Deep Neural Networks with Efficient Guaranteed Invariances

Matthias Rath[*,†]
[*]Cross-Domain Computing Solutions
Robert Bosch GmbH
Stuttgart, Germany

Alexandru Paul Condurache[*,†]
[†]Institute for Signal Processing
University of Lübeck
Lübeck, Germany

Abstract

We address the problem of improving the performance and in particular the sample complexity of deep neural networks by enforcing and guaranteeing invariances to symmetry transformations rather than learning them from data. Group-equivariant convolutions are a popular approach to obtain equivariant representations. The desired corresponding invariance is then imposed using pooling operations. For rotations, it has been shown that using invariant integration instead of pooling further improves the sample complexity. In this contribution, we first expand invariant integration beyond rotations to flips and scale transformations. We then address the problem of incorporating multiple desired invariances into a single network. For this purpose, we propose a multi-stream architecture, where each stream is invariant to a different transformation such that the network can simultaneously benefit from multiple invariances. We demonstrate our approach with successful experiments on Scaled-MNIST, SVHN, CIFAR-10 and STL-10.

1 INTRODUCTION

Deep Neural Networks (DNNs) are one of the core drivers of technological progress in various fields such as speech recognition, machine translation, autonomous driving or computer vision (LeCun et al., 2015). At the core of their success lies the ability to process large amounts of data to yield solutions with remarkable generalization properties (Lust and Condurache, 2020b). However, in many practical applications, the data is expensive to collect, store and label. Furthermore, it is rather difficult for humans to understand how such correlation-based methods work, which

is a necessary first step in optimizing or adapting them to new application domains. Additionally, a certain degree of understanding and trust in the DNN's output is essential, e.g., when safety plays a major role (Lust and Condurache, 2020a).

Often, *prior knowledge* about transformations that modify the desired output in a predictable way is available before training. Leveraging prior knowledge increases the interpretability of DNNs while also improving the sample complexity – hence reducing the amount of data needed to obtain a desired performance. When incorporating this valuable prior knowledge into deep learning architectures, it is advantageous to *guarantee* the corresponding in- or equivariances. We differentiate between *invariance*, which is the property of a map to yield the same output for a transformed input and *equivariance* which is the property of a map to preserve the transformation of the input, such that the output is transformed predictably. One common example of leveraging equivariance in DNNs are convolutional layers (Fukushima, 1980; LeCun et al., 1990). These are equivariant to translations and can be easily used to generate a translation-invariant representation (e.g. by pooling).

For many tasks, we can define a set of *symmetry transformations* that affect the desired output in a predictable way. For example, in the case of image classification, symmetry transformations of an input object map it within the same image class and thus do not change the desired classification output. Typical examples are rotations, translations and scales. Enforcing invariance to such transformations as an inductive bias decreases the sample complexity by reducing the search space while training a DNN. Moreover, guaranteed invariances contribute towards gaining an intuition on how inference is conducted in the DNN. Cohen and Welling (2016) first used *group equivariant convolutions* (G-Convs) in DNNs to enforce equivariance to transformations such as rotations and flips (Cohen and Welling, 2016; Worrall et al., 2017; Weiler et al., 2018b; Weiler and Cesa, 2019) or scales (Xu et al., 2014; Kanazawa et al., 2014; Sosnovik et al., 2020, 2021). For classification DNNs, those equivariant layers are usually followed by a max pooling operation among the group and spatial dimensions
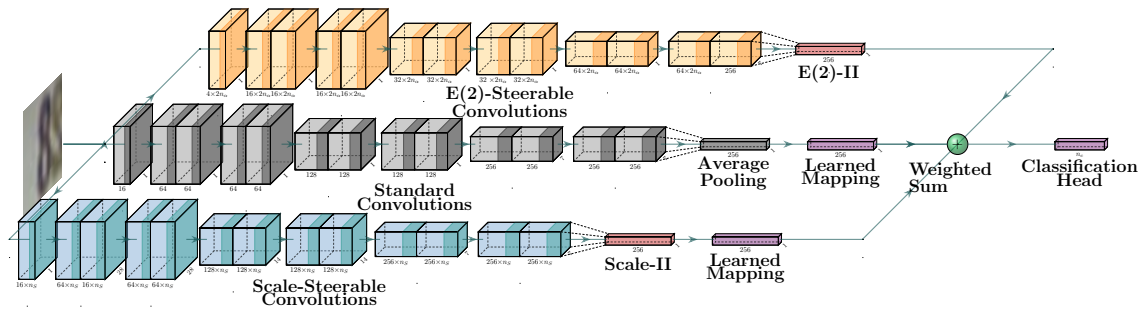
Figure 1: Triple-stream invariant Wide-ResNet16-4 architecture. Includes standard convolutions (grey), rotation-flip-steerable convolutions (E(2), orange), scale-steerable convolutions (blue), invariant integration layers (red), a weighted sum (green) and fully connected layers (purple). Residual shortcut connections are omitted for clarity.

to obtain invariant features that are processed by the final classification layers. While max pooling guarantees invariance, it destroys important information that could be leveraged by a classifier and therefore lacks efficiency. Since the transfer from equi- to invariant features has not been extensively investigated, it promises further improvement capabilities.

Invariant Integration (II) is an algorithm to construct a complete feature space with respect to (w.r.t.) a transformation group (Schulz-Mirbach, 1992). So far, II has been used to replace the global spatial pooling operation for rotation-invariant classification DNNs. This resulted in an improved sample complexity by efficiently leveraging the available prior knowledge while adding targeted model capacity (Rath and Condurache, 2020, 2022). However, II has not been extended to other relevant symmetry transformations such as *scales*.

Group-equivariant DNNs usually incorporate prior knowledge about a single transformation. It is an open challenge how to proceed when *multiple symmetries* are involved because it may be impossible to solve the constraints needed to design transformation-steerable filters depending on the involved groups. Even when avoiding the constraints via interpolation methods, simply expanding the regular equivariant G-Convs is computationally inefficient since the representation grows multiplicatively. For example, for a kernel with 8 rotations and 4 scales, we would have to store $8 \cdot 4 = 32$ responses per kernel.

In this contribution, we extend the II framework beyond rotations and efficiently apply it to multiple transformations at once via a multi-stream architecture. Our **core contributions** are:

- We adapt rotation-II to also include flips to achieve **invariance** to the 2D Euclidean group **E(2)**.

- We **expand II towards scales**, thus covering a larger set of symmetry transformations.

- We address the issue of **multiple invariances** within a single architecture that effectively **combines** several streams, each one with **specific invariances** (see Figure 1). This significantly **extends the practical applicability** of **II**.

- We **evaluate** our approach on **Scaled-MNIST** and on the **real-world datasets** SVHN, CIFAR-10 and STL-10. On STL-10 using only labeled data, we report new **state-of-the-art** results.

2 RELATED WORK

2.1 Group-Equivariant Neural Networks

Group-equivariant convolutional layers were proposed by Cohen and Welling (2016) and applied to discrete 90° rotations and flips by transforming the filters and storing all responses among a *group channel*. This approach uses the *regular* group-representation. Extensions apply this principle to finer-grained rotations via interpolation (Bekkers et al., 2018; Hoogeboom et al., 2018), rotation-steerable filters (Weiler et al., 2018b; Weiler and Cesa, 2019) or by learning all rotated versions of a filter with invariant coefficients (Diaconu and Worrall, 2019). The maximum response can be stored as the orientation in a vector field (Marcos et al., 2017). This is closely related to the *irreducible representation* which achieves continuous rotation-equivariance via complex-valued responses (Worrall et al., 2017). In general, it has been proven that G-Convs are the most general equivariant linear map and a necessary condition for equivariant DNNs (Kondor and Trivedi, 2018; Cohen et al., 2019b; Esteves, 2020).

Besides rotations, in- or equivariance to scale transformations plays a major role in many practical applications. In- or equivariance can again be achieved by sharing filters among different scales using bi-linear interpolation (Xu et al., 2014), scaling the input (Kanazawa et al., 2014) or processing the maximum response and the corresponding scale as a vector field (Marcos et al., 2018). Filters

for scale-equivariant convolutions can be constructed using scale-steerable filters with log-radial harmonics (Ghosh and Gupta, 2019), Hermite polynomials (Sosnovik et al., 2020), optimized discrete bases (Sosnovik et al., 2021), or with separable Fourier-Bessel bases (Zhu et al., 2019). A scale-equivariant G-Conv operating on scale-spaces was introduced in (Worrall and Welling, 2019).

Other work investigates general input domains such as the 3D Euclidean space (Worrall and Brostow, 2018; Weiler et al., 2018a; Cesa et al., 2022), spheres (Cohen et al., 2018; Coors et al., 2018; Esteves et al., 2020; Defferrard et al., 2020; Kondor et al., 2018; Jiang et al., 2019; Shakerinava and Ravanbakhsh, 2021), general manifolds (Cohen et al., 2019a; Finzi et al., 2020), or general groups (Bekkers, 2020; Finzi et al., 2021). Moreover, equivariant versions of non-linear maps such as attention and transformers have been introduced (Fuchs et al., 2020, 2021; Romero and Hoogendoorn, 2020; Romero et al., 2020; Romero and Cordonnier, 2021; Hutchinson et al., 2021; He et al., 2021). The non-linearities and sub-sampling layers used in equivariant CNNs have been investigated in (Franzen and Wand, 2021; Xu et al., 2021).

2.2 Invariant Neural Networks

When solving tasks that require invariance, group-equivariant DNNs are usually followed by a global (max) pooling layer among the group and spatial dimensions. While max pooling guarantees invariance, it is affected by a loss of information, since all but the maximum value are discarded. Obtaining invariance in a more sophisticated way thus promises to further improve invariant DNNs.

One approach to achieve invariance while also ensuring separability is *Invariant Integration* (II), proposed by Schulz-Mirbach (1992, 1994). II is an algorithm to construct a complete feature space w.r.t. a group, i.e., similar patterns are mapped to the same point while distinct patterns are mapped to distinct points. II has been used in combination with conventional machine learning classifiers for image classification (Schulz-Mirbach, 1995), event detection within a cascaded feature extractor to obtain invariance to anthropometric changes (Condurache and Mertins, 2012) or robust speech recognition (Müller and Mertins, 2009, 2010, 2011). Rotation-II has been used to replace the global spatial pooling layer within rotation-invariant deep learning architectures (Rath and Condurache, 2020, 2022) and shown to further increase the sample complexity of such networks. Puny et al. (2021) solve the group average, which II is based on, for larger, intractable groups by integrating over a subset. They applied their method to classification for motion-invariant point clouds and graph DNNs integrating over the whole DNN. The invariant integral has also been used to prove that in- and equivariance improve generalization when the target distribution is in- or

equivariant (Elesedy and Zaidi, 2021).

Whereas Rath and Condurache (2020, 2022) focused on II for the group of rotations, we expand this framework to scales as well as flips (using E(2)). Thereby, we show that the framework can be expanded to general group transformations and generally improves the sample complexity of group-equivariant CNNs in classification tasks. Most related work focuses on single transformation groups. Through our multi-stream architecture, we propose a novel approach that allows our network to learn the best possible combination of invariant features among multiple transformations at once. Another method that combines equivariance to both rotations and scales is the Polar Transformer Network (PTN) which process inputs in the polar coordinate system (Esteves et al., 2018). However, working in polar coordinates, although advantageous for rotations, may prove to be detrimental to translation equivariance. Indeed PTNs are by design invariant to translation, but it is not clear how much other relevant information is destroyed. At the same time, Spatial Transformer Networks (Jaderberg et al., 2015) in general cannot offer invariance guarantees, as they rely on the localization network to learn the correct transformation. The invariance is not fully guaranteed but approximated w.r.t the estimated transformation (STNs) or object center (PTNs). The same is valid for deformable (Dai et al., 2017) and tiled convolutions (Le et al., 2010).

3 THEORETICAL BACKGROUND

3.1 In- and Equivariance

In- and equivariance are mathematical concepts describing the behavior of a map $f : \mathbb{R}^n \to \mathbb{R}^m$ under transformations of the input that can be modeled using the mathematical abstraction of a group. A group is a set G equipped with a group operation $\cdot : G \times G \to G$ fulfilling the four group axioms: closure, associativity, identity and invertibility. The map f is called equivariant w.r.t. G, if left group actions $L_g x$ acting on the input $x \in \mathbb{R}^n$ result in predictable changes $L_{g'}$ of the output

$$\forall x \, \forall g \, \exists g' \text{ s.t. } f(L_g x) = L_{g'} f(x), \qquad (1)$$

where the left group action $L_g : \mathbb{R}^n \to \mathbb{R}^n$ is defined for each group element $g \in G$. The group elements g and g' are not necessarily equal, i.e., the transformation of the output can be different from the one applied on the input, but is predictable. If the output does not change, i.e. $\forall x \, \forall g, f(L_g x) = f(x)$, the function is called invariant.

In the context of CNNs, the network and the layers are maps between feature spaces that can be described by $f : \mathbb{Z}^2 \to \mathbb{R}^n$. The effect of input transformations on the feature maps and outputs can thus be studied using Group Theory. In the course of the paper, we use left actions on the input x by lifting the left group action of G on $\mathbb{Z}^2$ via

$L_g x(y) = x(g^{-1}y)$ where $y \in \mathbb{Z}^2$ are the pixel coordinates. The transformed input is equivalent to the original input at the point $g^{-1}y$ that gets mapped to y by g.

3.2 Group-Equivariant Convolutions

Group-equivariant convolutions are the most general linear map achieving equivariance to a transformation group G (Kondor and Trivedi, 2018; Cohen et al., 2019b; Esteves, 2020). Cohen and Welling (2016) first used G-Convs in the context of DNNs to learn representations with guaranteed equivariance to group transformations. The continuous G-Conv of two functions f and ψ is defined as

$$(f \star_G \psi)(u) = \int_{g \in G} f(g)\psi(u^{-1}g)d\mu(g), \qquad (2)$$

where $d\mu(g)$ is the Haar measure with $\int_{g \in G} d\mu(g) = 1$. The in- and output are defined on the group itself with $g, u \in G$. The *standard convolution* is a special case of G-Convs where the elements are given by $y, t \in \mathbb{Z}^2$ and the inverse action $t^{-1}y$ results in shifts $y - t$. A DNN is group equivariant, if and only if each of its mappings is equivariant to or commutes with the group (Kondor and Trivedi, 2018). In many cases, the transformation group G can be split into the translation group T defined on $\mathbb{Z}^2$ and the corresponding quotient group $H = G/T$. In order to achieve equivariance to G, a H-equivariant convolution can be applied at all spatial locations of the input using a standard convolution. Different representations can be used to compute and store the equivariant features. *Irreducible* representations require the minimal possible number of parameters, but often involve complex calculations. For 2D rotations, using irreducible representations results in complex-valued feature spaces where the orientation information is stored via the complex phase. The *regular* representation of a discrete group stores all responses of transformed filters among an additional channel called *group channel*.

To compute regular G-Convs, either the input or the filters need to be transformed for all $g \in G$. A simple method is to transform the filters using interpolation methods such as bi-linear interpolation. However, this introduces sampling artifacts and consequently weakens the equivariance guarantees. An alternative approach is to use transformation-steerable filters, which have been first introduced for rotations (Freeman and Adelson, 1991). While steerable filters introduce a computational overhead compared to simpler interpolation methods, arbitrarily transformed versions can be calculated in closed-form and are thus not afflicted by sampling effects. This concept can be effectively used for G-Convs by restricting the learned filters to linear combinations of steerable basis filters. Weiler et al. (2018b) built steerable filters for rotation G-Convs using a Gaussian kernel which Weiler and Cesa (2019) expanded to the general E(2)-group. Sosnovik et al. (2020, 2021) constructed scale-steerable filter CNNs using 2D Hermite polynomials or a

learned discrete basis. We use the state-of-the-art methods for rotation- (E(2)-STCNNs, Weiler and Cesa 2019) and scale-invariant (DISCO, Sosnovik et al. 2021) tasks as our baseline.

3.3 Invariant Representations in DNNs

Invariance plays a major role in many DNN applications. For the example of classification, input transformations that do not change the desired class output should not change the learned feature space. Group-equivariant DNNs typically use pooling to transfer from equi- to invariant representations. When operating on regular G-Convs, the pooling procedure is two-fold: Pooling among the transformation channel creates an equivariant representation where an input-transformation induces the same transformation in the feature space; and pooling among the spatial dimension obtains the final invariance. For closed groups, such as rotations and flips, spatial average or max pooling can be used to obtain invariant representations. This is different for the scale group, where average pooling over the spatial dimension does not lead to invariant, but rather homogeneous features, i.e., scaling by s modifies the output by multiplying with s^2.

3.4 Invariant Integration

Invariant Integration is an algorithm to construct a complete feature space $\mathcal{F}$ w.r.t. a transformation group G introduced in Schulz-Mirbach (1992). A feature space is complete, if all equivalent patterns w.r.t. the transformation are mapped to the same point while all distinct patterns are mapped to different points. For this mapping, II uses the group average $A[f](x)$ which integrates over all possible transformations $g \in G$ of an input x processed by a polynomial f

$$A[f](x) = \int_{g \in G} f(L_g x)d\mu(g). \qquad (3)$$

In our case, x is the output of the final feature map after pooling among the group dimension. For f, Schulz-Mirbach (1994); Schulz-Mirbach (1995) used the set of monomials $m(x) = \prod_{i=1}^{M} x_i^{b_i}$ with $\sum_i b_i \leq |G|$, defined as a product of individual signal values x_i with exponents b_i, which have been shown to be a good choice to maintain a high expressiveness of the invariant features (Noether, 1916). For 2D rotations and translations, II with monomials within a local neighborhood defined by distances d_i results in

$$A[\mathrm{m}](x) = \frac{1}{N_\phi UV} \sum_{\phi, t} \prod_{i=1}^{M} x[t - L_\phi d_i]^{b_i}. \qquad (4)$$

Rath and Condurache (2020) applied II on top of equivariant G-Convs within a DNN. The II layer is differentiable, if $x_i > 0 \, \forall i$, which allows for an end-to-end optimization of

Matthias Rath[⋆,†], **Alexandru Paul Condurache**[⋆,†]

the DNN via backpropagation. To select a meaningful subset of monomials, an iterative algorithm based on the least square solution of a linear classifier can be used. However, a pruning-based approach leads to a more streamlined training procedure (Rath and Condurache, 2022). Additionally, selecting the monomials can be avoided by replacing them with alternatives well-known in deep learning literature such as self-attention, a weighted sum (WS) or a multi-layer-perceptron. For rotation-II, the special case using a WS achieves the best performance while being easier to train (Rath and Condurache, 2022). II using a WS with a learnable kernel $\psi \in \mathbb{R}^{k \times k}$ applied to N_ϕ finite rotations $\phi \in [0°, \frac{360°}{N_\phi}, \ldots]$ obtained using bi-linear interpolation with input dimensions $U \times V$ is defined as

$$A[\text{WS}](x) = \frac{1}{N_\phi UV} \sum_{\phi,t} \sum_{y \in \mathbb{Z}^2} x(y) L_\phi \psi(y-t). \quad (5)$$

4 METHOD

In this section, we extend rotation-II using a WS to the E(2)-group involving rotations and flips (Section 4.1). We then show why II cannot be straightforwardly applied to scale transformations and introduce an alternative to obtain scale-invariants based on II (Section 4.2). We then propose a scale-invariant CNN including Scale-II (Section 4.3). Finally, we introduce a multi-stream DNN architecture that efficiently combines invariances to multiple transformations within a single DNN (Section 4.4).

4.1 E(2)-Invariant Integration

Equation 5 can straightforwardly be expanded to each discrete subgroup of E(2) (as used by Weiler and Cesa 2019), which contains flips $L_f \psi(y)$ in addition to rotations

$$A[\text{WS}](x) = \frac{1}{2N_\phi UV} \sum_{f,\phi,t} \sum_{y \in \mathbb{Z}^2} x(y) L_\phi L_f \psi(y-t). \quad (6)$$

4.2 Scale-Invariant Integration

In images, objects naturally appear at different scales, e.g., due to variable camera-to-object distances. Hence, DNNs for object classification or detection benefit from invariance to scales. We propose to use II in combination with a scale-equivariant CNN to obtain scale-invariant features. In comparison to the rotation group, discrete scale transformations are not circular and non-invertible due to the loss of information (e.g. during down-scaling). Thus, scales do not satisfy all group axioms and can only be modeled as a semi-group. Schulz-Mirbach (1992) demonstrated that it is impossible to construct invariants by integrating over the scale semi-group while at the same time achieving separability. This prohibits constructing a complete feature space w.r.t scales using the standard II approach. Nev-

ertheless, the group average w.r.t. translations is a homogeneous function w.r.t. scales when using polynomials (Schulz-Mirbach, 1994). This means that the effect of scaling by s on the features obtained using translation-II is defined as $A[f](L_s x) = s^K A[f](x)$ where the order K is defined by the polynomial order of f with the scale operator $L_s[f](y) = f(s^{-1}y) \ \forall s > 0$.

As shown by Schulz-Mirbach (1994), a complete feature space w.r.t. the scale-translation semi-group $G_S = S \rtimes T$ can be calculated by dividing homogeneous functions of the same order. When using monomials m, one resulting scale-invariant integral is given by dividing monomials of the same order $\sum b_{1,i} = \sum b_{2,i}$ with $t \in \mathbb{Z}^2$

$$A_{G_S}[m](x) = \frac{A_T[m_1](x)}{A_T[m_2](x)} = \frac{\sum_t \prod_i x(t - d_{1,i})^{b_{1,i}}}{\sum_t \prod_i x(t - d_{2,i})^{b_{2,i}}}. \quad (7)$$

The special case of translation-II that combines values within a fixed neighborhood using a learnable WS as function f results in a standard convolution followed by Average Pooling and is equivalent to a polynomial of order 1. Consequently, we can choose a divisor of the same homogeneous order to obtain a scale-invariant representation and use the mean of the feature map. We thus introduce the WS-based scale-group average A_{G_S} based on Average Pooling over a standard convolution without bias with translations $L_t, y, t \in \mathbb{Z}^2$ divided by the mean

$$A_{G_S}[\text{WS}](x) = \frac{A_T[\text{WS}](x)}{\sum_y x(y)} = \frac{\sum_y \sum_t x(y)\psi(y-t)}{\sum_y x(y)}. \quad (8)$$

Proof of Invariance. Since translation-II and the mean are both homogeneous w.r.t. scales with factor s^2, it is easy to see that dividing them leads to a scale-invariant solution

$$\frac{A_T[f](L_s x(y))}{\sum_y L_s x(y)} = \frac{s^2 A_T[f](x)}{s^2 \sum_y x(y)} = \frac{A_T[f](x)}{\sum_y x(y)}. \quad (9)$$

4.3 Application to DNNs

Inspired by the work on rotations in Rath and Condurache (2020, 2022), we apply E(2)- and Scale-II (Formulas 6 - 8) on top of the corresponding equivariant features learned using regular G-Convs with steerable filters: E(2)-STCNNs (Weiler and Cesa, 2019) and DISCO (Sosnovik et al., 2021). Those features are processed via max pooling among the group dimension. II then replaces the spatial pooling operation to obtain invariant features that are in turn processed by the final classification layers. For E(2)-II, we use the WS approach. For Scale-II, we investigate both proposed variants. For Scale-II with monomials, we follow Rath and Condurache (2022) and randomly select monomials that are iteratively pruned during training to find the most relevant ones. We ensure the same monomial order between dividend and divisor by normalizing the divisor's exponents with $\frac{\sum_i b_{1,i}}{\sum_i b_{2,i}}$. Additionally, $x_i > 0$ ensures a

differentiable solution. For Scale-II with WS, each convolution depends on all input channels. Consequently, we divide by the mean over all input channels for the WS-based II. Moreover, it is important that $\sum_y x(y) > 0$. Hence, we apply both II variants to the output of a ReLU layer minimized to a small value $\epsilon > 0$.

4.4 Multi-Stream Invariance

To obtain features with guaranteed invariance to multiple transformations, we implement a DNN architecture with multiple streams. Each stream is invariant to a dedicated transformation enforced using G-Convs and II (see Figure 1). While multiple transformations could in theory be embedded into a single network, the regular representations the network needs to process would grow with $\mathcal{O}(N \cdot M)$ for group sizes N and M. In contrast, using a dedicated stream per transformation only increases the number of representations by $\mathcal{O}(N + M) \ll \mathcal{O}(N \cdot M)$.

The input is processed separately by each transformation-invariant stream using G-Convs with steerable filters. II is applied on top of the learned scale- and E(2)-equivariant features to obtain invariant representations. For the standard convolution stream (std.) we use average pooling. Thus, we have $\mathbf{x}_j \in \mathbb{R}^{C_j}$ with $j \in \{e2, \text{scale}, \text{std}\}$. We combine two or three streams, each one invariant to either rotations and flips, scales or translations (std.), using two steps. First, we map all features to the same dimension using a linear map $\tilde{\mathbf{x}}_j = \mathbf{W_j}\mathbf{x_j}$ with $\mathbf{W_j} \in \mathbb{R}^{C_{\text{map}} \times C_j}$. We then combine these streams via a normalized learnable WS, e.g. for the case of three streams $\mathbf{x}_{\text{combined}} = \mathbf{w}_{e2} \circ \tilde{\mathbf{x}}_{e2} + \mathbf{w}_{\text{scale}} \circ \tilde{\mathbf{x}}_{\text{scale}} + \mathbf{w}_{\text{std}} \circ \tilde{\mathbf{x}}_{\text{std}}$ with $\mathbf{w}_j \in \mathbb{R}^{C_{\text{map}}}$ initialized to 1 and normalized s.t. $(\mathbf{w}_{e2} + \mathbf{w}_{\text{scale}} + \mathbf{w}_{\text{std}})_i = 1$ with $i = 1, \ldots, C_{\text{map}}$ and the Hadamard product $\circ$ inspired by the learnable channel-wise scaling used in BatchNorm layers (Ioffe and Szegedy, 2015). This approach allows to combine the invariant features with an explicitly learned factor s.t. the network can learn, which invariances are the most relevant for the corresponding task.

Each stream is pre-trained individually. We then combine the architecture, freeze all convolutional weights and only train the linear maps $\mathbf{W}_j$, the WS $\mathbf{w}_j$ and the classification weights $\mathbf{w}_{\text{out}} \in \mathbb{R}^{C_{\text{map}} \times n_c}$ with n_c classes. The combination head worked best, when mapping the other streams to the E(2)-output, i.e., $C_{\text{map}} = C_{e2}$ and keeping $\mathbf{W}_{e2}$ fixed as the identity. Appendix B.1 provides results with different combinations, e.g., mapping all streams or concatenation. Our dedicated training procedure allows to fine-tune each stream individually, which provides a good initialization point for the combination head and further improves the sample complexity compared to a full end-to-end training (cf. Appendix B.2). While this causes an overhead at train time, all operations can easily be fused into a single network at inference time.
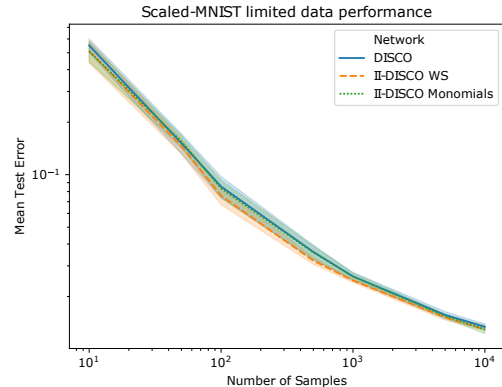

Figure 2: Log. Test Error (TE) on Scaled-MNIST subsets.

5 EXPERIMENTS & DISCUSSION

The proposed scale-II algorithm is evaluated on Scaled-MNIST (Sohn and Lee, 2012) and the invariant multi-stream networks on SVHN (Netzer et al., 2011), CIFAR-10 (Krizhevsky, 2009) and STL-10 (Coates et al., 2011). We use the full dataset and limited subsets with N_t samples to assess the sample complexity. The subsets are sampled with constant class balance and are the same for all variants. For Scaled-MNIST, we use $N_t \in \{10, 50, 100, 500, 1k, 5k, 10k, 12k\}$. For SVHN and CIFAR-10 we use $N_t \in \{100, 500, 1k, 5k, 10k, 50k\}$.

For SVHN, CIFAR-10 and STL-10 we use Wide-ResNets (WRNs, Zagoruyko and Komodakis 2016) as backbone. We use the respective *standard data augmentations*: scales for Scaled-MNIST, no augmentations for SVHN, shifts and crops for CIFAR-10, and shifts, crops and Cutout (Devries and Taylor, 2017) for STL-10. For all single-streams, the number of trainable parameters is constant. We report results for the multi-stream architectures with full streams and constant number of parameters. We optimize all hyper-parameters (HPs) using a 80:20 validation split and Bayesian Optimization with Hyperband (BOHB, Falkner et al. 2018). For all II-WS-layers, we use $k = 3$ and a constant number of in- and output channels. For II with monomials, we use an iterative pruning-based selection with $n_m = \{25, 12, 5\}$ monomial pairs after 0, 5 and 10 epochs following Rath and Condurache (2022). The exact HPs, optimization settings and network architectures can be found in Appendix C. If not mentioned otherwise, we report the mean and standard deviation over three runs.

5.1 Evaluating Scale-Invariant Integration

We evaluate our scale-II layer on the Scaled-MNIST dataset, which consists of hand-written digits artificially scaled with factor $s \in [0.3, 1]$. We use the architecture from Sosnovik et al. (2020) (SES-CNN) and Sos-

Matthias Rath[*,†], Alexandru Paul Condurache[*,†]

Table 1: Invariance Error Δ when obtaining a scale-invariant representation using the respective layer directly on the input and a randomly initialized scale-equivariant network.

Layer	Input	CNN
Average Pooling	0.222	0.090
Mixed Pooling	0.017	0.039
Scale-II Monomials	1.24e-4	**2.64e-7**
Scale-II WS	**2.97e-9**	5.94e-3

Table 2: Test Error (TE) on Scaled-MNIST using a CNN with 5 layers, data augmentation with random scales and an upsampling layer to double the input size.

Method	Scale-II	TE [%]
CNN	-	1.60 ± 0.09
SES-CNN	-	1.42 ± 0.07
DISCO	-	1.35 ± 0.05
II-DISCO	Monomials	**1.30** ± 0.06
II-DISCO	WS	**1.30** ± 0.02

novik et al. (2021) (DISCO) built of three convolutional and two dense layers using $n_S = 4$ scales and replace the final global pooling layer with the scale-II layer. Following Sosnovik et al. (2020, 2021), we compare the invariance error of the scale-II layers to the methods they use to obtain scale-invariant representations: a mixed pooling approach including average and max pooling for Scaled-MNIST and average pooling for STL-10. The invariance error is a simplified version of the equivariance error given as $\Delta = \frac{1}{S} \sum_s \frac{|\psi(x) - \psi(L_s x)|_2^2}{|\psi(x)|_2^2}$. We compute Δ when directly processing the input, i.e., ψ is the II- or pooling-layer and when ψ is the randomly initialized CNN including the respective layer using 100 samples from Scaled-MNIST scaled with $s \in [0.5, 0.55, ..., 1.0]$. The results in Table 1 show that scale-II guarantees invariance as opposed to the pooling approaches. While Scale-II with WS achieves a better error directly on the input, the monomial variant is slightly better when applied within the DNN.

We then evaluate the performance of our scale-II layer on Scaled-MNIST for classification. Here, scale-invariance is paramount to obtain correct results because the test set contains more variability than the training set, thus benefiting scale-invariant algorithms. Table 2 shows the results on the full dataset, Figure 2 with limited training data. We report mean and standard deviation over the six pre-defined data splits. II outperforms the pooling approach on full data and in the limited sample regime highlighting the improved sample complexity of our approach. In summary, the mixed pooling approach used by Sosnovik et al. (2021) does not guarantee scale-invariance which leads to a decreased performance. II allows for invariance guarantees and the WS-variant is easier to optimize than the monomials-variant. Hence, II with a WS outperforms the latter in the limited data domain and is used for all further experiments.

5.2 Multi-Stream Digit Classification

The Street View House Number (SVHN) dataset contains single digits taken from house numbers. SVHN includes digits with different colors, font types, orientations and backgrounds and is thus harder to solve than MNIST. For all experiments on SVHN, we use the core training data, a WRN16-4 architecture, $n_r = 8$ rotations for all E(2)-G-

Convs and $n_S = 3$ scales for the scale stream. The II step is calculated using the same number of rotations and flips. The results on the full SVHN dataset are shown in Table 3 while Figure 3 depicts the results using limited training data.

Using II instead of pooling improves the accuracy for all invariant architectures showing that II better preserves the information when transferring to invariance. Both invariant single-stream networks outperform the std. CNN in the limited and full data domain. This indicates that invariance to rotations and flips as well as scales is valuable information the classifier needs to learn during training – and the training data does not cover enough variability w.r.t rotations and scales for the baseline to learn this information. The rotation- and flip-invariant DNN achieves a better sample complexity than the scale-invariant one, indicating that rotation-invariance is more valuable for this dataset.

The multi-stream networks significantly outperform all single-stream variants including the baseline in all data regimes. The multi-stream architecture learns meaningful combinations of the generated invariants and is able to automatically choose the invariance best-fit for the training data at hand. This allows for best performances on all dataset sizes. In addition, for the rather simple task of classifying numbers, combining only a rotation- and a scale-invariant stream outperforms a standard CNN and leads to almost the same performance as with additional std. convolutions. In this problem setup, rotation- and scale-invariances seem sufficient for optimal performance and are able to recover all necessary object invariances. We conjecture that the learned features in the network's layers within the invariant streams focus on global invariances like illumination and noise while the dedicated II layer handles specific object invariances, e.g. to scales, similar to the scattering transformation (Oyallon et al., 2019).

5.3 Multi-Stream Object Classification

Finally, we evaluate our proposed architecture for the more complex task of object classification on CIFAR-10 and STL-10. Both datasets contain RGB images of ten different object classes. STL-10 is a subset of ImageNet containing 5k training images that is commonly used as a bench-

Table 3: TE on SVHN, CIFAR-10 and STL-10. WRN16-4, WRN28-10 and WRN16-8 are used as baseline architecture. ⋆ indicates constant number of parameters.

II	Streams	Invariance	SVHN [%]	C-10 [%]	STL-10 [%]
	PTN	Rot. & Scale	2.97	6.72	-
x	Single	Std.	2.93 ± 0.02	3.89 ± 0.08	12.02 ± 0.05
x	Single	E(2)	2.64 ± 0.05	2.91 ± 0.13	9.80 ± 0.40
x	Single	Scale	2.71 ± 0.01	4.04 ± 0.03	8.07 ± 0.08
✓	Single	E(2)	2.36 ± 0.05	2.95 ± 0.04	7.67 ± 0.07
✓	Single	Scale	2.54 ± 0.04	3.91 ± 0.12	7.92 ± 0.09
✓	Dual⋆	Scale & E(2)	2.20 ± 0.05	2.96 ± 0.10	6.38 ± 0.15
✓	Dual	Scale & E(2)	2.12 ± 0.11	2.75 ± 0.04	5.95 ± 0.11
✓	Triple⋆	Scale, E(2) & Std.	2.29 ± 0.03	2.74 ± 0.08	6.46 ± 0.08
✓	Triple	Scale, E(2) & Std.	$\mathbf{2.10} \pm 0.07$	$\mathbf{2.68} \pm 0.03$	$\mathbf{5.90} \pm 0.05$

Table 4: Role of the II layer for our triple-stream network on STL-10. An 'x' marks an invariant stream without II.

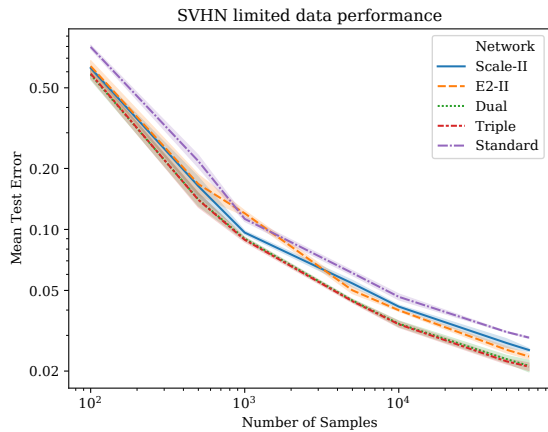| II		
E(2)	Scale	TE [%]
x	x	7.51 ± 0.12
x	✓	7.35 ± 0.09
✓	x	6.34 ± 0.07
✓	✓	$\mathbf{5.90} \pm 0.05$

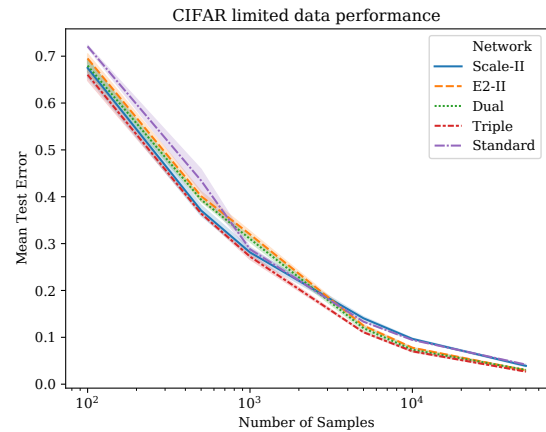Figure 3: Log. TE on subsets of SVHN with full streams.

Figure 4: TE on subsets of CIFAR-10 with full streams.

mark for the limited data performance of object classification networks. STL-10 is more challenging than CIFAR-10 since it contains bigger and more diverse images. For CIFAR-10, we use WRN28-10 and the E(2)-stream with 8, 8 and 4 rotations per residual block. For STL-10, we use WRN16-8 and the E(2)-architecture with 8, 4 and 1 rotation per residual block (Weiler and Cesa, 2019). The E(2)-II-layer is used for 4-rotations and flips or flips-only, respectively. For both datasets, the scale-stream uses 3 scales (Sosnovik et al., 2021). The STL-10 and CIFAR-10 results on full data are shown in Table 3, CIFAR-10 results on limited data in Figure 4. For a fair comparison on STL-10, we adapted the official implementation of Sosnovik et al. (2021) which uses a stride of 1 in the initial convolutional layer by using stride 2 as in Devries and Taylor (2017); Weiler and Cesa (2019).

On CIFAR-10, we again demonstrate an increased sample efficiency of the invariant streams leading to superior performance for small dataset sizes (Figure 4). While the E(2)-invariant network is able to outperform the std. baseline in the full data regime, the scale-invariant network achieves

subpar performance. We interpret the latter as a sign of less variance along the scale mode in this problem setup due to the rather small images contained in CIFAR-10. Thus, other invariances are more important to decide for the correct class. The scale-invariant stream seems to be too restrictive in the sense that it is unable to learn the full set of object invariants that the baseline architecture leverages to classify the objects. On the full data, II improves the performance for the scale-invariant architecture. However, the performance on the E(2)-invariant architecture is only on par. Nevertheless, Rath and Condurache (2022) show that a rotation-invariant architecture using II outperforms the one with pooling in limited data regimes even when achieving slightly worse results on full data. We further investigate the advantages of II in Section 5.4. Our combined network is able to achieve the best results for all data regimes by learning to combine the best information at each dataset size. The triple stream outperforms the dual variant which indicates that the std. stream is able to capture important additional object invariants that are neglected by the restricted, invariant streams.

Matthias Rath[*,†], **Alexandru Paul Condurache**[*,†]

On STL-10, the E2-II network outperforms its counterpart without II significantly. Scale-II also slightly improves upon the baseline and II is clearly beneficial when combining multiple streams (see Table 4). Our multi-stream network achieves a new state-of-the-art result (Table 3), even with constant number of parameters. This shows that incorporating prior knowledge about multiple transformations improves the performance of classification DNNs in the limited data domain, even for complex real-world datasets. Additionally, the results show that the features learned by each stream are complementary, preserved by our II layers and are effectively combined by our proposed multi-stream head. The multi-stream architecture successfully improves the sample complexity while raising the number of parameters – hence increasing generalization.

5.4 Ablation Studies

Multi-Stream: Number of Parameters. In addition to the full multi-stream architecture, we report the performance when keeping the number of parameters constant. Therefore, we shrink each stream by factor 2 or 3 for the dual or triple-stream network, respectively. The results are shown in Table 3 marked with $\star$. Limited data results for SVHN and CIFAR-10 are shown in Appendix A.

Our approach still achieves state-of-the-art performance on all datasets and in all data domains. Combining multiple streams is beneficial, even with constant number of parameters. The dual-stream performs slightly better than the triple-stream. We believe this occurs since the individual streams of the triple-stream architecture are too thin, particularly in early layers.

Furthermore, we train a triple-stream architecture consisting of three standard streams on STL-10. We achieve a test error (TE) of 10.29% which is worse than the single-stream networks with invariance. This shows that the enforced invariance plays a key role for our multi-stream networks.

Multi-Stream: Importance of Invariant Integration. To quantify and demonstrate the importance of the II layer, we compare our multi-stream architecture including II to variants without II on STL-10. This includes a multi-stream architecture, where only pooling is used. The results in Table 4 demonstrate that on a more complex classification task, in low-data regime (i.e. when the training data does not properly cover all variability present in the test data) the multi-stream approach works best when both streams use II rather than pooling.

Training a standard WRN augmented with random 90° rotations and scales $s \in [0.25, 1]$ on STL-10 achieves a TE of 21.80%, which is clearly detrimental compared to the performance without those augmentations (12.08%). Hence, layer-wise, guaranteed in- and equivariance play a key role in the improved sample complexity of our approach.

6 CONCLUSION

In this contribution, we expanded II to scale transformations and showed its effectiveness on Scaled-MNIST. Since Scale-II using a WS is easier to optimize than the monomial variant, we applied it in a multi-stream DNN which besides scales includes a standard convolutional and a rotation-and-flip-invariant stream. This multi-stream DNN covers a variety of practically interesting use cases as shown by an improved sample complexity on SVHN and CIFAR-10 and new state-of-the-art results on STL-10 using only labeled data. We impose invariance to scales and rotation-and-flips in dedicated streams that also learn other global invariances and cover the remaining object invariances with a standard convolutional stream. This guarantees multiple invariances without suffering from the multiplicative increase when directly combining the groups.

Our framework is thought to leverage and honor prior knowledge. Therefore, it is focused on invariance guarantees, which may be rather restrictive in some cases. Specifically, invariance guarantees improve the sample complexity of DNNs leading to a performance boost, when training data is limited. In the large data domain, Vision Transformers (Dosovitskiy et al., 2021) with less geometrical constraints outperform conventional CNNs. Hence, our experiments focus on small-scale datasets. II is a general method that can be expanded beyond rotations and scales, but is restricted to transformations that can be modeled as (semi-) groups. Furthermore, we require an equivariant backbone before transferring to invariance. Nevertheless, the multi-stream network can in theory be enhanced with streams that achieve invariance without using II or G-Convs.

In the future, it is interesting to apply II to tasks where equivariance is helpful, e.g. to infer the pose in object detection. II could still be used for the parts of the network that benefit from invariance. On CIFAR-10 and SVHN, more sophisticated architectures and training methods than WRNs achieve a better performance (Foret et al., 2021; Lim et al., 2019). For a fair comparison, we stuck to WRNs in our experiments. Nevertheless, guaranteed invariances can generally be applied to many architectures in order to improve their sample complexity.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M.,

Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Bekkers, E. J. (2020). B-spline cnns on lie groups. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Bekkers, E. J., Lafarge, M. W., Veta, M., Eppenhof, K. A. J., Pluim, J. P. W., and Duits, R. (2018). Rototranslation covariant convolutional networks for medical image analysis. In *Medical Image Computing and Computer Assisted Intervention - MICCAI 2018 - 21st International Conference, Granada, Spain, September 16-20, 2018, Proceedings, Part I*, pages 440–448.

Cesa, G., Lang, L., and Weiler, M. (2022). A program to build e(n)-equivariant steerable CNNs. In *International Conference on Learning Representations*.

Coates, A., Ng, A. Y., and Lee, H. (2011). An analysis of single-layer networks in unsupervised feature learning. In Gordon, G. J., Dunson, D. B., and Dudík, M., editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011*, volume 15 of *JMLR Proceedings*, pages 215–223. JMLR.org.

Cohen, T., Weiler, M., Kicanaoglu, B., and Welling, M. (2019a). Gauge equivariant convolutional networks and the icosahedral CNN. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, pages 1321–1330.

Cohen, T. and Welling, M. (2016). Group equivariant convolutional networks. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 2990–2999.

Cohen, T. S., Geiger, M., Köhler, J., and Welling, M. (2018). Spherical cnns. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*.

Cohen, T. S., Geiger, M., and Weiler, M. (2019b). A general theory of equivariant cnns on homogeneous spaces. In Wallach, H. M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. B., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 9142–9153.

Condurache, A. P. and Mertins, A. (2012). Sparse representations and invariant sequence-feature extraction for event detection. *VISAPP 2012 - Proceedings of the International Conference on Computer Vision Theory and Applications*, 1.

Coors, B., Condurache, A. P., and Geiger, A. (2018). Spherenet: Learning spherical representations for detection and classification in omnidirectional images. In Ferrari, V., Hebert, M., Sminchisescu, C., and Weiss, Y., editors, *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part IX*, volume 11213 of *Lecture Notes in Computer Science*, pages 525–541. Springer.

Dai, J., Qi, H., Xiong, Y., Li, Y., Zhang, G., Hu, H., and Wei, Y. (2017). Deformable convolutional networks. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 764–773. IEEE Computer Society.

Defferrard, M., Milani, M., Gusset, F., and Perraudin, N. (2020). Deepsphere: a graph-based spherical CNN. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Devries, T. and Taylor, G. W. (2017). Improved regularization of convolutional neural networks with cutout. *CoRR*, abs/1708.04552.

Diaconu, N. and Worrall, D. E. (2019). Learning to convolve: A generalized weight-tying approach. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, pages 1586–1595.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. (2021). An image is worth 16x16 words: Transformers for image recognition at scale. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Elesedy, B. and Zaidi, S. (2021). Provably strict generalisation benefit for equivariant models. In Meila, M. and Zhang, T., editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 2959–2969. PMLR.

Esteves, C. (2020). Theoretical aspects of group equivariant neural networks. *CoRR*, abs/2004.05154.

Esteves, C., Allen-Blanchette, C., Zhou, X., and Daniilidis, K. (2018). Polar transformer networks. In *International Conference on Learning Representations*.

Esteves, C., Makadia, A., and Daniilidis, K. (2020). Spinweighted spherical cnns. In Larochelle, H., Ranzato, M.,

Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Falkner, S., Klein, A., and Hutter, F. (2018). BOHB: Robust and efficient hyperparameter optimization at scale. In *Proceedings of the 35th International Conference on Machine Learning*, pages 1436–1445.

Finzi, M., Stanton, S., Izmailov, P., and Wilson, A. G. (2020). Generalizing convolutional neural networks for equivariance to lie groups on arbitrary continuous data. *CoRR*, abs/2002.12880.

Finzi, M., Welling, M., and Wilson, A. G. (2021). A practical method for constructing equivariant multilayer perceptrons for arbitrary matrix groups. In Meila, M. and Zhang, T., editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 3318–3328. PMLR.

Foret, P., Kleiner, A., Mobahi, H., and Neyshabur, B. (2021). Sharpness-aware minimization for efficiently improving generalization. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Franzen, D. and Wand, M. (2021). General nonlinearities in so(2)-equivariant cnns. In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems*, volume 34, pages 9086–9098. Curran Associates, Inc.

Freeman, W. T. and Adelson, E. H. (1991). The design and use of steerable filters. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13(9):891–906.

Fuchs, F., Worrall, D. E., Fischer, V., and Welling, M. (2020). Se(3)-transformers: 3d roto-translation equivariant attention networks. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Fuchs, F. B., Wagstaff, E., Dauparas, J., and Posner, I. (2021). Iterative se(3)-transformers. In Nielsen, F. and Barbaresco, F., editors, *Geometric Science of Information - 5th International Conference, GSI 2021, Paris, France, July 21-23, 2021, Proceedings*, volume 12829 of *Lecture Notes in Computer Science*, pages 585–595. Springer.

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

Ghosh, R. and Gupta, A. K. (2019). Scale steerable filters for locally scale-invariant convolutional neural networks. *CoRR*, abs/1906.03861.

He, L., Dong, Y., Wang, Y., Tao, D., and Lin, Z. (2021). Gauge equivariant transformer. In Ranzato, M., Beygelzimer, A., Dauphin, Y. N., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 27331–27343.

Hoogeboom, E., Peters, J. W. T., Cohen, T. S., and Welling, M. (2018). Hexaconv. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

Hutchinson, M., Lan, C. L., Zaidi, S., Dupont, E., Teh, Y. W., and Kim, H. (2021). Lietransformer: Equivariant self-attention for lie groups. In Meila, M. and Zhang, T., editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 4533–4543. PMLR.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Bach, F. R. and Blei, D. M., editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 448–456. JMLR.org.

Jaderberg, M., Simonyan, K., Zisserman, A., and Kavukcuoglu, K. (2015). Spatial transformer networks. In *Advances in Neural Information Processing Systems 28*, pages 2017–2025. Curran Associates, Inc.

Jiang, C. M., Huang, J., Kashinath, K., Prabhat, Marcus, P., and Nießner, M. (2019). Spherical cnns on unstructured grids. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Kanazawa, A., Sharma, A., and Jacobs, D. W. (2014). Locally scale-invariant convolutional neural networks. *CoRR*, abs/1412.5104.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In Bengio, Y. and LeCun, Y., editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Kondor, R., Lin, Z., and Trivedi, S. (2018). Clebsch-gordan nets: a fully fourier space spherical convolutional neural network. In Bengio, S., Wallach, H. M., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*, pages 10138–10147.

Kondor, R. and Trivedi, S. (2018). On the generalization of equivariance and convolution in neural networks to the

action of compact groups. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 2752–2760.

Krizhevsky, A. (2009). Learning multiple layers of features from tiny images,. Technical report.

Le, Q. V., Ngiam, J., Chen, Z., hao Chia, D. J., Koh, P. W., and Ng, A. Y. (2010). Tiled convolutional neural networks. In Lafferty, J. D., Williams, C. K. I., Shawe-Taylor, J., Zemel, R. S., and Culotta, A., editors, *Advances in Neural Information Processing Systems 23: 24th Annual Conference on Neural Information Processing Systems 2010. Proceedings of a meeting held 6-9 December 2010, Vancouver, British Columbia, Canada*, pages 1279–1287. Curran Associates, Inc.

LeCun, Y., Bengio, Y., and Hinton, G. E. (2015). Deep learning. *Nature*, 521(7553):436–444.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1990). Handwritten digit recognition with a back-propagation network. In Touretzky, D., editor, *Advances in Neural Information Processing Systems (NIPS 1989)*, volume 2, Denver, CO. Morgan Kaufman.

Lim, S., Kim, I., Kim, T., Kim, C., and Kim, S. (2019). Fast autoaugment. In Wallach, H. M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. B., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 6662–6672.

Lust, J. and Condurache, A. P. (2020a). Gran: An efficient gradient-norm based detector for adversarial and misclassified examples. In *28th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, ESANN 2020, Bruges, Belgium, October 2-4, 2020*, pages 7–12.

Lust, J. and Condurache, A. P. (2020b). A survey on assessing the generalization envelope of deep neural networks at inference time for image classification. *CoRR*, abs/2008.09381.

Marcos, D., Kellenberger, B., Lobry, S., and Tuia, D. (2018). Scale equivariance in cnns with vector fields. *CoRR*, abs/1807.11783.

Marcos, D., Volpi, M., Komodakis, N., and Tuia, D. (2017). Rotation equivariant vector field networks. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 5058–5067. IEEE Computer Society.

Müller, F. and Mertins, A. (2009). Invariant-integration method for robust feature extraction in speaker-independent speech recognition. In *INTERSPEECH 2009, 10th Annual Conference of the International Speech Communication Association, Brighton, United Kingdom, September 6-10, 2009*, pages 2975–2978.

Müller, F. and Mertins, A. (2010). Invariant integration features combined with speaker-adaptation methods. In *INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September 26-30, 2010*, pages 2622–2625.

Müller, F. and Mertins, A. (2011). Contextual invariant-integration features for improved speaker-independent speech recognition. *Speech Communication*, 53(6):830–841.

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. (2011). Reading digits in natural images with unsupervised feature learning. *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*.

Noether, E. (1916). Der endlichkeitssatz der invarianten endlicher gruppen. *Mathematische Annalen*, 77:89–92.

Oyallon, E., Zagoruyko, S., Huang, G., Komodakis, N., Lacoste-Julien, S., Blaschko, M. B., and Belilovsky, E. (2019). Scattering networks for hybrid representation learning. *IEEE Trans. Pattern Anal. Mach. Intell.*, 41(9):2208–2221.

Puny, O., Atzmon, M., Ben-Hamu, H., Smith, E. J., Misra, I., Grover, A., and Lipman, Y. (2021). Frame averaging for invariant and equivariant network design. *CoRR*, abs/2110.03336.

Rath, M. and Condurache, A. P. (2020). Invariant integration in deep convolutional feature space. *28th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, ESANN 2020, Bruges, Belgium, October 2-4, 2020*, pages 103–108.

Rath, M. and Condurache, A. P. (2022). Improving the sample-complexity of deep classification networks with invariant integration. *Proceedings of the 17th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2022)*.

Romero, D. W., Bekkers, E. J., Tomczak, J. M., and Hoogendoorn, M. (2020). Attentive group equivariant convolutional networks. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 8188–8199. PMLR.

Romero, D. W. and Cordonnier, J. (2021). Group equivariant stand-alone self-attention for vision. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Romero, D. W. and Hoogendoorn, M. (2020). Co-attentive equivariant neural networks: Focusing equivariance on transformations co-occurring in data. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Schulz-Mirbach, H. (1992). On the existence of complete invariant feature spaces in pattern recognition. In *Pattern Recognition: Eleventh International Conference 1992*, pages 178 – 182.

Schulz-Mirbach, H. (1994). Algorithms for the construction of invariant features. In *Tagungsband Mustererkennung 1994 (16. DAGM Symposium), Reihe Informatik Xpress, Nr.5*, pages 324–332.

Schulz-Mirbach, H. (1995). Invariant features for gray scale images. In *Mustererkennung 1995, 17. DAGM-Symposium, Bielefeld, 13.-15. September 1995, Proceedings*, pages 1–14.

Shakerinava, M. and Ravanbakhsh, S. (2021). Equivariant networks for pixelized spheres. In Meila, M. and Zhang, T., editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 9477–9488. PMLR.

Sohn, K. and Lee, H. (2012). Learning invariant representations with local transformations. In *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*. icml.cc / Omnipress.

Sosnovik, I., Moskalev, A., and Smeulders, A. W. M. (2021). DISCO: accurate discrete scale convolutions. *CoRR*, abs/2106.02733.

Sosnovik, I., Szmaja, M., and Smeulders, A. W. M. (2020). Scale-equivariant steerable networks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Weiler, M. and Cesa, G. (2019). General e(2)-equivariant steerable cnns. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 14334–14345.

Weiler, M., Geiger, M., Welling, M., Boomsma, W., and Cohen, T. (2018a). 3d steerable cnns: Learning rotationally equivariant features in volumetric data. In Bengio, S., Wallach, H. M., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*, pages 10402–10413.

Weiler, M., Hamprecht, F. A., and Storath, M. (2018b). Learning steerable filters for rotation equivariant cnns. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 849–858.

Worrall, D. E. and Brostow, G. J. (2018). Cubenet: Equivariance to 3d rotation and translation. In *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part V*, pages 585–602.

Worrall, D. E., Garbin, S. J., Turmukhambetov, D., and Brostow, G. J. (2017). Harmonic networks: Deep translation and rotation equivariance. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 7168–7177.

Worrall, D. E. and Welling, M. (2019). Deep scale-spaces: Equivariance over scale. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 7364–7376.

Xu, J., Kim, H., Rainforth, T., and Teh, Y. W. (2021). Group equivariant subsampling. In Ranzato, M., Beygelzimer, A., Dauphin, Y. N., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 5934–5946.

Xu, Y., Xiao, T., Zhang, J., Yang, K., and Zhang, Z. (2014). Scale-invariant convolutional neural networks. *CoRR*, abs/1411.6369.

Zagoruyko, S. and Komodakis, N. (2016). Wide residual networks. In Wilson, R. C., Hancock, E. R., and Smith, W. A. P., editors, *Proceedings of the British Machine Vision Conference 2016, BMVC 2016, York, UK, September 19-22, 2016*. BMVA Press.

Zhu, W., Qiu, Q., Calderbank, A. R., Sapiro, G., and Cheng, X. (2019). Scale-equivariant neural networks with decomposed convolutional filters. *CoRR*, abs/1909.11193.

APPENDIX

A   LIMITED DATA STUDIES WITH CONSTANT PARAMETERS

In this section, we provide limited data graphics for SVHN (see Figure 5) and CIFAR-10 (see Figure 6) when using a constant number of parameters for our multi-stream network. The results on the full dataset are shown in the main paper in Table 3. For the multi-stream networks with constant parameters, we observe a significant increase in the limited data domain and when using the full dataset compared to the baseline methods. While the dual stream performs slightly better on SVHN and STL-10, the triple stream achieves better performance in limited domains and on CIFAR. We conjecture that when the model capacity is limited the E(2)- and the scale-stream are able to learn most object invariances the standard stream would cover. Nevertheless, the standard stream adds the uncovered invariances leading to a slight performance boost. For the multi-stream networks with constant parameters, we used a naive down-scaling of the individual streams, i.e., the number of channels was simply divided by $\sqrt{2}$ or $\sqrt{3}$ per stream, respectively. We also did not further fine-tune hyper-parameters (HPs) for those down-scaled networks. With this straightforward approach, we already outperformed all single stream networks in limited data domains. In the future, further improvements could be achieved by investigating more sophisticated branching methods in order to enable a better allocation of how much capacity the network should spend for each invariant stream.
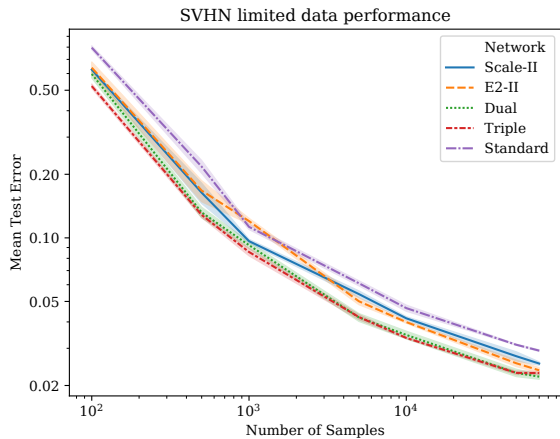
Figure 5: Log Test Error on limited subsets of SVHN with constant parameters.
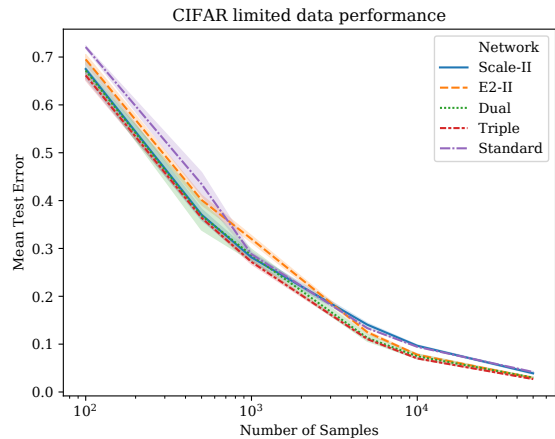
Figure 6: Test Error on limited subsets of CIFAR-10 with constant parameters.

B   ABLATION STUDIES

B.1   Comparison of Multi-Stream Heads

In this section, we compare different versions of our proposed Multi-Stream head to a concatenation head using the triple-stream architecture on STL-10. Map-*Stream* describes which stream the respective outputs are mapped to, i.e., the respective stream uses an non-learnable identity mapping. When using Map-All, all streams use a learnable mapping at the same time. The results are shown in Table 5. Mapping all features to the rotation stream and adding them via a weighted-sum leads to the best results and outperforms a naive concatenation of features along the channel dimension.

B.2   Full End-to-End Training

As a further ablation, we conducted a "true" end-to-end training of our dual stream with random initialization and constant number of parameters on STL-10, where we achieve 7.86 % Test Error (TE) without tuning any stream-specific hyper-parameters (HPs). We expect this to be slightly improvable when optimizing the stream-wise hyper-parameters.

Nevertheless, the true end-to-end training performs worse than our more intricate training procedure (6.38 % TE). We conjecture, that optimizing each stream individually provides a good initialization point for the combination head and

Matthias Rath[⋆,†], **Alexandru Paul Condurache**[⋆,†]

Table 5: Comparison of Multi-Stream heads on STL-10 using the triple stream architecture.

Combination-Type	Test Error [%]
Concat	6.82 ± 0.11
Map-E(2)	$\mathbf{5.90} \pm 0.05$
Map-Scale	7.21 ± 0.06
Map-Std	6.65 ± 0.10
Map-All	6.89 ± 0.06

Table 6: II-DISCO HPs on Scaled-MNIST. Parameters with $\star$ were optimized using BOHB.

HP	II-DISCO WS	II-DISCO Monomials
Batch Size	128	128
Learning Rate $\star$	5e-3	1e-3
Weight Decay $\star$	5e-7	5e-6
BatchNorm Decay $\star$	0.01	5e-4
Dropout Rate $\star$	0.1	0.7

reduces the solution space the optimizer needs to consider. Overall, our learning strategy leads to a further improved sample complexity of the multi-stream architecture.

C IMPLEMENTATION DETAILS & HYPERPARAMETERS

We present details about the network architectures and the HPs needed to reproduce our results. We optimized the HPs using Bayesian optimization with hyperband (BOHB) (Falkner et al., 2018). For the baselines, we used the HPs reported in the respective papers or official implementations. If no validation split was pre-defined, we used a 80-20 train-validation split for the HP optimization. All networks were trained using a single (Scaled-MNIST, SVHN) or two (CIFAR-10, STL-10) NVIDIA GTX-1080 Ti GPUs. For SVHN, CIFAR-10 and STL-10 we used Wide-ResNets (WRNs) as baseline architectures (Zagoruyko and Komodakis, 2016).

For the E(2)-invariant network, we reduced the size of the final trivial representation to the same size used by the Standard- and Scale-WRN. For the original implementation, the final representation is multiplied by $\sqrt{|G|}$ where $|G|$ is the order of the group the last block is equivariant to (Weiler and Cesa, 2019). For the Scale-II WRNs, we added BatchNorm after the II layer to stabilize the gradients back-propagated through the division within the Scale-II.

For the dual- and triple-stream architectures with constant parameters, we naively divided each channel size C by $\lceil \frac{C}{\sqrt{2}} \rceil$ or $\lceil \frac{C}{\sqrt{3}} \rceil$, respectively. We did not further tune HPs for the smaller-sized streams.

We built upon the official code-bases of Sosnovik et al. (Sosnovik et al. (2021), https://github.com/ISosnovik/disco), as well as Weiler and Cesa (Weiler and Cesa (2019), https://github.com/QUVA-Lab/e2cnn), which we both ported to Tensorflow v2.3. We used the models with a constant number of parameters and inserted the II layer to replace the spatial pooling operation. Specifically for DISCO, we had to thoroughly re-create all default settings from PyTorch in Tensorflow in order to recreate the performance. This included variable initializers, batch norm parameters and the Adam optimizer's HPs. Additionally, we had to use a custom-written *grouped convolution* within the scale-equivariant convolution code rather than using *tf.nn.conv2d* with a filter with reduced input-channels which implicitly also calculates a grouped convolution.

C.1 Scaled-MNIST

On Scaled-MNIST, we used the CNN proposed by Sosnovik et al. (2020, 2021) composed of three convolutional and two dense layers with an effective kernel size of 7 and $n_S = 4$ scales. We adapted it by using scale-II with $k = 3$ instead of the spatial pooling layer that is a combination of average and max pooling. We trained our network for 60 epochs using the Adam optimizer (Kingma and Ba, 2015) with step-wise learning rate decay of 0.1 after 20 and 40 epochs, l_2-regularization and data augmentation, where we artificially scaled the input with $s \in [0.5, 2]$. The network details can be found in Table 7 and the used HPs in Table 6.

Table 7: II-DISCO architecture on Scaled-MNIST.

Layer	Output Size	C_{in}	C_{out}	n_S	ReLU	BatchNorm	Dropout
Upsampling	56×56	1	1	-	x	x	x
Scale-LiftConv	56×56	1	32	3	✓	✓	x
MaxPool	28×28	32	32	3	x	x	x
Scale-Conv1	28×28	32	63	3	✓	✓	x
MaxPool	14×14	63	63	3	x	x	x
Scale-Conv2	14×14	63	95	3	✓	✓	x
Scale-MaxProj	14×14	95	95	1	x	x	x
Scale-II	1×1	95	95	-	x	x	x
Dense1	-	95	256	-	✓	✓	✓
Dense2	-	256	10	-	x	x	x

Table 8: HPs used for SVHN experiments. Parameters with * were optimized using BOHB.

HP	E(2)-II	Scale-II
Batch Size	128	128
Learning Rate *	5e-3	0.2
Weight Decay *	5e-3	1e-4
Dropout Rate *	0.5	0.4

C.2 SVHN

On the SVHN dataset, we used a WRN16-4 with pre-activation nonlinearites as baseline architecture. The E(2)-networks are equivariant to flips and $n_r = 8$ rotations for each convolutional layer. The scale-convolutions use $n_S = 3$ scales. We used E(2)-II with $k = 3$, $n_r = 8$ angles and $n_F = 2$ flips using bi-linear interpolation where necessary. We applied Scale-II with $k = 3$. We followed the training approach by Zagoruyko and Komodakis (2016). We used an SGD optimizer with Momentum 0.9, trained for 160 epochs and reduced the learning rate by 0.1 after 80 and 120 epochs. We did not apply any data augmentation. The detailed architecture of the E(2)-II-WRN16-4 is shown in Table 10, the architecture of the Scale-II-WRN16-4 in Table 11. The used HPs are shown in Table 8.

For the classification head combining several invariant streams, we froze all trained convolutional and II layers and combined the individual streams via a learnable mapping and a weighted sum. We then trained those layer as well as the final dense layers. We used the same training settings as above, but divided the number of training epochs as well as all epoch-dependent HPs by 4. The HPs of the refinement training are listed in Table 9.

C.3 CIFAR-10

On CIFAR-10, we used a WRN28-10 as the baseline network. The E(2)-networks are equivariant to flips and $n_r = 8$ rotations for the first residual block, $n_r = 4$ rotations for the second and third residual blocks. The scale-convolutions use $n_S = 3$ scales. We used E(2)-II with $k = 3$, $n_r = 4$ angles and $n_F = 2$ flips. We applied Scale-II with $k = 3$. We again followed the training approach by Zagoruyko and Komodakis (2016). We used an SGD optimizer with Momentum 0.9, trained for 200 epochs and reduced the learning rate by 0.2 after 60, 120 and 160 epochs. We used data augmentation with random pads-and-crops as well as random flips. The detailed architecture of the E(2)-II-WRN28-10 is shown in Table 14, the architecture of the Scale-II-WRN28-10 in Table 15. The HPs can be found in Table 12. We used the same approach for the combination heads as for SVHN where we divided all epochs by 4. The HPs of the re-training are shown in Table 13.

C.4 STL-10

On STL-10, we used a WRN16-8 as the baseline network. The E(2)-networks are equivariant to flips as well as $n_r = 8$ rotations for the first residual block, $n_r = 4$ rotations for the second and $n_r = 1$ rotations for the third residual block. The scale-convolutions use $n_S = 3$ scales. We used E(2)-II with $k = 3$, $n_r = 1$ angles and $n_F = 2$ flips and Scale-II with $k = 3$. We used an SGD optimizer with Nesterov Momentum 0.9, trained for 1000 epochs and reduced the learning rate by 0.2 after 300, 400, 600 and 800 epochs. We used data augmentation with random pads-and-crops, flips and Cutout (Devries and Taylor, 2017). The detailed architecture of the E(2)-II-WRN16-8 is shown in Table 18, the architecture of the Scale-II-WRN16-8 in Table 19. We adapted the official implementation of the Scale-networks to use a stride of 2 instead of 1 in the initial residual block – as done by Weiler and Cesa (2019); Devries and Taylor (2017). The used HPs can be found

Matthias Rath[⋆,†]**, Alexandru Paul Condurache**[⋆,†]

Table 9: Classification head HPs on SVHN.

HP	Dual	Triple
Batch Size	128	128
Learning Rate	0.01	0.01
Weight Decay	1e-4	1e-4

Table 10: E(2)-II-WRN16-4 architecture for SVHN. Each equivariant residual block consists of two consecutive convolution layers (e.g. E(2)-Conv1.1 and E(2)-Conv1.2) and a shortcut connection with a 1x1-convolution whenever the output size changes.

Layer	Output Size	C_{in}	C_{out}	n_r	n_F	ReLU	BN	Dropout
E(2)-LiftConv	32×32	3	4	8	2	✓	✓	x
E(2)-Conv1.1	32×32	4	16	8	2	✓	✓	x
E(2)-Conv1.2	32×32	16	16	8	2	✓	✓	✓
E(2)-Conv1.3	32×32	16	16	8	2	✓	✓	x
E(2)-Conv1.4	32×32	16	16	8	2	✓	✓	✓
E(2)-Conv2.1	16×16	16	32	8	2	✓	✓	x
E(2)-Conv2.2	16×16	32	32	8	2	✓	✓	✓
E(2)-Conv2.3	16×16	32	32	8	2	✓	✓	x
E(2)-Conv2.4	16×16	32	32	8	2	✓	✓	✓
E(2)-Conv3.1	8×8	32	64	8	2	✓	✓	x
E(2)-Conv3.2	8×8	64	64	8	2	✓	✓	✓
E(2)-Conv3.3	8×8	64	64	8	2	✓	✓	x
E(2)-Conv3.4	8×8	64	256	1	1	✓	✓	✓
E(2)-II	-	256	256	-	-	x	x	x
Dense	-	256	10	-	-	x	x	x

in Table 16. We used the same approach for the combination heads as for SVHN and CIFAR-10, but this time divided all epochs by 10. The HPs of the re-training can be found in Table 17.

D   BROADER SOCIETAL IMPACT

Our proposed method increases the sample-efficiency of DNNs for classification tasks. It can thus be used to train supervised models when training data is scarce or expensive to label, i.e., in fields such as medical imaging or autonomous driving. Since the methods proposed in this paper are quite general and not bound to a specific application, they can be used for any type of classification network processing images. This includes potentially harmful applications. Even without harmful intents, training on biased datasets as well as misclassifications can lead to unintended negative consequences such as a wrong medical treatment.

The authors firmly renounce using our proposed methods with any harmful intents. When applying our methods, it is inevitable to monitor the decisions made by the network specifically based on ethical standards, such that the classifier does not decide in a harmfully biased way. Impactful decisions based on the output of DNNs, e.g. deciding on a medication or driving autonomously, need to be carefully supervised by human experts and/or redundant systems where appropriate.

E   LICENSES

We implemented all networks using Tensorflow v2.3 which is licensed under Apache 2.0 (Abadi et al., 2015). We built upon the code-bases of Sosnovik et al. (2021) which is under MIT license and Weiler and Cesa (2019) distributed under BSD Clear license. We ported both frameworks from PyTorch to Tensorflow v2.3. The used datasets Scaled-MNIST (Sohn and Lee, 2012), SVHN (Netzer et al., 2011), CIFAR-10 (Krizhevsky, 2009) and STL-10 (Coates et al., 2011) are not under license.

F   AMOUNT OF COMPUTE

For our experiments on Scaled-MNIST and SVHN, we used a single Nvidia 1080 GTX Ti. On CIFAR-10 and STL-10, we used two NVIDIA 1080 GTX Tis. For Scaled-MNIST, we performed 6 runs à 8 sizes, 2 architectures, which equates to 96

Table 11: Scale-II-WRN16-4 architecture for SVHN. Each equivariant residual block consists of two consecutive convolution layers (e.g. Scale-Conv1.1 and Scale-Conv1.2) and a shortcut connection with a 1x1-convolution whenever the output size changes.

Layer	Output Size	C_{in}	C_{out}	n_S	ReLU	BN	Dropout
Scale-LiftConv	32×32	3	16	3	✓	✓	x
Scale-Conv1.1	32×32	16	64	3	✓	✓	x
Scale-Conv1.2	32×32	64	64	3	✓	✓	✓
Scale-Conv1.3	32×32	64	64	3	✓	✓	x
Scale-Conv1.4	32×32	64	64	3	✓	✓	✓
Scale-Conv2.1	16×16	64	128	3	✓	✓	x
Scale-Conv2.2	16×16	128	128	3	✓	✓	✓
Scale-Conv2.3	16×16	128	128	3	✓	✓	x
Scale-Conv2.4	16×16	128	128	3	✓	✓	✓
Scale-Conv3.1	8×8	128	256	3	✓	✓	x
Scale-Conv3.2	8×8	256	256	3	✓	✓	✓
Scale-Conv3.3	8×8	256	256	3	✓	✓	x
Scale-Conv3.4	8×8	256	256	3	✓	✓	✓
Scale-MaxProj	8×8	256	256	1	x	x	x
Scale-II	1×1	256	256	-	x	✓	x
Dense	-	256	10	-	x	x	x

Table 12: HPs on CIFAR-10. Parameters with * were optimized using BOHB.

HP	E(2)-II	Scale-II
Batch Size	96	128
Learning Rate *	5e-3	0.1
Weight Decay *	5e-3	5e-4
Dropout Rate *	0.1	0.2

runs. In total, the GPU train time took ≈ 50 hours. For SVHN, we evaluated $3 \cdot 2 + 7 \cdot 3 \cdot 7 = 153$ runs with ≈ 1800 hours train time, for CIFAR-10 $3 \cdot 2 + 6 \cdot 3 \cdot 7 = 132$ runs with ≈ 3600 hours train time and $1 \cdot 3 \cdot 9 = 27$ runs with ≈ 650 hours train time for STL-10. We took the average train time for all architectures on each dataset to calculate those numbers. In total, we estimate the GPU time for the main results to $50 + 1800 + 2 \cdot 3600 + 2 \cdot 650 = 10350$ hours.

Matthias Rath[*,†]**, Alexandru Paul Condurache**[*,†]

Table 13: Classification head HPs on CIFAR-10.

HP	Dual	Triple
Batch Size	128	128
Learning Rate	0.01	0.01
Weight Decay	1e-3	1e-3

Table 14: E(2)-II-WRN28-10 architecture for CIFAR-10. Each equivariant residual block consists of two consecutive convolution layers (e.g. E(2)-Conv1.1 and E(2)-Conv1.2) and a shortcut connection with a 1x1-convolution whenever the output size changes.

Layer	Output Size	C_{in}	C_{out}	n_r	n_F	ReLU	BN	Dropout
E(2)-LiftConv	32×32	3	4	8	2	✓	✓	x
E(2)-Conv1.1	32×32	4	40	8	2	✓	✓	x
E(2)-Conv1.2	32×32	40	40	8	2	✓	✓	✓
E(2)-Conv1.3	32×32	40	40	8	2	✓	✓	x
E(2)-Conv1.4	32×32	40	40	8	2	✓	✓	✓
E(2)-Conv1.5	32×32	40	40	8	2	✓	✓	x
E(2)-Conv1.6	32×32	40	40	8	2	✓	✓	✓
E(2)-Conv1.7	32×32	40	40	8	2	✓	✓	x
E(2)-Conv1.8	32×32	40	40	8	2	✓	✓	✓
E(2)-Conv2.1	16×16	40	113	4	2	✓	✓	x
E(2)-Conv2.2	16×16	113	113	4	2	✓	✓	✓
E(2)-Conv2.3	16×16	113	113	4	2	✓	✓	x
E(2)-Conv2.4	16×16	113	113	4	2	✓	✓	✓
E(2)-Conv2.5	16×16	113	113	4	2	✓	✓	x
E(2)-Conv2.6	16×16	113	113	4	2	✓	✓	✓
E(2)-Conv2.7	16×16	113	113	4	2	✓	✓	x
E(2)-Conv2.8	16×16	113	113	4	2	✓	✓	✓
E(2)-Conv3.1	8×8	113	226	4	2	✓	✓	x
E(2)-Conv3.2	8×8	226	226	4	2	✓	✓	✓
E(2)-Conv3.3	8×8	226	226	4	2	✓	✓	x
E(2)-Conv3.4	8×8	226	226	4	2	✓	✓	✓
E(2)-Conv3.5	8×8	226	226	4	2	✓	✓	x
E(2)-Conv3.6	8×8	226	226	4	2	✓	✓	✓
E(2)-Conv3.7	8×8	226	226	4	2	✓	✓	x
E(2)-Conv3.8	8×8	226	640	1	1	✓	✓	✓
E(2)-II	-	640	640	-	-	x	x	x
Dense	-	640	10	-	-	x	x	x

Table 15: Scale-II-WRN28-10 architecture for CIFAR-10. Each equivariant residual block consists of two consecutive convolution layers (e.g. Scale-Conv1.1 and Scale-Conv1.2) and a shortcut connection with a 1x1-convolution whenever the output size changes.

Layer	Output Size	C_{in}	C_{out}	n_S	ReLU	BN	Dropout
Scale-LiftConv	32×32	3	16	3	✓	✓	x
Scale-Conv1.1	32×32	16	160	3	✓	✓	x
Scale-Conv1.2	32×32	160	160	3	✓	✓	✓
Scale-Conv1.3	32×32	160	160	3	✓	✓	x
Scale-Conv1.4	32×32	160	160	3	✓	✓	✓
Scale-Conv1.5	32×32	160	160	3	✓	✓	x
Scale-Conv1.6	32×32	160	160	3	✓	✓	✓
Scale-Conv1.7	32×32	160	160	3	✓	✓	x
Scale-Conv1.8	32×32	160	160	3	✓	✓	✓
Scale-Conv2.1	16×16	160	320	3	✓	✓	x
Scale-Conv2.2	16×16	320	320	3	✓	✓	✓
Scale-Conv2.3	16×16	320	320	3	✓	✓	x
Scale-Conv2.4	16×16	320	320	3	✓	✓	✓
Scale-Conv2.5	16×16	320	320	3	✓	✓	x
Scale-Conv2.6	16×16	320	320	3	✓	✓	✓
Scale-Conv2.7	16×16	320	320	3	✓	✓	x
Scale-Conv2.8	16×16	320	320	3	✓	✓	✓
Scale-Conv3.1	8×8	320	640	3	✓	✓	x
Scale-Conv3.2	8×8	640	640	3	✓	✓	✓
Scale-Conv3.3	8×8	640	640	3	✓	✓	x
Scale-Conv3.4	8×8	640	640	3	✓	✓	✓
Scale-Conv3.5	8×8	640	640	3	✓	✓	x
Scale-Conv3.6	8×8	640	640	3	✓	✓	✓
Scale-Conv3.7	8×8	640	640	3	✓	✓	x
Scale-Conv3.8	8×8	640	640	3	✓	✓	✓
Scale-MaxProj	8×8	640	640	1	x	x	x
Scale-II	1×1	640	640	-	x	✓	x
Dense	-	640	10	-	x	x	x

Table 16: HPs on STL-10. Parameters with $\star$ were optimized using BOHB.

HP	E(2)-II	Scale-II
Batch Size	96	96
Learning Rate $\star$	2e-3	0.02
Weight Decay $\star$	1e-2	1e-3
Dropout Rate $\star$	0.1	0.25

Table 17: Classification head HPs on STL-10.

HP	Dual	Triple
Batch Size	128	128
Learning Rate	0.01	0.01
Weight Decay	1e-4	1e-4

Table 18: E(2)-II-WRN16-8 architecture for STL-10. Each equivariant residual block consists of two consecutive convolution layers (e.g. E(2)-Conv1.1 and E(2)-Conv1.2) and a shortcut connection with a 1x1-convolution whenever the output size changes.

Layer	Output Size	C_{in}	C_{out}	n_r	n_F	ReLU	BN	Dropout
E(2)-LiftConv	96×96	3	4	8	2	✓	✓	x
E(2)-Conv1.1	48×48	4	32	8	2	✓	✓	x
E(2)-Conv1.2	48×48	32	32	8	2	✓	✓	✓
E(2)-Conv1.3	48×48	32	32	8	2	✓	✓	x
E(2)-Conv1.4	48×48	32	32	8	2	✓	✓	✓
E(2)-Conv2.1	24×24	32	90	4	2	✓	✓	x
E(2)-Conv2.2	24×24	90	90	4	2	✓	✓	✓
E(2)-Conv2.3	24×24	90	90	4	2	✓	✓	x
E(2)-Conv2.4	24×24	90	90	4	2	✓	✓	✓
E(2)-Conv3.1	12×12	90	362	1	2	✓	✓	x
E(2)-Conv3.2	12×12	362	362	1	2	✓	✓	✓
E(2)-Conv3.3	12×12	362	362	1	2	✓	✓	x
E(2)-Conv3.4	12×12	362	512	1	1	✓	✓	✓
E(2)-II	-	512	512	-	-	x	x	x
Dense	-	512	10	-	-	x	x	x

Table 19: Scale-II-WRN16-8 architecture for STL-10. Each equivariant residual block consists of two consecutive convolution layers (e.g. Scale-Conv1.1 and Scale-Conv1.2) and a shortcut connection with a 1x1-convolution whenever the output size changes.

Layer	Output Size	C_{in}	C_{out}	n_S	ReLU	BN	Dropout
Scale-LiftConv	96×96	3	32	3	✓	✓	x
Scale-Conv1.1	48×48	32	128	3	✓	✓	x
Scale-Conv1.2	48×48	128	128	3	✓	✓	✓
Scale-Conv1.3	48×48	128	128	3	✓	✓	x
Scale-Conv1.4	48×48	128	128	3	✓	✓	✓
Scale-Conv2.1	24×24	128	256	3	✓	✓	x
Scale-Conv2.2	24×24	256	256	3	✓	✓	✓
Scale-Conv2.3	24×24	256	256	3	✓	✓	x
Scale-Conv2.4	24×24	256	256	3	✓	✓	✓
Scale-Conv3.1	12×12	256	512	3	✓	✓	x
Scale-Conv3.2	12×12	512	512	3	✓	✓	✓
Scale-Conv3.3	12×12	512	512	3	✓	✓	x
Scale-Conv3.4	12×12	512	512	3	✓	✓	✓
Scale-MaxProj	12×12	512	512	1	x	x	x
Scale-II	1×1	512	512	-	x	✓	x
Dense	-	512	10	-	x	x	x