Supplementary Material: A Simple Strategy to Provable Invariance via Orbit Mapping

Kanchana Vaishnavi Gandikota[1], Jonas Geiping[2], Zorah Lähner[1], Adam Czapliński[1], Michael Möller[1]

[1] University of Siegen,[2] University of Maryland

1 Extension of Orbit Mapping to Equivariant Networks

The *equivariance* of $\mathcal{G}$ preserves the structure of transformations $g \in S$ of input data in the elements $y \in \mathcal{Y}$ (including, but not limited to, the case where $\mathcal{X} \equiv \mathcal{Y}$). The *equivariance* of $\mathcal{G}$ to S is defined as

$$\mathcal{G}(g(x);\theta) = g(\mathcal{G}(x;\theta)) \quad \forall x \in \mathcal{X},\ g \in S,\ \theta \in \mathbb{R}^p. \tag{1}$$

We now show that equivariant networks can be designed by applying all transformations in S to the input x.

Proposition 1. *Let S define a group action on $\mathcal{X}$. A network $\mathcal{G}$ is equivariant under the group action of S if it can be written as*

$$\mathcal{G}(x;\theta) = \mathcal{G}_1(\{g(\mathcal{G}_2(g^{-1}(x);\theta_2)) \mid g \in S\};\theta_1) \tag{2}$$

for some other arbitrary network $\mathcal{G}_2 : \mathcal{X} \times \mathbb{R}^{p_2} \to \mathcal{X}$, and a network $\mathcal{G}_1 : 2^{\mathcal{X}} \times \mathbb{R}^{p_1} \to \mathcal{X}$ that commutes with any element $h \in S$, i.e., for $h \in S$, and $Z \subset \mathcal{X}$, it satisfies $\mathcal{G}_1(h(Z);\theta_2) = h(\mathcal{G}_1(Z;\theta_2))$, where $h(Z)$ denotes the set obtained by the applying h to every element of Z.

Proof. We want to show that a network satisfying the condition (5) is equivariant. Let $h \in S$ be arbitrary. Note that

$$\{g \mid g \in S\} = \{h^{-1}g \mid g \in S\} \tag{3}$$

such that a substitution of variables from $g \in S$ to $z = h^{-1}g \in S$ (i.e., $g = hz$ and $z^{-1} = g^{-1}h$) yields

$$\begin{aligned}&\{g(\mathcal{G}_2(g^{-1}(h(x));\theta_2)) \mid g \in S\}\\ =&\{h(z(\mathcal{G}_2(z^{-1}(x);\theta_2))) \mid z \in S\}.\end{aligned}$$

This means that we can also write

$$\begin{aligned}\mathcal{G}(h(x);\theta) &= \mathcal{G}_1(\{h(z(\mathcal{G}_2(z^{-1}(x);\theta_2))) \mid z \in S\};\theta_1)\\ &= \mathcal{G}_1(h(\{z(\mathcal{G}_2(z^{-1}(x);\theta_2)) \mid z \in S\});\theta_1)\\ &= h(\mathcal{G}_1(\{z(\mathcal{G}_2(z^{-1}(x);\theta_2)) \mid z \in S\});\theta_1)\\ &= h(\mathcal{G}(x;\theta))\end{aligned}$$

which yields the desired equivariance under the assumed commutative property.

The work [2] can be interpreted as an instance of the construction in Proposition 1, where equivariant linear layers w.r.t. rotations by 90 degrees are obtained by choosing $\mathcal{G}_2$ to be a simple convolution and $\mathcal{G}_1$ to be the summation over all (finitely many) elements of the set. Subsequently, they nest these layers with component-wise (and therefore inherently equivariant) non-linearities.

While Proposition 1 is stated for general groups, realizations of such constructions often rely on the ability to list an entire orbit of the group. In the following we show an efficient solution to obtain equivariant networks using orbit mapping.

Proposition 2 (Orbit mapping for equivariant networks). *Let h be an orbit mapping that satisfies $h(S \cdot x) \in S \cdot x$ for all x. Any network $\mathcal{G} : \mathcal{X} \times \mathbb{R}^p \to \mathcal{X}$ that can be written as*

$$\mathcal{G}(x;\theta) = \hat{g}^{-1}(\mathcal{G}_2(\hat{g}(x);\theta)) \tag{4}$$

for an arbitrary network $\mathcal{G}_2 : \mathcal{X} \times \mathbb{R}^p \to \mathcal{X}$ and $\hat{g} \in S$ denoting the element that satisfies $\hat{g}(x) = h(S \cdot x)$ is equivariant.

Proof. We want to show that a network satisfying the condition (4) is equivariant. Consider an input $a = r(x)$ to the network, where r denotes an arbitrary element of S. We first need to determine the element $\tilde{g} \in S$ such that $\tilde{g}(a) = h(S \cdot a)$. From the definition of the orbit, it follows that $S \cdot x = S \cdot r(x)$, such that our orbit mapping satisfies remains the same, i.e., $h(S \cdot x) = h(S \cdot a) = \hat{g}(x)$. Solving the equation $\tilde{g}(a) = \hat{g}(x)$ with $a = r(x)$, i.e., $x = r^{-1}(a)$ for $\tilde{g}$ yields $\tilde{g} = \hat{g}r^{-1}$. Now it follows that

$$\begin{aligned}\mathcal{G}(r(x);\theta) = \mathcal{G}(a;\theta) &= \tilde{g}^{-1}(\mathcal{G}_2(\tilde{g}(a);\theta)) \\ &= r(\hat{g}^{-1}(\mathcal{G}_2(\tilde{g}(a);\theta))) \\ &= r(\hat{g}^{-1}(\mathcal{G}_2(\hat{g}(x);\theta))) \\ &= r(\mathcal{G}(x;\theta)),\end{aligned}$$

which concludes the proof.

2 A Discussion on Isometry Invariance

Here, we will elaborate on how the functional map framework [29] can be seen as an application of our orbit mapping for isometry invariance. Functional maps are a widely used method to find correspondences between isometric shapes, and we will show here that the framework fits within our proposed theory. Non-rigid correspondence is a notoriously hard problem, and joint optimization within a larger framework makes it even more complex. To resolve this the idea of functional maps is to change the representation of the correspondence from point-wise to function-wise. By choosing the eigenfunctions of the Laplace-Beltrami operator [31] as the basis for functions on the shapes, the problem becomes a least squares problem aligning suitable descriptor functions in the space of functions.

Here, $F \in \mathcal{F}(\mathcal{X})$ and $G \in \mathcal{F}(\mathcal{Y})$ are descriptor functions on the shapes $\mathcal{X}$ and $\mathcal{Y}$ respectively. They are assumed to take similar values on corresponding points on $\mathcal{X}, \mathcal{Y}$, and generate the designated orbit element within our framework. These descriptors are projected onto the eigenfunctions of $\mathcal{X}, \mathcal{Y}$, named Φ, Ψ respectively. These projections are the chosen elements of the orbit we will align, and, for isometries and sufficiently comparable descriptors, the projections can be aligned by an orthogonal transformation generating the group action which is exactly the functional map C. The vanilla functional map optimization looks like this:

$$\arg\min_{C \in O(k)} \|C\Phi^{-1}F - \Psi^{-1}G\|_2^2 \tag{5}$$

Functional maps are often used when shape correspondence is required within another framework, and has been used in many deep learning applications [7],[16],[22]. Due to its wide application, we will not provide extra experiments to show its efficacy but want to emphasize that this is a possible implementation of our theory.

3 Stability of gradient based orbit mapping

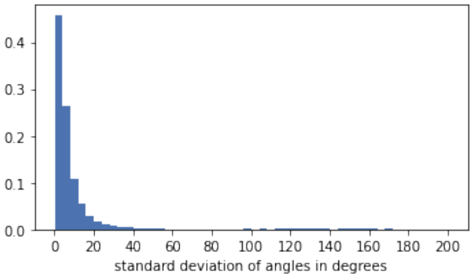
In this section we analyze the stability of the proposed gradient based orbit mapping strategy for discrete images. While the proposed gradient based orbit mapping our approach leads to unique orientation as long as $\int_Z \nabla u(z)\ dz$ is non-zero, practically, the magnitude of $\int_Z \nabla u(z)\ dz$ and interpolation artifacts affect the stability of the orbit mapping. While one could possibly use forward or central differences to calculate gradients at pixels along approximate circles, this further deteriorates the stability of orbit mapping. This is seen in Tab. 1 a) which shows the mean standard deviation orientation of orbit-mapped images when input images rotated in steps of 1 degree using bilinear interpolation. We find that using forward differences to approximate the gradient has the most instability. In the following section, we derive a necessary condition for provable invariance using general convolution kernels (instead of gradients in x and y direction), where we show that forward differences does not satisfy these conditions for any rotation.

Tab. 1 b) shows the histogram of standard deviations in orientation for CIFAR10 images when calculating exact gradients along the circle. The standard deviations of predicted orientations of over 78% of the images is less than 10 degrees, and over 44% of images is less than 4 degrees, indicating a relatively stable orbit mapping for these images. However, a fraction of images also have a higher variance, in predicted orientation possibly due to small values of the integral. Tab. 1 c) shows that our gradient based orbit mapping is fairly robust to small additive Gaussian noise.

a)

Dataset	Exact	Central Diff.	Forward Diff
CIFAR10	10.46	12.47	23.89
CUB200	9.05	14.56	24.75

b)



c)

Dataset	clean	σ^2=0.01	σ^2=0.05	σ^2=0.1
CIFAR10	10.46	11.36	14.08	16.69
CUB200	9.05	10.55	15.99	20.610

Table 1: Stability and robustness of proposed gradient based Orbit Mapping strategy. a) The mean standard deviation values of angles in degrees over the images in dataset are reported when rotating images based on exact gradients computed along circle using bilinear interpolation, and approximate gradients using finite differences along pixels closest to the circle. b) The histogram of standard deviations of the predicted orientation in degrees for CIFAR10. c) The mean standard deviation values of angles in degrees over the images in CIFAR10 dataset are reported, for different levels of additive Gaussian noise.

4 Invariance to image rotations using convolution kernels

Following the notation from the paper, let $u(z)$ denote the continuous image function with $z \in \mathbb{R}^2$ representing the spatial coordinates of an image. The invariance set for the orbit of continuous image rotations is

$$S = \{g : \mathcal{X} \to \mathcal{X} \mid g(u)(z) = (u \circ r(\alpha))(z), \text{ for } \alpha \in \mathbb{R}\},$$
$$\text{and } r(\alpha) = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{pmatrix} \text{ is the rotation matrix.}$$

Let us consider two kernels $k_i : \mathbb{R}^2 \to \mathbb{R}$, $i = \{1, 2\}$. We now investigate the convolution of a kernel with a rotated image $(u \circ r(\alpha))(z)$

$$\begin{aligned} (k_i * u \circ r(\alpha))\,(z) &= \int_{\mathbb{R}^2} k_i(x)(u \circ r(\alpha))(z - x)dx \\ &= \int_{\mathbb{R}^2} k_i(x)u(r(\alpha)z - r(\alpha)x)dx \\ &= \int_{\mathbb{R}^2} k_i(r^T\varphi)u(r(\alpha)z - \varphi)d\varphi \\ &\qquad \text{with } \varphi = r(\alpha)x \end{aligned}$$

Now assume

$$\begin{pmatrix} k_1(r^T(\alpha)\varphi) \\ k_2(r^T(\alpha)\varphi) \end{pmatrix} = r^T(\alpha) \begin{pmatrix} k_1(\varphi) \\ k_2(\varphi) \end{pmatrix}. \qquad (6)$$

Then

$$\begin{pmatrix} (k_1 * (u \circ r(\alpha)))\,(z) \\ (k_2 * (u \circ r(\alpha)))\,(z) \end{pmatrix} = \int_{\mathbb{R}^2} r^T(\alpha) \begin{pmatrix} k_1(\varphi) \\ k_2(\varphi) \end{pmatrix} u(r(\alpha)z - \varphi) d\varphi.$$

$$= r^T(\alpha) \begin{pmatrix} (k_1 * u)(r(\alpha)z) \\ (k_2 * u)(r(\alpha)z) \end{pmatrix}$$

Then for a suitable set Z which makes the integral rotationally invariant, (e.g. circles around image center)

$$\int_Z \begin{pmatrix} (k_1 * (u \circ r(\alpha)))\,(z) \\ (k_2 * (u \circ r(\alpha)))\,(z) \end{pmatrix} dz = r^T(\alpha) \int_Z \begin{pmatrix} (k_1 * u)(\varphi) \\ (k_2 * u)(\varphi) \end{pmatrix} d\varphi \tag{7}$$

And we can determine the optimal rotation as solution to

$$\hat{g} = \arg\max_{g \in S} \left\langle \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \int_Z \begin{pmatrix} k_1 * u \\ k_2 * u \end{pmatrix} (z)\ dz \right\rangle \tag{8}$$

whose solution is given by $\hat{\alpha}$ such that

$$\begin{pmatrix} \cos\hat{\alpha} \\ \sin\hat{\alpha} \end{pmatrix} = \frac{\int_Z \begin{pmatrix} k_1 * u \\ k_2 * u \end{pmatrix} (z)\ dz}{\left\| \int_Z \begin{pmatrix} k_1 * u \\ k_2 * u \end{pmatrix} (z)\ dz \right\|} \tag{9}$$

We can see that (6) is a necessary condition to ensure invariance to image rotations using orbit mapping with (9) employing convolution kernels k_1 and k_2. For discrete convolution kernels, eq. (6) is not exactly satisfied for arbitrary rotations due to discretization problem. We can deduce necessary conditions on discrete kernels k_1 and k_2 to satisfy eq. (6) for rotations in multiples of 90^o. For square kernels k_1 and k_2 of size $N \times N$, we find that

$$k_1[i,j] = k_1[N-i+1, N-j+1] \text{ and} \tag{10}$$

$$k_2 = k_1 \circ r(-90^o) \tag{11}$$

are necessary to satisfy the condition (6) for $\alpha = 90^o$.
For $N = 2$, this gives kernels of the form

$$k_1 = \begin{pmatrix} a & b \\ -b & -a \end{pmatrix} \text{ and } k_2 = \begin{pmatrix} -b & a \\ -a & b \end{pmatrix}$$

For $N = 3$,

$$k_1 = \begin{pmatrix} a & b & c \\ d & 0 & -d \\ -c & -b & -a \end{pmatrix} \text{ and } k_2 = \begin{pmatrix} -c & d & a \\ -b & 0 & b \\ -a & -d & c \end{pmatrix}$$

Note that computing gradients using central differences satisfies (10) and (11), whereas using forward differences does not satisfy these conditions. Therefore, we observe more instabilities in orbit mapping when forward differences are used for gradient computation, see Tab. 1.

5 Details about the Experimental Setting

In the following we provide the detailed training settings used in our experiments.

5.1 Rotation invariance for images

For our experiments with image rotational invariance, we used Pytorch(v.1.8.1), python(v.3.8.8), torchvision(v.0.9.1). The exact training protocol is provided below.

CIFAR10 We trained a Resnet18 [4] on the CIFAR 10 dataset, using stochastic gradient descent with initial learning rate 0.1, momentum 0.9, and weight decay 5e-4. Additionally, we trained a small Convnet and a linear model which used an initial learning rate of 0.01. For all the models, the learning rate is decayed by a factor of 0.5 whenever the validation loss does not decrease for 5 epochs. Training data is augmented using random horizontal flips, random crops of size 32 after zero-padding by 4 pixels. We divide the training data into train (80%) and validation (20%) sets. Networks are trained for 150 epochs with a batch size of 128 and we report the results on the test set using the model with best validation accuracy. The experiments with CIFAR10 were performed partially on a machine with one Nvidia TITAN RTX, and partially on machine with 4 NVIDIA GeForce RTX 2080 GPUs.

HAM10000 We fine-tuned an imagenet pretrained[1] NFNet-F0 [1] on HAM10000 dataset [9]. The dataset is split into 8912 train and 1103 validation images using stratified split, ensuring there are no duplicates with the same lesion ids in the train and validation sets. Training data is augmented using random horizontal and vertical flips and color jitter, and randomly oversample the minority classes to mitigate class imbalance. The network is finetuned for 5 epochs, with a batch size of 128 and learning rate of 1e-4, weight decay of 5e-4 using Adam optimizer [5] with exponential learning rate decay, with factor 0.2. For training using TI-pool which uses 4 rotated copies of images, we reduce the batch size to 32 to fit the GPU memory. For experiments with STN we use a 3 layered CNN with convolution filers of size 3×3 followed by 2 fully connected layers for pose prediction. For experiment with ETN we use a CNN with 4 conv layers with 64 channels and 2 fully connected layers for pose prediction. We report results using final iterate on the validation set. The experiments with HAM10000 dataset were partially performed on a machine with one NVIDIA TITAN RTX card, and partially on machine with 4 NVIDIA GeForce RTX 2080 GPUs.

CUB200 This is a small dataset containing 11,788 images of birds, split into 5994 images for training and 5794 test images. Since training a network from scratch gives low accuracies (around 35% clean accuracy with Resnet-50), we instead perform finetuning using an imagenet pretrained Resnet-50 from pytorch torchvision (v.0.9.1) on CUB-200 dataset [10]. The training data is augmented using random horizontal flips, random resized crops of size 224. The network

---

[1] pretrained model from `https://github.com/rwightman/pytorch-image-models` licensed Apache 2.0

is finetuned for 60 epochs with batch size of 128 and initial learning rate of 1e-4, using Adam optimizer [5] , weight decay of 5e-4, with exponential learning rate decay, with factor 0.9. . For training using TI-pool which uses4 rotated copies of images, we reduce the batch size to 64 to fit in the GPU memory. For experiment with ETN we use a CNN with 4 conv layers with 64 channels and 2 fully connected layers for pose prediction. We report the accuracies using the final iterate on the test set. The experiments on CUB-200 dataset were performed on machine with 4 NVIDIA GeForce RTX 2080 GPUs.

All the three image datasets including HAM10000 dataset [9] used in our experiments are publicly available and widely used in machine learning literature. To the best of our knowledge these do not contain offensive content or personally identifiable information.

5.2 Rotation and Scale invariance for 3D point clouds

We investigate invariance to rotations and scale for 3D point clouds with the task of point cloud classification on the *modelnet40* dataset [11]. For this dataset note the asset descriptions at `https://modelnet.cs.princeton.edu/`: "All CAD models are downloaded from the Internet and the original authors hold the copyright of the CAD models. The label of the data was obtained by us via Amazon Mechanical Turk service and it is provided freely. This dataset is provided for the convenience of academic research only." We use the resampled version of `shapenet.cs.stanford.edu/media/modelnet40_normal_resampled.zip`. We follow the hyperparameters of [7,8] with improvements from the implementation of [12] on which we base our experiments. We train a standard PointNet for 200 epochs with a batch size of 24 with Adam [5] with base learning rate of 0.001, weight decay of 0.0001. During training we sample 1024 3D points from every example in *modelnet40*, randomly scale with a scale from the interval $[0.8, 1.25]$, and randomly translate by an offset of up to 0.1 - if not otherwise mentioned in our experiments. This is the training procedure proposed in [12]. However, we always train the model for the the full 200 epochs and report final *class* accuracy based on the final result - we do not report instance accuracy. We further report invariance tests based on the final model.

As described in the main body, we evaluate rotational invariance by testing on 16×16 regularly spaced angles from $[0, 2\pi]$, rotating along xy and yz axes. We evaluate scaling invariance by testing the scales $\{0.001, 0.01, 0.1, 0.5, 1.0, 5.0, 10, 100, 1000\}$. All experiments for this dataset were run on three single GPU office machines, containing an NVIDIA TITAN Xp, and two GTX 2080ti, respectively.

6 Additional Numerical Results

6.1 Invariance to continous image rotations

Discretization effects in CUB200 We further investigate the effect of discretization using different interpolation schemes for rotation on higher resolution on the CUB-200 dataset (trained at 224x224 resolution) fine-tuned using Resnet-50. Tab. 2 shows the results of different training schemes with and without our orbit mapping (OM) obtained when using different interpolation schemes for rotation. Besides standard training (*Std.*), we use rotation augmentation (RA), and the adversarial training and regularization from [3,13]. Even for this higher resolution dataset, the worst-case accuracies between different types of interpolation may differ by more than 15%.

Train	OM	Clean.	Average			Worst-case		
			Nearest	Bilinear	Bicubic	Nearest	Bilinear	Bicubic
Std.	✗	**77.41±0.33**	37.67±0.35	52.45±0.29	51.87±0.31	3.19±0.49	8.07±0.35	8.16±0.33
	✓Train+Test	71.19±0.34	63.35±0.30	71.56±0.34	70.93±0.35	40.63±0.48	58.80±0.39	59.02±0.41
RA.	✗	69.89±0.28	67.61±0.33	70.12±0.34	68.83±0.37	34.88±0.47	41.01±0.41	40.50±0.43
	✓Test	69.41±0.31	69.19±0.32	69.27±0.29	68.53±0.38	48.63±0.43	56.28±0.39	55.86±0.40
	✓Train+Test	70.35±0.46	69.41±0.23	70.72±0.18	70.37±0.34	47.92±0.26	57.54±0.39	57.62±0.14
Advers.	✗	64.54±0.17	53.74±0.65	64.07±0.25	63.22±0.54	26.63±0.79	42.82±0.60	42.44±0.55
Mixed	✗	68.56±0.46	57.17±0.60	65.91±0.42	65.76±0.51	28.06±0.58	42.87±0.32	42.92±0.38
Advers.-KL	✗	64.47±0.35	53.93±0.35	64.65±0.26	64.02±0.34	26.94±0.46	43.04±0.63	42.61±0.37
Advers.-ALP	✗	64.63±0.31	55.56±0.67	64.34±0.17	63.21±0.24	29.55±0.69	43.63±0.21	43.48±0.32
ETN	✗	64.14±0.24	64.26±0.65	66.95±0.42	64.32±0.62	43.33±1.01	52.85±1.12	49.72±1.31
TIpool	✗	76.80±0.25	60.67±0.79	74.90±0.15	74.82±0.24	36.06±1.12	59.04±0.37	59.50±0.41
TIpool-RA	✗	73.47±0.48	72.30±0.51	74.71±0.29	73.65±0.36	57.22±0.64	62.82±0.56	62.31±0.42
TIpool	✓Train+Test	76.82±0.15	68.50±0.58	**77.18±0.18**	**77.04±0.16**	49.85±0.65	**69.19±0.36**	**69.64±0.33**
TIpool-RA	✓Train+Test	74.78±0.20	**73.79±0.48**	75.89±0.17	75.07±0.16	**59.57±0.57**	67.78±0.20	67.64±0.18

Table 2: Effect of augmentation and including gradient based orbit mapping *(OM)* on robustness to rotations with different interpolations for CUB200 classification using Resnet50. Shown are clean accuracy on standard test set and average and worst-case accuracies on rotated test set. Mean and standard deviations over 5 runs are reported.

In particular, adversarial training with bi-linear interpolation is still more vulnerable to image rotations with nearest neighbor interpolation. Even the learned ETN also exhibits similar behavior. While our approach is also affected by the interpolation effects, the vulnerability to nearest neighbor interpolation is ameliorated when using rotation augmentation. We obtain best results using orbit mapping in conjunction with the discrete invariant approach [6]

Effect of Network architecture for CIFAR10 To investigate the effectiveness of our approach, we experiment three different network architectures: *i) a linear network, ii) a 5-layer convnet ii) a Resnet18.* We compare the performance of our orbit mapping approach with training schemes, i.e. augmentation and adversarial training for rotational invariance in Tab. 3. For all the three architectures considered, our orbit mapping together with rotation augmentation consistently results in the most accurate predictions in the worst case.

Comparing Computation Complexity for CIFAR10 In Tab. 4, the training times using different approaches are compared for rotation-invariant CIFAR10 classification. It can be noted that the proposed gradient based orbit mapping is significantly easier and computationally cheaper to train in comparison with other approaches for incorporating invariance. In contrast, adversarial training is the most computationally expensive approach.

Network	Train	OM	Std.	Average			Worst-case		
				Nearest	Bilinear	Bicubic	Nearest	Bilinear	Bicubic
Linear	Std.	✗	**38.89±0.17**	25.31±0.21	25.57±0.22	25.48±0.24	2.50±0.11	3.56±0.17	3.26±0.11
		✓Train+Test	31.87±0.10	**31.25±0.04**	**31.58±0.05**	**31.33±0.04**	13.08±0.23	18.85±0.21	18.21±0.21
	RA	✗	29.73±0.18	30.66±0.03	30.77±0.03	30.72±0.03	14.30±0.42	18.31±0.29	16.94±0.37
		✓Test	30.60±0.13	30.52±0.07	30.65±0.08	30.54±0.09	16.83±0.47	21.17±0.28	20.37±0.26
		✓Train+Test	31.06±0.26	31.07±0.11	31.27±0.10	31.13±0.09	**19.19±0.28**	**24.25±0.31**	**23.68±0.31**
	Advers.	✗	28.82±0.77	29.46±0.60	29.62±0.56	29.36±0.56	11.45±0.81	14.20±0.93	13.65±0.55
Convnet	Std.	✗	**86.12±0.33**	32.01±0.32	35.97±0.26	38.15±0.36	0.85±0.09	0.57±0.06	0.89±0.14
		✓Train+Test	76.13±0.96	64.34±0.35	71.21±0.96	**74.61±0.84**	25.78±0.49	49.60±0.79	55.57±0.81
	RA	✗	75.03±0.99	71.77±0.84	65.45±0.66	70.22±0.66	27.96±0.50	27.06±0.61	32.51±0.53
		✓Test	70.12±0.64	67.64±0.55	61.03±0.67	66.09±0.71	39.01±0.57	42.88±0.90	49.39±0.68
		✓Train+Test	74.30±0.77	**73.24±0.58**	69.52±0.53	73.38±0.59	**46.25±0.54**	**53.36±0.57**	**59.04±0.53**
	Advers.	✗	72.96±0.95	62.08±0.59	**74.29±0.88**	73.86±0.76	26.24±0.43	50.99±0.54	52.46±0.51
Resnet18	Std.	✗	**93.98±0.32**	35.12±0.81	40.06±0.44	42.81±0.50	0.79±0.38	1.31±0.13	2.22±0.17
		✓ Train+Test	87.99±0.43	72.40±0.33	**84.12±0.55**	**86.61±0.49**	34.57±0.94	68.60±0.81	74.49±0.84
	RA	✗	85.54±0.72	80.47±0.74	75.99±0.72	79.47±0.65	45.50±0.83	44.71±0.74	50.50±0.78
		✓ Test	79.26±0.42	74.93±0.51	69.31±0.65	73.94±0.63	48.93±0.75	52.18±0.91	58.69±0.78
		✓ Train+Test	85.40±0.57	**84.37±0.58**	81.82±0.59	84.82±0.52	**66.22±0.75**	**71.09±1.01**	**76.44±0.89**
	Advers.	✗	69.32±1.61	61.73±1.12	68.54±0.68	68.00±0.31	36.95±0.97	50.21±0.55	49.73±0.98

Table 3: Comparing rotational invariance using training schemes vs. orbit mapping for CIFAR10 classification using *i) Linear network ii) 5-layer Convnet iii) Resnet18*. Shown are the mean clean accuracy and the average and worst case accuracies when test images are rotated in steps of 1 degree. The mean and standard deviation values over 5 runs are reported.

Method	Std.	STN	ETN	Adv.	OM
Train-time/epoch	18.05±0.05	18.90±0.05	18.89±0.07	72.09±0.18	18.59±0.04

Table 4: Average training time per epoch in seconds for different approaches to incorporate rotation invariance, with Resnet18 as base architecture for CIFAR10 classification. Training time correspond to runs on a machine with single Titan-RTX GPU.

Comparing Computational Complexity of ROTMNIST Tab. 5 compares the computational complexity of the D4/C4 and D16/C16 models. The D16/C16 model has significantly higher computational complexity than the D4/C4 model, though the number of learnable parameters is nearly same. The network size of D16/C16 network s higher due to more rotated copies of the filters, resulting in larger training and inference times. Orbit mapping adds no learnable parameters and increases training time very marginally (∼0.3 seconds/epoch). Training times correspond to runs on a machine with single Titan-RTX GPU.

References

1. Brock, A., De, S., Smith, S.L., Simonyan, K.: High-performance large-scale image recognition without normalization. arXiv preprint arXiv:2102.06171 (2021)
2. Cohen, T., Welling, M.: Group equivariant convolutional networks. In: International conference on machine learning. pp. 2990–2999 (2016)

OM	D4/C4 Train-time/epoch	D16/C16 Train-time/epoch
✗	4.47 s	41.89 s
✓	4.78 s	42.08 s

Table 5: Comparing computational complexity of D4/C4 and D16/C16 models. Orbit mapping adds no learnable parameters and increases training time very marginally (∼0.3 seconds/epoch). Training times correspond to runs on a machine with single Titan-RTX GPU.

3. Engstrom, L., Tran, B., Tsipras, D., Schmidt, L., Madry, A.: Exploring the landscape of spatial robustness. In: International Conference on Machine Learning. pp. 1802–1811 (2019)
4. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
5. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
6. Laptev, D., Savinov, N., Buhmann, J.M., Pollefeys, M.: Ti-pooling: transformation-invariant pooling for feature learning in convolutional neural networks. In: IEEE conference on computer vision and pattern recognition. pp. 289–297 (2016)
7. Qi, C.R., Su, H., Mo, K., Guibas, L.J.: Pointnet: Deep learning on point sets for 3d classification and segmentation. In: Conference on Computer Vision and Pattern Recognition (CVPR) (2017)
8. Qi, C.R., Yi, L., Su, H., Guibas, L.J.: Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In: Advances in Neural information processing systems (2017)
9. Tschandl, P., Rosendahl, C., Kittler, H.: The ham10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions. Scientific data **5**(1), 1–9 (2018)
10. Wah, C., Branson, S., Welinder, P., Perona, P., Belongie, S.: The caltech-ucsd birds-200-2011 dataset (2011)
11. Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., Xiao, J.: 3d shapenets: A deep representation for volumetric shapes. In: IEEE conference on computer vision and pattern recognition. pp. 1912–1920 (2015)
12. Yan, X.: Pointnet/pointnet++ pytorch. https://github.com/yanx27/Pointnet_Pointnet2_pytorch (2019)
13. Yang, F., Wang, Z., Heinze-Deml, C.: Invariance-inducing regularization using worst-case transformations suffices to boost accuracy and spatial robustness. Advances in Neural information processing systems pp. 14757–14768 (2019)