Bachelor's project

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

Marcel Heshmati Rød

Group Equivariant Convolutional Neural Networks

Bachelor's project in Mathematics
Supervisor: Elena Celledoni, Brynjulf Owren
June 2020

NTNU
Norwegian University of
Science and Technology

Abstract

Detecting and classifying features in an image is an important subtask in building algorithms that interact with the real world. In modern applications, this task is solved using deep learning with convolutional layers. A problem with this method is that it responds in unpredictable ways when an input image is rotated, leading to unstable outputs if nothing is done to mitigate it. While this problem is usually solved by giving the learning algorithms more data, there are other alternatives. This thesis explores ways to exploit rotational symmetries by algorithmic construction, using both group convolutions and harmonic networks, and attempts to measure the benefits of using them.

The resulting algorithms learn faster on rotationally invariant classification problems but are still outperformed by traditional methods when substantial data augmentation is used. Harmonic networks are an architecture that lends itself to rotated inputs, improving on the test error of a basic convolutional network on the rotated MNIST dataset from $\sim 5.0\%$ to $\sim 1.7\%$. Group convolutions and harmonic networks have several advantages during the training phase and often do better than traditional methods when data is limited.

Acknowledgements

First of all, I want to thank my thesis supervisors Brynjulf and Elena. When they suggested to me the topic of group equivariance in neural networks, I knew little about the field. Their technical expertise and responsiveness have aided me greatly in my journey to understand everything from the major outline of the thesis to the fine details. Additionally, their support and guidance has welcomed me into the world of academia, and continues to encouraged me to pursue it as a career choice.

I also want to thank Rebecca for helping me write, and for supporting me during hardships. Her incredible communication skills and technical understanding have been invaluable. When proofreading, Rebecca has not only caught many slip-ups, she has also taught me a lot about my writing and clarity.

Contents

Chapter 1

Introduction

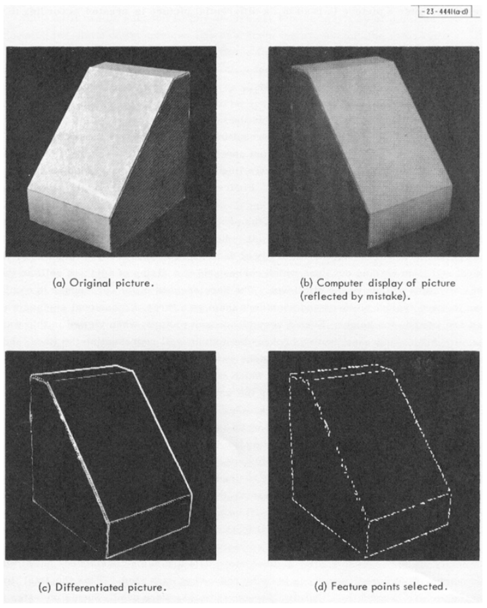
For humans and many other animals, visual perception is the most important sense. Almost every aspect of our lives relies on vision, from driving to work in the morning to recognizing a friend in a crowd.

The process of perception is an automatic and mostly unconscious process to us and is rarely duly appreciated. The amount of biological engineering involved in every facet of it is staggering. In order to read this sentence, tens of millions of photons in the visual spectrum have to scatter off the page, get focused by the iris, and hit the retina, initiating a chemical chain reaction that feeds into to the brain. This, however, is only the beginning of perception.

On their own, the signals entering the brain have no more meaning than a list of numbers, yet these signals are sufficient input for everything from reading a book to driving a car to playing a video game. How is this possible? The problem of explaining visual perception, let alone recreating it, has been baffling scientists for centuries.

Computer vision is the science of automatically extracting high-level information from images, and has long been striving to be as successful as us humans are without even trying. Historically, attempts at computer vision algorithms have been "hard coded"—handwritten as a solution to a specific task. In recent years, however, focus has shifted towards designing algorithms that are told what to do and not how to do it, learning the specifics on their own. The change in direction from theory and design based algorithms to data-driven models is a step closer to how humans actually see.

The data-driven approach has proven to be superior to writing algorithms entirely by hand, but is still far from perfect; nearly all modern



(a) Original picture.
(b) Computer display of picture (reflected by mistake).
(c) Differentiated picture.
(d) Feature points selected.

Figure 1.1: Edge detection in the 1960s. Source: Lawrence Roberts – "Machine perception of three-dimensional solids" [13]

visual algorithms need to be trained. This process can take a large amount of processing power and will often also require massive amounts of data.

1.1 A Brief History of Computer Vision

The scientific field of computer vision (CV) began in the 1960s. Although the growth of the field was largely an inevitability, much of the early work was influenced by Lawrence Roberts' 1963 PhD thesis on perceiving three-dimensional solids from images [13].

The early days of CV saw high levels of confidence and low success rates. Famously, Seymour Papert at MIT thought in the early 60s that a small group of students would be able to solve a large chunk of "the vision problem" in one summer [12]. This initiative towards the goal of object recognition came up short.

Almost two decades later, in 1980, following new research in neuroscience, the Japanese computer scientist Kunihiko Fukushima created a model he named the Neocognitron [4]. This model was an early version of a neural network using convolutional layers. The model was later improved in 1989 by Yann LeCun's LeNet [7], in which he applied the then new backpropagation algorithm to Fukushima's convolutional architecture. LeNet is surprisingly similar to the convolutional networks we still use today.

In modern times, computer vision has exploded into many branches and subfields. Data availability, computational resources, and open development communities are important factors in driving this rapid growth. Advanced applications of computer vision, along with hardware to support them, enable applications like autonomous vehicles, instant face recognition, and optical character recognition.

1.2 Problem Statement

Current image processing techniques benefit greatly from the way they use translation symmetries, but other potentially useful symmetries are largely ignored. For flat images, rotation symmetries seem promising to improving performance. This thesis will explore methods of ensuring rotational stability by construction, and evaluate the benefits of doing so.

Chapter 2

Theoretical Background

Before getting into methods, some terms will be defined. Note that there is a glossary on page 42. Terms with definitions in the glossary will be marked with dotted underlines.

2.1 Fundamentals

2.1.1 Convolutions

A convolution[1] is a mathematical operation on two functions, resulting in a new function that combines the properties of the input functions. For two functions $f(t)$ and $g(t)$, we denote their convolution as $[f * g](t)$. In the case of continuous functions on a domain X (which in this thesis will be $\mathbb{R}$ or $\mathbb{R}^2$), the convolution can be expressed as the integral

$$[f * g](t) = \int_X f(\tau)g(\tau - t)d\tau. \tag{2.1}$$

This integral can be interpreted in several ways. For example, it has the effect of placing the function $g(t)$ at every point τ weighted by the value at $f(\tau)$. This can be seen in Figure 2.1.

For functions f and g on a discrete domain Y, an analogous discrete

[1]There is some confusion in the term "convolution", as it is used differently in computer vision and mathematics. What we refer to as a convolution here would be called a cross-correlation in mathematics. Every proof would be similar for mathematical convolution, but we use cross-correlation because it is easier to deal with. We will refer to the operation as "convolution", keeping with computer vision convention.

convolution can be written similarly as

$$[f * g](t) = \sum_{y \in Y} f(y)g(y - t). \tag{2.2}$$

We will later see that it is useful to define convolutions on images, both continuous ((2.1) with $X = \mathbb{R}^2$) and discrete ((2.2) with $Y = \mathbb{Z}^2$). These are both called planar convolutions.

2.1.2 Group Theory

Group theory is the study of the particular algebraic structure known as a group. A group is a set with a binary operation.

Let G be a group. Then G has four important properties:

1. For any $g_1, g_2, g_3 \in G$, $(g_1g_2)g_3 = g_1(g_2g_3)$ (associativity).
2. For any $g_1, g_2 \in G$, $g_1g_2 \in G$ (closure).
3. There exists an element $e \in G$ s.t. for any $g_1 \in G$ $eg_1 = g_1$ (identity element).
4. For any $g \in G$ there exists an inverse element $g^{-1} \in G$ s.t. $g^{-1}g = gg^{-1} = e \in G$ (inverse).

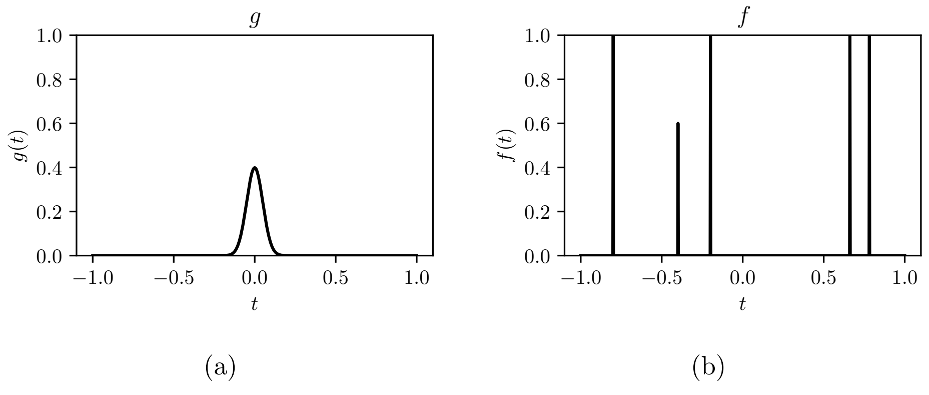
Group Actions

We will use groups in the context of group actions. Group actions are an extension of groups that additionally allow for the group elements to act on a set.

Let G be a group and X be a set. We say that $\cdot$ is a group action of G on X if $\cdot$ is a map $\cdot : G \times X \to X$ satisfying the conditions

1. $e \cdot x = x$ for all $x \in X$,
2. $(g_1g_2) \cdot x = g_1(g_2 \cdot x)$ for all $x \in X$ and all $g_1, g_2 \in G$.

If these properties are satisfied, we say that X is a G-set.



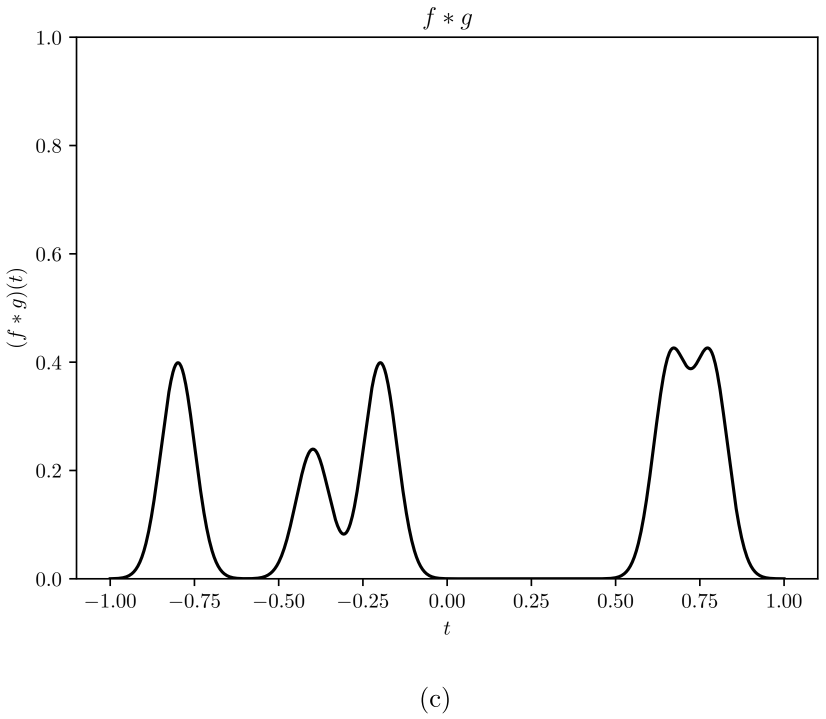


Figure 2.1: The sum of several delta functions f is convolved with a Gaussian function g to give $(f * g)$, which has several Gaussians combined in the location of the delta functions of f. Notice that convolution with g has the effect of "smearing out" the function f.

Important to this thesis are the group actions on 2D images defined by

1. Translations by vector t, $\mathcal{T} = \{\mathcal{T}_t \mid t \in \mathbb{R}^n\}$, with $\mathcal{T}_{t_1}\mathcal{T}_{t_2} = \mathcal{T}_{t_1+t_2}$, and $\mathcal{T}_t \cdot x = x$ translated by t.
2. Rotations by angle θ, $\mathcal{R} = \{\mathcal{R}_\theta \mid \theta \in [0, 2\pi)\}$, with $\mathcal{R}_{\theta_1} \cdot \mathcal{R}_{\theta_2} = \mathcal{R}_{\theta_1+\theta_2}$ and $\mathcal{R}_\theta \cdot x = x$ rotated by θ.
3. Roto-translations by angle θ and vector t, $\mathcal{T}\mathcal{R} = \{\mathcal{T}_t\mathcal{R}_\theta \mid \mathcal{T}_t \in \mathcal{T}, \mathcal{R}_\theta \in \mathcal{R}\}$, with $\mathcal{T}_{t_1}\mathcal{R}_{\theta_1}\mathcal{T}_{t_2}\mathcal{R}_{\theta_2} = \mathcal{T}_{t_1+\mathcal{R}_{\theta_1}t_2}\mathcal{R}_{\theta_1+\theta_2}$ (proof in section A.1) and $\mathcal{T}_t\mathcal{R}_\theta \cdot x = x$ rotated by θ then translated by t.

Rotations and translations do not commute. That is, in general $\mathcal{R}_\theta\mathcal{T}_t \neq \mathcal{T}_t\mathcal{R}_\theta$.

2.1.3 Functions as Images

Usually, images are treated as arrays of data. For the purposes of this thesis, we will instead refer to images as functions. An image F is a map $F : X \to \mathbb{R}^n$ from position to "pixel value". The images used for calculations will be treated as if they are defined on infinite domains $\mathbb{R}^2$ and $\mathbb{Z}^2$ while manipulating them.

Translations and rotations are also defined on these image functions. For a function $f : X \to Y$

$$[\mathcal{T}_t \cdot f](x) = f(\mathcal{T}_{-t} \cdot x) = f(x - t) \tag{2.3}$$

and

$$[\mathcal{R}_\theta \cdot f](x) = f(\mathcal{R}_{-\theta} \cdot x) \tag{2.4}$$

for $X = \mathbb{R}^2$ or $X = \mathbb{Z}^2$.

For the sake of brevity, the operator "$\cdot$" will mostly be omitted. Whether the operation is a group action or a group operation can be inferred by the operand types.

2.2 Machine Learning

A machine learning algorithm is an algorithm that is able to learn from data. Tom Mitchell defines this process of machine learning in [10]: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, measured by P improves with experience E."

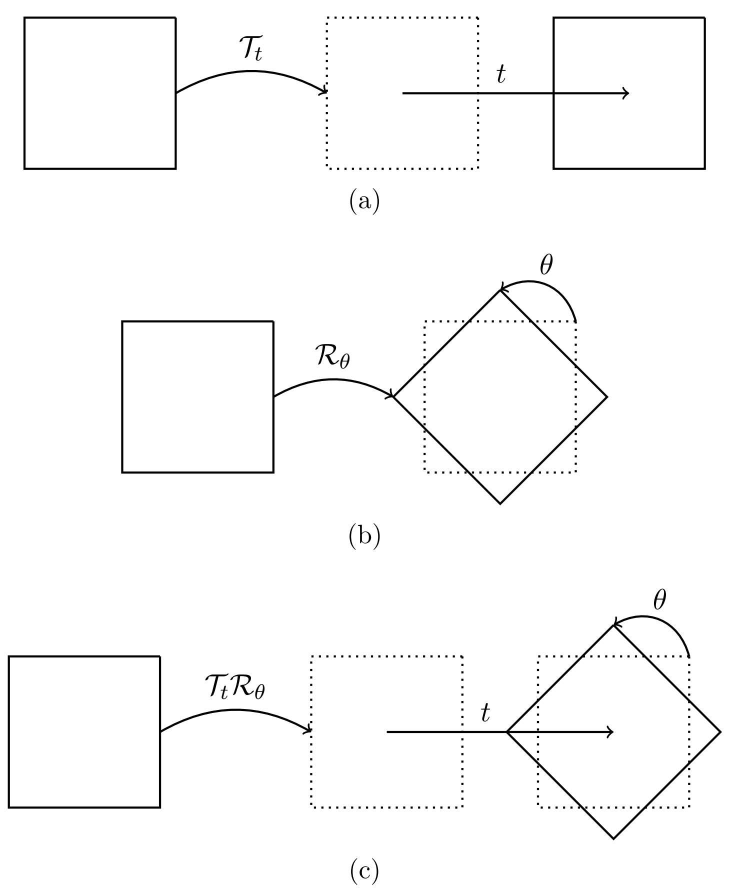


Figure 2.2: Translation, rotation, and their ordered composition. Note that in (c) the rotation is applied in global coordinates before the translation.

While there are several ways to learn from experience, we consider only supervised learning for the purpose of this thesis. In supervised learning, our algorithm is given an array of example inputs along with the correct outputs. By changing the algorithm slightly many times, it is nudged closer to being able to give the correct answers to the examples we have given. Given enough experience, it should also be able to produce the desired outputs when given new inputs.

2.2.1 Image Classification

For now, we will only consider the task T of image classification. Given a set of input images, we want our program to give each image an output label corresponding to what it sees. In this domain, we use a set of labeled training data as our experience E, and the proportion of correctly labeled images in a test set as our performance measure P. Mathematically, one might describe the classification problem using a set of possible input images X and a set of possible labels Y. Denote by $f : X \to Y$ the unknown function mapping an image to its appropriate label. The classifier can then be seen as a function $\hat{f}_\theta : X \to Y$ that adjusts parameters θ to best approximate f.

2.2.2 Neural Networks

One of the most widely used architectures for learning from examples is the feedforward neural network. The basic mechanism of such a neural network is to repeatedly transform data in different phases, often called layers. For a set of inputs X and set of outputs Y, given an input $x_0 \in X$, the network constructs an output $x_N \in Y$ using intermediary representations $\{x_i\}_{i=1}^{N-1}$ which are created using piecewise differentiable functions $\{f_i\}_{i=0}^{N-1}$.

$$x_0 \xrightarrow{f_0} x_1 \xrightarrow{f_1} \ldots \xrightarrow{f_{N-2}} x_{N-1} \xrightarrow{f_{N-1}} x_N \tag{2.5}$$

The functions f_k can either be constant or subject to change by varying parameters θ. Layers that are subject to θ are known as “trainable” because their parameters can be tweaked by optimization algorithms. The output of a neural network can be differentiated with respect to θ, and this derivative is used to update θ and make the model better.

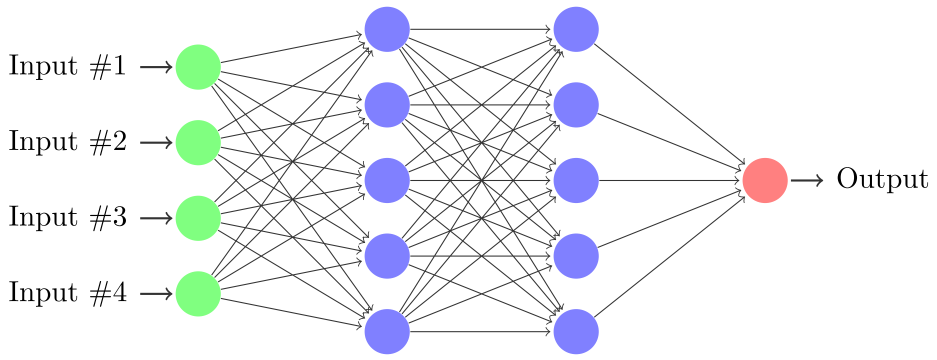


Figure 2.3: A dense neural network. Every node in each layer is connected to every node in the next.

Convolutional Neural Networks

A convolutional neural network (CNN) is a neural network that makes use of convolutional layers. Instead of connecting every input to every output at every layer, convolutional layers limit connections at every step. The two important ways in which convolutional layers limit themselves are weight sharing and locality. Subsection 2.2.4 will discuss the consequences of these limitations.

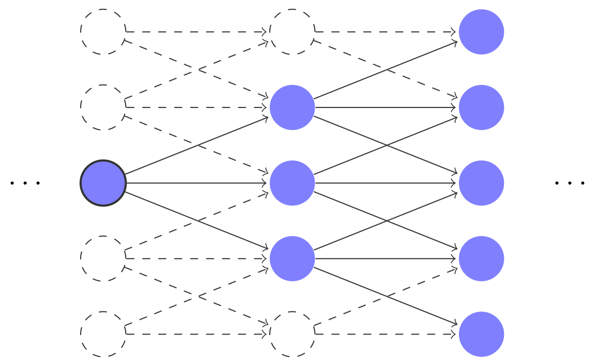


Figure 2.4: In a CNN, each layer has a restricted influence on the next. In this figure, weight sharing means that per-layer the edges from each of the nodes are the same.

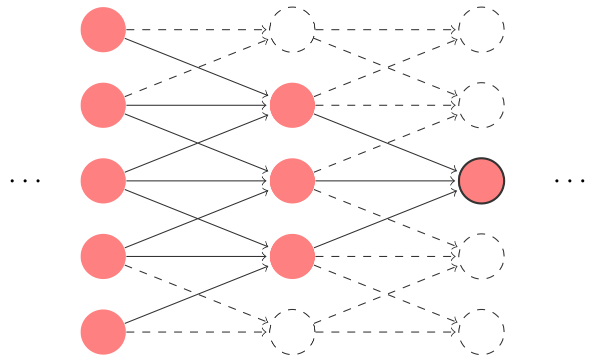


Figure 2.5: A node in a CNN will only be influenced by nodes in a restricted region of the previous layers.

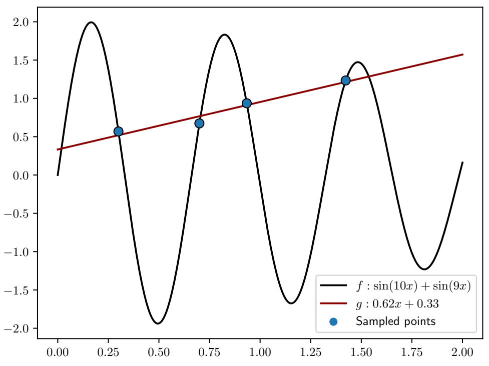


Figure 2.6: A simple yet biased function generalizes better than a needlessly complex one. Although f fits the sample points exactly, it is more likely that the ground truth is simple, as captured by g.

2.2.3 The Bias–Variance Tradeoff

Fully connected architectures such as the one in Figure 2.3 are very flexible. With the right weight settings, a dense neural network like this can approximate any function, even if it has only one hidden layer [9]. But while it may be technically possible to reach an optimal solution, that does not mean that the training procedure will lead to it. In fact, a fully connected neural network will have large set of completely different solutions that fit the input data well.

The bias–variance tradeoff describes a way of decomposing the expected error of a predictive model into three terms.

1. The irreducible error term arises from inaccuracies when sampling, or inherent randomness in the model.

2. The bias error term is related to the complexity of the model. If the model is too simple to fit the true function f, this term will be high.

3. The variance error term is also related to the complexity of the model, but instead grows as complexity increases. Using too complex a model will frequently lead to solutions that strive to reduce the error on the training set at the cost of a higher test error.

The optimal model is complex enough to be able to fit the true function, but not so complex that it's unlikely for it to converge correctly (see Figure 2.7).

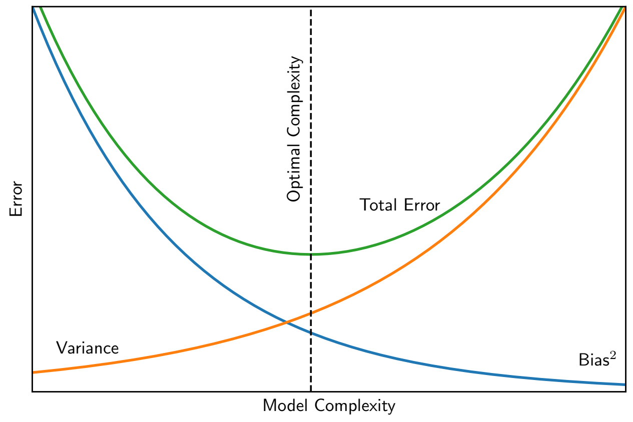


Figure 2.7: Optimizing for the total error is a compromise between between bias and variance.

These terms also have precise definitions and derivations as found in [11].

2.2.4 CNNs on Images

CNNs applied to images are a great example of the bias–variance tradeoff. By making restrictions to the model, a CNN is far better at avoiding the error term due to complexity that so often dominates dense networks.

Why do CNNs work so well for images? As seen in Figure 2.5, a node in a later layer will be influenced by a small area of the previous layer, which is influenced by a larger area of the layer before and so on. This restriction encourages the network to construct useful abstractions at every layer of the network, gradually building up to a high-level understanding of the image.

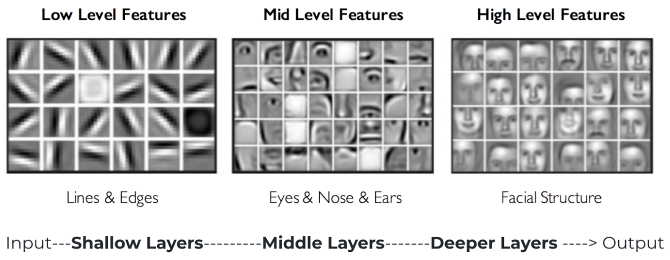


Figure 2.8: Features in a convolutional neural network. The deeper into the network, the more useful and high level the features are. Source: Alexander Amini

Chapter 3

Convolutions and Group Theory

3.1 Symmetry and Equivariance

We say an operation f on an object $\boldsymbol{O}$ has a set of symmetries S if for any transformation $\mathcal{T} \in S$ it follows that $f(\boldsymbol{O}) = f(\mathcal{T}[\boldsymbol{O}])$. This definition of symmetry is no more than a mathematical formulation of the everyday use of the word. In the same situation, one might also say that f is an invariant of $\{\mathcal{T}[\boldsymbol{O}] \mid \mathcal{T} \in S\}$, or that f is invariant to S.

To give an example, if f is a function that doesn't distinguish between corners of a polygon, the symmetries of a square $\boldsymbol{O}$ with respect to f would be the set of reflections and rotations that result in the same shape, as seen in Figure 3.1b. Then f is invariant to these symmetries.

Equivariance is a relaxed extension to the concept of symmetry. Instead of requiring the function value to be unchanged by a symmetry group, an equivariant function must allow for such an action to be reversed after the fact. In technical terms, a function f is said to be an equivariant map if both its domain and its codomain can be acted on by the same group such that the function commutes over the group action. For $f : X \to Y$ a function and G a group on X and Y we say that f is an equivariant map if

$$f(g \cdot_X x) = g \cdot_Y f(x) \quad \forall x \in X, \tag{3.1}$$

where the binary operator $\cdot_X : G \times X \to X$ is the group action of G on X and $\cdot_Y : G \times Y \to Y$ is the group action of G on Y.

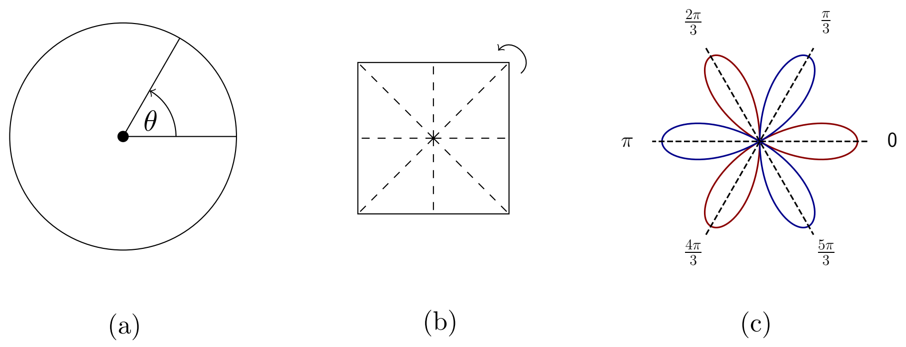


Figure 3.1: Different symmetries. 3.1a shows circular symmetry, allowing for rotations and mirroring at any angle. 3.1b shows a square, allowing 90° rotations and mirrors around the marked axes. 3.1c shows a circular harmonic function, which allows for mirrors at any of the axes, as well as rotations in multiples of $\frac{2\pi}{3}$ radians.

3.2 Group Convolutions

In subsection 2.1.1, discrete convolutions were introduced with the equation

$$[f * g](t) = \sum_{y \in Y} f(y) g(y - t). \tag{2.2 revisited}$$

The usual form of this convolution uses the group operation of translation, where the domain of the output function parametrizes the group of all translations to the filter g. In this case, we can write (2.2) as

$$[f * g](t) = \sum_{y \in \mathbb{Z}^2} f(y) g(y - t). \tag{3.2}$$

3.2.1 Equivariance of Planar Convolution

Translating the left function f in (3.2) gives

$$[[\mathcal{T}_s f] * g](t) = \sum_{y\in\mathbb{Z}^2} f(y-s)g(y-t) \tag{3.3}$$

$$= \sum_{y\in\mathbb{Z}^2} f(y)g(y+s-t) \tag{3.4}$$

$$= \sum_{y\in\mathbb{Z}^2} f(y)g(y-(t-s)) \tag{3.5}$$

$$= [\mathcal{T}_s[f*g]](t). \tag{3.6}$$

Equivalently, it can be shown that

$$[f * [\mathcal{T}_s g]](t) = [\mathcal{T}_{-s}[f*g]](t), \tag{3.7}$$

so the planar convolution is equivariant to translation in the two input functions. This result can be proven for continuous planar convolutions by replacing the sum with an integral and $\mathbb{Z}^2$ with $\mathbb{R}^2$.

It can be shown that the planar convolution is not equivariant to rotation, however:

$$[[\mathcal{R}_\theta f] * g](t) \neq [\mathcal{R}_\theta[f*g]] \tag{3.8}$$

3.2.2 Generalization

It turns out that equivariance is not restricted to planar convolutions. Let U be a group with a group action defined on $\mathbb{Z}^2$, $u, v \in U$, and $f, g : U \to \mathbb{R}^n$ functions. We define the group convolution as

$$[f*g](v) = \sum_{w\in U} f(w)g(v^{-1}w). \tag{3.9}$$

Recall that $u \in U$ can act on functions ($uf(x) = f(u^{-1}x)$), and let u act on f in (3.9). Then

$$[[uf]*g](v) = \sum_{w\in U} f(u^{-1}w)g(v^{-1}w) \tag{3.10}$$

$$= \sum_{w\in U} f(w)g(v^{-1}uw) \tag{3.11}$$

$$= \sum_{w\in U} f(w)g((u^{-1}v)^{-1}w) \tag{3.12}$$

$$= [u[f*g]](v). \tag{3.13}$$

Again, equivalently

$$[f * [ug]](v) = [u^{-1}[f * g]](v). \tag{3.14}$$

This shows that any convolution of two functions on the domain of a group is equivariant to transformations by that group. It follows directly from (3.13) and (3.14) that functions defined under the roto-translational group are equivariant to rotations and translations.

3.2.3 Harmonic Convolutions

Instead of changing the method of convolution, constraining the parameters of the planar convolution can also result in the desired rotational equivariance. Consider the continuous convolution defined in (2.1) with $X = \mathbb{R}^2$,

$$[f * g](t) = \int_{\mathbb{R}^2} f(\tau)g(\tau - t)d\tau, \tag{2.1 revisited}$$

with $g(r, \varphi) = R(r)$. Let F be a translated and rotated image, represented as $F = \mathcal{T}_t\mathcal{R}_\theta G$. Then

$$\begin{aligned}
[W_m * F](x) &= \int_{\mathbb{R}^2} W_m(y)F(y - x)\mathrm{d}y &(3.15)\\
&= \int_{\mathbb{R}^2} W_m(r, \phi)F(\mathcal{T}_{-x}y)\mathrm{d}y &(3.16)\\
&= \int_\Phi \int_R W_m(r, \phi)\left[\mathcal{T}_x F\right](r, \phi)r\mathrm{d}r\mathrm{d}\phi &(3.17)\\
&= \int_\Phi \int_R W_m(r, \phi)\left[\mathcal{T}_t\mathcal{R}_\theta\mathcal{T}_x G\right](r, \phi)r\mathrm{d}r\mathrm{d}\phi &(3.18)\\
&= \int_\Phi \int_R W_m(r, \phi)\left[\mathcal{R}_\theta\mathcal{T}_{\mathcal{R}_{-\theta}t}\mathcal{T}_x G\right](r, \phi)r\mathrm{d}r\mathrm{d}\phi &(3.19)\\
&= \int_\Phi \int_R W_m(r, \phi)\left[\mathcal{T}_{\mathcal{R}_{-\theta}t}\mathcal{T}_x G\right](r, \phi - \theta)r\mathrm{d}r\mathrm{d}\phi &(3.20)\\
&= \int_\Phi \int_R W_m(r, \phi + \theta)\left[\mathcal{T}_{\mathcal{R}_{-\theta}t}\mathcal{T}_x G\right](r, \phi)r\mathrm{d}r\mathrm{d}\phi, &(3.21)
\end{aligned}$$

where (A.1) from Appendix A is used to get from (3.18) to (3.19). Using $W_m = R(r)e^{im\phi}$ and the continuous version of (3.7) gives

$$[W_m * F](x) = \int_{\Phi}\int_{R} R(r)e^{im(\phi+\theta)}\left[\mathcal{T}_{\mathcal{R}_{-\theta}t}\mathcal{T}_x G\right](r,\phi) r \mathrm{d}r \mathrm{d}\phi, \tag{3.22}$$

$$= e^{im\theta}\int_{\Phi}\int_{R} R(r)e^{im\phi}\left[\mathcal{T}_{\mathcal{R}_{-\theta}t}\mathcal{T}_x G\right](r,\phi) r \mathrm{d}r \mathrm{d}\phi, \tag{3.23}$$

$$= e^{im\theta}[W_m * [\mathcal{T}_{\mathcal{R}_{-\theta}t}G]] \tag{3.24}$$

$$= e^{im\theta}\mathcal{T}_{\mathcal{R}_{-\theta}(-t)}[W_m * G]. \tag{3.25}$$

Thus, harmonic convolutions in continuous space are equivariant under the group of roto-translations.

Chapter 4

Equivariant Methods

4.1 Invariance and Equivariance

Recall from section 3.1 that invariance of a function to translation means that its output (in the codomain) is unchanged after translating an input (in the domain). Elements of the domain are also mapped to the same elements of the codomain after having been acted on by a symmetry group. Given a function $f : X \to Y$ and a symmetry group S,

$$f(x) = y \iff f(s \cdot x) = y \quad \forall x \in X, y \in Y, s \in S. \tag{4.1}$$

This property is very useful in image classification. We do not want to have to retrain a cat-finding neural network for every possible position a cat can be in. A cat has been moved 20 pixels to the right is still a cat.

If an image classification method should be invariant to translation, why are convolutions not invariant, but rather, equivariant? The problem with using invariant methods internally is precisely that they are invariant; they lose all information about the transformations they are invariant to. As described in subsection 2.2.4, learning from images usually involves gradually expanding the size and complexity of features as each layer acts on an image or an earlier feature map. These new features depend greatly on positions of pixels or features in the previous layers. So, while the classification algorithm should be invariant to translation, each individual step cannot be.

Instead, using an equivariant function in each layer still allows for variation and information to pass through, but guarantees that a transformation in the input image can be reversed by applying a predictable transformation on its output. If the next layer uses a function that is equivariant to the predictable transformation, and so on, the functions can be chained to make

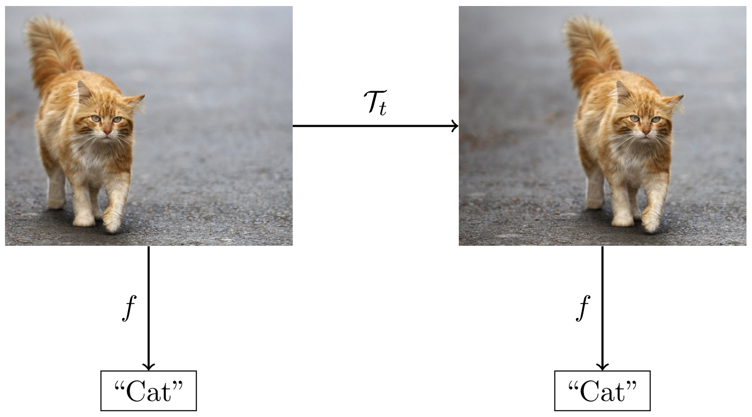


Figure 4.1: After translating the cat to the left with $\mathcal{T}_t$, the classifier should still know that the image is of a cat.

a chunk of the network equivariant to the transformation. For translation equivariant functions f, h, their composition is also translation equivariant:

$$f(h(x-t)) = f(\mathcal{T}_t h(x)) = \mathcal{T}_t f(h(x)) \tag{4.2}$$

Notice now, that if we applied an invariant function to the composition of f and h, the complete composite function would be translation invariant. The final layer in a neural network is often but not necessarily invariant to relevant transforms.

4.2 The Brute Force Approach

By far, the most common way of coercing invariant results from a neural network is brute force. This means that we input the same training images several times but with different transformations applied. Making changes to inputs during training in this way is called data augmentation, and is also used to prepare for other factors such as changes in brightness, noise, and skew.

There are some strong advantages to using data augmentation, most important of which is the ease of use. If a neural architecture performs well

Figure 4.2: In computer vision, the Picasso effect describes what happens when functions on images are spatially invariant. An invariant face detector would not care where two eyes, a nose, and a mouth are located as long as they are present.

on images without data augmentation, it is also likely that it will generalize well using brute force. Additionally, if data quantity is low in the original dataset, brute forcing can improve the performance of your network.

On the other hand, as the name may imply, brute force is far from efficient when training. Running through the same images, but rotated and transformed in many different ways, will always increase the training time by a large factor. In the case where the dataset already is sizeable, artificially increasing the size of it can potentially harm performance due to "forgetfulness" unless the augmented dataset is properly shuffled (this is expensive in terms of memory use).

4.3 Group Convolutional Neural Networks

An alternative way of achieving equivariance or invariance is to enforce it by design. As discussed in section 3.2, the group convolution is a generalization of the standard planar convolution to any symmetry group that can act on functions. The approach is fairly simple: in order to be equivariant to a new symmetry group, add a parameter to the domain of the convolution that can be indexed by elements of this group. The result is a function in which a transformation from the symmetry group has an equivalent effect if applied

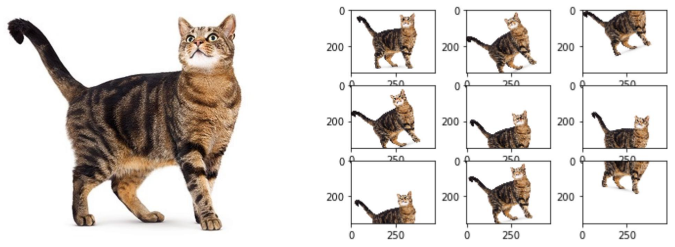


Figure 4.3: An image of a cat is rotated and obscured before it is inputted to a classifier as training data. Source: Suki Lau

to the index of the resulting function or to the image/filter. Internally, the group convolution is denoted as

$$[f * g](v) = \sum_{w \in U} f(w)g(vw). \tag{(3.9) revisited}$$

In the first layer, however, the functions to be convolved are images that cannot be indexed by elements of the group. Instead, the initial layer sums over the domain of the input image and the filter is transformed by the group.

$$[f * g](v) = \sum_{y \in \mathbb{Z}^2} f(y)[vg](y) \tag{4.3}$$

The result is an output function with domain U.

For proof of the equivariance property of the generalized group convolution, see section 3.2.

4.3.1 P4M

P4M is the most flexible group equivariant method proposed by Cohen and Welling in 2016 [3]. The method is simply the generalized group convolution where the symmetry group is the group P4M, the group of reflections, discrete translations, and 90° rotations.

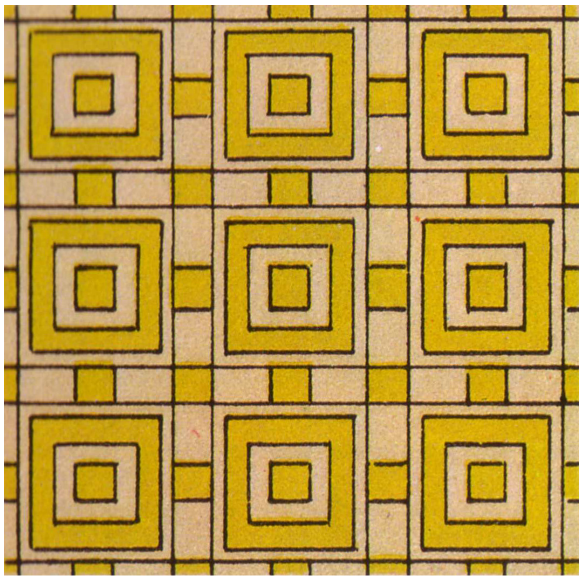


Figure 4.4: P4M is one of several “wallpaper groups”. If the pattern were tiled infinitely, it would be invariant to discrete translations, reflections, and 90° rotations. This pattern is from an ornamental painting in Nineveh, Assyria. Source: Wikimedia Commons

Chapter 5

Harmonic Convolutional Neural Networks

We have seen that discrete group convolutions can be equivariant to discrete transformations as part of their design. Although equivariance to reflections and 90° rotations is useful, we will see that equivariance to any angle of rotation is preferred. It is possible to use finer grained structures that are similar P4M along with resampling techniques for more rotational freedom, but implementations are, for the most part, impractical. In this chapter, we will explore an alternative architecture known as Harmonic Networks, first proposed in [15].

5.1 Overview

A harmonic network is a complex valued convolutional neural network where the filters are constructed so that they ensure local rotational equivariance. Each filter is constructed with a rotation order $\Delta m \in \mathbb{Z}$ in mind. Importantly, given an input F and a filter $W_{\Delta m}$ of rotation order Δm, the complex convolution yields

$$[W_{\Delta m} * (\mathcal{R}_\theta F)] = e^{i\Delta m\theta}[W_{\Delta m} * F]. \qquad \text{(Follows from (3.25))}$$

Since the convolution operation is either an integral or a sum, it is also linear, and repeated applications each deposit an exponential factor. Using (A.11) from Appendix A gives

$$\{\text{Filters of order } \Delta m_1, \Delta m_2, \ldots, \Delta m_k\} \text{ convolved with } \mathcal{R}_\theta F \tag{5.1}$$

$$= e^{i\left(\sum_{i=1}^{k} \Delta m\right)\theta}\{\text{Filters of order } \Delta m_1, \Delta m_2, \ldots, \Delta m_k\} \text{ convolved with } F. \tag{5.2}$$

Therefore, if $\sum_{i=1}^{k} \Delta m = 0$, the composition of convolutions is invariant to rotation. This is the basis of the harmonic network.

5.2 Harmonic Convolution

5.2.1 Complex Convolution

While the convolution is a fairly general operation, the fast and widely available implementations of it do not support complex numbers. Instead, 2D convolution of complex variables is rewritten in terms of real valued convolutions as

$$\begin{aligned}[W * F]_{\mathbb{C}} = {} & \big[\mathrm{Re}(W) * \mathrm{Re}(F)\big]_{\mathbb{R}} - \big[\mathrm{Im}(W) * \mathrm{Im}(F)\big]_{\mathbb{R}} \\ & + i\Big(\big[(\mathrm{Re}(W) * \mathrm{Im}(F)\big]_{\mathbb{R}} + \big[\mathrm{Im}(W) * \mathrm{Re}(F))_{\mathbb{R}}\big]\Big).\end{aligned} \tag{5.3}$$

This is implemented in Algorithm 1.

Algorithm 1: ComplexConvolution

```
Data: Complex valued filter $W$, complex valued feature map $F$
Result: The convolution of $W$ and $F$
real ← conv2d($W$.real, $F$.real) − conv2d($W$.imag, $R$.imag)
imag ← conv2d($W$.real, $F$.imag) + conv2d($W$.imag, $F$.real)
return real + $i \cdot$ imag
```

5.2.2 Harmonic Convolutional Layer

A harmonic convolutional layer represents one of the vertical sections in Figure 5.1. The inputs to such a layer are streams containing several channels of complex valued images.

```
Algorithm 2: HarmonicClassifier
  Data: Dictionary of streams heading into the layer, $S$
  Result: Dictionary of streams coming out of the layer
  output_streams ← empty dictionary
  for stream_to = 0 to streams_out − 1 do
      for stream_from = 0 to streams_in − 1 do
          $\Delta m$ ← stream_to − stream_from
          $W_{\Delta m}$ ← recall or create filter with rotation order $\Delta m$
          output_streams[stream_to] ← ComplexConvolution($W_{\Delta m}, S[\Delta m]$)
      end
  end
  return output_streams
```

5.3 Architecture

The classifier is laid out much like a traditional CNN [8], but with all the operations restricted to ones that preserve the rotational equivariance property of the inputs at each layer.

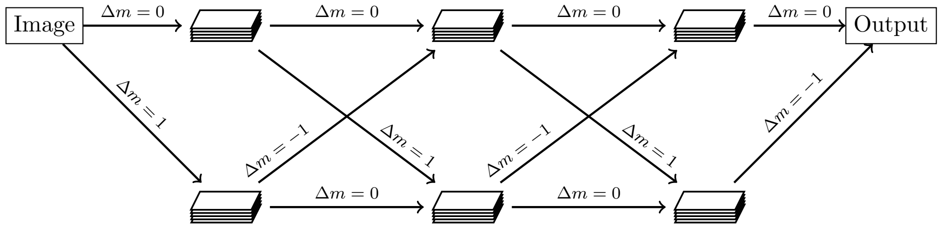


Figure 5.1: Channels can be grouped into internally equivariant equivalence classes, called "streams" by Worrall et al. [15]. The edges in the graph represent convolutions with a harmonic filter of the annotated order. Equivariance is maintained within streams, and the topmost stream has the property of rotational invariance. Note that $\sum_i \Delta m_i = 0$ along every path from the image to the output, so by (5.2) the method is rotationally equivariant.

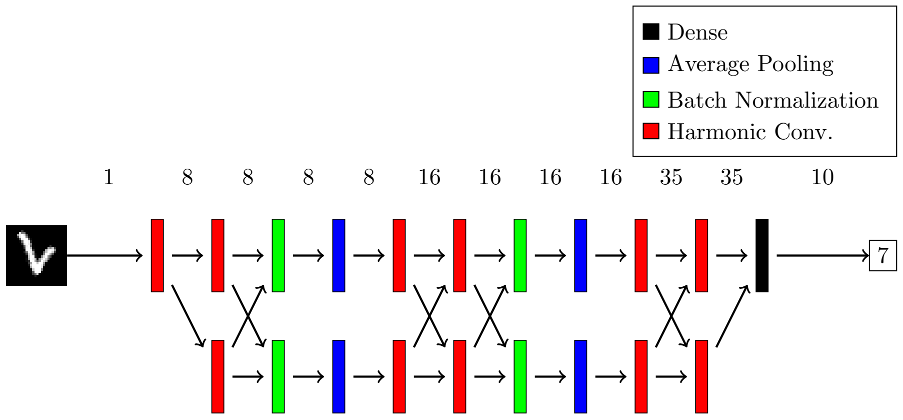


Figure 5.2: The architecture of the model used for classification of rotated handwritten digits. Each rectangle represents a layer. The numbers above the arrows signify number of channels in the representation.

Algorithm 3: HarmonicClassifier

```
Data: List of images to classify $I$, list of layers $L$ in the current
      network
Result: List of output classes, corresponding to the input set.
$O \leftarrow$ empty list
foreach image $\in I$ do
    $x \leftarrow$ image
    foreach layer $\in L$ do
        $x \leftarrow$ layer$(x)$
    end
    append $x$ to $O$
end
return $O$
```

5.4 Constructing the Filters

For an ordinary convolutional layer with n channels going in and m channels going out, one typically needs $n \cdot m$ trainable filters. The additional requirement of respecting the rotational order m of the separate streams means that each filter must have a rotational order Δm. A set of $n \cdot m$ filters is needed for every Δm used in the layer, resulting in $n \cdot m \cdot N_{\Delta m}$ harmonic filters.

5.4.1 Constraints

Recall from subsection 3.2.3 that the filters used for harmonic convolution are of the form

$$W_{\Delta m}(r, \phi) = R(r)e^{i(\Delta m \cdot \phi + \beta)},$$

where $R(r)$ and β are trainable parameters. β is a simple scalar in the interval $[0, 2\pi)$, but for $R(r)$, we decompose the function into its truncated complex Fourier series. The truncation acts much like a low-pass filter, so noisy filters are not possible.

$$R(r) = \sum_{n=-N}^{N} c_n e^{inr}, \quad c_{-n} = \overline{c_n}, \tag{5.4}$$

where the restriction on c_n ensures that $R(r)$ is a real-valued function.

The coefficients $\{c_n\}_{n=-N}^{N}$ are complex-valued, but must come from a finite number of real-valued trainable weights. Using the schema in (5.5), we construct coefficients c_n from a weight vector of odd length.

$$\begin{aligned} c_0 = w_0 \qquad c_1 &= w_1 + iw_2 \quad \dots \quad c_N \ = w_{2N-1} + iw_{2N} \\ c_{-1} &= w_1 - iw_2 \quad \dots \quad c_{-N} = w_{2N-1} - iw_{2N} \end{aligned} \tag{5.5}$$

By construction, the imaginary part of $R(r)$ is 0, so we get the real expression

$$\begin{aligned} R(r) &= \sum_{n=-N}^{N} c_n e^{inr} = \sum_{n=-N}^{N} (a_n + b_n i)(i \sin(nr) + \cos(nr)) \\ &= \sum_{n=-N}^{N} (a_n \cos(nr) - b_n \sin(nr)) \\ &= a_0 + \sum_{n=0}^{N} ((a_n + a_{-n}) \cos(nr) + (b_{-n} - b_n) \sin(nr)) \end{aligned} \tag{5.6}$$

where

$$a_n = w_{|n|}, \quad b_n = \begin{cases} w_{|n|+1}, & n > 0 \\ -w_{|n|+1}, & n < 0 \\ 0, & n = 0 \end{cases}, \tag{5.7}$$

so

$$R(r) = w_0 + \sum_{n=1}^{N} (2w_n \cos(nr) - 2w_n \sin(nr)). \tag{5.8}$$

Finally, the radial part $R(r)$ and the angular part $\Phi(\phi)$ are combined to make the complete filters

$$\begin{aligned} W_{\Delta m}(r,\phi) &= R(r)\Phi(\phi) = R(r)e^{i(\Delta m\phi+\beta)} \\ &= R(r)(i\sin(\Delta m\phi+\beta)+\cos(\Delta m\phi+\beta)). \end{aligned} \tag{5.9}$$

Algorithm 4: CreateFilter

```
**Data:** Filter weights $\{w_n\}_{n=0}^{2N+1}$, offset $\beta$, rotation order $\Delta m$
**Result:** Filter for convolution
$R(r) \leftarrow w_0 + \sum_{n=1}^{N}(2w_n\cos(nr) - 2w_{n+1}\sin(nr))$
$\Phi(\phi) \leftarrow i\sin(\Delta m\phi) + \beta) + \cos(\Delta m\phi + \beta)$
**return** $R(r)\Phi(\phi)$
```

5.4.2 Discretization

Unlike the trainable filters used in normal convolution layers, harmonic filters are continuous. Since we wish to apply the filters to images which are sampled on a 2D-grid, the filters are also discretized. Pointwise nonlinearities on the image commute, so applying the convolutions after sampling both the image and the filters is equivalent to convolving a continuous filter with a continuous image and the sampling result. In the algorithms described, the filters are sampled at 5×5 points.

5.5 Other Layers

In order to improve performance, certain additional layers were used. These layers are similar to layers that are used in convolutional neural networks, but have been adapted to conserve equivariance.

5.5.1 Non-Linearities

In a neural network, a non-linearity is a non-linear function applied to the outputs of a layer. Non-linearities are a bit tricky in the case of equivariant networks. Ruining the equivariance property is a potential problem that needs to be avoided. A non-linearity $Z : \mathbb{C} \to \mathbb{C}$ must act only on the magnitude of the feature map elementwise, so

$$Z(z) = Z(re^{im\theta}) = Z'(r)e^{im\theta}. \tag{5.10}$$

for some $Z' : \mathbb{R}^+ \to \mathbb{R}$.

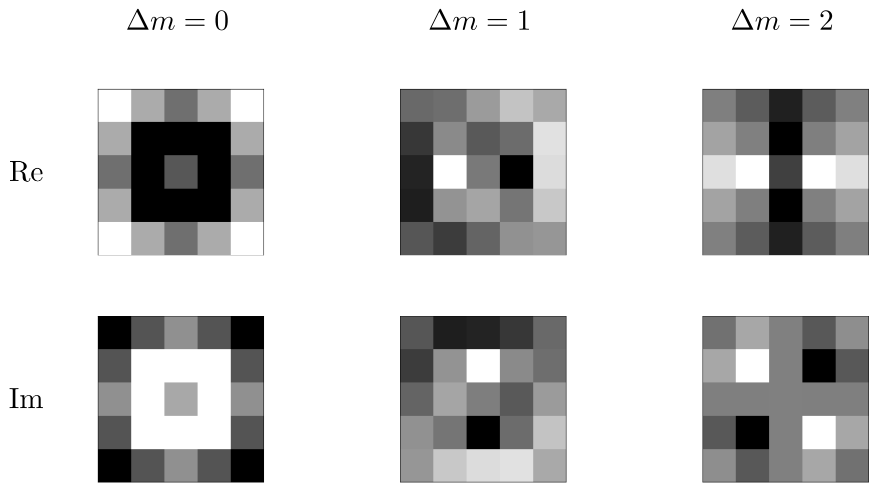


Figure 5.3: Examples of what the filters may look like after training a harmonic network.

The only non-linearity used in this implementation is the complex shifted ReLU

$$\mathrm{ReLU}_b(re^{im\theta}) = \mathrm{ReLU}(r+b)e^{im\theta}. \tag{5.11}$$

Here, $\mathrm{ReLU}(x) = \max(0, x)$.

5.5.2 Normalization

It is known that neural networks with randomly initialized weights can scale and transform inputs in ways that are unfortunate to the training process, often leading to exploding gradients and therefore numerically unstable neural networks. Batch normalization [5] is a common mitigation technique for this situation in convolutional neural networks.

In a batch normalization layer, inputs are rescaled in order to lessen the spread and the mean while keeping the variation. In essence, an empirical batch mean μ_B and empirical variance σ_B^2 is calculated in order to rescale the outputs to

$$y_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B + \epsilon}}, \tag{5.12}$$

where x_i is the input and y_i is the output in the i-th batch. ϵ is a small constant. In a harmonic network, batch normalization layers act on the magnitude of the inputs.

5.5.3 Pooling

Pooling layers are a known way of reducing the size of mappings to a more reasonable size, while keeping important information from previous layers [14]. Usually, max-pooling is used for this purpose, and it usually performs the best. However, using max-pooling would ruin the equivariance property [15]. Average-pooling is used as a substition. It can be shown that average-pooling does not affect the equivariance property.

Chapter 6

Training and Results

6.1 Evaluation

6.1.1 Loss Functions

To improve the performance of a machine learning algorithm, we must first define a measure of performance. In machine learning, the primary measure of performance is known as the loss function (also known as a negative objective function in statistics). The greater the value of the loss function, the worse the model is performing. A loss function typically has the structure

$$Loss(\hat{y}, y, \theta) = L(\hat{y}, y) + \lambda R(\theta), \tag{6.1}$$

where $\hat{y} = f(x)$ is the predicted output from the function, y is the target output, and θ are the parameters of the function. $\lambda \geq 0$ is known as the regularization coefficient, and R is a regularization function which increases with model complexity. L is a quasimetric on the set of outputs, and is the main feature of the loss function, representing the distance between a predicted and a desired output.

6.1.2 Classification Losses

For classification problems, we normally construct functions that output probability distributions $\hat{y}$, representing the confidence of a model in different classes. For training, a one-hot vector of the true class is used as y. The simplest possible choice for a loss function is the 0–1 loss:

$$L(\hat{y}, y) = I(\operatorname{argmax} \hat{y} \neq \operatorname{argmax} y), \tag{6.2}$$

where I is the indicator function. While this technically works as a loss function, an ideal one is more transparent about what aspects lead to high loss, and is differentiable without having to make poor approximations.

The categorical cross entropy is a concept from information theory that happens to also be useful as a loss function in machine learning. In short, the cross entropy between two distributions is a measure of the relative entropy between two distributions.

$$L(\hat{y}, y) = H(\hat{y}, y) = -\sum_i y_i \log(\hat{y}_i) \tag{6.3}$$

It is well known that the categorical cross entropy is the best performing loss function for classification tasks.

6.1.3 Accuracy

After the network has finished training, it is usually evaluated using a dedicated test dataset. It is crucial that this dataset has not been used in any of the previous steps, as this would give the model an unfair advantage. For classification problems where the distribution of labels is uniform, a common measure of the ultimate performance of a model is its prediction accuracy. The accuracy is the ratio of correct predictions to the number of predictions.

$$\text{Accuracy} = \frac{\#\text{ correct predictions}}{\#\text{ of predictions}} \tag{6.4}$$

6.2 Data

The MNIST handwritten digit dataset is a de facto standard in image classification, and was also used for this project. In order to test for rotation specific performance characteristics, however, a rotated version of it was used. The rotated MNIST dataset is a relatively standard test for rotation invariant models, and was standardized in [6].

6.3 Training

The neural network was implemented using tools from Keras [2] (training, stitching together layers) along with TensorFlow 2 [1] (automatic differentiation, convolution operations, vectorized math). The training process was automated by Keras, which ran optimization for categorical cross entropy (6.3). Training the harmonic network on the rotated MNIST dataset on a GPU (NVIDIA GTX 1070) takes around 40 seconds per epoch with the network sketched in Figure 5.2.

6.4 Results

Model	**Test Error (%)**
CNN*	4.88
P4CNN (Cohen, Welling)	2.28
Harmonic Network (2 streams)*	1.75
Harmonic Network (3 streams)*	1.71
Harmonic Network (Worrall et al.)	1.69

Table 6.1: Test accuracy after converging on the rotated MNIST dataset. *Results from own testing.

Table 6.1 shows the test error rate ($(1 - \text{Accuracy}) \cdot 100\%$) for the CNN as well as various equivariant networks when learning to classify the rotated MNIST dataset. The P4CNN reduces the error rate from the regular CNN by 53.2%, and the harmonic network reduces the error rate from the regular CNN by 65.4%.

In Figure 6.1, we see how the performance of the harmonic network and the CNN scale with the dataset size n when the inputs are augmented. For $n < 10^4$, the harmonic network performs better by a wide margin, but when $n > 10^5$, the traditional CNN performs equally.

Figure 6.2 shows the performance on the rotated MNIST dataset when neither of the models have ever seen a rotated image. The models are both trained on a training set of n images from the regular MNIST dataset, then tested on the rotated MNIST dataset. The harmonic network performs better than the traditional CNN, but not by much.

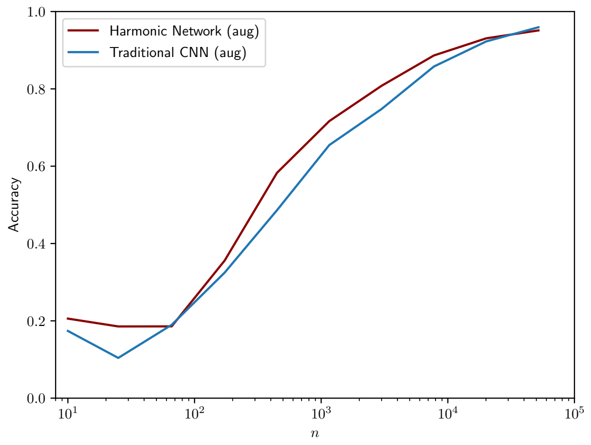


Figure 6.1: Test accuracies for the different models when varying the training set size n and training for 10 epochs. For small n, the harmonic network outperforms a CNN, even with an augmented dataset. For large n, the difference is negligible.

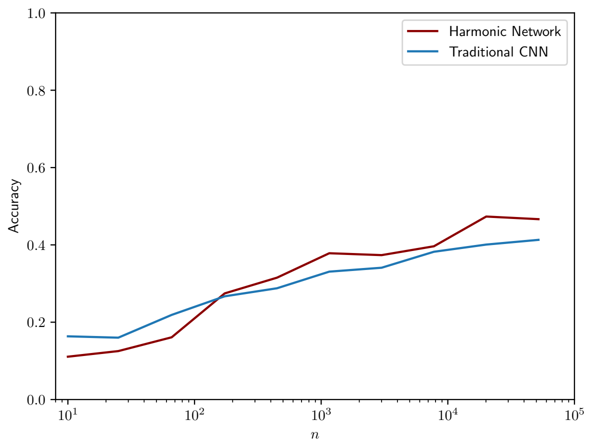


Figure 6.2: The two networks are trained on normal MNIST, but tested on the rotated version. The harmonic network always outperforms a traditional CNN.

Chapter 7

Discussion

7.1 Summary

Constraining convolutional neural networks to be equivariant to operations other than translation is not only possible — it can also be beneficial. There are ways of doing this in a discrete manner, resulting in models like the P4M CNN in subsection 4.3.1, and in a continuous manner, leading to the harmonic network in chapter 5.

7.2 Interpretations

The results show that rotationally equivariant neural networks perform better than other architectures for certain use-cases. When data availability is an issue, harmonic neural networks perform well on tasks that demand rotational understanding. When training on the regular MNIST dataset and being tested on the rotated one, the harmonic network performs better, but not by as much as initially expected. This discrepancy may be explained by the model's reliance on the final dense layer being trained to be invariant to rotation.

7.3 Implications

The harmonic network is a useful model when rotational symmetries are important. A possible use for this is in histopathology, where tissue samples like the one in Figure 7.1 are analyzed for anomalies. In this application, data is relatively scarce, and it is important that none of it goes to waste.

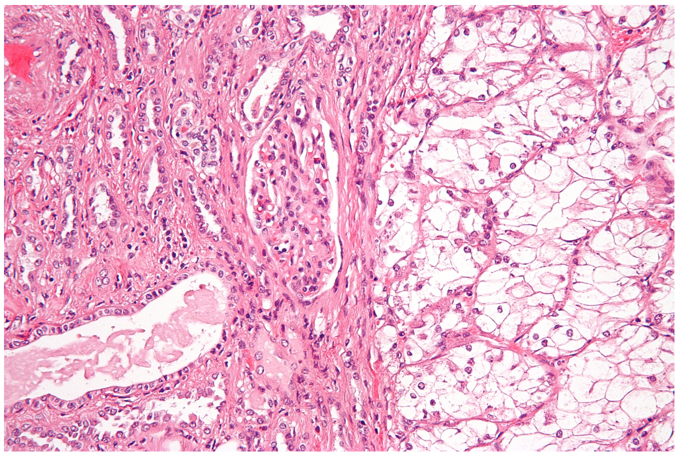


Figure 7.1: Colored tissue sample of renal cell cancer. Detecting features on top down images like this is a prime use of planar and rotational equivariance and invariance.

Images are also taken top-down, so any orientation and reflection of the input has no effect on the desired output.

There is also a possibility for equivariant blocks to be used as parts of a larger network, much like planar convolutions are used as building blocks greater models.

7.4 Recommendations

This paper only briefly considers applications of the explored methodologies, and was tested only on the rotated MNIST dataset with the task of classification. It might be worth measuring the performance of this method for other purposes on well known colored image datasets, such as segmentation with BSD500 or classification with CIFAR10.

Future work could also consider expanding and tweaking the set of harmonic layers in order to better the perfomance of harmonic networks.

In addition, the group convolutions in section 3.2 can easily be generalized to act on other dimensions of input data. For example, it might be possible to write convolutions that are equivariant to time. This may open up possibilities for performing convolutions on video data that also relate to the time component.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] François Chollet et al. Keras. `https://keras.io`, 2015.

[3] Taco S. Cohen and Max Welling. Group equivariant convolutional networks. *33rd International Conference on Machine Learning, ICML 2016*, 6:4375–4386, 2016.

[4] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, Apr 1980.

[5] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[6] Hugo Larochelle, Dumitru Erhan, Aaron Courville, James Bergstra, and Yoshua Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, page 473–480, New York, NY, USA, 2007. Association for Computing Machinery.

[7] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

[8] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[9] Moshe Leshno, Vladimir Ya. Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6(6):861 – 867, 1993.

[10] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[11] Brady Neal. On the bias-variance tradeoff: Textbooks need an update, 2019.

[12] Seymour A. Papert. The summer vision project. `https://dspace.mit.edu/handle/1721.1/6125`, 1960.

[13] Lawrence Roberts. *Machine Perception of Three-Dimensional Solids*. MIT, 01 1963.

[14] Stanford UFLDL. Pooling. `http://deeplearning.stanford.edu/tutorial/supervised/Pooling/`, 2013.

[15] D. E. Worrall, S. J. Garbin, D. Turmukhambetov, and G. J. Brostow. Harmonic Networks: Deep Translation and Rotation Equivariance. *ArXiv e-prints*, December 2016.

Glossary

equivariance A property of a function that allows for its domain and codomain to be acted on by the same symmetry group such that the function commutes with the group action. 16

invariance The property of a function that actions in a symmetry group on its domain have no effect on their mapping to the codomain. 21

non-linearity Also known as a non-linear map. A function that does not preserve the operations of addition and scalar multiplication. 31

one-hot vector Vector with zeros everywhere but one field, often written $e_i = \begin{bmatrix} 0, & 0, & \ldots, & \underset{i\text{-th element}}{1}, & \ldots, & 0, & 0 \end{bmatrix}^\intercal$. 34

quasimetric A function with all the properties of a metric except possibly symmetry. 34

stream A portion of the intermediary data that has the same rotation order. 27

Appendices

Appendix A

Mathematical Properties

A.1 Combining Translations and Rotations in $\mathbb{R}^2$

Given $\mathcal{T}_{\boldsymbol{t}} \in \mathcal{T}$ and $\mathcal{R}_\theta \in \mathcal{R}$ along with $f : \mathbb{R}^2 \to \mathbb{R}^n$ an element of a $\mathcal{T}$- and $\mathcal{R}$-set. Then

$$\begin{aligned} \mathcal{T}_t \mathcal{R}_\theta f(x) &= \mathcal{R}_\theta f(x - t) \\ &= f(\mathcal{R}_{-\theta}(x - t)) \\ &= f(\mathcal{R}_{-\theta} x - \mathcal{R}_{-\theta} t) \\ &= \mathcal{T}_{\mathcal{R}_{-\theta} t} f(\mathcal{R}_{-\theta} x) \\ &= \mathcal{R}_\theta \mathcal{T}_{\mathcal{R}_{-\theta} t} f(x), \end{aligned} \tag{A.1}$$

and the other way round

$$\begin{aligned} & \mathcal{T}_t \mathcal{R}_\theta = \mathcal{R}_\theta \mathcal{T}_{\mathcal{R}_{-\theta} t} \\ \implies & \mathcal{R}_\theta \mathcal{T}_t = \mathcal{T}_{\mathcal{R}_\theta t} \mathcal{R}_\theta. \end{aligned} \tag{A.2}$$

Using these properties we can now derive the composition of two roto-translations:

$$\begin{aligned} \mathcal{T}_{t_1} \mathcal{R}_{\theta_1} \cdot \mathcal{T}_{t_2} \mathcal{R}_{\theta_2} &= \mathcal{T}_{t_1} (\mathcal{R}_{\theta_1} \mathcal{T}_{t_2}) \mathcal{R}_{\theta_2} \\ &= \mathcal{T}_{t_1} (\mathcal{T}_{\mathcal{R}_{\theta_1} t_2} \mathcal{R}_{\theta_1}) \mathcal{R}_{\theta_2} \\ &= (\mathcal{T}_{t_1} \mathcal{T}_{\mathcal{R}_{\theta_1} t_2})(\mathcal{R}_{\theta_1} \mathcal{R}_{\theta_2}) \\ &= \mathcal{T}_{t_1 + \mathcal{R}_{\theta_1} t_2} \mathcal{R}_{\theta_1 + \theta_2} \end{aligned} \tag{A.3}$$

A.2 Composite Convolution

Let W_n and W_n be harmonic filters, and F a continuous image. Then

$$\begin{aligned}[W_n * \mathcal{T}_s[W_m * \mathcal{T}_t \mathcal{R}_\theta F]] &= [W_n * \mathcal{T}_s e^{im\theta}[W_m * \mathcal{T}_{\mathcal{R}_{-\theta} t} F]] && \text{(A.4)}\\ &= e^{im\theta}[W_n * \mathcal{T}_s[W_m * \mathcal{T}_{\mathcal{R}_{-\theta} t} F]]. && \text{(A.5)}\end{aligned}$$

Let $G(t) = [W_m * \mathcal{T}_t F]$. Then

$$[\mathcal{R}_\theta G](t) = G(\mathcal{R}_{-\theta}) = [W_m * \mathcal{T}_{\mathcal{R}_{-\theta} t} F], \tag{A.6}$$

so

$$\begin{aligned}[W_n * \mathcal{T}_s[W_m * \mathcal{T}_t \mathcal{R}_\theta F]] &= e^{im\theta}[W_n * \mathcal{T}_s[W_m * \mathcal{T}_{\mathcal{R}_{-\theta} t} F]] && \text{(A.7)}\\ &= e^{im\theta}[W_n * \mathcal{T}_s \mathcal{R}_\theta G] && \text{(A.8)}\\ &= e^{im\theta} e^{in\theta}[W_n * \mathcal{T}_{\mathcal{R}_{-\theta} s} G] && \text{(A.9)}\\ &= e^{i(m+n)\theta}[W_n * \mathcal{T}_{\mathcal{R}_{-\theta} s} G]. && \text{(A.10)}\end{aligned}$$

In particular, for $s = 0$, we get

$$[W_n * [W_m * \mathcal{T}_t \mathcal{R}_\theta F]] = e^{i(m+n)\theta}[W_n * [W_m * \mathcal{T}_t F]]. \tag{A.11}$$