Rotation Invariant Filters in Convolutional Neural Networks

applied to segmentation of aerial images for land-use classification

Aniket Dhar

Master Thesis Report

TUDelft

Rotation Invariant Filters in Convolutional Neural Networks

applied to segmentation of aerial images for land-use classification

by

Aniket Dhar

in partial fulfilment of the degree of

Master of Science
in Embedded Systems

at the Delft University of Technology,
to be defended publicly on Wednesday, November 21, 2018 at 10:00 AM.

Student number:	4615808	
Project duration:	March 1, 2018 – November 21, 2018	
Thesis committee:	Prof. Dr. Marcel Reinders,	TU Delft, chair
	Dr. J. C. van Gemert,	TU Delft, supervisor
	Dr. Odette Scharenborg,	TU Delft, commitee member
	Daan van der Maas,	CIV, Rijkswaterstaat, daily supervisor

TU Delft

Preface

This report presents the work done in my master thesis. The main content of this report is the scientific paper as the first chapter. This is followed by related chapters that explain supplementary details about the work. The thesis has been done in close collaboration with Vision Lab at TU Delft and DataLab at Rijkswaterstaat, Delft.

This work would not have been possible without the constant and effective guidance of my TU Delft supervisor Dr. J. C. van Gemert. He conceptualised the idea of the research and has been immensely patient and forbearing with me throughout the discourse; always getting me back on track whenever I veered away from the goal.

My daily supervisor from Rijkswaterstaat DataLab, Daan van der Maas, has been a constant support. He has always encouraged me and provided constructive criticism, whenever necessary. I thank him, my other colleagues from DataLab, and Rijkswaterstaat in general, for the conducive work environment I have been provided with.

No journey is complete without friends. I have been lucky to find some great souls I could share my journey with. They have been there in my bad days and the worse, and always showered love, hope and encouragement.

Finally, and most importantly, I can not thank my family enough for their emotional and financial support throughout. It has all been possible because of them.

Aniket Dhar
Delft, November 2018

Contents

1 Scientific Paper **1**

2 Introduction **13**
2.1 Motivation . . . 13
2.2 Research Objective . . . 14
2.3 Outline . . . 14

3 Background on Neural Networks **15**
3.1 Neural Networks . . . 15
3.1.1 Activation . . . 16
3.1.2 Optimizers and Loss Functions . . . 17
3.2 Convolutional Neural Networks . . . 17
3.2.1 Convolution Layer . . . 17
3.2.2 Pooling Layer . . . 18
3.3 Deep Learning . . . 19
3.3.1 Image Classification . . . 19
3.3.2 Semantic Segmentation . . . 20

4 Structured Receptive Fields **23**
4.1 Scale Space Theory . . . 23
4.2 Structured Receptive Fields . . . 24
4.3 Structured Convolution Layer . . . 24
4.4 Network . . . 25

5 Rotational Invariant Filters **27**
5.1 Schmid filter bank . . . 27
5.2 Constructing rotational invariant filters . . . 28

6 Performance Evaluation Method **29**
6.1 Classification . . . 29
6.1.1 Accuracy . . . 29
6.2 Segmentation . . . 30
6.2.1 Intersection over Union . . . 30

7 Additional Experiments **31**
7.1 MNIST Rotated . . . 31
7.2 Class distribution in segmentation problem . . . 32

Bibliography **33**

1 Scientific Paper

Rotation invariant filters in CNNs

applied to segmentation of aerial images for land-use classification

Aniket Dhar
TU Delft
The Netherlands
A.Dhar@student.tudelft.nl

Abstract

Convolutional neural networks are showing incredible performance in image classification, segmentation, object detection and other computer vision applications in recent years. But they lack understanding of affine transformations to input data. In this work, we introduce rotational invariant convolutional neural networks that learn rotational invariance by design, and not from data. We build rotation invariant filters through parametric learning of linear combination of a basis set of filters, rather than modelling the filters ourselves. Our approach combines the learning capability of CNNs with custom filter selection. We show stability in performance under rotations in input images. We first validate our findings for classification on MNIST and then for multi-class semantic segmentation on the DeepGlobe 2018 Satellite Image Understanding Challenge.

1. Introduction

Convolutional neural networks (CNNs) are showing incredible performance in image classification, segmentation, object detection and other computer vision applications in recent years. An important aspect of human visual recognition system is that we can identify and classify objects and scenes irrespective of affine transformations and deformations in the viewpoint. Computer vision systems are yet to achieve that level of invariance to changes in their input. Rotation of object or scene is a common affine transformation that we encounter quite frequently. But conventional CNNs do not have the ability to handle rotations well. We propose to design CNNs with rotation-invariant kernels implemented as structured receptive fields [1] (i.e. linear combination of a basis filter set), to make the networks invariant to rotations.

Rotation is an important aspect in images which don't have a specific sense of direction; for example, images of land cover from a satellite or a drone (see figure 1). Similar examples can be seen in astronomical or cellular microscopic images. In such a case, any arbitrary rotation in the image plane does not change the context of the viewpoint and hence, should not affect the classification. CNNs perform well in domains where the sense of direction is similar to the domain it has been trained on. But rotations in the test domain drastically change prediction outcome. We aim to make CNNs invariant to rotation so that any rotation in the input space does not affect its classification ability.

CNNs learn to incorporate translation and rotation invariance to some extent through i) pooling/subsampling layers and ii) data augmentation. Max pooling achieves partial invariance to small translations and rotations because the max of a region depends only on the single largest feature present there. Average pooling also intuitively introduces some amount of local invariance to the position of the features as it averages over all regions. Data augmentation synthetically generates new training samples through geometric distortion according to a known noise model. This is a very widely used technique to incorporate spatial invariance. But this can exponentially increase the number of training samples and also increase intra-class variance; so the capacity of the learning model also needs to be high. Learning invariance from data can, therefore, be very complex and time-consuming.

We aim to enable CNNs with the ability to be invariant to rotations by design, and independent of the data. Following the assumption that all images, and hence, filters learnt in CNN are spatially coherent and can be decomposed into a smooth compact filter basis [1], we pose a constraint on the basis filters to be invariant to spatial rotations by themselves. A linear combination of rotation invariant set of basis filters is able to incorporate rotational invariance in a CNN by design. We chose the family of Schmid filters [2] which have spatial rotational symmetry to be our basis filter set. We optimise the weight parameters that combine the basis filters into effective convolutional kernels. The optimisation is done through backpropagation and hence, the network can learn the effective filters while abiding by the constraint of rotational symmetry.

In this paper, we use fixed filter bases as kernels of a CNN while maintaining parameter optimisation through backpropagation to learn effective linear combinations of the basis filters. We introduce the constraint of rotational invariance in the basis set so as to attain rotational invariance in the CNN. An overview of the linear combination has been shown in figure 2.

Our main contributions are:

- Incorporating rotational invariance in CNNs by replacing the convolution kernels with a linear combination of spatially symmetric fixed basis filters that can be effectively learned through backpropagation.
- Validating rotational invariance on both classification and segmentation problems; emphasising the fact that the achieved solution is generic and can be easily incorporated into existing deep-learning architectures.

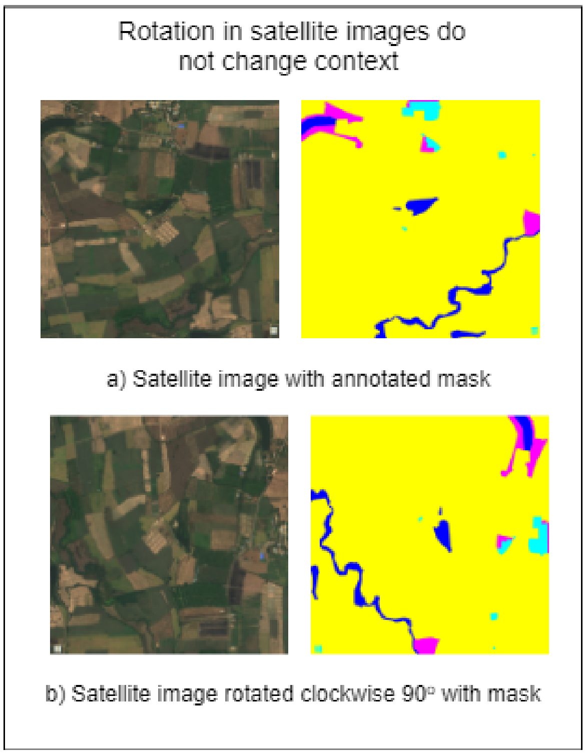

Rotation in satellite images do not change context
a) Satellite image with annotated mask
b) Satellite image rotated clockwise 90° with mask

Figure 1: Rotation in aerial or satellite images do not change the context of the view.

2. Related Work

2.1. Rotational Invariance

Transformation invariance has been a popular topic in computer vision for a long time, and we find different approaches in the literature to attain this. A very

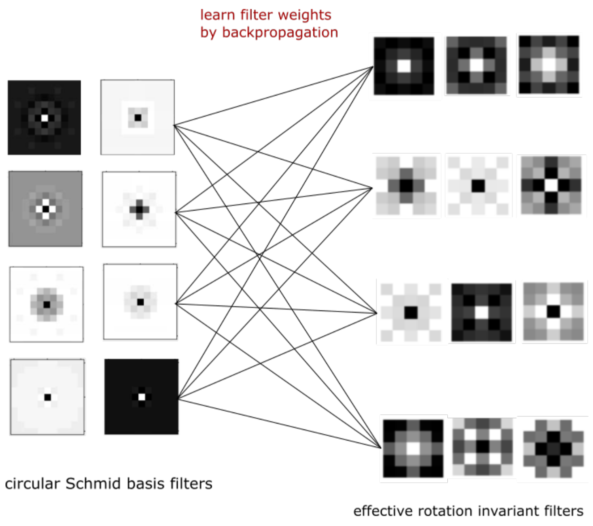

learn filter weights by backpropagation
circular Schmid basis filters
effective rotation invariant filters

Figure 2: Overview of how linear combination of Schmid basis filters produces effective filter kernels which are invariant to rotations.

common approach is to find invariant or equivariant local feature descriptors [3, 4, 5, 6]. Many papers try to analyse transformation-invariant or equivariant representations [7, 8, 9, 10, 11]. Another major approach is to model the transformations within the network as shown in [12, 13, 14, 15, 16, 17].

Invariance, equivariance and equivalence to transformations in image representations are studied through an empirical linear relationship between original and transformed images by Lenc & Vedaldi [8]. The transformation property of representations using group representation theory has been presented by Cohen & Welling [7]. This has been extended through *Steerable filters* [9] by Weiler *et al*. *Scattering networks* [10] by Bruna & Mallat explicitly design invariance through wavelet convolutions and group averaging while some CNNs learn filter banks with local transformations [11]. We do not try to estimate transformation representations of our filters but constrain them to be rotationally symmetric.

Hinton [12] and Hinton *et al*. [13] try to impose a canonical frame of reference on the object independent of viewpoint variations, called a *capsule*. The generative capsule model is able to learn relative relationships between objects in the view and represents it numerically. Each capsule learns an implicitly defined visual object within a limited set of transformations and outputs both the detection result and information about the transformation from the canonical form of the object. Tieleman [14] enhances this work to make the generative capsules capable of full affine transformations. In 2015, Google introduced *Spatial Trans-*

former Networks [15] which explicitly allow transformation of data within the network without any manual intervention. The Spatial Transformer module can be inserted into existing components of a convolutional architecture because of its differentiable property. *Inverse Compositional Spatial Transformer Networks* [16] improve this approach by implementing the image transformation through an iterative approach inspired by Lucas-Kanade optical flow algorithm. Microsoft in 2017 introduced *Deformable Convolutional Networks* [17] with deformable convolution and RoI pooling layers which augment the spatial sampling locations with learnable offsets. This enhances transformation modelling capability of CNNS. Unlike them, we do not indulge in the mathematical modelling of transformations in our image or feature maps.

Our method adopts the concept of fixed filter bases like in Scattering Network [10] but maintains the capability of CNNs to learn combination of bases to form effective filter kernels. Steerable filters [9] also employ the same concept of learning filters but unlike them, we do not compute orientation dependent responses to get steerable filters; rather we demand our filters to be rotation invariant by design.

2.2. Structured Receptive Fields

Structured Receptive Field (SRF) [1] is a type of convolution model introduced by Jacobsen *et al*. that constructs convolutional kernels by learning linear combinations of a fixed set of basis filters. This derives its inspiration from the scale-space theory [18] which expresses images as functions with a continuous scale parameter. The Gaussian is an important mathematical function that can change its spatial extent according to its standard deviation which is a continuous parameter. The Gaussian derivatives do not introduce any artefacts and can represent any image realistically. SRF applies Gaussian derivative basis filters as function priors, but still maintains the flexibility of CNNs to learn linear combinations of the bases to encode complex feature space.

In our work, we derive heavily from the idea of constructing a 2D convolution kernel as a linear combination of fixed basis filters with weight parameters that can be learnt through backpropagation. We choose the Schmid basis instead of Gaussian derivatives.

3. Methodology

3.1. Schmid Filter Bank

The Schmid filter bank consists of 13 rotationally invariant filters first introduced by Cornelia Schmid, to get rotation invariant descriptors for texture-like visual structure recognition [2]. These are Gabor-like filters that combine frequency and scale together:

$$F(x,y,\tau,\sigma) = F_0(\tau,\sigma) + \cos\frac{\sqrt{x^2+y^2}\pi\tau}{\sigma} e^{-\frac{x^2+y^2}{2\sigma^2}}, \tag{1}$$

where σ is the standard deviation of the Gaussian envelope and τ is the number of cycles of the harmonic function within the Gaussian envelope. $F_0(\tau,\sigma)$ is added to obtain a zero DC component, making it robust to illumination changes.

We use 13 filters in our filter banks with pre-determined σ and τ. For our experiments, we use σ values between 0.33 and 1.66. The τ values lie between 1 and 4. Smaller scales use smaller τ values to avoid high-frequency responses. Figure 3 shows the Schmid filter bank with a spatial resolution of 49. In our experiments, we use filters with a spatial resolution of 3 and 5 only.

3.2. Rotation Invariant Convolution Layer

In normal CNNs, the network learns convolutional kernels from pixel values. In structured receptive fields, the 2D convolution kernel is expressed as a linear combination of unique Gaussian derivative basis functions. The weight values of the linear combination are learnt through backpropagation. A convolution kernel $F(x,y)$ where x, y denote the spatial dimensions can be expressed as a linear combination of i unique basis Gaussian derivatives ϕ

$$F(x,y) = \alpha_1\phi_1 + \ldots + \alpha_n\phi_i, \tag{2}$$

where $\alpha_1, ..., \alpha_i$ are the parameters being learnt.

The network learns the α values by mini-batch stochastic gradient descent and the derivatives of the loss function are calculated in terms of the α through backpropagation.

In our implementation, we replace the Gaussian derivative filter bank with our Schmid filter bank. Since the basis filters themselves have spatial rotational symmetry, their linear combination is bound to have the same property.

We implement a custom 2D convolution layer that takes in the basis filters as one of the initialisation parameters. The layer initialises the α as random uniform distribution, normalised between [-0.01, 0.01]. These specific initialisation values of α are claimed to work the best in converging the values during training [20]. The convolution kernel is now constructed as the sum of the basis filters weighted by the α values as in equation 2. The basis filter values (ϕ) are specified to be non-trainable. Only the combination weights (α) are trainable in the constructed 2D kernel. The basis weights (α) and the basis filter weights (ϕ) are independent of each other. Thus, the custom convolution layer can already calculate the effective filter kernel and apply it to the images. This layer can be easily incorporated in any deep learning framework.

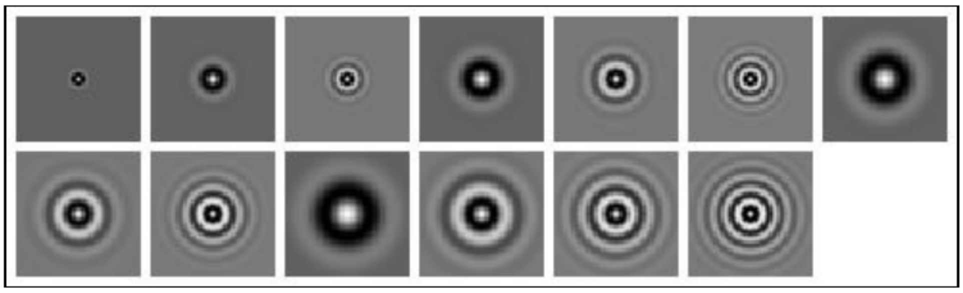


Figure 3: Schmid(S) filter bank; 13 rotationally invariant isotropic "Gabor-like" filters [19].

Another way to implement the custom filters is to convolve the image with the basis filters and then linearly recombine them with $1x1$ convolutions to get the required number of filters in the consecutive layer. The final effective filters obtained through both methods are similar. We decided to follow the first approach as i) it is more generic to most architectures as many of them do not use $1x1$ convolutions, and ii) it is easier to stack up multiple custom convolution layers this way. Algorithm 1 shows our approach of parameter learning.

Algorithm 1 Algorithm: Learning convolution kernels

```
1: **Input**: input feature maps of each training sample, basis filters
2: generate weights of linear combination as random uniform values in [-0.01, 0.01] as learnable parameters
3: define kernel shape according to input and output channels
4: compute the effective convolution kernel as weighted linear combination of basis filters
5: compute the convolution of the input feature map with the effective convolution kernel
6: obtain the output feature map and apply activation
7: compute the gradient of the loss function wrt the weights
8: update the weight parameters
9: **Output**: output feature maps
```

3.3. Network

3.3.1 Classification Network: Network-In-Network

For the classification problems, we chose the Network-In-Network (NiN) [21] architecture. It is a powerful classification network and can be called as a precursor to the famous Inception Net [22] architecture. The NiN architecture has spatial convolution layers followed by $1x1$ convolutions which form non-linear combinations of the spatial layer outputs. The $1x1$ convolutions act as cross-channel parametric pooling layers. The NiN implementation does not have a fully-connected layer. The final layer is a global average pooling layer which takes the average over all the feature maps and hence enforces direct correspondence between feature maps and categories. Each convolution layer is followed by a non-linear (ReLu) activation.

In our implementation of the NiN, we replace the spatial convolution layers with our custom rotation invariant convolution layers. These layers combine the basis rotational filters into the effective convolution filter kernel. The basis set is non-trainable and no parameters are learnt for the basis filters. The $1x1$ convolution layers further combine the kernels together through cross-channel pooling. This means all parameters are concerned with the combination of basis filters and not learning the filters themselves.

In this work, we have used all the 13 filters in the Schmid filter bank in each of the custom rotational layers. The σ and τ parameters of the filters as described in equation 1 are kept same for each layer. The custom convolution layer learns the α values from equation 2 to linearly combine the basis into effective kernels, which are invariant to spatial rotations. We use 3 layers of custom convolution, each followed by 2 $1x1$ convolutions and a pooling layer. The last $1x1$ filter before the global average pooling brings down the number of output filters to the number of classes in the classification problem. Some effective rotation invariant kernels from the 2^{nd} custom convolution layer are visualised in figure 4.

3.3.2 Segmentation Network: U-Net

We choose the popular U-Net architecture [23] for our experiments on the semantic segmentation problem. The U-Net is a popular segmentation architecture introduced on biomedical image segmentation but found usage in many other segmentation problems henceforth. The architecture derives from the idea of Fully Convolutional Networks for image segmentation [24] which removed fully-connected layers and introduced a full convolutional pipeline for segmentation problems. This allows segmentation maps to be

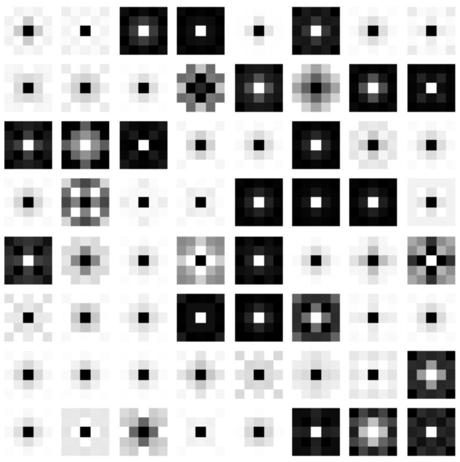


Figure 4: Visualisation of rotation invariant convolution kernels from the 2nd custom convolution layer of our NiN architecture

generated for images of any size and is much faster compared to the patch classification approach used earlier. Almost all of the subsequent state-of-the-art approaches on semantic segmentation adopted this paradigm.

One important problem of using CNNs for segmentation is pooling layers. Pooling increases the field of view but discards the ”where” information. The U-Net introduces an encoder-decoder architecture where the encoder gradually reduces the spatial dimension through pooling while the decoder gradually recovers the original dimension and object details through upsampling layers. The upsampling operators are used to replace the pooling operations and hence, incorporate the location information in the network. They also increase the spatial resolution so as the output of the network has same dimension as the input image. The U-Net also has skip connections which combine the high resolution features from the downsampling path to features in the upsampling path which helps in better localisation.

The fully convolutional structure makes the U-Net an apt choice for our consideration. In our version of the U-Net, we replace all the spatial convolution layers in the encoder part by our custom convolution layers with rotation invariant kernels. The decoder part is left as it is with convolution and upsampling layers.

4. Experiments

The experiments have been performed in two different contexts- i) classification and ii) segmentation. For each case, the experiments are partitioned into two parts; i) representative power of the kernels, in terms of performance measure (classification accuracy, mean intersection-over-union) and ii) rotational invariance, represented as difference in accuracy measure on the introduction of rotation in test samples. All experiments are done using the Keras [25] deep learning library with TensorFlow [26] backend.

4.1. Classification

In the classification experiments, we use the popular MNIST dataset [27] of handwritten digits. The architecture used is Network-in-Network (NiN) with custom convolution layers made up of Schmid basis filters. We analyse the performance of our architecture against the standard NiN, a NiN with Gaussian filters and a NiN with combination of both Gaussian and Schmid filters. In all classification experiments, we train on different sample sizes of the MNIST dataset, from 50,000 through 300. Different sample sizes are used to analyse the effect on performance of our model on small datasets.

4.1.1 Models

We use 4 different models in our classification experiments; all based on the NiN architecture. This section gives the architectural insight of each of the four models.

- **NiN_cnn**: This is the standard NiN architecture [21] with 3 blocks of conventional spatial convolution filters followed by $1x1$ convolutions.
- **NiN_gauss**: In this model, we replace all the spatial convolution filters by Gaussian derivative filters upto the 4th order. This is similar to the NiN-RFNN model from the SRF paper [1].
- **NiN_schmid**: Here, we replace all the spatial convolutions by Schmid filters. This model is the focus of our experiments, and we want to compare it with the other ones.
- **NiN_mixed**: In this model, we replace the spatial convolution layers by the combination of both Gaussian and Schmid filters. The filters are not mixed together, but are applied separately to the input and the output feature maps are concatenated and passed on to the $1x1$ layers.

4.1.2 Spatial extent of custom filters

The variance of the Gaussian function describes its spatial extent. We have tried to fix the σ parameters of our filters such that the spatial convolution kernel can capture most of the Gaussian envelope. That holds true for both Gaussian derivative and Schmid filter bases.

Parameter	Value												
σ_0	2	4	4	6	6	6	8	8	8	10	10	10	10
σ_1	0.58	0.83	0.83	1.68	1.68	1.68	2.5	2.5	2.5	4.18	4.18	4.18	4.18
σ_2	0.46	0.66	0.66	1.34	1.34	1.34	2	2	2	3.34	3.34	3.34	3.34
σ_3	0.35	0.5	0.5	1.01	1.01	1.01	1.5	1.5	1.5	2.51	2.51	2.51	2.51
σ_4	0.23	0.33	0.33	0.67	0.67	0.67	1	1	1	1.67	1.67	1.67	1.67
σ_5	0.12	0.17	0.17	0.34	0.34	0.34	0.5	0.5	0.5	0.84	0.84	0.84	0.84
τ	1	1	2	1	2	3	1	2	3	1	2	3	4

Table 1: Set of parameters used in our NiN_schmid model to check the performance of the filters with different set of σs. σ_0 is the original set from the paper [2]. τ is kept constant. The sets cover quite an extensive range of values for σ for small kernel sizes.

For the Gaussian derivative basis, we have used $\sigma = 0.33, 0.67$ for all the basis filters for convolution layers of size $3x3$ and $5x5$ respectively. For the Schmid basis, the original σ values from the paper [2] are too high for kernel sizes in our architectures. We tried different sigma values to check which best suits our approach. For the Schmid filter basis, we use the same set of sigma values for every layer. The different σ and τ parameters that have been tried are shown in table 1.

We use the different set of parameters shown in 1 in our NiN_schmid model to check the performance of the filters with different set of σs. The experiments are done on the MNIST dataset with a sample size of 1000 under exactly same training conditions. The performance comparison is shown in table 2.

Performance comparison for different σs						
σ	σ_0	σ_1	σ_2	σ_3	σ_4	σ_5
Accuracy	56.73	71.21	75.99	74.61	**78.26**	73.21

Table 2: Accuracy results on MNIST dataset for NiN_schmid model for different σ values. Here σ_i refers to the list of σ values in table 1, All the experiments have been run for a sample size of 1000 under same training conditions. The results are mean of top-3 performances. σ_4 set performs the best and is chosen for our experiments.

The σ_4 set which performs the best as seen in table 2, is used for our Schmid basis filters throughout the rest of the experiments.

4.1.3 Experiment 1: Accuracy in Classification

The aim of this experiment is to analyse the representative power of our model across the other models mentioned above. The dataset has been segmented into train, validation and test sets. The training is done for 1200 epochs with an adam optimiser [28] with standard learning parameters. Changing the learning rate did not bring any significant changes in the training process. A validation step is performed after every training epoch and the model weights from the best validation performance are saved to tackle overfitting during training. The input images are normalised and the dataset mean is subtracted. No pre-training has been done as it was deemed superfluous for the MNIST dataset. Results are obtained on the test set and the mean top-5 classification accuracy values are compared in table 3.

Performance comparison on MNIST dataset				
Sample size	NiN_cnn	NiN_gauss	NiN_schmid	NiN_mixed
50000	99.65	99.25	97.65	99.44
20000	99.50	99.14	96.28	99.13
10000	99.28	99.12	94.82	99.05
5000	98.59	98.63	92.04	98.72
2000	97.50	97.49	85.27	97.92
1000	96.59	96.54	78.26	96.36
500	94.44	94.31	70.01	94.18
300	92.81	91.84	62.52	92.23

Table 3: Accuracy results on MNIST dataset for different sample sizes. The experiment compares 4 models under different sample sizes. NiN_schmid does not perform as well in terms of representational power.

As seen from the table 3, the CNN, Gaussian and mixed models perform fairly similar; but our Schmid model lags significantly in terms of performance. To analyse this difference, we tried to visualise the kernel weights of the trained NiN_schmid model and compare it to those of the NiN_cnn model. The visualisation, as shown in figure 5, shows that the kernels learnt by our NiN_schmid are significantly simpler than the kernels of NiN_cnn. That explains the performance of our model. We also verify through this visualisation that our trained Schmid kernels are in-fact invariant to spatial rotations. We quantify this invariance in our next experiment.

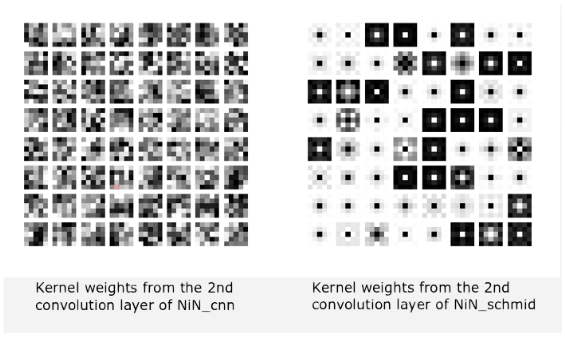

Kernel weights from the 2nd convolution layer of NiN_cnn
Kernel weights from the 2nd convolution layer of NiN_schmid

Figure 5: Visualisation of kernel weights from the 2nd convolution layer of NiN_cnn and NiN_schmid. Even though NiN_schmid are rotation invariant, they are much simpler as compared to NiN_cnn filters.

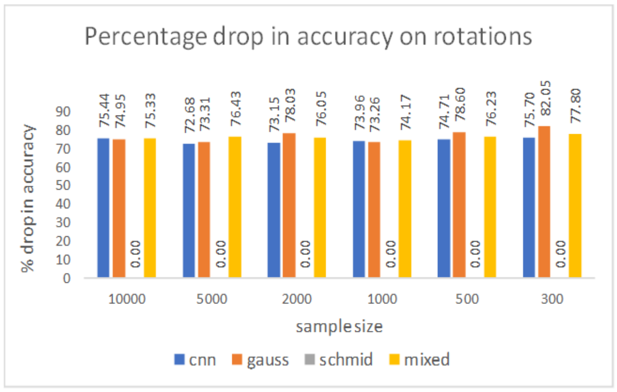


Figure 6: Visualisation of percentage drop in accuracy when rotations at right angle multiples are introduced in the test set. The values have been calculated as % drop in accuracy wrt accuracy without rotation. Schmid rotational filters show no change in accuracy on rotation.

4.1.4 Experiment 2: Rotational Invariance in Classification

The aim of this experiment is to analyse the rotational invariance of the models under consideration. For this experiment, we rotate the test set in right angle rotations i.e. 90, 180 and 270 degrees. The training process and hyperparameter values are exactly the same as the previous experiment. The trained models are evaluated on the rotated versions of the test sets and their accuracy is noted. We are interested in the difference of the accuracy values between the non rotated and rotated versions. The top-5 accuracy measures for 90, 180 and 270 degrees are averaged together to get the result for rotation. The accuracy results on the rotated test sets are shown in table 4.

Av. accuracy measure on right-angle rotations on MNIST				
Sample size	NiN_cnn	NiN_gauss	NiN_schmid	NiN_mixed
10000	24.38	24.83	94.82	24.44
5000	26.94	26.32	92.04	23.27
2000	26.18	21.42	85.27	23.46
1000	25.15	25.81	78.26	24.89
500	23.88	20.18	70.01	22.39
300	22.56	16.48	62.52	20.47

Table 4: Top-5 average accuracy results on MNIST dataset rotated at 90, 180 and 270 degrees. The top-5 accuracy values from each rotated set are averaged; and then those 3 are averaged to get the values in the table. The experiment compares 4 models under different sample sizes with rotations. Only Schmid filters do not change performance on rotations.

The figure 6 shows the percentage drop in accuracy for each model on introduction of right-angle rotations in the test set. As we can see, the NiN_schmid model shows no drop in performance while all the other models show a performance dip of about 75% each.

We visualise the feature maps generated by a random test sample and its rotated version from the same model. The feature maps, as shown in figure 7 are exactly rotated versions of each other. Pixel value difference is calculated between the rotated versions and found to be zero. Thus, experiment 2 firmly validates our hypothesis that the Schmid rotational kernels are invariant to spatial rotations.

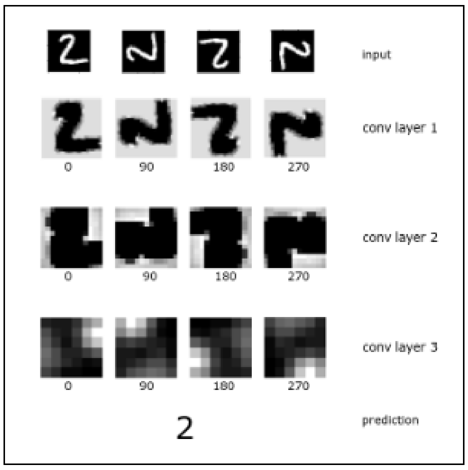


Figure 7: Visualisation of one feature map from each layer of NiN_schmid model. When input is rotated, the feature maps generated are rotated versions of each other.

4.2. Segmentation

In the second part of experiments, we focus on semantic segmentation of aerial images for land cover classification. We use the dataset from DeepGlobe Land Cover Classification Challenge [29] which contains 803 satellite images in RGB format. The challenge is a multi-class segmentation task to detect 7 different classes such as urban, agriculture, rangeland, forest, water, barren, and unknown. This is the first public dataset of its kind providing high-resolution sub-meter satellite images on rural areas [30]. The annotations are in the form of RGB mask images with 7 classes following the Anderson classification [31].

As satellite images of land cover do not have a sense of orientation, this dataset augurs well for our experiments on rotation invariance. As for the classification, we look for representative power first, and then invariance. We use the standard U-Net architecture [23] for our baseline model. We compare it with our implementation which replaces all spatial convolution layers in the U-Net encoder with our rotation invariant convolution layer; the decoder remains unchanged. We divide the dataset into 600 training, 100 validation and 100 test images.

4.2.1 Models

We use 4 different models in our segmentation experiment.

- **unet_norot:** standard U-Net
- **unet_rot:** standard U-Net trained on random right-angle rotations(90, 180, 270) in training set
- **unet_schmid_norot:** U-Net with rotation invariant convolution layers in encoder
- **unet_schmid_rot:** U-Net with rotation invariant convolution layers in encoder trained on random right-angle rotations in training set

4.2.2 Experiment 3: Accuracy in Segmentation

The aim of the experiment is to analyse the representative power of our implementation in segmentation problems. Training is done for 350 epochs with adam optimiser with a learning rate of 0.005. Validation step is prformed after every epoch and best validation weights are saved to regulate overfitting. We use pixelwise Intersection over Union (IoU) as the accuracy metric which is calculated, for m images and each $j \in N$ classes, as

$$IoU_j = \frac{\sum_{i=1}^{m} TP_{ij}}{\sum_{i=1}^{m} TP_{ij} + \sum_{i=1}^{m} FP_{ij} + \sum_{i=1}^{m} FN_{ij}}; \quad (3)$$

where TP_{ij} is the total number of correctly predicted pixels in image i that belong to class j; FP_{ij} is the total number of pixels in image i wrongly classified in class j; and FN_{ij} is the total number of pixels in image i that are wrongly predicted as any class except j. The final score is calculated as average over the N classes, expressed as

$$mIoU = \frac{1}{N}\sum_{j=1}^{N} IoU_j. \quad (4)$$

The 'unknown' class is exempted from the metric calculation.

The results obtained are shown in table 5 under the $mIoU_{norot}$ column. All values are average of top-3 $mIoU$ results. As in the classification problem, the models with rotational filters lack in representative power as compared to standard convolutional models.

Performance comparison on DeepGlobe dataset			
model	$mIoU_{norot}$	$mIoU_{rot}$	% drop
unet_norot	65.58	63.25	3.55
unet_rot	69.06	68.72	0.49
unet_schmid_norot	40.42	40.40	0.05
unet_schmid_rot	43.99	43.99	0.016

Table 5: Results on DeepGlobe dataset for land cover classification. The $mIoU_{norot}$ are results obtained on test set with no rotated samples; while $mIoU_{rot}$ are results of test set with randomly rotated samples at 90, 180 and 270 degrees. Our models show invariance to introduction of rotation in the test set.

4.2.3 Experiment 4: Rotational Invariance in Segmentation

In this experiment, we try to validate rotational invariance of our segmentation model and compare it with traditional U-Net models. This is an extension to experiment 3, where we evaluate the models on the same test set; but with random rotations at right-angles (90, 180 and 270). We expect i) performance to drop for the U-Net models with conventional filters (unet_norot and unet_rot); and ii) U-Net models with rotational filters (unet_schmid_norot and unet_schmid_rot) to be consistent in their performance and also consistent between themselves.

The results obtained are shown in table 5 under the $mIoU_{rot}$ column. It can be seen that performance of the standard U-Net falls when rotated samples are introduced in the test set; even for the unet_rot model which has been trained on rotated samples. Models with rotational filters achieve invariance to rotated samples in the test set; irrespective of whether they have been trained on rotated samples. So our first expectation is fulfilled; while the second is partially fulfilled as the unet_schmid_rot performs better than unet_schmid_norot across multiple runs. To analyse this, we compare feature maps for a sample test image and it's rotated version, shown in figure 8.

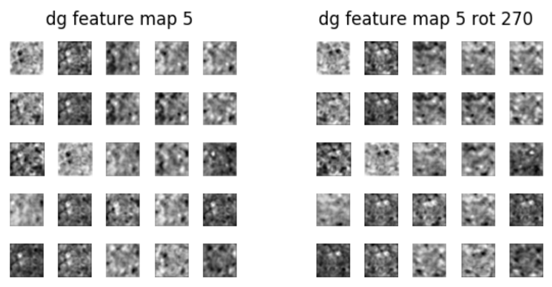


Figure 8: Visualisation of some feature maps from the 5th convolution layer in decoder of unet_schmid_norot model generated by a sample test image and its 270 degree rotated version. On visual inspection, the feature maps look like rotated versions of each other.

Comparison shows a constant intensity difference and pixel differences mainly around the edges, as shown in figure 9. This difference in the forward pass probably causes the gradient updates to be different and hence introduces the difference in performance of our two models. The intensity difference is found to originate from the batch-normalisation layers. The pixel differences at the edges can possibly be removed by a more effective padding technique. This could not be tended to due to time constraints, and is suggested in future work recommendation.

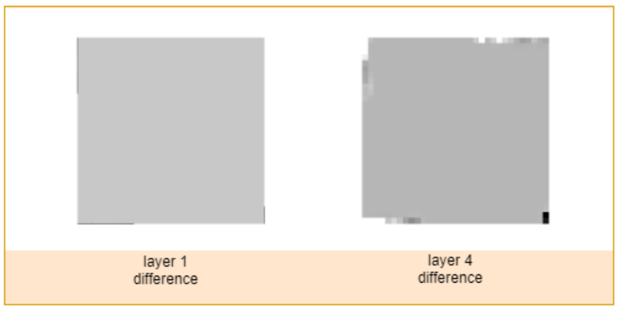


Figure 9: Pixel difference of feature maps from the 1st and 4th convolution layers in decoder of unet_schmid_norot generated by a sample test image and its 270 degree rotated version. We see pixel differences at borders and constant intensity difference throughout.

5. Conclusion and Future Work

Rotation invariance in convolution neural networks can be achieved by learning effective rotation-invariant kernels through linear combination of Schmid basis filters. The rotation invariant layers are learnt through backpropagation and can be easily integrated into any existing deep learning architecture. We validate our initial hypotheses through experiments on MNIST classification and DeepGlobe land cover segmentation. We have illustrated that the invariance obtained is by design and not learnt from the training data. This is effective for datasets that do not possess a sense of orientation.

The performance in segmentation can be made more consistent by effective padding techniques and possibly adapting better pre-processing approaches that eliminate the need of BatchNormalisation. Our rotational kernels, also, fail to achieve performance at par with conventional CNN kernels. This can be an area of future research. Another important area of research could be learning the spatial extent of the basis filters according to the dataset. The original σ parameters of the Schmid filters are too high for the small kernels that are generally used in deep learning models nowadays. So, it could be interesting if the network is able to learn the σ value itself.

References

[1] Jörn-Henrik Jacobsen, Jan C. van Gemert, Zhongyu Lou, and Arnold W. M. Smeulders. Structured receptive fields in cnns. *CoRR*, abs/1605.02971, 2016. 1, 3, 5

[2] C. Schmid. Constructing models for content-based image retrieval. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 39–45, 2001. 1, 3, 6

[3] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60:91–110, 2004. 2

[4] David G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2*, ICCV '99, pages 1150–, Washington, DC, USA, 1999. IEEE Computer Society. 2

[5] Tony Lindeberg. Feature detection with automatic scale selection. *Int. J. Comput. Vision*, 30(2):79–116, November 1998. 2

[6] Krystian Mikolajczyk and Cordelia Schmid. A performance evaluation of local descriptors. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(10):1615–1630, October 2005. 2

[7] Taco S. Cohen and Max Welling. Group equivariant convolutional networks. *CoRR*, abs/1602.07576, 2016. 2

[8] Karel Lenc and Andrea Vedaldi. Understanding image representations by measuring their equivariance and equivalence. *CoRR*, abs/1411.5908, 2014. 2

[9] Maurice Weiler, Fred A. Hamprecht, and Martin Storath. Learning steerable filters for rotation equivariant cnns. *CoRR*, abs/1711.07289, 2017. 2, 3

[10] Joan Bruna and Stephane Mallat. Invariant scattering convolution networks. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(8):1872–1886, August 2013. 2, 3

[11] Kihyuk Sohn and Honglak Lee. Learning invariant representations with local transformations. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ICML'12, pages 1339–1346, USA, 2012. Omnipress. 2

[12] Geoffrey F. Hinton. A parallel computation that assigns canonical object-based frames of reference. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'81, pages 683–685, San Francisco, CA, USA, 1981. Morgan Kaufmann Publishers Inc. 2

[13] Geoffrey E. Hinton, Alex Krizhevsky, and Sida D. Wang. Transforming auto-encoders. In *Proceedings of the 21th International Conference on Artificial Neural Networks - Volume Part I*, ICANN'11, pages 44–51, Berlin, Heidelberg, 2011. Springer-Verlag. 2

[14] Tijmen Tieleman. *Optimizing neural networks that generate images*. PhD thesis, University of Toronto, 2014. 2

[15] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial transformer networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, pages 2017–2025, Cambridge, MA, USA, 2015. MIT Press. 2, 3

[16] Chen-Hsuan Lin and Simon Lucey. Inverse compositional spatial transformer networks. *CoRR*, abs/1612.03897, 2016. 2, 3

[17] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. *CoRR*, abs/1703.06211, 2017. 2, 3

[18] Andrew P. Witkin. Scale-space filtering. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'83, pages 1019–1022, San Francisco, CA, USA, 1983. Morgan Kaufmann Publishers Inc. 3

[19] Visual geometry group, department of engineering science, university of oxford. `http://www.robots.ox.ac.uk/~vgg/research/texclass/filters.html`. Accessed: 2018-11-05. 4

[20] Sten Goes. Learning the scale of image features in convolutional neural networks, 2017. 3

[21] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *CoRR*, abs/1312.4400, 2013. 4, 5

[22] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014. 4

[23] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015. 4, 8

[24] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *CoRR*, abs/1411.4038, 2014. 4

[25] Keras documentation. `https://keras.io/`. Accessed: 2018-10-30. 5

[26] Tensorflow documentation. `https://www.tensorflow.org/`. Accessed: 2018-10-30. 5

[27] The mnist database. `http://yann.lecun.com/exdb/mnist/`. Accessed: 2018-10-30. 5

[28] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. 6

[29] Deepglobe cvpr 2018 - satellite challenge. `http://deepglobe.org/`. Accessed: 2018-11-05. 8

[30] Ilke Demir, Krzysztof Koperski, David Lindenbaum, Guan Pang, Jing Huang, Saikat Basu, Forest Hughes, Devis Tuia, and Ramesh Raskar. Deepglobe 2018: A challenge to parse the earth through satellite images. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2018. 8

[31] R. Anderson, Ernest E. Hardy, John T. Roach, and Richard E. Witmer. A land use and land cover classification system for use with remote sensor data. *USGS Prof. Pap*, page 964, 1976. 8

2 Introduction

Over the past decade, deep learning has become one of the most popular fields in machine learning, owing to rising computational power, increased storage capacity and extensive methods of data collection. Deep learning techniques have been utilised in a plethora of artificial intelligence applications, notably computer vision.

Deep convolutional networks have been at the heart of most computer vision applications, being widely used as state-of-the-art for image classification, detection, segmentation and so on. Convolutional neural networks (CNNs) have established themselves as the champion of image processing tasks as they can effectively capture locally connected features in an image. An important aspect of CNNs is that they can capture translational invariance in their local encoded features to some extent through convolution and pooling operations. This invariance property adds generalisation to its results and helps CNNs to perform well while limiting the parameter count as compared to fully connected networks.

One important invariance property that CNNs fail to capture is *rotational* invariance. Rotation is an extremely common phenomena in case of images. Hence, rotational invariance will be essential in computer vision applications where an arbitrary rotation in the image plane does not objectively affect the inherent information contained in the objects or scene involved. Such examples are commonly found in aerial/satellite and microscopic images.

This work aims to tap into the potential of convolutional networks to attain rotational invariance with the help of rotationally invariant kernels. These kernels are learnt, through backpropagation, from a fixed number of custom filters which are invariant to rotation. The chapter presents motivation of this work, the research questions involved and an overall outlook of the report.

2.1. Motivation

Transformation invariance has been a topic of interest in the computer vision community for a long time. One of the affine transformation that we encounter frequently is rotation. Rotation is a common phenomenon in aerial or satellite images, telescopic images of celestial bodies and many biomedical image applications. So, understanding rotation is a very important aspect for the CNN, if applied to such datasets.

In general, CNNs learn to incorporate translation and rotation invariance to some extent through i) pooling/subsampling layers and ii) data augmentation. Max pooling achieves partial invariance to small translations because the max of a region depends only on the single largest feature. Average pooling also intuitively introduces some amount of local invariance to the position of the features. Data augmentation means synthetic generation of new training samples through geometric distortion according to a known noise model [11]. This is a very widely used technique to incorporate spatial invariance. But the main problem in this approach is that it exponentially increases the number of training samples; so the capacity of the learning model also needs to be high. Learning invariance from data can, hence, be very complex and time consuming.

If we look at the literature, there are several methods that have been applied to tackle the problem of rotations. A very common approach is to find invariant or equivariant local feature descriptors [25, 27–29]. Many papers try to analyse transformation-invariant or equivariant representations [4, 8, 22, 34, 36]. Another major approach is to model the transformations within the network as shown in [9, 13, 14, 17, 23, 35].

Invariance, equivariance and equivalence to transformations in image representations are studied through empirical linear relationship between original and transformed images by Lenc & Vedaldi [22]. The transformation property of representations using group representation theory has been presented by Cohen & Welling [8]. This has been extended through *Steerable filters* [36] by Weiler *et al.*. *Scattering networks* [4] by Bruna & Mallat explicitly design invariance through wavelet convolutions and group averaging while some CNNs learn filter banks with local transformations [34]. Hinton [14] and Hinton *et al.* [13] try to impose a canonical frame of reference on the object independent of viewpoint variations. Tieleman [35] enhances this work to make the generative capsules capable of full affine transformations. *Spatial Transformer Networks* [17] explicitly allow transformation of data within the network without any manual intervention. *Inverse Compositional Spatial Transformer Networks* [23] improve this approach by implementing the image transformation using an iterative approach inspired by Lucas-Kanade optical flow algorithm. *Deformable Convolutional Networks* [9] have deformable convolution and RoI pooling layers which augment the spatial sampling locations with learnable offsets. This enhances transformation modelling capability of CNNS.

All of these approaches either try to model rotation invariant features to describe images or transform either the feature maps or convolution filters to achieve invariance. This motivates us to look for a more novel solution where the invariance is embedded in the convolution filter design itself. We propose to select a set of spatially symmetric filters and let the network learn linear combinations of them; so that the effective filters remain symmetric in construction.

2.2. Research Objective

The sole objective of this research work is to incorporate rotation invariant filters to convolutional neural networks. The aim is a generic solution to the problem of rotation that should not depend on the data and can be easily incorporated into existing architectures.

- *Can we achieve rotational invariance in neural networks if we use rotationally symmetric filter?*

This can be broken down into further sub-questions as:

1. Can we construct rotation invariant filters by learning linear combinations of fixed rotation filters?
2. Do our rotational filters make the network response stable against rotations in its input images?

2.3. Outline

The main focus of this report is the scientific paper in chapter 1. The thesis report follows up with a chapter on the basic theoretical knowledge of neural networks needed to understand the work done in this research. Then, we discuss the *Structured Receptive Fields* [16] which has heavy influence on the methods we have used. This is succeeded by a chapter on the rotational filters used in our work and construction of our effective kernels. The report concludes with a brief take on the additional experiments that have been performed but not included in the paper.

3 Background on Neural Networks

This chapter provides the background theoretical information on neural networks needed for clear insight and a better understanding of this work. This work researches on the ability of convolutional neural networks (CNNs) to learn rotational invariance and how that can improve image classification and semantic segmentation problems. This chapter starts with a basic overview of what neural networks are, and how they work. This is followed by a more detailed explanation on the working of CNNs in classification and segmentation. We will also introduce popular CNN architectures that are used in this work.

3.1. Neural Networks

A neural network or artificial neural network (ANN) is a connected acyclical graph of nodes called neurons. An artificial neuron is obtained by re-formulating a simplified function of biological neuron into a mathematical model [5]. Its schematic structure is shown in figure 3.1. The corresponding biological references are shown to highlight the parallels.

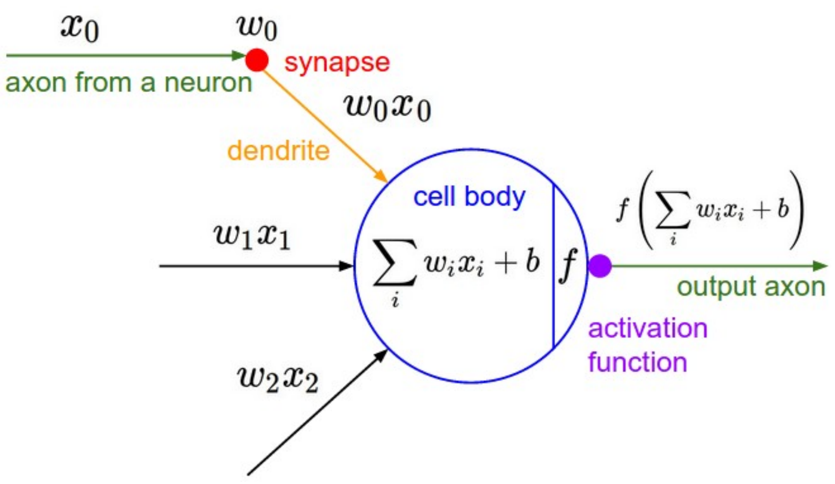


Figure 3.1: A mathematical model of an artificial neuron which is inspired by the neurological structure of the brain. The neuron has got 3 inputs, 3 learnable weights and a bias, and applies an activation to obtain the final output [18].

Each neuron in a network has n incoming inputs $X = [x_1 x_2 ... x_n]^T$ with weights $W = [w_1 w_2 ... w_n]$ and biases $B = [b_1 b_2 ... b_n]$. A neuron then computes

$$u = W.X + B = \sum_{i=0}^{n} w_i.x_i + b_i, \tag{3.1}$$

and forwards it through some nonlinearity known as an activation function, to generate the output.

Geometrically, the operation in Equation 3.1 can be interpreted as a linear transformation of the input vector X and then shifting the origin by the bias term. The computation $W.X+b$ is, hence, an affine transformation. A non-linear activation f is applied to the affine transform

$$o = f(u). \tag{3.2}$$

In figure 3.1 the neuron has got 3 inputs, 3 learnable weights and a bias, and applies an activation to obtain the final output.

A neural network is composed of interconnected neurons that exchange information between themselves. The connections have numerical weights that are tuned through back-propagation during training so that a trained network responds appropriately when presented with an image or pattern to recognize [18]. General neural networks are fully-connected i.e. neurons between two adjacent layers are connected to each other. Each layer has many neurons that respond to different combinations of inputs from the previous layers. Neurons within a layer do not share any connections.

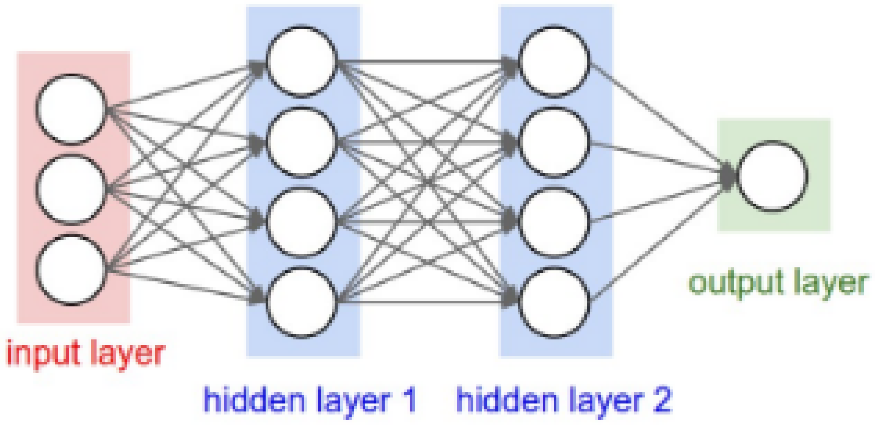


Figure 3.2: A regular 3-layer neural network with three inputs, two hidden layers of 4 neuron each and an output layer [18].

3.1.1. Activation

The activation function determines whether a neuron would fire or not and adds non-linearity to the output which helps to capture complex arbitrary functional mappings between inputs and output. Some common activations are Sigmoid, Hyperbolic tangent (Tanh) and Linear Rectified Units (ReLu) [30, 37].

The sigmoid function does not have a zero-centred output and has a *vanishing gradient* problem as the gradients saturate and die. Tanh solves the first problem but suffers from the latter. ReLu solves the gradient problem and widely accelerates the convergence of stochastic gradient descent [20].

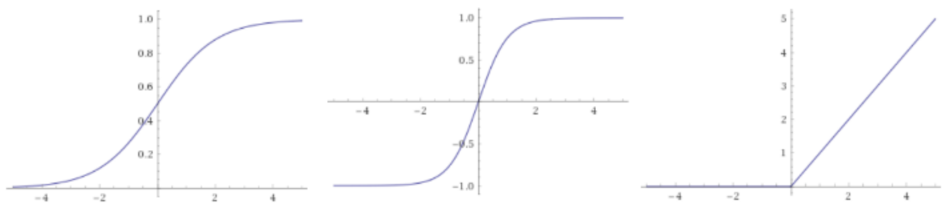


Figure 3.3: Sigmoid, Tanh and ReLu activation functions respectively from left to right

The ReLu function is defined as $A(x) = max(0, x)$ which thresholds the output at zero. But this can cause some neurons to never fire at all, known as the dead neuron problem. That can be handled by using Leaky ReLu or Randomized Leaky ReLu. Instead of thresholding at zero, Leaky ReLu has a small negative slope.

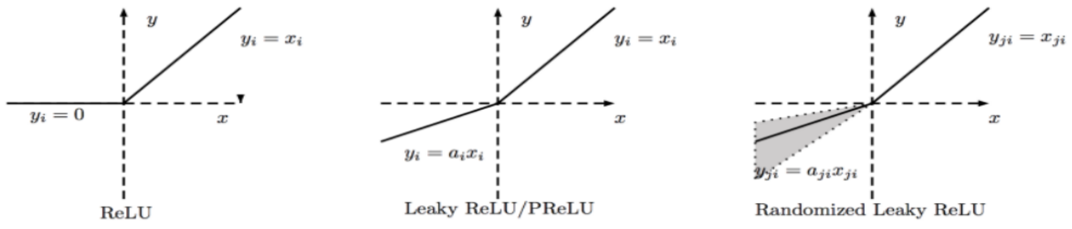


Figure 3.4: First figure on the left shows ReLu activation function which clips the negative part of the signal to zero. This can give rise to no activation from neurons, known as dead neurons. The dead neuron problem can be handled by Leaky/Randomized Leaky ReLu by leaking a small negative slope.

3.1.2. Optimizers and Loss Functions

The learning process of a neural network tries to minimize a loss function i.e. the difference between ground truth and network prediction. The network is initialized with random parameter values and the optimization process updates the parameters through gradient descent to significantly reduce the loss. The gradient of the loss function is calculated through *backpropagation*.

A loss function and an optimizer are chosen specifically to the problem at hand. A loss function should be differentiable so that it can be optimized through backpropagation. Some commonly used loss functions are mean square error, cross-entropy, hinge etc. Commonly used optimizers are Stochastic Gradient Descent (SGD), RMSprop, Adagrad, Adam etc.

3.2. Convolutional Neural Networks

A Convolutional Neural Network(CNN) is a special type of neural network which consists of one or more convolutional layers, followed by pooling and fully connected layers. The design of a CNN is motivated by the visual cortex in the brain and they are the default choice for handling image data in machine learning [5, 18]. A typical CNN architecture is shown in figure 3.5. The convolution and pooling layers are explained below.

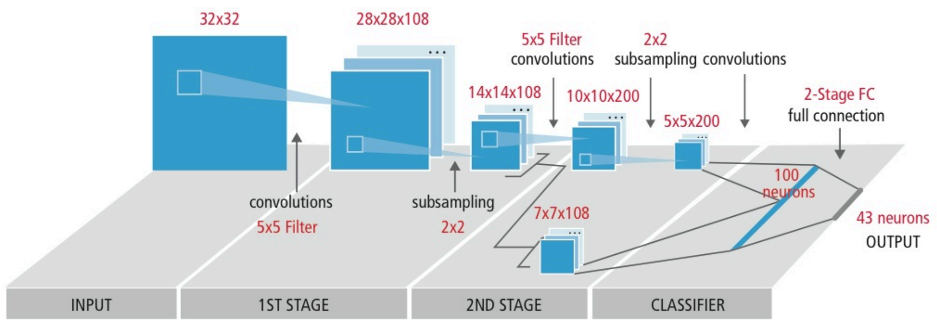


Figure 3.5: Typical block diagram of a CNN architecture. This architecture has 2 convolution layers followed by 2 pooling layers. The final layer is fully connected which provides classification output.

3.2.1. Convolution Layer

The convolution layer is the most important part of a CNN. Each convolution layer has a set of learnable filters which are spatially small but cover the depth of the input volume. The filters slide over the entire image and calculate dot products between the input values and the filter weights. This produces a 2-dimensional activation map which is the response of the filter to the given input. Through the training process, the network updates the filter weights and hence learns filters which can activate at certain visual features. A CNN usually has many such convolutional layers stacked together. Generally, the initial layers tend to learn edges, colours and orientations and progressively they learn blobs and more complex patterns.

Unlike fully-connected layers, convolutional layers connect neurons to only a local region of the input volume. The extent of spatial connectivity is known as the *receptive field* hyperparameter of the neuron, which is equal to the filter size. Some other important hyperparameters are the *depth* of the output filters, *stride* with which we slide the filters over the image, and the amount of *zero-padding* around the image border.

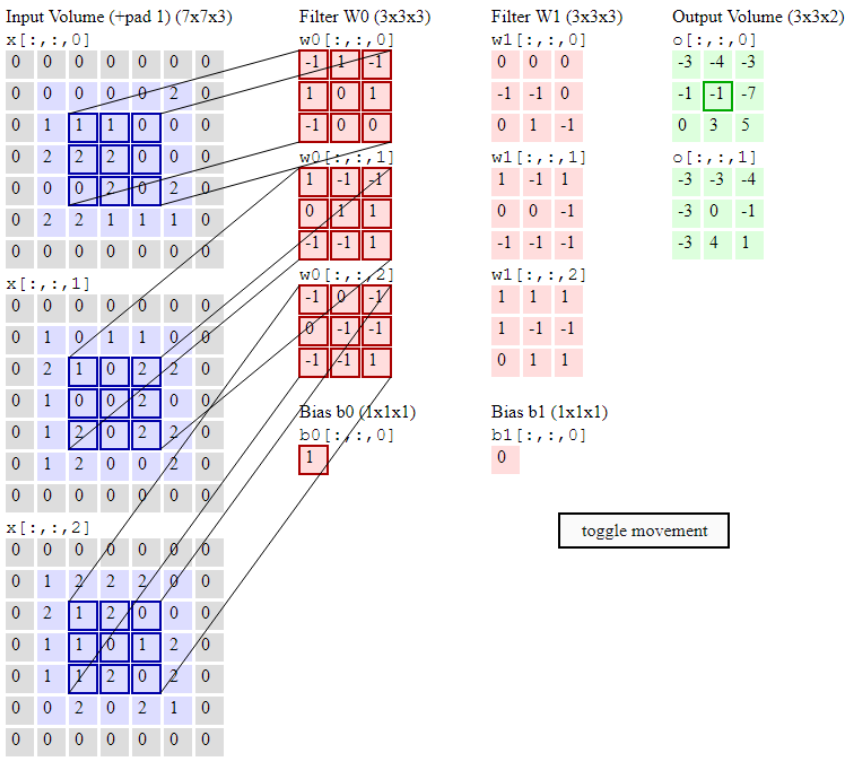


Figure 3.6: Visualization of a convolutional layer. The input volume is of size W1=5,H1=5,D1=3, and the convolution layer parameters are K=2,F=3,S=2,P=1. The output activations (green) are obtained by element wise multiplication of input (blue) with filter weights (red) and offset by bias. [18]

3.2.2. Pooling Layer

The convolution layers are generally followed by pooling layers. Pooling helps to reduce the spatial dimensions of the representation, and hence limit parameter size and control overfitting.

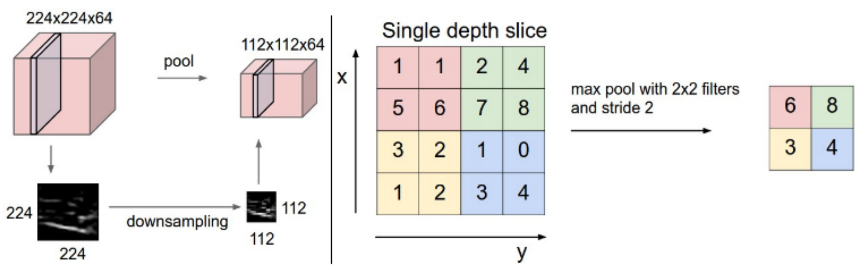


Figure 3.7: Left image shows how pooling operation reduces the spatial dimensions except the depth. Right image shows a representation of how MaxPool works [18].

The most common pooling operation is *MaxPooling* which takes the maximum value from a $n * n$ grid of the image, hence, reducing the spatial dimension n folds. This is presented in figure 3.7. Another popular pooling operation is *AveragePooling* which takes the average of the values in the grid space.

3.3. Deep Learning

Deep Learning is the subsection of machine learning that learns patterns from the data through a combination of neural network layers, stacked on top of each other, that learn successively complex feature representations. The *deep* in deep learning refers to the number of neural network layers used in the model, also known as the *depth* of the model. Any model that uses more than one or two network layers is generally referred to as deep. Modern deep learning models can contain tens or even hundreds of successive layers [7].

In this section, we briefly describe the role of deep learning in image classification and segmentation problems. The corresponding network architectures used in this work are also introduced.

3.3.1. Image Classification

Image classification is a common problem in computer vision where the task is to assign a particular label to an image, from a fixed category of labels. It is, generally, a supervised learning problem. In the recent past, deep learning has achieved tremendous progress in image classification problems, even surpassing human performance. Deep convolutional neural networks have been the popular choice in image classification since AlexNet [21] was introduced in 2012.

In our work, we have used the Network-In-Network (NIN) architecture [24] for image classification experiments. The following section gives a brief description of the same.

Network-in-Network

Conventional neural networks use linear filters followed by non-linear activation functions for feature extraction. The Network-In-Network architecture proposes a new take on how the convolution layers are designed. The NIN architecture introduces two new concepts in terms of network architecture which are explained below in details:

- **MLPConv Layer**: The NIN paper claims that the convolution filters as generalised linear models provide low levels of abstraction. They work on the assumption that the latent samples i.e. the variations of the features are linearly separable. But this is not always the case. Hence, the introduction of non-linear function approximator can work as better feature extractors.

  The paper introduces the concept of replacing the linear convolution filter with a multilayer perceptron (mlp). This new layer is called the `mlpconv` layer which maps the local receptive field to an output feature vector. This approach has two important benefits:

  - multilayer perceptron is compatible with backpropagation; so it can be easily introduced to existing architecture
  - multilayer perceptron can act as a deep model itself leading to a rich separation between latent features

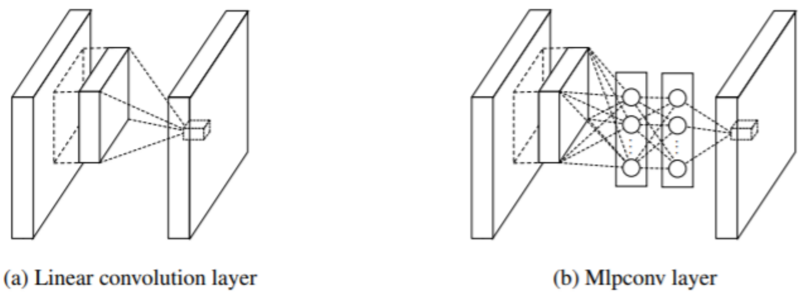


Figure 3.8: Comparison of linear convolution layer and mlpconv layer. The linear convolution layer includes a linear filter while the mlpconv layer includes a micro network [24].

- **Global Average Pooling**: Conventional CNN architectures have one or more fully-connected layers followed by a `softmax` activation to get the class probabilities. So they treat the convolution layers as feature extractors, followed by traditional neural network classifier. The issue with this approach is that it is hard to decode how the final fully-connected layers map to class probability. Also, these layers contain a large number of parameters to learn and hence are prone to overfitting.

  The paper proposes a new strategy called `global average pooling` to replace fully-connected layers. The final `mlpconv` layer generates as many feature maps as the number of classification categories. The `global average pooling` layer takes an average of each feature map and feeds that directly to the `softmax` activation layer. The advantages are:

  - it is a structural regularizer that enforces correspondence between feature maps and categories, so feature maps can be interpreted as category confidence.
  - it reduces parameter count as there are no weights to learn unlike fully-connected layers.
  - it sums up spatial information and hence, it is more robust to spatial translations of the input.

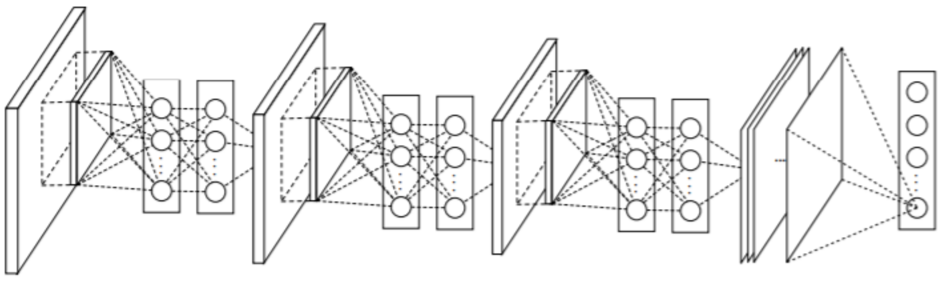


Figure 3.9: The overall structure of Network In Network with three mlpconv layers and one global average pooling layer [24].

1x1 Convolution: In terms of implementation, the `mlpconv` layers are mathematically equivalent to convolutions with $1x1$ spatial receptive field. $1x1$ convolution acts like coordinate-dependent transformation in the filter space. They pool features across various channels of a given layer. This is followed by a non-linear activation, usually ReLU.

3.3.2. Semantic Segmentation

Semantic segmentation is an understanding of image scene at pixel level i.e. every pixel is labelled with the class of its enclosing object or region. The aim of segmentation is to simplify or highlight the image for better analysis and understanding. It is typically used to locate objects and boundaries in images. In recent times, segmentation is an important problem in computer vision and it's applications involve autonomous driving, human-computer interaction, virtual reality, land-cover classification and so on.

With the advent and improvement in deep learning techniques, deep architectures are commonly used to tackle segmentation problems, generally using convolutional neural networks. In 2014, Fully Convolutional Networks (FCN) [26] by Long *et al.*from Berkeley, popularised CNN architectures for dense predictions without any fully connected layers. This allowed segmentation maps to be generated for images of any size and was also much faster compared to the patch classification approach used earlier. Almost all the subsequent state of the art approaches on semantic segmentation adopted this paradigm [6].

Apart from fully connected layers, one of the main problems with using CNNs for segmentation is pooling layers. Pooling layers increase the field of view and are able to aggregate the context while discarding the 'where' information. However, semantic segmentation requires the exact alignment of class maps and thus, needs the 'where' information to be preserved [6].

This gave rise to the encoder-decoder architecture where the encoder gradually reduces the spatial dimensions with pooling while the decoder gradually recovers the original dimension and object details. In our work, we have used a popular encoder-decoder architecture called the U-Net [32] which is briefly described below.

U-Net

U-Net is a popular end-to-end encoder-decoder architecture for semantic segmentation. This was introduced for bio-medical image segmentation and has outperformed prior best methods on segmentation. This architecture is built upon the concept of Fully Convolutional Network (FCN) and modified to get better segmentation results.

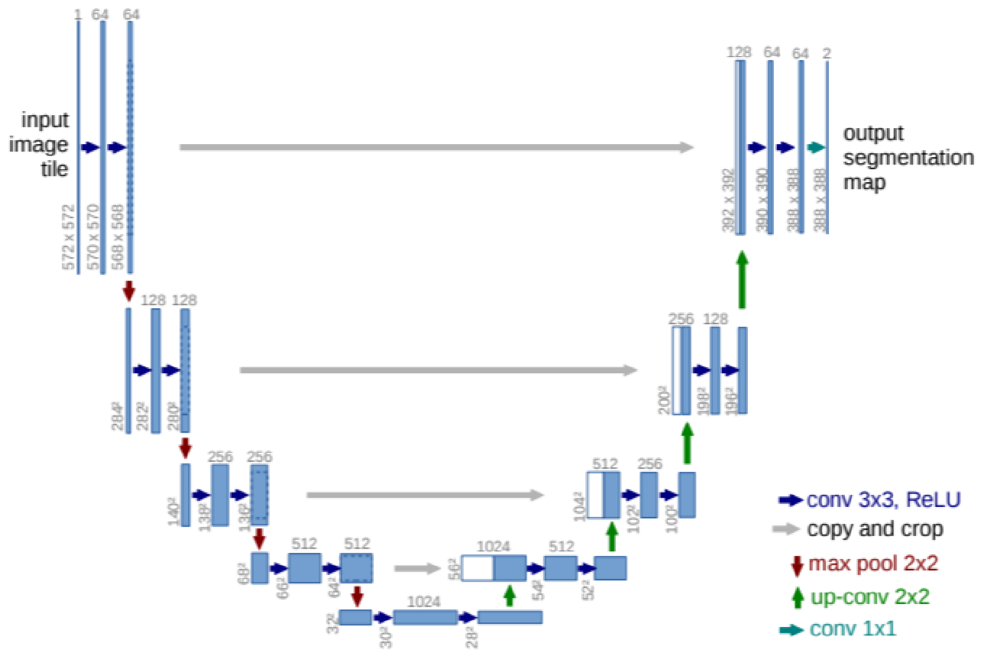


Figure 3.10: U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations [32].

The main differences of the U-Net from FCN is that (1) it is symmetric and (2) there are skip connections between the downsampling and upsampling path which are concatenated together. The U-Net architecture has 3 parts

- **Contracting/Downsampling**: The downsampling path is composed of 4 blocks, each having
  - 2 (3x3) Convolution + activation with batch normalization
  - A (2x2) Max Pooling
- **Bottleneck**: The bottleneck has 2 (3x3) convolutional layers with batch normalization and dropout
- **Expanding/Upsampling**: The upsampling path is composed of 4 blocks, each having
  - Deconvolution with stride=2
  - Concatenation of corresponding skip connections from the downsampling path
  - 2 (3x3) Convolution + activation with batch normalization

Upsampling:
The upsampling operator is used to replace the pooling operations and hence, incorporate the location information in the network. They also increase the spatial resolution of the output so as the output of the network has the same dimension as the input images. In the upsampling layer, a large number of feature channels are used so that the network can propagate information to the higher resolution layers.

Skip Connections:
In order to improve localisation, high-resolution feature maps from the encoder path are concatenated with the upsampled output. The corresponding feature maps need to be of the same spatial dimensions, which is handled by the upsampling layers. The following convolution layer after the concatenation can learn to assemble more precise output based on the information from the skip connection layers.

The U-Net combines location information from encoder path and contextual information from the decoder path to obtain good segmentation maps. The lack of fully-connected layers means images can be input to the network independent of size.

4 Structured Receptive Fields

This chapter gives a brief description of a type of convolutional architecture known as the *Structured Receptive Field* [16]. The filters in a structured receptive field can be constructed from a fixed set of basis filters through parametric learning of their linear combinations. The chapter describes their theoretical conception and implementation in a convolutional neural network.

The chapter starts with an explanation of the scale-space theory in section 4.1 that motivates the inception of structured receptive fields. The following section 4.2 discusses the mathematical construction of structured receptive filters using Gaussian derivative basis filters. We show how convolution filters can be expressed as parameterised functions where the parameters define the filter weights. The next section 4.3 describes the method of implementing the filters in a structured convolution layer. The last section 4.4 reasons the choice of architecture for the structured receptive fields and explains why our method of implementation is more generic towards the choice of architecture.

4.1. Scale Space Theory

In scale-space theory, the features present in an image is expressed as a parameterised function with a continuous scale parameter. Images are convolved with filters of increasing size, while linearly decreasing the spatial resolution at every step. This uniquely encodes information from the image. The Gaussian is an important mathematical function that can change its spatial extent according to its standard deviation, which is a continuous parameter. Gaussian and Gaussian derivatives do not introduce any artefacts and can be used to represent any image realistically. In other words, any image can be decomposed into a linear combination of Gaussian and Gaussian derivative functions. So, the Gaussian derivative filters are appropriate to be used as a basis to describe image features.

In CNNs, the convolutional filters capture the image features while the pooling layers reduce the spatial extent at each level but increase the receptive field of the filter kernel. Thus, the local structure in the image is encoded into increasingly smaller feature vectors; while the receptive field of the filters enlarges to better comprehend the global structure. This is similar in concept to the scale-space and hence, gives rise to the idea of decomposing images into a linear combination of fixed basis filters in CNNs.

Structured receptive field, thus tries to incorporate fixed filter bases as function priors like Scattering Network [4], but still maintains the learning capacity of CNNs to learn the linear combination of the bases to formulate effective convolution filters. This leads to the structured receptive field neural network, called as RFNN in the paper.

4.2. Structured Receptive Fields

Structured receptive fields consider images as a function in scale-space. So, the convolutional kernels can also be expressed as parameterised functions. Any filter can be approximated by the Taylor expansion around a upto order M as

$$F(x) = \sum_{m=0}^{M} \frac{F(a)^m}{m!}(x-a)^m. \tag{4.1}$$

Exact derivatives of this function can be obtained through convolution with Gaussian derivatives like

$$G(.;\sigma) * F(x) = \sum_{m=0}^{N} \frac{(G(.;\sigma)^m * F)(a)}{m!}(x-a)^a, \tag{4.2}$$

where $*$ denotes convolution, $G(.;\sigma)$ is a Gaussian kernel with scale σ and $G(.;\sigma)^m$ is the m^{th} order Gaussian derivative with respect to its spatial variable. This shows that a pixel representation of an image in a CNN can be equivalently described as a convolution with a basis of weighted Gaussian derivatives. This forms the mathematical basis of the structured receptive fields.

The Gaussian derivatives at different order can be obtained by multiplying orthogonal Hermite polynomials H_m with the Gaussian envelope as

$$G(.;\sigma)^m = (-1)^m \frac{1}{\sqrt{\sigma^m}} H_m(\frac{x}{\sigma\sqrt{2}}) \cdot G(x;\sigma). \tag{4.3}$$

The Hermite polynomials used here are ”physicists” Hermites which are expressed in recurrence equations as

$$\begin{aligned} H_i(x) &= 2xH_{i-1}(x) - 2(i-1)H_{i-2}(x), \\ H_1(x) &= 2x, \\ H_0(x) &= 1. \end{aligned} \tag{4.4}$$

The representation in equation 4.2 allows to approximate any image’s local geometry at location x with scale σ up to any order of precision M. So, at infinite precision, an RFNN translates into a general CNN. For human visual perception, an order of 4 can capture all important information [19]. This mitigates the effect of the loss in universality that occurs due to learning just a subset of all possible filters that a CNN can learn. The structured receptive fields can, thus, capture most of the filters of interest.

4.3. Structured Convolution Layer

In structured receptive fields, the 2D convolution kernel is expressed as a weighted linear combination of unique Gaussian derivative basis functions. The weights of the linear combination are learnt through backpropagation. A convolution kernel function $F(x,y)$, where x, y denote the spatial dimensions, can be expressed as a linear combination of i unique basis Gaussian derivatives ϕ

$$F(x,y) = \alpha_1\phi_1 + \cdots + \alpha_n\phi_i, \tag{4.5}$$

where $\alpha_1, ..., \alpha_i$ are the parameters being learnt.

The network learns the α values by mini-batch stochastic gradient descent and the derivatives of the loss function are calculated in terms of the α through backpropagation. The α weights and the basis filters are independent of each other and hence, the basis filters are never learnt in this process.

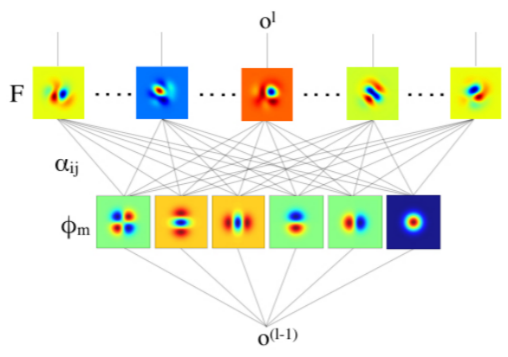


Figure 4.1: Basic building block of Receptive Field Neural Networks. A linear combination of a limited basis filter set ϕ_m yields an arbitrary number of effective filters. In this example, ϕ_m consists of Gaussian derivatives upto the 2^{nd} order. The weights α_{ij} are learned by the network. Image source [16].

In the SRF paper, the filter banks are learnt as a combination of the fixed basis filters with a $1x1$ convolution layer with the depth equal to the basis order. The image is convolved with the basis filter kernels and the resultant feature maps are recombined by propagating them via a $1x1$ convolution layer which outputs feature maps with the required shape for the next convolution layer. In this way, the effective kernel is never actually computed. The feature maps generated after convolution with basis filters and recombining them through cross-channel pooling are equivalent to the feature maps after convolution with actual effective filters. This way, structured receptive field approach cuts down hugely on the number of parameters learnt by the network. This leads to their impressive classification performance on very small datasets. But this method puts on the constraint of always having $1x1$ convolution layers in the architecture.

In our implementation, we write our custom convolution layers which formulate the 2D convolution kernel as the linear combination of the basis with randomly initialised weights (α). We force the α values to be the only trainable parameters and the basis filters are kept constant. The custom convolutional layer intakes the basis filters and computes the convolutional kernel as in equation 4.5 according to the input and output channel dimensions. In this way, the effective filters are actually calculated in our custom convolution layer with random α initialisation. During training, only the α parameters are updated by backpropagation and we get the desired learnt kernel. Though this increases the parameter count of the network, it makes it generic to use in any deep learning architecture where we can just replace the standard convolution layer with our custom one which takes the basis filter set as an extra parameter. Our method also makes it very easy to implement in some deep learning frameworks like Keras; where otherwise you have to take care of the kernel shape at each layer.

4.4. Network

The SRF paper uses the Network in Network (NiN) architecture [24] because it suits their implementation of the structured convolution layers. The NiN architecture has spatial convolution layers followed by $1x1$ convolutions which form non-linear combinations of the spatial layer outputs. Also, the NiN implementation does not have a fully-connected layer which means all parameters are concerned with the combination of basis filters.

The RFNN version of NiN replaces the spatial convolution layers with the structured convolution layers. Each convolution layer is followed by a non-linear activation. No parameters are learnt in the basis set. The $1x1$ convolution layers form linear combinations of the basis set according to the equation 4.5. There is no non-linearity between ϕ_m and α_{ij} layer in the RFNN. The number of filters in the basis convolution layer depends on the order and scale of the basis set chosen.

Our implementation of the SRF is slightly different in the sense that the custom convolutional layer already forms the linear combination of the basis set according to equation 4.5.

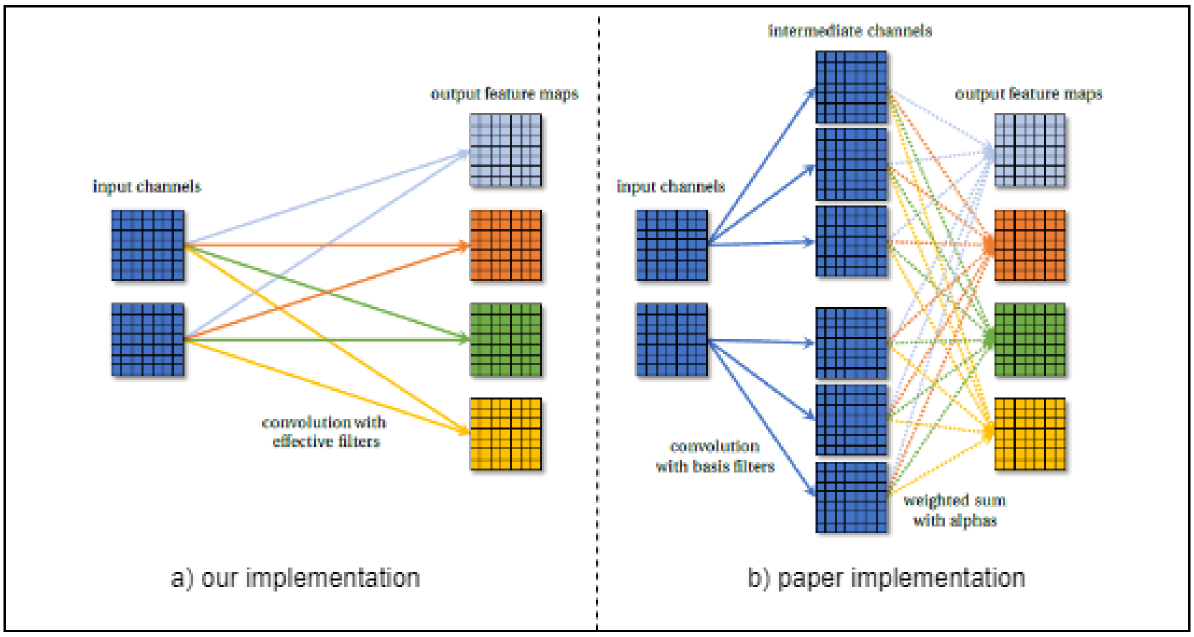


Figure 4.2: Comparison of the implementation of Structured Receptive Fields. Figure 4.2.a on the left shows our method of implementation where we directly compute the effective filters and then do the convolution with the input feature maps. on the right, 4.2.b shows the approach from the paper where they input feature maps are convolved with basis filters and then linearly combined together to get the output feature maps. This implementation needs less parameter learning and hence is faster. Our implementation is easier to integrate into any deep-learning framework. Image source [12].

The number of output layers needed can be specified exactly like in a normal convolution layer. So, the $1x1$ convolution layers form further linear combinations of the structured convolutional kernels. We can add non-linearity even to the custom layer as it already computes the effective filter. This implementation is, thus, generic and can easily replace any normal convolution layer in a deep learning architecture.

5 Rotational Invariant Filters

This chapter introduces the rotational filters that have been used in this work. Our requirement for the research is a set of filters

- which have spatial rotational symmetry
- their spatial extent can be controlled by a single continuous parameter

The first condition is necessary to make our effective kernels invariant to rotation. The second condition allows us to use these filters as structured receptive fields [16].

We choose the Schmid filter basis which satisfies both the conditions. Section 5.1 introduces the Schmid filter bank and describes their mathematical formulation. The following section 5.2 describes how the filters have been incorporated as structured rotational fields to construct effective rotation invariant filters. We visualise the constructed filters to check their rotational symmetry.

5.1. Schmid filter bank

The Schmid filter bank consists of 13 rotationally invariant filters introduced by Schmid to get rotation invariant descriptors for texture-like visual structure recognition [33]. These are Gabor-like filters that combine frequency and scale together:

$$F(x, y, \tau, \sigma) = F_0(\tau, \sigma) + \cos \frac{\sqrt{x^2 + y^2}\pi\tau}{\sigma} e^{-\frac{x^2+y^2}{2\sigma^2}}, \tag{5.1}$$

where σ is the standard deviation of the Gaussian envelope and τ is the number of cycles of the harmonic function within the Gaussian envelope. $F_0(\tau, \sigma)$ is added to obtain a zero DC component, making it robust to illumination changes.

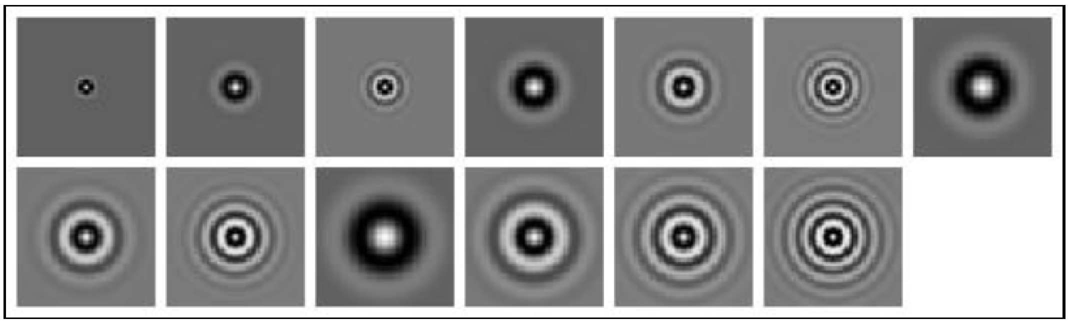


Figure 5.1: Schmid(S) filter bank; 13 rotationally invariant isotropic "Gabor-like" filters [3] with a spatial resolution of 49.

As seen in figure 5.1, the filters have spatial symmetry, thus conforming to our first condition. Also, since the filter is constructed as a Gaussian envelope multiplied by a cosine, the spatial extent is governed by the variance i.e. the σ parameter of the filters; hence abiding by the second condition. The 13 filters in the Schmid filter bank have pre-determined parameter values with the σ, τ pair taking values (2,1), (4,1), (4,2), (6,1), (6,2), (6,3), (8,1), (8,2), (8,3), (10,1), (10,2), (10,3) and (10,4). Smaller scales use smaller τ values to avoid high-frequency responses. The Python code to generate the filters has been adopted from the Matlab code presented in [3].

The σ parameter values of the original filters are found to be too high for our application with kernel sizes of the order $3x3$ and $5x5$. We have tried different σ values and came up with the best set suitable for our experiments which are [0.23, 0.33, 0.33, 0.67, 0.67, 0.67, 1, 1, 1, 1.67, 1.67, 1.67, 1.67].

5.2. Constructing rotational invariant filters

Our idea is to incorporate the Schmid filter basis into the structured receptive fields. We compute the effective 2D convolution kernels from the Schmid filters, rather than the Gaussian derivatives. In the structured receptive fields, the construction of an effective 2D kernel $F(x, y)$, where x, y denote the spatial dimensions, is governed by the equation

$$F(x, y) = \alpha_1 \phi_1 + \cdots + \alpha_n \phi_i; \tag{5.2}$$

where ϕ_i denotes i unique Schmid basis filters here and α denotes the weights of linear combination.

The method of implementation of the rotational kernels is exactly the same as the Structured Convolution Layer explained in 4.3. The figure 5.2 visualises the original basis and some of the recombined filters obtained from them.

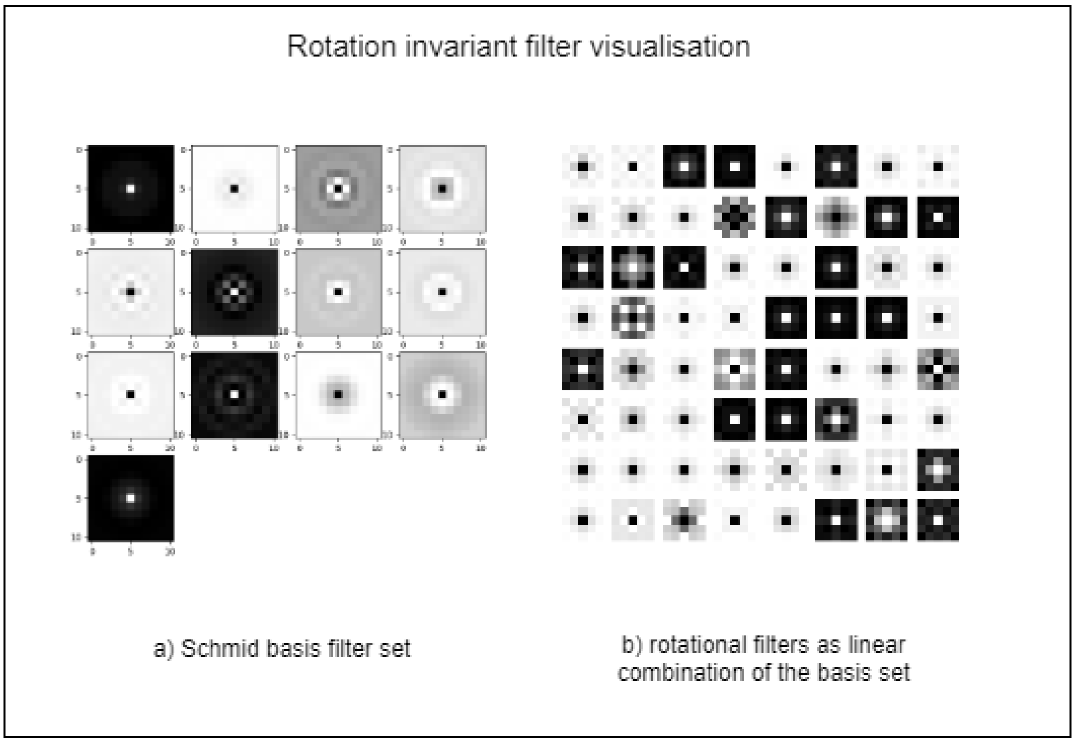

Rotation invariant filter visualisation
a) Schmid basis filter set
b) rotational filters as linear combination of the basis set

Figure 5.2: a) Schmid(S) filter bank, b) effective rotational kernels obtained from learning the linear combination of the basis filters. We can see that all the filters have rotational symmetry.

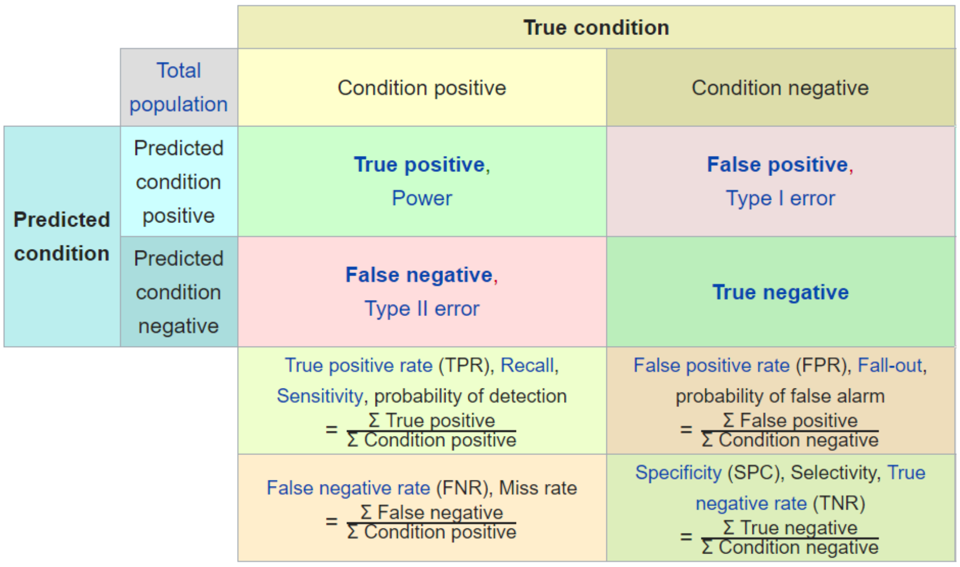
6 Performance Evaluation Method

We use the standard classification accuracy measure for our classification experiments and the mean intersection over union measure for our segmentation experiments. This chapter explains briefly the mathematical formulation of the metrics and their implementation in our problems.

6.1. Classification

Classification accuracy is a straight-forward metric that calculates the ratio of correctly predicted instances among all instances present.

In pattern recognition, classification accuracy is measured by the parameters true positives, true negatives, false positives, and false negatives [31]. They are explained briefly in the figure 6.1.



		True condition	
	Total population	Condition positive	Condition negative
Predicted condition	Predicted condition positive	**True positive**, Power	**False positive**, Type I error
	Predicted condition negative	**False negative**, Type II error	**True negative**
		True positive rate (TPR), Recall, Sensitivity, probability of detection $= \frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$	False positive rate (FPR), Fall-out, probability of false alarm $= \frac{\Sigma \text{ False positive}}{\Sigma \text{ Condition negative}}$
		False negative rate (FNR), Miss rate $= \frac{\Sigma \text{ False negative}}{\Sigma \text{ Condition positive}}$	Specificity (SPC), Selectivity, True negative rate (TNR) $= \frac{\Sigma \text{ True negative}}{\Sigma \text{ Condition negative}}$

Figure 6.1: Different accuracy and error measures under classification context.

6.1.1. Accuracy

Classification accuracy is measured as the true-positive rate, also known as recall or sensitivity and generally calculated as

$$recall = \frac{TP}{TP + FN}, \tag{6.1}$$

where TP and FN are true positive and false negative measures respectively.

6.2. Segmentation

Our segmentation problem calculates accuracy through the mean intersection over union (mIoU) metric; which is a very common metric in use for semantic segmentation.

6.2.1. Intersection over Union

Intersection over Union, also known as the Jaccard index or the Jaccard similarity coefficient, is a evaluation metric used to compare the similarity or difference between two sample sets, introduced by botanist Paul Jaccard [15]. given two bounding boxes as sample sets, this index is defined as the size of the intersection divided by the size of the union of the sample sets:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}. \tag{6.2}$$

If A and B are both empty, we define $J(A,B) = 1$. So, $0 \leq J(A,B) \leq 1$.

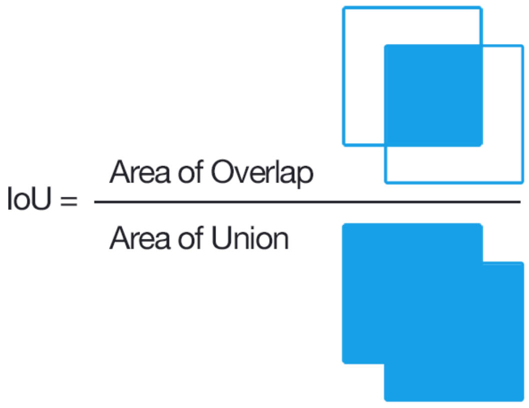


Figure 6.2: IoU measure expressed as ratio of intersection over union of prediction and ground-truth.

In segmentation problems, the two sample sets are the annotated image (the ground truth) and the predicted image as the network output. The numerator computes the area of overlap between the ground-truth and the prediction; while the denominator is the area encompassed by both the prediction image and the ground-truth image. This is shown in figure 6.2

In terms of classification accuracy measures, the intersection over union, for m images and each $j \in N$ classes, can be expressed as

$$IoU_j = \frac{\sum_{i=1}^{m} TP_{ij}}{\sum_{i=1}^{m} TP_{ij} + \sum_{i=1}^{m} FP_{ij} + \sum_{i=1}^{m} FN_{ij}}; \tag{6.3}$$

where TP_{ij} is the total number of correctly predicted pixels in image i that belong to class j; FP_{ij} is the total number of pixels in image i wrongly classified in class j; and FN_{ij} is the total number of pixels in image i that are wrongly predicted as any class except j. The final score is calculated as average over the N classes, expressed as

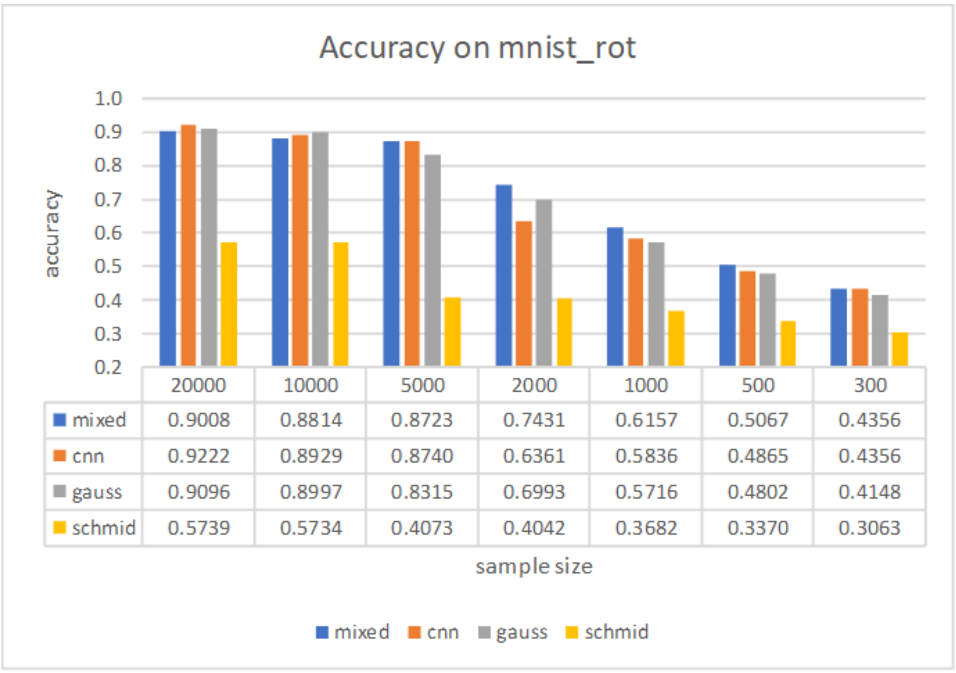
$$mIoU = \frac{1}{N} \sum_{j=1}^{N} IoU_j. \tag{6.4}$$

7
Additional Experiments

This chapter describes the additional experiments and experimental details that are not included in the paper.

7.1. MNIST Rotated

We have tried to check classification accuracy on the rotated MNIST dataset [2]. The digits are rotated by an angle generated uniformly between 0 and 2π radians. This experiment is exactly similar to experiment 1 described in the scientific paper section 4.1.3. The models follow the Network-in-Network architecture and are trained for 1200 epochs with an Adam optimizer with default parameters. The results are shown below in figure 7.1.



	20000	10000	5000	2000	1000	500	300
🟦 mixed	0.9008	0.8814	0.8723	0.7431	0.6157	0.5067	0.4356
🟧 cnn	0.9222	0.8929	0.8740	0.6361	0.5836	0.4865	0.4356
⬜ gauss	0.9096	0.8997	0.8315	0.6993	0.5716	0.4802	0.4148
🟨 schmid	0.5739	0.5734	0.4073	0.4042	0.3682	0.3370	0.3063

Figure 7.1: Acuuracy results on MNIST rotated dataset. cnn model is a normal NiN architecture; gauss and schmid have spatial convolution filters replaced by Gaussian derivative and Schmid filters respectively; mixed has both Gaussian and Schmid filters concatenated. Our schmid model performs worse than the other models that learn better representation from variations in the input data.

The results do not show significant performance for our Schmid filters as in experiment 1 described in the paper. Our intuition is that the CNN and Gaussian models have better representative power and hence, learn better features from the data itself.

7.2. Class distribution in segmentation problem

We used the DeepGlobe Land Cover classification dataset [1] for our segmentation experiments. Understanding the pixel distribution of different classes in the dataset was essential for our experiments on rotational invariance. Since, convolutional neural networks learn positional information from the input images, the learned models can have certain bias to specific classes in specific locations of the predicted output images. So, it was important for us to check the frequency of positional distribution of different classes over the entire dataset. If the distribution is not uniform enough, it is imperative to say that conventional CNNs would learn that bias during training.

The pixel proportions of different classes is given in table 7.1

Class proportions in DeepGlobe Land Cover dataset		
class	pixel count	proprotion
Urban	642.4M	9.35
Agriculture	3898.0M	56.76
Rangeland	701.1M	10.21
Forest	944.4M	13.75
Water	256.9M	3.74
Barren	421.8M	6.14
Unknown	3.0M	0.04

Table 7.1: Class proportions in DeepGlobe Land Cover dataset [10]. There is a significant difference in proportion of existence; so we try to find the uniformity of distribution also.

We visualised frequency distribution maps for each of the classes in our segmentation problem; except the 'unknown' class. Each map is normalised and hence provides a good idea about the class distributions.

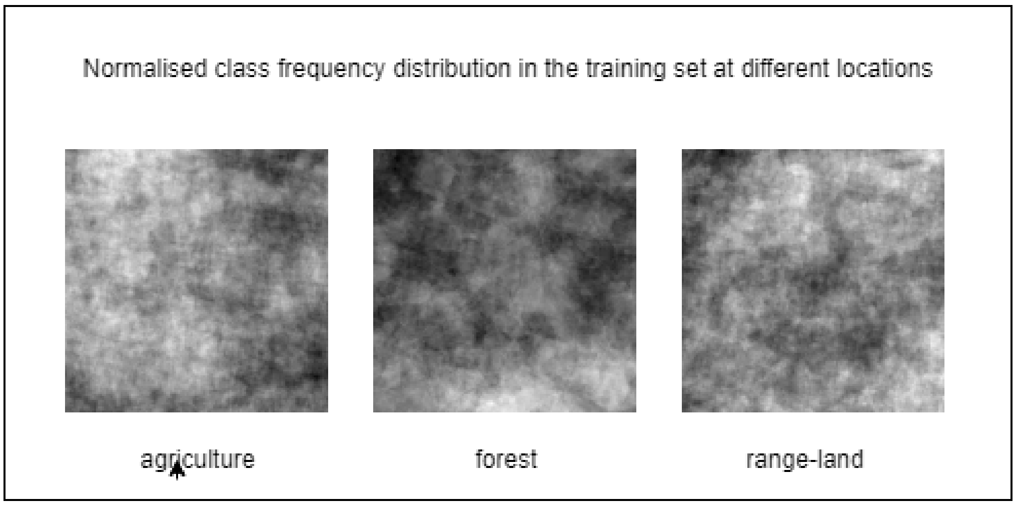


Figure 7.2: Frequency distribution at every spatial location over the entire training set for specific classes. We can see that distribution is not uniform. This means the dataset has some bias towards certain class pixels at certain spatial locations.

The visualisations show that there is certain bias of different classes towards specific locations in the dataset. That means the dataset does not have exactly uniform distribution of different classes; hence, the convolutional neural network would have some bias in training and therefore, the trained model would not be rotation invariant.

Bibliography

[1] Deepglobe cvpr 2018 - satellite challenge. http://deepglobe.org/. Accessed: 2018-11-05.

[2] Variations on the mnist digits. http://www.iro.umontreal.ca/~lisa/twiki/bin/view.cgi/Public/MnistVariations. Accessed: 2018-11-08.

[3] Visual geometry group, department of engineering science, university of oxford. http://www.robots.ox.ac.uk/~vgg/research/texclass/filters.html. Accessed: 2018-11-05.

[4] Joan Bruna and Stephane Mallat. Invariant scattering convolution networks. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(8):1872–1886, August 2013. ISSN 0162-8828. doi: 10.1109/TPAMI.2012.230. URL http://dx.doi.org/10.1109/TPAMI.2012.230.

[5] Cadence. Using convolutional neural networks for image recognition. .

[6] Sasank Chilamkurthy. A 2017 guide to semantic segmentation with deep learning. http://blog.qure.ai/notes/semantic-segmentation-deep-learning-review. Accessed: 2018-11-07.

[7] François Chollet. *Deep Learning with Python*, volume 1st. Manning Publications Co., 2017.

[8] Taco S. Cohen and Max Welling. Group equivariant convolutional networks. *CoRR*, abs/1602.07576, 2016. URL http://arxiv.org/abs/1602.07576.

[9] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. *CoRR*, abs/1703.06211, 2017. URL http://arxiv.org/abs/1703.06211.

[10] Ilke Demir, Krzysztof Koperski, David Lindenbaum, Guan Pang, Jing Huang, Saikat Basu, Forest Hughes, Devis Tuia, and Ramesh Raskar. Deepglobe 2018: A challenge to parse the earth through satellite images. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2018.

[11] Nikita Dvornik, Julien Mairal, and Cordelia Schmid. Modeling Visual Context is Key to Augmenting Object Detection Datasets. In *ECCV 2018 - European Conference on Computer Vision*, pages 1–17, Munich, Germany, September 2018. URL https://hal.archives-ouvertes.fr/hal-01844474.

[12] Sten Goes. Learning the scale of image features in convolutional neural networks, 2017.

[13] Geoffrey E. Hinton, Alex Krizhevsky, and Sida D. Wang. Transforming auto-encoders. In *Proceedings of the 21th International Conference on Artificial Neural Networks - Volume Part I*, ICANN'11, pages 44–51, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 978-3-642-21734-0. URL http://dl.acm.org/citation.cfm?id=2029556.2029562.

[14] Geoffrey F. Hinton. A parallel computation that assigns canonical object-based frames of reference. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'81, pages 683–685, San Francisco, CA, USA, 1981. Morgan Kaufmann Publishers Inc. URL http://dl.acm.org/citation.cfm?id=1623264.1623282.

[15] Paul Jaccard. The distribution of the flora in the alpine zone.1. *New Phytologist*, 11 (2):37–50, 2 1912. ISSN 1469-8137. doi: 10.1111/j.1469-8137.1912.tb05611.x. URL https://doi.org/10.1111/j.1469-8137.1912.tb05611.x.

[16] Jörn-Henrik Jacobsen, Jan C. van Gemert, Zhongyu Lou, and Arnold W. M. Smeulders. Structured receptive fields in cnns. *CoRR*, abs/1605.02971, 2016. URL `http://arxiv.org/abs/1605.02971`.

[17] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial transformer networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, pages 2017–2025, Cambridge, MA, USA, 2015. MIT Press. URL `http://dl.acm.org/citation.cfm?id=2969442.2969465`.

[18] Andrej Karpathy. Notes for cs231n convolutional neural networks for visual recognition, 2015. Stanford University [2015].

[19] J J Koenderink and A J van Doom. Representation of local geometry in the visual system. *Biol. Cybern.*, 55(6):367–375, March 1987. ISSN 0340-1200. doi: 10.1007/BF00318371. URL `http://dx.doi.org/10.1007/BF00318371`.

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*, pages 1106–1114, 2012. URL `http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks`.

[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA, 2012. Curran Associates Inc. URL `http://dl.acm.org/citation.cfm?id=2999134.2999257`.

[22] Karel Lenc and Andrea Vedaldi. Understanding image representations by measuring their equivariance and equivalence. *CoRR*, abs/1411.5908, 2014. URL `http://arxiv.org/abs/1411.5908`.

[23] Chen-Hsuan Lin and Simon Lucey. Inverse compositional spatial transformer networks. *CoRR*, abs/1612.03897, 2016. URL `http://arxiv.org/abs/1612.03897`.

[24] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *CoRR*, abs/1312.4400, 2013. URL `http://arxiv.org/abs/1312.4400`.

[25] Tony Lindeberg. Feature detection with automatic scale selection. *Int. J. Comput. Vision*, 30(2):79–116, November 1998. ISSN 0920-5691. doi: 10.1023/A:1008045108935. URL `https://doi.org/10.1023/A:1008045108935`.

[26] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *CoRR*, abs/1411.4038, 2014. URL `http://arxiv.org/abs/1411.4038`.

[27] David G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2*, ICCV '99, pages 1150–, Washington, DC, USA, 1999. IEEE Computer Society. ISBN 0-7695-0164-8. URL `http://dl.acm.org/citation.cfm?id=850924.851523`.

[28] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60:91–110, 2004.

[29] Krystian Mikolajczyk and Cordelia Schmid. A performance evaluation of local descriptors. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(10):1615–1630, October 2005. ISSN 0162-8828. doi: 10.1109/TPAMI.2005.188. URL `https://doi.org/10.1109/TPAMI.2005.188`.

[30] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 807–814, USA, 2010. Omnipress. ISBN 978-1-60558-907-7. URL `http://dl.acm.org/citation.cfm?id=3104322.3104425`.

[31] David L. Olson and Dursun Delen. *Advanced Data Mining Techniques*. Springer Publishing Company, Incorporated, 1st edition, 2008. ISBN 3540769161, 9783540769163.

[32] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015. URL `http://arxiv.org/abs/1505.04597`.

[33] C. Schmid. Constructing models for content-based image retrieval. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 39–45, 2001.

[34] Kihyuk Sohn and Honglak Lee. Learning invariant representations with local transformations. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ICML'12, pages 1339–1346, USA, 2012. Omnipress. ISBN 978-1-4503-1285-1. URL `http://dl.acm.org/citation.cfm?id=3042573.3042745`.

[35] Tijmen Tieleman. *Optimizing neural networks that generate images*. PhD thesis, University of Toronto, 2014.

[36] Maurice Weiler, Fred A. Hamprecht, and Martin Storath. Learning steerable filters for rotation equivariant cnns. *CoRR*, abs/1711.07289, 2017. URL `http://arxiv.org/abs/1711.07289`.

[37] Bing Xu, Ruitong Huang, and Mu Li. Revise saturated activation functions. *CoRR*, abs/1602.05980, 2016. URL `http://arxiv.org/abs/1602.05980`.