Multi-dimensional feature and data mining

Georgiou, T.

5

Clifford convolution inspired orientation equivariant CNNs

Convolutional Neural Networks (CNNs) such as VGG [338], ResNet [127] and DenseNet [146], are very powerful models for processing image and sensor data. However, a notable drawback for their application in many domains is their sensitivity to rotations of the input data. In order to alleviate this issue several methods have been proposed. Most of these implement orientation pooling through max-pooling as a core aspect to achieve rotational invariance or equivariance. However, such operations are very computationally intensive since all the activations have to be computed for many different orientations. In this chapter, we take advantage of vector field representation to repeatably calculate the angle of the kernel with the input pattern. This is achieved by generalizing the basic convolutional kernels to realize Clifford convolutions. With this architecture, rotation angles can be calculated without computing activations for all orientations. Since the rotation angle is computed from the Clifford kernels and the input alone, this information can be utilized in the learning process through backpropagation. The proposed method is evaluated on the rotated MNIST [200, 197] on classification performance (rotation invariance) and orientation prediction (rotation equivariance). We show that this method improves the equivariance property of networks over max-pooling, preserving or even improving classification performance while having lower computational cost.

5.1 Introduction

Convolutional Neural Networks (CNNs) are very powerful models that can extract high level information from their input, such as the contents of an image [302]. State of the art networks, such as AlexNet [184], NiN [216], VGG nets [338], ResNet [127], and DenseNet [146], achieve this by combining a hierarchy of feature extractors, i.e., convolutional layers, to complex and deep architectures. A limitation of the convolution operation is that it is not rotation equivariant, and thus the output signal changes if input signal is rotated. As a result, if a network needs to identify the same pattern at different orientations it would require multiple kernels.

There are multiple potential benefits in a rotational invariant or equivariant convolution operator as proposed in this work. The number of trainable parameters might be reduced [409, 408] since no additional kernels are required for detecting rotated patterns. Due to the resulting models being simpler, they will also be less prone to overfitting and have lower memory footprints. Additionally, in some cases it is essential to extract the orientation of the detected patterns, such as for aerial imagery [233] and robotic grasp detection [204].

From the research literature on rotational equivariance, four main groups can be identified. One is to rotate the input in multiple orientations and then pool the top most activations from the different orientations [189]. The second rotates the kernels from all layers and forwards the activations of different orientations throughout the network, and then pools the activations from different orientations at the latest layer [449]. The third group of methods is similar to the second group, with the difference that orientation pooling happens at all layers and all positions of the feature maps [233]. Thus the orientation information is still propagated, whilst the number of trainable parameters and the information flow does not explode. The last method is inspired by the group convolutional networks [55] and convolves in the orientation dimension as well. This combined with an initial input of convolutions through all rotations and orientation pooling in the last layer results in one of the highest performing methods to date. Nonetheless, since all equivariant layers are convolving in the orientation dimension as well, it is a very computationally intensive approach. All four methods increase the memory footprint of the networks and incur additional computational cost.

In this work, we take advantage of vector field representation to calculate the angle between two vector fields, i.e., the kernel and input signal. This is achieved by using convolutional kernels which utilize operations from Clifford Algebra [316, 75]. This enables the detection of rotated patterns without significant memory and computational overhead, but also allows for the extraction of the rotation angles of the detected patterns. The orientation angles are calculated from the kernels and the

layer input alone, and thus can contribute to the learning process through back-propagation. Even though the extractable rotation angles are in principle continuous, we employ rotation quantization in order to reduce complexity. However, we reconstruct (approximate) continuous angles by employing a correction mechanism. Our experiments show that our method increases the rotation equivariance of the models over max-pooling based networks. At the same time it manages to maintain and in some cases improve the classification performance while being less memory and computationally intensive.

5.2 Related work

An early approach to rotation equivariant CNNs is the Spatial Transform Networks (STN) [157]. A specific module is proposed which transforms the network input before it is passed to the prediction network. Although this method does not suffer from the extra memory footprint and computational demand of computing feature maps for all orientations, the invariance is not intrinsic but rather learned by the network, and thus restricted by the training data. TI-pooling [189] rotates the input multiple times and processes it with Siamese networks. After the first fully connected (FC) layer, they perform orientation pooling and thus the next FC layer takes as input rotation equivariant features.

Another group of works rotate the feature maps in every layer and propagate all the information to the next layers. At a final stage the orientation and spatial information are pooled together. The Orientation Response Networks (ORN) [449] additionally define the orientation alignment layer, a SIFT like alignment applied before the orientation pooling. The same principle is also applied in the RI-LBC [443].

The Rotation Equivariant Networks (RotEqNet) [233] calculate the response at each layer and position for a number of rotated kernels, which represent vector fields. Then, the most dominant orientation per location and output channel is computed via max-pooling, and the result is represented by a new vector field with the response of the convolution as the magnitude at the respective angle. The approach presented here is similar in the sense that the activations are also vector fields, and that different orientations in all positions and channels throughout the network are accounted for.

Another approach to rotation equivariance is the group convolution [55], which define operation groups, e.g. rotation, and perform convolution on all the operations in the group. The result are filters that have equivariant response to all operations in the group. The main difference with the above is that all rotations are being processed and the equivalent information propagated in the network instead of performing orientation pooling and propagating only a fraction of the information. This approach

is utilized by the steerable filter CNNs defined by [409]. These networks make use of the group convolutions in combination with steerable filters to produce rotation equivariant nets. At the last convolution layer they perform orientation pooling and finally three fully connected layers perform the classification. The work of Weiler et al. [408] created a more general formulation regarding and re-implement existing architectures, such as the steerable CNNs [409] and propose new architectures based on their finding and existing operators. With a combination of cyclic equivariant group with a cyclic and flip group (only in the first few layers) they achieve state of the art performance in the MNIST-rot dataset. It should be noted that most of the networks in this cluster of methods, i.e., utilizing group convolutions, only perform orientation pooling before the fully connected layers and thus process much more information than most methods.

The aforementioned methods perform orientation pooling in various variants, where to the best of our knowledge the best performing methods incorporate max-pooling for orientation pooling. In this paper we propose a method which specifically tackles the orientation estimation and therefore we compare to max-pooling as the state of the art orientation pooling mechanism in our experiments. Additional techniques which would influence the performance, such as the steerable filters [409], could be combined with our approach. In order to evaluate our claim we re-implement the RotEqNet [233], as well as group convolutional CNNs with steerable filters [409, 408] using the proposed Clifford convolution inspired layer for orientation pooling.

Clifford convolution has been previously used for pattern recognition on 3D vector fields [75]. Patterns, such as vortices, were identified using predefined kernels, that represent those patterns, i.e., hand crafted feature extraction was done. As far as the authors know, this is the first time Clifford algebra has been used in the area of deep learning where multiple layers of filters (so called features) are learned.

5.3 Clifford convolutions and calculation of rotation angles

One central aspect of Clifford Algebras is the definition of general products between vectors and their geometric interpretation. Most relevant for this work is the ability to estimate the angle between two vector fields by performing two convolutions [75],

$$\tan \phi_{i,j} = \frac{conv_2(i,j)}{conv_0(i,j)} \;\Rightarrow\; \phi_{i,j} = \arctan \frac{conv_2(i,j)}{conv_0(i,j)}, \tag{5.1}$$

where

$$conv_0(i',j') = \sum_{c_i}^{c_{in}} \sum_{i}^{F_h} \sum_{j}^{F_w} \vec{W}_{i,j,c_i} \cdot \vec{O}_{i'-\frac{F_h}{2}+i,j'-\frac{F_w}{2}+j,c_i}$$

$$conv_2(i',j') = \sum_{c_i}^{c_{in}} \sum_{i}^{F_h} \sum_{j}^{F_w} (W_{0,i,j,c_i} \cdot O_{1,i'-\frac{F_h}{2}+i,j'-\frac{F_w}{2}+j,c_i} - W_{1,i',j',c_i} \cdot O_{0,i'-\frac{F_h}{2}+i,j'-\frac{F_w}{2}+j,c_i}) \tag{5.2}$$

where $\vec{O}_{i',j',c_i}, \vec{W}_{i,j,c_i} \in R^2$ are multi-channel 2D signal and kernel vector fields with discrete coordinates $(i',j') \in [(0,0),(O_h,O_w)]$ and $(i,j) \in [(0,0),(F_h,F_w)]$, where O_h, O_w, F_h, F_w are the image (O) and filter (F) height and width respectively:

$$\vec{O}_{i',j',c_i} = (O_{0,i',j',c_i} \ , \ O_{1,i',j',c_i}) \tag{5.3}$$

The original formulation considers continuous vector fields and integrations instead of summations [75]. Depending on the instantiation of $\vec{O}$ and $\vec{W}$, as well as the angle between them, the computation might be inaccurate. In order minimize this error, we calculate the angles for four orientations of the kernels for successive angles of $\frac{\pi}{2}$. The kernel orientation that produced the smallest angle is considered the most accurate, and thus we calculate the final angle from this choice. Experimentally we found that choosing the orientation that produced the highest value of $conv_0$ results in more accurate networks and thus it is chosen for all our experiments.

To illustrate the angle measurement, we construct a random 50-channel 2D vector field $\vec{O}$ on a 3×3 grid. Each magnitude is randomly drawn from a uniform distribution in $[0,1)$ and each angle in $[0,2\pi)$. To construct example filters, we rotate $\vec{O}$ randomly and add Gaussian noise, $\vec{W}_{ij} = rotate_{i'j' \to ij}(\vec{O}_{i'j'},\phi) + \vec{G}_{ij}$. Both the grid positions and the 2D vectors are rotated by a random angle ϕ. We denote the rotated kernel $\vec{W}$ by angle ϕ as $\vec{W}_\phi$.

We generate $10,000$ such random vector fields and filters and calculate the angle of the original fields ($\vec{O}$) to the rotated fields ($\vec{W}$) with and without the addition of noise. Figure 5.1 shows the distribution of the difference between the calculated angle and the real angle used for the rotation. Even with the addition of noise, the average absolute difference is less than 10^{-2} radians.

5.4 Layer construction

5.4.1 Forward pass computations

The rotation equivariant processing layer proposed in this work can be decomposed into five steps $[C_1, C_5]$ which is schematically shown in Figure 5.2. For the sake of

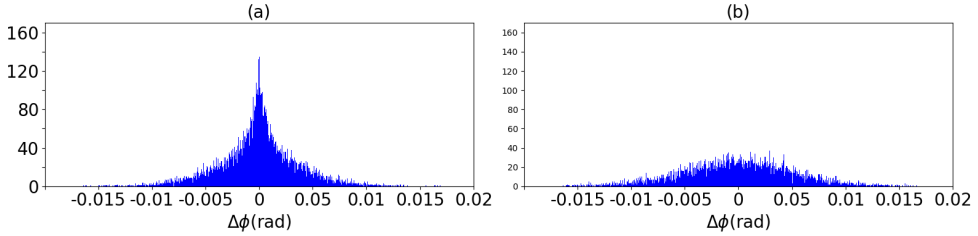


Figure 5.1: Distribution of the difference, in radians, between the rotation angle and the calculated one. (a) The difference without Gaussian noise added to the rotated signal. (b) The difference with Gaussian noise where $\mu = 0$ and σ is 10 percent of the maximum possible value of the signal.

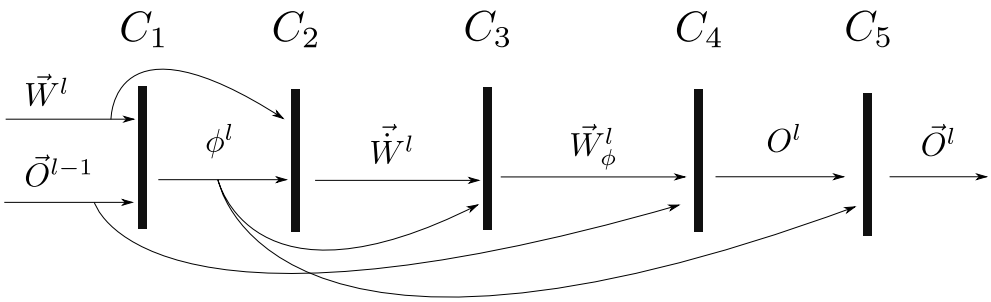


Figure 5.2: Computational pipeline of a filter $\vec{W}^l$ in layer l. C_{1-5} define the different operations. Input to the pipeline is a kernel of dimensions (F_h, F_w, c_{in}) and a window of the input with the same size with the kernel. For that combination, an angle ϕ^l is computed which is used to rotate the kernel. The rotated kernel is used to compute the scalar convolution which is then transformed to a vector.

simplicity we define the following shorthand indices:

$$\begin{aligned} p_w &: \text{weight position, } (i, j, c_{in}, c_{out}) \\ p_o &: \text{output position, } (i', j', c_{out}) \\ p_{in} &: \text{input position, } (i'', j'', c_{in}) \end{aligned} \tag{5.4}$$

For clarity, we note that if certain indices of a Tensor are not displayed, then the complete dimensionality is considered.

The first step (C_1) calculates the angle $\phi^l_{p_o}$ between the input to this layer $\vec{O}^{l-1}$ and a kernel $\vec{W}^l_{c_{out}}$ for all output positions and all output channels c_{out}. In the second and third step the rotated kernels $\vec{W}^l_{\phi\ p_o}$ are calculated ($C_{2,3}$), for all output positions and channels, where the first step C_2 only rotates in the 2D vector space for each channel and grid location (indicated by $\vec{W}$), and in the third step C_3 the rotation is

also performed with respect to the grid indices (i, j) using bi-linear interpolation, i.e., plane rotation. Without loss of generality we can consider both steps as one operation. The step C_4 performs a scalar convolution of the rotated kernels and the input. The scalar convolution result $O^l_{p_o}$ and the angle $\phi^l_{p_o}$ used to compute it are considered as vectors in the polar coordinate system which are then transformed to the Cartesian representation at step C_5.

In large networks, calculating rotated kernels, $\vec{W}^l_{\phi\ p_o}$, for all positions and channels is very inefficient. Thus, we precompute the rotation of the kernels for B angles, $\phi_b = b\frac{2\pi}{B}$ $(b = 0, ..., B-1)$. At each output point we calculate an angle $\phi^l_{p_o}$ using formulas 5.1 and 5.2 and then select the precomputed kernel with the closest angle to $\phi^l_{p_o}$. If the input pattern is significantly different from the kernels, the computed angle might still be very large. We set a maximum threshold for the angle, in which case the output $O^l_{p_o}$ is set to zero. The final pipeline can be seen in Figure 5.3. In order to reconstruct continuous angles ϕ and the respective scalar convolution result O_ϕ from the quantized angles ϕ_b, we make the assumption that for a small angle ϵ and the output of the scalar convolution at the most appropriate angle ϕ, O_ϕ, the convolution can be approximated as $O_{\phi+\epsilon} = O_\phi \cos\epsilon$. In our case this becomes:

$$O_{\phi_b} = O_\phi \cos\epsilon_b \Rightarrow O_\phi = \frac{O_{\phi_b}}{\cos\epsilon_b}, \tag{5.5}$$

where ϕ_b and $\epsilon_b = \phi - \phi_b$ are the closest quantized angle and the difference to the continuous angle. We also considered calculating the Taylor expansion to calculate the response in the real angle ϕ given the response at angle ϕ_b. For every order of the Taylor expansion an extra convolution operation has to be computed, which adds a lot to the final computational complexity. Thus we only implement a first order Taylor expansion:

$$O_\phi = O_{\phi_b} + \frac{\frac{\partial O_\phi}{\partial \phi}(\phi_b)}{1!} \cdot (\phi - \phi_b) \tag{5.6}$$

In our experiments, the correction in Equation 5.5 produced significantly better results than the first order Taylor expansion approximation 5.6. Given that it is also less computationally intensive, we use this correction for all our experiments.

5.4.2 Back propagation

Implementing the back propagation algorithm is straight forward for the operations C_4 and C_5, whereas some explanation is needed for operations C_1 and $C_{2,3}$ (for simplicity, we group C_2 and C_3, without effecting the derivations.) At operations $C_{2,3}$ the output is:

$$\vec{W}^l_{\phi\ p_o} = vector\_field\_rotation(\vec{W}^l_{c_{out}}, \phi^l_{p_o}), \tag{5.7}$$

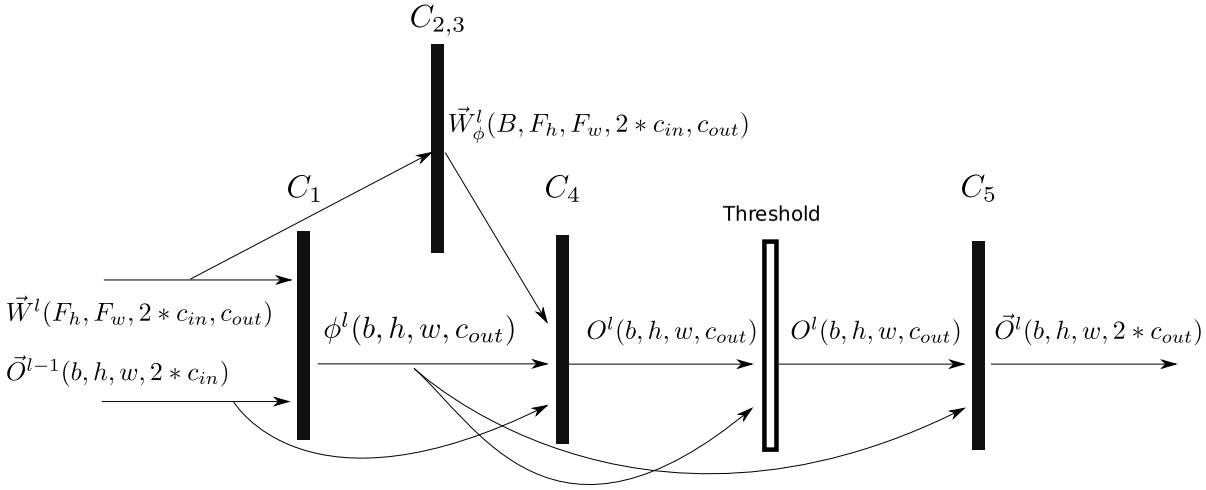


Figure 5.3: Implementation Workflow. In parentheses are the shapes of the respective tensors, where b is the batch size, h, w, c_{in} the size of the input, c_o the number of channels and F_h, F_w the kernel height and width respectively. Finally, B is the number of bins used for the precomputed rotated kernels.

where $\vec{W}^l_{c_{out}}$ is the complete vector field kernel of shape $(F_h, F_w, 2 \times c_i)$. For the back propagation two sets of gradients need to be computed, $\frac{\partial E}{\partial \vec{W}^l_{c_{out}}}$ and $\frac{\partial E}{\partial \phi^l_{p_o}}$. Since $\vec{W}^l_{\phi\ p_o}$ is the rotated $\vec{W}^l_{c_{out}}$, we get:

$$\begin{aligned}\frac{\partial E}{\partial \vec{W}^l_{c_{out}}} &= vector\_field\_rotation(\frac{\partial E}{\partial \vec{W}^l_{\phi\ p_o}}, -\phi^l_{p_o}),\ \frac{\partial E}{\partial \phi^l_{p_o}} \\ &= \sum_i \sum_j \sum_{c_{in}} \frac{\partial E}{\partial \vec{W}^l_{\phi\ p_o p_w}} \cdot \frac{\partial \vec{W}^l_{\phi\ p_o p_w}}{\partial \phi^l_{p_o}}\end{aligned} \tag{5.8}$$

To avoid confusion we note that $\vec{W}^l_{\phi\ p_o p_w}$ refers to the rotated weights for the output position p_o indexed by p_w. The last term in Equations 5.8 above can be estimated with the precomputed kernel weight sets for the quantized angle ϕ_b as:

$$\frac{\partial \vec{W}^l_{\phi_b}}{\partial \phi^l} = \frac{\vec{W}^l_{\phi_{b+1}} - \vec{W}^l_{\phi_{b-1}}}{2\frac{2\pi}{B}} \tag{5.9}$$

To back propagate the gradient through step C_1, where $\phi^l_{p_o} = \arctan(\frac{conv_2}{conv_0})$, we need to calculate two sets of gradients:

$$\frac{\partial E}{\partial conv_0} = -\frac{\partial E}{\partial \phi^l_{p_o}} \frac{conv_2}{conv_0^2 + conv_2^2},\ \frac{\partial E}{\partial conv_2} = \frac{\partial E}{\partial \phi^l_{p_o}} \frac{conv_0}{conv_0^2 + conv_2^2} \tag{5.10}$$

Equations 5.10 introduce numeric instabilities when both $conv_0$ and $conv_2$ are very close to zero, and even undefined when exactly zero. As a remedy, we set a small threshold for $conv_0$ and $conv_2$, below which we turn off the feature map at the specific location. The derivatives of $conv_0$ and $conv_2$ can easily be calculated from their definitions in Eq. 5.2. The analytical derivations can be found in the Appendix A.

5.5 Experiments

5.5.1 Datasets and ground truth

In order to evaluate our method, we perform experiments on rotation invariance and rotation equivariance. With that goal in mind we utilize the MNIST-rot [200, 197] dataset and a car orientation dataset [131]. Moreover, we construct our own version of the MNIST-rot, as explained in Section 5.5.4, to help us evaluate the rotation equivariance property of the methods tested.

For the RotEqNet [233] networks and their Clifford convolution counterparts we extract the gradients of the color from a 3×3 window and utilize them as input, in order to have a vector field as input.

5.5.2 Networks

For our experiments we utilize two network architectures, i.e. the $SFCNN$ [409, 408] and the RotEqNet [233]. The $SFCNN$ represent the state of the art in rotation equivariance and equivariance, utilizing group convolutions whilst the RotEqNet remains one of highest performing approaches and its much less computationally intensive than the group convolution approach.

Each layer of the RotEqNet performs convolutions in a number of predefined orientations of the kernels and then forwards the maximum response. Furthermore, it utilizes the orientation information to create a vector field as an output, i.e., the magnitude of each vector is the maximum response and the angle is the rotation angle of the orientation that produced that maximum response. Every convolutional layer is followed by batch normalization and a ReLU activation function (both applied on the magnitudes of the vectors). We also experimented with batch normalization with only rescaling, as proposed by the authors [233], but in our experiments the full batch normalization proved to produce better results. After a number of layers global pooling is applied and the resulted information is processed by a number of fully connected layers which perform classification.

The Clifford convolution (cc) inspired counterparts of the RotEqNet follow the same architecture with a few differences. Instead of full batch normalization we apply only rescaling (as with the cc networks it proved to result in higher performance) and the addition of a second normalization which makes the average norm of a square window, with size equal to the next layer's kernel size, to be one (Equation 5.11). This is needed in the cc networks due to the threshold introduced at the end of Section 5.4.2. With varying input norm the effect of the threshold changes. For fair comparison we did also try to add them at the original RotEqNet and found that it marginally improves the performance and thus utilized it in all our experiments.

$$norm(\vec{O}^{l-1}) = \frac{\vec{O}^{l-1}}{\lambda_O},\ \lambda_O = \sqrt{\frac{F_w^l F_h^l}{O_w^{l-1} O_h^{l-1}}} ||O^{l-1}||_2 \tag{5.11}$$

For the $SFCNN$ we follow the proposed hyperparameter settings proposed in [408] and reproduced their results. The first layer outputs the response of the networks for the predefined orientations (sixteen orientations). The rest of the convolutional layers besides the standard spatial 2D convolution, convolve in the angular dimension as well and output responses for all orientations. Before their three fully connected layers they perform both spatial and orientation pooling by applying max-pooling.

The cc counterpart utilizes the input layer and group convolutions as with the max-pooling version. The difference lies in the orientation pooling layer in which we

Table 5.1: Number of channels per layer. CL denotes Convolutional Layer, FC are Fully Connected and Out is the classification layer. With (/2) we denote layers that are followed by 2x2 spatial pooling. (-) denotes layers for which the kernel size is the same as the input and do not use padding, producing output size 1x1.

	CL1	CL2 (/2)	CL3	CL4 (/2)	CL5	CL6(-)	FC	FC	Out
kernel size	9	7	7	7	7	5	-	-	-
$rotEqNet$ [233]	16	32	36	36	64	96	96	96	10
$SFCNN$ [409]	24	32	36	36	64	96	96	96	10
$D_{16\|5}C_{16}{}^{*}$ [408]	24	32	36	36	64	96	96	96	10

utilize the cc approach.

All the above networks have fully connected layers performing the classification and thus rendering the whole network as rotation invariant and not rotation equivariant. In order to test rotation equivariant networks, we consider the fully connected layers as convolutional layers with kernel size 1×1 and thus we are able to utilize the orientation equivariant operators above. The output of the network outputs classification probabilities (after a soft max activation) as well as a prediction angle, either produced by the argmax for max-pooling or by the cc operator. In the case of the cc operator, the produced angle is a function of the weights and the input and thus can be used for extra supervision, by utilizing the ground truth angle of the examples. The resulted networks approximate end-to-end orientation equivariant behavior.

For our experiments we implemented our forward pass and back propagation steps for using the "new op" module of Tensorflow [1], both for CPU and GPU computations. Our source code has been tested with Tensorflow versions 1.12 and 1.13, with CUDA version 10.0. All experiments were done on nVidia Geforce GTX 1080Ti and RTX 2080Ti GPUs.

5.5.3 MNIST-rot

For all our experiments, we use the number of layers as well as number and sizes of kernels as in [409, 408, 233]. All the relevant parameters can be seen in Table 5.1. For reference we also show the parameters and performance of the $D_{16|5}C_{16}$ [408] network since, to the best of our knowledge, it has the highest performance on the rotated MNIST dataset. It should be noted that it is very similar to the $SFCNN$, utilizing group convolutions. The main difference is that it performs convolution over the flip operation as well for the first few layers.

Table 5.2: Classification accuracy of the networks on the rotated MNIST dataset. * denotes results reported in the respective papers. Values in parentheses denote average performance over 5 runs and outside the parenthesis is the maximum performance achieved. In case there is no value in parenthesis the presented value is the performance of the only run.

	op	Acc. our lr scheduler	Acc. [408] lr scheduler
$RotEqNet$ [233]	mp	98.954 (98.918)	99.064 (99.031)
$RotEqNet$	cc	99.088 (99.071)	99.004 (98.974)
$SFCNN$ [409]	mp	99.358 (**99.308**)	99.320 (99.294)
$SFCNN$ [409]	cc	99.318	99.238 (99.166)
$RotEqNet^*$ [233]	mp	-	- (98.99)
$SFCNN^*$ [409]	mp	-	- (99.286)
$D_{16\|5}C_{16}{}^*$ [408]	mp	-	- (**99.318**)

For all networks we trained using the Adam optimizer [175] with learning rate 10^{-3} and a mini-batch size of 128. We drop the learning rate to 10^{-5} at 30 epochs and train for 320 epochs overall. We also train with the scheduler defined in [409] for fairness. The angle threshold defined in Section 5.4.1 is set to $\frac{\pi}{2}$, as it proved to increase the performance. The threshold due to gradient instabilities, described in Section 5.4.2, is set to 10^{-3}.

Following the practice in literature [233, 409, 408] we set the number of orientations to $B = 16$. For both Clifford and max-pooling based networks. We found out that training with 16 bins and testing with 32 slightly increases the performance, while keeping the training times lower. For the $SFCNN$ this is not possible since increasing the number of bins B changes the shape of the weight matrices as well. The results can be seen in Table 5.2.

We should note that as it is apparent that the learning rate scheduler of [408] is not favorable to the Clifford convolution (cc) inspired operator, since using our scheduler increases the performance of the networks. When replacing the max-pooling operation with the cc operator, the accuracy of the $RotEqNet$ model increases by a small margin. When applied on the $SFCNN$ the result remains approximately the same (with our learning rate scheduler). As a benefit, our method has much less computational intensity discussed in Section 5.5.6.

5.5.4 Enriched MNIST-rot

As a second test, we wanted to also predict the angles of the digits. Unfortunately the original MNIST-rot dataset does not provide the angles with which each image is rotated and thus it is not possible to evaluate the performance of the networks. Thus, we create our own MNIST-rot in the same way that is described in the original work [200, 197], with the only difference that we use cubic spline interpolation instead of bilinear interpolation for rotation. Moreover, we save the angle used to rotate each digit.

The networks in the previous section can not predict any angles and thus we adjust them. We change the fully connected layers (even the output) to convolutional, with kernel size 1x1. Then we perform the same orientation prediction as with the rest convolutional layers, i.e., max-pooling with argmax or the one proposed here (Equations 5.1, 5.2). Similarly, for the $SFCNN$ networks we remove the orientation pooling layer after the last convolution, transform the two fully connected layers to group convolutional layers with filter size 1x1 and make the output layer orientation equivariant with either argmax or Clifford convolution to predict the angles (like the RotEqNet).

Our method is the only one that can utilize angle information as a supervisory signal (without big adjustments on the network). Thus we implement two setups. In the first setup the networks are trained only with class labels and the angle is left unsupervised. For the second setup the angle information is also utilized during training. The networks that utilize the max-pooling with argmax can not utilize angle information. Thus we include a second output branch (in parallel to the first) which is tasked to predict the sinus and cosine of the angle. This second branch consists of two layers. We follow the approach of [233], where a hand crafted fully connected layer of the form $[[\cos(0), \cos(1 \cdot \frac{2\pi}{B}), \cos(2 \cdot \frac{2\pi}{B}), \cdots, \cos((B-1) \cdot \frac{2\pi}{B})], [\sin(0), \sin(1 \cdot \frac{2\pi}{B}), \sin(2 \cdot \frac{2\pi}{B}), \cdots, \sin((B-1) \cdot \frac{2\pi}{B})]]$ takes as input the responses of all orientations of the previous layer. A final fully connected layer predicts the sinus and cosine of the angle. An interesting remark is that this method utilizes a secondary network that predicts the angle, increasing the complexity of the network and its trainable parameters. Meanwhile, the Clifford convolution approach calculates the angle from the existing weight vectors and their respective input. It is also important to note that the unsupervised setup does not utilize this new branch and the predicted angle is the result of the argmax function.

During test, we expect that the networks that utilized the angle information while training will be able and predict the correct orientation of the input. In the case of unsupervised angles, we expect the output of the network to be rotation equivariant. Thus, there should be a steady offset between the angles predicted by the network

and the ground truth angles. In order to test the angle prediction performance, we compute the average angle difference between predicted and ground truth angles on the training set. We use that offset to "correct" the predicted test angles and measure the difference to the ground truth.

To measure the performance on angle prediction, the absolute value ($\in [0, \pi)$) of the difference between the predicted angles and the ground truth angles is averaged out. The results of the unsupervised angle experiment can be seen in Table 5.3a and those of the supervised angle experiment in Table 5.3b. For a fair comparison we also train the networks from the previous section, i.e., not equivariant fully connected layers with this version of the MNIST-rot (four first rows of Table 5.3a).

From the first four rows of Table 5.3a, we see that the methods have similar performance on the proposed MNIST-rot with the original one in Table 5.2. It is important to note that all experiments here use our learning rate scheduler instead of the one used in [408]. Regarding classification accuracy, the fully convolutional and end-to-end rotation equivariant networks seem to have similar performance to the non-fully convolutional counterparts. Regarding the angle prediction (unsupervised in this case), the inclusion of the cc operator increases the performance of the $RotEqNet$ by a significant margin. This is not the case though with the $SFCNN$. There the unsupervised angle prediction becomes worse than the performance of the max-pooling. In a closer inspection we found out that this is a result of the vector field construction we used for the $SFCNN$, where most of the vectors end up being parallel due to the fact that the group convolutional layers approximate very well rotation equivariant behavior.

When the angle information is used to supervise the networks, it is apparent that the cc operator improves the angle prediction performance over the introduced branch used for the networks utilizing the max-pooling operator. Regarding classification performance we see a similar trend to our previous two experiments.

As an extra experiment, we train the networks on only straight digits (from the original MNIST) and measure their performance on the test set of our MNIST-rot. For training we pick randomly 12K digits from the original MNIST (the same for all networks) and train with the learning rate scheduler proposed in this work. The results can be seen in Table 5.4. It is obvious that since the extra branch introduced for the previous experiment explicitly learns to predict the angles and is not inherently rotation equivariant, it does not generalize to unseen angles and its performance is very poor. On the other hand the cc operator manages to maintain high angle prediction performance.

Table 5.3: Classification accuracy and angle prediction performance of the networks on the enriched rotated MNIST dataset. "(Full)" donates networks that are equivariant end-to-end, i.e., they output angle of prediction. The first column denotes the method, the second the orientation pooling (op) method, while the third and forth columns the classification accuracy and average absolute angle difference (in radians) respectively. Values in parentheses denote average performance over 5 runs and outside the parenthesis is the maximum performance achieved. In case there is no value in parentheses the presented value is the performance of the only run. In case of two values separated by a '/', the first value is the model that achieved maximum classification accuracy while the second is the model that achieved minimum average absolute angle difference.

	op	Accuracy	Angle Prediction
$RotEqNet$ [233]	mp	99.06 (98.99)	-
$RotEqNet$	cc	99.14 (99.12)	-
$SFCNN$ [409]	mp	99.36 (99.32)	-
$SFCNN$ [409]	cc	99.28	-
$RotEqNet$ (Full) [233]	mp	99.05/98.92 (98.99)	0.4441/0.3711 (0.4301)
$RotEqNet$ (Full)	cc	99.15/99.08 (99.09)	0.2970/0.2756 (0.3059)
$RotEqNet$ (0.5 dropout, Full)	cc	99.14/99.11 (99.10)	0.2720/0.2579 (0.2696)
$SFCNN$ (Full)	mp	99.37 (99.36)	0.1798 (0.1843)
$SFCNN$ (Full)	cc	99.29 (99.27)	0.1985 (0.2227)

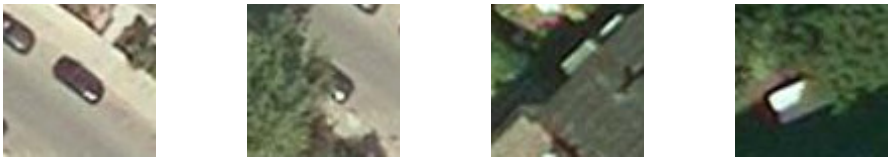
(a) Unsupervised angles.

	op	Accuracy	Angle Prediction
$RotEqNet$ (Full) [233]	mp	98.93/98.88 (98.84)	0.1156/0.1140 (0.1200)
$RotEqNet$ (Full)	cc	99.15 (99.10)	0.0985 (0.1005)
$RotEqNet$ (0.5 dropout, Full)	cc	99.29 (99.24)	0.1095 (0.1110)
$SFCNN$ (Full)	mp	99.38/99.35 (99.34)	0.0688/0.0659 (0.0694)
$SFCNN$ (Full)	cc	99.33/99.28 (99.26)	0.0608/0.0580 (0.0620)

(b) Supervised angles.

Table 5.4: Classification accuracy and angle prediction performance of the networks on the our rotated MNIST dataset, with supervised angles trained on 12K digits of the original MNIST dataset. "(Full)" donates networks that are equivariant end-to-end, i.e., they output angle of prediction. The first column denotes the method, the second the orientation pooling (op) method, while the third and forth columns the classification accuracy and average absolute angle difference (in radians) respectively.

	op	Accuracy	Angle Prediction
$RotEqNet$ (Full) [233]	mp	98.33	1.5693
$RotEqNet$ (0.5 dropout, Full)	cc	98.41	0.1892
$SFCNN$ (Full)	mp	99.12	0.3183
$SFCNN$ (Full)	cc	98.99	0.1727



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

Figure 5.4: Example images from the vehicle orientation dataset. (a) and (b) are positive examples (include vehicles), (c) and (d) are negative examples.

5.5.5 Vehicle Orientation

As mentioned above we also utilize the vehicle orientation dataset from [131]. This dataset consists of 15 aerial images with labeled vehicles and their orientations. In order to include non trivial negative examples, i.e., crops with no vehicle in them, we add specific crops from these images that do not include vehicles. We automatically extract negative patches that are challenging by utilizing the results of the detector of [129]. We pick the patches that have high scores, i.e., high confidence for vehicle detection, but have very small or no overlap to positive crops. We use the same number of negative examples are there are positive in the train set and no negative examples in the test set. To further enrich the dataset, we extract patches of resolution 76×76 randomly rotate them and crop the center 48×48 pixels to be used as input to the networks. With this dataset we train on both vehicle detection by patch classification and orientation estimation. Some pictures from our dataset can be seen in Figure 5.4. The network structures can be seen in Table 5.5 and the results in Table 5.6.

Our results show that in the case of the car vehicle detection and orientation

Table 5.5: Number of channels per layer. CL denotes Convolutional Layer, FC are Fully Connected and Out is the classification layer. With (/2) we denote layers that are followed by 2x2 spatial pooling. (-) denotes layers for which the kernel size is the same as the input and do not use padding, producing output size 1x1.

	CL1	CL2 (/2)	CL3	CL4	CL5(-)	Out
kernel size	9	7	7	5	7	-
$rotEqNet$ [233]	4	8	8	8	16	2
$SFCNN$ [409]	4	8	8	8	16	2

Table 5.6: Classification accuracy and angle prediction performance of the networks on the vehicle orientation dataset, with supervised angles. The first column denotes the method, the second the orientation pooling (op) method, while the third and forth columns the classification accuracy and average absolute angle difference (in radians) respectively. Values in parentheses denote average performance over 5 runs and outside the parenthesis is the maximum performance achieved. In case of two values separated by a '/', the first value is the model that achieved maximum classification accuracy while the second the model that achieved minimum average absolute angle difference.

	op	Accuracy	Angle Prediction
$RotEqNet$ [233]	mp	87.08 (83.88)	0.5355 (0.5894)
$RotEqNet$	cc	93.06/91.39 (91.39)	0.3694/0.3614 (0.4433)
$SFCNN$	mp	94.02 (93.83)	0.4348 (0.4796)
$SFCNN$	cc	96.41 (95.50)	0.3325 (0.4093)

Table 5.7: Time required to process a mini-batch of size 200. "(Full)" donates networks that are equivariant end-to-end, i.e., they output angle of prediction. The first column denotes the method, the second the orientation pooling (op) method, while the third and forth columns the average time in seconds on a GPU and a CPU respectively.

	op	GPU Time (s)	CPU time (s)
$RotEqNet$ (Full) [233]	mp	**0.1248**	3.1202
$RotEqNet$ (Full)	cc	0.4880	**1.8781**
$SFCNN$ (Full)	mp	0.2180	8.3354

prediction dataset the cc networks, both $RotEqNet$ and $SFCNN$ outperform their max-pooling counterparts both in classification accuracy and angle estimation.

5.5.6 Computational complexity

The last experiment is designed to compare the time each network requires to process data. For this test, we process with each network 100 batches of batch size 200 and average the the time required to precess each batch. The results can be seen in Table 5.7. For each layer, the $RotEqNet$ with max-pooling requires B convolutions to be performed, a $reduce\_max$ and $argmax$ function. In the current implementation, the equivalent cc layer requires 9 convolutions and an $arctan$ function, which overall is much lower than the $RotEqNet$. The $SFCNN$ on the other hand, not only requires B convolutions, but the dimensionality of the filters and input is also B times larger than the equivalent $RotEqNet$. Thus, we expect the cc implementation to be much faster on average than the other two approaches. Unfortunately, this is not always the case. In particular, for GPU computations the cc operator is much slower than the max-pooling counterpart. We believe that there are two main reasons for this finding. The first is that the efficiency of our code is not on par with the native code of Tensorflow. Moreover, when it comes to GPU computations, the architecture is such that the constant change of the filter being used is penalized a lot due to intense input/output overhead. This is not the case with CPU computations, where we can observe the benefit of the cc approach. Nonetheless, the difference we observe is still smaller than the one expected, leading us to the belief that with higher optimization the cc method can have a big benefit in computational cost and processing time.

5.6 Conclusion and future work

In this chapter, vector field representation is utilized and a new operator, based on Clifford convolution, is proposed for measuring feature angles in images and construct rotation equivariant CNNs. The angles are computed solely based on the kernels and the input signal, enabling the use of this information during training through back-propagation, which increases the accuracy of the resulting networks while being memory and computationally more efficient. We compare our networks to the state of the art rotation equivariant method, using orientation pooling through max-pooling.

The new operator is evaluated with two network architectures representative of the state of the art on the rotated MNIST and a vehicle orientation dataset [131] in rotation invariance, equivariance and time complexity. Our experiments show that our method is competitive to the state of the art in classification accuracy, while gaining higher accuracy in angle prediction and having a lower computational and memory resource demand.

Our work clearly demonstrates the potential of the proposed method in achieving orientation equivariant networks. However, more detailed insights are necessary. For example, the influence of the filter size on the performance, or the robustness and performance with increasing depth of networks has to be investigated. Applying our method to other datasets will give valuable insights as well. A very promising route is provided by the possibility to combine our Clifford convolution approach with other state of the art architectures, such as the res-block [127] and dense-block [146].