Topological Approaches to Deep Learning

Gunnar Carlsson
Department of Mathematics, Stanford University
Stanford, California 94305

Rickard Brüel Gabrielsson
Department of Computer Science, Stanford University
Stanford, California, 94305

November 6, 2018

1 Introduction

Deep neural networks [10] are a powerful and fascinating methodology for solving problems with large and complex data sets. They use directed graphs as a template for very large computations, and have demonstrated a great deal of success in the study of various kinds of data, including images, text, time series, and many others. One issue that restricts their applicability, however, is the fact that it is not understood in any kind of detail how they work. A related problem is that there is often a certain kind of overfitting to particular data sets, which results in the possibility of so-called adversarial behavior, where they can be made to fail by making very small changes to image data that is almost imperceptible to a human. For these reasons, it is very desirable to develop methods for gaining understanding of the internal states of the neural networks. Because of the very large number of nodes (or neurons), and because of the stochastic nature of the optimization algorithms used to train the networks, this is a problem in data analysis, specifically for unsupervised data analysis. The initial goal of the work in this paper was to perform topological data analysis (TDA) on the internal states of the neural nets being trained on image data to demonstrate that TDA can provide this kind of insight, as well as to understand to what extent the neural net recapitulates known properties of the mammalian visual pathway. We have carried out this analysis, and the results are reported in Section 4. We show that our findings are quite consistent with the data analytic results on image patches in natural images obtained in [2]. In addition, we are able to study the learning process in one example, and also to study a very deep pre-trained neural network, with interesting results which clarify the roles played by the different layers in the network.

Having performed these experiments, we became interested in the question of how to apply the knowledge obtained from our study to deep learning more generally. In particular, we asked how one might generalize the convolutional neural net (CNN) construction to other data sets, so as to obtain methods for constructing efficient nets that are well adapted to other large classes of data sets, or individual data sets. We found that the key idea from the image CNN construction is

the fact that the set of features (pixels) is endowed with a geometry, which can be encoded in a metric, coming from the grid in which the pixels are usually arranged. However, in most data sets, one has one or more natural notions of distance between features, and generalizations based on such metrics appeared to be a potentially very powerful source of methods for constructing neural nets with restrictions on the connections based on such a metric. The idea of studying geometric properties of features has been foreseen by M. Robinson in [11] under the heading of *topological signal processing*. The second goal for us in this paper, then, is to introduce a mathematical formalism for constructing neural network structures from metric and graph based information on the feature space of a data set. We also find that this formalism simplifies and makes precise the specification of neural networks even while using standard methods. In Section 5.2 we evaluate the improvements possible from the very simplest application of this idea. The nature of the improvements come in two directions. The first is in speeding up the learning process. The training of neural nets can be quite a time consuming process, and it is clearly desirable to lower the cost (in time) of training. We found that the methods were more effective on more complex data sets, which is encouraging. A second kind of improvement is in the direction of *generalization*. When training on image data sets, it is standard procedure to select two subsets of the data set, one the training set and the other the test set. The network is trained on the training set, and accuracy is evaluated on the test set. This procedure is designed to guard against overfitting, and the accuracy often achieves very impressive numbers. However, one can consider the problem of training on one data set of images and evaluating on an entirely different data set. For example, there are two familiar data sets consisting of images of digits, one MNIST [7] and the other SVHN [16]. The first is a relatively "clean" data set, The second is actually obtained from images of numbers for the addresses of houses. One could attempt to train on MNIST and evaluate accuracy not on a different subset of MNIST, but rather SVHN. Surprisingly, the results of this process yield abysmal results, with an accuracy very close to that achieved by random selection of classifications. We demonstrate that by the use of the methods we have discussed one can improve the accuracy significantly, although still not to an acceptable level. It suggests that further application of the methods could give us much improved generalization.

We identify three separate scenarios giving rise to geometric information about the feature space. The first is where by its very construction, a set of features is equipped with a geometric structure. Typical examples of this situation are images or time series, where, for example, the pixels (features of images) are designed with a rectangular geometry in mind. The second is where a geometry is obtained from studies such as that performed in [2]. Finally, there is a situation where one is given a more or less general data matrix with numerical entries, and imposes a metric on it via standard choices of metric such as Euclidean, Hamming, etc. Once this has been done, it is important to be able to compress this geometric information into a smaller representation, something which can be achieved by the Mapper construction [12].

We believe that the study of the geometry of the feature space attached to various kinds of data sets will be a very powerful tool that can inform the construction and improve the performance of neural networks. Additionally, because we have incorporated geometric methods in the constructions, we also believe that our formalism opens the door to more sophisticated, detailed, and nuanced mathematical analysis of neural networks.

2 Neural Nets

This section will introduce feed-forward neural nets as well as the special case of convolutional neural nets (CNN's).

Definition 2.1 *By a* feed-forward system *of depth r we will mean a directed acyclic graph Γ with vertex set $V(\Gamma)$ with the following properties.*

1. *$V(\Gamma)$ is decomposed as a disjoint union*
$$V(\Gamma) = V_0(\Gamma) \sqcup V_1(\Gamma) \sqcup \cdots \sqcup V_r(\Gamma)$$
2. *If $v \in V_i(\Gamma)$, then every edge of the form (v, w) of Γ has $w \in V_{i+1}(\Gamma)$.*
3. *The nodes in $V_0(\Gamma)$ (respectively $V_r(\Gamma)$) are called* initial nodes *(respectively* terminal nodes*).*
4. *We assume that for every non-initial node $w \in V_i(\Gamma)$, there is at least one $v \in V_{i-1}(\Gamma)$ so that (v, w) is an edge in Γ.*
5. *For each vertex v of Γ, we denote by $\Gamma(v)$ (respectively $\Gamma^{-1}(v)$) the set of all vertices w of Γ so that (v, w) (respectively (w, v)) is an edge of Γ.*

The sets $V_i(\Gamma)$ are referred to as the layers *of the feed-forward system. We say that a layer $V_i(\Gamma)$ is* locally finite *if the sets $\Gamma^{-1}(v)$ are finite for all $v \in V_i(\Gamma)$. By a* sub-feed-forward system *of a feed-forward system Γ of depth r, we mean a directed subgraph $\Gamma_0 \subseteq \Gamma$ so that the graph Γ_0 and the families of vertices $V_0(\Gamma) \cap \Gamma_0, \ldots, V_r(\Gamma) \cap \Gamma_0$ themselves form a feed-forward system. In particular, it must be the case that for each $v \in \Gamma_0$, the set $\Gamma^{-1}(v) \cap \Gamma_0$ must be non-empty.*

Remark 2.1 Note that we do not assume that Γ is finite. It is sometimes useful to use infinite feed forward systems as idealized constructions with useful finite systems contained in it.

Remark 2.2 We have described only the simplest kinds of structures used in neural nets. There are many others, which can also be described using the methodology we are introducing, but we leave them to future work.

It is also useful to have a slightly different point of view on feed-forward systems. Recall that a correspondence from a set X to a set Y is a subset $\mathcal{C} \subseteq X \times Y$. It is clear that one can compose correspondences, and for any correspondence $\mathcal{C} : X \to Y$ we will write $\mathcal{C}(x) = \{y \in Y | (x, y) \in \mathcal{C}\}$ and $\mathcal{C}^{-1}(y) = \{x \in X | (x, y) \in \mathcal{C}\}$. We also say that a correspondence $\mathcal{C} : X \to Y$ is *surjective* if $\mathcal{C}^{-1}(y) \neq \emptyset$ for all $y \in Y$. These notions are familiar, but we give some particular examples that will be relevant for the construction of convolutional neural networks.

Example 2.1 Given any two sets X and Y, we have the *complete correspondence* $\mathcal{C}^c(X, Y) : X \to Y$, defined b y $\mathcal{C}^c(X, Y) = X \times Y$.

Example 2.2 Given any map of sets $f : X \to Y$, we have the *functional correspondence* $\mathcal{C}_f : X \to Y$ attached to f, defined to consist of the points in the graph of f, defined to be $\{(x, f(x))|x \in X\}$.

Example 2.3 Let $\mathcal{C} : X \to U$ and $\mathcal{D} : Y \to W$, we define the *product correspondence*

$$\mathcal{C} \times \mathcal{D} : X \times Y \to U \times W$$

by the requirement that $((x, y)(u, w)) \in \mathcal{C} \times \mathcal{D}$ if and only if $(x, u) \in \mathcal{C}$ and $(y, w) \in \mathcal{D}$.

Example 2.4 Let X be a metric space, with distance function d. Suppose further that we are given a non-negative threshold r. Then we define $\mathcal{C}_d(r) : X \to X$, the *metric correspondence with threshold* r from X to itself, by $\mathcal{C}_d(r)(x) = \{x'|d(x, x') \leq r\}$. It will occasionally be useful to permit the definition of metric spaces to include the possibility of infinite values. The three axioms of metric spaces extend in a natural way to this generality.

Example 2.5 Let Γ be graph, with vertex set $V = V(\Gamma)$. Then the *graphical correspondence* $\mathcal{C}_\Gamma : V \to V$ is defined by $(v, v^{'}) \in \mathcal{C}_\Gamma$ if and only if $(v, v^{'})$ is an edge in Γ.

We now give the definition of a kind of object that is completely equivalent to a feedforward system.

Definition 2.2 *Let $\mathcal{I}_r$ denote the totally ordered set $\{0, 1, \ldots, r\}$ regarded as a category. By a* generator *for an r-layer feed-forward system, we will mean a functor F from the category $\mathcal{I}_r$ to the category $\underline{Cor}$ of finite sets and correspondences. The associated feed-forward system has as its vertex set $\coprod F(i)$, and where there is a connection from $v \in F(i)$ to $w \in F(j)$ if and only if (1) $j = i + 1$ and (2) $(v, w) \in F(i \to i + 1)$.*

Feed-forward systems are used to describe and specify certain computations. The nodes are considered variables, so will be assigned numerical values which we call r_v. The nodes in the 0-th or initial layer are regarded as input variables, so they are in one to one correspondence with variables that are attached to a data set.

Definition 2.3 *By an* activator*, we will mean a triple (μ, S, f), where μ is a commutative semigroup structure on $\mathbb{R}$, S is a subsemigroup of the multiplicative semigroup of $\mathbb{R}$, and $f : \mathbb{R} \to \mathbb{R}$ is a function, which we call the* cutoff function*. Given a feedforward structure Γ, an* activation system *for Γ is a choice of an activator (μ_v, S_v, f_v) for each non-initial vertex of Γ. A* coefficient system *for a feed-forward system Γ and activation system (μ_v, S_v, f_v) is a choice of element $\lambda_{(u,v)} \in S_v$ for each edge (u, v) of Γ.*

Remark 2.3 Typically we use only a small number of distinct activators, and also assign all the nodes in a given layer the same activator. For purposes of this paper, the only semigroup structures on $\mathbb{R}$ we use are the additive structure and the commutative operation $(x, y) \to \max(x, y)$. Also, for the purposes of this paper, the only choices for S will be either all of $\mathbb{R}$ or $\{1\}$, but in other

contexts there might be other choices. The cutoff function may be chosen to be the identity, but in general is a continuous function that is a continuous version of a function that is zero below a threshold and 1 above it. The ring $\mathbb{R}$ can be replaced by other rings, such as the field with two elements, which can be useful in Boolean calculations.

We now wish to use this data to construct functions on the input data. We assume we are given a locally finite feed-forward structure Γ, equipped with an activation system $\mathcal{A} = (\mu_v, S_v, f_v)$ and a coefficient system $\Lambda = \{\lambda_{(u,v)}\}$. For each i, with $0 \leq i \leq r$, we set W_i equal to the real vector space of functions from $V_i(\Gamma)$ to $\mathbb{R}$. We now define a function $\varphi_i = \varphi_i(- : \mathcal{A}, \Lambda) : W_{i-1} \to W_i$, for $0 \leq i \leq r$, on a function $g : V_{i-1} \to \mathbb{R}$ by

$$\varphi_i(g)(v) = f_v(\sum_{(u,v)\in\Gamma} \lambda_{(u,v)} g(u))$$

Note that the sum is computed using the monoid structure μ_v, and is taken over all edges of Γ with terminal vertex v. This set is finite by the local finiteness hypothesis. We have now constructed functions $\varphi_i : W_{i-1} \to W_i$ for all $0 \leq i \leq r$, and therefore can construct the composite

$$\Phi = \Phi(-; \mathcal{A}, \Lambda) = \varphi_r \circ \varphi_{r-1} \circ \cdots \circ \varphi_1$$

from W_0 to W_r, i.e. a function from the input set to the output set.

The final requirement is the choice of a *loss function*. Given a set of points $\mathfrak{D} \subseteq W_0$, and a function $F : \mathfrak{D} \to W_r$, the goal of deep learning methods is to construct a function φ as above that best approximates the function F in a sense which has yet to be defined. If the function is viewed as a continuous function to the vector space W_r, then the finding the best L_2 approximation is quite reasonable, and the L_2 distance from the approximating function to F will be defined to be the loss function. If, however, the output function is categorical, i.e. has a finite list of values, then it is often the case that the possible outputs are identified with the vertices in the standard $(n-1)$-simplex

$$\{(x_1, \ldots, x_n) | x_i \geq 0 \text{ for all } i \text{ and } x_1 + \cdots + x_n = 1\}$$

in $\mathbb{R}^n$, and other loss functions are more appropriate. The output function still takes continuous valued, and the goal becomes to fit a continuous function to the discrete one. One could of course do this directly, but it has been found that fitting certain transformations of the continuous function perform better. One very common choice is the following. Suppose that from the construction of the neural net, it is known that the values of the neurons in the terminal layer are always positive real numbers. Define $\sigma_n : \mathbb{R}^n_+ \to \mathbb{R}^n_+$ by

$$\sigma_n(x_1, \ldots, x_n) = \frac{1}{x_1 + \cdots + x_n}(x_1, \ldots, x_n) \tag{2–1}$$

The function σ_n takes its values in the standard $(n-1)$ simplex. The *softmax* function is the composite $\sigma \circ \exp$, where exp denotes the function $(x_1, \ldots, x_n) \to (e^{x_1}, \ldots, e^{x_n})$ from $\mathbb{R}^n$ to $\mathbb{R}^n$. A standard procedure for optimizing fitting a continuous function F with discrete values $\{\alpha_1, \alpha_2, \ldots \alpha_n\}$ is to minimize the L_2 error of the transformed function

$$\sigma_n \circ exp \circ \varphi$$

where n is the number of neurons in the output layer. This notion of loss or error is referred to as the softmax loss function.

Deep learning proceeds to minimize the chosen loss function of the difference between $\Phi(-;\mathcal{A},\Lambda)$ and a given function g over the possible choices of the coefficients $\lambda_{(v,w)}$ using a stochastic variant of the gradient descent method. Note that F is typically empirically observed, it is not given as a formula. The optimization process often is time consuming, and occasionally becomes stuck in local optima. We refer to a feed-forward system equipped with activation system as a *neural net*.

Definition 2.4 *Consider a locally finite feed-forward system Γ, possibly infinite, equipped with an activation system $\mathcal{A}$. Let $\Gamma_0 \subseteq \Gamma$ be a sub-feed-forward system. If $\mathcal{A}$ is an activation system on Γ, then it is clear that its restriction $\mathcal{A}|\Gamma_0$ to Γ_0 is an activation system for Γ_0 and that similarly, a coefficient system Λ on Γ restricts to an coefficient system $\Lambda|\Gamma_0$ on Γ_0. We will call the neural net $(\Gamma_0, \mathcal{A}|\Gamma_0)$ the* restriction *of the neural net $(\Gamma,\mathcal{A})$ to Γ_0.*

There is an additional kind of structure on a feed-forward system that is particularly useful for data sets of images, as well as other types of data.

Definition 2.5 *By a* convolutional structure *on a layer $V_i(\Gamma)$ in a feed-forward system Γ we mean a pair $(\simeq,\psi)$, where $\simeq$ is an equivalence relation on the set of vertices of $V_i(\Gamma)$, and where ψ is an assignment of a bijection*

$$\psi_{(v,w)} : \Gamma^{-1}(v) \to \Gamma^{-1}(w)$$

for any pair (v,w) in $\simeq$, satisfying the requirement that $\psi_{(v,w)} = \psi^{-1}_{(w,v)}$ and $\psi_{(w,v)} = \psi_{(w,u)} \circ \psi_{(u,v)}$ when defined. An activation system for Γ is said to be adapted *to the convolutional structure on a layer $V_i(\Gamma)$ if whenever $v \simeq w$, it is the case that $(\mu_v, S_v, f_v) = (\mu_w, S_w, f_w)$. A coefficient system $\{\lambda_{(v,w)}\}$ for the neural net $(\Gamma,\mathcal{A})$ is* adapted *to a convolutional structure $\{\simeq, \psi_{(v,w)}\}$ if it satisfies the compatibility requirement that whenever $v \simeq w$, then we have*

$$\lambda_{(u,v)} = \lambda_{(\psi_{(v,w)}(u),w)}$$

for all $u \in \Gamma^{-1}(v)$.

Example 2.6 Suppose that a layer $V_i(\Gamma)$ and the layer $V_{i-1}(\Gamma)$ are acted on by a group G, and suppose further that for any $v \in V_{i-1}(\Gamma)$ and $w \in V_i(\Gamma)$, (v,w) is an edge in Γ if and only if (gv, gw) is an edge for all $g \in G$. Suppose further that the actions on both $V_{i-1}(\Gamma)$ and $V_i(\Gamma)$ are free, so that the only element of G that fixes a node is the identity element. We define an equivalence relation $\simeq$ on $V_i(\Gamma)$ by declaring that $v \simeq w$ if and only if there is an element $g \in G$ so that $gv = w$. Because of the freeness of the action, v and w determine g uniquely. We define the bijection $\psi_{(v,w)} : \Gamma^{-1}(v) \to \Gamma^{-1}(w)$ to be multiplication by g. Because the group preserves the directed graph structure in Γ, g does carry $\Gamma^{-1}(v)$ to $\Gamma^{-1}(w)$. The application of this idea to data sets of images uses the group $\mathbb{Z}^2$, whose points correspond to an infinite pixel grid. We call structures defined this way *Cayley structures*.

The description of a convolutional layer in Example 2.6 is useful in many situations where the group, and therefore the feed-forward system, are infinite. Nevertheless, it is useful to adapt the networks to finite regions in the grid, such as $N \times N$ grids within an infinite pixel grid. This fact motivates the following definition.

Definition 2.6 *We suppose that we have a feed-forward structure Γ, a layer $V_i(\Gamma)$ equipped with a convolutional structure $\{\simeq, \psi_{(v,w)}\}$, and a sub-feed-forward structure $\Gamma_0 \subset \Gamma$. The restriction of the equivalence relation to $V_i(\Gamma_0)$ does give an equivalence relation on $V_i(\Gamma_0)$, but it does not necessarily have the property that the restriction of the bijections $\psi_{(v,w)}$ to $\Gamma^{-1}(v) \cap V_{i-1}(\Gamma_0)$ remains a bijection. We will define an equivalence relation $\simeq_0$ on $V_i(\Gamma_0)$ by declaring that $v \simeq_0 w$ if and only if (a) $v \simeq w$ as vertices in $V_i(\Gamma)$ and (b) $\psi_{(v,w)}$ restricts to a bijection from $\Gamma^{-1}(v) \cap V_{i-1}(\Gamma_0)$ to $\Gamma^{-1}(w) \cap V_{i-1}(\Gamma_0)$. This clearly produces a convolutional structure on the layer $V_i(\Gamma_0)$ in the feed-forward structure Γ_0, which we refer to as the* restriction *of the convolutional structure e $\{\simeq, \psi_{(v,w)}\}$ on $V_i(\Gamma)$ to Γ_0.*

3 Natural Images and Convolutional Neural Nets

Data sets of images are of a great deal of interest for many problems. For example, the task of recognizing hand drawn digits or letters from images taken of them is a very interesting problem, and an important test case. Neural net technology has been successfully applied to this situation, but in many ways the success is not well understood, and it is believed that it is often due to overfitting. Our goal is to understand the operation of this methodology better, and to use that understanding to improve performance in terms of speed, and of the ability to generalize from one data set to another. In this section we will discuss image data sets, the feed-forward systems that have been designed specifically for them, the extent to which the neural networks act similarly to learning in humans and primates, and how such insights can be used to speed up and improve generalization from one image data set to another.

By an *image*, we will mean an assignment of numbers (gray scale values) to each pixel of a pixel array, typically arranged in a square. The image can be regarded as a P-vector, where P denotes the number of pixels in an array. However, the grid structure of the pixels tells us that there is additional information, namely a geometric structure on the set of coordinates in the vector. It turns out to be useful to build neural nets with a specific structure, reflecting this fact. For simplicity of discussion, it turns out to be useful to build infinite but locally finite models first, and then realize the actual computations on a finite subobject of these infinite constructions, by restricting the support of the activation systems we consider in the optimization. We will be specifying our neural networks by generators. First, we let $\mathbb{Z}$ denote the integers. By $\mathbb{Z}^2 = \mathbb{Z} \times \mathbb{Z}$ we will mean the metric space whose elements consist of ordered pairs of integers, and where the distance function is the restriction of the L^∞ distance on $\mathbb{R}^2$. We of course have the metric correspondences from $\mathbb{Z}^2$ to itself. We will define another family of correspondences called *pooling correspondences*. For any pair of integers $m \leq n$, let $[m, n]$ denote the intersection of the interval $[m, n]$ in the real line with the integers. Let N denote a positive integer, and define a correspondence $\pi(m, n, N)$ to be α^{-1} where $\alpha : \mathbb{Z} \to \mathbb{Z}$ is defined by $\alpha(x) = [Nx + m, Nx + n]$. We have two parameters that are of interest for these correspondences, the *stride*, which is the integer N, and the *width*, which is the

integer $n - m + 1$. To give a sense of the nature of these correspondences, consider the situation with stride and width both equal to 2, and with $m = 0$. In this case, it is easy to check that the correspondence $\pi(0, 1, 2)$ is given by $x \to \lfloor \frac{x}{2} \rfloor$. In general, if the stride is equal to the width, the correspondence $\pi(m, n, N)$ is actually functional, and the corresponding function is N to 1. We'll write $\pi^s(m, n, N)$ for the s-fold product of $\pi(m, n, N)$ as a correspondence from $\mathbb{Z}^s$ to itself.

It will be useful to have a language to describe the layers in a feed-forward system in terms of the generators.

Definition 3.1 *Let Γ denote a feed-forward system, with generator $F : \mathcal{I}_r \to \underline{Cor}$. For any $i \in \mathcal{I}_r$, we consider the i-th layer $F(i)$ as well as the correspondence $\theta_i = F(i - 1 < i) : F(i - 1) \to F(i)$.*

1. *We say the layer $F(i)$ is* fully connected *if θ_i is the complete correspondence $\mathcal{C}^c(F(i-1), F(i))$, as defined in Example 2.1.*
2. *We say $F(i)$ is* grid convolutional *if there are sets X and Y, so that θ_i is of the form*

$$\mathcal{C}^c(X, Y) \times \mathcal{C}_d(r) : X \times \mathbb{Z}^2 \to Y \times \mathbb{Z}^2$$

   where $\mathcal{C}_d(r)$ is a metric correspondence as defined in Example 2.4.

3. *We say $F(i)$ is* pooling *if θ_i is of the form*

$$\mathcal{C}^c(X, Y) \times \pi^2(m, n, N) : X \times \mathbb{Z}^2 \to Y \times \mathbb{Z}^2$$

Remark 3.1 The reason for taking the product of convolutional or pooling correspondences with complete correspondences is in order to accommodate the idea of including numerous copies of a grid within a layer, but with the understanding that the graph connections between any copy of a grid in $F(i-1)$ and any copy in $F(i)$ are identical. This is exactly what the product correspondence achieves.

We are now in a position to build some convolutional neural networks. We will do so by constructing a generator. The generator is a functor that can be specified by a diagram like the following, where writing $X(n)$ denotes a set of cardinality n.

$$X(1) \times \mathbb{Z}^2 \xrightarrow{\mathcal{C}^c \times \mathcal{C}_d(1)} X(64) \times \mathbb{Z}^2 \xrightarrow{\mathcal{C}^c \times \pi^2(0,1,2)} X(64) \times \mathbb{Z}^2 \xrightarrow{\mathcal{C}^c} X(10) \tag{3–2}$$

To further simplify the description, we note that there is product decomposition of the functor F. For an two functors $F, G : \underline{C} \to \underline{Cor}$, we can form the product functor $F \times G$, which is defined to be the point wise product on object, and which also forms the product correspondences. It is clear from the description above that the functor we have described decomposes as the functor $F_0 \times F_1$, where F_0 is given by

$$X(1) \xrightarrow{\mathcal{C}^c} X(64) \xrightarrow{\mathcal{C}^c} X(64) \xrightarrow{\mathcal{C}^c} X(1)$$

and F_1 by

$$\mathbb{Z}^2 \xrightarrow{\mathcal{C}_d(1)} \mathbb{Z}^2 \xrightarrow{\pi^2(0,1,2)} \mathbb{Z}^2 \xrightarrow{\mathcal{C}^c} X(10)$$

This kind of decomposition is ubiquitous for neural networks, where there is one functor F consisting entirely of complete correspondences. We will say a generator F is *complete* if each of the correspondences $F(i < i+1)$ is a complete correspondence, and describe generators F as $F = F^c \times F^s$, where F^c is a complete correspondence, and F^s will be referred to as the *structural generator*. We note that a complete correspondence F is completely determined by the cardinalities of the sets $F(i)$, and so we specify F by its list of cardinalities. We say that the *type* of a complete generator $F : \mathcal{I}_r \to \underline{Cor}$ is the list of integers

$$[\#F(0), \#F(1), \ldots, \#F(r)]$$

and note that the type determines the structure of F.

4 Findings

Because of the stochastic nature of the optimization algorithms used in convolutional neural nets, the problem of understanding how they function is a problem in data analysis. What we mean by this is that it is a computational situation where there are outliers which are not meaningful, and a useful analysis must proceed by understanding what the most common (or dense) phenomena are, in a way that permits one to ignore the outliers, which will be sparse. Before diving into the methodology and results of our study, we will talk about earlier work [2] on the statistics of natural images which is quite relevant to our results on convolutional neural nets.

The work in [2] was a study of a data set constructed by Mumford et al in [8] based on a database of images collected by van Hateren and van der Schaaf [13]. The images were taken in Groningen in the Netherlands, and Mumford et al collected a data set consisting of 3×3 patches, thresholded from below by variance of the patch. Each patch consists of nine gray scale values, one for each pixel. The data was then mean centered, and the contrast (a weighted version of variance) was normalized to have value 1. This means that the data can be viewed as residing on the sphere S^7, a subspace of $\mathbb{R}^8$. Finally, the data was filtered by *codensity*, a function on the data set defined at a point x to be the distance from x to its k-th nearest neighborhood. The integer k is a parameter, much as variance is a parameter in kernel density estimators, and the codensity varies inversely with density.

What was done in [2] as to select a threshold value ρ (a percentage) for the codensity computed for a value k, and consider only points whose codensity was less than ρ. For example, one might study the set of data points which are among the lowest 25% in codensity, computed for the parameter value $k = 300$. This was carried out in [2] for a 30% threshold value, and for the parameter values $k = 300$ and $k = 15$.

These diagrams were obtained by examining the data following persistent homology computations which showed $\beta_1 = 1$ in the case of Figure 1 and $\beta_1 = 5$ in the case of Figure 2 (note that in the case of Figure 2 the model is not actually three disjoint circles, instead each of the secondary circles

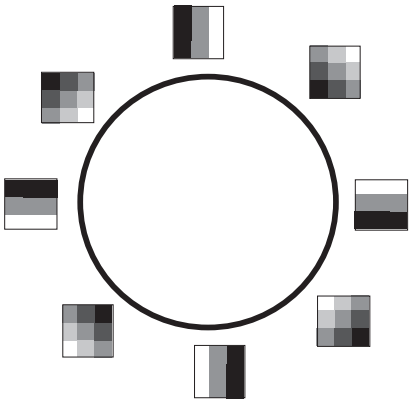

Figure 1: $k = 300, \rho = 30\%$

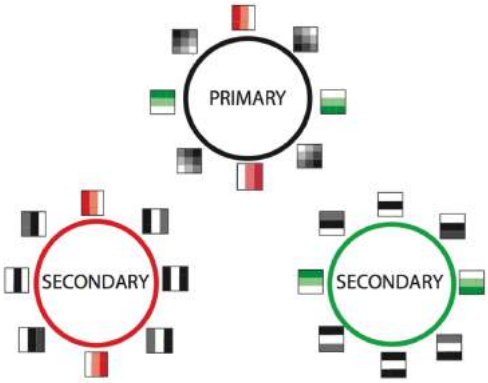

Figure 2: $k = 15, \rho = 30\%$

intersects the primary circle in two data points. The work in [2] went further and found more relaxed thresholds that yielded a Klein bottle instead of just a one skeleton, indicating that more is going on. It meant that the data set actually included arbitrary rotations of the two secondary circles in Figure 2. The original motivation for the work in [13] and [8] was to understand if analysis of the spaces of patches in natural images is reflected in the "tuning" of individual neurons in the primary visual cortex. We set out to determine if the statistical analysis of [2] has a counterpart in convolutional neural networks for the study of images. The following are insights we have obtained.

- The role of thresholding by density or proxies for density is crucial in any kind analysis of this kind. Without that a very small number of outliers can drastically alter the topological model from something giving insight to something essentially useless.
- The development of neural networks was based on the the idea that neural networks are analogous to networks of neurons in the brains of mammals. There is an understanding [5] that the primary visual cortex acts as a detector for edges and lines, and also that higher level components of the visual pathway detect more complex shapes. We perform an analysis analogous to the one in [2], and show that it gives results consistent with the density analysis performed there.
- We demonstrate that our observations can be used to improve the ability of a convolutional neural network to generalize from one data set to another.
- We demonstrate that the results can be used to speed up the learning process on a data set of images

We next describe the way that the data analysis was performed. We suppose that we have fixed an architecture for a convolutional neural network analysis of a data set of images, using grid layers as described in Section 3. We used an architecture in which the correspondences $\mathcal{C}_d(1)$ described the connections into a convolutional layer, where d is the L^∞ metric on the grids. This means that any node in a grid layer is connected to the nodes which belong to a 3×3 patch surrounding it. The weights therefore constitute a vector in $\mathbb{R}^9$, which corresponds exactly to raw data used in [8].

The data points will be referred to as *weight vectors*. In [4], we performed analyses on data sets constructed this way using a methodology identical to that carried out in [8] and [2]. The rest of this section will describe the results of this study.

We first discuss the two data sets that we studied. The first is MNIST [7], which is a data set of images of hand drawn digits. The images are given as 28×28 gray scale images. For this data set, we used an architecture described as follows. The depth is 6, and the generator F is a product of two generators, F^c and F^s. The complete factor F^c is of type $[1, 64, 64, 32, 32, 64, 1]$, and the structural factor has the form

$$G_{28} \xrightarrow{\mathcal{C}_d(1)} G_{28} \xrightarrow{\pi^2(0,1,2)} G_{14} \xrightarrow{\mathcal{C}_d(1)} G_{14} \xrightarrow{\pi^2(0,1,2)} G_7 \xrightarrow{\mathcal{C}^c} X(1) \xrightarrow{\mathcal{C}^c} X(10) \tag{4–3}$$

where $G_i \subseteq \mathbb{Z}^2$ denotes an $i \times i$ grid, $X(i)$ denotes a set of cardinality i, and the output layer $X(10)$ is identified with the ten digits $0, 1, \ldots, 9$. This feed-forward structure embeds as a sub-feed-forward structure of the structure F^s_∞ obtained by replacing all the finite grids G_i with copies of $\mathbb{Z}^2$, into which they embed. Therefore, the layers $F^s(1) = G_{28}$ and $F^s(3) = G_{14}$ inherit a convolutional structure from the Cayley convolutional structure (defined in Definition 2.6) on F^s_∞, which is the convolutional structure we use. The activation systems are defined using two different activation functions f. The first is the *rectifier*, which denotes the function $f(x) = \max(0, x)$, and which is often also denoted by *ReLU*. The second is the identity function and the third is the exponential function $\exp(x) = e^x$. The activation system is given on the layers $F(1)$ and $F(3)$ by $(+, \mathbb{R}, ReLU)$, on the layers $F(2)$ and $F(4)$ by $(\max, \{1\}, id)$, on the layer $F(5)$ by $(+, \mathbb{R}, ReLU)$, and on the layer $F(6)$ by $(+, \mathbb{R}, \exp)$. The loss function (defined on the layer $F(6)$) is the function σ_n defined in (2–1) above.

We now look at results for the neural net trained on MNIST. Figure 3 shows a Mapper analysis of the data set of weight vectors in the first convolutional layer in the neural net described above. The neural net was trained 100 separate times, and each training consisted of 40,000 iterations of the gradient descent procedure. In each node, one can form the average (in $\mathbb{R}^9$) of the vectors in that node. The patches surrounding the Mapper model are such averages taken in representative nodes of the model near the given position. We see that the Mapper model is in rough agreement with the circular model in Figure 1 above.

In Figure 4, we see persistence barcodes computed for for the data set. The computation confirms the presence of connectedness of the data set as well as the presence of a significant loop, which is a strong indication that the Mapper model is accurately reflecting the structure of the data set. Figure 5 shows a Mapper model of the second convolutional layer. One observes that there appear to be patches which are roughly like those in the primary circle, but the structure is generally more diffuse that what appeared in the first layer. Persistence barcodes did not confirm a loop in this case.

The second data set is CIFAR-10 [6], which is a data set of 32×32 color images objects divided into 10 classes, namely airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. The color is encoded using the RGB system, so that each pixel is actually equipped with three coordinates, one for each of the three colors red, green, and blue. There are different options about how to analyze color image data, and we examined three of them.

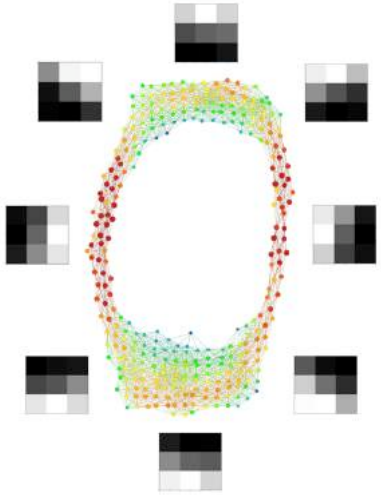

Figure 3: MNIST layer 1

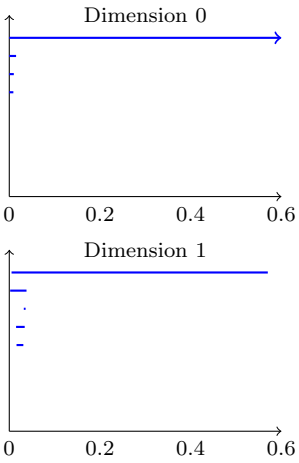

Figure 4: Barcode for layer 1

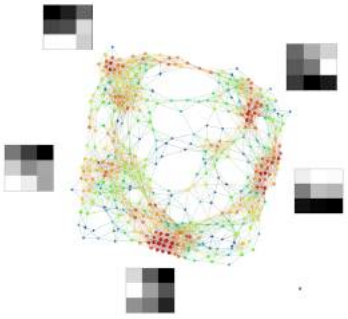

Figure 5: MNIST layer 2

1. Reduce the colors to a single gray scale value by taking a linear combination of the three color values, and then analyze the data set as a collection of gray scale images. We used the combination $.2989 \cdot R + .5870 \cdot G + .1140 \cdot B$. This choice is one standard choice made for this kind of problem. See https://en.wikipedia.org/wiki/Luma_%28video%29 for a discussion.

2. Study the individual color channels separately, producing three separate gray scale data sets, one each for red, green and blue.

3. Consider all three color channels together, and build a neural network to accommodate that. This means in particular that the input layer will need to include three copies of the 32×32 grid.

For options (1) and (2), we constructed a neural net very similar to the one used for MNIST. Its complete factor F^c is of type $[1, 64, 64, 32, 32, 64, 1]$, identical to the one used for MNIST. The structural factor F^s has the form

$$G_{32} \xrightarrow{\mathcal{C}_d(1)} G_{32} \xrightarrow{\pi^2(0,2,2)} G_{16} \xrightarrow{\mathcal{C}_d(1)} G_{16} \xrightarrow{\pi^2(0,1,2)} G_8 \xrightarrow{\mathcal{C}^c} X(1) \xrightarrow{\mathcal{C}^c} X(10) \qquad (4\text{–}4)$$

The generator is identical to the one for MNIST except for the substitution of G_{32}, G_{16}, and G_8 for G_{28}, G_{14}, and G_7, respectively, and for the substitution of a pooling layer of width 3 as the correspondence between $F^s(1)$ and $F^s(2)$. The activation systems are identical to those in the MNIST case, as is the loss function. For option (3), it is necessary to form an additional complete factor G of type $[3, 1, 1, 1, 1, 1, 1]$, and form the product $F^c \times F^s \times G$ as the generator. Of course, the 3's correspond to the set $\{R, G, B\}$. The activation systems and loss functions are identical in all three cases.

We first performed an analysis in the case of option (1). The results were not as clear as in the MNIST analysis, but did give some indications of interesting structure. In particular, the second layer had the Mapper model shown in Figure 6 below. Notice that the primary circle is included, together with a kind of "bullseye" patch which does not appear even in the Klein bottle model given in [2]. We also analyzed option (3) above. In this case, the result was quite striking. A Mapper model of the first layer appears in Figure 7, which we see recovers the three circle model of [2], and where a persistence barcode for this space appears in Figure 8. We also analyzed option 2 above, and found strong primary circles in that case. The findings confirm that generally, the convolutional neural network well reflects the density analysis in [2], as well as the results on the primary visual cortex given in [5]. Moreover, the detection of the bullseye shown in Figure 6 demonstrates that the higher levels of the neural network find more complex patches, not accounted for by the low level analysis encoded in the Klein bottle of [2]. This is also consistent with our understanding of the visual pathway, in which there are higher level units above the primary visual cortex that capture more "abstract" shapes.

We also examined the learning process for CIFAR-10. We did this by performing the analysis in the case of option (1) above at various stages of the optimization algorithm. Figure 9 shows the results for both first and second layers. The numbers below the models show the number of iterations corresponding to the models above them. Most of the models shown are "carpets", which simply reflects the choice of two filter functions for the model. This means that they are not topologically interesting by themselves. However, each node in the a Mapper model consists of a collection of

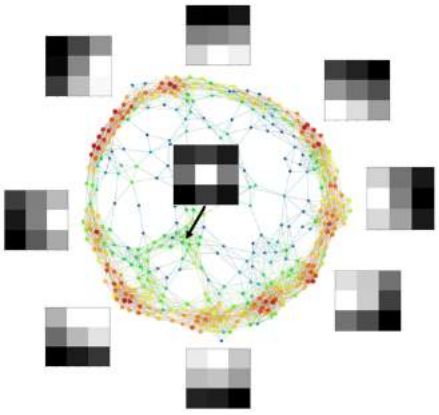

Figure 6: CIFAR-10 layer 2, gray scale

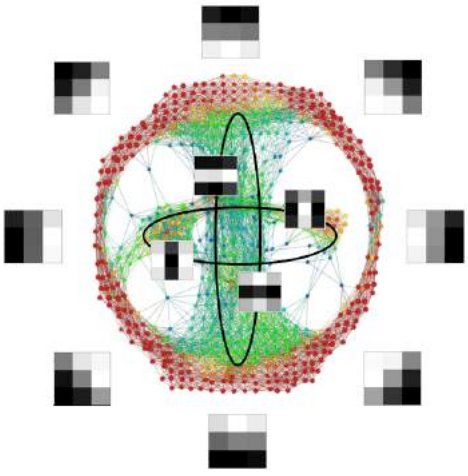

Figure 7: First layer, CIFAR-10, separate colors

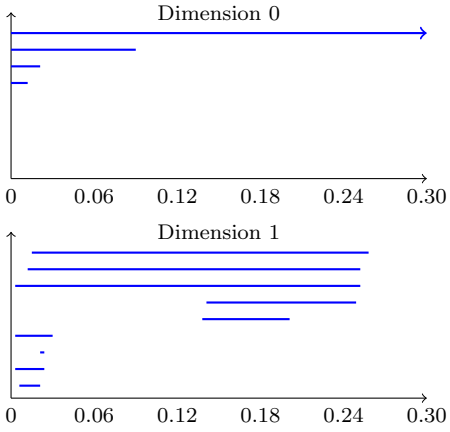

Figure 8: Persistence barcode, Figure 7

data points, and the cardinality of that set becomes a function on the set of vertices of the model. Sub- or superlevel sets of that function can then give interesting information, loosely correlated with density. The models in Figure 9 illustrate this, particularly strongly in the first layer. We note that the first layer, beginning with something near random after 100 iterations, organizes itself into a recognizable primary circle after 200 iterations, remains at that structure until roughly 900 iterations, when the circle begins to "degrade", and instead form a structure which is capturing patches more like those of the secondary circles. The second layer, on the other hand, is not demonstrating any strong structure until it has undergone 1000 or 2000 iterations, when one begins to see the primary circle appearing. One could interpret this as a kind of compensation for the changes occurring in the first layer.

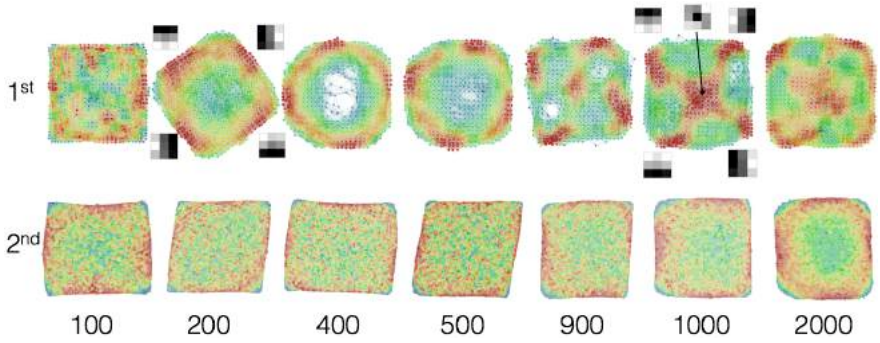

Figure 9: CIFAR-10 learning

Finally, we examined a well known pretrained neural network, VGG16, trained on Imagenet, a large image data base [15],[3]. This neural net has 13 convolutional layers, and so permits us to study seriously the "responsibilities" of the various layers. Mapper models of the sets of weight vectors

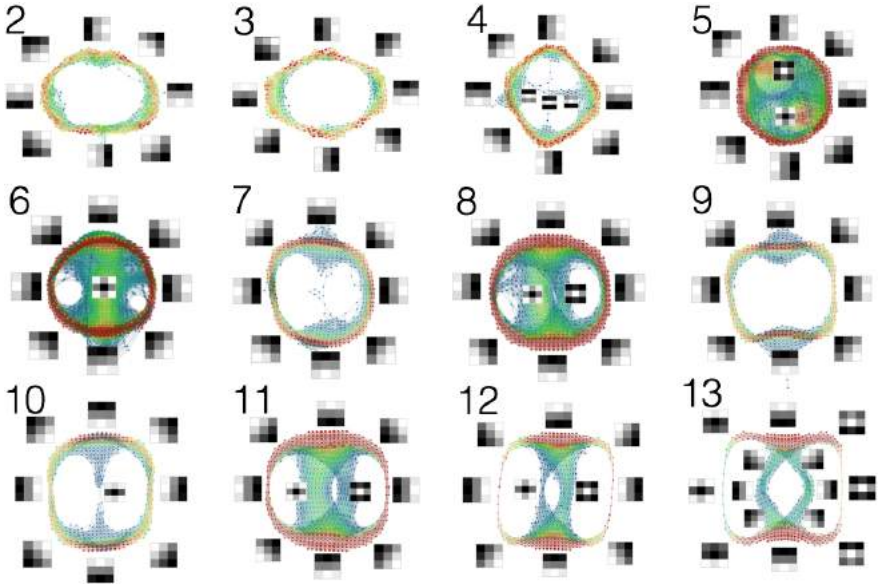

Figure 10: VGG16

for layers 2-13 are shown in Figure 10. In this case, the neural net has sufficiently many grids in each layer to construct a useful data set from this network alone. Observe that the first two layers give exactly a primary circle, and that after that more complex things appear. Secondary circle patches occur in layer 4, and in higher layers we see different phenomena occurring, including the bullseye we saw in CIFAR-10, as well as crossings of lines. One interesting line of research would be to assemble all of these different phenomena into a single space, including the Klein bottle. The advantage of doing this is that it will permit feature generation in terms of functions on the space, such as was done in [14], or improved compression algorithms as in [9]. For now, the outcome demonstrates with precision how the higher layers encode higher layers of abstraction in images, as occurs in the mammalian visual pathway.

5 Feature Geometries and Architectures

5.1 Generalities

Since CNN's have demonstrated a great deal of success on data sets of images, the idea of trying to generalize it suggests itself. To perform the generalization, one must identify what properties of image data sets are being used, and how. There are two key properties.

- **Locality:** The features in image data set (i.e.pixels) are equipped with a geometry, i.e. that of a rectangular grid. That grid is critical in restricting the connections in the corresponding feed-forward structure, and that restriction can be formulated very simply in terms of the L^∞ distance function on the grid, as we have seen in our constructions of CNN's in Section 4. This observation suggests that one can use other metric spaces to restrict the connections in architectures based on these metric spaces. We note that the grid geometry can be regarded as a discretization of the geometry of the plane, or of a square in the plane.
- **Homogeneity:** The convolutional neural net is equipped not only with a connection structure, but a choice of *convolutional structure* (as in Definition 2.5), which creates its own restrictions on the features created in the neural net. Because it requires that weight vectors associated with one point in the grid be identical with those constructed at other points, the convolutional property should be interpreted as a kind of homogeneity. In addition to putting drastic limitations on the features being created in the neural net, this restriction encodes a property of image data sets that we regards as desirable, namely that the same object occurring in different parts of an image should be detected in an identical fashion.

What we would like to do is to describe how the two properties above can be used to construct neural nets in an analogous fashion, to improve performance on image data sets and to generalize the ideas to more general data sets. In order to have a notion of locality, we will need to understand data sets in terms of the geometry of their sets of features. We identify at least three methods in which feature sets can obtain a geometry.

1. **A priori geometries:** The prime example here is the image situation, where the grid ge-

ometry is explicitly constructed in the construction of the data. The continuous version of this geometry is that of the plane. Other examples would include time series, where the a priori continuous geometry is the line, or periodic time series, where the geometry is that of the circle. The geometries for the building of the neural net would be discretizations of these geometries, obtained by selecting discrete subsets, often in a regular way.

2. **Geometries obtained from data analysis:** The data analysis performed in [2] or [4] reveals that the frequently occurring local patches in images concentrate around a primary circle, and that these patches are well modeled by particular functions which can be algebraically defined. We will show below that this fact permits the construction of a set of features for images which admit a circular geometry. One could also construct a Klein bottle based set of features and a corresponding Klein bottle based geometry on that set.

3. **Purely data drive geometries:** In many situations one does not want to perform a detailed modeling procedure for the set of features, but nevertheless wants to use feature geometries to restrict connections in neural nets which are designed to learn a function based on the features. In this case, one can use the Mapper methodology [12] to obtain discretized versions of geometries on the feature space, well suited to the construction of neural nets.

Section 3 can be regarded as a discussion of one case where an a priori geometry is available, so we will not discuss it further. Instead, we will give examples of data analytically obtained geometries and purely data driven constructions.

5.2 Data-analytically Defined Geometries

We first consider the data analytic work that was done in [2] and [4]. We find that the frequently occurring patches are approximable by discretizations of linear intensity functions onto a 3×3 grid. To be specific, we regard the pixels in a 3×3 patch to be embedded in the square $I^2 = [-1, 1] \times [-1, 1]$, as the subset $\mathcal{L} = \{-1, 0, 1\} \times \{-1, 0, 1\}$. The discretization operation can be considered as the restriction of a function on I^2 to $\mathcal{L}$. We consider the set of linear functions in two variables given by the formulae

$$f_\theta(x, y) = x \cos(\theta) + y \sin(\theta)$$

The set of functions is parametrized by the circle valued parameter θ. For each f_θ, we can construct a function on an image as follows. Let $(m, n) \in \mathbb{Z}^2$ denote a particular pixel in the grid defining an image data set $\mathcal{D}$ consisting of images p, with $p(m, n)$ denoting the gray scale value of an image within the data set. Given an angle θ, we now define a function $q_{m,n,\theta}(p)$ on $\mathcal{D}$ by the formula

$$q_{m,n,\theta}(p) = \sum_{(i,j)\in\mathcal{L}} p(m + i, n + j) \cdot f_\theta(i, j)$$

In this case the continuous geometry associated to the feature space for these images is $\mathbb{R}^2 \times S^1$. The discretization will be choosing a rectangular lattice L for $\mathbb{R}^2$ in the usual way, and by choosing the set μ_n of n-th roots of unity for the circular factor. So the discretized form is $\mathbb{Z}^2 \times \mu_n$. This set is a metric space in its own right, and we can use the metric correspondences defined in Example 2.4 to construct generators and neural nets based on this geometry.

Remark 5.1 There are similar synthetic models with a Klein bottle K replacing S^1. There are natural choices for discretizations of K as well.

We have demonstrated that there are methods of imposing locality on new features that have been constructed based on the data analysis of image patches and of weight vectors in convolutional neural nets. For this construction, there are also convolutional structures as defined in Definition 2.5. In fact, they are Cayley structures in the sense of Example 2.6, as we can readily see from the observation that the metric space $\mathbb{Z}^2 \times \mu_n$ is equipped with a free and transitive action by the group $\mathbb{Z}^2 \times \mathbb{Z}/n\mathbb{Z}$, and this group action determines a Cayley convolutional structure. This gives a number of possibilities for the construction of new feed-forward systems with feature geometries taken in to account. To see how these might look, let's consider the feed-forward system F described in (3–2) above. F is broken into a product $F = F^s \times F^c$, where F^c is a complete generator, and the structural factor F^s is given by

$$\mathbb{Z}^2 \xrightarrow{\mathcal{C}_d(1)} \mathbb{Z}^2 \xrightarrow{\pi^2(0,1,2)} \mathbb{Z}^2 \xrightarrow{\mathcal{C}^c} X(10)$$

The idea will be to construct new structural factors by taking products with generators involving only μ_n for various n's. We'll call these generators *angular factors*. The simplest one is of the form

$$\mu_n \xrightarrow{\mathcal{C}^c} X(1) \xrightarrow{\mathcal{C}^c} X(1) \xrightarrow{\mathcal{C}^c} X(1)$$

Here $X(1)$ denotes a one element set. The corresponding structural factor including the grids would then be

$$\mathbb{Z}^2 \times \mu_n \xrightarrow{\mathcal{C}_d(1) \times \mathcal{C}^c} \mathbb{Z}^2 \times X(1) \simeq \mathbb{Z}^2 \xrightarrow{\pi^2(0,1,2)} \mathbb{Z}^2 \xrightarrow{\mathcal{C}^c} X(10) \times X(1) \simeq X(10)$$

The effect of this modification is simply to use the newly constructed features directly in the computation. It permits the algorithm to use them directly rather than having to "learn" them. Another angular factor is

$$\mu_n \xrightarrow{\mathcal{C}^c} \mu_n \xrightarrow{\mathcal{C}^c} X(1) \xrightarrow{\mathcal{C}^c} X(1)$$

Forming the product of this angular factor with F^s and ultimately F^c as well produces a feed-forward structure which creates new angular factors in layer 1. The corresponding neural networks would be able to learn angle dependent functions from earlier angular functions. Yet another angular factor would be the following.

$$\mu_n \xrightarrow{\mathcal{C}_d(\xi)} \mu_n \xrightarrow{\mathcal{C}^c} X(1) \xrightarrow{\mathcal{C}^c} X(1)$$

where ξ is the distance from $(1,0)$ to the primitive root of unity $\zeta_n = (\cos(\frac{2\pi}{n}), \sin(\frac{2\pi}{n}))$. Adding this angular factor to F^s creates new angular features in layer 1, allows these angular features to learn from angular and raw features, and further restricts that learning so that a given angular feature would only depend on raw values and angular features in the input that are near to the given feature in the metric on μ_n. This is the angular analogue to the idea that a convolutional neural net permits a feature in a convolutional layer to depend only on features in the preceding layer that are spatially close to the given feature, in this case in the a priori geometry on pixel space.

There is also an analogue for μ_n to the pooling correspondences $\pi(m,n,N)$ defined in Section 3. They are correspondences from $\pi_{m,n} : \mu_{mn} \longrightarrow \mu_n$, and they are defined by

$$\pi_{m,n}(\zeta_{mn}^x) = \zeta_n^{\lfloor \frac{x}{m} \rfloor}$$

It is easy to verify that this is well-defined. We have only created analogues to the correspondences $\pi(0,n-1,n)$ from Section 3, but analogues for other values of m,n, and N exist as well. We could now construct a new angular factor

$$\mu_{mn} \xrightarrow{\mathcal{C}_d(\xi)} \mu_{mn} \xrightarrow{\pi_{m,n}} \mu_n \xrightarrow{\mathcal{C}^c} X(1)$$

which would incorporate pooling in the angular directions as well. Each of these constructions have analogues for the case of the Klein bottle geometries.

We have some preliminary experiments involving the simplest versions of these geometries. We have used them to study MNIST, as well as the SVHN data set [16]. SVHN is another data set of images of digits, collected by taking photographs of house numbers on mailboxes and on houses. For these studies, we have simply modified the feed-forward systems by constructing the product of the existing structural factors described in (4–3) and (4–4) with an additional structural factor of the form

$$(\mu_{16})_+ \xrightarrow{\mathcal{C}^c} X(1) \xrightarrow{\mathcal{C}^c} X(1) \xrightarrow{\mathcal{C}^c} X(1) \xrightarrow{\mathcal{C}^c} X(1) \xrightarrow{\mathcal{C}^c} X(1) \xrightarrow{\mathcal{C}^c} X(1) \tag{5–5}$$

where $(\mu_{16})_+$ plus denotes μ_{16} with a disjoint point added. This additional point is there so that we include the original "raw" pixel features. This amounts to including the "angular" coordinates described above as part of the input data, and using it to inform the higher level computations. We have two results, one in the direction of speeding up the learning process and the other concerning the generalization from a network trained on MNIST to SVHN.

- We found substantial improvement in the training time for both MNIST and SVHN when using the additional angular features. A factor of 2 speed up was realized for MNIST, and a factor of 3.5 for SVHN. MNIST is a much cleaner and therefore easier data set, and we suspect that the speed up will in general be larger for more complex data sets.
- We also examined the degree to which a network trained on one data set (MNIST) can achieve good results on another data set (SVHN). Using the standard convolutional network for images, we found that a model trained on MNIST applied to SVHN achieved roughly 10% accuracy. Since there are 10 distinct outcomes, this is essentially the same as selecting a classification at random. However, when we built the corresponding model using the additional factor (5–5) above, we found that the accuracy improved to 22%. Of course, one wants much higher accuracy, but what this finding demonstrates is that this generalization problem can be substantially improved using these methods.

In these examples, we have only used the simplest versions of the constructions we have discussed in Section 2. The possibilities that we envision going forward include taking products with structural factors of the form

$$(\mu_{4n})_+ \xrightarrow{\mathcal{C}_d(\xi_{4n})_+} (\mu_{4n})_+ \xrightarrow{(\pi_{4n,2n})_+} (\mu_{2n})_+ \xrightarrow{\mathcal{C}_d(\xi_{2n})_+} (\mu_{2n})_+ \xrightarrow{(\pi_{2n,n})_+} (\mu_n)_+ \xrightarrow{\mathcal{C}^c} X(1) \xrightarrow{\mathcal{C}^c} X(1) \tag{5–6}$$

The correspondences $\mathcal{C}_d(\xi_{2^i n})_+$ and $(\pi_{2^i n, 2^{i-1} n})_+$ for $i = 0, 1, 2$ in this feed-forward system are straightforward generalizations of $\mathcal{C}_d(\xi_{2^i n})$ and $\pi_{2^i n, 2^{i-1} n}$ to the situation where the disjoint base point + has been added. $(\mathcal{C}_d(\xi_n))_+$ is obtained by constructing a metric on $(\mu_n)_+$ for which the distance from the point + to each of the elements of μ_n, as well as all the distances between adjacent roots of unity, are all equal to ξ_n. It is not hard to see that this can be done. $(\pi_{2n,n})_+$ is the functional correspondence which is equal to $\pi_{2n,n}$ on μ_n and which carries the point + to +. The effect of this construction is that it would include angular features at the higher layers, and that it would restrict the angular features that are constructed to include only those which involve nearby angular features in the preceding layers.

5.3 Purely Data Driven Geometries

Suppose that we are given a data set defined by a data matrix D, with the rows corresponding to the data points and the columns corresponding to the features, but that we have no theory for the features analogous to the one described in [2]. What we generally have, though, are metrics on the set of features. If the matrix entries are continuous, one can use Euclidean distance of the features viewed as column vectors. There are variants, such as mean centered and/or variance normalized versions, correlation distance, angle distance, etc. If the entries of the matrix are binary, then Hamming distance is an option. In general, it is most often possible to define, in natural ways, metrics on the set of columns. This means that the feature set is a metric space, and therefore that we already have the possibility of carrying out part of the process used on image data sets, namely the construction of the correspondences $\mathcal{C}_d(r) : X \to X$, where X denotes the feature set. We refer to the column space equipped with a metric as the *feature space*. These can be used to create a counterpart for the initial convolutional layers in the feed-forward system, but it does not give a counterpart to the pooling correspondences. The pooling correspondences are important because they allow one to study features that are more broadly distributed in the geometry of the feature space. To construct deeper networks, one may also need an analogue for higher level convolutional layers. There is an approach using the Mapper methodology introduced in [12] that will directly construct a counterpart to pooling methodology.

We recall that the output of Mapper, applied to a finite metric space X, is a graph Γ, with an assignment to each vertex v of Γ a subset X_v of X, having the following two properties:

1. Every point $x \in X$ is contained in X_v for some vertex v of Γ.
2. Two vertices v and w of Γ are connected by an edge if and only if $X_v \cap X_w \neq \emptyset$.

We observe that this means that if we have two Mapper models $(\Gamma, \{X_v\}_{v \in V(\Gamma)})$ and $(\Gamma', \{X'_w\}_{w \in V(\Gamma')})$ on the same metric space X, then there is a well-defined correspondence

$$\mathcal{C}(\Gamma, \Gamma') : V(\Gamma) \to V(\Gamma')$$

defined by the property that for $(v, w) \in V(\Gamma) \times V(\Gamma')$, $(v, w) \in \mathcal{C}(\Gamma, \Gamma') : V(\Gamma) \to V(\Gamma')$ if and only if $X_v \cap X'_w \neq \emptyset$.

These properties allows us to construct two specific correspondences. Given a metric space X and a Mapper model $(\Gamma, \{X_v\}_{v\in V(\Gamma)})$ for the feature space of a data matrix, we have the *augmentation correspondence* $\varepsilon : X \to V(\Gamma)$, defined by $(x, v) \in X \times V(\Gamma)$ if and only if $x \in X_v$. We also have the correspondence $\mathcal{C}(\Gamma, \Gamma) : V(\Gamma) \to V(\Gamma)$.

Remark 5.2 The correspondence $\mathcal{C}(\Gamma, \Gamma)$ is simply the graph correspondence $\mathcal{C}_\Gamma$ defined in Example 2.5.

To define analogues to pooling correspondences, we need a bit more detail on the Mapper construction. It begins with one or more projections $f : X \to \mathbb{R}$, which we call *filters*. Typically there are only a small number of f's, perhaps 1,2, or 3, and we denote the collection of filters by $\{f_\alpha\}_{\alpha\in A}$, where it is understood that $\#A$ is small. We now construct a family of open coverings of the real line.

Definition 5.1 *Given a pair (l, s) of real numbers with $l > s$, we define the covering $\mathcal{U}(l, s)$ to consist of all intervals of the form $(ks - \frac{l}{2}, ks + \frac{l}{2})$, $k \in \mathbb{Z}$. The condition $l > s$ guarantees that the family is a covering. Given a pair (l, s), we defined the* double *of (l, s) to be the pair $(2l, 2s)$. $\mathcal{U}(2l, 2s)$ covers $\mathbb{R}$ with intervals of double the length of the intervals comprising $\mathcal{U}(l, s)$. We refer to l as the* length *and s as the* stride.

Let n denote the cardinality of A, and equip A with a total ordering, so $A = \{\alpha_1, \ldots, \alpha_n\}$. Let $F : X \to \mathbb{R}^n$ denote the product $f_{\alpha_1} \times \cdots \times f_{\alpha_n}$. For each filter f_α, we choose a pair (l_α, s_α). For each $\alpha \in A$, we let $\mathcal{U}_\alpha = \mathcal{U}(l_\alpha, s_\alpha)$, and let $\mathcal{U}_\alpha = \{I^\alpha_{\beta_\alpha}\}_{\beta_\alpha\in B_\alpha}$, where B_α is an indexing set for the intervals in $\mathcal{U}_\alpha$. We now construct the product covering $\mathcal{U}_{\alpha_1} \times \mathcal{U}_{\alpha_2} \times \cdots \times \mathcal{U}_{\alpha_n}$ of $\mathbb{R}^n$, which consists of sets of the form $I^{\alpha_1}_{\beta_{\alpha_1}} \times \cdots \times I^{\alpha_n}_{\beta_{\alpha_n}}$, for all choices of n-tuples $(\beta_{\alpha_1}, \ldots, \beta_{\alpha_n})$ in $B_{\alpha_1} \times \cdots \times B_{\alpha_n}$. We denote this covering by $\mathcal{V} = \{V_j\}_{j\in J}$, where J is the indexing set. We now create overlapping subsets (bins) of the form $F^{-1}(V_j)$. These sets form a covering of X. The algorithm defined in [12] next proceeds by clustering (using a predefined clustering algorithm) each of the bins, creating a partition of each bin. The vertex set of the Mapper model Γ of X consists of one element for each block of each partition of each bin, and we declare that two vertices are connected by an edge if and only if the corresponding blocks overlap. Note that the blocks can overlap because the bins overlap. It is clear from the construction that this construction has the two properties ascribed to it above. Note that the construction described above depended only on the choices (l_α, s_α). The result of applying the above construction to the choices $(2l_\alpha, 2s_\alpha)$ will be referred to as the *double* of Γ, and we will denote it by $\Gamma(1)$, where it is understood that $\Gamma = \Gamma(0)$. We can iterate this process to obtain a sequence of Mapper models $\Gamma(0), \Gamma(1), \ldots, \Gamma(r)$, where $\Gamma(i + 1)$ should be viewed as a "coarsening" of $\Gamma(i)$ or "lower resolution model" than $\Gamma(i)$. Just as we use pooling correspondences to pass from a higher resolution image to a lower resolution image, so we can now use the correspondences $\mathcal{C}(\Gamma(i), \Gamma(i + 1))$ as methods from passing to high resolution to lower resolution versions of the feature space for an arbitrary data matrix.

We show how this will work by constructing an analogue of the depth 6 generator constructed for MNIST in (4–3) above. We suppose that we have selected a metric on the column space of our data matrix, and further that we have built a Mapper model $\Gamma = \Gamma(0)$, together with the doublings $\Gamma(1)$

and $\Gamma(2)$. Further, we suppose we are trying to solve an N-outcome classification problem, where N was 10 in the actual MNIST case. As in the MNIST case, the generator will decompose as a product of a complete generator F^c and a structural generator F^s. The complete generator can be chosen arbitrarily. The analogue to the structural generator in (4–3) is given by the following.

$$X \xrightarrow{\varepsilon} \Gamma(0) \xrightarrow{\mathcal{C}(\Gamma(0),\Gamma(1))} \Gamma(1) \xrightarrow{\mathcal{C}(\Gamma(1),\Gamma(1))} \Gamma(1) \xrightarrow{\mathcal{C}(\Gamma(1),\Gamma(2))} \Gamma(2) \xrightarrow{\mathcal{C}^c} X(1) \xrightarrow{\mathcal{C}^c} X(N)$$

Unlike the data analytically driven neural nets, this construction has not yet been done but is in development.

Finally, we point out that one need not adhere rigidly to the doubling strategy described above. Choosing any families of coverings that are increasing, in the sense that l and s are both increasing, also can give families of correspondences that can act as replacements for pooling correspondences.

References

[1] G. Carlsson, *Topology and data*, Bull. Amer. Math. Soc. **46** (2009), 255-308

[2] G. Carlsson, T. Ishkhanov, V. de Silva, and A. Zomorodian, *On the local behavior of spaces of natural images*, Intl. Jour. Computer Vision, **76**, (2008), 1-12

[3] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. FeiFei. *Imagenet: A large-scale hierarchical image database*, In IEEE Conference on Computer Vision and Pattern Recognition, 2009.

[4] R.B. Gabrielsson and G. Carlsson, A look at the topology of convolutional neural networks, arXiv:1810.03234v1

[5] D. H. Hubel and T.N. Wiesel, *Receptive fields, binocular interaction and functional architecture in the cat's visual cortex*, Journal of Physiology, 160(1): 106-154, 1962.

[6] A. Krizhevsky, *Learning multiple layers of features from tiny images*, Technical report, University of Toronto, 2009.

[7] Y. LeCun, *The MNIST database of handwritten digits*, http://yann.lecun.com/exdb/mnist.

[8] A.B. Lee, K.S. Pedersen, and D. Mumford, *The non-linear statistics of high-contrast patches in natural images*, Intl. Jour.of Computer Vision, **54** (1-3), (2003) 83-103.

[9] A. Maleki, M. Shahram, and G. Carlsson, *Near optimal coder for image geometries*, Proc. IEEE Int. Conf. Image Processing (ICIP), San Diego, CA, 2008 (paper can be found at https://www.ece.rice.edu/ mam15/Kleinlet_fullversion)

[10] K. Priddy and P. Keller, **Artificial Neural Networks. An Introduction**, SPIE Press, 2005.

[11] M.Robinson, **Topological Signal Processing**, Springer Verlag, 2014.

[12] G. Singh, F. Mémoli, and G. Carlsson, *Topological methods for the analysis of high dimensional data sets and 3D object recognition*, SPBG, 2007, 91-100

[13] J. H. van Hateren and A. van der Schaaf, *Independent component filters of natural images compared with simple cells in primary visual cortex*, Proceedings of the Royal Society of London Series B, 265, 1998, 359-366.

[14] J. Perea and G. Carlsson, *A Klein Bottle-based dictionary for texture representation*, International Journal of Computer Vision, vol. 107, 75-97, 2014.

[15] K. Simonyan and A. Zisserman, *Very Deep Convolutional Networks for Large-Scale Image Recognition*, CoRR 1409.1556, 2014.

[16] http://ufldl.stanford.edu/housenumbers/