RotaTouille: Rotation Equivariant Deep Learning for Contours

Odin Hoff Gardaa
University of Bergen
odin.garda@uib.no

Nello Blaser
University of Bergen
nello.blaser@uib.no

Abstract

Contours or closed planar curves are common in many domains. For example, they appear as object boundaries in computer vision, isolines in meteorology, and the orbits of rotating machinery. In many cases when learning from contour data, planar rotations of the input will result in correspondingly rotated outputs. It is therefore desirable that deep learning models be rotationally equivariant. In addition, contours are typically represented as an ordered sequence of edge points, where the choice of starting point is arbitrary. It is therefore also desirable for deep learning methods to be equivariant under cyclic shifts. We present RotaTouille, a deep learning framework for learning from contour data that achieves both rotation and cyclic shift equivariance through complex-valued circular convolution. We further introduce and characterize equivariant non-linearities, coarsening layers, and global pooling layers to obtain invariant representations for downstream tasks. Finally, we demonstrate the effectiveness of RotaTouille through experiments in shape classification, reconstruction, and contour regression.

1 Introduction

Equivariance and invariance are the central concepts in geometric deep learning. Designing architectures that respect certain symmetries, often defined in terms of group actions, allows us to incorporate prior geometric knowledge about the data into the learning process. This can lead to models that generalize better, require less data, and are more efficient by reducing the effective hypothesis space. Equivariance is especially useful when the task requires the model to be sensitive to transformations of the input, such as translation, rotation, or permutation, while still producing consistent and meaningful output. Invariance, on the other hand, is desirable when the output should remain unchanged under transformations. Convolutional neural networks (CNNs) illustrate both concepts in image analysis. Convolutions are translation-equivariant, so shifting an input image shifts the feature maps accordingly, which is useful for segmentation, while applying global pooling after the convolutional layers achieves invariance, allowing classification to be insensitive to the object's position. Graph neural networks (GNNs) offer another example: they are often designed to be invariant or equivariant under permutations of node orderings (graph isomorphisms). A comprehensive overview of methods and concepts in geometric deep learning can be found in [9] and [17].

In this paper, we focus on what we will refer to as *contours*. Contours are complex-valued signals on finite cyclic groups, i.e., functions $\mathbb{Z}_n \to \mathbb{C}^k$, where $\mathbb{Z}_n$ denotes the cyclic group of order n, and $\mathbb{C}^k$ is the k-dimensional complex vector space consisting of k-tuples of complex numbers. See Fig. 1 for an illustration of a simple contour. Contours can serve as sparse representations of object boundaries, but they are not limited to simple closed shapes. Contours can also accommodate more general signals, including self-crossing curves and multichannel contours. For example, contours occur naturally as cell shapes in cell morphology [7, 10, 29], and as orbit plots in the vibrational analysis of rotating machinery [11, 21, 22]. Our method uses complex-valued neural networks, which employ complex-valued weights and are often employed in applications such as radar imaging [16, 34], MRI fingerprinting and reconstruction [14, 41], and other areas where data are naturally represented in the complex

O. H. Gardaa et al., RotaTouille: Rotation Equivariant Deep Learning for Contours. *Proceedings of the Fourth Learning on Graphs Conference (LoG 2025)*, PMLR 269, Arizona State University, Phoenix, USA, December 10–12, 2025.

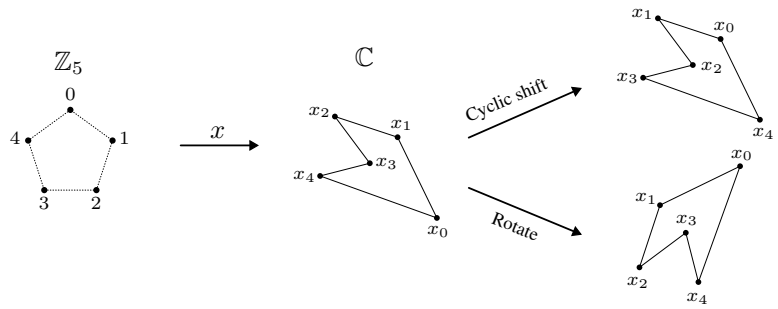

Figure 1: An illustration showing a contour $x\colon \mathbb{Z}_n \to \mathbb{C}^k$ where $n = 5$ and $k = 1$, and how a cyclic shift by one and a rotation by $\pi/2$ radians changes the image of x in $\mathbb{C}$. For readability, we write $x_i = x(i)$. For $k > 1$, one can think of the image of x as a stack of contours, one for each copy of $\mathbb{C}$.

domain. See [4, 27] for a detailed overview of complex-valued neural networks and their applications. Complex-valued CNNs have also been applied to images, showing self-regularizing effects [18].

1.1 Contributions

The group $G_n = \mathbb{Z}_n \times S^1$ acts on a contour $x\colon \mathbb{Z}_n \to \mathbb{C}^k$ by combining two operations: the cyclic group $\mathbb{Z}_n$ acts by cyclically *shifting* the starting point of the contour, and S^1 acts by *rotating* the image of x about the origin in each copy of the complex plane. These actions are illustrated in Fig. 1. We propose rotation- and cyclic shift-equivariant (as well as invariant) layers for deep learning on contours, leveraging complex-valued convolutions over the cyclic group (circular convolution). Convolution is well known for its cyclic shift-equivariance, and working in the complex domain also ensures equivariance to planar rotations. Our framework, **RotaTouille**, comprises the following main layer types:

- **Convolution layers.** Linear equivariant layers based on complex circular convolutions.
- **Activation functions.** Equivariant non-linear functions applied element-wise allowing the network to learn more complex functions.
- **Coarsening layers.** Equivariant local pooling layers downsampling the signal by coarsening the domain.
- **Invariant layers.** Global pooling layers producing real-valued invariant embeddings.

This layer taxonomy aligns well with the Geometric Deep Learning Blueprint proposed in [9, p. 29]. Throughout the paper, we show that the proposed layers are indeed equivariant and provide a classification of all possible equivariant non-linear activation functions. We evaluate RotaTouille in different tasks, including shape classification, shape reconstruction, and node-level curvature regression. Furthermore, we compare the effect of various design choices in an ablation study. Our implementation, including all code required to reproduce the experiments, is publicly available.[1]

1.2 Related Work

Shape analysis is a central topic in computer vision and machine learning, with early approaches often based on hand-crafted descriptors computed from contours. Examples include Curvature Scale Space (CSS) representations [1, 30] and generalized CSS (GCSS) [6], which were later used in the DeepGCSS neural network classifier [31]. The use of neural networks for contour data appeared already in [19], where the authors used fully connected neural networks to classify contours. Other classical works combined contour fragments with skeleton-based features to improve shape recognition [36–38]. The shape context (SC) descriptors [5], based on log-polar histograms, have also been shown to be effective in capturing the local geometric structure in a rotation-invariant way.

More recently, deep learning methods have become more popular for contour analysis. ContourCNN [15] models planar contours using real-valued circular convolutions on point sequences, with a custom pooling strategy to discard shape-redundant points. A similar convolutional approach was proposed in [28]. Two-dimensional CNNs have also been applied to contour images [3, 11, 22] and

[1] https://github.com/odinhg/rotation-equivariant-contour-learning

transformer-based architectures have been adapted for shape data using SC descriptors in [23]. None of these deep learning-based methods is intrinsically equivariant or invariant to rotations and they instead rely on data augmentation or manual feature extraction of invariant features. ShapeEmbed was recently introduced in [33] as a self-supervised variational autoencoder (VAE) that takes the pairwise distance matrix of contour points as input. It produces latent representations that are invariant to a fixed set of symmetries, including rotations and cyclic shifts.

Several works have explored learning rotation equivariant or invariant representations more directly, particularly for images and point clouds. The O2-VAE model [10] applies steerable CNNs [13] to cell images, encoding them into a latent space that is invariant to planar rotations. Although effective in capturing texture- and intensity-based features, such approaches operate on pixel grids rather than directly on contour data. Group-equivariant CNNs (G-CNNs) [12] provide a general framework for building networks equivariant to discrete symmetry groups, though they have been applied mostly to images. Closer in spirit to our approach are quaternion neural networks for 3D point clouds [39], where rotation-equivariant layers are constructed using the action of unit quaternions on $\mathbb{R}^3$. Topological methods have also been applied to contour data. Persistent homology has been used to extract stable shape features for morphological classification and comparison [7, 29].

2 Methodology

We begin by defining contours and the action of the group $\mathbb{Z}_n \times S^1$ on them, along with the concepts of equivariant and invariant maps. Subsequently, we introduce the various equivariant and invariant layers that form RotaTouille. For completeness, *groups* and *group actions* are defined in Section A.1.

2.1 Contours

For an integer $n > 0$, we let $\mathbb{Z}_n = \{0, 1, \dots, n-1\}$ denote the *cyclic group of order* n under addition modulo n. A *contour* is a function $x\colon \mathbb{Z}_n \to \mathbb{C}^k$ where $k > 0$. The set of all contours $X_n^k = \mathrm{map}(\mathbb{Z}_n, \mathbb{C}^k)$ is a vector space over $\mathbb{C}$ with addition and scalar multiplication defined pointwise. For any function $f\colon S \to \mathbb{C}^k$ where S is a set, we use the shorthand notation $f_j = \pi_j \circ f$ where $\pi_j\colon \mathbb{C}^k \to \mathbb{C}$ is the projection onto the j-th coordinate.

2.2 Group Actions and Equivariant Maps

The *circle group* $S^1 \subseteq \mathbb{C}$ consists of all unit complex numbers and the group product is given as complex multiplication. We consider the product group $G_n = \mathbb{Z}_n \times S^1$ and let it act on contours as follows: for $(l, w) \in G_n$ and a contour $x \in X_n^k$ we define $(l, w) \cdot x$ to be the contour mapping $q \mapsto wx(q - l)$. Here, w acts by scalar multiplication in $\mathbb{C}^k$ and $q - l$ is computed modulo n. We show that this does indeed define a (left) group action in Proposition A.3.

A function $f\colon X_n^k \to X_m^k$ where m is a divisor of n is called *equivariant* if $f((l, w) \cdot x) = (l, w) \cdot f(x)$ for all $x \in X_n^k$ and all $(l, w) \in G_n$. That is, the function commutes with our group action on contours. A function $f\colon X_n^k \to Y$ where Y is a set is called *invariant* if $f((l, w) \cdot x) = f(x)$ for all $x \in X_n^k$ and all $(l, w) \in G_n$. The concepts of invariance and equivariance are more general than described here and can be defined for any group action, including group representations (also known as linear actions). For a broader exposition, see [9, Chapter 3.1] or [42, Chapter 2.1].

2.3 Circular Convolution as Linear Equivariant Layer

If a map $T\colon X_n^1 \to X_n^1$ satisfies $T(wx) = wT(x)$ and $T(x + y) = T(x) + T(y)$ for all $w \in S^1$ and $x, y \in X_n^1$, then T is automatically $\mathbb{C}$-linear. This is a consequence of the fact that every complex number can be written as a (nonunique) finite sum of unit complex numbers. Now, suppose T also commutes with cyclic shifts, so T is any additive map that is equivariant with respect to our group action: $T((l, w) \cdot x) = (l, w) \cdot T(x)$. Then it is well known that T is necessarily the circular convolution operator. For completeness, we include a proof of this in Proposition A.4.

The *circular convolution* $*\colon X_n^1 \times X_n^1 \to X_n^1$ is defined by letting $(y * x)(q) = \sum_{j=0}^{n-1} y(j)x(q - j)$ for $x, y \in X_n^1$ and $q \in \mathbb{Z}_n$. In convolutional neural networks, we tend to work with kernels smaller than the signal. If $y \in X_m^1$ for some $m \leq n$, we extend y with zeros to X_n^1 before convolving with

x. That is, we let $y(q) = 0$ whenever $q \geq m$. We also want our equivariant layers to handle multi-channel signals, both as inputs and as hidden representations. By convention, we integrate information across the input channels by summing. Fixing $\phi \in X_m^k$, we define the *circular convolution operator* $\mathrm{Conv}_\phi \colon X_n^k \to X_n^1$ as

$$\mathrm{Conv}_\phi(x) = \sum_{j=1}^{k} \phi_j * x_j$$

for $x \in X_n^k$. We refer to ϕ as a *filter (or kernel)* that is typically learned during training, and m as the *kernel size*. Note that we do not use an additive bias term as this would break rotational equivariance. In practice, we often have multiple filters in each convolutional layer. Let $\Phi = (\phi^1, \dots, \phi^{k'})$ be a collection of k' filters, often referred to as a *filter bank*. The *convolutional layer* denoted $\mathbf{Conv}_\Phi \colon X_n^k \to X_n^{k'}$ is defined coordinate-wise by letting $\mathbf{Conv}_\Phi(x)_j = \mathrm{Conv}_{\phi^j}(x)$ for all $j = 1, \dots, k'$. We show that the convolutional layer satisfies the equivariance property. The proof that this satisfies equivariance is provided in Section A.2.

Proposition 2.1. *The convolutional layer $\mathbf{Conv}_\Phi \colon X_n^k \to X_n^{k'}$ is equivariant, that is, for all $(l, w) \in G_n$ and $x \in X_n^k$, we have $(l, w) \cdot \mathbf{Conv}_\Phi(x) = \mathbf{Conv}_\Phi((l, w) \cdot x)$.*

2.4 Non-linear Activation Functions

To learn more complex, non-linear functions on the space of contours, we need to introduce non-linear activation functions between the linear layers. We also want these activation functions to be equivariant. Any function $a \colon \mathbb{C} \to \mathbb{C}$ can be extended to a function $X_n^k \to X_n^k$ by point-wise application in each coordinate, and this function is equivariant if and only if $a(wz) = wa(z)$ for all $z \in \mathbb{C}$ and $w \in S^1$. We classify such functions in the following proposition proven in Section A.2:

Proposition 2.2. *A function $a \colon \mathbb{C} \to \mathbb{C}$ satisfies the equivariance condition $a(wz) = wa(z)$ for all $z \in \mathbb{C}$ and $w \in S^1$ if and only if there exists a function $g \colon [0, \infty) \to \mathbb{C}$ such that $a(z) = g(|z|)z$ for all $z \in \mathbb{C}$.*

Functions on this form already appear in the existing literature on complex-valued neural networks and we list some of them in Table 1.

Table 1: Examples of equivariant activation functions for contours.

Activation Function	**Choice of $g(r)$**
Siglog [41]	$(r+1)^{-1}$
ModReLU[2] [2, 43]	$\mathrm{ReLU}(r+b)r^{-1}$
Amplitude-Phase [20]	$\tanh(r)r^{-1}$

2.5 Coarsening Functions

A *coarsening (or local pooling) function* is an equivariant function $P \colon X_n^k \to X_m^k$ with $m < n$. In this section, we suppose that $n = mp$ for some integer $p > 1$ and think of p as the coarsening factor. We fix two positive integers n_s and n_d called *stride* and *dilation*, respectively, and let $\bigoplus \colon \mathbb{C}^p \to \mathbb{C}$ be any function such that $\bigoplus(wz) = w \bigoplus(z)$ for all $w \in S^1$ and $z \in \mathbb{C}^p$. For convenience, we write $\bigoplus_{j=0}^{p-1} z_j$ instead of $\bigoplus(z_0, \dots, z_{p-1})$. Examples include magnitude-based argmax $\bigoplus_{j=0}^{p-1} z_j = \arg\max_{j=0,\dots,p-1} |z_j|$, and the mean $\bigoplus_{j=0}^{p-1} z_j = p^{-1} \sum_{j=0}^{p-1} z_j$ for $z \in \mathbb{C}^p$. For a contour $x \in X_n^1$ we define the coarsening function $P_\oplus \colon X_n^1 \to X_m^1$ by letting

$$P_\oplus(x)(q) = \bigoplus_{j=0}^{p-1} x(qn_s + n_d j)$$

for all $q \in \mathbb{Z}_m$ and extend this to a function $P_\oplus \colon X_n^k \to X_m^k$ coordinate-wise. We refer to the case with $n_s = 1$ and $n_d = m$ as *coset pooling*. That is, we pool over the cosets $q + \langle m \rangle = \{q, q+m, \dots, q+(p-1)m\}$ for $q \in \mathbb{Z}_m$ and the function $P_\oplus$ is equivariant as we have

[2]The ModReLU activation function has a learnable bias parameter $b \in \mathbb{R}$.

$$P_{\oplus}((l,w)\cdot x)(q) = \bigoplus_{j=0}^{p-1} wx((q+mj)-l) = w\bigoplus_{j=0}^{p-1} x((q-l)+mj) = ((l,w)\cdot P_{\oplus}(x))(q).$$

For $n_s = p$ and $n_d = 1$, we get *strided pooling*, which is the most common type of pooling operation in CNNs. However, strided pooling is only approximately equivariant in the sense that cyclically shifting the input signal by lp, cyclically shifts the output signal by l:

$$P_{\oplus}((lp,w)\cdot x)(q) = \bigoplus_{j=0}^{p-1} wx((qp+j)-lp) = w\bigoplus_{j=0}^{p-1} x((q-l)p+j) = ((l,w)\cdot P_{\oplus}(x))(q).$$

In other words, strided pooling is equivariant with respect to the subgroup $p\,\mathbb{Z}_n \times S^1$ of G_n, but is true to the assumption that points close in the domain $\mathbb{Z}_n$ tend to be mapped to close points in $\mathbb{C}^k$. Coset pooling, on the other hand, is truly equivariant, but does not aggregate locally.

2.6 Invariant Feature Extraction

A *global pooling function* is an invariant map $A\colon X_n^k \to Y$ where Y is some set. We will only consider $Y = \mathbb{R}^k$. The idea then is that A aggregates channel-wise information into a real-valued contour embedding that can be used in downstream tasks. In the implementation, we use a combination of the mean and maximum of absolute values similar to what is done in [36]. Specifically, we define the global pooling function $A\colon X_n^k \to \mathbb{R}^k$ by setting the i-th component of $A(x)$ to be

$$\alpha n^{-1} \sum_{j=0}^{n-1} |x_i(j)| + (1-\alpha) \max_{j=0,\dots,n-1} |x_i(j)| \tag{1}$$

for $x \in X_n^k$ and where $\alpha \in [0,1]$ is a learned parameter.

3 Experiments

We perform various experiments to assess the feasibility of RotaTouille. This section outlines the details of preprocessing and implementation, introduces the datasets, and presents the results.

3.1 Preprocessing and Implementation Details

Data preprocessing. For image data, we extract contours through a three-step process:

1. **Binarization.** For grayscale images, we convert the images to binary images by applying thresholding, for example, by using Otsu's method [32].
2. **Contour extraction.** We use the OpenCV library [8] to extract contours from the binary images, selecting the one with the largest area in the case of multiple contours.
3. **Equidistant resampling.** We resample the contour to a fixed number of points, equidistantly with respect to the Euclidean distance.

See Fig. 2 for an illustration of the image-to-contour conversion process.

Before passing the contours to the model, they are centered at their mean and rescaled by dividing by the standard deviation of the magnitudes, as this showed to improve convergence of the complex-valued models. The image datasets are standardized using statistics computed on the training dataset.

Multi-scale invariant features. Choosing the optimal kernel size in 1D CNNs is not straightforward. One approach is to build an ensemble of models with different kernel sizes to capture features at multiple scales. While effective, this increases the number of parameters and computational cost. In our classification experiments, extracting invariant features at different network depths improved performance without adding learnable parameters. Figure 3 illustrates the structure of a convolutional block.

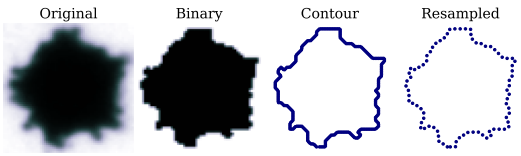

Figure 2: Image-to-contour conversion process. The input image is binarized, contours are extracted, and then resampled to a fixed length with equidistant points.

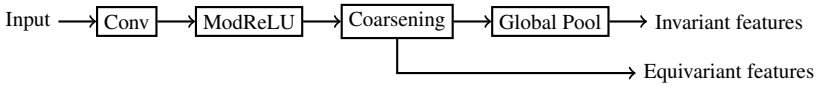

Figure 3: Multi-scale invariant feature extraction. After each convolutional block, a global pooling operation produces an invariant feature vector at that scale. Equivariant features are passed to the next block, and the final feature vector concatenates the invariant vectors from all depths.

Contour re-centering. Before every convolutional layer and global pooling layer, we re-center the contour channel-wise as this showed improvements in training stability and performance of our method. This re-centering is an equivariant operation.

Model architectures. For classification and regression, we use the ModReLU activation with strided local pooling and global pooling layers that aggregate via a learnable combination of mean and absolute-value maximum. These choices were guided by preliminary experiments on related datasets, which indicated robustness. The classification model has seven convolutional layers followed by a fully connected head, while the regression model uses four convolutional layers with a linear head. For the `RotatedMNIST` classification task, radial histogram features are concatenated with the invariant features from the global pooling layers before the classification head. The contour autoencoder has five convolutional layers in both encoder and decoder, including pooling and unpooling layers. Full architecture details are provided in Section A.4.

Training, model selection and evaluation. We use the Adam optimizer in all experiments and use 10% of the training data for validation to choose the best model for evaluation on the separate test data. For the classification tasks, the test data is randomly rotated to test the robustness of the candidate models. In this case, we also apply random rotation to augment the training data for the non-invariant baseline models to make the comparison fair. Each experiment is repeated ten times with different seeds, and we report mean test scores together with standard deviations. For more details on hyperparameter values for the different model and dataset combinations, see Table 5 in the appendix.

3.2 Datasets

Now, we describe the five datasets used in our experiments. We have three datasets for shape classification, one for shape reconstruction, and one for node-level regression.

Fashion MNIST. The Fashion MNIST dataset [45] contains $60{,}000$ training and $10{,}000$ test grayscale images of clothing items, each of size 28×28 pixels. We convert these images to contours by applying the image to contour conversion process described earlier. For the baseline CNN, we create two versions of the dataset: one with filled contours and one with unfilled contours. The filled contours are binary images where the contour is filled in, while the unfilled contours are binary images where only the contour is drawn. Hence, we discard all texture information in the images and only use the shape of the clothing items. We refer to this dataset as `FashionMNIST`.

ModelNet. We create a multi-channel dataset based on ModelNet [44], which contains 3D CAD models from various object categories. We select the classes `bottle`, `bowl`, `cone`, and `cup`, chosen for their tendency to form a single connected component in cross-section. For each model, we uniformly sample a point cloud and divide it into four disjoint volumes along the second axis. From each volume, we generate a 64×64 binary image by projecting points onto an orthogonal plane, then extract contours from the four image channels. Each sample thus consists of a stack of four contour slices. The final `ModelNet` dataset contains 644 training examples and 160 test examples.

Rotated MNIST. The rotated MNIST dataset [25] is a modified MNIST [26] where each image is randomly rotated by an angle uniformly sampled from $[0, 2\pi)$. It contains 12,000 training and 50,000 test examples of size 28×28 pixels, grayscale, with handwritten digits 0–9. We convert the images to contours using the process described above and include a simple rotation-invariant texture feature based on the radial histogram (RH). The RH captures the distribution of pixel intensities by dividing the image into 14 radial bins and counting pixels in each bin. We refer to this dataset as `RotatedMNIST`.

Cell Shapes. We use a subsample of the Profiling Cell Shape and Texture (PCST) benchmark dataset introduced [10] with 1000 shapes from each of the 9 categories corresponding to different combinations of eccentricity and boundary randomness. We extract contours and refer to this dataset as `PCST`.

Curvature contours. We construct a synthetic dataset for curvature regression. Given a continuous contour $\gamma\colon [0, 2\pi) \to \mathbb{C}$ where $\gamma(t) = x(t) + iy(t)$ with x and y twice differentiable, the *curvature* κ is defined in terms of the first and second derivatives as

$$\kappa = \frac{|x'y'' - y'x''|}{(x'^2 + y'^2)^{3/2}}.$$

The curvature measures the local deviation of the curve from a straight line. The `Curvature` dataset is generated as follows: First, we sample a number of modes m uniformly from $\{2, 3, 4, 5\}$. Then, for each mode $k = 1, \dots, m$, we sample four coefficients $a_k^x, b_k^x, a_k^y, b_k^y$ independently from the uniform distribution on $[-1, 1]$, and define the contour $\gamma = x + iy$ where

$$x(t) = \sum_{k=1}^{m} \left(a_k^x \cos(kt) + b_k^x \sin(kt)\right) \quad \text{and} \quad y(t) = \sum_{k=1}^{m} \left(a_k^y \cos(kt) + b_k^y \sin(kt)\right).$$

This construction ensures periodicity and smoothness while introducing controlled geometric variability through the random coefficients and number of modes. Lastly, we sample 100 points equidistant in arc length, and use the curvature in these points as the ground truth. See Fig. 4 for examples of generated curves. We discard contours with maximum curvature greater than 1000 to avoid extreme values. We generate 2000 contours for training and 1000 contours for test data.

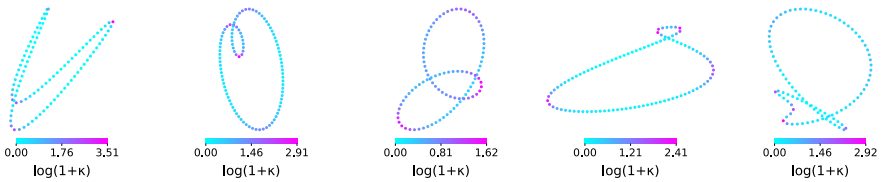

Figure 4: Five example contours from the `Curvature` dataset with log-curvature values colored.

3.3 Results

Shape classification. We evaluate RotaTouille on the classification datasets `FashionMNIST`, `ModelNet`, and `RotatedMNIST`. The cross-entropy loss function is used for training and performance is measured using test accuracy (or test error). As a first proof of concept, we compare RotaTouille with the baseline models on `FashionMNIST`. The baseline CNN uses contour images, and we test both filled and unfilled contours. We also include a baseline graph neural network based on the graph convolutional network (GCN) introduced in [24]. The graph is a fixed cycle graph with the Cartesian coordinates of the contour points as the node features. As an additional baseline, we implement the ContourCNN method using the optimal configuration reported in [15]. To evaluate RotaTouille on multichannel data, we compare it with the baseline models on `ModelNet`. The results are listed in Table 2, where RotaTouille slightly outperforms the baseline models on both datasets. The GCN model performs poorly on `FashionMNIST`, but is comparable to RotaTouille on the `ModelNet` dataset.

The `RotatedMNIST` dataset was originally designed to evaluate model robustness to rotations and has been widely used as a benchmark in prior work. We evaluate RotaTouille on this dataset to compare with existing methods. We report test error, defined as $1 -$ accuracy, directly quantifying

Table 2: Comparison of accuracies on the `FashionMNIST` and `ModelNet` contour datasets between RotaTouille and the baseline models. Accuracies are computed on the test dataset.

Model	**Accuracy**	
	`FashionMNIST`	`ModelNet`
CNN (filled contours)	0.849 ± 0.002	0.898 ± 0.032
CNN (unfilled contours)	0.852 ± 0.001	0.905 ± 0.012
GCN	0.626 ± 0.003	0.923 ± 0.027
ContourCNN	0.771 ± 0.004	0.849 ± 0.022
RotaTouille	0.867 ± 0.002	0.934 ± 0.016

misclassification and allowing comparison with prior works using the Rotated MNIST dataset. As shown in Table 3, using only contours yields a test error of 5.70%, which is not particularly competitive. However, augmenting our method with the simple radial histogram (RH) feature reduces the error to 3.72%, outperforming some of the other methods, including a image-based CNN used in [12]. While this does not match the accuracy of the most competitive methods, it demonstrates that a contour-based representation, when combined with a basic invariant feature, can achieve competitive performance on a challenging rotation benchmark.

Table 3: Comparison of test errors (in percent) on the `RotatedMNIST` dataset between existing methods and RotaTouille, both with and without the invariant radial histogram (RH) feature. Results marked with * are taken from previous papers (not re-implemented).

Method	**Test error** $(\%)$
SVM (RBF kernel)* [25]	10.38 ± 0.27
TIRBM* [40]	4.2
RC-RBM+Gradients IHOF* [35]	3.98
CNN* [12]	5.03 ± 0.0020
P4CNN* [12]	2.28 ± 0.0004
H-Net* [43]	1.69
GCN	48.44 ± 0.79
ContourCNN [15]	21.79 ± 1.12
RotaTouille (contours only)	5.70 ± 0.13
RotaTouille (contours + RH feature)	3.72 ± 0.11

Shape reconstruction. To demonstrate the flexibility of RotaTouille, we include an unsupervised learning task. Here, we train an autoencoder for contour reconstruction on the `PCST` dataset, and compare it to a similar architecture on binary images. We use the mean square error (MSE) as the reconstruction loss function. Using 10% of the dataset, we evaluate performance by visual inspection and compute the intersection over union (IoU) between the original shape and its reconstruction. For the contour-based model, we first convert the contours to binary images to compute the IoU scores.

See Fig. 5 for visualizations of the reconstructions. Both models are able to reconstruct the overall shapes, but seem to struggle with high frequency details. Based on the appearance of the reconstructions, our method is able to capture sharp corners better than the image-based model. This is particularly clear in example 2 and 3 of Fig. 5. Another advantage of the contour-based model is that it is guaranteed to produce valid contours, while the image-based model can produce invalid shapes with holes or multiple components. Moreover, it is not restricted to a fixed pixel grid. In terms of IoU, both models scores 0.97 on the validation data.

Curvature regression. We also consider a node-level regression task in which the goal is to predict the curvature at each point of a discrete contour from the `Curvature` dataset. For this task, we use the mean absolute error (MAE) as both the loss function and the evaluation metric. As baselines, we implement a 1D real-valued CNN and a graph-based GCN model operating on the Cartesian coordinates of the contour points, as well as a circle fitting method that estimates curvature by fitting a circle to every three consecutive contour points and computing the curvature from the fitted radius.

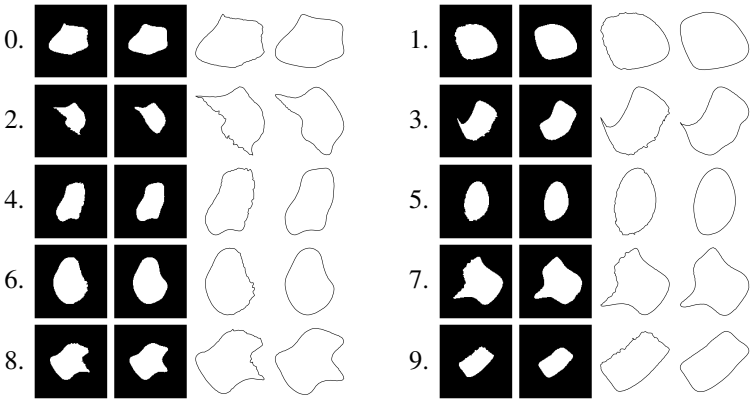

Figure 5: Ten randomly selected `PCST` validation examples (two per row). From left to right: original image, image-based autoencoder reconstruction, original contour, contour-based autoencoder reconstruction.

The baseline CNN employs circular convolutions like ContourCNN, but without priority pooling. As shown in Table 4, RotaTouille achieves the lowest MAE and highest R^2.

Table 4: Mean absolute errors (MAE) and R^2 scores on the test set for the curvature regression task.

Model	**MAE**	$\mathbf{R^2}$
Circle fitting	0.4411	0.1071
CNN (real-valued)	0.4479 ± 0.0046	0.2220 ± 0.0077
GCN	0.9063 ± 0.0003	-0.0064 ± 0.0001
RotaTouille	0.3944 ± 0.0092	0.2480 ± 0.0149

3.4 Ablation Study

We investigated how different design choices in our framework affect model performance. All ablation studies were conducted on the `FashionMNIST` contour dataset, with all other components and the architecture kept fixed as in the main experiments. We evaluate several configurable components of the model. Local pooling (coarsening) is tested with no pooling, or with strided and coset pooling using mean, max, or a learnable aggregation. We test the activation functions listed in Table 1. The sampling rate of contour points is varied across 16, 32, 64, 128, and 256, while kernel sizes range from 3 to 13 in odd increments. For global pooling, we compare mean, max, and a learnable combination as in Eq. (1). Numerical results are provided in Section A.5 in the appendix.

We observe minimal performance differences across configurations, except for the local pooling strategy: Despite its approximate equivariance, strided pooling consistently outperforms coset pooling, while omitting pooling yields similar accuracy but increases training time due to more contour points. Performance is largely robust to kernel size, while sampling rate has a small positive effect, with mean accuracy increasing from 0.855 (16 points) to 0.878 (256 points). This robustness likely reflects the simple shapes in `FashionMNIST`. For more complex shapes, appropriate choice of sampling rate and kernel sizes may be needed to capture high-frequency details.

4 Conclusion

We introduced a framework for deep learning on contours that is equivariant to rotations by defining an action of the group $\mathbb{Z}_n \times S^1$ on contours and constructing corresponding complex-valued convolutional layers. We also developed non-linear activation functions and pooling operations that preserve equivariance, as well as a global pooling layer to produce invariant features. While the performance gains are modest, RotaTouille provides a easy-to-implement framework that explicitly encodes rotational symmetry on contour data. This makes it a promising option for practical applications in shape analysis and other fields where rotation equivariance and invariance is essential.

References

[1] Sadegh Abbasi, Farzin Mokhtarian, and Josef Kittler. "Curvature scale space image in shape similarity retrieval". In: *Multimedia systems* 7 (1999), pp. 467–476. 2

[2] Martin Arjovsky, Amar Shah, and Yoshua Bengio. "Unitary evolution recurrent neural networks". In: *International conference on machine learning*. PMLR. 2016, pp. 1120–1128. 4

[3] Habibollah Agh Atabay. "Binary shape classification using convolutional neural networks". In: *IIOAB J* 7.5 (2016), pp. 332–336. 2

[4] Joshua Bassey, Lijun Qian, and Xianfang Li. "A survey of complex-valued neural networks". In: *arXiv preprint arXiv:2101.12249* (2021). 2

[5] Serge Belongie, Jitendra Malik, and Jan Puzicha. "Shape context: A new descriptor for shape matching and object recognition". In: *Advances in neural information processing systems* 13 (2000). 2

[6] Ameni Benkhlifa and Faouzi Ghorbel. "A novel 2D contour description generalized curvature scale space". In: *Representations, Analysis and Recognition of Shape and Motion from Imaging Data: 6th International Workshop, RFMI 2016, Sidi Bou Said Village, Tunisia, October 27-29, 2016, Revised Selected Papers 6*. Springer. 2017, pp. 129–140. 2

[7] Yossi Bokor Bleile, Patrice Koehl, and Florian Rehfeldt. "Persistence diagrams as morphological signatures of cells: A method to measure and compare cells within a population". In: *arXiv preprint arXiv:2310.20644* (2023). 1, 3

[8] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000). 5

[9] Michael M Bronstein et al. "Geometric deep learning: Grids, groups, graphs, geodesics, and gauges". In: *arXiv preprint arXiv:2104.13478* (2021). 1–3

[10] James Burgess et al. "Orientation-invariant autoencoders learn robust representations for shape profiling of cells and organelles". In: *Nature Communications* 15.1 (2024), p. 1022. 1, 3, 7

[11] R Caponetto et al. "Deep learning algorithm for predictive maintenance of rotating machines through the analysis of the orbits shape of the rotor shaft". In: *International Conference on Smart Innovation, Ergonomics and Applied Human Factors*. Springer. 2019, pp. 245–250. 1, 2

[12] Taco Cohen and Max Welling. "Group equivariant convolutional networks". In: *International conference on machine learning*. PMLR. 2016, pp. 2990–2999. 3, 8

[13] Taco S Cohen and Max Welling. "Steerable CNNs". In: *arXiv preprint arXiv:1612.08498* (2016). 3

[14] Elizabeth Cole et al. "Analysis of deep complex-valued convolutional neural networks for MRI reconstruction and phase-focused applications". In: *Magnetic resonance in medicine* 86.2 (2021), pp. 1093–1109. 1

[15] Ahmad Droby and Jihad El-Sana. "ContourCNN: convolutional neural network for contour data classification". In: *2021 International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)*. IEEE. 2021, pp. 1–7. 2, 7, 8, 17

[16] Jingkun Gao et al. "Enhanced radar imaging using a complex-valued convolutional neural network". In: *IEEE Geoscience and Remote Sensing Letters* 16.1 (2018), pp. 35–39. 1

[17] Jan E Gerken et al. "Geometric deep learning and equivariant neural networks". In: *Artificial Intelligence Review* 56.12 (2023), pp. 14605–14662. 1

[18] Nitzan Guberman. "On complex valued convolutional neural networks". In: *arXiv preprint arXiv:1602.09046* (2016). 2

[19] Lalit Gupta, Mohammad R Sayeh, and Ravi Tammana. "A neural network approach to robust shape classification". In: *Pattern Recognition* 23.6 (1990), pp. 563–568. 2

[20] Akira Hirose. "Complex-Valued Neural Networks: Distinctive Features". In: *Complex-Valued Neural Networks*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 17–56. ISBN: 978-3-642-27632-3. DOI: 10.1007/978-3-642-27632-3_3. 4

[21] Leonardo S Jablon et al. "Diagnosis of rotating machine unbalance using machine learning algorithms on vibration orbital features". In: *Journal of Vibration and Control* 27.3-4 (2021), pp. 468–476. 1

[22] Haedong Jeong et al. "Rotating machinery diagnostics using deep learning on orbit plot images". In: *Procedia Manufacturing* 5 (2016), pp. 1107–1118. 1, 2

[23] Qi Jia et al. “A rotation robust shape transformer for cartoon character recognition”. In: *The Visual Computer* (Oct. 2023). ISSN: 1432-2315. DOI: 10.1007/s00371-023-03123-2. URL: https://doi.org/10.1007/s00371-023-03123-2. 3

[24] TN Kipf. “Semi-Supervised Classification with Graph Convolutional Networks”. In: *arXiv preprint arXiv:1609.02907* (2016). 7, 17

[25] Hugo Larochelle et al. “An empirical evaluation of deep architectures on problems with many factors of variation”. In: *Proceedings of the 24th international conference on Machine learning*. 2007, pp. 473–480. 7, 8

[26] Yann LeCun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (2002), pp. 2278–2324. 7

[27] ChiYan Lee, Hideyuki Hasegawa, and Shangce Gao. “Complex-Valued Neural Networks: A Comprehensive Survey”. In: *IEEE/CAA Journal of Automatica Sinica* 9.8 (2022), pp. 1406–1426. DOI: 10.1109/JAS.2022.105743. 2

[28] Makrem Mhedhbi, Slim Mhiri, and Faouzi Ghorbel. “A new deep convolutional neural network for 2D contour classification”. In: (2022). 2

[29] Zoë Migicovsky et al. “Rootstock effects on scion phenotypes in a ‘Chambourcin’ experimental vineyard”. In: *Horticulture Research* 6.1 (May 2019), p. 64. ISSN: 2052-7276. DOI: 10.1038/s41438-019-0146-2. 1, 3

[30] Farzin Mokhtarian, Sadegh Abbasi, and Josef Kittler. “Efficient and robust retrieval by shape content through curvature scale space”. In: *Image databases and multi-media search*. World Scientific, 1997, pp. 51–58. 2

[31] Mallek Mziou-Sallami et al. “DeepGCSS: a robust and explainable contour classifier providing generalized curvature scale space features”. In: *Neural Computing and Applications* 35.24 (Aug. 2023), pp. 17689–17700. ISSN: 1433-3058. DOI: 10.1007/s00521-023-08639-1. URL: https://doi.org/10.1007/s00521-023-08639-1. 2

[32] Nobuyuki Otsu et al. “A threshold selection method from gray-level histograms”. In: *Automatica* 11.285-296 (1975), pp. 23–27. 5

[33] Anna Foix Romero et al. “ShapeEmbed: a self-supervised learning framework for 2D contour quantification”. In: *arXiv preprint arXiv:2507.01009* (2025). 3

[34] Theresa Scarnati and Benjamin Lewis. “Complex-valued neural networks for synthetic aperture radar image classification”. In: *2021 IEEE Radar Conference (RadarConf21)*. IEEE. 2021, pp. 1–6. 1

[35] Uwe Schmidt and Stefan Roth. “Learning rotation-aware features: From invariant priors to equivariant descriptors”. In: *2012 IEEE conference on computer vision and pattern recognition*. IEEE. 2012, pp. 2050–2057. 8

[36] Wei Shen et al. “Bag of shape features with a learned pooling function for shape recognition”. In: *Pattern Recognition Letters* 106 (2018), pp. 33–40. 2, 5

[37] Wei Shen et al. “Shape recognition by bag of skeleton-associated contour parts”. In: *Pattern Recognition Letters* 83 (2016), pp. 321–329. 2

[38] Wei Shen et al. “Shape recognition by combining contour and skeleton into a mid-level representation”. In: *Pattern Recognition: 6th Chinese Conference, CCPR 2014, Changsha, China, November 17-19, 2014. Proceedings, Part I 6*. Springer. 2014, pp. 391–400. 2

[39] Wen Shen et al. “3D-rotation-equivariant quaternion neural networks”. In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XX 16*. Springer. 2020, pp. 531–547. 3

[40] Kihyuk Sohn and Honglak Lee. “Learning invariant representations with local transformations”. In: *arXiv preprint arXiv:1206.6418* (2012). 8

[41] Patrick Virtue, X Yu Stella, and Michael Lustig. “Better than real: Complex-valued neural nets for MRI fingerprinting”. In: *2017 IEEE international conference on image processing (ICIP)*. IEEE. 2017, pp. 3953–3957. 1, 4

[42] Maurice Weiler et al. *Equivariant and coordinate independent convolutional networks*. World Scientific Singapore, 2023, p. 110. 3

[43] Daniel E Worrall et al. “Harmonic networks: Deep translation and rotation equivariance”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 5028–5037. 4, 8

[44] Zhirong Wu et al. “3D shapenets: A deep representation for volumetric shapes”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1912–1920. 6

[45] Han Xiao, Kashif Rasul, and Roland Vollgraf. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*. Aug. 28, 2017. arXiv: cs.LG/1708.07747 [cs.LG]. 6

A Appendix

A.1 Groups and Group Actions

We recall some basic definition from group theory used in this manuscript. A *group* G is a non-empty set together with a binary operation $G \times G \to G$ mapping $(g, h) \mapsto gh$ and a distinguished element $e \in G$ called the *identity element* of G such that the following identities hold:

Associativity. For all $g, h, k \in G$, we have $(gh)k = g(hk)$,

Identity. For all $g \in G$, we have $eg = g = ge$, and

Inverses. For all $g \in G$, there exists an $h \in G$ such that $gh = e = hg$. The element h is unique, and we usually denote it by g^{-1}.

Given two groups G and H, one can form the *product group* $G \times H$ consisting of all pairs (g, h) with $g \in G$ and $h \in H$. The binary operation is defined coordinate-wise as $(g_1, h_1)(g_2, h_2) = (g_1 g_2, h_1 h_2)$ for $g_1, g_2 \in G$ and $h_1, h_2 \in H$.

For a group G with identity element e, and a set S, a *(left) group action* of G on S is a map $G \times S \to S$ mapping $(g, x) \mapsto g \cdot x$ that satisfies the following two conditions:

Identity. For all $x \in S$, we have $e \cdot x = x$, and

Compatibility. For all $g, h \in G$ and $x \in S$, we have $g \cdot (h \cdot x) = (gh) \cdot x$.

Proposition A.3. *Let $G_n = \mathbb{Z}_n \times S^1$ and $X_n^k = \mathrm{map}(\mathbb{Z}_n, \mathbb{C}^k)$. The map $G_n \times X_n^k \to X_n^k$ mapping $((l, w), x) \mapsto (l, w) \cdot x$ where $((l, w) \cdot x)(q) = wx(q - l)$ for $q \in \mathbb{Z}_n$ is a group action.*

Proof. Let $x \in X_n^k$. The identity element in G_n is the element $(0, 1)$, so the map $e \cdot x$ sends every $q \in Z_n$ to $1x(q - 0) = x(q)$. For the compatibility condition, we let $(l, w), (l', w') \in G_n$, $q \in \mathbb{Z}_n$ and straight-forward computation shows that

$$\begin{aligned}(((l, w)(l', w')) \cdot x)(q) &= ((l + l', ww') \cdot x)(q) = ww' x(q - (l + l')) \\ &= w' w x(q - l - l') = ((l', w') \cdot ((l, w) \cdot x))(q).\end{aligned}$$

Since this holds for all $q \in \mathbb{Z}_n$, the maps $((l, w)(l', w')) \cdot x$ and $(l', w') \cdot ((l, w) \cdot x)$ are the same. □

A.2 Proofs

Proposition A.4. *If $T \colon X_n^1 \to X_n^1$ is $\mathbb{C}$-linear and commutes with cyclic shifts, then T is a circular convolution operator.*

Proof. Let $\delta_j \colon [n] \to \mathbb{C}$ be the Kronecker delta function defined by

$$\delta_j(q) = \begin{cases} 1 & \text{if } q = j, \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

Every $x \in X_n^1$ can be written uniquely as $x = \sum_{j=0}^{n-1} x(j)\delta_j$. If we write $S \colon X_n^1 \to X_n^1$ for the cyclic shift operator defined by $S(x)(q) = x(q-1)$, then we can express the cyclic shift equivariance as $T \circ S = S \circ T$. On basis elements, we have $S(\delta_j) = \delta_{j+1}$ so $\delta_j = S^j(\delta_0)$ for all $j \in \mathbb{Z}_n$. Thus, we can rewrite $x = \sum_{j=0}^{n-1} x(j) S^j(\delta_0)$ and since T is linear and commutes with S, we have that

$$T(x) = T\left(\sum_{j=0}^{n-1} x(j) S^j(\delta_0)\right) = \sum_{j=0}^{n-1} x(j) T(S^j(\delta_0)) = \sum_{j=0}^{n-1} x(j) S^j(T(\delta_0))$$

and hence $T(x)(q) = \sum_{j=0}^{n-1} x(j) y(q - j)$ where $y = T(\delta_0) \in X_n^1$. This is precisely the circular convolution of x and y. □

Proposition 2.1. *The convolutional layer $\mathbf{Conv}_\Phi \colon X_n^k \to X_n^{k'}$ is equivariant, that is, for all $(l, w) \in G_n$ and $x \in X_n^k$, we have $(l, w) \cdot \mathbf{Conv}_\Phi(x) = \mathbf{Conv}_\Phi((l, w) \cdot x)$.*

Proof. For $(l, w) \in G_n$, $q \in \mathbb{Z}_n$ and $\phi \in \Phi$, we have

$$((l,w) \cdot \mathrm{Conv}_\phi(x))(q) = w \sum_{j=1}^{k} (\phi_j * x_j)(q-l) = \sum_{j=1}^{k} w(\phi_j * x_j)(q-l)$$
$$= \sum_{j=1}^{k} (\phi_j * ((l,w) \cdot x_j))(q) = \mathrm{Conv}_\phi((l,w) \cdot x)(q)$$

for any $x \in X_n^k$. Since $\mathbf{Conv}_\Phi$ and the group action are defined coordinate-wise, we are done. □

Proposition 2.2. *A function $a\colon \mathbb{C} \to \mathbb{C}$ satisfies the equivariance condition $a(wz) = wa(z)$ for all $z \in \mathbb{C}$ and $w \in S^1$ if and only if there exists a function $g\colon [0, \infty) \to \mathbb{C}$ such that $a(z) = g(|z|)z$ for all $z \in \mathbb{C}$.*

Proof. If $a(z) = g(|z|)z$, then $a(wz) = g(|wz|)wz = wg(|z|)z = wa(z)$ since $|w| = 1$. Now, assume that a satisfies the equivariance condition. We have $a(0) = wa(0)$ for all $w \in S^1$, so $a(0)$ must be zero. If $z \neq 0$, write $z = rw$ with $r > 0$ and $w \in S^1$, and observe that $a(z) = a(rw) = wa(r) = zr^{-1}a(r)$. Define the function g by letting $g(0) = 0$ and $g(r) = r^{-1}a(r)$ whenever $r \neq 0$. □

A.3 Training Details

Table 5: Hyperparameters for the training of the models. All models were trained using the Adam optimizer with the specified learning rate. Hyperparameter values were chosen based on performance on validation data. The number of (real) parameters is an estimate of the model complexity. Random rotations were applied to improve generalization for the non-equivariant and non-invariant models.

Model	Dataset	LR	Batch Size	Epochs	Parameters	Data Aug.
CNN (2D)	`FashionMNIST`	0.01	128	200	294 666	Yes
GCN	`FashionMNIST`	0.001	128	200	74 826	Yes
ContourCNN	`FashionMNIST`	0.001	128	200	42 874	Yes
RotaTouille	`FashionMNIST`	0.0005	128	200	65 089	No
CNN (2D)	`ModelNet`	0.01	16	200	1 143 012	Yes
GCN	`ModelNet`	0.001	16	200	75 594	Yes
ContourCNN	`ModelNet`	0.001	16	200	42 964	Yes
RotaTouille	`ModelNet`	0.0005	16	200	64 747	No
GCN	`RotatedMNIST`	0.001	128	200	74 826	Yes
ContourCNN	`RotatedMNIST`	0.001	128	200	42 874	Yes
RotaTouille	`RotatedMNIST`	0.0005	128	200	65 089	No
RotaTouille + RH	`RotatedMNIST`	0.0005	128	200	66 881	No
CNN (2D)	`PCST`	0.001	32	200	5 315	Yes
RotaTouille	`PCST`	0.001	32	200	4 378	No
CNN (1D)	`Curvature`	0.001	32	100	34 033	Yes
GCN	`Curvature`	0.001	32	100	41 473	Yes
RotaTouille	`Curvature`	0.001	32	100	27 269	No

A.4 Model Details

Here, we provide detailed descriptions of the model architectures used in the experiments.

A.4.1 RotaTouille Models

Classification model. The *ConvBlock* used in the classifier consists of a sequence of operations that together form the basic building block of the model. The structure can be summarized schematically as

$$\text{ConvLayer} \to \text{ModReLU} \to \text{BatchNorm} \to (\text{Coarsening}) \to \text{Global pooling}.$$

The block begins with a *ConvLayer*, which performs a complex-valued circular convolution. This is followed by the equivariant activation function *ModReLU*. Next, *BatchNorm* is applied, operating only on the magnitudes (absolute values) of the complex activations to stabilize training. If the coarsening factor $p > 1$, a *Coarsening* step is applied, performing local pooling through a learnable combination of the mean and absolute-value maximum. The resulting contour is then forwarded to the next ConvBlock. Finally, a *Global pooling* layer produces real-valued invariant features using a learnable combination of the mean and absolute-value maximum, which are subsequently concatenated and used by the classification head. The full classifier architecture is listed in Table 6.

Table 6: RotaTouille classifier architecture used for the `FashionMNIST`, `ModelNet` and `RotatedMNIST` datasets. For the `ModelNet` dataset, the number of input channels in the first layer is 4 instead of 1.

Layer	Input $\to$ Output	Kernel	p	Notes
		Feature extractor		
ConvBlock	$1 \to 8$	9	1	
ConvBlock	$8 \to 8$	9	2	
ConvBlock	$8 \to 16$	9	1	
ConvBlock	$16 \to 16$	9	2	
ConvBlock	$16 \to 35$	9	1	
ConvBlock	$35 \to 35$	9	1	
ConvBlock	$35 \to 10$	9	1	
		Classifier head		
Linear	$m_1 + m_2 \to 128$	-	-	Followed by BatchNorm, Dropout (0.5) and ReLU. The number of invariant features from the global pooling layers is denoted by m_1, and m_2 is any additional invariant features such as radial pixel intensity histograms.
Linear	$128 \to n$	-	-	Where n is the number of classes.

Contour autoencoder model. The contour autoencoder used with the `PCST` dataset consists of an encoder and a decoder part. The *EncoderBlock* used in the autoencoder consists of a sequence of operations that progressively transform and coarsen the input representation. The structure can be summarized schematically as

$$\big(\text{ConvLayer} \to \text{BatchNorm} \to \text{Amplitude-Phase}\big) \times 2 \to \text{Coarsening}.$$

Each block consists of a *ConvLayer* performing complex-valued circular convolution, followed by *BatchNorm* on magnitudes and an *Amplitude-Phase* activation function as an equivariant nonlinearity. This sequence is repeated twice. Finally, *Coarsening* is applied using strided convolution with stride p and kernel size p. If the number of input and output channels match, a skip connection adds the input contour to the output. The *DecoderBlock* is identical to the encoder blocks, except the order is reversed, batch normalization is omitted and coarsening is replaced with strided transposed convolutions for unpooling. The full autoencoder architecture is described in Table 7.

Regression model. For the node-level regression model, we do not include any local pooling layers since we are predicting one curvature value per point in the input contour. We take absolute values before the regression head to obtain rotation invariant features, but still maintaining cyclic shift equivariance. The regression model used with the `Curvature` dataset is described in Table 8.

Table 7: RotaTouille contour autoencoder architecture.

Layer	Input → Output	Kernel	p
Encoder			
EncoderBlock	$1 \to 4$	11	2
EncoderBlock	$4 \to 4$	9	2
EncoderBlock	$4 \to 4$	7	2
EncoderBlock	$4 \to 4$	5	2
EncoderBlock	$4 \to 4$	3	2
Decoder			
DecoderBlock	$4 \to 4$	3	2
DecoderBlock	$4 \to 4$	5	2
DecoderBlock	$4 \to 4$	7	2
DecoderBlock	$4 \to 4$	9	2
DecoderBlock	$4 \to 1$	11	2

Table 8: RotaTouille regression model architecture.

Layer	Input → Output	Kernel	Notes
Equivariant feature extractor (complex-valued)			
ConvLayer	$1 \to 8$	5	Followed by ModReLU and BatchNorm.
ConvLayer	$8 \to 16$	5	Followed by ModReLU and BatchNorm.
ConvLayer	$16 \to 32$	5	Followed by ModReLU and BatchNorm.
ConvLayer	$32 \to 64$	5	Followed by ModReLU and BatchNorm.
Regression head (real-valued)			
Linear	$64 \to 1$	-	Element-wise absolute values as input.

A.4.2 Baseline Models

2D CNN classifier for images. The baseline 2D CNN classifier uses 2D convolutions with batch normalization, ReLU activations, and max pooling, followed by a fully connected classifier. The architecture is summarized in Table 9.

Table 9: Baseline 2D CNN classifier for images. For the `ModelNet` dataset, the number of input channels in the first layer is 4 instead of 1.

Layer	Input → Output	Kernel	Pool	Notes
Feature extractor				
Conv2d	$1 \to 32$	3	-	Followed by BatchNorm and ReLU.
Conv2d	$32 \to 64$	3	-	Followed by BatchNorm and ReLU.
MaxPool2d	-	-	2	
Conv2d	$64 \to 64$	3	-	Followed by BatchNorm and ReLU.
Conv2d	$64 \to 64$	3	-	Followed by BatchNorm and ReLU.
MaxPool2d	-	-	2	
Classifier head				
Linear	$m \to 64$	-	-	Followed by BatchNorm, Dropout (0.5) and ReLU. The embedding size m depends on the input size.
Linear	$64 \to n$	-	-	Where n is the number of classes.

2D CNN autoencoder. The 2D CNN autoencoder is composed of convolutional downsampling blocks, called *ConvBlock2d*, which consist of two consecutive sequences of convolution, batch normalization, and ReLU activation, followed by a downsampling operation. The upsampling counterpart, *DeconvBlock2d*, consists of an upsampling operation followed by two consecutive

sequences of ReLU activation and convolution. We use strided (transposed) convolutions for the upsampling and downsampling operations. The full architecture is summarized in Table 10.

Table 10: Baseline 2D CNN autoencoder.

Layer	Input → Output	Kernel	Pool / Upsample
	Encoder		
ConvBlock2d	$1 \to 4$	3	4
ConvBlock2d	$4 \to 4$	3	4
ConvBlock2d	$4 \to 8$	3	4
	Decoder		
DeconvBlock2d	$8 \to 4$	3	4
DeconvBlock2d	$4 \to 4$	3	4
DeconvBlock2d	$4 \to 1$	3	4

Graph convolutional network (GCN). The GCN classification and regression models are both based on GCN layers [24] for cyclic graphs with self-loops. The full architecture is summarized in Table 11. For regression on the `Curvature` dataset, we use $n_{\text{layers}} = 3$ and $n_{\text{layers}} = 5$ for classification tasks.

Table 11: Baseline GCN architectures for contour classification and node-level regression. For the `ModelNet` dataset, the number of input channels in the first layer is 8 instead of 2.

Layer	Input → Output	Notes
	Feature extractor	
GCNLayer	$2 \to 128$	Repeated n_{layers} times with ReLU activation.
	Classifier head	
GlobalMeanPool	$128 \to 128$	Global average pooling across contour points.
Linear	$128 \to 64$	Followed by BatchNorm, Dropout (0.5) and ReLU.
Linear	$64 \to n$	Where n is the number of classes.
	Regression head	
Linear	$128 \to 64$	Followed by BatchNorm, Dropout (0.5) and ReLU.
Linear	$64 \to 1$	

ContourCNN and 1D CNN. The ContourCNN is used for contour classification, while the 1D CNN is used for node-level regression. Each *CircConvBlock* in the ContourCNN consists of a circular 1D convolution, batch normalization, ReLU activation, and priority pooling as in [15]. Both networks operate on real-valued contour coordinates.

Table 12: Baseline ContourCNN classifier architecture used for contour classification. For the `ModelNet` dataset, the number of input channels in the first layer is 8 instead of 2.

Layer	Input → Output	Kernel	Notes
	Feature extractor		
CircConvBlock	$2 \to 32$	3	Priority pooling to $l = 40$.
CircConvBlock	$32 \to 64$	3	Priority pooling to $l = 30$.
CircConvBlock	$64 \to 128$	3	Priority pooling to $l = 20$.
	Classifier head		
GlobalMeanPool	$128 \to 128$	-	Global average pooling across contour points.
Linear	$128 \to 80$	-	Followed by BatchNorm, Dropout (0.5) and ReLU.
Linear	$80 \to n$	-	Where n is the number of classes.

Table 13: Baseline 1D CNN regressor architecture used for node-level regression.

Layer	Input → Output	Kernel	Notes
		Feature extractor	
Conv1d	$2 \to 16$	5	Followed by BatchNorm and ReLU.
Conv1d	$16 \to 32$	5	Followed by BatchNorm and ReLU.
Conv1d	$32 \to 64$	5	Followed by BatchNorm and ReLU.
Conv1d	$64 \to 64$	5	Followed by BatchNorm and ReLU.
		Regression head	
Linear	$64 \to 1$	-	

A.5 Ablation Study Results

Here, we list all numerical results for the ablation study in Section 3.4.

Table 14: Comparison of test accuracies on the `FashionMNIST` dataset for different choices of **local pooling (coarsening) functions** and aggregation functions. Note that in some training runs, coset pooling led to numerical instabilities (NaN losses), preventing convergence. For consistency, we report mean accuracy scores only over the successfully converged.

Pooling type	Aggregation	Accuracy
No Pooling	–	0.869 ± 0.015
Strided	Mean	0.868 ± 0.002
Coset	Mean	0.827 ± 0.025
Strided	Max	0.865 ± 0.003
Coset	Max	0.842 ± 0.003
Strided	Combined	0.867 ± 0.002
Coset	Combined	0.841 ± 0.001

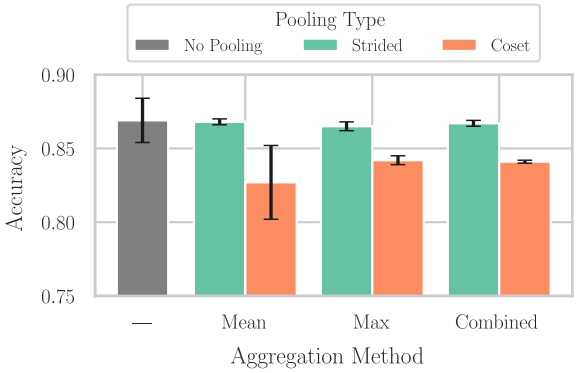

Figure 6: A bar plot visualizing the numerical results in Table 14.

Table 15: Comparison of test accuracies on the `FashionMNIST` dataset for different choices of **non-linear activation functions**. See Table 1 for definitions of the activation functions used.

Activation function	**Accuracy**
Siglog	0.868 ± 0.002
ModReLU	0.867 ± 0.002
Amplitude-Phase $(\tanh)$	0.869 ± 0.001

Table 16: Comparison of test accuracies on the `FashionMNIST` dataset for different choices of the **number of sampled contour points**.

Contour points	**Accuracy**
16	0.855 ± 0.003
32	0.866 ± 0.003
64	0.873 ± 0.002
128	0.877 ± 0.002
256	0.878 ± 0.002

Table 17: Comparison of test accuracies on the `FashionMNIST` dataset for different choices **kernel sizes**.

Kernel size	**Parameters**	**Accuracy**
3	33 997	0.870 ± 0.003
5	44 361	0.875 ± 0.002
7	54 725	0.876 ± 0.002
9	65 089	0.876 ± 0.002
11	75 453	0.875 ± 0.002
13	85 817	0.876 ± 0.002

Table 18: Comparison of test accuracies on the `FashionMNIST` dataset for different choices of the aggregation function used on the contour points' absolute values in the **global pooling** layer for producing invariant representaion.

Aggregation function	**Accuracy**
Mean	0.864 ± 0.003
Max	0.869 ± 0.002
Combined	0.867 ± 0.002