POZNAŃ UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTING AND TELECOMMUNICATIONS
INSTITUTE OF COMPUTING

MASTER THESIS

Equivariant neural networks for image recognition

Author:
Kamil Burdziński

Tutor:
dr hab. inż. Wojciech Kotłowski

Contents

1 Introduction

1.1 Background

The last couple of years saw a rapidly surging interest in neural networks invariant to certain groups of transformations. This property is very much sought after because under mild assumptions it guarantees the output of the network won't change when transformed according to chosen transformations. Related desirable quality of neural networks is equivariance which on the other hand causes output to change in the same way the input changes. Growing body of work presents wide variety of techniques applied to ensure these properties.
However so far almost all of published papers focused exclusively on geometric symmetries of images, e.g. rotations, scaling or reflection. Transforms related to lightning of the image, like contrast or brightness have been largely ignored.

1.2 Objectives

In this work we address the following problems:

1. Is constructing neural networks either equivariant or invariant to lightning symmetries feasible? If so then how do they compare to networks lacking these properties?

2. What's the degree of equivariance of such models? Is the equivariance exact or does inevitable discretization impose significant error? Is the magnitude of error more or less the same as for the geometric transformations?

In order to answer these questions, we carry out the following tasks:

1. Possibly extending notions of contrast, brightness, gamma correction and color balance from images to tensors of arbitrary dimensionality.

2. Constructing neural network layers invariant to various lightning symmetries.

3. Adapting existing architectures to ensure equivariance to some or possibly all of mentioned lightning transformations.

4. Comparing constructed models with control group on image classification tasks.

5. Estimating numerically degree of invariance and equivariance of constructed models.

2 Theoretical background

2.1 Group theory

Modern literature regarding equivariant neural networks is based heavily on group theory [13]. We start with review of its basic notions. **Group** G is a set together with binary operation $(G, \bullet)$ satisfying following axioms:

1. Closure: for any $g, h \in G$, $g \bullet h \in G$
2. Associativity: $g \bullet (h \bullet j) = (g \bullet h) \bullet j$ for any $g, h, j \in G$
3. Identity: there exists a neutral element e such that $e \bullet g = g \bullet e = g$ for any $g \in G$
4. Inverse element: for every $g \in G$ exists element $h \in G$ such that $g \bullet h = h \bullet g = e$. The element inverse to g is usually written as g^{-1}

Group G might also posses additional structure. A **topological group** is a group together with a topology on the underlying set, such that its binary operation:

$$\bullet : G \times G \to G, \quad (g, h) \mapsto g \bullet h$$

and inversion map:

$$^{-1} : G \to G, \quad g \mapsto g^{-1}$$

are continuous. Additionally if the underlying topological structure is some smooth n-manifold and both maps are smooth, the group is called **Lie group** [20]. Familiar examples of Lie groups include translation group $(\mathbb{R}^n, +)$ - set of real vectors with addition or $(GL(n, \mathbb{R}), \cdot)$ - set of real $n \times n$ matrices of nonzero determinant with multiplication.
Every Lie group G has an associated structure $\mathfrak{g}$ called **Lie algebra**. For our purposes we can treat it as a real vector space $\mathbb{R}^n$ where n equals the dimension of G as manifold. The connection between group and its Lie algebra is established through two maps. **Log** is a function from G to $\mathfrak{g}$ and **Exp** is its inverse mapping elements of $\mathfrak{g}$ to elements of G.

Just as we can form functions between sets assigning elements of one set to the elements of other, it's possible to construct functions between groups. Typically we are interested in functions preserving to some extent the group structure, that is of the form.

$$f : G \to H, \qquad f(x \bullet y) = f(x) * f(y) \tag{1}$$

where $\bullet$ is the binary operation in group G and $*$ is the binary operation in group H. Such functions are called **group homomorphisms**. Bijective homomorphism is called **isomorphism**.

In this work we are mainly interested in groups in context of their actions on 3D tensors. Given a group G and set X, **group action** of G on X is defined as a function

$$\theta : G \times X \to X \tag{2}$$

(with $\theta(g, x)$ often written as $g \cdot x$) satisfying following axioms:

1. Identity: $e \cdot x = x$ for every $x \in X$, where e is the neutral element of G
2. Compatibility: $g \cdot (h \cdot x) = (g \bullet h) \cdot x$ for all g and h in G and x in X

The simplest possible group action is the trivial group action, that is $g \cdot x = x$ for any g and x. Familiar nontrivial examples include isometries of $\mathbb{R}^n$, e.g. translations - action of $(\mathbb{R}^n, +)$ given by $v \cdot x = v + x$ or rotations - action of $SO(n, \mathbb{R})$ given by $r \cdot x = Rx$ where R is matrix representation of rotation and multiplication on the right is matrix-vector multiplication.
Now suppose we are given some function (not necessarily homomorphism) $f : G \to \mathbb{R}^n$. We can define natural action of G on set of such functions by:

$$(g \cdot f)(x) = f(g^{-1} \bullet x) \tag{3}$$

It's a group action because:

$$(e \cdot f)(x) = f(e^{-1} \bullet x) = f(e \bullet x) = f(x)$$

and

$$((g \bullet h) \cdot f)(x) = f((g \bullet h)^{-1} \bullet x) = f(h^{-1} \bullet g^{-1} \bullet x) = (h \cdot f)(g^{-1} \bullet x) = (g \cdot (h \cdot f))(x)$$

2.2 Equivariant and invariant functions

Key properties of neural networks in regard to group actions are equivariance and invariance. These can be abstractly defined as follows:

Let group G act on sets X and Y. We call function $f : X \to Y$ **equivariant** with respect to action of G if for all $g \in G$ and $x \in X$

$$f(g \cdot x) = g \cdot f(x) \tag{4}$$

In turn, we call f **invariant** with respect to action of G if

$$f(g \cdot x) = f(x) \tag{5}$$

In fact invariance is a special case of equivariance where action of G on Y is trivial. It's however conceptually cleaner to make distinction between both properties. Intuitively equivariance tells us that when input changes, the output changes accordingly and invariance – that action of G has no effect on output.

If we want to construct a neural network $\mathcal{N}$ equivariant to some specified group action, we need to make sure each of its layers is equivariant. If we treat $\mathcal{N}$ as a series

of functions $f_0, f_1, ..., f_n$ then

$$\begin{aligned}\mathcal{N}(g \cdot x) &= f_n \circ f_{n-1} \circ \cdots \circ f_1 \circ f_0(g \cdot x) \\ &= f_n \circ f_{n-1} \circ \cdots \circ f_1 \circ (g \cdot f_0(x)) \\ &= \cdots \\ &= g \cdot f_n \circ f_{n-1} \circ \cdots \circ f_1 \circ f_0(x) \\ &= g \cdot \mathcal{N}(x)\end{aligned}$$

where $g \cdot x$ is the action of g on x. Often we don't want the network to be fully equivariant; the desired extent of equivariance depends on the task. For problems of image-to-image type like segmentation full equivariance to transformations like rotation or translation is desirable. However for problems where the input data gets more and more "squished" in consecutive layers, we need to break equivariance at some point. Like for example in classification, where final layers are often fully connected and not equivariant.

The easiest way to assert the network is invariant to some transformation is making its very first layer f_0 invariant:

$$\begin{aligned}\mathcal{N}(g \cdot x) &= f_n \circ f_{n-1} \circ \cdots \circ f_1 \circ f_0(g \cdot x) \\ &= f_n \circ f_{n-1} \circ \cdots \circ f_1 \circ f_0(x) \\ &= \mathcal{N}(x)\end{aligned}$$

Equivalently [17] one can make the couple first layers equivariant and a single layer following them invariant. For example if f_0 is equivariant and f_1 invariant we get:

$$\begin{aligned}\mathcal{N}(g \cdot x) &= f_n \circ f_{n-1} \circ \cdots \circ f_1 \circ f_0(g \cdot x) \\ &= f_n \circ f_{n-1} \circ \cdots \circ f_1 \circ (g \cdot f_0(x)) \\ &= f_n \circ f_{n-1} \circ \cdots \circ f_1 \circ f_0(x) \\ &= \mathcal{N}(x)\end{aligned}$$

and the network is again invariant. Modern literature is often not so clear regarding terminology and networks of the above type are sometimes also called equivariant.

2.3 Equivariant neural networks

2.3.1 GCNN

The primary example of equivariant neural networks are convolutional neural networks which are (roughly) equivariant to translations. The four basic components of CNN are convolutional layers, pooling layers, normalization layers and activation functions. These common building blocks are approximately equivariant to translations:

- Convolutional layer is equivariant outside of image's borders. The border effects can to some extent be fixed by padding image with values present in the border or their averages.

- MaxPooling layer is locally invariant – that is small translations don't affect the layer's output, but translations which are multiplicities of window's size cause equivariant behaviour. In average pooling local behaviour is more complicated, but they are also equivariant to window-sized translations.
- Usual normalization schemes like batch or instance normalization depend on global statistics like mean and standard deviation which change slightly during translation so they are not exactly equivariant. However given proper padding method these changes can be made very small if translation is not too big.
- Activation functions are applied pointwise so, they are perfectly equivariant outside borders.

Of course the most crucial parts of CNN are convolution layers. At each layer CNN takes as input 3D tensor which can be thought of as a stack of functions $f : \mathbb{Z}^2 \to \mathbb{R}^C$ where C is number of channels. Convolutional layer correlates it with a set of filters ψ^i and each filter can also be thought of as a stack of functions $\psi^i : \mathbb{Z}^2 \to \mathbb{R}^C$:

$$(f \star \psi^i)(x) = \sum_{y \in \mathbb{Z}^2} \sum_{c=1}^{C} f_c(y) \psi_c^i(y - x) \tag{6}$$

By generalizing convolutions and architecture of CNNs we would like to construct models equivariant or invariant to some specified set of image transformations. One of the earliest works on this problem appeared in [11]. There the so called *Group Equivariant Convolutional Neural Networks* (G-CNNs) equivariant to possibly any discrete set of transformations were proposed. Their main idea is generalizing the convolution layer by lifting tensors the network operates on to higher dimensions. Just like traditionally the last two dimensions encode height and width, these new dimensions can encode information about additional transformations. Higher dimensional tensors are then thought of as stacks of functions $F : G \to \mathbb{R}^C$, where G is the group of transformations. G is always a group of translations plus some set of other symmetries, e.g. rotations or scaling. Pictures 1 and 2 show feature maps for $p4$ – group of translations and rotations and for $p4m$ – group of translations, rotations and reflections. Convolutional filters of respective GCNNs are shaped in the same way but their heights and widths are smaller (typically 3×3 or 5×5 just like in normal CNNs). On architectural level transition from CNN to GCNN is achieved by replacing the first convolutional layer with lift operation and the subsequent convolutions with group convolutional layers.

Lift operation is a particular kind of convolution also proposed in [11] that transforms usual function stack f on $\mathbb{Z}^2$ into function stack F on group G. Computationally it takes as input usual 3d tensor (channel, height, width) and produces N-dimensional tensor with $N > 3$:

$$\mathit{Lift}(f)(g) = F(g) = (f \star \psi^i)(g) = \sum_{y \in \mathbb{Z}^2} \sum_{c=1}^{C} f_c(y) \psi_c^i(g^{-1}y) \tag{7}$$

Note that we are iterating y over $\mathbb{Z}^2$ but since $\mathbb{Z}^2$ is the translation part (subgroup) of G, y is also an element of G and so is $g^{-1}y$. As a consequence filters have values defined

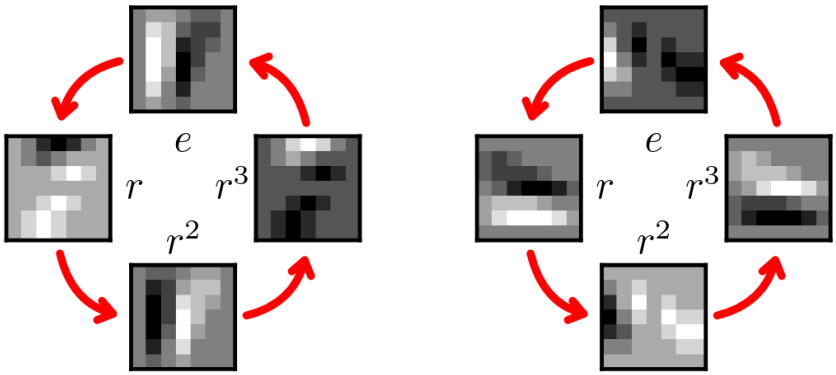


Figure 1: On the left: schematic representation of feature map F flowing through p4-equivariant GCNN. Computationally F can be thought of as tensor of shape $4 \times H \times W$, where each of 4 submaps represents one of rotations by $0°$, $90°$, $180°$ or $270°$. On the right, the same feature map rotated by $90°$ counterclockwise – $r \cdot F$ in the sense of equation 3. Note that every submap follows the red arrow as well as gets rotated. This stems from the structure of $p4$ group. Taken from [11].

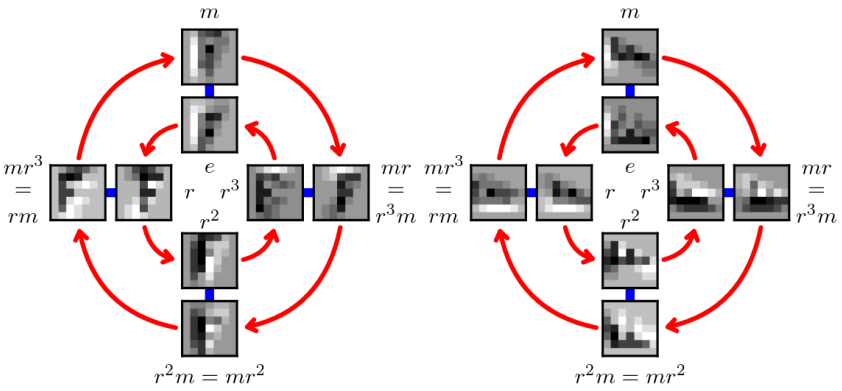


Figure 2: Feature map of p4m-equivariant GCNN and its rotation by r. Taken from [11].

on the group. Once the input tensor is lifted, it assumes values on G and in consecutive convolutional layers it undergoes **group convolutions**:

$$F(g) = (f \star \psi^i)(g) = \sum_{y \in G} \sum_{c=1}^{C} f_c(y) \psi_c^i(g^{-1}y) \tag{8}$$

Near the end of the network signals are back again projected to original dimensionality by a single **projection layer**. This layer acts like pooling but on the previously added group dimension and not the spatial dimensions. Depending on implementation it computes either sum, mean or maximal value. In this work we use max.

While in equation 6 computation is done by sliding y on $\mathbb{Z}^2$, in equation 8 it's replaced by more general iteration on all group elements. If we replace G by $\mathbb{Z}^2$ and set $g = x$ we obtain equation 6 back since in $\mathbb{Z}^2$, x^{-1} is just $-x$. We can prove operator

defined by equation 8 is equivariant to action of any element h of G:

$$\begin{aligned}((h \cdot f) \star \psi)(g) &= \sum_{y \in G} \sum_{c=1}^{C} f_c(h^{-1}y)\psi_c(g^{-1}y) \\ &= \sum_{hy \in G} \sum_{c=1}^{C} f_c(y)\psi_c(g^{-1}hy) \\ &= \sum_{hy \in G} \sum_{c=1}^{C} f_c(y)\psi_c((h^{-1}g)^{-1}y) \\ &= (f \star \psi)(h^{-1}g) \\ &= (h \cdot (f \star \psi))(g)\end{aligned}$$

Iterating over hy is the same as iterating over y as both mean simply summation over all of elements of G. The proof goes similarly for equation 7.

2.3.2 Related work

The ideas contained in [11] were further generalized to new possible groups of transformations, domains and neural network architectures. [12, 18, 14] describe convolutional layers equivariant to the group of 3D rotations $SO(3)$. They make heavy use of spherical harmonics for parametrization and speedup of computation. Equivariance to $SO(3)$ is beneficial in any tasks regarding real world environments like depth estimation or 3D objects classification [14]. It helps also at prediction of properties of molecules described in terms of 3D-coordinates [15]. In less obvious way $SO(3)$-equivariant GCNNs are used to process data whose natural domain is spherical, e.g. Earth weather data or images from drone camera [12].

Another particularly well-researched group of transformations is scaling. [25] formalizes problem in terms of semigroup theory and uses gaussian kernels together with downsampling to eliminate higher frequencies which would otherwise might cause artifacts in image. [7] constructs general architecture equivariant to any (Lie) group of transformations specified by programmer. The scaling is then represented abstractly as the group $(\mathbb{R}_+, \cdot)$. [24] uses fixed base of steerable filters and constructs actual filters as linear combinations of these.

[7, 15, 17] propose general frameworks for GCNNs equivariant to any Lie group. In order to parametrize the manifold and make sense of distances on it, they use Lie algebras and exponential maps. This way instead of dealing with often unintuitive structure of Lie groups, we can compute everything with familiar operations on vectors. For example instead of computing distance between two rotation matrices directly, it suffices to compute the difference between the vectors corresponding to these matrices in the Lie algebra. [15] and [7] are described more closely in the next section 2.3.3.

[17, 23, 22] add attention mechanisms and ensure they are also equivariant with respect to chosen group of transformations. [17] extends ideas from [15] and is able to handle arbitrary Lie groups. [23, 22] operate on discrete (finite) groups of transformations and use circulant attention matrices to make them equivariant to rotations. All of these architectures require unfortunately a lot of memory – one of models constructed

for CIFAR10 dataset in [23] uses 72GBs of RAM.

All the works mentioned so far specify explicitly and precisely the transformations the network is supposed to be equivariant to. In [8] on the other hand kind of transformation is stated, but the network has to learn its extent on its own during usual training procedure, e.g. model should be equivariant to rotations from $[-\theta; \theta]$ interval but parameter θ is to be learned from data alone.

The overall framework of GCNNs was summarized in [18] and [10] in terms of group theory, group actions, fourier analysis and representation theory.

2.3.3 Lie GCNNs

The models described in [15] and [7] operate in similar fashion. First they lift the input tensor, so that it gains a new dimension representing chosen group of transformations, just like in [11]. Then the signal goes through a series of typical neural components – convolutions, pooling layers, activation functions and normalization layers. The last three types of layers are mostly the same as in regular CNNs. While architecture of NN in [7] resembles traditional VGG, [15] constructs ResNet-like models. Given equivariant components from [7], going from VGG to ResNet is purely a matter of engineering and lining up the dimensions – if both normal and residual connections in ResNet are equivariant, then so is their sum. Mathematical formulation of convolutions is also essentially the same. Just like in [11], after the tensor is lifted, it takes values on the group and convolutional layers compute the convolution with respect to the group:

$$(Kf)(g) = \int_G K(g^{-1}h)f(h)d\mu(h) \tag{9}$$

where G is the group, K is convolutional kernel, f is signal and $d\mu$ is the Haar measure of group G which makes the integration possible. However given the discrete nature of input signals, this integral also needs to be discretized and turned into a sum. This is achieved by sampling elements from G according to its Haar measure and computing sum over them:

$$(Kf)(g) = \frac{1}{N}\sum_{i=1}^{N} K(g^{-1}h_i)f(h_i) \tag{10}$$

where N is number of sampled elements. The lifting itself is done in pretty much the same way, except that group G in this case is just square grid $\mathbb{Z}^2$.

All of these elements are common to both papers. Where the works differ significantly is the method by which convolutional kernels are constructed. Equation 9 implies that kernels must have values defined for all elements of group G. This requirement doesn't hold in practice due to discretization so we could potentially model kernels as tensors just like in regular GCNNs. Such representation however causes increase in computation proportional to dimensionality (as manifold) of the transformation group. And within single dimension it's proportional to number of sample points. For this reason both papers model kernels as highly parametrized continuous functions. [15] takes conventional approach of deep learning – every kernel is small neural network. It takes as input a vector in Lie algebra of G and gives values of an associated elements of G on output. [7] on the other hand uses fixed basis of functions

– so called B-splines. B-splines are smooth, strictly positive symmetric functions defined on all of $\mathbb{R}$. They also take as input vectors from lie algebra and compute output as weighted sum of differently centered individual B-splines. The weights are learnable parameters in this case. The exact form of B-splines doesn't matter that much, though they very closely resemble the gaussian function. More precisely if we are given element g for which we want to compute value of the kernel K, we do so by equation

$$K(g) = \sum_{i=1}^{N} c_i B\left(\frac{\text{Log } g_i^{-1} g}{S}\right) \tag{11}$$

where Log is the logarithmic map from G to its Lie algebra and c_i are learnable parameters. Each B-spline is centered around some element g_i and is scaled by scalar constant S. Note that Log $g_i^{-1}g$ is really a vector $\mathbf{x}$ in $\mathbb{R}^n$, whereas B-splines are defined for real numbers. Extension to vector spaces is done via multiplication:

$$B(\mathbf{x}) = B(x_1)B(x_2)\cdots B(x_n) \tag{12}$$

where $\mathbf{x} = [x_1, x_2, \ldots, x_n]$.

2.4 Image symmetries

In this section we define and generalize symmetries of images that we wish neural networks to respect. First we describe properties related to lightning in image - contrast, brightness, color balance and gamma correction. Then geometric operators - rotations, scaling and shear - are presented. Throughout this section and for the rest of the work, "image" is understood to be an array of floating point numbers in range $[0; 1]$ of shape $(3, H, W)$.

2.4.1 Contrast

Though notion of contrast has many different definitions and variations, one popular formula [1, 2] for its adjustment by factor of a is

$$C_a(X) = \mathit{CLIP}\left(a \cdot X + (1-a) \cdot E\left[\frac{w_1 X_r + w_2 X_g + w_3 X_b}{w_1 + w_2 + w_3}\right]\right) \tag{13}$$

where *CLIP* is function clipping values to $[0; 1]$ range, X_r, X_g and X_b are red, green and blue channels of the image and the expectation on the right is mean value of grayscale version of image. Grayscale is understood to be a weighted average of image's channels. Changing the contrast by factor of 1 has no effect, using the factor of 2 is supposed to double the contrast, factor of 0.5 halves the contrast, etc. This definition however is not suitable for usage in the context of neural networks. First of all it assumes the tensor has 3 channels, while we would like to have a definition independent of number of channels. The concept of green channel or red channel also doesn't exist for tensors with more channels. Therefore we choose to generalize the definition by replacing weighted average with usual average. The second problem is clipping values. Working only with values in range $[0; 1]$ at all times would make training a neural network

a lot harder and perhaps significantly diminish its performance. To avoid these effects we entirely give up on clipping values. The final formula for changing contrast is following:

$$\mathcal{C}_a(x) = ax + (1-a)\mu_x \tag{14}$$

where μ_x is the average value of whole tensor. It's worth noting, that removing the clipping changes the general behavior of the transformation. While it remains more or less unchanged for values of a close to 1, differences grow as me move further away from identity transformation.

This operator possesses a number of interesting properties:

- $\mathcal{C}_a$ is linear operator on vector space of images of fixed size:

  $\mathcal{C}_a(x+y) = a(x+y) + (1-a)\mu_{x+y} = ax + ay + (1-a)\mu_x + (1-a)\mu_y = \mathcal{C}_a(x) + \mathcal{C}_a(y)$
  $\mathcal{C}_a(\lambda x) = a\lambda x + (1-a)\mu_{\lambda x} = \lambda ax + (1-a)\lambda\mu_x = \lambda(ax + (1-a)\mu_x) = \lambda\mathcal{C}_a(x)$

- Under map composition, set $\mathcal{C} = \{\mathcal{C}_a | a \in \mathbb{R}_+\}$ forms a Lie group isomorphic to $(\mathbb{R}_+, \cdot)$ – set of positive real numbers with multiplication:

  1. Closure:
     $\mathcal{C}_a \circ \mathcal{C}_b(x) = \mathcal{C}_a(bx + (1-b)\mu_x) = abx + a(1-b)\mu_x + (1-a)\mu_{bx+(1-b)\mu_x} =$
     $abx + a(1-b)\mu_x + (1-a)b\mu_x + (1-a)(1-b)\mu_x = abx + (1-ab)\mu_x = \mathcal{C}_{ab}(x)$
  2. Associativity:
     $\mathcal{C}_a \circ (\mathcal{C}_b \circ \mathcal{C}_c(x)) = \mathcal{C}_a(\mathcal{C}_{bc}(x)) = \mathcal{C}_{abc}(x) = \mathcal{C}_{ab} \circ \mathcal{C}_c(x) = (\mathcal{C}_a \circ \mathcal{C}_b) \circ \mathcal{C}_c(x)$
  3. Identity:
     $\mathcal{C}_1 \circ \mathcal{C}_a(x) = \mathcal{C}_a \circ \mathcal{C}_1(x) = \mathcal{C}_a(x)$
  4. Inverse element:
     $\mathcal{C}_a \circ \mathcal{C}_{a^{-1}}(x) = \mathcal{C}_{a^{-1}} \circ \mathcal{C}_a(x) = \mathcal{C}_1(x)$

  The homomorphism is simply $h : \mathcal{C}_a \mapsto a$

  $$h(\mathcal{C}_a)h(\mathcal{C}_b) = ab = h(\mathcal{C}_{ab})$$

  It is bijective so it's isomorphism.

- Equation 14 defines a group action of $\mathcal{C}$ on space of images: $\mathcal{C}_1 \cdot x = 1x + (1-1)\mu_x = x$ and checking the compatibility axiom is the same as checking the closure axiom above.

- $\mathcal{C}_n$ increases standard deviation of image n times:
  $\sigma(\mathcal{C}_n(x)) = \sigma(nx + (1-n)\mu_x) = \sigma(nx) = n\sigma(x)$

Visually the difference between operators using weighted and unweighted averages is small. Picture 3 shows series of photographies from STL10 dataset with with varying contrast changes.

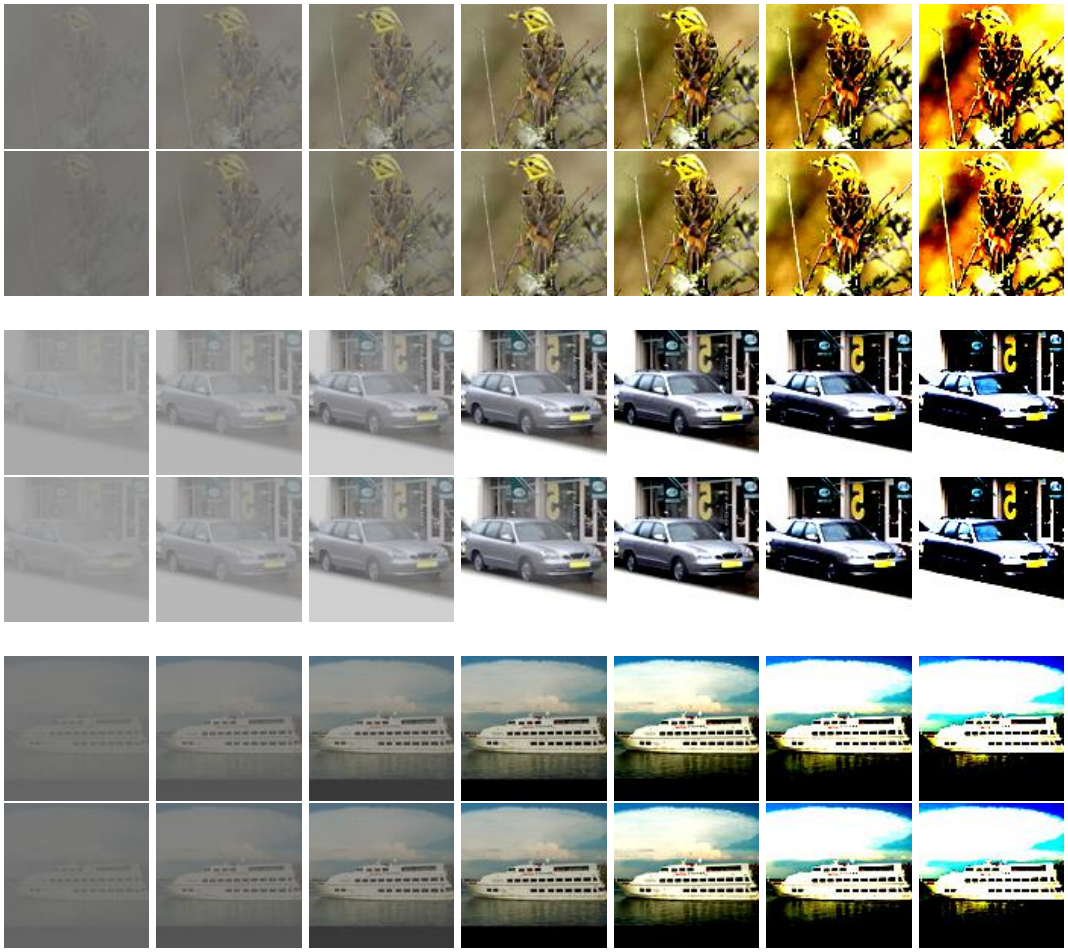


Figure 3: Differences between weighted and unweighted contrast change operator. Top rows: C_a operator (equation 13). Bottom rows: $\mathcal{C}_a$ operator (equation 14 with clipping).

2.4.2 Brightness

Change in image brightness is defined similarly to change in contrast, except the subtraction of image's mean is skipped:

$$B_a(X) = CLIP\left(aX\right) \tag{15}$$

Again omitting the clipping, the brightness change operator is defined simply as

$$\mathcal{B}_a(x) = ax \tag{16}$$

All properties characterising the $\mathcal{C}$ operators listed above are also true for the set $\mathcal{B} = \{\mathcal{B}_a | a > 0\}$. It's worth noting, that in this case removing the clipping step doesn't change the definition so significantly. In fact both formulas agree as long as $a \leq 1$, that is as along as brightness is not increased. Picture 4 shows images with brightness varying according to equation 15.

$a = 0.1$ $a = 0.25$ $a = 0.5$ $a = 1$ $a = 2$ $a = 4$ $a = 10$

Figure 4: Images from STL10 dataset with brightness changed with operator B_a.

2.4.3 Color balance

Suppose we are given a 3-channel image represented as a list of channels $[X_r, X_g, X_b]$. Then the operator changing the color balance to the balance corresponding to the temperature T is defined as

$$\mathcal{W}_T\left([X_r, X_g, X_b]\right) = [T_r X_r, T_g X_g, T_b X_b] \tag{17}$$

where T_r, T_g, T_b are RGB components corresponding to temperature T. For example for $T = 1000K$, T_r, T_g and T_b equal $1, 0.0401$ and 0 respectively [3]. T ranges from $1000K$ to $40000K$ with steps of $100K$ though there is not much change after the $15000K$ threshold. Figure 5 shows part of possible temperature spectrum together with colors characterising temperatures in terms of uint8 RGB triplets. Picture 6 shows images with varying color balance.

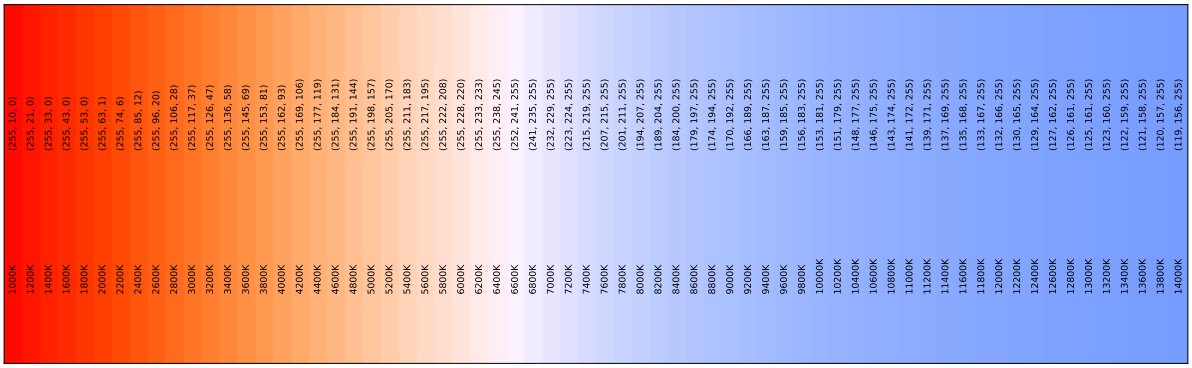


Figure 5: Part of spectrum of color balance temperatures and colors with steps of 200K.

Unlike contrast or brightness, the color balance is inherently bound to colors of image, so it's very hard to come up with reasonable generalization to higher dimensions which would allow us to measure equivariance. We choose not to do that and instead only construct networks invariant to changes in color balance.

$T = 1000K$ $T = 3000K$ $T = 5000K$ $T = 7000K$ $T = 9000K$

Figure 6: Images from STL10 dataset with color balance changed with operator $\mathcal{W}_T$.

2.4.4 Gamma correction

In virtually all computer vision libraries [4, 5, 6] gamma correction of image X is defined as

$$G_{a,c}(X) = cX^a \tag{18}$$

where c and $a \in \mathbb{R}_+$ and exponentiation is done entrywise. Multiplication by c is often optional and we drop it for simplicity. Figure 7 shows examples of pictures with various gamma settings.

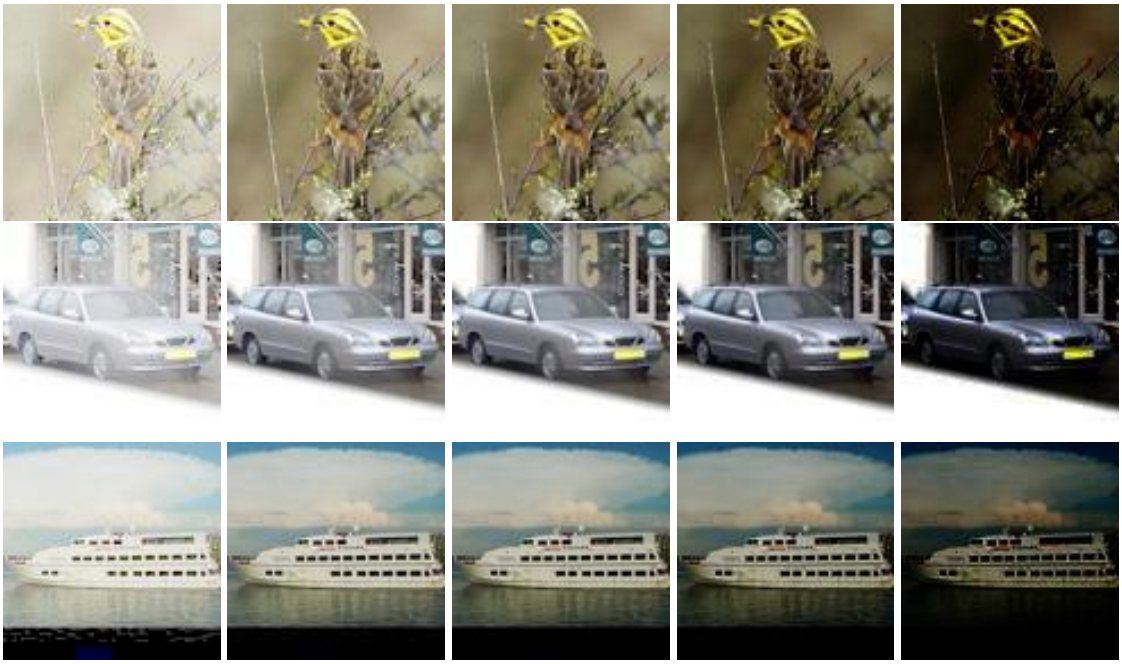
Unfortunately this definition is troublesome when applied to tensors with negative values as for example $a = \frac{1}{2}$ results in complex values. Also since every number has n complex roots of degree n, a problem with selecting the proper root appears. In order to avoid these problems, we restrict computation to real numbers and generalize equation 18 by:

$$\mathcal{G}_a(X) = s(X) \cdot |X|^a \tag{19}$$

where $s(X)$ is a tensor of the same shape as X containing signs of its entries and $|X|$ contains absolute values of entries. Exponentiation is again computed entrywise. This way we separate signs and magnitudes.

Unlike the three previous operators, $\mathcal{G}_a$ is nonlinear – $(X+Y)^a \neq X^a+Y^a$ unless $a = 1$. This family of transformations forms however a group under function composition $\mathcal{G}_a \circ \mathcal{G}_b = \mathcal{G}_{ab}$. It's isomorphic to $(\mathbb{R}_+, \cdot)$ with isomorphism f given by $f : \mathcal{G}_a \mapsto a$ and acts on the space of tensors of fixed shape:

- $\mathcal{G}_1(X) = X^1 = X$
- $\mathcal{G}_a\left(\mathcal{G}_b\left(X\right)\right) = \mathcal{G}_a\left(s\left(X\right) \cdot \left|X\right|^b\right) = s\left(s\left(X\right) \cdot \left|X\right|^b\right) \cdot \left|s\left(x\right) \cdot \left|X\right|^b\right|^a = s\left(s\left(X\right)\right) \cdot$



$a = 0.33$ $a = 0.75$ $a = 1$ $a = 1.5$ $a = 3$

Figure 7: Images from STL10 dataset with varying gamma settings.

$$\left||X|^b\right|^a = s\left(X\right) \cdot |X|^{ba} = \mathcal{G}_{ab}(X)$$

2.4.5 Rotation

When speaking of image rotating, we typically mean rotation around image's center. In coordinate system with $(0, 0)$ representing the center, rotation of coordinates can be represented as an element of $SO(2)$ group - 2×2 matrix of the form

$$\mathcal{R}_\theta = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

which transforms coordinates (2d vectors) by matrix multiplication. After rotating coordinates, pixels of original image need to be resampled in some way, which slightly distorts the picture unless the rotation angle is a multiple of $90°$. If we want rotations to preserve all information present in picture we need to take care of appropriate expansion of borders and possible padding with 0s. It's also important from more theoretical point of view - expansion assures rotations compose nicely, that is form a group action on the space of images. Figure 8 illustrates necessity of border expansion. Lack of expansion causes rotation composition to lose part of information which moves outside the border.

While filters present in GCNNs are higher dimensional than in regular CNNs, spatial interpretation of the last two dimensions remains unchanged. Therefore it makes sense to generalize $\mathcal{R}$ to transform N-dimensional tensor by separately rotating last two dimensions of every subtensor and stacking them in original shape. The same goes for scaling and shear transformations described next.

Figure 8: Difference between rotation with and without border expansion. Left column shows the original image and its rotation by 90°. In middle column image is first rotated by 45°with expansion, then rotated again and cropped to original size. In the right column picture is simply rotated twice by 45°.

2.4.6 Scaling

Scaling is a zoom-in or zoom-out transformation in reference to the center of the image. Scaling of coordinates can be written as scalar matrix

$$\mathcal{S}_k = \begin{bmatrix} k & 0 \\ 0 & k \end{bmatrix} \tag{20}$$

Just like in rotation, after rescaling of coordinates, image needs to be resampled. Figure 9 shows varying degree of scaling. Note that by default scaling-up causes loss of information, so we might want to use image expansion as well. However while using finite expansion (increasing image size by at most $\sqrt{2}$) we can make rotations into a group action, it's not the case for scaling. In order to preserve all information during scaling by factor of $k > 1$, we would have to increase image size by the same factor. For this reason scaling is often modeled not as a group, but rather as a semi-group or monoid. This however is not really any obstacle – what matters is that locally it still

has structure of a Lie group.

$k = 0.5$ $k = 0.75$ $k = 1$ $k = 1.5$ $k = 2$

Figure 9: Images from STL10 dataset with scaled by a factor of k.

2.4.7 Shear

Shear operator 'tilts' the coordinates. If we use the standard coordinate system centered in the middle of the image, the first vector e_1 remains unchanged while e_2 is mapped to $\lambda e_1 + e_2$:

$$\mathcal{SH}_\lambda = \begin{bmatrix} 1 & \lambda \\ 0 & 1 \end{bmatrix} \tag{21}$$

where $\lambda \in \mathbb{R}$. Parameter λ is theoretically unbounded so the remarks from scaling section apply here as well. In practice however $\lambda = \pm -1$ causes tilt of $45°$ which is already very significant so we only consider $\lambda \in [-1; 1]$. Exemplary effect of transformation is shown in figure 10.

$\lambda = -1$ $\lambda = -0.5$ $\lambda = 0$ $\lambda = 0.5$ $\lambda = 1$

Figure 10: Images from STL10 dataset transformed by $\mathcal{SH}_\lambda$ for varying λ.

3 Image symmetry implementation

3.1 Invariant Layers

3.1.1 CBW Layer

As described in section 2.2, in order to construct a network invariant to some group of transformations, either the first layer has to be invariant or first layers have to be equivariant with an invariant layer following them. In this section we construct networks of both of these types. First we need to construct layers invariant to changes in contrast, brightness, color balance and gamma correction as defined in 2.4. If possible it would be best to use a single function invariant to all of these transformations at once. While we weren't able to come up with such layer, we can do the next best thing and build layers invariant to three of these transformations. We show that standard instance normalization is invariant to contrast, brightness and color balance changes.

Instance normalization layer *CBW* transforms each channel X_c of image X by:

$$CBW(X_c) = \frac{X_c - E[X_c]}{\sigma(X_c)}$$

where $E[X_c]$ is mean value of X_c and σ is its standard deviation. Each instance in batch is normalized separately. *CBW* is invariant to changes in contrast $\mathcal{C}$:

$$\begin{aligned}
CBW(\mathcal{C}_a(X)_c) &= \frac{aX_c + (1-a)E[X]_c - E\left[aX_c + (1-a)E[X]_c\right]}{\sigma(aX_c + (1-a)E[X]_c)} \\
&= \frac{aX_c + (1-a)E[X]_c - aE[X_c] - (1-a)E[X]_c}{a\sigma(X_c)} \\
&= \frac{aX_c - aE[X_c]}{a\sigma(X_c)} \\
&= \frac{X_c - E[X_c]}{\sigma(X_c)} \\
&= CBW(X_c)
\end{aligned}$$

Similarly for brightness changes $\mathcal{B}$:

$$\begin{aligned}
CBW(\mathcal{B}_a(X)_c) &= \frac{aX_c - E\left[aX\right]_c}{\sigma(aX_c)} \\
&= \frac{aX - aE[X_c]}{a\sigma(X_c)} \\
&= \frac{X_c - E[X_c]}{\sigma(X_c)} \\
&= CBW(X_c)
\end{aligned}$$

Color balance changes $\mathcal{W}$ act on individual channels in the same way $\mathcal{B}$ does so:

$$\begin{aligned} \mathit{CBW}(\mathcal{W}_T(X)_c) &= \frac{T_c X_c - E\left[T_c X_c\right]}{\sigma(T_c X_c)} = \\ &= \frac{T_c X - T_c E[X_c]}{T_c \sigma(X_c)} = \\ &= \frac{X_c - E[X_c]}{\sigma(X_c)} = \\ &= \mathit{CBW}(X_c) \end{aligned}$$

Therefore any neural network with *CBW* as first layer makes the network invariant to considered image transformations. We denote this type of networks as {model-name}+InCBW0, e.g. Plain+InCBW0 or RotEq+InCBW0. Where 0 refers to placement of the invariant layer at the beginning of the network. We also consider invariant networks with *CBW* layer placed further down the processing stream. As mentioned earlier, constructing such network requires all layers before *CBW* to be equivariant. Since the equivariance to changes in color balance is not defined, we focus on contrast and brightness. In the next section 3.2 we present variants of common layers like convolution or pooling equivariant to these transformations. We use these components to construct invariant networks denoted InB1, InB2, InB3. Numbers again refer to placement of the *CBW* layer – the higher the number, the deeper invariant layer is placed.

3.1.2 GBW Layer

Just like *CBW* layer is invariant to changes in contrast, we can construct similar normalization layer *GBW* invariant to changes in gamma:

$$\mathit{GBW}(X_c) = \frac{\log X_c - E[\log X_c]}{\sigma(\log X_c)} \tag{22}$$

where $\log(X)$ is entrywise logarithm of X. The base of the logarithm is not very important – invariant behaviour doesn't depend on its value; in implementation we use base e. Now let us assume that we're given an image – $3 \times H \times W$ tensor – with strictly positive values.. Operators $\mathcal{G}$, $\mathcal{B}$ and $\mathcal{W}$ act on it entrywise, so we can as well analyze transformation of individual channels of the image. Let X be a single channel. We prove 22 is invariant to gamma, brightness and color balance changes. $\mathcal{B}$ and $\mathcal{W}$ transform single channel in the same way, by multiplication, so we only need proofs for $\mathcal{G}$ and $\mathcal{B}$:

$$
\begin{aligned}
\mathit{GBW}(\mathcal{G}_a(X)) &= \mathit{GBW}(X^a) \\
&= \frac{\log(X^a) - E[\log(X^a)]}{\sigma(\log(X^a))} \\
&= \frac{a\log(X) - E[a\log(X)]}{\sigma(a\log(X))} \\
&= \frac{a\log(X) - aE[\log(X)]}{a\sigma(\log(X))} \\
&= \frac{\log(X) - E[\log(X)]}{\sigma(\log(X))} \\
&= \mathit{GBW}(X)
\end{aligned}
$$

$$
\begin{aligned}
\mathit{GBW}(\mathcal{B}_a(X)) &= \mathit{GBW}(aX) \\
&= \frac{\log(aX) - E[\log(aX)]}{\sigma(\log(aX))} \\
&= \frac{\log(X) + \log(a) - E[\log(X) + \log(a)]}{\sigma(\log(X) + \log(a))} \\
&= \frac{\log(X) + \log(a) - E[\log(X)] - \log(a)}{\sigma(\log(X))} \\
&= \frac{\log(X) - E[\log(X)]}{\sigma(\log(X))} \\
&= \mathit{GBW}(X)
\end{aligned}
$$

There is however slight technical issue with definition 22. Since images contain values from interval $[0; 1]$, taking their logarithm directly is impossible because of cells containing 0. This problem is solved easily by adding some small constant value ϵ to every cell before computing logarithm. The exact definition is then the following:

$$
\mathit{GBW}_\epsilon(X_c) = \frac{\log(X_c + \epsilon) - E[\log(X_c + \epsilon)]}{\sigma(\log(X_c + \epsilon))} \tag{23}
$$

To be precise, this small correction breaks the invariance properties. We can however treat $X_c + \epsilon$ as a new input of the network and GBW_ϵ is then invariant to this new input. ϵ is small so X_c and $X_c + \epsilon$ are very similar (or even indistinguishable to human eye), so they should be labeled by the network as belonging to the same class anyway. In implementation we use $\epsilon = \frac{1}{255}$ – the smallest possible difference between pixel values. Due to logarithm only being defined for positive values, we place *GBW* layer only at the very beginning of networks. This type of models is marked as {model-name}+InGBW0. In order to prevent division by 0 in case of constant channels, it's necessary to add another small constant to denominator – we use value 10^{-6}. We do the same in case of *CBW* layer. These approximations also cause loss of strict invariance, but again, values are very small so behaviour of the layer doesn't change that much. Precise error values

are discussed in section 4.4.

3.2 Equivariant models

3.2.1 Brightness equivariance

Recall that layer f is equivariant to change of brightness if $f(aX) = af(X)$ for all $a > 0$. We begin with showing that common building blocks of CNNs except for normalization fulfill this condition:

- **Convolution** is linear operation so $\mathit{Conv}(aX) = a\mathit{Conv}(X)$
- **Pooling** - all commonly used pooling layers are equivariant:
  - MaxPooling amounts to taking maximal value of some set and $\max\{ax_1, ax_2, \cdots, ax_n\} = a \max\{x_1, x_2, \cdots, x_n\}$
  - AveragePooling is precisely expectation operation $E[X]$ and $E[aX] = aE[X]$
  - EuclideanNormPooling - $\sqrt{(ax_1)^2 + \cdots + (ax_n)^2} = a\sqrt{x_1^2 + \cdots + x_n^2}$
- **Activation functions** – equivariant pointwise activation function is characterised by equation $f(ax) = af(x)$ for all $a > 0$. Assuming f is differentiable, $af'(ax) = af'(x)$, so $f'(ax) = f'(x)$ which means that $f'(a) = f'(1)$ for $a > 0$ and $f'(b) = f'(-1)$ for $b < 0$. This implies f has form of generalized ReLU function

  $$f(x) = \begin{cases} ax & \text{for some } a \in \mathbb{R} \quad \text{if } x < 0 \\ bx & \text{for some } b \in \mathbb{R} \quad \text{if } x \geq 0 \end{cases}$$

  In particular we can use usual ReLU function.
- **Normalization** As shown above, instance normalization is invariant to brightness changes. It's easy to see that it also holds for BatchNormalization. This stems mainly from division by standard deviation which is equivariant to $\mathcal{B}$. In fact the numerator of *CBW* is also equivariant to $\mathcal{B}_a$: $aX_c - E[aX_c] = a(X_c - E[X_c])$, so division by any factor involving a will break equivariance. For the same reason, we can't pass the numerator through any nonlinear function, e.g. square root. Therefore if we want the normalization to zero out mean value of input, we must divide by a constant, e.g. 1. In implementation we use mean normalization layer

  $$\mathit{MN}(X) = X - E[X] \tag{24}$$

  Lack of division by standard deviation or some other dispersion metric might have an adverse effect on training procedure as there is no mechanism to keep values flowing through the network bounded, which might easily result in gradient explosion. In experiments in section 4.2 we find the training on one of datasets indeed deteriorates, but numerical errors don't occur.

And so using common components with standard normalization scheme replaced by mean normalization we can build model equivariant to brightness changes $\mathcal{B}$. In experiments refer to this type of architecture as BrightnessEq.

3.2.2 Lie GCNN equivariant models

For geometric transformations, we implement Lie GCNN based on B-splines [7] in PyTorch framework. The only tough part is really implementing general operations of lifting and group convolution. After that implementing model equivariant to any group only requires implementing operations of these group – taking inverse, multiplication, distance, etc. In this way we construct models equivariant to well researched groups like rotation and scaling groups as well as shear transform which hasn't come up in literature review.

We now consider remaining lightning symmetries $\mathcal{C}$ and $\mathcal{G}$. Es expected, they are not as well-behaved as $\mathcal{B}$. While MaxPooling is also equivariant to them, the rest of described above layers isn't (see experiments in section 4.4). For the rest of this section we construct suitable convolutional layers similar to those of Lie GCNNs (section 2.3.3), activation functions and normalization layers equivariant to $\mathcal{C}$ and $\mathcal{G}$ operators.

- The crucial part of pipeline is convolution layer. Nature of geometric and lightning symmetries is fundamentally mathematically different. While the first transform coordinates or grid of image, the latter group acts pointwise on values present in the tensor without changing coordinates in any way. This property renders the implementation used for geometric models unfit. Where in case of geometric symmetries different transformed locations on original grid are sampled, in case of lightning symmetries these would be the exact same locations. This in turn would cause creation of identical kernels and unnecessary repetition of computation. Therefore it's obligatory to transform the kernels in some other way consistent with equations of lifting and convolution. We implement such mechanism using the transformations from transformation group. That is if we want the layer to be equivariant with respect to $\mathcal{G}_c$, then its kernel K assumes values at gamma level c equal to base kernel gamma-corrected by factor of c:

  $$K(g) = K(x, c) = \mathcal{G}_c(K(x, 0)) \tag{25}$$

  where g is represented as a pair (spatial location, gamma level).

- Because of aggregation of global mean of tensor in contrast equation 14, designing a nontrivial pointwise activation function equivariant to $\mathcal{C}$ operator might be impossible – value in any given cell would automatically depend on values of all other cells. Instead we test numerically equivariance of broad range of popular activation functions and find that function Softsign defined as $\mathit{Softsign}(x) = \frac{x}{1+|x|}$ performs fairly well.

  Operator $\mathcal{G}$ proves to be more manageable and exactly equivariant activation function can be found, though in the end it turns out to be fairly obvious. If we want f to be equivariant to gamma equation 19, then in particular it needs to fulfill condition

  $$f\left(x^c\right) = f\left(x\right)^c$$

for any $x > 0$ and $c > 0$. To obtain analytic form of f let

$$f(x_0) = y_0$$

for some constants $x_0 \neq 1$ and y_0. Then

$$f\left(x_0^c\right) = y_0^c$$

Now we can treat x_0^c as the argument and set $x_0^c = z$, so

$$f(z) = y_0^{\log_{x_0} z} = y_0^{\frac{\log_{y_0} z}{\log_{y_0} x_0}} = z^{\log_{x_0} y_0}$$

but $\log_{x_0} y_0$ is constant, so f is unique power function passing through (x_0, y_0). For negative arguments this definition can be extended either symmetrically ($f(x) = f(-x)$) or antisymmetrically ($f(-x) = -f(x)$). We choose the latter option to enable function to assume negative values. After some trial and error we arrive at the final definition

$$f_a(x) = s(x)|x|^a$$

with s representing the sign of x. In implementation, in order to make f slowly increasing, we choose $a = 0.3$. $f_{0.3}$ is shown in the figure 11. Unfortunately such activation function always leads to gradient explosion after processing from 2 to 20 data batches. In consequence also in this case we use *Softsign* function.

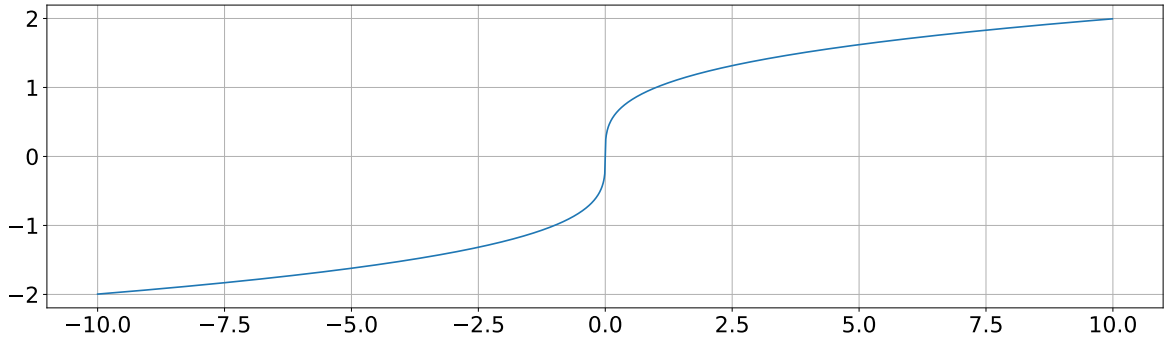


Figure 11: Activation function $s(x)|x|^{0.3}$.

- Similarly to brightness, the contrast operator $\mathcal{C}$ is equivariant to mean normalization and invariant to standard normalization, so to build an equivariant model, we need to use the mean normalization.

  Situation is a lot more complicated with gamma correction. Due to its highly nonlinear nature, operator $\mathcal{G}$ doesn't seem to be equivariant to any commonly used normalization layer. We test models with and without BatchNormalization and compare their accuracies and equivariance degrees.

4 Experiments

4.1 Experimental setup

The implementation was written in Python language using PyTorch, Numpy, Pillow and Torchvision libraries and is available at `https://github.com/kamieen03/mgr`. Implementation of B-spline GCNNs is based on original Tensorflow version from [7]. Implemented models were trained and tested on CIFAR10 [19], CIFAR100 [19] and STL10 [9] datasets. For STL10 only the labeled part of training set was used.

All implemented models are based on ResNet architecture [16]. Every network starts with initial convolution, normalization, activation function and pooling layer. After that either 6 (in CIFAR models) or 8 (in STL10 models) residual bottlenecks are placed. Bottlenecks come in pairs, that is bottleneck no. $2k$ and no. $2k+1$ always have the same number of channels. We use list notation to denote size of network, for example [*n1*; *n2*; *n3*] is CIFAR model with 2 bottlenecks with $n1$ channels, 2 bottlenecks with $n2$ channels and 2 bottlenecks with $n3$ channels placed consecutively. Every network ends with fully-connected layer. Diagrams 12 and 13 show structure of ResNet with 6 bottleneck blocks and a single bottleneck. The types of *normalizations* and *activation functions* used throughout the network depend on the type of model.

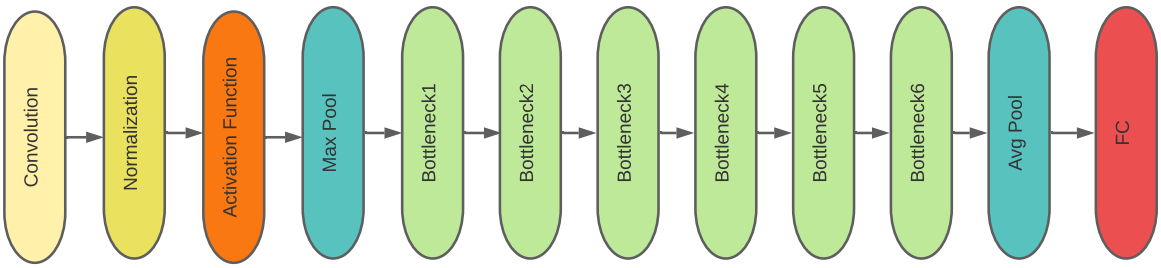


Figure 12: Architecture of ResNet network with 6 residual bottlenecks.

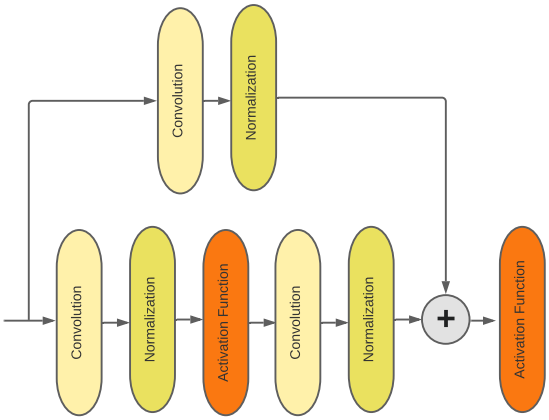


Figure 13: A single residual bottleneck of ResNet. The top branch consists of *convolution* and *normalization* only if the bottleneck is supposed to downsample the input. Otherwise it performs identity transformation.

Overall there are 3 types of models:

- **PlainResNet** is regular ResNet as described in [16].
- In **BsplineResNet** usual convolutional layers are replaced by B-spline group convolutional layers and the first convolution is replaced by lift operation. Network is parametrized by group G it's supposed to be equivariant to and size of B-spline basis – N in equation 11. Model name is determined by G
- **BResNet** resemble Plain models, but standard normalization is replaced by mean normalization layer 24, so that the first couple layers are equivariant to change in brightness. The exact number of equivariant layers is hyperparameter. In *BrightnessEq* model all layers before final fully-connected layer are equivariant. In *InBk* networks, a single *CBW* layer is placed in the first Bottleneck of kth pair with all earlier Bottlenecks equivariant.

Plain and Bspline models can additionally have *CBW* or *GBW* layer at the very beginning of the network. In such case "*+InCBW0*" or "*+InGBW0*" is added to their names. All architectures are detailed in table 1 and 2. Particular layer sizes of models in each table were selected in such a way to assure all of them have roughly the same number of trainable parameters. For CIFAR models it's $3\,200\,000$ while for STL10 about $5\,700\,000$.

Model type	Model name	Numbers of channels	Initial invariant layer	Equivariance group	B-spline basis size
PlainResNet	Plain	[80; 160; 256; 256]	—	—	—
	Plain+InCBW0	[80; 160; 256; 256]	CBW	—	—
	Plain+InGBW0	[80; 160; 256; 256]	GBW	—	—
BsplineResNet	RotEq	[32; 64; 64; 64]	—	SO(2)	12
	RotEq+InCBW0	[32; 64; 64; 64]	CBW	SO(2)	12
	RotEq+InGBW0	[32; 64; 64; 64]	GBW	SO(2)	12
	ScaleEq	[48; 64; 100; 128]	—	Scale group	5
	ScaleEq+InCBW0	[48; 64; 100; 128]	CBW	Scale group	5
	ScaleEq+InGBW0	[48; 64; 100; 128]	GBW	Scale group	5
	SchearEq	[48; 64; 100; 128]	—	Schear group	5
	SchearEq+InCBW0	[48; 64; 100; 128]	CBW	Schear group	5
	SchearEq+InGBW0	[48; 64; 100; 128]	GBW	Schear group	5
	GammmaEq	[48; 64; 100; 128]	—	Gamma group	5
BResNet	BrightnessEq	[80; 160; 256; 256]	—	—	—
	InB1	[80; 160; 256; 256]	—	—	—
	InB2	[80; 160; 256; 256]	—	—	—
	InB3	[80; 160; 256; 256]	—	—	—

Table 1: STL10 models

Model type	Model name	Numbers of channels	Initial invariant layer	Equivariance group	B-spline basis size
PlainResNet	Plain	[80; 160; 256]	—	—	—
	Plain+InCBW0	[80; 160; 256]	CBW	—	—
	Plain+InGBW0	[80; 160; 256]	GBW	—	—
BsplineResNet	RotEq	[32; 48; 64]	—	SO(2)	12
	RotEq+InCBW0	[32; 48; 64]	CBW	SO(2)	12
	RotEq+InGBW0	[32; 48; 64]	GBW	SO(2)	12
	ScaleEq	[64; 96; 128]	—	Scale group	3
	ScaleEq+InCBW0	[64; 96; 128]	CBW	Scale group	3
	ScaleEq+InGBW0	[64; 96; 128]	GBW	Scale group	3
	SchearEq	[48; 72; 108]	—	Schear group	5
	SchearEq+InCBW0	[48; 72; 108]	CBW	Schear group	5
	SchearEq+InGBW0	[48; 72; 108]	GBW	Schear group	5
	GammmaEq	[48; 72; 108]	—	Gamma group	5
BResNet	BrightnessEq	[80; 160; 256]	—	—	—
	InB1	[80; 160; 256]	—	—	—
	InB2	[80; 160; 256]	—	—	—
	InB3	[80; 160; 256]	—	—	—

Table 2: CIFAR models

4.2 Image classification

We begin experiments with comparison of classification accuracy of individual models on all 3 datasets. For STL10 and CIFAR10 every network is trained for 150 epochs. This number was derived from first exploratory tests on STL10, where various variants of PlainResNet had shown little to no improvement after 100th epoch. Due to limited computational budget CIFAR100 models are trained for 100 epochs. Training is done using AdamW optimizer [21] with weight decay coefficient equal to 0.02 and initial learning rate set to 0.001. Size of single mini-batch equals 256. During training, input images are first zero-padded with 2 pixels on each side, cropped to original size and then flipped horizontally with probability 0.5. In CIFAR10 experiments with color jitter augmentations additional transforms of either contrast of brightness are applied. In all line plots below y axis indicates classification accuracy either on train or test set, while x axis enumerates epochs.

Comparison of PlainResNet with equivariant models

First we compare base models – Plain, RotEq, ScaleEq, ShearEq, GammaEq and BrightnessEq. The goal of experiment is to determine whether at fixed number of parameters, equivariant architectures have advantage over usual ResNet model.

Looking first at training runs in figure 14 Plain's performance stands out. It trains the fastest of all models but this should be no surprise. After all it has the greatest number of filters in each layer, so it's the most prone to rapid decrease of loss and

overfitting. It's especially apparent on the hardest dataset – CIFAR100, where even though training accuracy continues to grow, test accuracy staggers around 40th epoch and drops gently later on.

Another interesting case is BrightnessEq. While it has no problem training on CIFAR, its performance is underwhelming on STL. At first it seemed like result of particularly bad seed of network, so we've rerun the experiment but behaviour was essentially the same. Either characteristic of images in dataset or size their size impacts the training procedure negatively.

Analyzing plots in the test column, we see that the two best generalizing models are Plain and RotEq. While Plain performs better on STL10 and CIFAR10, on CIFAR100 RotEq is the only model to beat the 50% threshold. It should also be noted that again, probably due to high capacity, Plain's test curves are the most unstable – see for example 20% dips on STL10. The worst models are perhaps ScaleEq and GammaEq. On every dataset they seem to end up as the last ones in terms of accuracy ranking. Also even though BrightnessEq has a lot of problems training on STL10, it still reaches their level in the end.

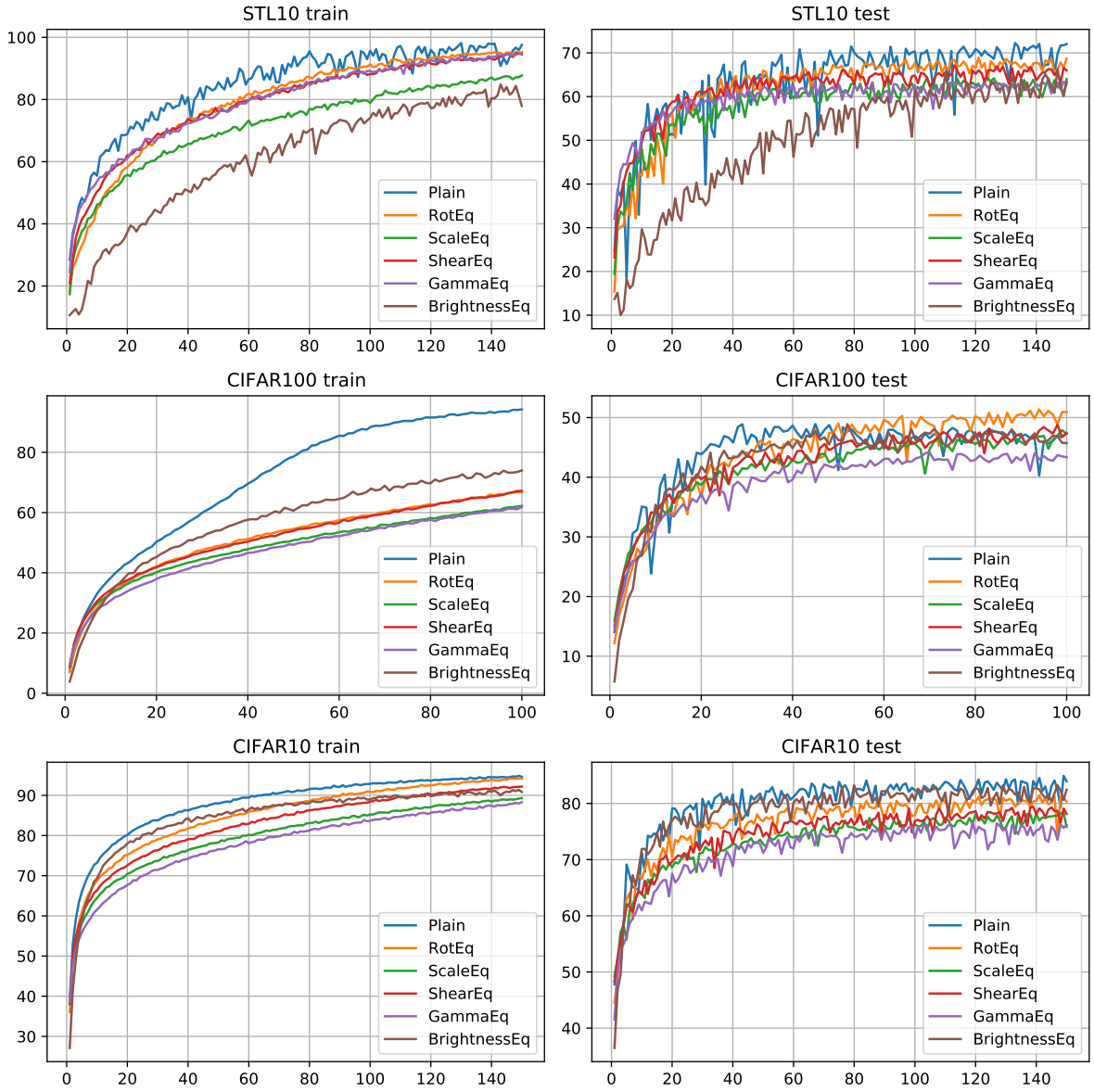


Figure 14: Comparison of classification accuracy of Plain and equivariant models on all datasets. X axis indicates number of epochs. Left column shows performance on training set; right column on the test set.

Comparison of bare, InCBW and InGBW versions of models

We now group base models with their InCBW and InGBW versions and compare their classification accuracy. Figures 15, 16 and 17 show experiments on STL10, CIFAR100 and CIFAR10 datasets respectively.

First thing to notice in STL10 runs is the difference between Plain models and the other networks. Test plots of Plain+InCBW0 and Plain+InGBW0 seem to be a lot more stable than that of bare Plain model. Their peaks and valleys have way smaller magnitude. InCBW performs a bit better than Plain, while InGBW version a bit worse. All of this is also true for CIFAR tests. Additionally on STL10, training curves indicate that Plain+InGBW0 trains slower than other two, though again more stable. Plots of RotEq, ScaleEq and ShearEq on the other hand are extremely similar to each other. Their bare, CBW and GBW versions perform basically the same, the only difference is somewhat slower training of InGBW types. Relative weakness of InGBW versions stands out more in CIFAR plots. In every of eight subplots, they are more or less dominated by bare and InCBW types. InCBW in turn performs either worse (CIFAR100) or the same (CIFAR10) as bare version.

Overall using CBW layer seems mostly advantageous as it stabilizes the prediction when training in low data regime and doesn't really hurt performance of more restricted (Eq) models. GBW layer in turn while also possessing the first quality, visibly impacts accuracy negatively. This might stem from characteristic of distributions created by GBW layer or more precisely by logarithm. Let X be an input image. If $\log\left(\frac{1}{255}\right)$ is the smallest value of $\log(X)$ and $\log\left(\frac{256}{255}\right)$ is the biggest value then the argument mapped halfway in-between them is

$$\exp\left(\frac{\log\left(\frac{1}{255}\right)+\log\left(\frac{256}{255}\right)}{2}\right)=\sqrt{\exp\left(\log\left(\frac{256}{255^2}\right)\right)}=\sqrt{\frac{256}{255^2}}=\frac{16}{255}$$

Numerically

$$\log\left(\frac{1}{255}\right)\approx -5.54 \qquad \log\left(\frac{16}{255}\right)\approx -2.77 \qquad \log\left(\frac{256}{255}\right)\approx 0.01$$

After logarithmization values are standardized but the fact remains that about half of unique input values are dedicated just to short interval $\left[\frac{1}{255};\frac{16}{255}\right]$. This asymmetrically big attention to dark pixels might be one of the factors negatively influencing model's accuracy.

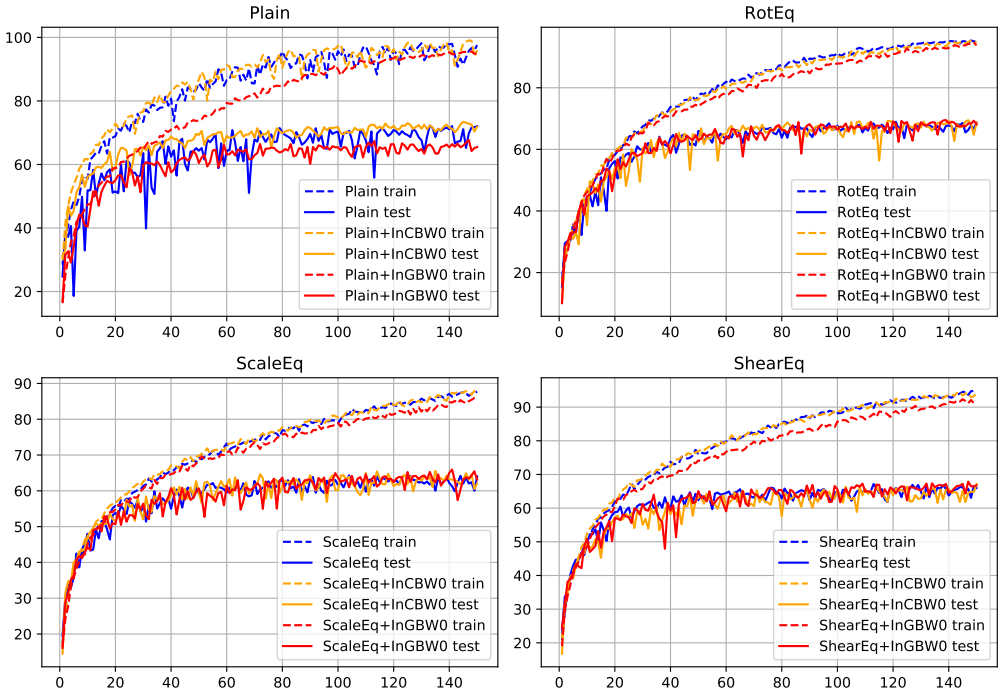


Figure 15: Comparison of classification accuracy of base models and their invariant versions on STL10 dataset.

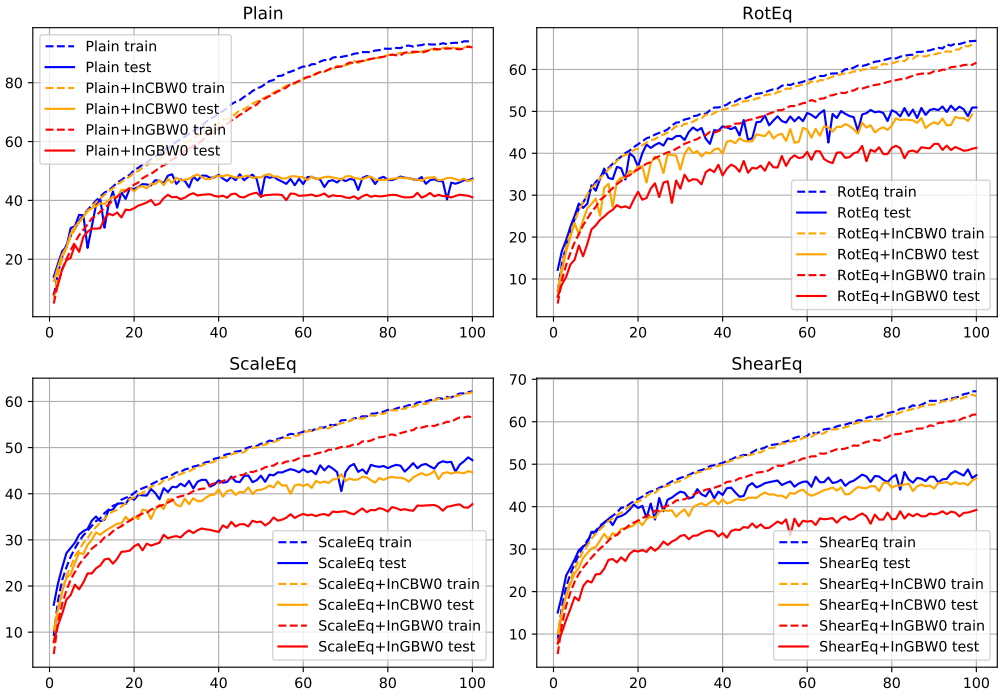


Figure 16: Comparison of classification accuracy of base models and their invariant versions on CIFAR100 dataset.

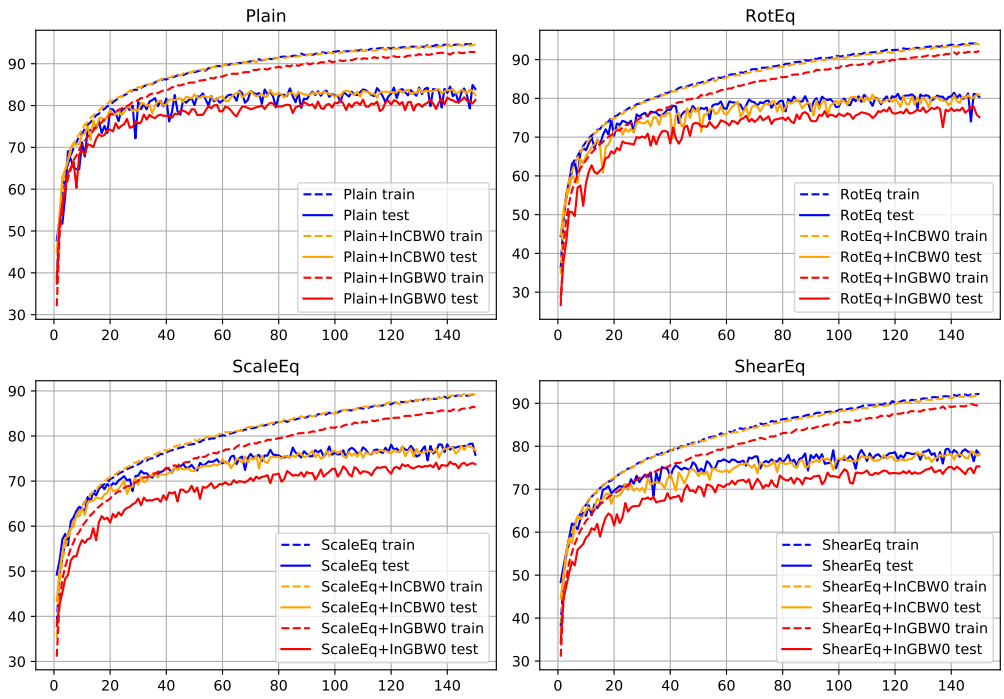


Figure 17: Comparison of classification accuracy of base models and their invariant versions on CIFAR10 dataset.

Comparison of brightness invariant models

Differences between architectures invariant and equivariant to changes in brightness are shown in figure 18. To be precise BrightnessEq isn't exactly equivariant – its last fully-connected layer isn't equivariant due to bias vector. In classification problem however, what's important is that significant portion of the network exhibits equivariant behaviour. By that logic though model InB5 should also probably called equivariant as its first 4 residual bottlenecks are equivariant. All in all boundary between invariance and equivariance in NN literature is fluid and one should always pay attention what is meant by these terms. This particular is best interpreted as asking whether depth at which network becomes invariant plays any role. The answer to this question seems to be yes, but only if the task is sufficiently hard.

Starting with the easiest CIFAR10 dataset we see test accuracy of all models differs only slightly, though it's worth noting that InB1 is ahead of others for most of the training time. Training curves on the other hand reveal significant difference of training speed of InB1 and BrightnessEq.

In CIFAR100 experiments the differences are more pronounced. InB1 clearly dominates the rest, while InB5 and BrightnessEq are the lest accurate. It's test curve is becomes stable after 50th epoch whereas InB5's doesn't. Training plots confirm as well that InB1 and InB3 are able to optimize the fastest.

InB5 does relatively poorly also on STL10. Curiously, skipping BrightnessEq, it's the Plain model whose training curve is the most jarred

Overall InB1 network seems to slightly outperform the default Plain+InCBW0 model. The only difference between them is availability of the brightness information to the first two convolutional layers and intermediate ReLU activation. This might help InB1

to differentiate between classes if the interclass variation of brightness is significant. This turns out to be true for CIFAR100 but not really for CIFAR10 and STL10 (see figure 19), which could explain big differences in performance between individual networks on the first dataset. Comparison of 5 most accurate networks for each dataset is shown in figure 20.

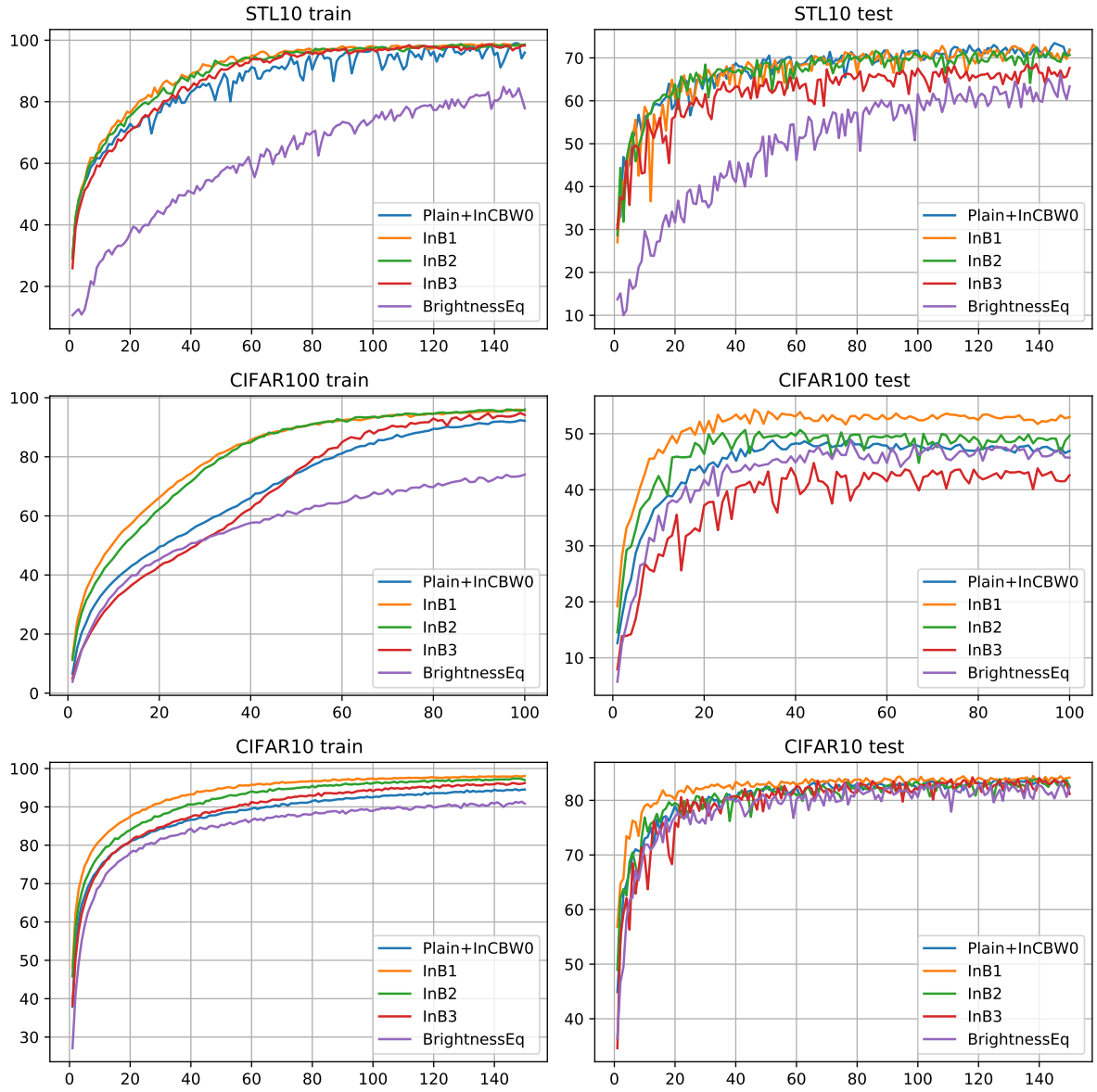


Figure 18: Comparison of classification accuracy of brightness equivariant and invariant models.

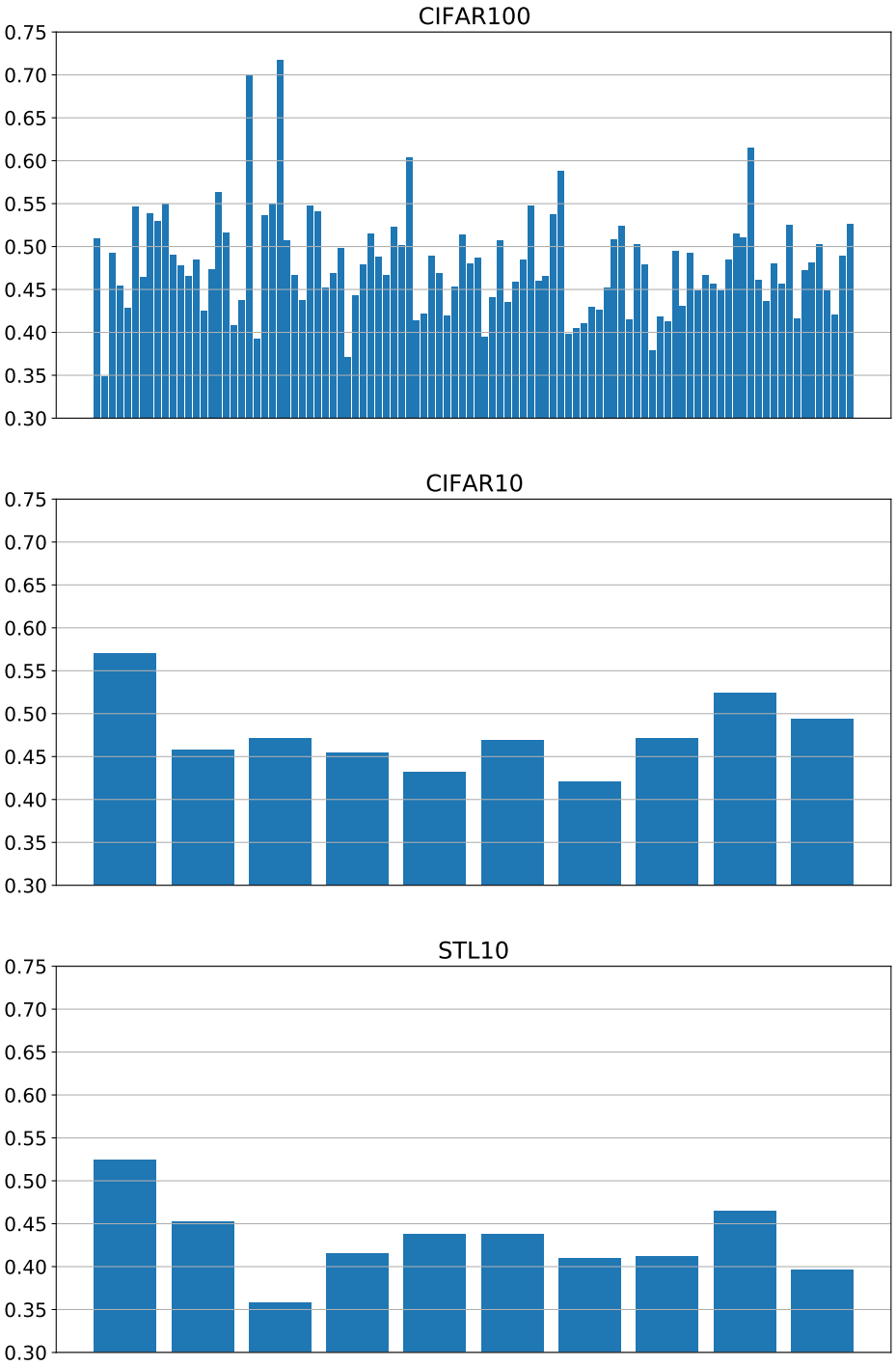


Figure 19: Mean brightness of every class in CIFAR100, CIFAR10 and STL10 datasets as measured on test parts.

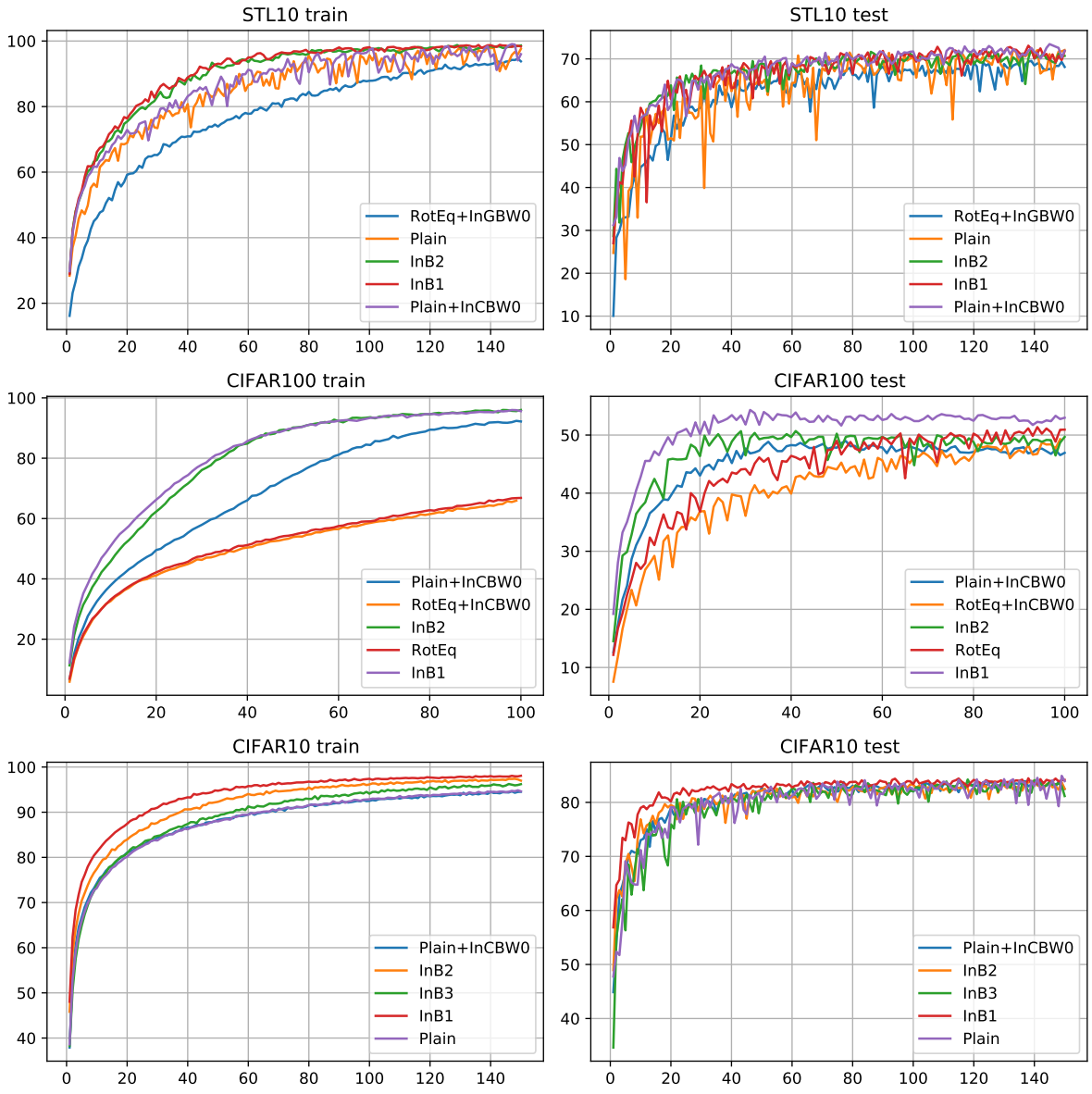


Figure 20: Comparison of classification accuracy of 5 best models for each dataset. Models were ranked according to maximum test set accuracy during training.

Model name	No augmentation	Augmentation
Plain	84.9	84.5
Plain+InCBW0	84.0	84.9
Plain+InGBW0	82.0	82.1
RotEq	81.5	81.3
RotEq+InCBW0	81.0	80.9
RotEq+InGBW0	77.9	79.5
ScaleEq	78.3	77.9
ScaleEq+InCBW0	77.8	78.5
ScaleEq+InGBW0	74.3	74.9
SchearEq	79.4	78.9
SchearEq+InCBW0	78.8	79.3
SchearEq+InGBW0	75.3	77.0
GammaEq	76.7	76.4
BrightnessEq	83.3	82.9
InB1	84.5	84.5
InB2	84.0	83.1
InB3	84.2	83.9

Table 3: Comparison of classification accuracy on CIFAR10 with and without data augmentation.

4.3 Generalizations

In this subsection we analyze effectiveness of data augmentation on accuracy of models as well as degrees of generalizations to lightning transforms attained by individual networks. We train separate set of 17 models on CIFAR10 dataset augmented with images of lowered brightness or contrast. Specifically during training, before passing through the network, each image x undergoes one of two transformations selected with equal probability:

- either the brightness is lowered according to equation 16: $x \mapsto \mathcal{B}_a(x)$
- or the contrast is lowered according to equation 13: $x \mapsto C_a(x)$

where factor a comes from uniform probability distribution $a \sim U[0.3; 1]$ and contrast operator C_a is the weighted, clipped version. Originally we intended for augmentations to also make increase in brightness and contrast possible. However in all exploratory experiments such modifications caused numerical errors and gradient explosions due to very bright images being present in the dataset.

Table 3 contains maximal test accuracies of models trained with and without augmentation. The test instances are unaltered. We conclude there is no real difference between the two sets of models. The augmented networks are not always better and even when they are, their advantage is mostly less than one percent. The only model which seems to benefit from augmentation is RotEq+InGBW0 where the difference is 1.6%, but this might be just a statistical effect. Numbers in the table 3 can be interpreted as coming from some probability distributions and so can their differences. The more models investigated, the higher the chance, one of them will perform significantly bet-

ter with augmentation. All in all there is no evidence for implemented augmentation scheme enhancing accuracy of prediction on unaltered test dataset.

The situation changes when we look at accuracy on transformed datasets. Matrices in figures 21, 22, 23 and 24 illustrate robustness of trained models to brightness, contrast, gamma and color balance transformations respectively. Specifically the test part of relevant dataset is first passed through a transform, the network then classifies transformed images and accuracy of prediction is measured. Matrices contain accuracies expressed in percentages. The dataset models have been trained on is written above each matrix; "CIFAR10 AUG" stands for CIFAR10 with augmentation scheme described above.

In 23 and 24 images are converted precisely according to $\mathcal{G}$ (equation 19) and $\mathcal{W}$ (equation 17) operators. $\mathcal{C}$ (equation 14) and $\mathcal{B}$ (equation 16) symmetries are in turn additionally clipped to $[0; 1]$ range so that their results are still viewable images. There's a lot of data to analyze, but conclusions mostly carry over between datasets:

- Starting with base Plain model and ignoring the AUG matrices for the moment, we observe that radical changes in brightness are detrimental to network's performance, where darkening image is significantly more impactful than brightening. This might be consequence of clipping – lowering brightness causes mean value of input to decrease linearly, whereas increasing causes only sublinear growth. The same phenomenon occurs in case of contrast and color balance changes. Bare Bspline models – RotEq, ScaleEq, ShearEq, GammaEq – behave essentially the same, though exact accuracies on the edges of matrices vary. Usually of all 5 networks, Plain performs the best.

- CBW and GBW layers work mostly just as their theoretical invariance properties suggest. Accuracy of InCBW and InGBW models doesn't decrease when lowering brightness and is still higher than their bare counterparts' when brightness increases. Invariance to color balance change is perfect expect for $T = 1000K$ which zeros out the blue channel which causes major drop in accuracy. For contrast change, CBW layer acts the same as for brightness change. Curiously the GBW also provides some degree of invariance to contrast, though it's not obvious from it's definition. It's especially visible when contrast is lowered. Similarly accuracy of InCBW models is only partially affected by gamma correction.

- BrightnessEq and InBk networks' accuracy doesn't change in low brightness. They are also partially invariant to other transforms but not as much as InCBW and InGBW models. At that InB1 performs visibly better than InB2 which in turn performs better than InB3 – the earlier placed the invariant layer, the better they accuracy preserved.

- Gamma Eq in turn performs a lot worse on heavily gamma-corrected images.

- Finally the data augmentation makes all the networks generalize better under brightness, contrast and gamma transforms. Given the character of augmentation the first two are not surprising, but the third is rather surprising. The difference is quite significant – for the $\mathcal{G}_0.333$ accuracy of Plain model goes up from 57% to 76%. It's especially surprising considering that images of lowered gamma

appear brighter than originals, while augmentation only supports lowering the brightness.

- Overall analyzing the CIFAR10 AUG matrices, very bright images seem to be the most challenging group of instances for all networks.

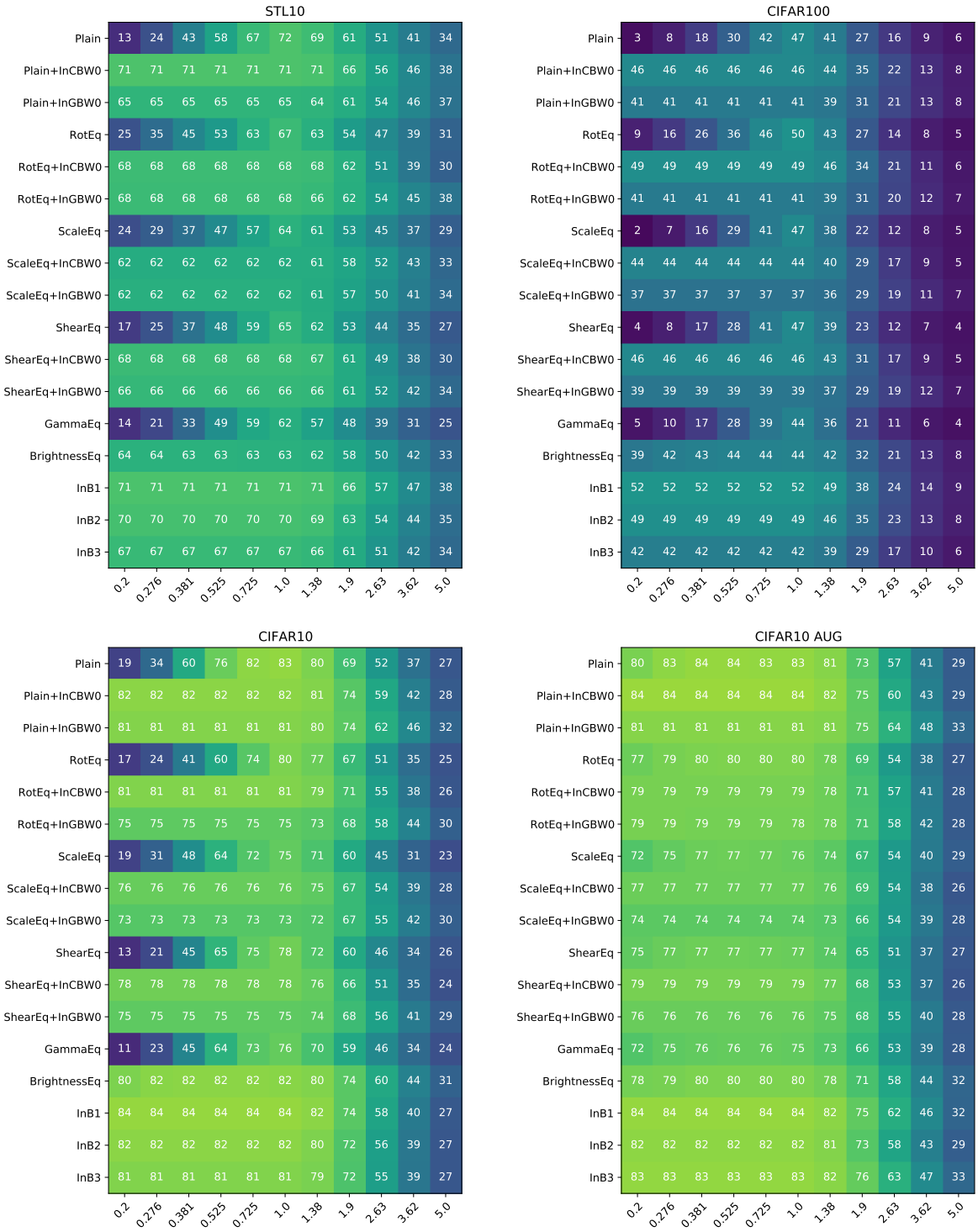


Figure 21: Classification accuracy of each model depending on the change of brightness $\mathcal{B}_a$ in input. X axis indicates value of parameter a.

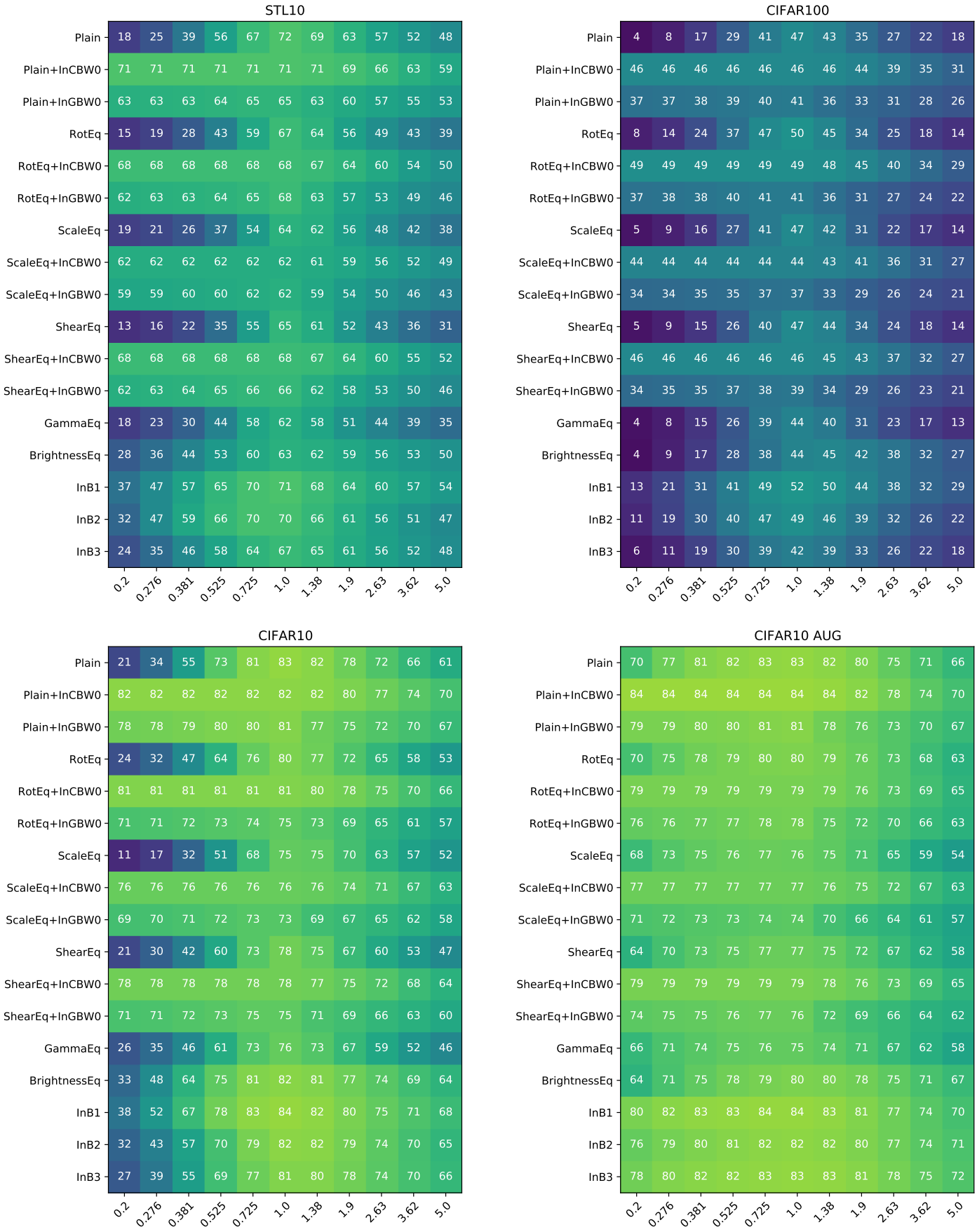


Figure 22: Classification accuracy of each model depending on the change of contrast $\mathcal{C}_c$ in input.

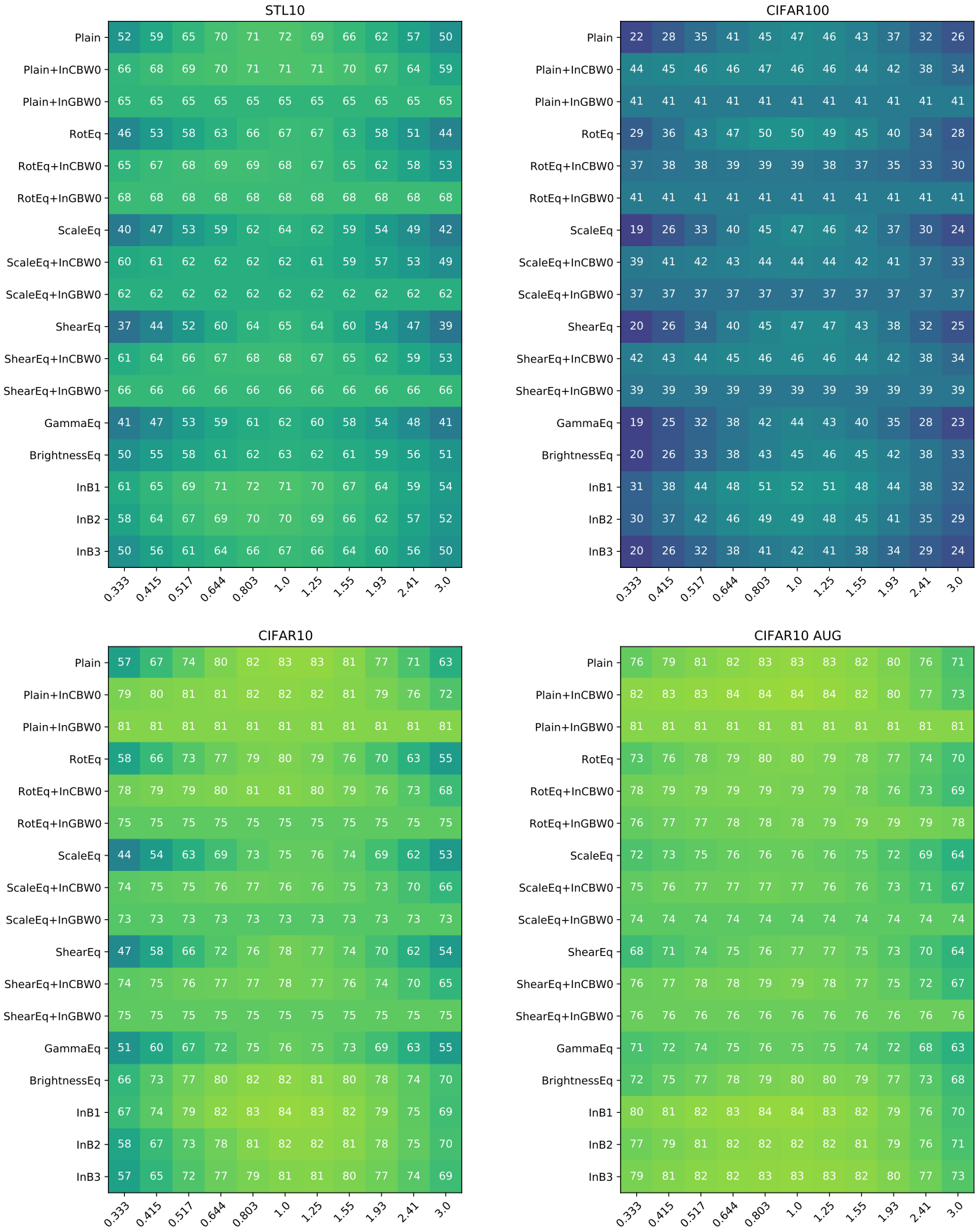


Figure 23: Classification accuracy of each model depending on the change of gamma $\mathcal{G}_c$ in input.

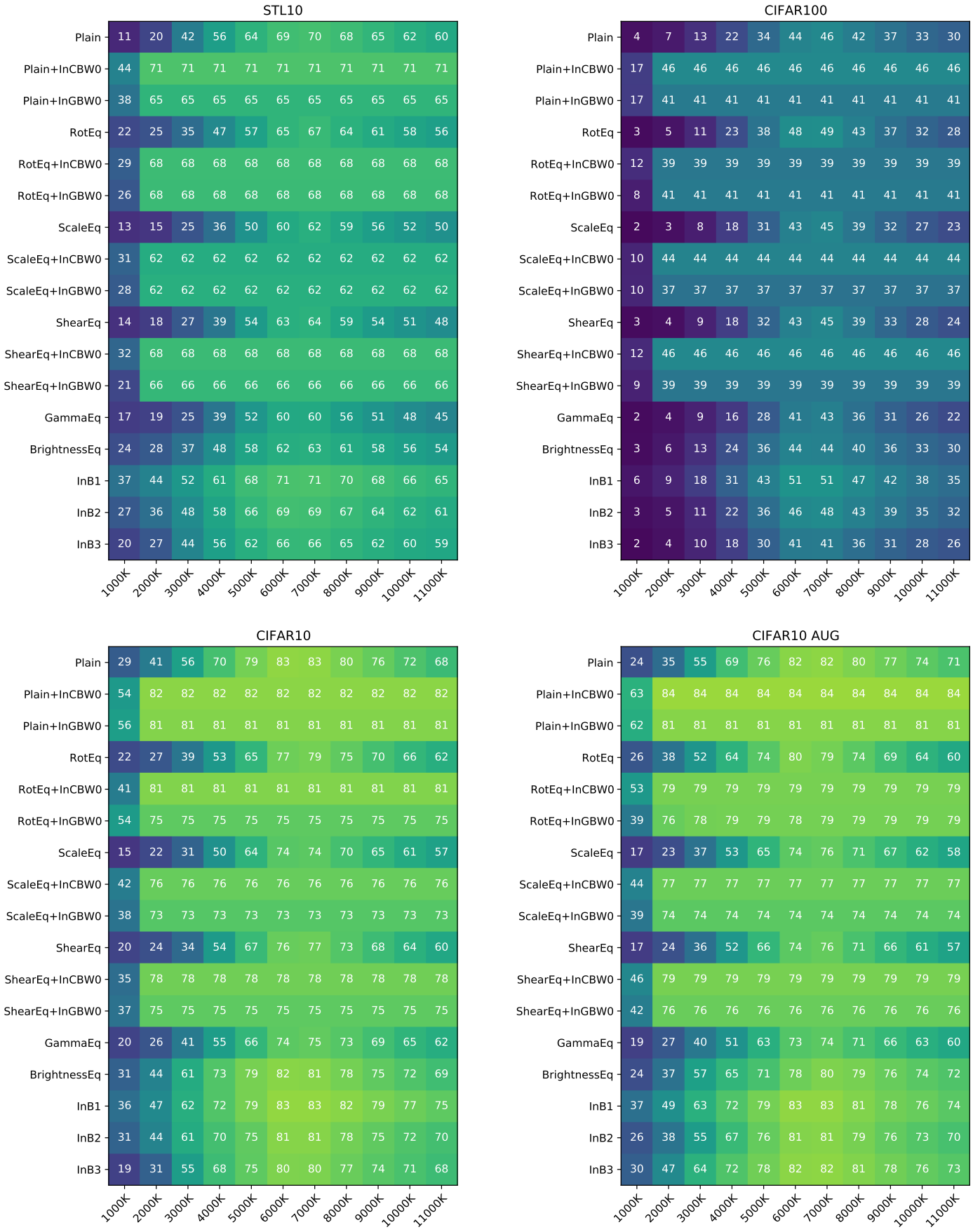


Figure 24: Classification accuracy of each model depending on the change of color balance $\mathcal{W}_T$ in input.

4.4 Degree of equivariance and invariance

Now we investigate degree of equivariance present at various depths of networks. While modern literature of GCNNs builds on thorough theoretical framework, claims of theoretical properties are hardly ever checked directly. [23] compares attention fields produced by equivariant attention mechanism visually. Of all papers cited in bibliography the only one containing numerical experiments is [24] where equivariance to scaling is measured. We reformulate slightly the definition of relative error used in [24] to make it symmetric. Let $\mathcal{N}_k$ be the kth layer of the network and $\mathcal{N}_{k:0} = \mathcal{N}_k \circ \cdots \circ \mathcal{N}_0$. The equivariance error of network $\mathcal{N}$ at depth k with respect to transformation $\mathcal{T}$ and input x equals

$$\mathcal{E}_{\mathcal{N}_k,\mathcal{T}}(x) = E\left[2\frac{|\mathcal{N}_{k:0}\left(\mathcal{T}(x)\right) - \mathcal{T}(\mathcal{N}_{k:0}(x))|}{|\mathcal{N}_{k:0}(\mathcal{T}(x))| + |\mathcal{T}(\mathcal{N}_{k:0}(x))|}\right]$$

$\mathcal{N}_{k:0}\left(\mathcal{T}(x)\right)$ as well as $\mathcal{T}(\mathcal{N}_{k:0}(x))$ are multidimensional tensors of the same shape and the division is performed entrywise, so its result is also a tensor which can be interpreted as an array of relative differences between $\mathcal{N}_{k:0}\left(\mathcal{T}(x)\right)$ and $\mathcal{T}(\mathcal{N}_{k:0}(x))$. In the end mean average value of this tensor is computed. The total error for dataset X is the average across all images in X:

$$\mathcal{E}_{\mathcal{N}_k,\mathcal{T}} = \frac{1}{|X|}\sum_{x\in X}\mathcal{E}_{\mathcal{N}_k,\mathcal{T}}(x)$$

Matrices in figures 25, 26, 27, 28, 29, show errors measured for networks trained on CIFAR10 dataset and multiplied by 100 – expressed in percentages. In each pair the left matrix shows error of base Plain model and the right one error of relevant Eq model. Labels at the bottom edge of matrix indicate depth at which equivariance error was measured. *conv1* and *lift* are the first convolutional layers, *norm1*, *ac1* and *maxpool1* are consecutive normalization, activation function and pooling right after the first convolution. Y axis shows factor by which input is transformed.

Overall we find errors of equivariance relatively high. The only model with fully equivariant layers is BrightnessEq, as it doesn't make any approximations. RotEq and ScaleEq make significant errors but they are visibly lower than Plain's. In the first one multiples of 90° are the most equivariant. This stems from lack of approximation in resampling and borders not being present in the image explicitly. In ScaleEq on the other errors are distributed very unevenly – the network seems to work better with zoomed-in pictures than with zoomed-out.

GammaEq and ShearEq in turn don't seem to be equivariant at all – its errors are the same or even worse than Plain model. High errors attained by networks at first glance seem to somewhat undermine sense of using such architectures. However in subsequent experiments we find that for geometric Eq models precise value of errors depends heavily on details of architecture – size of convolution kernels, stride, padding, etc. For each individual model errors can be decreased by large margin with these hyperparameters picked optimally. Such construction however would cause all models to have different architectures, sometimes even different number of layers. This in turn would render fair comparison of large number of networks within a single experiment questionable. In conclusion Lie GCNNs based on Bsplines, lifting and group convolu-

tion mechanisms can be made mostly equivariant with errors in deeper layers within $\approx 30\%$ level but each architecture should be adjusted individually in accordance to transformation group.

Regarding invariance, its error for layer $\mathcal{L}$ and transformation $\mathcal{T}$ is defined as

$$\mathcal{E}_{\mathcal{L},\mathcal{T}}(x) = 2\frac{|\mathcal{L}(\mathcal{T}(x)) - \mathcal{L}(x)|}{|\mathcal{L}(\mathcal{T}(x))| + |\mathcal{L}(x)|}$$

and again total error is average over whole dataset. Plots in figure 30 show error curves of CBW and GBW layers. Again brightness and contrast transformations are clipped to $[0; 1]$ range. It's worth noting that even though accuracy of InGBW models doesn't change under gamma transforms, the invariance error of GBW layer to change in gamma gets as high as 30%. Plots also reveal why GBW generalizes so well under change of contrast and CBW under change of gamma – the error curve of the first one seems to be bounded at the level of 60%, while the second grows rather gently.

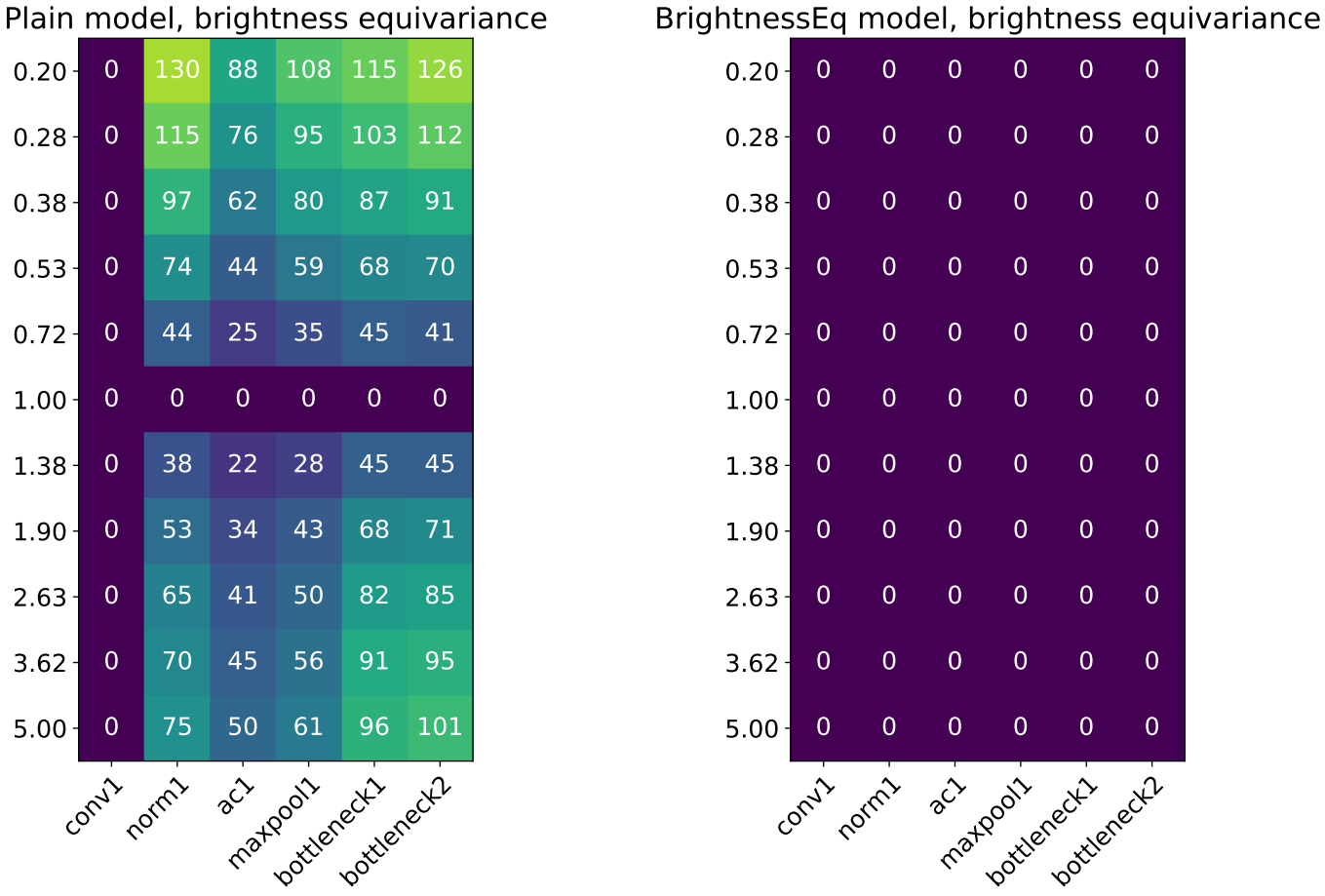


Figure 25: Equivariance error to brightness transform of Plain and BrightnessEq model.

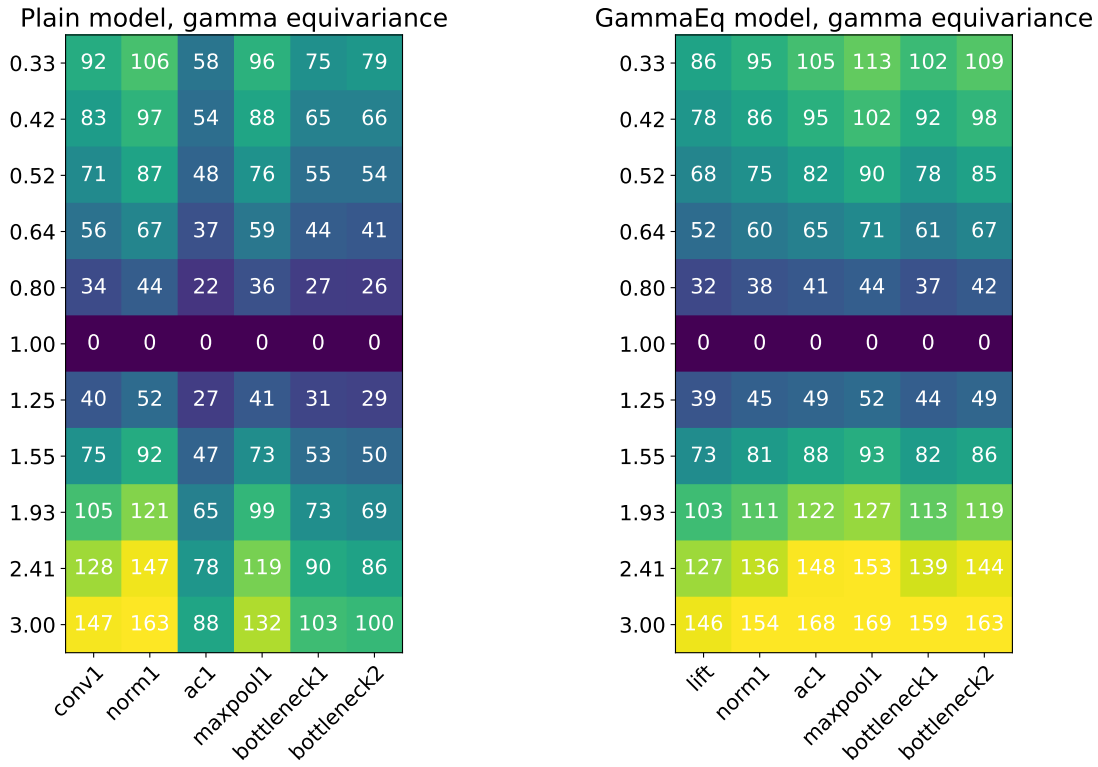


Figure 26: Equivariance error to gamma transform of Plain and GammaEq model.

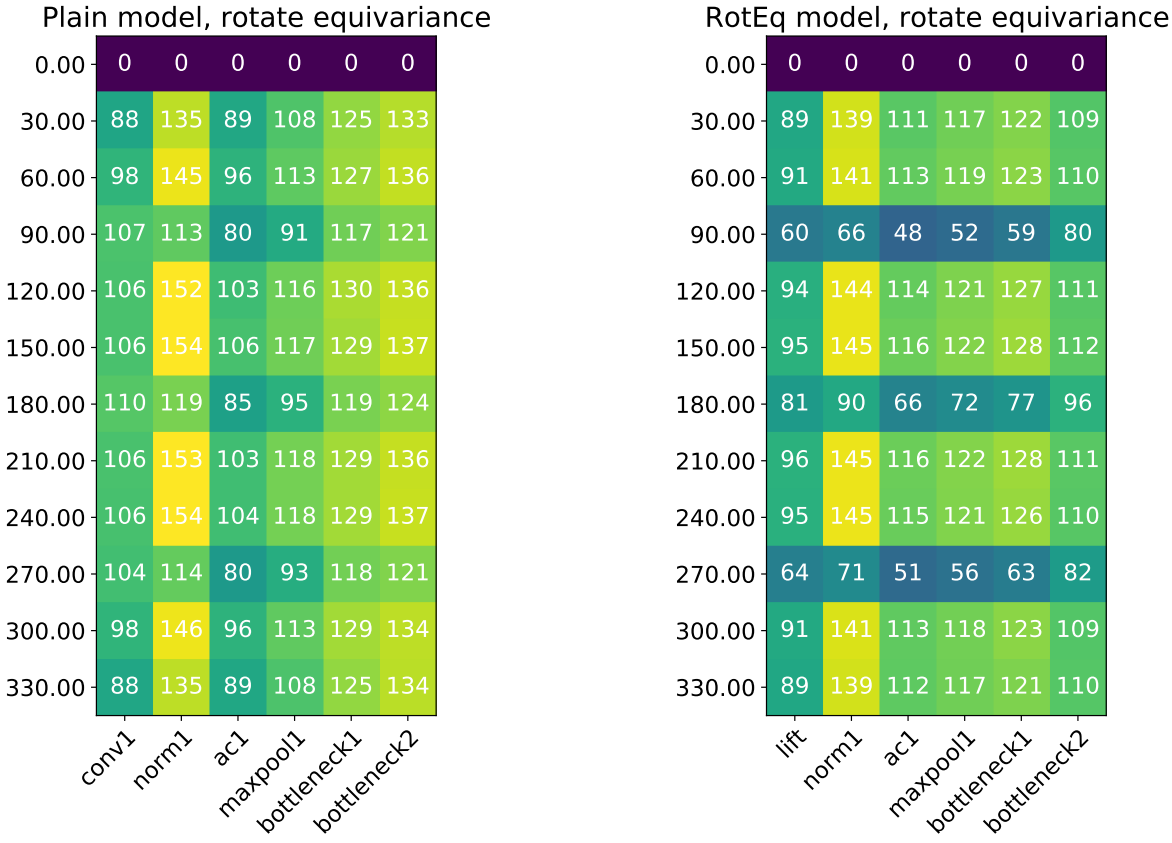


Figure 27: Equivariance error to rotations of Plain and RotEq model.

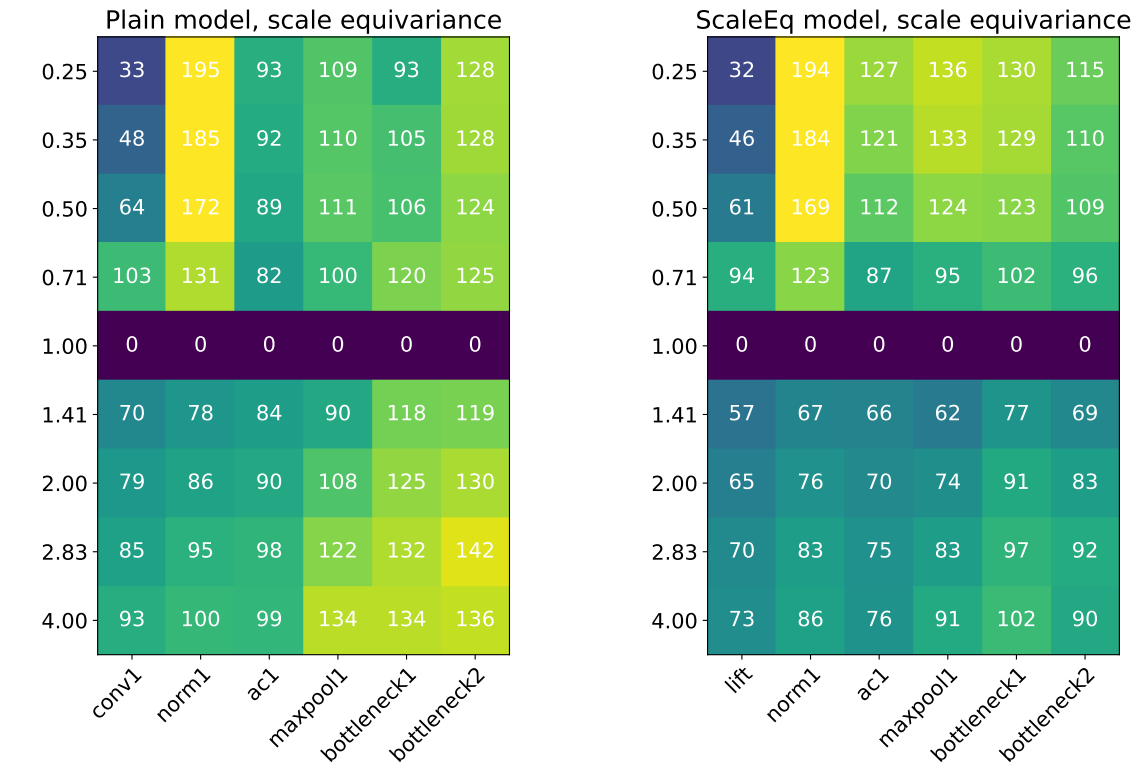


Figure 28: Equivariance error to scaling of Plain and ScaleEq model.

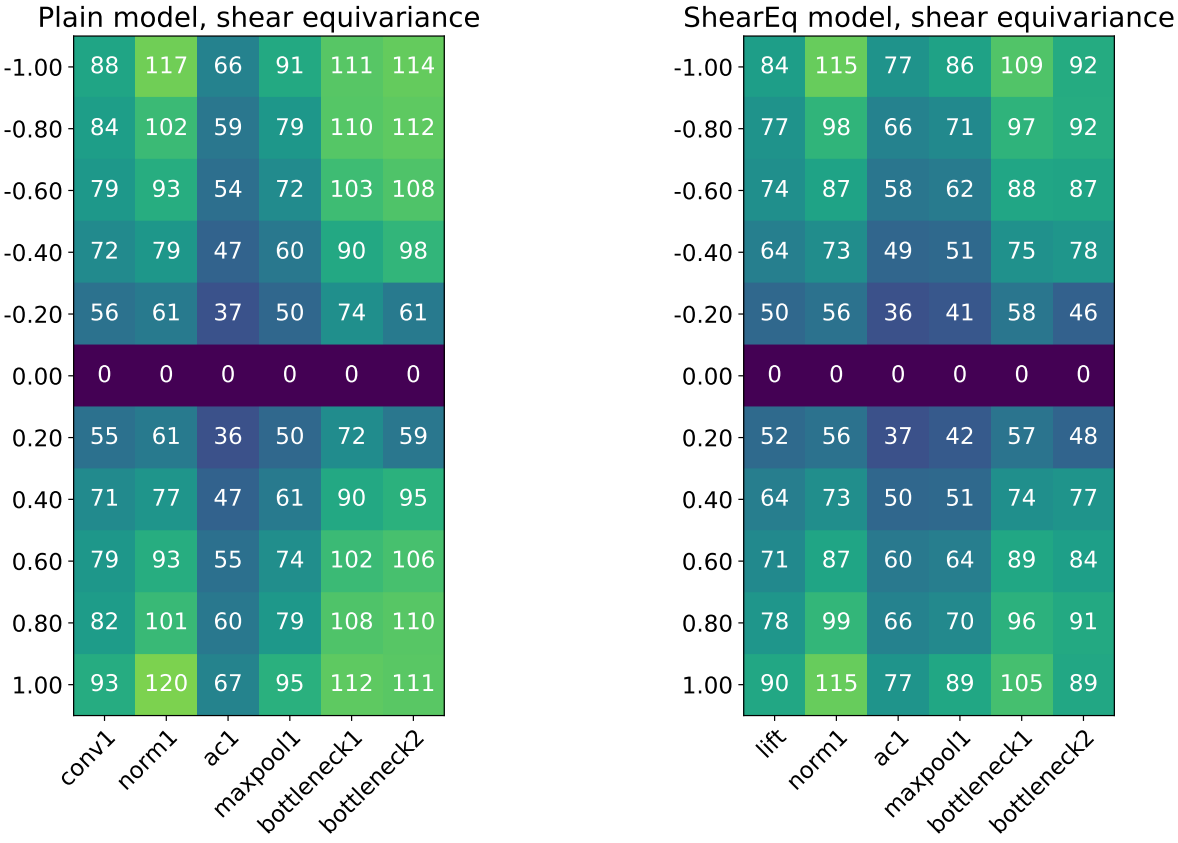


Figure 29: Equivariance error to shear transform of Plain and ShearEq model.

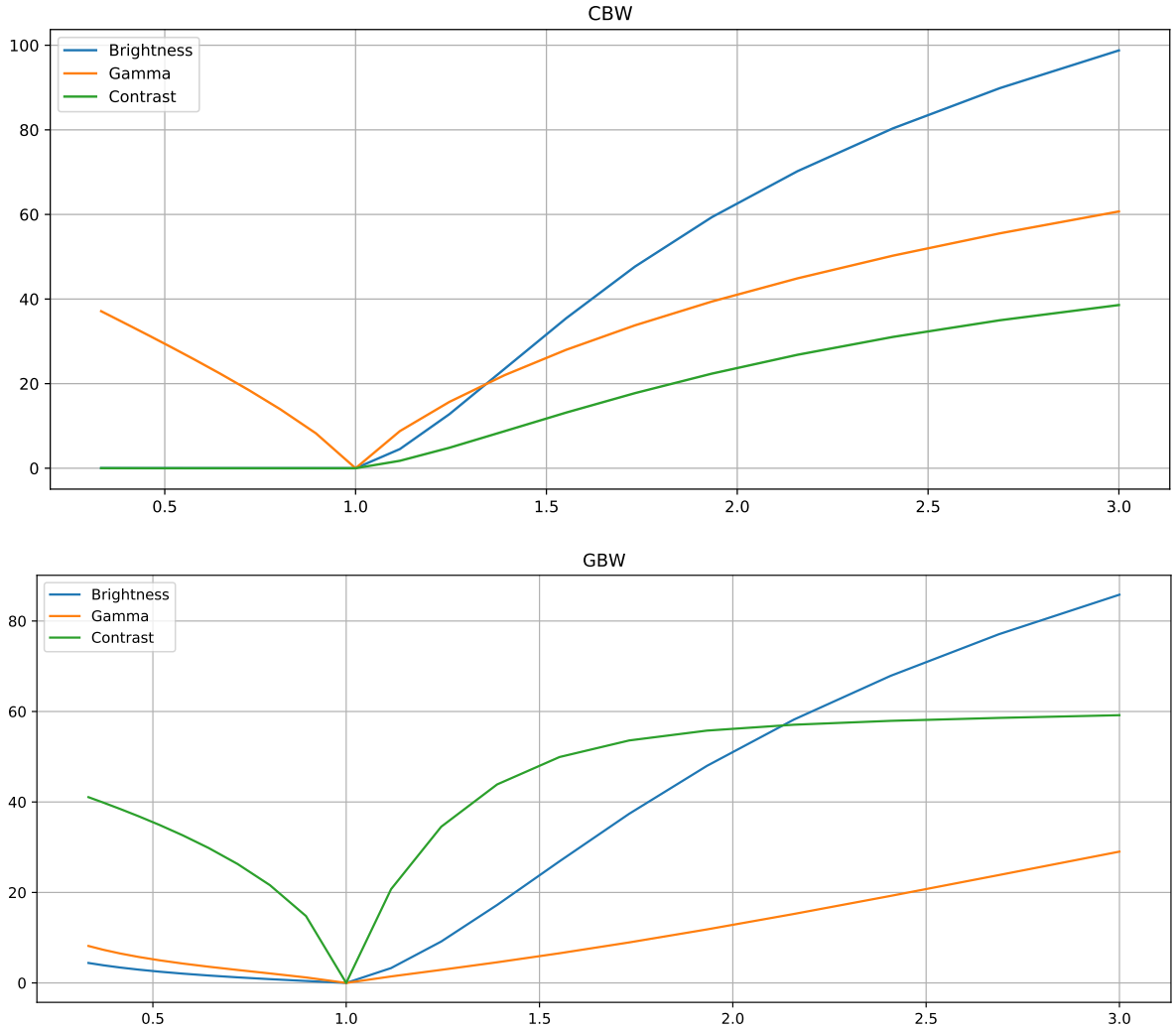


Figure 30: Invariance error to brightness, gamma and contrast transformations of CBW and GBW layers.

5 Conclusions & future work

Regarding the main objectives of this work we come to the following conclusions:

- In general networks equivariant to lightning-related transformations are for now hard to construct. The theoretical framework used in contemporary works doesn't fit very well with non-geometric symmetries. Even when it can be adapted in some way, it still makes heavy approximations, as lightning-like transforms tend to have structure of noncompact groups – either $(\mathbb{R}, +)$ or $(\mathbb{R}_+, \times)$.
- The only exception from this rule seem to be very simple transformations like brightness and perhaps some of its possible generalizations similar to color balance. Networks equivariant to these symmetries can be after some minor modifications realized relatively easily with traditional deep learning components.
- When speaking of lightning symmetries, using invariant layers either as the first layer of network or if possible after a couple equivariant layers seems to be the better choice:
  - They can be relatively easily found for many mathematically nontrivial transformations. While precise algorithm involving differential equations might be in general hard to construct, invariant layers seem to be built easily by alternating in some way one of standard forms of normalization.
  - They are preferable from computational point of view as they don't require any memory. Computational overhead is also bound just to single layer instead of being spread out across whole network which eases optimization.
  - For typical computer vision tasks like detection, segmentation or classification, it's the invariance that's required, not equivariance. In case of detection we don't really care how e.g. bright the output image is, as long as the bounding boxes are in the right places. And in case of classification or segmentation there isn't any output brightness to speak of.
- This stands in sharp contrast with geometric transforms. In their case constructing invariant layers placed at the beginning of network would require solving yet another computer vision problem. For example in case of rotations it would mean detecting relevant objects and estimating their rotation with respect to the ground or some other relevant object.
- While the evidence for any of tested invariant layers increasing accuracy on test part of datasets is vague at best, they without a doubt help generalize to out of distribution instances. Even though their numerically measured invariance errors are significant, experiments show accuracy is unchanged under various transformations. Though the layers are not exactly invariant, they perhaps still preserve overall structure of signals which aids generalization.
- In the same way contrast and brightness augmentation don't seem to make any difference on base test dataset, but clearly help generalize. Augmentation is worth applying even when invariant layers are used.

Possible future work includes:

- Exploring different transformations, like for example many types of possible contrast definitions.
- Devising algorithm for finding layers invariant to given group of transformations either based on its analytic description or numerical data.
- Designing broader equivariance and invariance measures taking into account structure preservation instead of just simple relative distances.
- Deriving analytic expressions of errors made by GBW layer for contrast changes and by CBW layer for gamma changes.
- With regard to geometric Lie GCNNs investigating more closely influence of network's architecture details on degree of equivariance is needed.

References

[1] URL: https://pytorch.org/vision/stable/transforms.html#torchvision.transforms.functional.adjust_contrast.

[2] URL: https://www.tensorflow.org/api_docs/python/tf/image/adjust_contrast.

[3] URL: https://imgaug.readthedocs.io/en/latest/source/api_augmenters_color.html#imgaug.augmenters.color.ChangeColorTemperature.

[4] URL: https://scikit-image.org/docs/stable/api/skimage.exposure.html#adjust-gamma.

[5] URL: https://docs.opencv.org/4.5.2/d3/dc1/tutorial_basic_linear_transform.html.

[6] URL: https://pytorch.org/vision/stable/transforms.html#torchvision.transforms.functional.adjust_gamma.

[7] Erik J Bekkers. *B-Spline CNNs on Lie Groups*. 2021. arXiv: 1909.12057 [cs.LG].

[8] Gregory Benton et al. *Learning Invariances in Neural Networks*. 2020. arXiv: 2010.11882 [cs.LG].

[9] Adam Coates, Andrew Ng, and Honglak Lee. "An Analysis of Single Layer Networks in Unsupervised Feature Learning". In: *AISTATS*. https://cs.stanford.edu/~acoates/papers/coatesleeng_aistats_2011.pdf. 2011.

[10] Taco Cohen, Mario Geiger, and Maurice Weiler. *A General Theory of Equivariant CNNs on Homogeneous Spaces*. 2020. arXiv: 1811.02017 [cs.LG].

[11] Taco S. Cohen and Max Welling. *Group Equivariant Convolutional Networks*. 2016. arXiv: 1602.07576 [cs.LG].

[12] Taco S. Cohen et al. *Spherical CNNs*. 2018. arXiv: 1801.10130 [cs.LG].

[13] D.S. Dummit and R.M. Foote. *Abstract Algebra*. Wiley, 2003. ISBN: 9780471433347. URL: https://books.google.pl/books?id=KJDBQgAACAAJ.

[14] Carlos Esteves et al. *Learning SO(3) Equivariant Representations with Spherical CNNs*. 2018. arXiv: 1711.06721 [cs.CV].

[15] Marc Finzi et al. *Generalizing Convolutional Neural Networks for Equivariance to Lie Groups on Arbitrary Continuous Data*. 2020. arXiv: 2002.12880 [stat.ML].

[16] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[17] Michael Hutchinson et al. *LieTransformer: Equivariant self-attention for Lie Groups*. 2021. arXiv: 2012.10885 [cs.LG].

[18] Risi Kondor and Shubhendu Trivedi. *On the Generalization of Equivariance and Convolution in Neural Networks to the Action of Compact Groups*. 2018. arXiv: 1802.03690 [stat.ML].

[19] Alex Krizhevsky. *Learning multiple layers of features from tiny images*. Tech. rep. 2009.

[20] J.M. Lee. *Introduction to Smooth Manifolds*. Graduate Texts in Mathematics. Springer New York, 2013. ISBN: 9780387217529. URL: https://books.google.pl/books?id=w4bhBwAAQBAJ.

[21] Ilya Loshchilov and Frank Hutter. *Decoupled Weight Decay Regularization*. 2019. arXiv: 1711.05101 [cs.LG].

[22] David W. Romero and Mark Hoogendoorn. *Co-Attentive Equivariant Neural Networks: Focusing Equivariance On Transformations Co-Occurring In Data*. 2020. arXiv: 1911.07849 [cs.CV].

[23] David W. Romero et al. *Attentive Group Equivariant Convolutional Networks*. 2020. arXiv: 2002.03830 [cs.CV].

[24] Ivan Sosnovik, Michał Szmaja, and Arnold Smeulders. *Scale-Equivariant Steerable Networks*. 2020. arXiv: 1910.11093 [cs.CV].

[25] Daniel E. Worrall and Max Welling. *Deep Scale-spaces: Equivariance Over Scale*. 2019. arXiv: 1905.11697 [cs.LG].

List of Figures

List of Tables