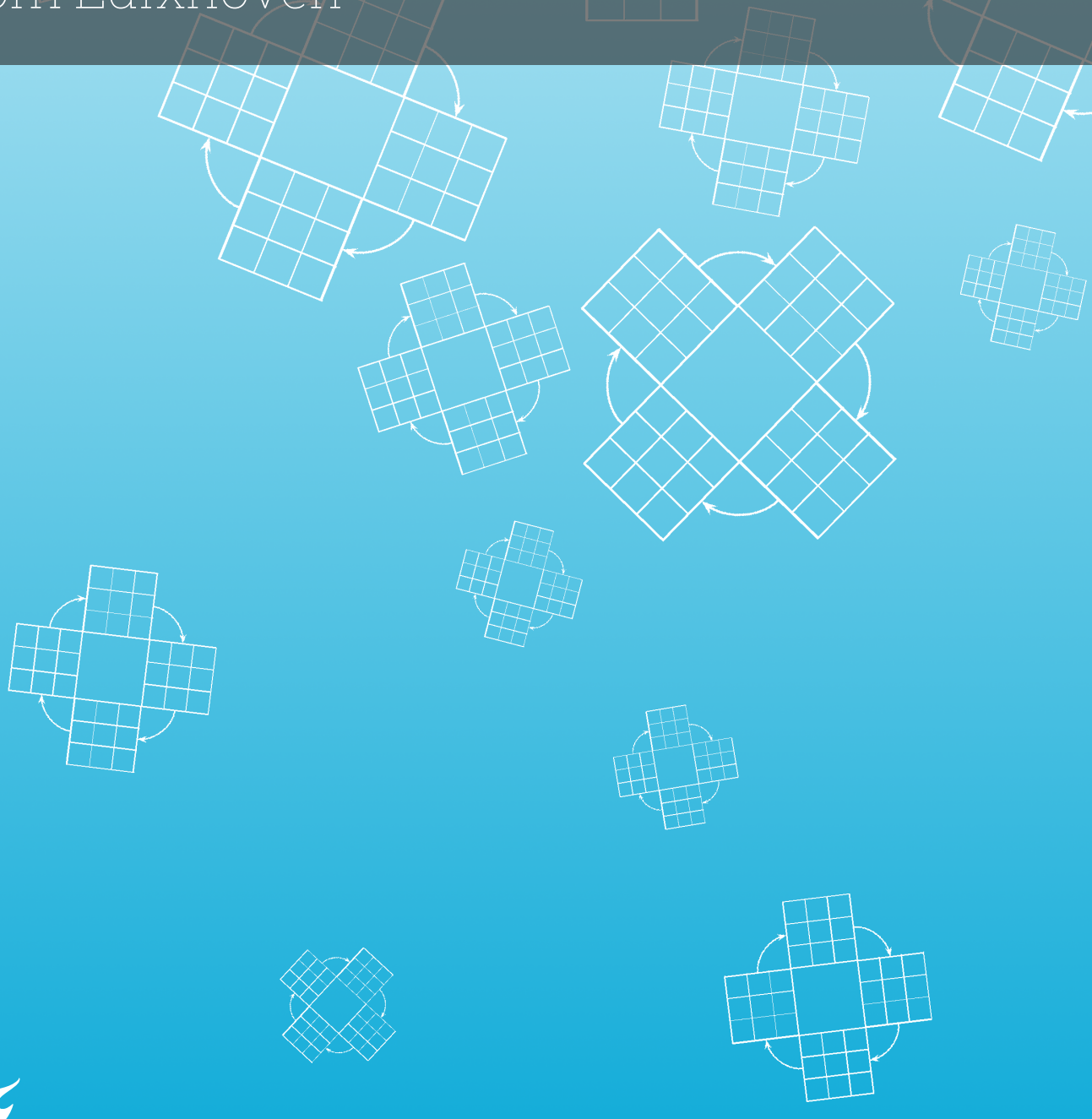
# Using and Abusing Equivariance

## Investigating Differences between Exact and Approximate Equivariance in Computer Vision

Tom Edixhoven



Delft University of Technology

TU Delft

# Using and Abusing Equivariance

## Investigating Differences between Exact and Approximate Equivariance in Computer Vision

by

Tom Edixhoven

| Student Name | Student Number |
| --- | --- |
| Tom Edixhoven | 4610075 |

| | | |
| --- | --- | --- |
| Thesis Committee: | Dr. J.C. van Gemert, | TU Delft, supervisor |
| | Dr. W.P. Brinkman, | TU Delft |
| Daily Supervisors: | Dr. J.C. van Gemert, | |
| | A. Lengyel | |
| Project Duration: | September, 2021 - January, 2023 | |

An electronic version of this thesis is available at `https://repository.tudelft.nl`

TU Delft

# Preface

This report contains the findings of my research on equivariance in computer vision to obtain the degree of Master of Science at the Delft University of Technology. The work presented in this report is especially interesting for people working with Group Equivariant Convolutions, as our findings provide a deeper understanding of how to use said convolutions and the implications of architectural choices when creating Group Equivariant Convolutional Neural Networks.

Before all else, I would like to thank my supervisors, Dr. Jan van Gemert and Attila Lengyel, as I have learned a great deal from them. Their enthusiasm and shared curiosity have helped me tremendously. Furthermore, I would like to thank them for their understanding and patience when I was unable to work for a few months. A special note of gratitude goes to Attila for choosing to remain my supervisor while being at the other side of the world. Next, I would like to thank Dr. Willem-Paul Brinkman for agreeing to be in my thesis committee. In addition, I sincerely appreciate the insights I have gained from my time with the members of the TU Delft Computer Vision Lab.

Finally, I am extremely grateful to my family and friends, who have provided me with a comfortable environment in which I could perform my research. A special note of appreciation goes out to my parents and my dear friend, Pierrick Spekreijse, whose support and willingness to hear me ramble have helped immensely.

*Tom Edixhoven*
*Delft, January 2023*

# Contents

# 1

# Introduction

In recent years computer vision models have taken the world by storm. From self-driving cars and face recognition to medical imaging, these models are everywhere and a world without them is becoming more and more unimaginable. The current dominant strategy for achieving state of the art results in many computer vision domains is using *Convolutional Neural Networks*, further referred to as CNNs. CNNs consist of several consecutive layers, each applying multiple operations on the input data. These layers typically include subsampling components, which are used to summarise information within the network and allow for drastically more efficient networks. However, the main component that makes CNNs so efficient is their use of symmetries that are present in our physical world. The backbone of CNNs is based on the *convolution* operator, which is an operator that is *equivariant* to the translation symmetry. This means that if an object is shifted in the input of a convolution, the output of the convolution is shifted equally. Translation equivariance brings great benefits for model efficiency and consistency, since it allows objects to appear at any location in the input image, as the location no longer plays a role. Thus, there is no need anymore to collect training data for objects at all possible locations in the input.

Translations are far from the only symmetry present in our physical world or in data. When collecting data, recording conditions such as camera position or camera rotation often introduce other symmetries into the data. Though these symmetries are present, it is generally undesirable for them to influence the output of a computer vision model. Take for example the field of histopathology, which consists of examining organic tissue under a microscope. Tissue is sampled and placed onto glass slides, making the rotation of the examined tissue innately arbitrary. A medical imaging model used for histopathology that changes its output when the input is rotated is therefore a cause for concern. Ideally, the model output should be independent of the rotation of the input. This is called *invariance* to rotation.

Traditional computer vision models using convolutions are not equivariant or invariant to symmetries other than translations. The seminal work of Cohen et al. [1] introduced a new type of group equivariant model, which is able to guarantee equivariance and invariance to other symmetries, such as the rotation symmetry. A group equivariant model that is invariant to rotations would therefore be ideal for fields such as histopathology. Yet, nearly all modern computer vision models contain subsampling components, which allow the group equivariant models to break their guarantee of equivariance and invariance.

This brings us to the main research questions of this thesis: How does subsampling affect the behaviour of roto-translation equivariant models used for computer vision? And, how can undesirable side-effects be mitigated? The work presented in this Chapter 2 shows that the broken guarantee of equivariance should not be ignored, as it affects the behaviour of many models containing subsampling components. Furthermore, it introduces a relatively simple solution to prevent models from abusing subsampling to break their equivariant constraints.

The rest of this thesis is divided into two parts: (1) a scientific article, included in Chapter 2, presenting the main academic findings from during the thesis, and (2) supplementary material, presented in Chapter 3, Chapter 4 and Chapter 5. The goal of the supplementary material is to allow readers not familiar with equivariance and with only basic knowledge about computer vision to read the article from Chapter 2 and think in depth about the concepts that are introduced. The supplementary material consists of three parts. Chapter 3 introduces the concepts of equivariance and symmetry groups. Chapter 4 introduces the basics of computer vision using convolutions and explains the relevant approach used to embed symmetries into convolutions. Furthermore, it explains the difference between symmetries learned by the model and symmetries that are inherent to the model itself. Lastly, Chapter 5 offers a short overview of the datasets used in Chapter 2.

# 2

# Scientific Article

# Using and Abusing Equivariance

Tom Edixhoven
Delft University of Technology
tom@edixhoven.net

## Abstract

*In this paper we show how Group Equivariant Convolutional Neural Networks use subsampling to learn to break equivariance to their symmetries. We focus on the 2D roto-translation group and investigate the impact of broken equivariance on network performance. We show that changing the input dimension of a network by as little as a single pixel can be enough for commonly used architectures to become approximately equivariant, rather than exactly. We investigate the impact of networks not being exactly equivariant and find that approximately equivariant networks generalise significantly worse to unseen symmetries compared to their exactly equivariant counterparts. However, when the symmetries in the training data are not identical to the symmetries of the network, we find that approximately equivariant networks are able to relax their own equivariant constraints, causing them to match or outperform exactly equivariant networks on common benchmark datasets.*

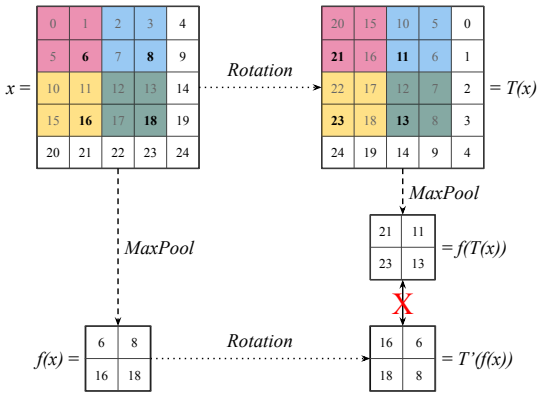



Figure 1. Example of how subsampling can break equivariance. Dotted arrows indicate a *Rotation* and the dashed arrows indicate a *MaxPool* subsampling layer with a kernel size and stride of 2. The locations where *MaxPool* applies its pooling are coloured. One can see that $f(T(x))$ and $T'(f(x))$ contain completely different numerical values, breaking equivariance.

## 1. Introduction

Nature contains a lot of symmetries [19] and networks used for computer vision have been shown to benefit greatly from prior knowledge of these symmetries. Most notably, the introduction of the convolution operator resulted in the creation of *Convolutional Neural Networks* (CNN) [31], which now form the backbone of many computer vision domains. Convolutions are *equivariant* to the translation symmetry [24], meaning that if an object in the input image is shifted, the output of the convolution is shifted equally. Due to translation equivariance, networks no longer have to explicitly learn to recognise objects at all possible locations, as the knowledge that location plays no role is embedded into the network.

Images, however, regularly contain other relevant symmetries for which CNNs are not equivariant. Take for example the field of histopathology, which entails the microscopic examination of organic tissue. In histopathology, the rotational orientation of the tissue is arbitrary [26]. A network that varies its output when the input is rotated is therefore a cause for uncertainty. More formally, the output of the network should be *invariant* to rotation, meaning that the output should not change when the input is rotated.

A major innovation in equivariance for computer vision was the introduction of *Group Equivariant Convolutions* (GEC) [4], which made it possible for CNNs to guarantee equivariance or invariance to a finite group of discrete transformations, also referred to as a *symmetry group*. Using GECs instead of standard convolutions to create a network yields a *Group Equivariant Convolutional Neural Network* (GCNN). Due to the group equivariant properties of GECs, GCNNs can be used to guarantee that the network output does not change when the input is rotated.

In this paper, we explore subsampling layers in GCNNs that allow the networks to break their guarantee of equivariance. Consider Fig. 1, where we have a *MaxPool* subsampling layer. One can see that the feature map resulting from

first rotating and then subsampling contains completely different numerical values then the result of first subsampling and then rotating. Because the values are different, we can conclude that this *MaxPool* layer is not equivariant to rotations. Whether a subsampling layer breaks equivariance is dependent on the width and height of the input, also referred to as *input dimension*. Including a subsampling layer that breaks equivariance in a GCNN will break the GCNNs guarantee of equivariance. However, subsampling layers are deemed almost essential for computer vision models and are used in nearly all GCNNs and modern CNNs. Typically, no distinction is made between GCNNs that do or do not contain subsampling layers that break equivariance. In this work, we show why a distinction should be made. We refer to networks in which subsampling layers break the guarantee of equivariance as *approximately equivariant* and networks in which the guarantee is not broken are referred to as *exactly equivariant*.

We offer the following contributions. We give a formal definition of exact equivariance under subsampling and can analyse when equivariance is broken. We show that approximately equivariant networks learn to become less equivariant and as a result generalise significantly worse to unseen symmetries compared to their exact counterparts. We show that slightly changing the input dimensions is often enough to make a network exactly equivariant rather than approximately equivariant.

## 2. Related Work

### 2.1. Equivariance in Deep Learning

CNNs are able to learn to become equivariant from data [10, 11, 30]. However, this does not guarantee equivariance to the symmetries in the data and results in a redundancy in the filters of the network. For example, the network learns one filter to detect horizontal lines and a separate one to detect vertical lines, rather than a single filter to detect lines. Much work has been written about how to efficiently teach networks equivariance to relevant symmetries during training, either by separating symmetry weights from filter weights [17,25,42], using contrastive learning [3,9] or using marginal likelihood [35,37]. However, while these methods significantly increase a network's ability to become equivariant, they do not guarantee it. Each method relies on the network learning the equivariance from the training data. However, the training data seldom guarantees a full and uniformly distributed representation of the relevant symmetries. These possible biases in the training data can then propagate into biases in the network. This can be cause for concern, as biased networks make systematic errors due to faulty assumptions about the data.

If the symmetry group for which a network needs to be equivariant is known, a common solution is to encode the symmetries into the network as prior knowledge using Group Equivariant Convolutions (GECs) [4]. GECs are equivariant to finite set of discrete transformations defined in a symmetry group. Filter weights are then shared within the GECs according to the transformations. Because the GECs include all transformations from their symmetry group, GECs guarantee equivariance to the symmetries, regardless of biases in the training data.

The introduction of GECs kickstarted much follow-up work, like extending their application from 2D planes to 3D manifolds [5, 13, 39] and the generalisation from discrete to continuous transformations using Lie algebra [6] or other means [34]. In this work, we focus on using GECs to become equivariant and invariant to the 2D roto-translation group, as the group has been proven to be useful in the field of histopathology [2, 26, 38] and processing satellite data [14, 27]. The 2D roto-translation groups consists of all rotations and translations in a 2-dimensional space. However, GECs are equivariant to discrete transformations and the 2D roto-translation group is continuous. Therefore, we make our networks equivariant to the *p4*-group [4], consisting of all compositions of translations and $90^\circ$ rotations, meaning that the networks are also equivariant to rotations of $180^\circ$ and $270^\circ$.

### 2.2. Breaking Equivariance

While CNNs are generally regarded to be translation equivariant, a plethora of work has shown that this is not completely the case. Convolutions and pooling with a stride larger than 1 have been shown to break translation equivariance [16, 40, 41]. CNNs have also been shown to be able to learn absolute positions, thereby breaking the translation equivariance [20]. This is important to note, as group convolutions assume that standard convolutions are translation equivariant to prove their equivariance to other transformations. Preventing networks from breaking their roto-translation equivariance has been investigated for reconstruction learning by introducing a group equivariant subsampling layer [40]. However, they do not investigate the effects it has on classification learning, where invariance is often more desirable than equivariance. In this work, we extend the current literature by investigating the influence of subsampling on roto-translation equivariance for classification.

The general proof for equivariance in GCNNs holds when the convolution convolves over the entire input. However, networks often unknowingly break this restriction. Pooling and strided convolutions are often used to aggregate local information and increase the receptive field of a network [18]. The combination of stride, input size and kernel size in subsampling layers can result in different values from the input feature map being sampled, resulting in approximate equivariance rather than exact equivari-

ance [32]. While it might seem like a minute detail, we find that it causes GCNNs to under perform relative to other equivariant networks in related works. Examples of rotation equivariant GCNNs exhibiting unexpected behaviour can be found in [1, 28]. In this work, we show that we can guarantee equivariance for GCNNs by introducing a relatively simple restriction on the combination of input size, kernel size and stride.

### 2.3. Relaxing Equivariant Constraints

Recent work has shown the possible benefits of relaxing equivariant constraints, showing that networks can gain performance by allowing them to learn to become less equivariant [33, 36]. This is relevant for our work, since being approximately equivariant seems to allow networks to relax their own equivariant constraints. Similarly to the aforementioned works, our solution also enables the user to make a conscious decision to relax the equivariant constraints on a network.

## 3. Methodology

### 3.1. Measuring Equivariance

We know that a network $f$ is equivariant to transformation $T$, when the output of $f$ on input $x$ changes predictably when $x$ is transformed by $T$. More formally, there exists a transformation $T'$ for which the following equation holds:

$$f(T(x)) = T'(f(x)). \tag{1}$$

GCNNs are equivariant to a set of transformations defined in a symmetry group $G$, where in practice, the transformations are stored in an additional group dimension in the feature maps. In the case of the *p4*-group, $T'$ consists of a rotation in the spatial dimensions and permutation of the group dimension on the feature map. This means that the feature maps $f(T(x))$ and $T'(f(x))$ can be computed separately. The equivariance error $\epsilon$ can then be defined as the difference between $f(T(x))$ and $T'(f(x))$. The outputs of $f(T(x))$ and $T'(f(x))$ are 3-dimensional feature maps, consisting of two spatial dimensions and one group dimension. To calculate the difference between the feature maps, we take the Mean Squared Error (MSE):

$$\epsilon = \frac{1}{ijk}\sqrt{\sum_i \sum_j \sum_k |f(T(x))_{ijk} - T'(f(x))_{ijk}|^2}, \tag{2}$$

where $i$ and $j$ sum over the spatial dimensions of the feature map and $k$ sums over the group dimension. A possible issue with using MSE is that it is sensitive to scaling. The same feature map but scaled will return a large error. However, since the same filters are used to calculate $f(T(x))$ and $T'(f(x))$, the scale of the feature maps should be approximately equal. Since Eq. (2) holds for any feature map with a group dimension, we can use Eq. (2) to measure $\epsilon$ throughout the entire convolutional part of the network.

### 3.2. Achieving Exact Rotation Equivariance

For a GCNN that is equivariant to rotations, the feature maps should retain the same numerical values, regardless of whether the input has been rotated. However, this restriction can be broken by layers that apply subsampling. We know that for a network $f$ to be equivariant to transformation $T$ on input $x$ Eq. (1) must hold. For $f$ we take a network consisting of a single *MaxPool* layer with a kernel size of 2 and a stride of 2. For $T$ we take a clockwise rotation of $90°$. Because sampling always starts at the top left part of the input, applying $T$ results in the input being shifted by a single pixel from the perspective of the *MaxPool* layer. This results in $T'(f(x))$ and $f(T(x))$ containing completely different numerical values, as shown in Fig. 1.

We propose a relatively simple solution that ensures the same indices from the feature map are sampled regardless of whether the input has been rotated or mirrored, making the network exactly equivariant. For comprehensibility, we prove that our solution holds for a square input $\in \mathbb{R}^{i\times i}$, and an arbitrary kernel size $k$ and stride $s$. However, the proof also holds for rectangular inputs $\in \mathbb{R}^{j\times i}$. Using these notations, we can prove that a layer is exactly equivariant to rotations and mirroring when the following equation holds:

$$(i-k) \mod s = 0. \tag{3}$$

A GCNN is exactly equivariant to rotations and mirroring if Eq. (3) holds for all layers in the network. The input dimensions of the network can be changed in order for Eq. (3) to hold for all layers in the network. Finding a correct input dimension requires a more in depth analysis of the network that is being used.

The rest of this subsection is used to prove Eq. (3) is correct. While interesting, it is not necessary to understand the proof to understand the other concepts discussed in this work.

We prove that Eq. (3) holds for a rotation of $90°$, thereby also proving that it holds for rotations of $180°$ and $270°$, as these can be composed using multiple rotations of $90°$. To prove Eq. (3) has to hold for a rotation of $90°$, we look at the set of indices corresponding to the input values used by the layer. These indices are referred to as the sampled indices. If the sampled indices remain the same under rotation, we know the network is equivariant to rotations, regardless of which values are in the input. We derive a new function called index, that returns the indices of the input values used by a convolutional or pooling layer to calculate the value located at index $(x, y)$ in the output:

$$\text{index}\left(\begin{bmatrix}x\\y\end{bmatrix}\right) = \left[\begin{bmatrix}sx\\sy\end{bmatrix}, \begin{bmatrix}sx+k-1\\sy+k-1\end{bmatrix}\right], \tag{4}$$

where $s$ is the stride used for subsampling and $k$ equals the kernel size. The output of the function is a square patch, denoted as $[\vec{u}, \vec{v}]$, where $\vec{u}$ represents the indices of the top left corner and $\vec{v}$ represents the indices of the bottom right corner. The sampled indices are then equal to all integer tuples within the patch.

We also introduce the function $R$, which takes an index $(x, y)$ as input and returns the indices rotated 90° counterclockwise:

$$R_n\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = \begin{bmatrix} y \\ n-1-x \end{bmatrix}, \tag{5}$$

where $n$ indicates the width and height of the feature map in which the index $(x, y)$ is located. We further generalise Eq. (5) to work on an input patch, rather than a single coordinate, giving us Eq. (6). Because a patch is indicated using the indices of the top left and bottom right corners, the output pairs $y_1$ with $x_2$ and $y_2$ with $x_1$, as a counterclockwise rotation on the top left corner makes it the bottom left corner and a counterclockwise rotation on the bottom right corner makes it the top right corner.

$$R_n\left(\left[\begin{bmatrix} x_1 \\ y_1 \end{bmatrix}, \begin{bmatrix} x_2 \\ y_2 \end{bmatrix}\right]\right) = \left[\begin{bmatrix} y_1 \\ n-1-x_2 \end{bmatrix}, \begin{bmatrix} y_2 \\ n-1-x_1 \end{bmatrix}\right] \tag{6}$$

Given our layer takes a feature map with a width and height of $i$ as input, we can write the width and height of the output feature map as

$$o = \lfloor \frac{i-k}{s} \rfloor + 1. \tag{7}$$

For a layer to be exactly equivariant, determining the sampled indices and then rotating should return the same result as rotating first and then determining the sampled indices, more formally denoted as

$$\text{index}\left(R_o\left(\begin{bmatrix} x \\ y \end{bmatrix}\right)\right) = R_i\left(\text{index}\left(\begin{bmatrix} x \\ y \end{bmatrix}\right)\right). \tag{8}$$

To solve the left-hand side, we substitute Eq. (5) as input into Eq. (4), yielding

$$\begin{aligned} &\text{index}\left(R_o\left(\begin{bmatrix} x \\ y \end{bmatrix}\right)\right) = \\ &\text{index}\left(\begin{bmatrix} y \\ \lfloor \frac{i-k}{s} \rfloor - x \end{bmatrix}\right) = \\ &\left[\begin{bmatrix} sy \\ s\lfloor \frac{i-k}{s} \rfloor - sx \end{bmatrix}, \begin{bmatrix} sy+k-1 \\ s\lfloor \frac{i-k}{s} \rfloor - sx + k - 1 \end{bmatrix}\right]. \end{aligned} \tag{9}$$

The same can be done for the right-hand side, by substituting Eq. (4) into Eq. (6), giving us

$$\begin{aligned} &R_i\left(\text{index}\left(\begin{bmatrix} x \\ y \end{bmatrix}\right)\right) = \\ &R_i\left(\left[\begin{bmatrix} sx \\ sy \end{bmatrix}, \begin{bmatrix} sx+k-1 \\ sy+k-1 \end{bmatrix}\right]\right) = \\ &\left[\begin{bmatrix} sy \\ i-k-sx \end{bmatrix}, \begin{bmatrix} sy+k-1 \\ i-1-sx \end{bmatrix}\right]. \end{aligned} \tag{10}$$

Filling in Eq. (9) and Eq. (10) into Eq. (8), we find two equations

$$s\lfloor \frac{i-k}{s} \rfloor - sx = i-k-sx, \tag{11}$$

$$s\lfloor \frac{i-k}{s} \rfloor - sx + k - 1 = i - 1 - sx. \tag{12}$$

Removing duplicate terms yields a single equation

$$s\lfloor \frac{i-k}{s} \rfloor = i - k. \tag{13}$$

Eq. (13) can be simplified to the one introduced in Eq. (3). The same method can be used to prove that Eq. (3) needs to hold for group convolutions that are equivariant to mirroring. The full proof can be found in Appendix A.

### 3.3. Measuring Output Invariance

To evaluate the invariance of a network output towards rotation, we evaluate the network on a set of rotations in $[0°, 360°)$. However, rotating an image to degrees other than $0°, 90°, 180°$ or $270°$ results in artefacts at the corners of the image, shown in Fig. 2 (left). To prevent these artefacts, we apply a *CircleCrop* during training and evaluation, only when an experiment evaluates on rotations other than multiples of 90°. *CircleCrop* works by setting all values whose coordinates are not inside the largest possible inscribed circle to 0, visualised in Fig. 2 (right). Other papers regularly use something similar or identical to *CircleCrop* to prevent unpredictable behaviour at rotations that are not multiples of 90° [21, 28]. Furthermore, it should be noted that using *Nearest Neighbour Interpolation* when rotating can also result in artefacts affecting network performance. Using *Bilinear Interpolation* is therefore more desirable.

## 4. Experiments

### 4.1. Breaking Equivariance

In this subsection, we show how networks can learn to break their equivariance to increase their performance. Furthermore, we show that networks do so on commonly used classification datasets.

Figure 2. **Left:** Rotated input without *CircleCrop*. **Right:** Rotated input with *CircleCrop*.

**Breaking Single Layer Equivariance.** If a GCNN is truly invariant, it should be unable to distinguish between an input $x$ and $T(x)$, where $T$ rotates its input by $90°$. In a fashion similar to how Kayhan et al. [20] showed that CNNs can break translation equivariance and learn absolute positions, we show that GCNNs can not only distinguish between $x$ and $T(x)$ in theory, but also in practice.

We construct a network consisting of:

1. A single rotation equivariant layer, with a kernel size of 3, a stride of 2, a single channel and a padding of 1.
2. An average-pooling layer, pooling over the spatial domain.
3. A max-pooling layer that pools over the group dimension. This turns the network from being equivariant to invariant.
4. A fully connected neural network to classify between both classes.

Furthermore, we define two separate inputs, $x_1 \in \mathbb{R}^{32\times32}$ and $x_2 \in \mathbb{R}^{33\times33}$. We then train two separate instances of the aforementioned network. The first is trained to distinguish between $x_1$ and $T(x_1)$, the second distinguishes between $x_2$ and $T(x_2)$.

We find that the network can perfectly distinguish between $x_1$ and $T(x_1)$, as visualised in Fig. 3, showing that the network is not invariant. However, the network is not able to distinguish between $x_2$ and $T(x_2)$, showing that the same network architecture is invariant for inputs in $\mathbb{R}^{33\times33}$, while not being invariant for inputs in $\mathbb{R}^{32\times32}$. This is in line with our findings in Sec. 3.2, as our network has a single convolutional layer with a kernel size of 3 and a stride of 2. Eq. (3) then holds for an input size of 33 and not for an input size of 32. These results can be generalised to even and uneven input sizes, shown in Tab. 1, given the layer has a stride of 2.

**Measured Equivariance.** We have shown that when the objective of a network is to break its equivariance, it will do so if possible. However, the question remains whether a network will also learn to do so when breaking equivariance is not explicitly the objective.

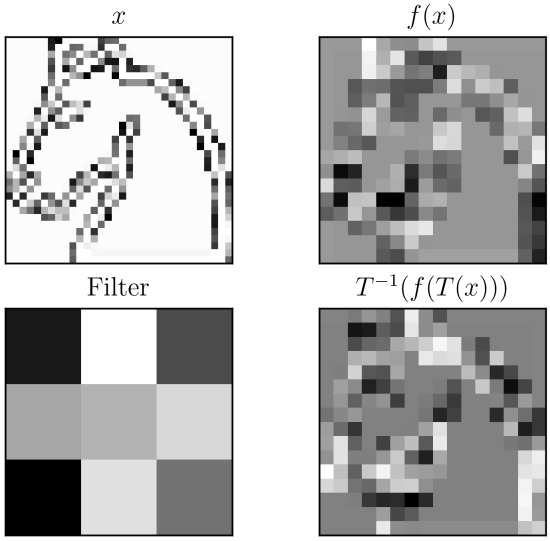



Figure 3. A subsampling Group Equivariant Convolution $f$, that is equivariant to the $90°$ rotation transformation $T$, can learn a filter that returns almost inverted values for $f(x)$ and $T^{-1}(f(T(x)))$, while these outputs should be identical in theory. Because the outputs are not equal, a network using convolution $f$ can perfectly distinguish between $x$ and $T(x)$.

| | Kernel Size | |
|---|---|---|
| Input Size | Even | Uneven |
| Even | **Exact** | Approx. |
| Uneven | Approx. | **Exact** |

Table 1. Given a stride of 2, this indicates whether the network was exactly equivariant. For a stride of 1, the network is always exactly equivariant.

To answer this question we first look at ImageNet. We create a rotation equivariant ResNet18 [15] by substituting standard convolutions with *p4*-convolutions. The network width is divided by $\sqrt{4}$ to keep the amount of parameters roughly equal to a standard ResNet18 and the input images are kept at their original $224 \times 224$ size, as this results in the network not being exactly equivariant. The network is trained to classify the standard ImageNet classes, so there is no explicit objective to distinguish between rotations. For each epoch, we then check the equivariance error at different depths in the network. A ResNet18 consists of 4 consecutive stages, therefore we measure the error after the first layer and after each stage. The error is then calculated using Eq. (2).

From the results, found in Fig. 4, we observe that at 30 epochs, when the learning rate is decreased, the equivariance error drops at most stages in the network. However,

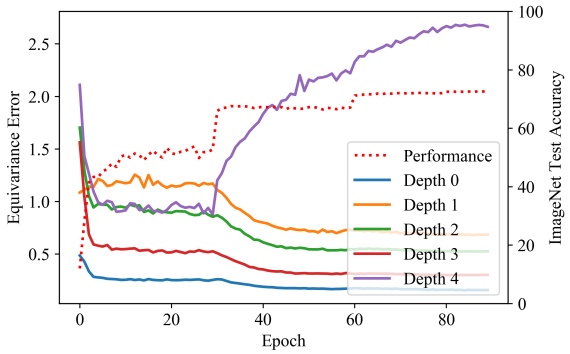



Figure 4. The measured equivariance error at different depths in a Rotation Equivariant ResNet18 network, trained on ImageNet. When the learning rate is lowered at epoch 30, the error drops throughout the entire network, except at the last layer where it increases drastically. The accuracy of the network is indicated as a dotted red line.

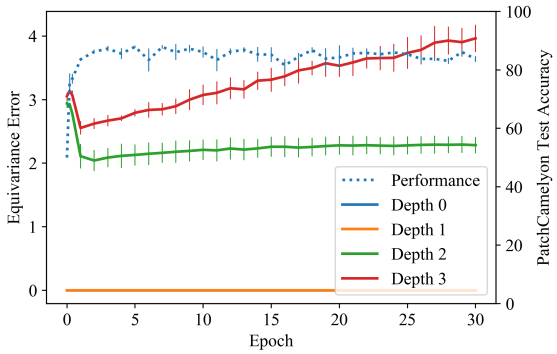



Figure 5. The measured equivariance error at different depths in a Rotation Equivariant ResNet44 network, trained on the rotation invariant dataset PatchCamelyon. One can see that even on a problem that is supposed to be rotation invariant, the network learns to become less equivariant.

the error at the last stage of the network increases drastically.

Secondly, we look at the PatchCamelyon dataset [38]. This dataset is interesting because it is pathology data, which, unlike ImageNet images, should be invariant to rotations. We use a similar setup to our previous ImageNet experiment, but we replace the ResNet18 with a ResNet44. The network width is decreased to achieve an amount of parameters that is roughly equal to the networks used by Veeling et al. [38] when evaluating on PatchCamelyon.

The results on PatchCamelyon can be found in Fig. 5. Because the first layer and the first stage have a stride of 1, the equivariance error is always 0 at the first 2 depth measurements. Interestingly, we can see that even for a rotation invariant problem, the network still learns to break its equivariance. Unlike ImageNet, the increased equivariance error does not seem to coincide with an increase is test accuracy.

We can conclude that a network that is equivariant to rotations can learn to abuse its approximate equivariance to become less equivariant to rotations. Moreover, we show that the network not only becomes less equivariant when trained on ImageNet, a dataset which contains only a limited amount of rotations, the network also learns to become less equivariant on the PatchCamelyon dataset, a dataset that should be rotation invariant by definition. A network learning differences between rotations in a rotation invariant setting is cause for concern, as the rotations are arbitrary and therefore should not contain any relevant information.

## 4.2. Impact of Exact Equivariance

To investigate the impact of exact equivariance, we look at performance on unseen rotations and seen rotations. For unseen rotations, not all rotations in the test set have been seen during training. For seen rotations, the networks are trained and tested on the same rotations.

### 4.2.1 Unseen Rotations Performance.

Due to the discrete nature of GCNNs, it is impossible to include all possible rotations in the group dimension. Therefore, it is important for the networks to generalise well to rotations that are not part of the group dimension. To compare how well approximately and exactly equivariant networks generalise to unseen rotations, we do a controlled experiment on MNIST [8], a dataset of handwritten digits between and including 0 and 9. Since MNIST only contains a limited amount of rotations, present due to slanted handwriting, we are able to control what rotations are included during training and testing by transforming the data.

For this experiment, we use the Z2CNN and P4CNN architectures introduced by Cohen et al. [4]. The Z2CNN consists of 6 layers of $3 \times 3$ convolutions, followed by a single $4 \times 4$ convolutional layer, each layer consisting of 20 channels. After each layer there are ReLU activations and batch normalisation. A dropout layer with a chance of 0.3 is added after layers 1 through 5, and a max-pooling layer with a stride of 2 after the second layer. The convolutional part is followed by a global spatial average-pooling layer, and lastly, a fully connected layer. The P4CNN architecture is created by substituting standard convolutions with *p4*-convolutions and introducing a group coset max-pooling layer before the fully connected layer. To keep the amount of parameters of Z2CNN and P4CNN roughly equal, the amount of channels in P4CNN is divided by $\sqrt{4}$.

We use an input size of $28 \times 28$ for exact equivariance, and input sizes $27 \times 27$ and $29 \times 29$ for approximate equivariance. All results are obtained by taking the average of

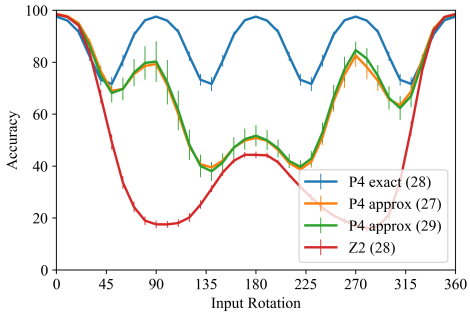



Figure 6. When training a network on a dataset without the symmetry group, one can assess how inherent the equivariance in the network is. One can see that when the training contains no data augmentations, an exactly equivariant network generalises significantly better than its approximate counterpart.

10 runs, each with a different random seed. The models are trained for 50 epochs using Adam [22] and an initial learning rate of 0.01, which is halved every 10 epochs.

The results in Fig. 6 show training on MNIST and evaluating on RotMNIST, a uniformly rotated version of MNIST. We can see that an exactly equivariant network will significantly outperform approximate counterparts on rotated samples. All the *p4*-equivariant networks still outperform the Z2CNN baseline. We also observe a much higher standard deviation in the performance of the approximately equivariant networks. The performance increase of Z2CNN at $180°$ can be attributed due to the rotational symmetries in the MNIST dataset. The $0, 1$ and $8$ classes stay roughly identical when rotated $180°$.

To further evaluate network generalisability to unseen rotations, we create two new versions of RotMNIST with biased rotation transformations. The rotation of each sample is chosen using a normal distribution. Both datasets use a mean rotation of $45°$, one has a standard deviation of $20°$ and the other of $40°$. We then train the networks on these datasets and evaluate them on all rotations.

The results in Fig. 7 show that, similarly to training on non-rotated data, the exactly equivariant network generalises noticeably better than the others. The exactly equivariant network almost becomes invariant to rotations in general, while all other networks exhibit a significant drop in performance on rotations that are not in the training data. Since the transformation distributions in the training data are often not known, it is important to properly generalise to instances of the transformation that are not in the training data.

| Model | Equivariance | MNIST | RotMNIST |
|---|---|---|---|
| Z2CNN | - (28) | $98.47 \pm 0.17$ | $91.60 \pm 1.25$ |
| P4CNN | Approx (27) | $98.52 \pm 0.26$ | $96.92 \pm 0.27$ |
| P4CNN | Exact (28) | $97.69 \pm 0.17$ | $96.89 \pm 0.21$ |
| P4CNN | Approx (29) | $98.42 \pm 0.25$ | $96.87 \pm 0.25$ |

Table 2. Network accuracy denoted as mean $\pm$ standard deviation on MNIST and RotMNIST test sets. The standard deviation is calculated using a 100 runs with different seeds. The equivariance column contains whether the network is exactly or approximately equivariant and contains the network input size in parentheses.

#### 4.2.2 Seen Rotations Performance

**MNIST and RotMNIST.** For the performance on rotations that are included in the training data, also referred to as seen rotations, we first look at accuracy on MNIST and RotMNIST using the same P4CNN and Z2CNN from Sec. 4.2.1. Each condition is ran 100 times under a different random seed in order to obtain a fair comparison.

The accuracies and their standard deviations are shown in Tab. 2. A more detailed description of the statistical analysis can be found in Appendix B. On MNIST the exactly equivariant network exhibits a performance drop between $0.65\%$ and $0.91\%$ compared to its approximately equivariant counterparts, which is confirmed to be statistically significant. A possible explanation is the inclusion of the 6 and the 9 in the dataset. We found that an exactly equivariant network had a slightly harder time distinguishing between the classes than a approximately equivariant network. This experiment can be found in Appendix C.

On RotMNIST, the exact network performs identically to the approximate networks, as the approximate networks are able to learn to become invariant from the transformations found in the training data.

**Other datasets.** We further evaluate the impact of exact and approximate equivariance on common benchmark datasets that contain rotational symmetries, as well as datasets that are not known for containing rotations. For datasets containing rotational symmetries, we investigate *Flowers102* [29], where most but not all classification categories contain rotational symmetries, and the aforementioned *PatchCamelyon* [38], which should be completely invariant to rotations. For datasets that do not contain many rotational symmetries, we use *Cifar10*, *Cifar100* [23] and *ImageNet* [7], as these datasets are commonly used to evaluate network architectures that are not equivariant to rotations.

The benchmark results can be found in Tab. 3. The exactly equivariant networks are generally matched or outperformed by their approximately equivariant counterparts,

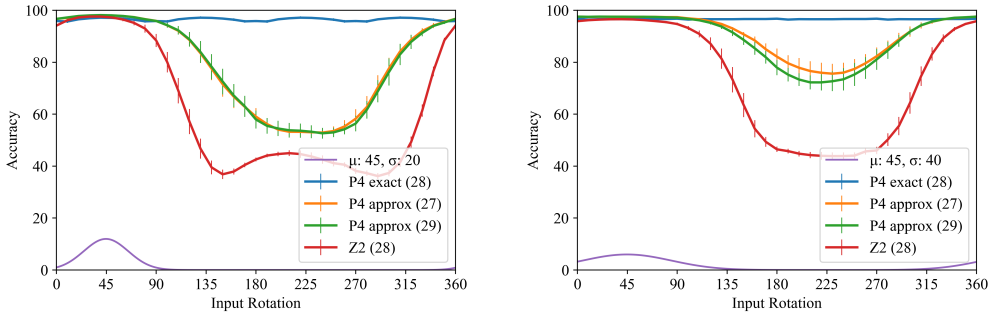



Figure 7. When training data contains a biased data augmentation, an exactly equivariant network is significantly better at generalising. For rotation, once the augmentation reaches a standard deviation of approximately $90°$, exact and approximate networks start to behave similarly. The inputs are rotated according to angles picked using a normal distribution, which are visualised in the bottom of the plots. Their mean and standard deviation are given in the legends.

even on datasets containing rotational symmetries. This seems to indicate that there lies value in relaxing the equivariant constraints of networks.

Furthermore, both *p4*-equivariant networks outperform their *z2*-equivariant counterparts, even when the dataset is not known for containing many rotational symmetries. This could indicate that the improvements from the group equivariant architecture might not be solely from equivariance, but could also originate from other traits of GCNNs. Other possible explanations are the increase in computations or the amount of gradients a GCNN uses compared to a standard CNN.

One explanation we explored further is whether rotation equivariant networks decide more on texture than on shape, which can be found in Appendix D. Our hypothesis was that a *p4*-equivariant network would be better at recognising textures, as textures often remain the same when rotated. However, our findings for this experiment were inconclusive.

## 5. Conclusion

In this work, we show that Group Equivariant Convolutions [4] can and do learn to break their equivariance towards the 2D roto-translation group in common use cases. We prove theoretically and empirically that changing the input size of the network is enough to prevent a network from breaking its equivariance. We find that exactly equivariant networks generalise significantly better to unseen rotations than their approximately equivariant counterparts, but that when the training data contains all relevant rotations there is no significant difference.

Interestingly, we also find results that suggest equivariant networks offer performance increases to datasets that do not contain the relevant transformations, suggesting that using GCNNs might offer benefits other than their equivariance. Furthermore, we find that relaxing equivariant constraints can be beneficial for network performance. However, relaxing equivariant constraints also allows networks to become biased towards the distribution of transformations in the training data.

## 6. Limitations and Future Work

**Limited to rotations.** The symmetries we investigated are limited to translations and rotations. While these symmetries are relevant, the work could benefit from being extended to include other symmetries. We have proven in the appendix that reflection equivariance can also be exact and approximate, however, we have not investigated the effects. More research could also be done in including more rotations in the group dimension, such as all rotations of $45°$, rather than rotations of $90°$.

**Influence of padding on equivariance.** During our experiments, we found that padding has a large influence on how well a network generalises to unseen rotations. We have not been able to find an explanation for this phenomenon. Therefore, it could be an interesting future work.

**Equivariance at various depths.** The results from Sec. 4.1 suggest that equivariant layers are more desirable at some depths in the network than others, since the equivariance error drops at some depths and rises at others. An interesting future work would be making a robust analysis of desirability of equivariance at different depths in a network.

**Application to rotation-invariant problems.** Another interesting extension would be to further investigate our results on PatchCamelyon, where we found that the approximately equivariant network learnt to break its equivariance in order to increase performance, even on a problem that is

| Dataset | Model | Approx. *p4*-equivariant | Exact. *p4*-equivariant | Standard CNN |
|---|---|---|---|---|
| Many Rotational Symmetries | | | | |
| Flowers102 | ResNet-44 | $86.28 \pm 1.32$ | $86.65 \pm 1.41$ | $82.18 \pm 0.53$ |
| PCam | ResNet-44 | $87.52 \pm 1.20$ | $87.4 \pm 0.71$ | $83.35 \pm 1.04$ |
| Limited Rotational Symmetries | | | | |
| ImageNet | ResNet-18 | 73.03 | 72.98 | 70.0 |
| CIFAR100 | ResNet-44 | 76.2 | 74.2 | 69.1 |
| CIFAR10 | ResNet-44 | $94.8 \pm 0.098$ | $94.4 \pm 0.25$ | $93.1 \pm 0.22$ |

Table 3. Accuracies of different models applied to common benchmarks. For datasets that are not too computationally expensive to run, the standard deviation is included in the accuracy. Datasets are categorised into containing many or limited rotational symmetries. Interestingly, even on datasets with limited rotational symmetries, *p4*-equivariant networks outperform networks that are not equivariant to rotations. Furthermore, approximately equivariant networks seem to match or outperform their exactly equivariant counterparts.

supposed to be invariant to rotation. With the rise of relaxed equivariant constraints [33, 36], an interesting question to ask would be whether we are actually achieving better performance or simply exploiting unknown biases in data or in the network.

## References


[1] Piyush Bagad, Floor Eijkelboom, Mark Fokkema, Danilo de Goede, Paul Hilders, and Miltiadis Kofinas. C-3PO: Towards rotation equivariant feature detection and description. In *3rd Visual Inductive Priors for Data-Efficient Deep Learning Workshop*, 2022. 3

[2] Erik J Bekkers, Maxime W Lafarge, Mitko Veta, Koen A J Eppenhof, Josien P W Pluim, and Remco Duits. Roto-Translation Covariant Convolutional Networks for Medical Image Analysis. In Alejandro F. Frangi, Julia A. Schnabel, Christos Davatzikos, Carlos Alberola-López, and Gabor Fichtinger, editors, *Medical Image Computing and Computer Assisted Intervention – MICCAI 2018*, pages 440–448. Springer International Publishing, 2018. 2

[3] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. ICML'20. JMLR.org, 2020. 2

[4] Taco Cohen and Max Welling. Group equivariant convolutional networks. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 2990–2999, New York, New York, USA, 20–22 Jun 2016. PMLR. 1, 2, 6, 8

[5] Taco S. Cohen, Maurice Weiler, Berkay Kicanaoglu, and Max Welling. Gauge equivariant convolutional networks and the icosahedral cnn. In *International Conference on Machine Learning*, 2019. 2

[6] Nima Dehmamy, Robin Walters, Yanchen Liu, Dashun Wang, and Rose Yu. Automatic symmetry discovery with lie algebra convolutional network. In *Neural Information Processing Systems*. arXiv, 2021. 2

[7] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. 7

[8] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012. 6

[9] Alexandre Devillers and Mathieu Lefort. Equimod: An equivariance module to improve self-supervised learning, 2022. 2

[10] Jonas Geiping, Micah Goldblum, Gowthami Somepalli, Ravid Shwartz-Ziv, Tom Goldstein, and Andrew Gordon Wilson. How much data are augmentations worth? an investigation into scaling laws, invariance, and implicit regularization, 2022. 2

[11] Jonas Geiping, Micah Goldblum, Gowthami Somepalli, Ravid Shwartz-Ziv, Tom Goldstein, and Andrew Gordon Wilson. How much data are augmentations worth? an investigation into scaling laws, invariance, and implicit regularization, 2022. 2

[12] Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A. Wichmann, and Wieland Brendel. Imagenet-trained cnns are biased towards texture; increasing shape bias improves accuracy and robustness, 2018. 13

[13] Pim De Haan, Maurice Weiler, Taco Cohen, and Max Welling. Gauge equivariant mesh {cnn}s: Anisotropic convolutions on geometric graphs. In *International Conference on Learning Representations*, 2021. 2

[14] Jiaming Han, Jian Ding, Nan Xue, and Gui-Song Xia. Redet: A rotation-equivariant detector for aerial object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2786–2795, June 2021. 2

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. 5

[16] Md Amirul Islam, Matthew Kowal, Sen Jia, Konstantinos G. Derpanis, and Neil D. B. Bruce. Global pooling, more than meets the eye: Position information is encoded channel-wise in cnns, 2021. 2

[17] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial transformer networks, 2015. 2

[18] Dong-Hwan Jang, Sanghyeok Chu, Joonhyuk Kim, and Bohyung Han. Pooling revisited: Your receptive field is suboptimal, 2022. 2

[19] Iain G. Johnston, Kamaludin Dingle, Sam F. Greenbury, Chico Q. Camargo, Jonathan P. K. Doye, Sebastian E. Ah-

nert, and Ard A. Louis. Symmetry and simplicity spontaneously emerge from the algorithmic nature of evolution. *Proceedings of the National Academy of Sciences*, 119(11):e2113883119, 2022. 1

[20] Osman Semih Kayhan and Jan C. van Gemert. On translation invariance in cnns: Convolutional layers can exploit absolute spatial location, 2020. 2, 5

[21] Jinpyo Kim, Wooekun Jung, Hyungmo Kim, and Jaejin Lee. Cycnn: A rotation invariant cnn using polar mapping and cylindrical convolution layers, 2020. 4

[22] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. 7

[23] Alex Krizhevsky. Learning multiple layers of features from tiny images. pages 32–33, 2009. 7

[24] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems*, 25, 01 2012. 1

[25] Denis Kuzminykh, Daniil Polykovskiy, and Alexander Zhebrak. Extracting invariant features from images using an equivariant autoencoder. In Jun Zhu and Ichiro Takeuchi, editors, *Proceedings of The 10th Asian Conference on Machine Learning*, volume 95 of *Proceedings of Machine Learning Research*, pages 438–453. PMLR, 14–16 Nov 2018. 2

[26] Maxime W. Lafarge, Erik J. Bekkers, Josien P. W. Pluim, Remco Duits, and Mitko Veta. Roto-translation equivariant convolutional networks: Application to histopathology image analysis, 2020. 1, 2

[27] Joshua Mitton and Roderick Murray-Smith. Rotation equivariant deforestation segmentation and driver classification, 2021. 2

[28] Hanlin Mo and Guoying Zhao. Ric-cnn: Rotation-invariant coordinate convolutional neural network, 2022. 3, 4

[29] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *Indian Conference on Computer Vision, Graphics and Image Processing*, Dec 2008. 7

[30] Chris Olah, Nick Cammarata, Chelsea Voss, Ludwig Schubert, and Gabriel Goh. Naturally occurring equivariance in neural networks. *Distill*, 5, 12 2020. 2

[31] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks, 2015. 1

[32] David W Romero, Erik J Bekkers, Jakub M Tomczak, and Mark Hoogendoorn. Attentive group equivariant convolutional networks. *arXiv preprint arXiv:2002.03830*, 2020. 3

[33] David W Romero and Suhas Lohit. Learning equivariances and partial equivariances from data. *arXiv preprint arXiv:2110.10211*, 2021. 3, 9

[34] Kai Sheng Tai, Peter Bailis, and Gregory Valiant. Equivariant transformer networks, 2019. 2

[35] Tycho F.A. van der Ouderaa and Mark van der Wilk. Learning invariant weights in neural networks. In James Cussens and Kun Zhang, editors, *Proceedings of the Thirty-Eighth Conference on Uncertainty in Artificial Intelligence*, volume 180 of *Proceedings of Machine Learning Research*, pages 1992–2001. PMLR, 01–05 Aug 2022. 2

[36] Tycho F. A. van der Ouderaa, David W. Romero, and Mark van der Wilk. Relaxing equivariance constraints with non-stationary continuous filters, 2022. 3, 9

[37] Mark van der Wilk, Matthias Bauer, ST John, and James Hensman. Learning invariances using the marginal likelihood, 2018. 2

[38] Bastiaan S. Veeling, Jasper Linmans, Jim Winkens, Taco Cohen, and Max Welling. Rotation equivariant cnns for digital pathology, 2018. 2, 6, 7

[39] Maurice Weiler, Patrick Forré, Erik Verlinde, and Max Welling. Coordinate independent convolutional networks – isometry and gauge equivariant convolutions on riemannian manifolds, 2021. 2

[40] Jin Xu, Hyunjik Kim, Tom Rainforth, and Yee Whye Teh. Group equivariant subsampling, 2021. 2

[41] Richard Zhang. Making convolutional networks shift-invariant again, 2019. 2

[42] Allan Zhou, Tom Knowles, and Chelsea Finn. Meta-learning symmetries by reparameterization, 2020. 2

## A. Proof Mirroring Equivariance

The proof for the mirroring transformation is practically identical to the proof given in Sec. 3.2. However, we replace rotation Eq. (5) with the mirroring Eq. (16) and Eq. (6) with Eq. (17). This leads to the equations shown below, eventually reaching the same conclusion as given in Sec. 3.2.

$$\text{index}\left(\begin{bmatrix}x\\y\end{bmatrix}\right) = \left[\begin{bmatrix}sx\\sy\end{bmatrix}, \begin{bmatrix}sx+k-1\\sy+k-1\end{bmatrix}\right]. \tag{14}$$

$$o = \lfloor \frac{i-k}{s} \rfloor + 1. \tag{15}$$

$$M_n\left(\begin{bmatrix}x\\y\end{bmatrix}\right) = \begin{bmatrix}n-1-x\\y\end{bmatrix}, \tag{16}$$

$$M_n\left(\left[\begin{bmatrix}x_1\\y_1\end{bmatrix}, \begin{bmatrix}x_2\\y_2\end{bmatrix}\right]\right) = \left[\begin{bmatrix}n-1-x_2\\y_1\end{bmatrix}, \begin{bmatrix}n-1-x_1\\y_2\end{bmatrix}\right] \tag{17}$$

$$\text{index}\left(M_o\left(\begin{bmatrix}x\\y\end{bmatrix}\right)\right) = M_i\left(\text{index}\left(\begin{bmatrix}x\\y\end{bmatrix}\right)\right). \tag{18}$$

$$\begin{aligned}&\text{index}\left(M_o\left(\begin{bmatrix}x\\y\end{bmatrix}\right)\right) = \\ &\text{index}\left(\begin{bmatrix}\lfloor \frac{i-k}{s} \rfloor - x\\y\end{bmatrix}\right) = \\ &\left[\begin{bmatrix}s\lfloor \frac{i-k}{s} \rfloor - sx\\sy\end{bmatrix}, \begin{bmatrix}s\lfloor \frac{i-k}{s} \rfloor - sx + k - 1\\sy+k-1\end{bmatrix}\right].\end{aligned} \tag{19}$$

$$\begin{aligned}&M_i\left(\text{index}\left(\begin{bmatrix}x\\y\end{bmatrix}\right)\right) = \\ &R_i\left(\left[\begin{bmatrix}sx\\sy\end{bmatrix}, \begin{bmatrix}sx+k-1\\sy+k-1\end{bmatrix}\right]\right) = \\ &\left[\begin{bmatrix}i-k-sx\\sy\end{bmatrix}, \begin{bmatrix}i-1-sx\\sy+k-1\end{bmatrix}\right].\end{aligned} \tag{20}$$

$$s\lfloor \frac{i-k}{s} \rfloor = i-k. \tag{21}$$

## B. Statistical Analysis of Significance

To determine the statistical significance of our results, we compare each pair of models using an independent t-Test testing the null hypothesis $H_0 : \mu_a = \mu_b$. We use a significance level $\alpha = 1.0 \times 10^{-2}$. However, since we perform 12 comparisons in total, we use Bonferroni correction and find a new significance level $\alpha = 8.33 \times 10^{-4}$. The p-values resulting the the t-Tests for MNIST can be found in Tab. 4 and in Tab. 5 for RotMNIST. The values were calculated using a 100 repeats for each condition to ensure we had a representative normal distribution for the performance. We then visually confirmed the performance distribution to be a normal distribution. Due to unequal variances between the performance of P4 and Z2 networks on RotMNIST, a Welch's t-Test was used to calculate the p-value for comparisons including the Z2 network.

| | P4 (27) | P4 (28) | P4 (29) |
|---|---|---|---|
| Z2 (28) | $1.43 \times 10^{-1}$ | $7.92 \times 10^{-82}$ | $1.09 \times 10^{-1}$ |
| P4 (27) | - | $1.19 \times 10^{-62}$ | $9.97 \times 10^{-2}$ |
| P4 (28) | - | - | $3.03 \times 10^{-57}$ |

Table 4. p-values for two sided t-Test for different networks trained on the MNIST dataset. The input dimension of the network is indicated using parentheses.

| | P4 (27) | P4 (28) | P4 (29) |
|---|---|---|---|
| Z2 (28) | $2.95 \times 10^{-99}$ | $1.53 \times 10^{-99}$ | $2.95 \times 10^{-99}$ |
| P4 (27) | - | $4.54 \times 10^{-1}$ | $1.70 \times 10^{-1}$ |
| P4 (28) | - | - | $4.44 \times 10^{-1}$ |

Table 5. p-values for two sided t-Test for different networks trained on the RotMNIST dataset. For p-values of comparisons containing the Z2 network, a Welch's t-Test is used due to unequal variances. The input dimension of the network is indicated using parentheses.

For MNIST, we find a significant difference between our exactly equivariant network and the other networks. For RotMNIST we find no significant differences between the P4 equivariant networks, but we do find that the Z2 equivariant network performs significantly worse than the others.

To assert the effect size, we look at the *95%-confidence intervals*, given in Tab. 6. We find that on MNIST, the exactly equivariant network has a performance drop between $0.65\%$ and $0.91\%$ compared to the other networks. On RotMNIST, P4 equivariant networks offer a performance increase between $4.97\%$ and $5.62\%$ compared to a standard CNN.

| Model | Equivariance | MNIST | RotMNIST |
|---|---|---|---|
| Z2CNN | - (28) | [98.44; 98.51] | [91.35; 91.85] |
| P4CNN | Approx (27) | [98.47; 98.57] | [96.86; 96.97] |
| P4CNN | Exact (28) | [97.66; 97.72] | [96.85; 96.93] |
| P4CNN | Approx (29) | [98.37; 98.47] | [96.82; 96.92] |

Table 6. Network accuracy confidence interval on MNIST and RotMNIST test sets. The standard deviation is calculated using a 100 runs with different seeds. The equivariance column contains whether the network is exactly or approximately equivariant and the input dimensions of the network in parentheses.

To make the analysis more robust, one could also choose to model the performance according to two independent

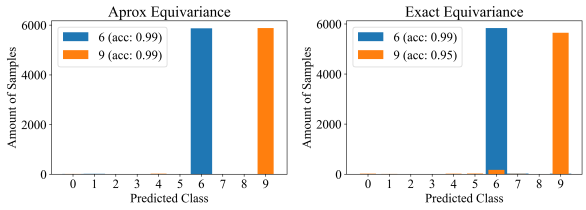



Figure 8. The predicted classes for the input classes 6 (blue) and 9 (orange). One can see that the exact GCNN makes slightly more errors when classifying a 9 than the approximate GCNN. **Left:** Approximate equivariance. **Right:** Exact equivariance.

variables: (1) type of equivariance and (2) input size. This would require the additional training of two Z2CNN networks, one with an input size of 27 and another with an input size of 29. Due to the scope of our work, we deemed our current statistical analysis to be sufficient.

## C. Distinguishing Between 6 and 9 Under Rotation

When discussing rotation equivariance and MNIST together, the problem of distinguishing 6s and 9s under rotation is often brought up. A 6 that is rotated $180°$ resembles a 9 and the same holds for a rotated 9 resembling a 6.

We found that exactly rotation equivariant networks perform slightly worse on MNIST than approximately rotation equivariant networks, showing a performance drop between $0.65\%$ and $0.91\%$. Examining the classification of the 6 and 9 classes for both networks, we find that the exact GCNN makes is slightly more eager to classify 9s as a 6 than the approximate GCNN. This could explain the performance drop. However, it still achieves an accuracies of $99\%$ for the 6 class and $95\%$ for the 9 class. The results are shown in Fig. 8.

Interestingly, we find that if we rotate the input $180°$ the accuracy of the approximate GCNN drops to $0\%$ for both classes, shown in Fig. 9. The accuracy for the exact GCNN stays exactly the same as in Fig. 8 (right), showing that is is able to distinguish 6s and 9s under rotation. Furthermore, it also shows that the approximately invariant network uses absolute rotation for classification, while an exactly invariant network does not.

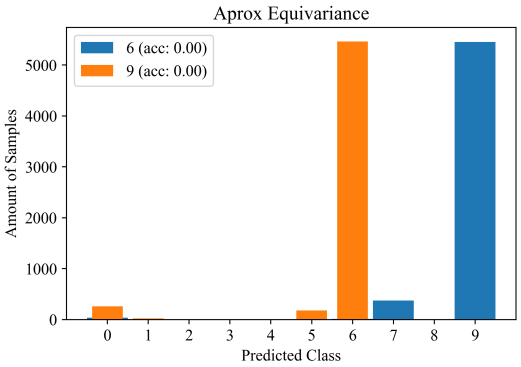



Figure 9. The predicted classes for an approximate GCNN on the 6 and 9 classes, when rotating the input $180°$. The performance drops to $0\%$, showing the network learns absolute rotation.

## D. Texture Based Decisions

We made multiple attempts at explaining why a *p4*-ResNet18 achieves a higher performance on ImageNet than a standard ResNet18, even though ImageNet is not generally known for containing rotations.

If the performance increase was due to some classes containing rotations, one would expect that the performance for both network would be equal for most classes with an exception of a few. However, this is not what we observed, as we observed a general increase in performance for almost all classes. This lead us to the hypothesis that the performance increase was not due to specific classes, but due to a more general phenomenon.

We hypothesised that the performance increase was due to the *p4*-ResNet18 being better at recognising textures. Since textures often occur in different orientations, a network that is better at recognising these orientations might use it to gain an advantage. To test this hypothesis, we made our own version of MNIST, called TexNIST, a visualisation can be found Fig. 10 (right). To create TexNIST we performed the following steps:

1. Scale each MNIST entry from $28 \times 28$ to $48 \times 48$.
2. Apply an pixel-wise square root to each entry make numbers bolder, visualised in Fig. 10 (left). This way the textures should be more readable.
3. Create a list of 10 textures. Each class is assigned a single texture, a random portion of which is sampled to texture each resized MNIST entry.
4. Each entry is made greyscale to prevent the model from learning colour rather than shape or texture.

We then took 2 networks: Z2CNN and P4CNN, and trained both on BoldMNIST and TexNIST. Each of the 4 trained networks was then evaluated on *p4* rotated versions and non-rotated versions of both datasets, resulting in the accuracies found in Tab. 7.

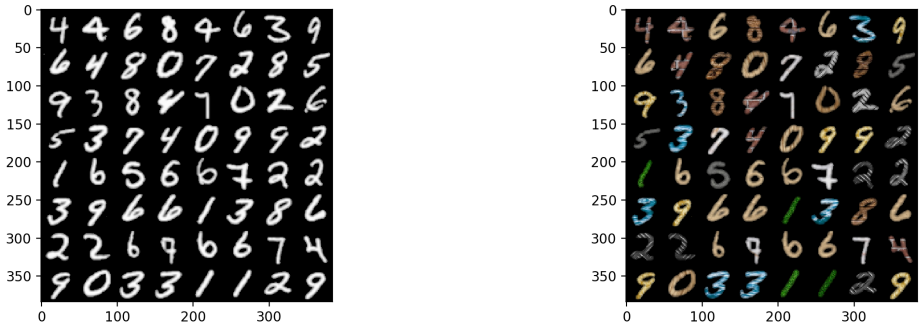


Figure 10. **Left:** A resized bolder version of MNIST, called *BoldMNIST*. **Right:** A coloured version of *TexNIST*. The version used for training is in black and white.

| Model | $Evaluation \rightarrow$ $Training \downarrow$ | BoldMNIST | TexNIST | p4-BoldMNIST | p4-TexNIST |
|---|---|---|---|---|---|
| P4CNN | BoldMNIST | $99.0 \pm 0.2$ | $90.0 \pm 3.6$ | $94.6 \pm 0.5$ | $80.3 \pm 4.6$ |
| | TexNIST | $24.5 \pm 4.4$ | $99.5 \pm 0.2$ | $21.1 \pm 3.2$ | $97.5 \pm 1.0$ |
| Z2CNN | BoldMNIST | $99.1 \pm 0.1$ | $92.3 \pm 2.6$ | $25.3 \pm 0.5$ | $22.6 \pm 1.3$ |
| | TexNIST | $36.6 \pm 3.9$ | $99.5 \pm 0.3$ | $10.8 \pm 1.0$ | $49.2 \pm 3.1$ |

Table 7. Accuracies for texture experiment.

The results are obtained from 5 repeats per condition. The P4CNN is only approximately equivariant for an input size of $48 \times 48$, as the performance of the network on TexNIST is not identical to the performance on p4-TexNIST. An interesting observation however is that, contrary to our hypothesis, Z2CNN seems to evaluate more based on texture than P4CNN. To make this observation, we look at the performance of a network trained on TexNIST and evaluated on BoldMNIST. A network that bases its choice more on texture will have a lower performance on BoldMNIST than a network that decides more based on shape. Our P4CNN and Z2CNN perform identically when trained and tested on TexNIST. However, when evaluated on BoldMNIST, the P4CNN accuracy drops to an average of $24.5 \pm 4.4$, while Z2CNN only drops to $36.6 \pm 3.9$, suggesting the P4CNN classifies more on shape than the Z2CNN.

To evaluate our hypothesis on a more natural dataset, we used the approach en implementation provided by Geirhos et al. [12] and compared a P4-ResNet18 and standard ResNet18 on their custom ImageNet dataset. The results for a selection of the classes can be found in Fig. 11. No clear difference between the networks decisions can be observed.

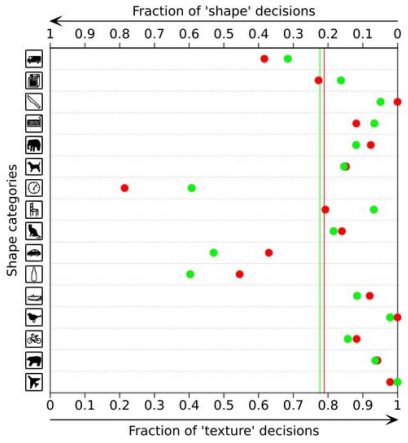



Figure 11. Evaluation of ResNet18 (red) and P4-ResNet18 (green) decisions on ImageNet, following the work by Geirhos et al. [12]

.

While the results for TexNIST raise some interesting suggestions, due to the contradicting results on the other dataset, we can not draw any hard conclusions from this experiment.

# 3

# Equivariance

In this chapter, the definitions for equivariance and symmetry groups are given. The goal is to offer a summarised overview of what these concepts are, so that they can be applied to computer vision tasks in the next chapter.

## 3.1. Equivariance and Invariance

A function $f$ is said to be equivariant to a transformation $T$ if applying $T$ to the input results in a predictable output $T'(f(x))$ for all possible inputs. We can formalize this into the following equation:

$$f(T(x)) = T'(f(x)) \tag{3.1}$$

In other words, first transforming by $T$ and then applying $f$ should give the same results as first applying $f$ and then transforming by $T'$, as visualised in Figure 3.1.

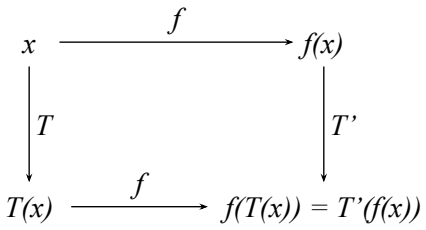



**Figure 3.1:** Visualisation of the definition of equivariance.

To give an example of an equivariant function, we know that scaling should be equivariant to rotation. First scaling and then rotating should give the same results as first rotating and then scaling. We can replace the $f$ function with scaling and the $T$ and $T'$ function with rotation, a visualisation of which is given in Figure 3.2.

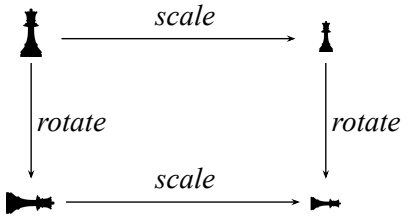



**Figure 3.2:** Example of equivariance.

Interestingly, $T$ does not need to be equal to $T'$. *Invariance* is a special case of equivariance, where $T'$ is the identity function and does not change its input. Therefore, a function $f$ is invariant to transformation $T$ if the following equation holds:

$$f(T(x)) = f(x) \tag{3.2}$$

In other words, the output of $f$ should not change when the transformation $T$ is applied to the input. For example, we can say that a function that returns the average colour of an image is invariant to a rotation transformation, as rotating the input does not affect the average colour. A visualisation of this invariant function is given in figure 3.3.

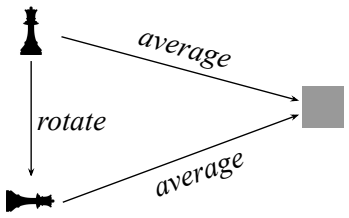



**Figure 3.3:** Example of invariance.

## 3.2. Symmetry Groups

Rather than being equivariant to specific transformations, functions are often described as being equivariant to specific symmetry groups. A symmetry group $G$ consists of a set of transformations for which the following conditions hold:

- For each pair of transformations $g, h \in G$, their composition $gh$ is also in the symmetry group. For example, this means that if the rotation of $90°$ is in $G$, the rotation of $180°$ should also be in $G$.
- For all transformations $g \in G$, their inverse transformation $g^{-1}$ should also be in $G$. Furthermore, composing $g$ with $g^{-1}$ should result in the identity transformation, more formally $g^{-1}gx = x$. For example, if $G$ were to contain a rotation of $90°$, the rotation of $-90°$ should also be in $G$.
- All transformations in the group should be associative, more formally $i(gh) = (ig)h \quad \forall i, g, h \in G$. In other words, the order in which the transformations are applied should not matter.

Using this definition, we can formalise Equation (3.1) to hold for groups rather than singular transformations, giving us:

$$f(T_g(x)) = T'_g f(x) \quad \forall g \in G \tag{3.3}$$

Symmetry groups can include continuous transformations, such as rotations. However, due to the discrete nature of computers, it is difficult to become equivariant towards continuous transformations. Therefore, continuous transformations are often approximated using a discretisation of the transformation.

The work presented in this thesis is based on three symmetry groups:

**The *z2*-Group** This thesis contains multiple references to the *z2*-group. This group includes all compositions of translations on a discrete 2-dimensional input. In the context of this work, a translation is a shift of 1 or more pixels in an image.

**The *p4*-Group.** The main focus of Chapter 2 is on the *p4*-group. This group includes all compositions of translations and rotations of $90^\circ$ on a discrete 2-dimensional input. Because the transformations contained in the group need to be discrete, the continuous rotation transformation is approximated with four rotations: $0^\circ$, $90^\circ$, $180^\circ$ and $270^\circ$.

**The *p4m*-Group.** The work presented in Chapter 2 is in theory also applicable to the *p4m*-group, which is equal to the *p4*-group but also includes the reflection transformation. Even though we do not explicitly mention the group in the paper, it is good to know of its existence, as it is often mentioned in related work.

# 4

# Equivariance in Computer Vision

The goal of this chapter is to offer an overview of the relevant concepts of computer vision and to offer an insight into how these concepts relate to equivariance.

## 4.1. What is Computer Vision?

Broadly speaking, computer vision is the science of solving how to make computers gain a high-level understanding of digital images or collections of images. Computer vision tasks are already highly prevalent in our society, from face recognition algorithms to self-driving cars. The sub-domain of computer vision that is most relevant for our scientific work is *Object Classification*, which consists of classifying one or more objects within an image. The current dominant strategy for these types of tasks is to use *Convolutional Neural Networks* (CNN), the first learnable version of which was introduced by Yann LeCun [5]. CNNs learn by showing them a lot of training data, also referred to as training. The network can then be tested on a separate test dataset to approximate how well the network will perform on data not previously seen during training.

## 4.2. Convolutional Neural Networks

Convolutional Neural Networks are a type of neural network that embed spatial information into the network using *convolutions*, which are translation equivariant functions. This means that, in theory, CNNs should be able to efficiently recognise objects regardless of their absolute position in the input image. A CNN should therefore be *z2*-equivariant.

A CNN is constructed by using several consecutive layers, with each layer performing multiple separate convolutions. The amount of layers in a network is referred to as network depth and the amount of convolutions per layer is referred to as network width. Layers often consist of more than just convolutions, however, these other components are less relevant to understand Chapter 2.

**Convolutions.** First, it is important to mention that most convolutions presented in computer vision literature are actually *cross-correlations*. In this subsection, the difference between the two is explained. In the rest of the work both are referred to as convolutions, as interchanging the two does not create any fundamental differences. For humans however, cross-correlations are much more intuitive, which is why convolutions are often explained using cross-correlations instead.

A convolution of a feature map $F$ with a filter (or kernel) $H$, resulting in a new feature map $G$ can be written as shown in Equation (4.1).

$$F * H = G \tag{4.1}$$

The filter $H$ is a square 2-dimensional matrix of real values with a size of $2k + 1$, also denoted as $H : \mathbb{R}^{2k+1 \times 2k+1}$. The width and height of $H$ is often referred to as the *kernel size*. Using the same notation, feature map $F$ can be denoted as $F : \mathbb{R}^{m \times n}$, with $m, n \geq 2k + 1$. Convolving $F$ with $H$ then results in a new slightly smaller feature map $G : \mathbb{R}^{m-2k \times n-2k}$. The precise calculation for $G$ is given in

Equation (4.2).

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} F[i-u, j-v]H[u,v] \tag{4.2}$$

While this equation can seem difficult to understand, once visualised, it becomes more intuitive. A cross-correlation is denoted as $G = F \star H$ and is extremely similar to a convolution. A cross-correlation can be denoted as Equation (4.3).

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} F[i+u, j+v]H[u,v] \tag{4.3}$$

From Equation (4.2) and Equation (4.3), one can see that the only difference between the convolution and the cross-correlation is in what order the values from the filter are applied to the feature map. Therefore, a convolution is simply a cross-correlation with its filter flipped both horizontally and vertically. As the values in filters are typically learnt by computers, using a cross-correlation rather than a convolution will result in the computer learning the same filter but flipped. Therefore, for computers, there is no practical difference between convolutions and cross-correlations.

We can visualise the cross-correlation equation $F \star H = G$ as shown in figure 4.1, where the cross-correlation has a *kernel size* of 3.

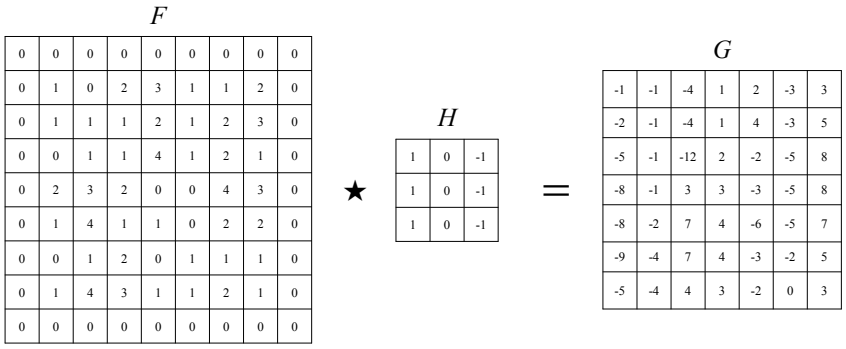



**Figure 4.1:** Cross correlation.

To offer more insight into how the cross-correlation is actually performed, Equation (4.3) can be visualised as Figure 4.3. Starting from the top left, a patch with a size equal to the filter is selected from the input feature map. The sum of the element-wise product between the patch and the filter is then used to calculate the respective value in the output feature map, visualised in Figure 4.2.

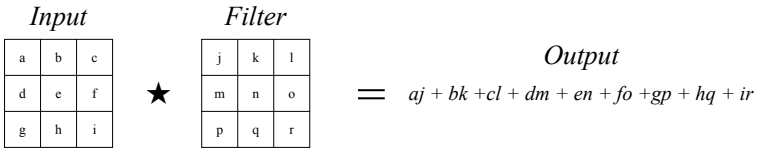



**Figure 4.2:** Single kernel multiplication

The filter is then shifted one column to the right and the same process is repeated. Once the filter cannot shift more to the right, it starts back at the left side of the input, but shifted one row down. This process is repeated until all values of $G$ are calculated. Figure 4.3 shows Equation (4.3) being applied for the first two steps of the process.

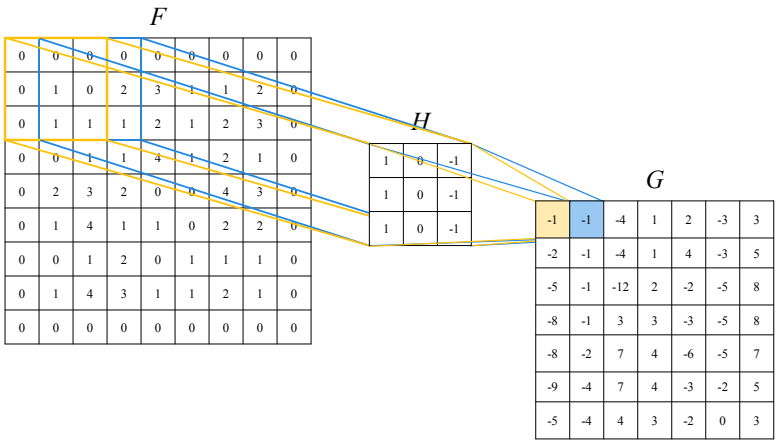



**Figure 4.3:** Cross Correlation.

When performing a convolution, the output $G$ is smaller than the input $F$. This is why input feature maps are often padded. *Padding* is adding additional numerical values to the borders of your feature map. The most frequently used type of padding is 0-padding, where all the added values are equal to 0. Padding can be used to create a larger feature map $F'$, such that the output of $F' \star H$ results in a feature map $G$ with the same dimensions as the original feature map $F$. Looking back at Figure 4.1 and Figure 4.3, one can see the input is 0-padded.

Because convolutions apply the same filter at all locations in the input feature map, convolutions and cross-correlations in theory return the same result regardless of where in the input values are located in the feature map. In other words, they are equivariant to translations. However, in practice they are only approximately equivariant to translations, as works have shown that convolutions and cross-correlations break translation equivariance [8, 9]. Furthermore, the padding explained in the previous paragraph introduces a new problem called *border effects* [2]. Border effects can make the absolute position of the input values influence the output, further breaking translation equivariance.

## 4.3. Subsampling

In CNNs, it is often desirable to summarize local spatial information from feature maps. This reduces the amount of necessary computations and is one of the concepts that allows computers to learn higher-level representations of the input images. For example, after multiple layers, this could allow a network to represent a dog that is in the input as a single numerical value. The concept of summarizing local spatial information is often referred to as subsampling. The two most common methods for subsampling are *strided convolutions* and *pooling*.

**Strided Convolutions.** Strided convolutions are extremely similar to the standard convolutions explained in Section 4.2. The only difference is that strided convolutions shift their kernel multiple pixels for each output value calculation, rather than a single pixel. The amount of pixels the kernel shifts is referred to as the *stride*. In Figure 4.4, an example is given of a convolutions with a stride of 2. Except for the stride, this convolution is identical to the one performed in Figure 4.3. One can see that the output in Figure 4.4 is smaller.

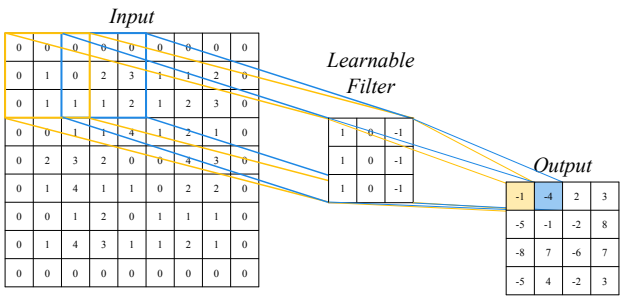



**Figure 4.4:** Strided Convolution.

**Pooling.** Pooling is similar to strided convolutions, as pooling also uses a *kernel size* and a *stride*. However, for pooling, the kernel size and stride are often equal. Rather than applying a filter at each location, pooling layers often apply a *max-function* or an *average-function*, taking either the maximum or average of the selected values. These types of pooling are referred to as *MaxPool* and *AvgPool* respectively. A visualisation of a *MaxPool* with a kernel size and stride of 2 is given in Figure 4.5. Each location in the input where the *max-function* is applied is coloured and the selected value is made bold.

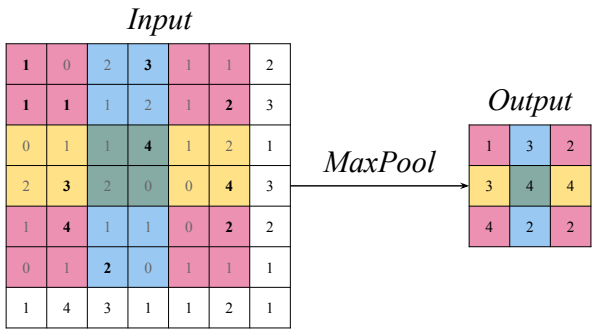



**Figure 4.5:** MaxPooling.

As you can see in Figure 4.5, there are a column and a row that are not sampled to compute the output. This is because the kernel size did not fit within the input feature map anymore, therefore skipping these values. Skipping specific values is one of the main causes of the problems discussed in Chapter 2. This phenomenon is not exclusive to pooling layers and can also occur in strided convolutions.

## 4.4. Desirability of Equivariance in Computer Vision

Equivariance is desirable in computer vision because it embeds prior knowledge of the physical world into neural networks. The introduction of convolutions was one of the biggest innovations in the field of computer vision, as it embedded translation equivariance as prior knowledge, becoming one of the core concepts that allows computer vision to be so efficient. A visualisation of convolutions being equivariant to translations is given in Figure 4.6. One can see that first convolving and then translating returns the same result as first translating and then convolving.

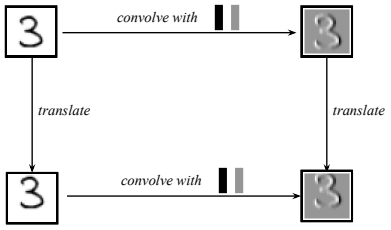



**Figure 4.6:** CNNs are equivariant to translations.

However, translations often are not the only relevant symmetry. Take for example the field of histopathology, which involves examining organics cells under a microscope. If the task is to find cancerous cells, it is undesirable for the location of the cell within the input image to influence the output. Therefore, the CNN should be translation equivariant. However, due to the nature of cells, their rotation within the input image should also not influence the output, which can be solved by becoming equivariant to rotations and translations, also referred to as *p4*-equivariance. Sadly, standard CNNs are not equivariant to rotations, as shown in Figure 4.7. One can see that the order of convolving and rotating influences the output.

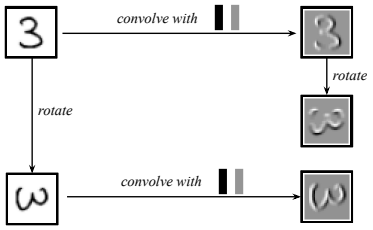



**Figure 4.7:** CNNs are not equivariant to rotations.

Being able to embed other types of equivariance into CNNs could therefore result in large performance increases.

## 4.5. Group Equivariant Convolutions

Many works have been published about learning and embedding equivariant properties into CNNs. For this work, the most relevant being the introduction of *Group Equivariant Convolutional Networks* (GCNN) by Taco Cohen [1]. In their work, they introduce Group Equivariant Convolutions (GEC), which are similar to standard convolutions, but have an additional group dimension.

This section strives to explain GECs as clearly as possible. The most important takeaway is that GECs are a special type of convolution that uses several standard convolutions in a clever way to become equivariant to symmetry groups other than the *z2*-group.

Normal CNNs are considered to be equivariant to the *z2*-group, which is why GCNNs start with a lifting layer, which *'lifts'* the current symmetry group of the network from the *z2*-group to another symmetry group. This layer applies the symmetry group to the filter used for convolution and then uses each resulting filter for a standard convolution on the input feature map. When creating a GCNN that is equivariant to the *p4*-group, the lifting layer is called a *P4Z2* convolution, lifting the symmetry group of the network from *z2* to *p4*. A visualisation of such a lifting layer is given in Figure 4.8. A single $3 \times 3$ filter is learnt and rotated $90°$ three times to obtain the other filters. Each of these filters is then used to obtain a seperate feature map using a standard convolution, explained in Section 4.2. The 4 resulting 2-dimensional feature maps together then form a new 3-dimensional feature map that is in the *p4*-group, rather than in the *z2*-group.

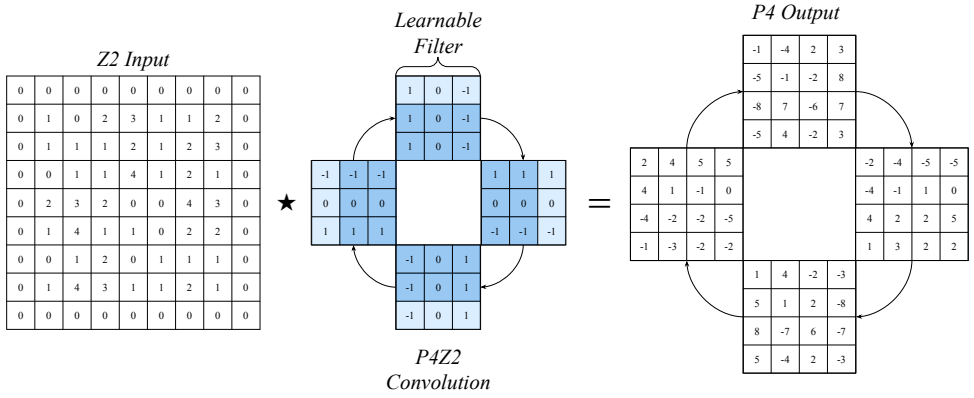



**Figure 4.8:** A P4Z2 lifting layer, consisting of 4 standard convolutions.

Once the symmetry group of the network has been lifted to the *p4*-group, a new type of convolution is needed to retain the symmetry group, namely the P4 convolution, visualised in Figure 4.9. While the P4Z2 convolution only learns a single $3 \times 3$ filter, the P4 convolution learns 4 $3 \times 3$ filters, each visualised as a separate colour. The light edge on each filter represents its rotation. The collection of these 4 filters is indicated as the *P4 Filter* in Figure 4.9. To obtain the other 3 P4 filters used in the P4 convolution, each filter in the learnt P4 filter is rotated and the filters are shifted by one in the group dimension. From a visual point of view, one can view it as rotating the entire P4 filter as one solid object.

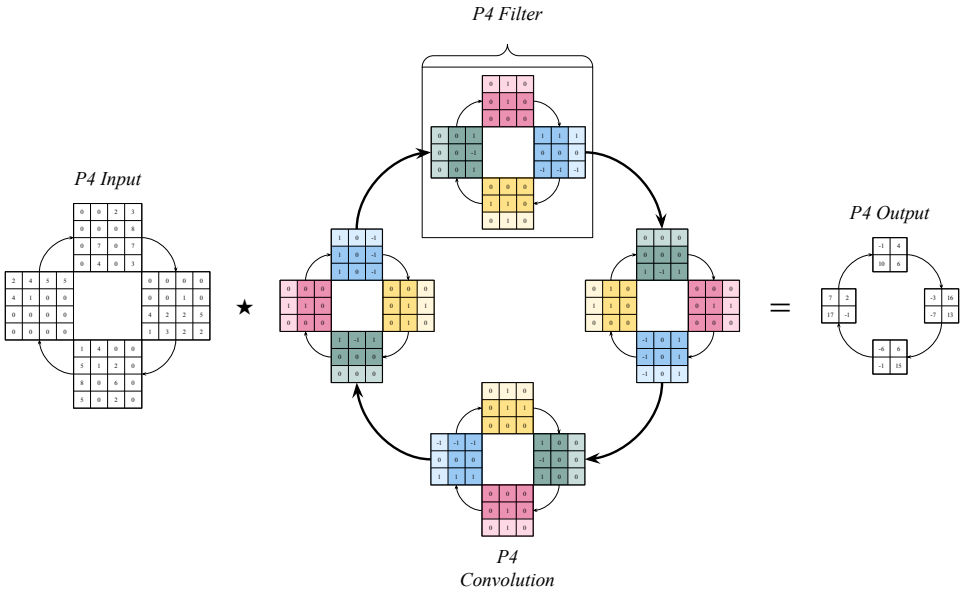



**Figure 4.9:** A P4 Convolution, consisting of 4 convolutions with a P4 filter, each consisting of 4 standard convolutions.

To fully understand Figure 4.9 and how to calculate the output, one has to understand how to convolve a P4 input with a single P4 filter, visualised in Figure 4.10. A convolution with a single P4 filter consists of 4 standard convolutions, where each group dimension in the input is convolved with the standard filter in the same group dimension from the P4 filter. The resulting feature maps are then summed to calculate the final feature map. A full P4 convolution consists of 4 convolutions with a P4 filter, one for each instance in the group dimension. This results in an output feature map that is also in the *p4*-group.

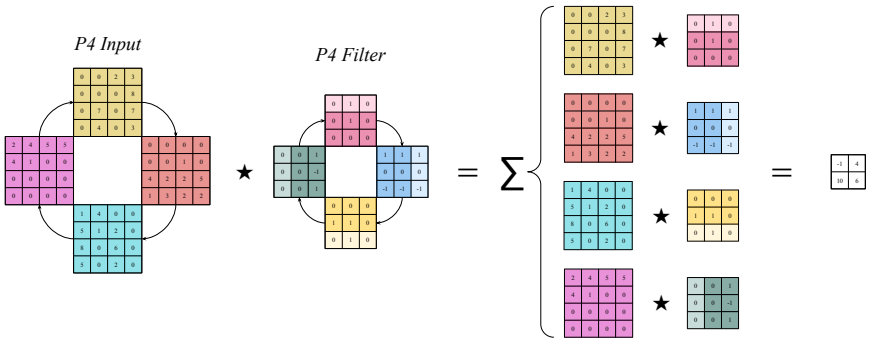



**Figure 4.10:** Convolution with a single P4 filter, consisting of 4 standard convolutions.

Finally, to create an invariant GCNN rather than an equivariant GCNN, a pooling layer over the group dimension can be introduced at the end of the network. This concept is called *coset pooling*. This can be done when invariance is more desirable than equivariance or to reduce the size of the network. It is important to note that when using this approach, a network that is not equivariant will also not become invariant.

## 4.6. Inherent and Learnt Equivariance

Both standard CNNs and GCNNs are able to learn to become equivariant from data. Therefore it is important to make the distinction between inherent and learnt equivariance. Standard CNNs are inherently equivariant to the *z2*-group, while *p4*-GCNNs are inherently equivariant to the *p4*-group. Both can learn equivariance to other symmetry groups. For example, a standard CNN can learn to become equivariant to the *p4*-group. This happens when the CNN learns filters that are *p4*-equivariant. An example of which is the blur filter, where the filter takes an average of the local area, a visualisation of which is given in Figure 4.11.

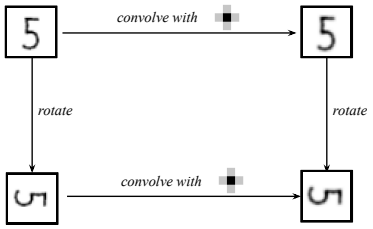



**Figure 4.11:** CNNs can learn to be equivariant to rotations.

However, learnt equivariance has multiple downsides. Firstly, the desired symmetry group might not be properly represented in the training data, making the network learn an incorrect symmetry. Secondly, learning equivariance often results in redundant filters in the network, wasting computations and space that could be used more efficiently. Lastly, it is not guaranteed that the symmetry is learnt from the data, as this is dependent on a large amount of parameters used when training and constructing the network.

# 5 Datasets

This chapter contains a few samples from the datasets mentioned in the paper, as to create an impression of the data. It should not be necessary to understand Chapter 2, but could offer some useful insights.

## 5.1. MNIST

The Modified National Institute of Standards and Technology (MNIST) dataset [4] is a dataset that consists of images of handwritten digits from 0 to 9. The database is extremely popular in deep learning research, as it is a relatively simple database to train and test on. It consists of 10.000 training images and 60.000 testing images, each having a dimension of $28 \times 28$ and being black and white. In the research presented in Chapter 2, *RotMNIST* is also used. RotMNIST is a uniformly rotated version of MNIST. A few samples from the dataset can be found in Figure 5.1.

**Figure 5.1:** MNIST samples

## 5.2. PatchCamelyon

PatchCamelyon (PCAM) [7] is a dataset from the histopathology field. It consists of 327.680 colour images, each with a dimension of $96 \times 96$. All images contain histopathologic scans of lymph node sections. The task associated with this dataset is to determine whether the image contains cancerous tissue. The images therefore each have a binary label, indicating whether they contain cancerous tissue or not. The appeal of this dataset for the work in Chapter 2 is that, in theory, the problem is rotation invariant, as the rotation of the scanned cells should not influence the classification. A few samples from the dataset can be found in Figure 5.2.

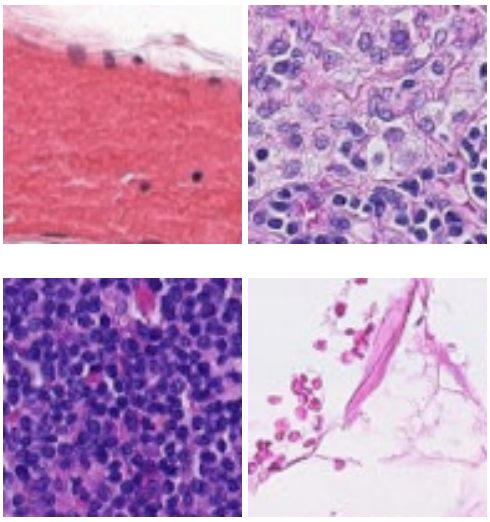


**Figure 5.2:** PatchCamelyon samples

## 5.3. Cifar

The Cifar10 dataset [3] is often used in deep learning, as it can offer a significant challenge while still being trainable within reasonable time on a single GPU. It consists of 60.000 $32 \times 32$ colour images, which are often split into 50.000 training images and 10.000 test images. The dataset contains 10 classes which the deep learning model should distinguish between. A few samples from the dataset can be found in Figure 5.3. Cifar100 is an extension of Cifar10 and contains 100 classes rather than 10.

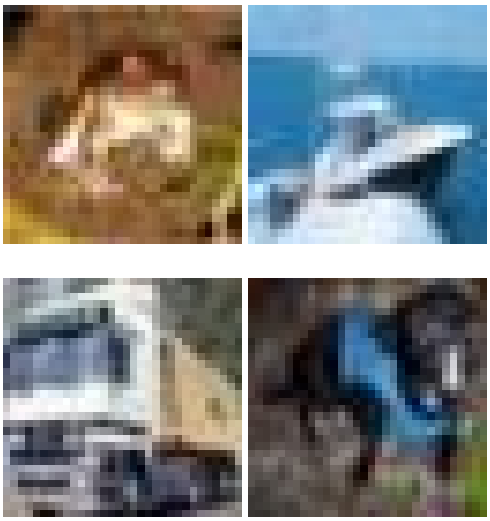


**Figure 5.3:** Cifar10 samples

## 5.4. ImageNet

ImageNet [6] is a dataset containing 14.197.122 colour images, each classifiable into one of a 1000 classes. It is often seen as one of the most significant challenges in deep learning research. Due to the enormous size of the dataset, training and testing on this dataset can take up to a week for a single model. This is why papers often train only a single model on the dataset, rather than using repeated runs. A few samples from the dataset can be found in Figure 5.4. As is evident from Figure 5.4, the images do not all have the same dimensions. The most common solution to this problem is to downscale the images and then take a $224 \times 224$ sample from the center of the image to feed into the model.

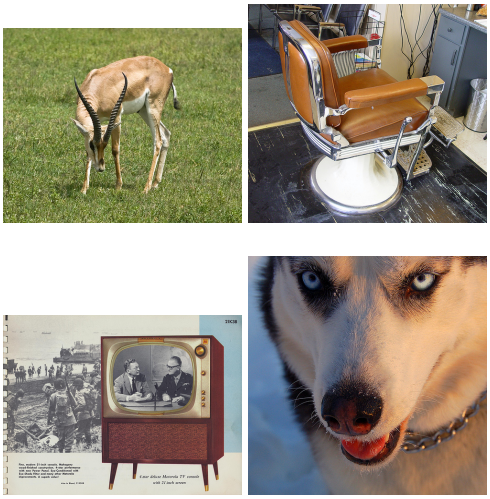


**Figure 5.4:** ImageNet samples

# References


[1] Taco S. Cohen and Max Welling. “Group Equivariant Convolutional Networks”. In: (2016). DOI: 10.48550/ARXIV.1602.07576. URL: https://arxiv.org/abs/1602.07576.

[2] Osman Semih Kayhan and Jan C. van Gemert. *On Translation Invariance in CNNs: Convolutional Layers can Exploit Absolute Spatial Location*. 2020. DOI: 10.48550/ARXIV.2003.07064. URL: https://arxiv.org/abs/2003.07064.

[3] Alex Krizhevsky. “Learning Multiple Layers of Features from Tiny Images”. In: (2009), pp. 32–33. URL: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

[4] Yann LeCun and Corinna Cortes. “MNIST handwritten digit database”. In: (2010). URL: http://yann.lecun.com/exdb/mnist/.

[5] Yann Lecun, Koray Kavukcuoglu, and Clement Farabet. “Convolutional Networks and Applications in Vision”. In: May 2010, pp. 253–256. DOI: 10.1109/ISCAS.2010.5537907.

[6] Olga Russakovsky et al. “ImageNet Large Scale Visual Recognition Challenge”. In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[7] Bastiaan S Veeling et al. “Rotation Equivariant CNNs for Digital Pathology”. In: (June 2018). arXiv: 1806.03962 [cs.CV].

[8] Jin Xu et al. *Group Equivariant Subsampling*. 2021. DOI: 10.48550/ARXIV.2106.05886. URL: https://arxiv.org/abs/2106.05886.

[9] Richard Zhang. *Making Convolutional Networks Shift-Invariant Again*. 2019. DOI: 10.48550/ARXIV.1904.11486. URL: https://arxiv.org/abs/1904.11486.