Rotation equivariant vector field networks

Diego Marcos *[1], Michele Volpi[1], Nikos Komodakis[2], and Devis Tuia[1]
[1]University of Zurich, [2]Ecole des Ponts, Paris Tech

Abstract

In many computer vision tasks, we expect a particular behavior of the output with respect to rotations of the input image. If this relationship is explicitly encoded, instead of treated as any other variation, the complexity of the problem is decreased, leading to a reduction in the size of the required model.

In this paper, we propose the Rotation Equivariant Vector Field Networks (RotEqNet)*, a Convolutional Neural Network (CNN) architecture encoding rotation equivariance, invariance and covariance. Each convolutional filter is applied at multiple orientations and returns a vector field representing magnitude and angle of the highest scoring orientation at every spatial location. We develop a modified convolution operator relying on this representation to obtain deep architectures. We test RotEqNet on several problems requiring different responses with respect to the inputs' rotation: image classification, biomedical image segmentation, orientation estimation and patch matching. In all cases, we show that RotEqNet offers extremely compact models in terms of number of parameters and provides results in line to those of networks orders of magnitude larger.*

1. Introduction

In many real life problems, such as overhead (aerial or satellite) or biomedical image analysis, there are no dominant up-down or left-right relationships. For example, when detecting cars in aerial images, the object's absolute orientation is not a discriminant feature. If the absolute orientation of the image is changed, e.g. by following a different flight-path, we would expect the car detector to score the exact same values over the same cars, just in their new position on the rotated image, independently from their new orientation along the image axes. In this case, we say that the problem is rotation *equivariant*: rotating the input is expected to result in the same rotation in the output. On the other hand, if we were confronted with a classification setting in which we are only interested in the presence or absence of cars in the whole scene, the classification score should remain the same, no matter the absolute orientation of the input scene. In this case the problem is rotation *invariant*. The more general case would be rotation *covariance*, in which the output changes as a function of the rotation of the input, with some predefined behavior. Taking again the cars example, a rotation covariant problem would be to retrieve the absolute orientation of cars with respect to longitude and latitude: in this case, a rotation of the image should produce a change of the predicted angle.

*Corresponding author: diego.marcos@geo.uzh.ch

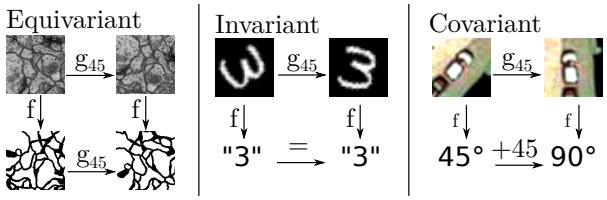

Figure 1. Desirable behaviors with respect to rotation of the inputs: (left) equivariance in segmentation; (center) invariance in classification; (right) covariance in absolute orientation estimation. g_{45} is an operator that rotates the input image by $45°$.

Throughout this article we will make use of the terms equivariance, invariance and covariance of a function $f(\cdot)$ with respect to a transformation $g(\cdot)$ in the following sense:

- **equivariance:** $f(g(\cdot)) = g(f(\cdot))$,
- **invariance:** $f(g(\cdot)) = f(\cdot)$,
- **covariance:** $f(g(\cdot)) = g'(f(\cdot))$,

where $g'(\cdot)$ is a second transformation, which is itself a function of $g(\cdot)$. With the above definitions, equivariance and invariance are special cases of covariance. We illustrate these properties in Fig. 1.

In this paper, we propose a CNN architecture that naturally encodes these three properties: RotEqNet. In the following, we will recall how CNNs achieve translation invariance, before discussing our own proposition.

1.1. Dealing with translations in CNNs

The success of CNNs is partly due to the translation equivariant nature of the convolution operation. The convolution of an image $\mathbf{x} \in \mathbb{R}^{M \times N \times d}$ with a filter $\mathbf{w} \in$

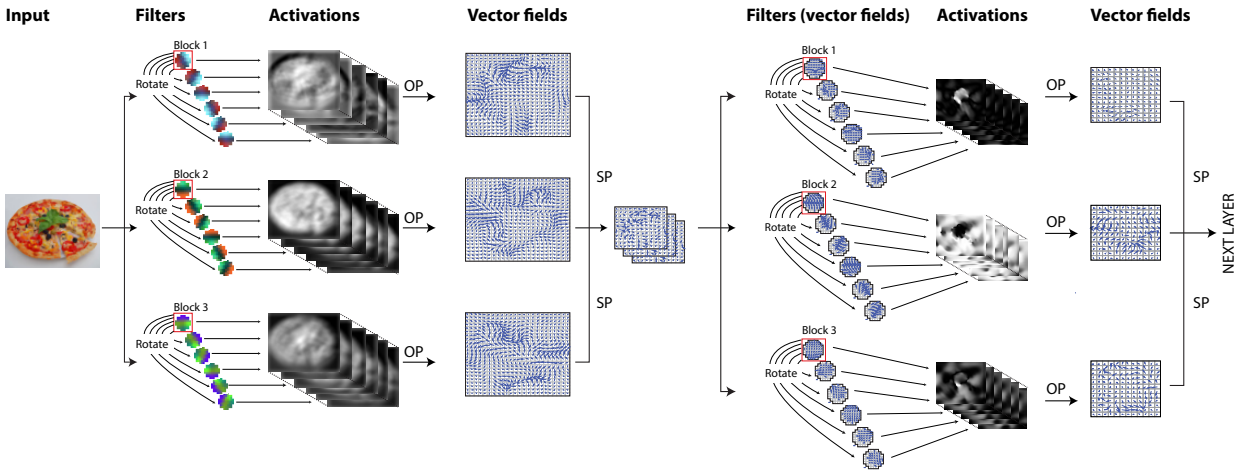

Figure 2. Example of the first two layers of RotEqNet. Each layer learns only three canonical filters (red squares) and replicates them across six orientations. The output of the first block are three vector field maps, which are further convolved by vector field filters in the second block (OP: orientation pooling; SP: spatial pooling).

$\mathbb{R}^{m \times n \times d}$, written $\mathbf{y} = \mathbf{w} * \mathbf{x}$, is obtained by applying the same scalar product operation over all overlapping $m \times n$ windows (unit stride) on $\mathbf{x}$. If $\mathbf{x}$ undergoes an integer translation in the horizontal and vertical directions by (p, q) pixels, the same pixel neighborhoods in $\mathbf{x}$ will exist in the translated $\mathbf{x}$, but again translated by (p, q) pixels. Therefore, any operation involving fixed neighborhoods such as the convolution is translation equivariant.

A crucial consequence of learning convolution weights is a drastic reduction in the number of parameters. Without the translation equivariance assumption, each local window would have a different set of weights. Forcing weights to be shared across locations, known as *weight tying*, reduces the number of learnable parameters proportionally to the number of pixels in the image and hardcodes translation equivariance within the model. This fact is vital for the applicability of deep neural networks to images [18].

1.2. Incorporating rotation equivariance in CNNs

RotEqNet shows similar advantageous characteristics when dealing with rotations: by encoding equivariance, we are able to strongly reduce the number of parameters while keeping similar or better accuracy across different tasks.

However, applying the exact same reasoning of weight tying for rotations is not straightforward. To follow the same logic, one should apply R rotated versions of each convolutional filter, resulting in R feature maps per filter. The dimensionality of subsequent filters would therefore increase with R, strongly increasing model size and requirements for runtime memory usage.

One way of reducing the size of the model while keeping rotation equivariance would be to propagate only the maximum value occurring across R feature maps. However, deeper layers would have no information about the orientation of features at previous layers.

We propose a trade-off between these two approaches by keeping the maximum value across the R feature maps, but in the form of a 2D vector field that captures its *magnitude* and *orientation* and propagates it through all the layers of the network.

2. Related work

Two families of approaches explicitly account for rotation invariance or equivariance: 1) those that transform the representation (image or feature maps) and 2) those that rotate the filters. RotEqNet belongs to the latter.

1) Rotating the inputs: Jaderberg *et al*. [14] propose the Spatial Transformer layer, which learns how to crop and transform a region of the image (or a feature map) before passing it to the next layer. This transforms relevant regions into a canonical form, improving the learning process by reducing geometrical appearance variations in subsequent layers. TI-pooling [16] inputs several rotated versions of a same image to the same CNN and then performs pooling across the different feature vectors at the first fully connected layer. Such scheme allows another subsequent fully connected layer to choose among rotated inputs to perform classification. Cheng *et al*. [5] employ in every minibatch several rotated versions of the input images. Their representations after the first fully connected layer are then encouraged to be similar, forcing the CNN to learn rotation invariance. Henriques *et al*. [11] warp the images such that the translation equivariance inherent to convolutions is transformed into rotation and scale equivariance.

On the one hand, these methods have the advantage of exploiting conventional CNN implementations, since they only act on data representations. On the other hand, they can only consider global transformations of the input images. While this is well suited for tasks such as image

classification, it limits their applicability to other problems (e.g. semantic segmentation), where the local relative orientation of certain objects with respect to surroundings is what matters. Instead, RotEqNet is based on specific CNN building blocks designed to deal with local orientation information. Therefore, RotEqNet can approach diverse tasks such as classification, fully convolutional semantic segmentation, detection and regression.

It is worth mentioning that standard data augmentation strategies belong to this first family. They rely on random rotations and flips of the training samples [24]: given abundant training samples and enough model capacity, a CNN might learn that different orientations should score the same by learning equivalent filters at different orientations [19]. Unlike this, RotEqNet is well suited for problems with limited training samples that can profit from reduced model sizes, since the behavior with respect to rotations is hard-coded and it does not need to be learned.

2) Rotating the filters: Gens and Domingos [10] tackle the problem of the exploding dimensionality (discussed in Sec. 1.2) by applying learnable pooling operations and sampling the symmetry space at each layer. This way, they avoid applying the filters exhaustively across the (high dimensional) feature maps by selectively sampling few rotations. By doing so, only the least important information is lost from layer to layer. Cohen *et al*. [6, 7] use a smaller symmetry group, composed of a flipping and four $90°$ rotations and perform pooling within the group. They apply it only in deeper layers, since they found that pooling in the early layers discards important information and harms the performance. Instead of explicitly defining a symmetry group, Ngiam *et al*. [21] pool across several untied filters, thus letting the network learn the type of invariance. Sifre *et al*. [23] use hand crafted wavelets that are separable in the roto-translational space, allowing for more efficient computations. Another approach to avoid the dimensionality explosion is to limit the depth of the network: Sohn *et al*. [26] and Kivinen *et al*. [15] propose such a scheme with Restricted Boltzmann Machines (RBM), while Marcos *et al*. [20] consider supervised CNNs consisting of a single convolutional layer.

These works find a compromise between the computational resources required and the amount of orientation information kept throughout the layers, by either keeping the model shallow or accounting for a limited amount of orientations. With RotEqNet, we avoid such compromise by pooling multiple orientations and passing forward both the maximum magnitude *and* the orientation at which it occurred. This modification allows to build deep rotation equivariant architectures, in which deeper layers are aware of the dominant orientations. At the same time, the dimensionality of feature maps and filters is kept low by discarding information about non-maximum orientations, thus reducing memory requirements.

The most similar approaches to RotEqNet are the recently proposed Harmonic Networks (H-Nets) [29] and Oriented Response Networks (ORN) [31], both of which use an enriched feature map explicitly capturing the underlying orientations. They do so by using either complex circular harmonics (H-Nets) or the full vector of oriented responses (ORN). H-Nets offer a very compact feature map, but are limited to learning filters that are a combination of circular harmonic wavelets. On the other hand, ORN allows to learn arbitrary filters, but relies on a much less compact representation of the feature maps, leading to heavier models both in terms of size and memory requirements. RotEqNet provides the best of both worlds: the compactness of the former with the flexibility of the latter. These properties make it particularly suitable to address problems characterized by limited training samples, as we will see in the experiments.

3. Rotation equivariant vector field networks

We focus on achieving rotation equivariance by performing convolutions with several rotated instances of the same *canonical* filter (see Fig. 2). The canonical filter $\mathbf{w}$ is rotated at R different evenly spaced orientations.In the experiments (Sec. 4) we deal with problems requiring either full invariance, equivariance or covariance, so we use the interval $\boldsymbol{\alpha} = [0°, 360°]$. However, this interval can be adapted to a known range of tilts. The output of the filter $\mathbf{w}$ at a specific location consists of the magnitude of the maximal activation across the orientations and the corresponding angle. If we convert this polar representation into Cartesian coordinates, each filter $\mathbf{w}$ produces a vector field feature map $\mathbf{z} \in \mathbb{R}^{H \times W \times 2}$, where the output of each location consists of two values $[u, v] \in \mathbb{R}^2$ implicitly encoding the maximal activation in both magnitude and direction. Since the feature maps have become vector fields, from this moment on the filters must also be vector fields, as seen in the right part of Fig. 2.

The advantage of representing $\mathbf{z}$ in Cartesian coordinates is that the horizontal and vertical components $[u, v]$ are orthogonal, and thus a convolution of the two vector fields can be computed on each component independently using standard convolutions (see Eq. (5)).

3.1. RotEqNet building blocks

RotEqNet requires specific building blocks to handle vectors fields as inputs and/or outputs (Fig. 2). In the following, we present our reformulation of traditional CNN blocks to account for both vector field activations and filters. The implementation[1] is based on the MatConvNet [27] toolbox[2].

[1] Will be made available at http://github.com/di-marcos/RotEqNet
[2] http://www.vlfeat.org/matconvnet

3.1.1 Rotating convolution (RotConv)

Given an input image with $m/2$ zero-padding $\mathbf{x} \in \mathbb{R}^{H+m/2 \times W+m/2 \times d}$, we apply the filter $\mathbf{w} \in \mathbb{R}^{m \times m \times d}$ at R orientations, corresponding to the angles:

$$\alpha_r = \frac{360}{R} r \quad \forall r = 1, 2 \dots R. \tag{1}$$

Each one of these rotated versions of the canonical filters (highlighted by red squares in Fig. 2) is computed by re-sampling $\mathbf{w}$ with bilinear interpolation after rotation of α_r degrees around the filter's center.

$$\mathbf{w}^r = g_{\alpha_r}(\mathbf{w}), \tag{2}$$

where g_α is the α degrees rotation operator. Interpolation is always required unless only rotations of multiples of $90°$ are considered. In practice, this means that the rotation equivariance will only be approximate.

Since the rotation can force weights near the corners of the filter to be relocated outside of its spatial support, only the weights within a circle of diameter m pixels are used to compute the convolutions. The output tensor $\mathbf{y} \in \mathbb{R}^{H \times W \times R}$ consists of R feature maps computed as:

$$\mathbf{y}^{(r)} = (\mathbf{x} * \mathbf{w}^r) \quad \forall r = 1, 2 \dots R, \tag{3}$$

where $(*)$ is the convolution operator. The tensor $\mathbf{y}$ encodes the roto-translation output space such that rotation in the input corresponds to a translation across the feature maps. Note that only the canonical filter $\mathbf{w}$ is actually stored in the model. During backpropagation, gradients corresponding to each rotated filter $\nabla \mathbf{w}^r$ are aligned back to the canonical form and added:

$$\nabla \mathbf{w} = \sum_r g_{-\alpha_r}(\nabla \mathbf{w}^r). \tag{4}$$

This block can be applied on conventional CNN feature maps (left side of Fig. 2) or on vector field feature maps (right side of Fig. 2). In the second case it is computed on each component *independently* and the resulting 3D tensors added:

$$(\mathbf{z} * \mathbf{w}) = (\mathbf{z}_u * \mathbf{w}_u) + (\mathbf{z}_v * \mathbf{w}_v), \tag{5}$$

where subscripts u and v denote the horizontal and vertical components.

It is important to note that the image rotation operator g_α requires an additional step when $\mathbf{w} \in \mathbb{R}^{m \times m \times 2}$ is a 2D vector field. The components of $\mathbf{w}^r = g_{\alpha_r}(\mathbf{w})$ have to be computed as:

$$\mathbf{w}_u^r = cos(\alpha_r) g_{\alpha_r}(\mathbf{w}_u) - sin(\alpha_r) g_{\alpha_r}(\mathbf{w}_v) \tag{6}$$

$$\mathbf{w}_v^r = cos(\alpha_r) g_{\alpha_r}(\mathbf{w}_v) + sin(\alpha_r) g_{\alpha_r}(\mathbf{w}_u) \tag{7}$$

3.1.2 Orientation pooling (OP):

Given the output 3D tensor $\mathbf{y}$, the role of the orientation pooling is to convert it to a 2D vector field $\mathbf{z} \in \mathbb{R}^{H \times W \times 2}$. This avoids the exploding dimensionality problem by only keeping information about the maximally activating orientation of $\mathbf{w}$. First, we extract a 2D map of the largest activation magnitudes, $\boldsymbol{\rho} \in \mathbb{R}^{H \times W}$, and their corresponding orientations, $\boldsymbol{\theta} \in \mathbb{R}^{H \times W}$. Specifically, for activations located at $[i, j]$:

$$\boldsymbol{\rho}[i, j] = \max_r \mathbf{y}[i, j, r], \tag{8}$$

$$\boldsymbol{\theta}[i, j] = \frac{360}{R} \arg\max_r \mathbf{y}[i, j, r]. \tag{9}$$

This can be treated as a polar representation of a 2D vector field as long as $\boldsymbol{\rho}[i, j] \geq 0 \quad \forall i, j$, a condition that is met when using any function on $\mathbf{y}$ that returns non-negative values prior to the OP. We employ the common Rectified Linear Unit (ReLu) operation, defined as $\texttt{ReLu}(x) = \max(x, 0)$, to $\boldsymbol{\rho}$, as it provides non-saturating, sparse non-linear activations offering stable training. Then, this representation can be transformed into Cartesian coordinates as:

$$\mathbf{u} = \texttt{ReLu}(\boldsymbol{\rho}) \cos(\boldsymbol{\theta}) \tag{10}$$

$$\mathbf{v} = \texttt{ReLu}(\boldsymbol{\rho}) \sin(\boldsymbol{\theta}) \tag{11}$$

with $\mathbf{u}, \mathbf{v} \in \mathbb{R}^{H \times W}$. The 2D vector field $\mathbf{z}$ is then built as:

$$\mathbf{z} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \mathbf{u} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \mathbf{v} \tag{12}$$

3.1.3 Spatial pooling (SP) for vector fields

Max-pooling is commonly used in CNNs to obtain some invariance to small deformations and reducing the size of the feature maps. This is done by downsampling the input feature map $\mathbf{x} \in \mathbb{R}^{M \times N \times d}$ to $\mathbf{x}_p \in \mathbb{R}^{\frac{M}{p} \times \frac{N}{p} \times d}$. This operation is performed by taking the maximum value contained in each one of the C non-overlapping $p \times p$ regions of $\mathbf{x}$, indexed by c. It is computed as $\mathbf{x}_p[c] = \max_{i \in c} \mathbf{x}[i]$, which can be expressed as:

$$\mathbf{y}_p[c] = \mathbf{y}[j], \text{ where } j = \arg\max_{i \in c} \mathbf{y}[i]. \tag{13}$$

This allows us to define a max-pooling for vector fields as:

$$\mathbf{z}_p[c] = \mathbf{z}[j], \text{ where } j = \arg\max_{i \in c} \boldsymbol{\rho}[i], \tag{14}$$

where $\boldsymbol{\rho}$ is a standard scalar map containing the magnitudes of the vectors in $\mathbf{z}$.

3.1.4 Batch normalization (BN) for vector fields

BN [13] normalizes every feature map in a mini-batch to zero mean and unit standard deviation. It improves convergence by training with stochastic gradient descent.

In our case, since working with vector fields of magnitude and orientation of activations, BN should only normalize magnitudes of the vectors to unit standard deviation. It would not make sense to normalize the angles, since their values are already bounded and changing their distribution would alter important information about relative and global orientations. Given a vector field feature map $\mathbf{z}$ and its map of magnitudes $\boldsymbol{\rho}$, we compute batch normalization as:

$$\hat{\mathbf{z}} = \frac{\mathbf{z}}{\sqrt{\text{var}(\boldsymbol{\rho})}}. \tag{15}$$

3.2. Computational considerations

Although RotEqNet allows for smaller models, they might require a higher count of convolutions than a comparable standard CNN. For instance, with the architecture used for MNIST-rot in Sec. 4, a standard CNN requires $4\times$ more filters per layer to saturate performance, compared to RotEqNet. At the same time, RotEqNet requires $R/4 = 4.25\times$ (for $R = 17$) more convolutions. This results in RotEqNet saving $10\times$ in model memory, $2\times$ in data memory at a price of requiring just $1.5\times$ more computing time. This is because, although the convolution count is higher, the number of feature maps per convolution is smaller. Less feature maps mean smaller convolution filters and the possibility to use larger mini batches, both factors contributing to a faster training.

4. Experiments

We explore the performance of RotEqNet on datasets where the orientation of the patterns of interest is arbitrary. This is very often the case in biomedical and abovehead imaging, since the orientation of the camera is usually not correlated with the patterns of interest. We apply RotEqNet to problems from these two fields, as well to MNIST-rot, a randomly rotated handwritten digit recognition benchmark. We also perform a study on the trade-off between invariance and accuracy in a synthetic patch matching problem. These case studies allow us to analyze the performance of RotEqNet in problems requiring equivariance, covariance and invariance to rotations and to analyze the effectiveness of RotEqNet to perform accurately with very small model architectures and limited training samples.

4.1. Invariance: MNIST-rot

MNIST-rot [17] is a variant of the original MNIST digit recognition dataset, where a random rotation between 0° and 360° is applied to each 28×28 digit image. The training set is also considerably smaller than the standard MNIST, with 12k samples, from which 10k are used for training and 2k for validation. The test set consists of 50k samples. Since we aim at predicting the correct label independently from the rotation, this problem requires rotation invariance.

Type	Size
Input	28×28
RotConv, 2×2 SP	9×9, 6 filt.
	$14 \times 14 \times 6$
RotConv, 2×2 SP	$9 \times 9 \times 6$, 16 filt.
	$7 \times 7 \times 16$
RotConv 2×2 SP	$9 \times 9 \times 16$, 32 filt.
	$1 \times 1 \times 32$
Fully connected	$1 \times 1 \times 32$, 128 filt.
	$1 \times 1 \times 128$
FC, Softmax	$1 \times 1 \times 128$, 10 filt.
Output	$1 \times 1 \times 10$

Table 1. Network architecture used on the MNIST-rot dataset. Layer parameters are in white and variables are shaded in gray.

Model: We test four CNN models with the same architecture, but different number of filters per layer. The largest model we used is shown in Table 1 and involves 100k parameters. The models are trained for 90 epochs, starting with a learning rate of 0.1 and reducing it gradually to 0.001. The weight decay is kept constant at 0.01. We use a dropout rate of 0.7 in the fully connected layer and batch normalization before every convolutional layer. The number of orientations is set to $R = 17$.

Test time data augmentation: We observe an important contribution of data augmentation at test time, a technique often used with approximately invariant or equivariant CNNs [9, 12]. In particular, we input to the network several rotated versions of the same image using fixed angles between 0° and 90°. Rotation-based data augmentation at test time might seem counter-intuitive in a rotation invariant model, but the different rotations coupled to resampling of images and filters (cf. Sec. 3.1.1) will produce slightly different activations. The final prediction is given by the average of such scores. We report results obtained with and without this type of augmentation.

Comparison to data augmented training: In order to disentangle the contributions of data augmentation and RotEqNet, we trained the RotEqNet model and a standard CNN with the same architecture and $10\times$ more parameters. In Tab. 4.1, we show the results for these models trained on both MNIST-rot and 10k digits from the original MNIST, with and without data augmentation. We observe how both methods complement each other.

Method	Error rate (in %)
SVM [17]	10.38±0.27
TIRBM [26]	4.2
H-Net [29]	1.69
ORN [31]	1.54
TI-pooling [16]	1.2
RotEqNet (Ours)	**1.09**
RotEqNet, only scalar field	2.01
RotEqNet, test-time augmentation	**1.01**

Table 2. Error rate on the MNIST-rot dataset trained on the train-val subset.

	Train on MNIST		Train on MNIST-rot	
	No augm.	Augm.	No augm.	Augm.
CNN	57%	2.3%	4.9%	2.2%
RotEqNet	20%	1.1%	1.4%	1.1%

Table 3. Results on MNIST and MNIST-rot using a standard CNN or RotEqNet, with and without data augmentation.

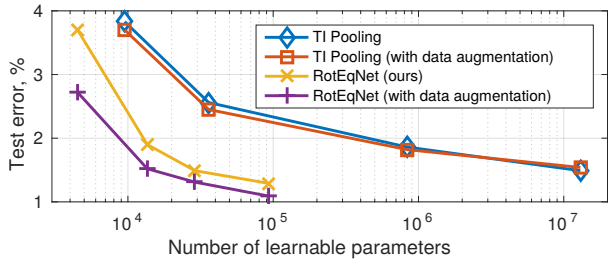

Figure 3. Performance of RotEqNet and TI-Pooling on MNIST-rot with respect to the number of parameters.

Results: We first studied the behavior of RotEqNet with respect to the total number of parameters and compared it to the state-of-the-art TI-pooling [16]. Figure 3 shows the results for both methods trained on the training set with different model sizes. The latter was achieved by varying the number of filters per layer, keeping the same architecture. RotEqNet requires approximately two orders of magnitude less parameters to obtain the same accuracy as TI-Pooling.

We report the test error in Table 2. RotEqNet obtains an error of 1.09%, a small improvement with respect to the state-of-the-art TI-pooling [16], but with almost 100× less parameters. Test-time data augmentation further reduces the error to 1.01%, thus improving significantly over TI-Pooling and over the more recent H-Net [29] and ORN [31].

4.2. Equivariance: ISBI 2012 Challenge

This benchmark [1] involves segmentation of neuronal structures in electron microscope (EM) stacks [3]. In this problem we need to precisely locate the neuron membrane boundaries. Therefore, a rotation of the inputs should lead to the same rotation in the output, making the ISBI 2012 problem a good candidate to study rotation equivariance.

The data consist of two EM stacks of *drosophila* neurons, each composed of 30 images of size 512 × 512 pixels (Fig. 4a). One stack is used for training and the other for testing. The ground truth for the training stack consists of densely annotated binary images (Fig. 4b). The ground truth for the test stack is private and the results are to be submitted to an evaluation server [3].

(a) (b) (c) (d)

Figure 4. Example validation image (#30) of the ISBI 2012 challenge. (a) Image (190 × 130 pixels). (b) Membrane ground truth. (c) The pre-processed 3-class ground truth: black is non-membrane, yellow is membrane center, red is membrane border and blue is non-class. (d) Probability map produced by RotEqNet.

Type	Size
Input	512 × 512
RotConv, OP, 2 × 2 SP	9 × 9, N filt.
	256 × 256 × N × 2
RotConv, OP, 2 × 2 SP	9 × 9, 2N filt.
	128 × 128×2N×2
RotConv OP, 2 × 2 SP	9 × 9×2N×2, 3N filt.
	64 × 64×3N×2
RotConv, OP Upsample and stack	9 × 9×3N×2, 4N filt.
	512 × 512 × 10N
RotConv fully connected	1 × 1 × 10N × 2, 5N filt.
	512 × 512 × 5N
RotConv OP	9 × 9×5N×2, 4N filt.
	512 × 512 × 4N
Fully connected	1 × 1×4N×2, 8N filt.
	512 × 512 × 8N
FC, Normalize	1 × 1 × 8N, 3 filt.
Output	512 × 512 × 3

Table 4. Network architecture used with ISBI 2012 challenge data. Layer parameters are in white and variables are shaded in gray.

Model: We transform the original binary problem into a three class segmentation problem: 1) non-membrane, 2) central membrane pixels and 3) external membrane pixels. Pixels in the membrane but not belonging to either 2) or 3) are considered to be unlabeled (Fig. 4c). This way, we can assign a higher penalization to the non-membrane pixels next to the membrane and a lower one to those in the middle of the cells. The central membrane scores are used as the final binary prediction (Fig. 4d).

Since we are dealing with a dense prediction problem with spatial autocorrelation at different resolution levels, we apply three RotConv blocks with spatial pooling. We then upsample the output of each block to the size of the original image, before concatenating them and applying two more RotConv blocks. Table 4 shows the architecture. The parameter N is used to change the size of the model. We evaluated the results with $N = 2$ and with an ensemble of three models, with $N = [1, 2, 3]$.

Comparison to data augmented training: we evaluated the RotEqNet model ($N = 2$) and an equivalent standard CNN with 10× more parameters on 5 held out validation images. RotEqNet seems not to profit as much from data augmentation as its standard CNN counterpart, but improves the CNN solution in all the cases considered, as illustrated in Table 4.2.

[3] http://brainiac2.mit.edu/isbi_challenge/

	No augm.	Augm.
CNN	0.9232	0.9572
RotEqNet	0.9726	0.9790

Table 5. ISBI results on the validations set using a standard CNN or RotEqNet, with and without data augmentation.

Method	Rand. Thin	Inf. Thin	# params.
CUMedVision [4]	0.9768	0.9886	-
IAL MC/LMC [2]	**0.9826**	**0.9894**	-
DIVE [9]	0.9685	0.9858	5.7M
PolyMtl [8]	0.9689	0.9861	11M
U-Net [22]	**0.9728**	**0.9866**	33M
RotEqNet ($N = 2$)	0.9599	0.9806	30k
RotEqNet, 3 models	0.9712	0.9865	100k

Table 6. Scores on the held out test set of the ISBI 2012 Challenge.

Results: A detailed explanation on the evaluation metrics used in the challenge can be found on the ISBI 2012 challenge website[3], as well as in [1]. The winners of the challenge were Chen *et al*. [4], although Beier *et al*. [2] have the highest scores at the time of writing. These two works rely on complex post-processing pipeline. Our rotation equivariant prediction provides results comparable to other state-of-the-art methods only relying on the raw CNN softmax output [8, 9, 22] (see Table 6).

4.3. Covariance: car orientation estimation

Estimating car orientations from above-head imagery requires rotation covariant models. We use the dataset provided by the authors of [11], which is based on Google Map images. It is composed by 15 tiles, where cars' bounding boxes and corresponding orientations come from manual annotation. We implement our approach in similarly to [11]. We crop a 48×48 square patch around every car, based on the bounding box center point. We then use these crops for both training and testing of the model. As in [11], we use the cars in the first 10 images (409 cars) for training and those in the last 5 images (209 cars) for testing. We did not use the cars whose center was nearer than 38 pixels from the image border, in order to avoid artifacts.

Type	Size
Input	48×48
RotConv	11×11, 3 filt.
OP	$38 \times 38 \times 4 \times 2$
RotConv	$11 \times 11 \times 3 \times 2$, 6 filt.
OP	$28 \times 28 \times 6 \times 2$
RotConv	$11 \times 11 \times 6 \times 2$, 3 filt.
OP, 2×2 SP	$9 \times 9 \times 3 \times 2$
RotConv fully	$9 \times 9 \times 3 \times 2$, 1 filt.
connected (FC1)	$1 \times 1 \times 21$
FC2, Hardcoded	$1 \times 1 \times 21$, 2 filt.
Output	$1 \times 1 \times 2$

Table 7. Architecture of the car orientation estimation network.Parameters are in white and variables are shaded in gray.

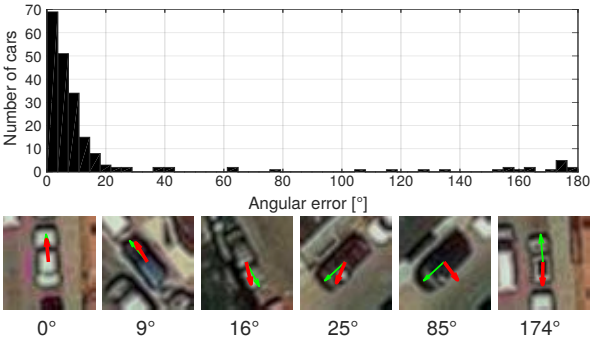

Figure 5. Distribution of the errors in the test set (top). Examples (bottom) of correctly and incorrectly identified orientations. Ground truth arrows in green (thin) and predictions in red (thick).

Model: We want to learn a covariant function with respect to rotations, since a rotation by $\Delta\alpha^\circ$ in the input image results in a change by $\Delta\alpha^\circ$ in the predicted angle. In particular, we train on sine and cosine of α°, since they are continuous with respect to $\Delta\alpha^\circ$. The network's architecture is illustrated in Table 7. For the output we use a tanh non-linearity, followed by a normalization of the output vector to unit-norm. The first fully connected layer (FC1) is a RotConv block with a single filter ($R = 21$) *not* followed by by an Orientation Pooling, meaning that the subsequent feature vector has 21 dimensions instead of just one. We can expect this vector to undergo a circular shift when the input image is subject to a rotation. We hardcode the two mappings of the following layer (FC2) to $[\sin(360/R), \sin(2 \cdot 360/R), \dots \sin(R \cdot 360/R)]$ and $[\cos(360/R), \cos(2 \cdot 360/R), \dots \cos(R \cdot 360/R)]$. This ensures that there will be no preferred orientations inherited from a biased training set. The weight decay and learning rate are 10^{-2} and $5 \cdot 10^{-3}$ respectively, for the 80 epochs. All the filters were initialized from a normal distribution with zero mean and $\sigma = 10^{-3}$. The final models correspond to the average of the weights of the last 30 epochs.

Results: Table 8 reports the average test error. The use of RotEqNet substantially improves the results, outperforming by more than 20% the previous state-of-the-art method [11]. In Fig. 5, we show the error distribution in the test set for the hybrid model. Note how most samples, 82.7%, are predicted with less than 15° of orientation error, while most of the contribution to the total error comes from the 6.7% of samples with errors larger than 150°, in which the front of the car has been mistaken with the rear.

Method	Avg. error (°)	# params
CNN [11]	28.87	27k
Warped-CNN [11]	26.44	27k
RotEqNet (Ours)	24.07	5k
RotEqNet (Ours)	**20.46**	9k

Table 8. Mean error in the prediction of car orientations.

Sensitivity to R: In order to study the sensitivity of RotEqNet to the number of angles R, we trained the model using $R = 21$ and tested it for different values (see Figure 6). We observed relatively small changes in the test error for $R > 17$.

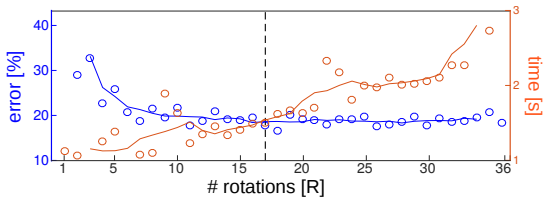

Figure 6. Error (left y-axis, blue) vs computational time (right y-axis, red) for the number of filters considered. The vertical dashed line denotes $R = 17$.

4.4. Invariance 2: robustness in patch matching

Patch matching is widely used in many image processing and computer vision problems, such as registration, 3D reconstruction and inpainting. The aim is to find matching pairs of patches (*e.g.* the same features in the two different images of the same object). In this setting, the differences in orientation are often considered to be a nuisance. Although handcrafted features such as SIFT are still widely used as baselines to measure similarity, recent works have shown that learning ad-hoc features with siamese CNNs [25, 30] can perform substantially better.In the following, we apply RotEqNet to analyze how this problem can benefit from a tunable amount of rotation invariance.

Depending on the problem at hand, one might have a prior on how much rotation invariance is required. Although CNN-based descriptors are more robust to relative rotations between matching pairs than SIFT, they still tend to perform poorly for large angular differences [25].

To showcase how RotEqNet allows to tune the amount of rotation invariance, we trained a siamese network with three RotConv blocks, with 3, 6 and 32 filters of size 9×9 respectively, totaling 40k parameters. The last fully connected block provides 32 scalar features. We trained it on 20k samples from the Notredame dataset [28] with a distance-based objective function [25, 30].

After training, the number of bins in the last Orientation Pooling layer can be modified, thus yielding multiple descriptors per sample. For instance, if the number of bins is set to 4, one 32-dimensional descriptor will be produced for each quadrant, thus resulting in a 128-dimensional descriptor for the patch. We analyze robustness in patch matching by increasing the rotation of the patches and the number of bins, and compare our results to those obtained by SIFT and the features from a pre-trained VGG network [25]. We use patches extracted from an urban photograph that are then paired to a shifted (by one pixel) and rotated version of itself. Results in Fig.7 show that RotEqNet with a single bin is much more robust to rotations than VGG and SIFT descriptors, even when the main orientation assignment is used. As a trade-off, it performs slightly worse for small rotations. However, by increasing the number of bins we can invert this tendency and improve the matching accuracy for small angles (and trade off accuracy on large rotations): using two bins (i.e. a 64-dimensional descriptor), we clearly outperform the baselines on small angles and still have 60% of correct matches for rotations around $45°$ (compared to less than 10% for SIFT and VGG).

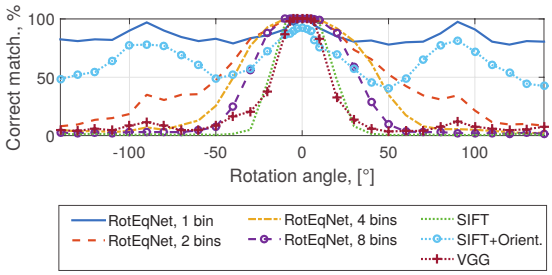

Figure 7. Matching accuracy vs. rotation applied to one of the elements in each matching pair in a synthetic dataset. RotEqNet allows to trade-off some accuracy at small rotations for more robustness by changing the number of bins in the last Orientation Pooling layer.

5. Limitations and future work

Forcing the Orientation Pooling block to choose the most activating orientation could result in exacerbating noise when there is no main orientation on either the input or the filter. This is because the arbitrarily chosen orientation can have a big impact on the output, and how it will interact with filters in the following layer, but no meaning. This problem is amplified by the use of scalar products between the vector elements of the filter and its input, which assumes that the orientation of these vectors is relevant. This issue could be improved by using a custom similarity metric between vector elements such that symmetries in the filters or the input are taken into account.

6. Conclusion

We have presented a new way of hard-coding into CNNs predefined behaviors with respect to rotations. This is achieved by applying each filter at different orientations and extracting a vector field feature map, encoding the maximum activation in terms of magnitude and angle.

Experiments on classification, segmentation, orientation estimation and matching show the suitability of this approach for solving a wide variety of problems that are inherently rotation equivariant, invariant or covariant. These results suggest that taking into account only the dominant orientations is sufficient to tackle successfully a range of problems.

References

[1] I. Arganda-Carreras, S. C. Turaga, D. R. Berger, D. Cireşan, A. Giusti, L. M. Gambardella, J. Schmidhuber, D. Laptev, S. Dwivedi, J. M. Buhmann, et al. Crowdsourcing the creation of image segmentation algorithms for connectomics. *Frontiers in Neuroanatomy*, 9, 2015. 6, 7

[2] T. Beier, B. Andres, U. Köthe, and F. A. Hamprecht. An efficient fusion move algorithm for the minimum cost lifted multicut problem. In *Proceedings of the European Conf. on Computer Vision (ECCV)*, pages 715–730. Springer, 2016. 7

[3] A. Cardona, S. Saalfeld, S. Preibisch, B. Schmid, A. Cheng, J. Pulokas, P. Tomancak, and V. Hartenstein. An integrated micro-and macroarchitectural analysis of the drosophila brain by computer-assisted serial section electron microscopy. *PLoS Biol*, 8(10):e1000502, 2010. 6

[4] H. Chen, X. J. Qi, J. Z. Cheng, and P. A. Heng. Deep contextual networks for neuronal structure segmentation. In *Proceedings of the Conf. on Artificial Intelligence (AAAI)*, 2016. 7

[5] G. Cheng, P. Zhou, and J. Han. RIFD-CNN: Rotation-Invariant and Fisher Discriminative convolutional neural networks for object detection. In *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 2884–2893, 2016. 2

[6] T. S. Cohen and M. Welling. Group equivariant convolutional networks. In *Proceedings of the International Conf. on Machine Learning*, pages 2990–2999, 2016. 3

[7] T. S. Cohen and M. Welling. Steerable CNNs. *arXiv preprint arXiv:1612.08498*, 2016. 3

[8] M. Drozdzal, E. Vorontsov, G. Chartrand, S. Kadoury, and C. Pal. The importance of skip connections in biomedical image segmentation. In *Proceeding of the Deep Learning and Data Labeling for Medical Applications Workshop (DLMIA)*, pages 179–187, 2016. 7

[9] A. Fakhry, H. Peng, and S. Ji. Deep models for brain em image segmentation: novel insights and improved performance. *Bioinformatics*, 32(15):2352–2358, 2016. 5, 7

[10] R. Gens and P. M. Domingos. Deep symmetry networks. In *Advances in neural information processing systems*, pages 2537–2545, 2014. 3

[11] J. F. Henriques and A. Vedaldi. Warped convolutions: Efficient invariance to spatial transformations. *arXiv preprint arXiv:1609.04382*, 2016. 2, 7

[12] G. Hu, Y. Yang, D. Yi, J. Kittler, W. Christmas, S. Z. Li, and T. Hospedales. When face recognition meets with deep learning: an evaluation of convolutional neural networks for face recognition. In *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 142–150, 2015. 5

[13] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, 2015. 4

[14] M. Jaderberg, K. Simonyan, A. Zisserman, et al. Spatial transformer networks. In *Advances in Neural Information Processing Systems*, pages 2017–2025, 2015. 2

[15] J. J. Kivinen and C. K. Williams. Transformation equivariant boltzmann machines. In *Proceedings of the International Conf. on Artificial Neural Networks (ICANN)*, pages 1–9. Springer, 2011. 3

[16] D. Laptev, N. Savinov, J. M. Buhmann, and M. Pollefeys. TI-pooling: transformation-invariant pooling for feature learning in convolutional neural networks. In *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition*, pages 289–297, 2016. 2, 5, 6

[17] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In *Proceedings of the International Conf. on Machine Learning (ICML)*, pages 473–480, 2007. 5

[18] Y. Le Cun, O. Matan, B. Boser, J. S. Denker, D. Henderson, R. Howard, W. Hubbard, L. Jacket, and H. Baird. Handwritten zip code recognition with multilayer networks. In *Proceeding of the IEEE Intl. Conf. on Pattern Recognition (ICPR)*, volume 2, pages 35–40. IEEE, 1990. 2

[19] K. Lenc and A. Vedaldi. Understanding image representations by measuring their equivariance and equivalence. In *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 991–999, 2015. 3

[20] D. Marcos, M. Volpi, and D. Tuia. Learning rotation invariant convolutional filters for texture classification. In *Proceedings of the International Conf. on Pattern Recognition*, 2016. 3

[21] J. Ngiam, Z. Chen, D. Chia, P. W. Koh, Q. V. Le, and A. Y. Ng. Tiled convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1279–1287, 2010. 3

[22] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Proceedings of the International Conf. on Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, pages 234–241. Springer, 2015. 7

[23] L. Sifre and S. Mallat. Rotation, scaling and deformation invariant scattering for texture discrimination. In *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 1233–1240, 2013. 3

[24] P. Y. Simard, D. Steinkraus, and J. C. Platt. Best practices for convolutional neural networks applied to visual document analysis. In *Proceedings of the International Conf. on Document Analysis and Recognition (ICDAR)*, volume 3, pages 958–962, 2003. 3

[25] E. Simo-Serra, E. Trulls, L. Ferraz, I. Kokkinos, P. Fua, and F. Moreno-Noguer. Discriminative learning of deep convolutional feature point descriptors. In *Proceeding of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 118–126, 2015. 8

[26] K. Sohn and H. Lee. Learning invariant representations with local transformations. In *Proceedings of the International Conf. on Machine Learning (ICML)*, pages 1311–1318, 2012. 3, 5

[27] A. Vedaldi and K. Lenc. MatConvNet – convolutional neural networks for MATLAB. In *Proceeding of the ACM Intl. Conf. on Multimedia*, 2015. 3

[28] S. A. Winder and M. Brown. Learning local image descriptors. In *Proceeding of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8. IEEE, 2007. 8

[29] D. E. Worrall, S. J. Garbin, D. Turmukhambetov, and G. J. Brostow. Harmonic networks: Deep translation and rotation equivariance. *arXiv preprint arXiv:1612.04642*, 2016. 3, 5, 6

[30] S. Zagoruyko and N. Komodakis. Learning to compare image patches via convolutional neural networks. In *Proceeding of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 4353–4361, 2015. 8

[31] Y. Zhou, Q. Ye, Q. Qiu, and J. Jiao. Oriented response networks. *arXiv preprint arXiv:1701.01833*, 2017. 3, 5, 6