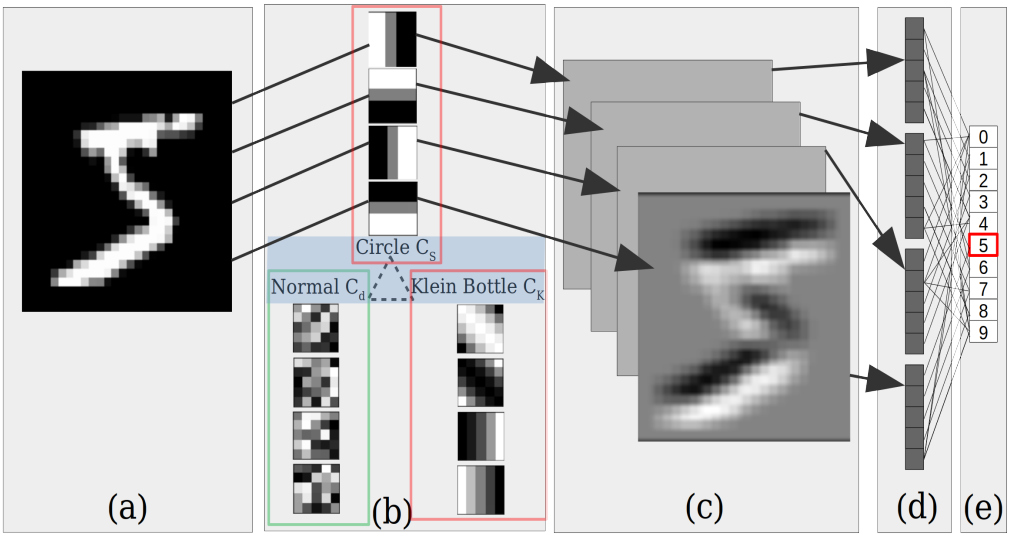


Figure 2: Visual guide to CNN (green rectangle) and TCNN (red rectangles) architectures. A typical CNN layer for image classification takes in a 2-dimensional array (a) as an input image and convolves this image through a system of spatially localized filters (b) to produce multiple slices of processed 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 COL and 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 on the set of slices (fixing weights between distant slices to be zero), and the nonzero weights are instantiated randomly and then allowed to vary during training as usual.

Synthetic experiments. The central hypothesis of this work is that constraining CNNs to train with the Klein bottle filters (KF) and with respect to the Klein bottle topology on slices (KOL) 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 μ_k and σ_k^2 are drawn from $N(.2, .04)$ and $\chi^2(1) \times .04$ respectively. We test our classification models when trained on $\mathbf{z}$ and tested on $\mathbf{x}$ and we also test the converse.

The results of this experiment on MNIST are displayed in Figure 3. 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 added noise. Note that Figure 3 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. For an exploration of alternative noise distributions, see Supplemental Section 4.4.

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. Also it is an interesting observation that the KOL and COL systems seem to learn slower but add robustness to avoiding 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.

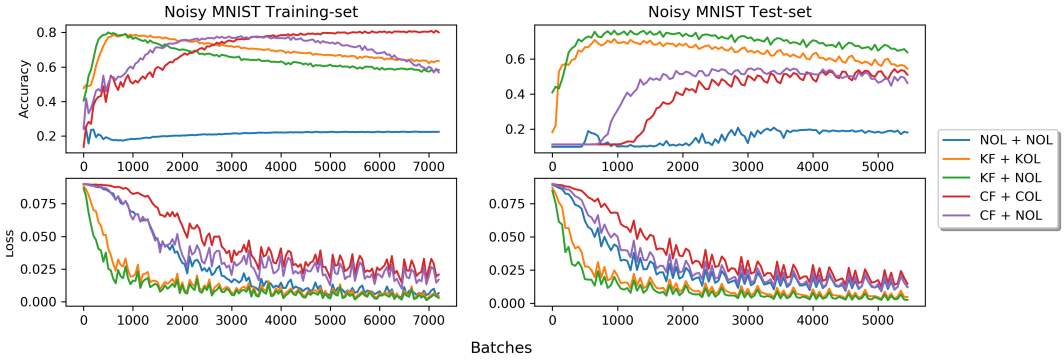


Figure 3: Two synthetic experiments on perturbed MNIST data, designed to test model assumptions. The first column displays the results of the experiment where noise is added to the training data but not the testing data. The second column displays the results of the experiment where the training data is the original MNIST data and the testing data has noise added. The first row is testing accuracy and the second row is training loss.

Interpretability. We contrast the interpretability of CNNs and TCNNs by demonstrating the difference in activations between an NOL, CF, and KF (Figure 1) filtered image from MNIST. The activations shown in Figure 1 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 1, it is empirically clear that the activations of the CF and KF layers are easier to interpret. 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.

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 4 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.

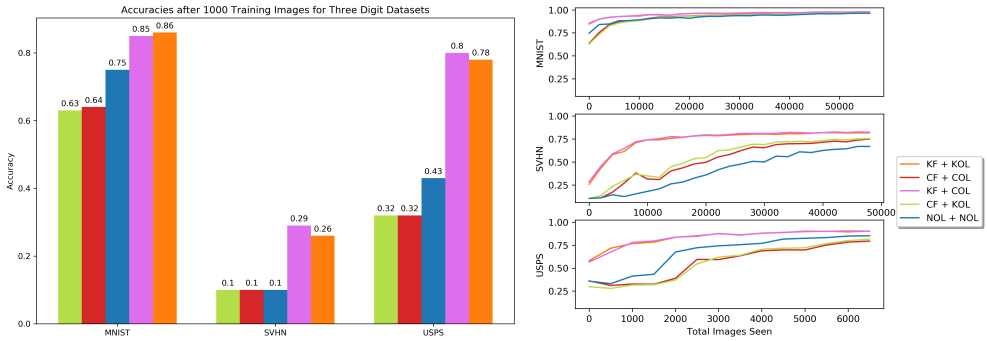


Figure 4: Left: Comparisons of testing accuracy after training on 1,000 images. Right: Full comparison of testing accuracy (y-axes) over a single epoch. MNIST was trained on 60,000, SVHN on 50,032, and USPS on 7291 images.

Generalizability. We compare model generalizability between several models trained on either SVHN or MNIST and tested on the other, and similarly across the Kaggle and CIFAR cats vs. dogs datasets. See Figure 5 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 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.

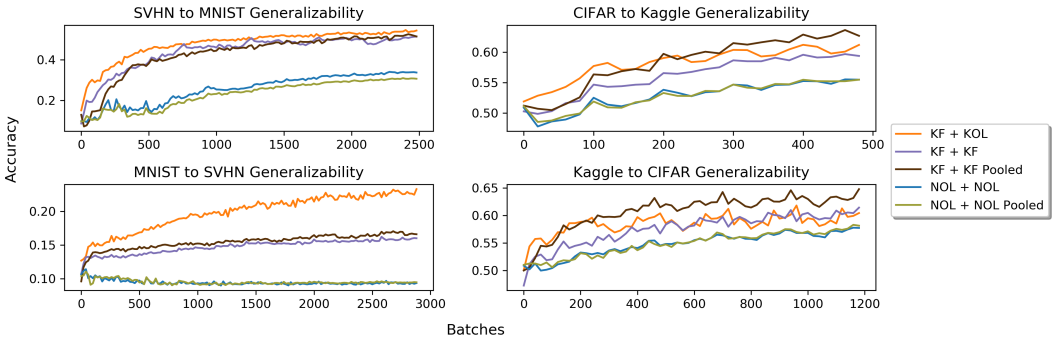


Figure 5: Left: Testing accuracy when generalizing from SVHN to MNIST and vice versa. Right: Testing accuracy when generalizing from CIFAR to Kaggle and vice versa.

4 Supplemental materials

4.1 Datasets considered

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

We consider 5 publicly available image datasets in this paper. In the table above we report the total size of available images and the dimensions of the images upon retrieval. For convenience, we provide a link to download each dataset. The datasets are all cited throughout the paper. 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.

4.2 Train/test splits

With regard to train/test splits, we perform two different types of experiment: (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 tested 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 test and train 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.

4.3 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. 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
1	1	16	5	$1e{-4}$	100	1
3	2	64	3	$1e{-5}$	100	5
4	2	64	3	$1e{-4}$	100	1
5	2	64	3	$1e{-5}$	100	5

Conv-layers. The number of convolutional layers was chosen to be 2 for all experiments outside of Figure 1. 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 was chosen to be 64 for all experiments outside of Figure 1.

Kernel size. Figure 1 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 features. 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 do 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 trained 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 of 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 1 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.

4.4 Synthetic experiment: sensitivity to noise

In our work, we present a synthetic experiment wherein we demonstrate the TCNN's ability to generalize to and from images with an added gaussian noise:

$$\mathbf{z}_k = \mathbf{x}_k + \mathcal{E}_k, \qquad \mathcal{E}_k \sim N(\mu_k, \sigma_k^2),$$

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 $N(\mu_k, \sigma_k^2)$. Here, we generate $\mu_k \sim N(.2, .04)$ and $\sigma_k^2 \sim \chi^2(1) \times .04$. In the first column of Supplemental Figure 6, 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.

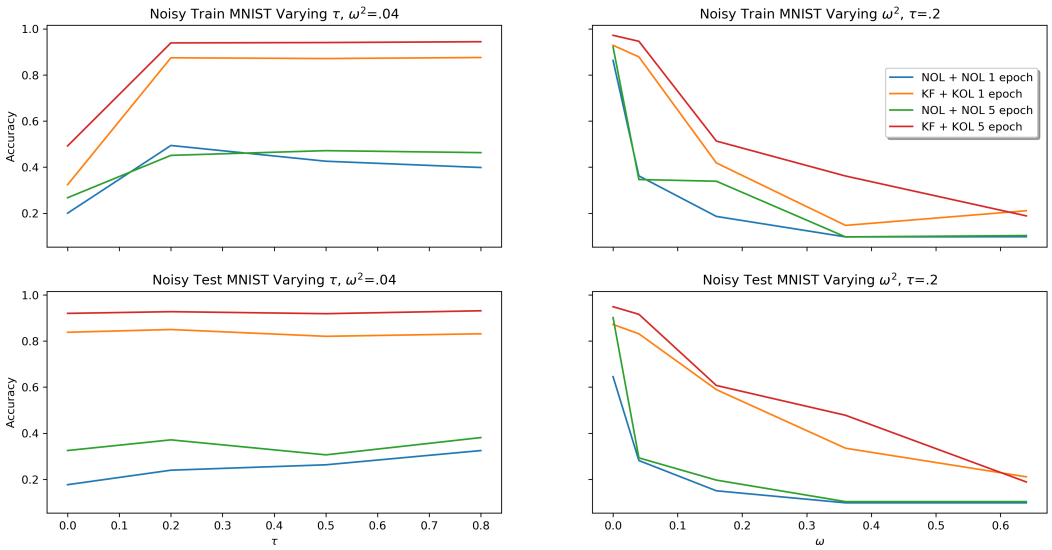


Figure 6: Sweep of distributions for synthetic MNIST data varying the mean and variance from which class-noise distributions are drawn. The first column shows the tested values of τ and the second column shows the tested values of ω^2. The first row shows accuracies when training on data with added noise 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 6 in which we train on data with added noise and test on the original data while varying τ and ω^2. When τ is 0, the noise added has very 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 noise-added 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.

Acknowledgments and Disclosure of Funding

Research has been partially supported by the Army Research Office (ARO) Grant # W911NF-17-1-0313 (VM), and the National Science Foundation (NSF) Grants # MCB-1715794 (EL, VM), DMS-1821241 (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.

References

[1] G. Carlsson and R. B. Gabrielsson. Topological approaches to deep learning. In *Topological Data Analysis*, pages 119–146, Cham, 2020. Springer International Publishing.

[2] G. Carlsson, T. Ishkhanov, V. de Silva, and A. Zomorodian. On the Local Behavior of Spaces of Natural Images. *International Journal of Computer Vision*, 76(1):1–12, Jan. 2008.

[3] Y. Guo, Y. Liu, A. Oerlemans, S. Lao, S. Wu, and M. S. Lew. Deep learning for visual understanding: A review. *Neurocomputing*, 187:27–48, 2016.

[4] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

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

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

[7] K. N. Kay, T. Naselaris, R. J. Prenger, and J. L. Gallant. Identifying natural images from human brain activity. *Nature*, 452(7185):352–355, 2008.

[8] A. Krizhevsky. Learning multiple layers of features from tiny images. *University of Toronto*, 05 2012.

[9] A. Krizhevsky, I. Sutskever, and G. 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.

[10] 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.

[11] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

[12] W. Rawat and Z. Wang. Deep convolutional neural networks for image classification: A comprehensive review. *Neural computation*, 29(9):2352–2449, 2017.