CLEI ELECTRONIC JOURNAL, VOLUME 24, NUMBER 1, PAPER 8, APRIL 2021

Invariance and Same-Equivariance Measures for Convolutional Neural Networks

Facundo Manuel Quiroga
Universidad Nacional de La Plata, Facultad de Informática,
Instituto de Investigación en Informática III-LIDI
La Plata, Argentina, 1900
fquiroga@lidi.info.unlp.edu.ar

Abstract

Neural networks are currently the state-of-the-art for many tasks.. Invariance and same-equivariance are two fundamental properties to characterize how a model reacts to transformation: equivariance is the generalization of both. Equivariance to transformations of the inputs can be necessary properties of the network for certain tasks. Data augmentation and specially designed layers provide a way for these properties to be learned by networks. However, the mechanisms by which networks encode them is not well understood.
We propose several transformational measures to quantify the invariance and same-equivariance of individual activations of a network. Analysis of these results can yield insights into the encoding and distribution of invariance in all layers of a network. The measures are simple to understand and efficient to run, and have been implemented in an open-source library.
We perform experiments to validate the measures and understand their properties, showing their stability and effectiveness. Afterwards, we use the measures to characterize common network architectures in terms of these properties, using affine transformations. Our results show, for example, that the distribution of invariance across the layers of a network has well a defined structure that is dependent only on the network design and not on the training process.

Keywords: Neural Networks, Equivariance, Invariance, Same-Equivariance, Transformations, Convolutional Neural Networks, CNN, measures

1 Introduction

Neural networks are currently the state of the art for many problems in machine learning. In particular, convolutional neural networks (CNNs) provide outstanding results for several signal processing applications [1].

Nevertheless, both neural networks and CNNs have difficulty learning good representations when inputs appear transformed. As an example, classification of texture or star images generally requires invariance to rotation, scale and/or translation transformation [2, 3].

The properties of invariance and equivariance explain how a model reacts to transformations of its inputs. Understanding the invariance and equivariance of a network [4] or any system [5] can help to improve their performance and robustness. In this work, we present techniques to measure invariance in neural networks. Using these measures, we focus on CNNs to shed some light on how and where these models represent and learn invariance to affine transformations.

Typical neural networks rely on feed-forward architectures. Feed-forward networks exclusively composed of dense layers can approximate smooth functions given enough parameters. CNNs usually employ a series of convolutional layers followed by one or two dense layers. Convolutional layers by themselves are, by definition, more parameter efficient than dense layers and also translation equivariant, but are not invariant nor equivariant to other transformations [2].

However, these models do not learn invariances or equivariances to rotations or other sets of transformations when trained with the usual stochastic gradient descent based algorithms and without data augmentation [6, 7]. Therefore, there is a need to understand how to obtain neural network models which can provide invariance to sets of transformations.

Recently, models such as Deep Symmetry Networks [8] or Steerable CNNs [9] were proposed to provide convolutional layers with rotation invariance or equivariance. These schemes were mostly based on modifying the filters so that they were invariant, or on employing a set of equivariant filters which were subsequently pooled to supply invariance [8, 9].

Other approaches such as Transformation-Invariant Pooling [10] made multiple predictions with several input versions generated via transformations to subsequently combine them via pooling. Alternatively, Spatial Transformer Networks [11] learned a canonical representation of the input by dynamically estimating an inverse affine transformation via a learned sub-network.

Data augmentation is usually employed to achieve partial invariance to geometric transformations of the input and improve generalization accuracy. Applied transformations are often mild, such as -15° to $+15^\circ$ rotations. However, full transformation invariance can be achieved by including an appropriate set of transformations. For example, for rotation transformations, a set formed via a dense sampling of the full range of angles.

This approach was studied for Deep Restricted Boltzmann Machines, HOGs and CNNs [6, 12, 13]. Although the employed architectures are simpler, they generally require more training epochs to explore the wide space of transformed inputs. Thus, given sufficient computational budget, typical CNNs with data augmentation could learn the same set of filters that other models include by design [13]. They can even learn arbitrary invariance and equivariance properties with heavy data augmentation [6]

However, some CNN models avoid dense layers entirely [14] while achieving performance similar to the state of the art, even when learning invariances with the help of data augmentation [15]. The reason behind this effect is that while traditional convolutional layers have a lower representational power than dense layers, and are not invariant individually, even with data augmentation, their output *can* be invariant or equivariant with respect to the input of an entire network composed of convolutions and activation functions [15].

In both cases, the network mechanisms to learn equivariant or invariant representations are not well-understood. Since most models require data augmentation during training, it is still unclear whether the model or the data augmentation provides the invariance [9–11]. Many proposed layers are individually invariant or equivariant, but no analysis was reported to understand how networks as a whole encode such properties. Several authors studied which methods work best to achieve invariance [6, 13], but no guiding principle in their analysis was employed except comparing the output accuracy. To better understand how these models acquire and encode invariance, we need better analysis tools such as invariance measures.

1.1 Invariance and Same-Equivariance Measures

Given a network and a set of transformations, invariance, same-equivariance and other related properties can be measured in different ways.

Analyzing if f is equivariant to a transformation t that operates on x requires finding a corresponding t' that operates on outputs [12]. A sufficient condition for the existence of t' is that f is invertible. Since CNNs are approximately invertible [12, 16], the existence of t' is very likely, at least in an approximate fashion. However, characterizing t' requires assuming its functional form and estimating its parameters [12]. Therefore, empirically analyzing the equivariance of a CNN can be difficult [12].

Measuring invariance does not require estimation of t'. Therefore, it is a more approachable property. Since invariance is a special case of equivariance in which t' is simply the identity transformation, we can exploit this special structure to measure invariance in simple and efficient ways. A similar approach can be applied for same-equivariance. In the following we will use the term equivariance to refer to invariance, same-equivariance or both.

Furthermore, equivariance can be measured in several locations of a network. The simplest measures quantify the equivariance of the final output of the network (ie. softmax layer for classification models) with respect to transformations of the input of the network such as images and rotations (figure 1 (a)). Alternatively, we can consider a single node or layer, and measure its equivariance with respect to its particular input and output, without taking into account the interactions with the rest of the network (figure 1 (b)), although in general these cases are simpler and can be handled analytically.

Given that most networks can be represented by an acyclic graph, we can generalize these notions by, for example, considering all intermediate values computed by the network as possible inputs or outputs. We will call these values *activations* of the network.

For example, we can consider all activations as outputs, and measure their equivariance with respect to the initial input to the network (figure 1 (c)). Alternatively, via topological sorting of the network graph, we can remove the first $k-1$ nodes or layers from consideration, and calculate the equivariance of the output of the network with respect to transformations of the input to node or layer k (figure 1 (d)). In summary, we can arbitrarily define a set of inputs to transform and a set of outputs where to evaluate equivariance or another such property.

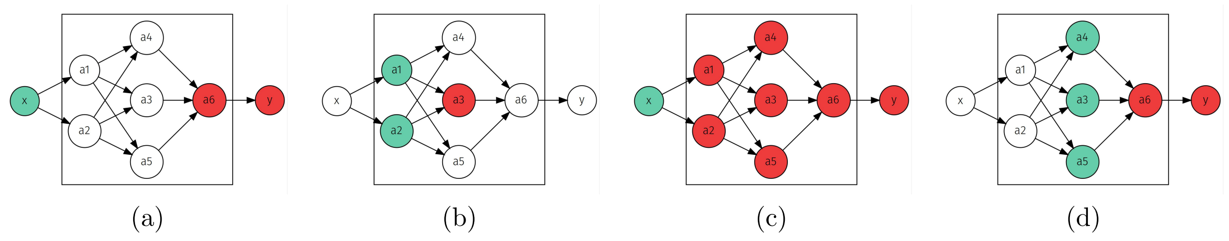


Figure 1: Different approaches for calculating equivariance in a neural network with activations $a_1, a_2, \dots a_6$. Green nodes indicate transformed inputs, and red nodes indicate where the equivariance is calculated. In (a), the equivariance of the final output $y = a_6(x)$ of the network is measured with respect to transformations of the initial input x. In (b), the equivariance of a single node or layer $a3$ is calculated with respect to its direct inputs a_1 an a_2. In (c), the equivariance of all nodes are calculated with respect to the input x. Finally, in (d) the equivariance of the output y is calculated with respect to the output of activations a_4, a_5 and a_6

1.2 Goals

Our main objective in this thesis is to contribute to the understanding and improvement of equivariance in neural network models. In terms of applications, we focus on handshape classification for sign language and other types of gestures using convolutional networks.

Therefore, we set the following specific goals:

1. Analyze CNN models design specifically for equivariance
2. Compare specific models and data augmentation as means to obtain equivariance. Evaluate transfer learning strategies to obtain equivariant models starting with non-equivariant ones.
3. Develop equivariance measures for activations or inner representations in Neural Networks. Implement those measures in an open source library. Analyze the measures behaviour, and compare with existing measures.
4. Characterize CNN models for image classification in terms of equivariance using the proposed measures.
5. Compare CNN models, with or without equivariance, for handshape classification.

Given the existence of multiple methods to achieve equivariance, and the lack of deep and rigorous comparisons among them, the scope of this work is limited to the analysis of the models and we therefore do not propose new equivariant network models.

1.3 Contributions

We present several contributions:

- A comparative analysis of Neural Network based models for sign language handshape classification.
- An analysis of strategies to achieve equivariance to transformations in neural networks:
  - Comparing the performance of strategies based on data augmentation and specially designed networks and layers.
  - Determining strategies to retrain networks so that they acquire equivariance.
- A set of measures to empirically analyze the equivariance of Neural Networks, as well as any other model based on latent representations, and the corresponding:
  - Validation of the measures to establish if they are indeed measuring the purported quantity.
  - Analysis of the different variants of the proposed measures.
  - Analysis of the properties of the measures, in terms of their variability to transformations, models and weight initialization.

- Analysis of the impact of several hyperparameters of the models on the structure of their equivariance, including Max Pooling layers, Batch Normalization, and kernel size.
- Analysis of the structure of the equivariance in several well known CNN models such as *ResNet All Convolutional* and *VGG*.
- Analysis of the impact on the equivariance of using specialized models to obtain equivariance such as *Transformational Invariance Pooling (TI-Pooling)*.
- Analysis of the class dependency of equivariance.
- Analysis of the effect of varying the complexity and diversity of the transformations on the measures.

1.4 Article organization

Given space constraints, this summary only details the main contributions of the thesis, which consists of the invariance and same-equivariance measures, and the consequent analysis of classical CNN models.

Section 2 presents the proposed equivariance measures. Section 3 analyzes some properties of the measures and employs them to characterize typical CNN models. Finally, Section 4 presents conclusions and future work.

2 Equivariance Measures

We present different types of measures:

- Variance-based invariance
- Distance-based invariance
- Variance-based same-equivariance
- Distance-based same-equivariance

These measures allow measuring equivariance in any model based on neural networks with high granularity, that is, in each *activation* or intermediate value computed by the network.

In this section, we define the measures along with a general framework for defining new transformational measures.

2.1 General framework

Our objective is to compute a measure of the activations of a model f with respect to a set of transformations T of its input x. In this case, our measures will be of invariance or same-equivariance, but they could also target another type of property which depends on transformations.

Given an input x, a neural network model contains computes many intermediate or hidden values, which we will call $a_1(x), \dots, a_k(x)$. In the interest of brevity, we will call such values $a_i(x)$ *activations* of the network f. This term is not to be confused with *activation functions* such as $ReLu$ or $TanH$. An activation *can* be the result of applying an activation function to a tensor, or simply the output of a convolutional or fully connected layer. Note that x always refers to the input of the whole network, and never to the input of an intermediate layer, as shown in Figure 2.

For example, let f be a two layered network with a convolutional layer followed by a ReLU activation functions and a fully connected layer. The output of a convolutional layer contains $H \times W \times C$ scalar *activations*. After applying the ReLU, we obtain another set of $H \times W \times C$ *activations*. By applying a flatten operation followed by a fully connected layer with D neurons to the output of the convolutional layer, we have another D *activations*. Therefore, there are $k = H * W * C + H * W * C + D$ activations in this network.

In this case, we have ignored the output of the flatten operation. In appropriate cases such as this, a subset of activations can be ignored for the computation of the measure, since their output is simply a reshape of other activations. In other cases, outputs such as perhaps the activations of a convolutional layer before the ReLU is applied can also be ignored to reduce the amount of information to analyse.

To measure the equivariance of a model f, we measure can measure the equivariance of the individual activations $a_1(x), \dots a_k(x)$. Since the measure can be defined for an activation independently of the rest, we focus on a single activation we will denote simply as $a(x)$.

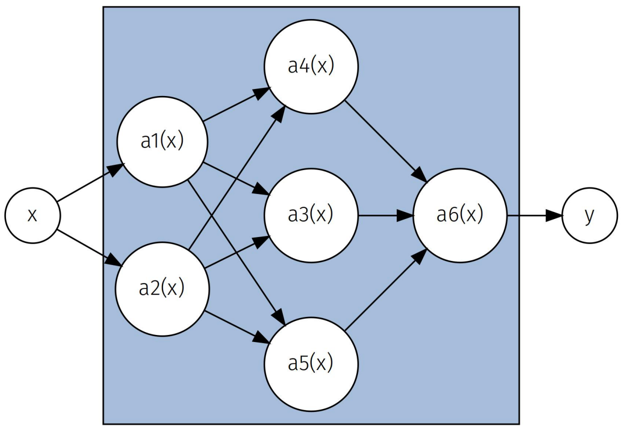
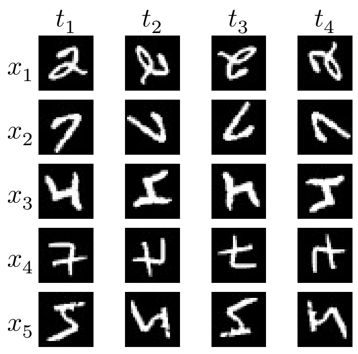


Figure 2: Diagram of a network f with its activations. Given an input x, the network computes its output $y = f(x)$, by calculating its activations $a_1(x), \dots, a_6(x)$. The final output value y is simply the value $a_6(x)$. Instead of considering each activation a_i as a function of the output of other activations that feed into it, the activation is viewed as a function of the original input x.



(a) Samples and transformations

$$\begin{array}{c|cccc}
 & t_1 & t_2 & t_3 & t_4 \\
\hline
x_1 & a(t_1(x_1)) & a(t_2(x_1)) & a(t_3(x_1)) & a(t_4(x_1)) \\
x_2 & a(t_1(x_2)) & a(t_2(x_2)) & a(t_3(x_2)) & a(t_4(x_2)) \\
x_3 & a(t_1(x_3)) & a(t_2(x_3)) & a(t_3(x_3)) & a(t_4(x_3)) \\
x_4 & a(t_1(x_4)) & a(t_2(x_4)) & a(t_3(x_4)) & a(t_4(x_4)) \\
x_5 & a(t_1(x_5)) & a(t_2(x_5)) & a(t_3(x_5)) & a(t_4(x_5))
\end{array}$$

(b) **ST** matrix

Figure 3: (a) Matrix of samples and their corresponding transformations, for $n = 5$ samples and $m = 4$ transformations. (b) Corresponding **ST** matrix containing the activation values corresponding to each input for a single activation a.

2.2 Sample-Transformation activation matrix (ST)

We define Sample-Transformations activation matrices (**ST**) (Figure 3), which provide the main context and notation for the definition of the transformational measures.

Given an activation a, a set of n samples $X = [\, x_1 \cdots x_n \,]$ and a set of m transformations $T = [\, t_1 \cdots t_m \,]$ defined over X, we can compute the value of a for all the possible transformations of the samples.

Let $x_{i,j} = t_j(x_i)$ be the sample obtained by applying transformation $t_j \in T$ to input $x_i \in X$. Given a, we define the sample-transformations activation matrix $\mathbf{ST}(a, X, T)$ of size $n \times m$ as follows:

$$\mathbf{ST}(a, X, T)_{i,j} = a(x_{i,j}) = a(t_j(x_i)) \qquad [1]$$

For simplicity, we will refer to $\mathbf{ST}(a, X, T)$ as $\mathbf{ST}(a)$ or simply **ST** whenever the context clearly determines a or X and T. Note that **ST** resembles the matrix of observations employed in a one-way ANOVA, where each transformation can be considered as a different *treatment*.

For a network f with k different activations, there are k associated **ST** matrices, which together form an $m \times n \times k$ cube (Figure 4)

2.2.1 *Efficient computation of the sample-transformations matrix* ST

The efficient computation of **ST** is crucial for practical computation of the measures. Given a single transformed input only a forward pass of the network is required to obtain the activations $a_1, \dots, a_k$, without

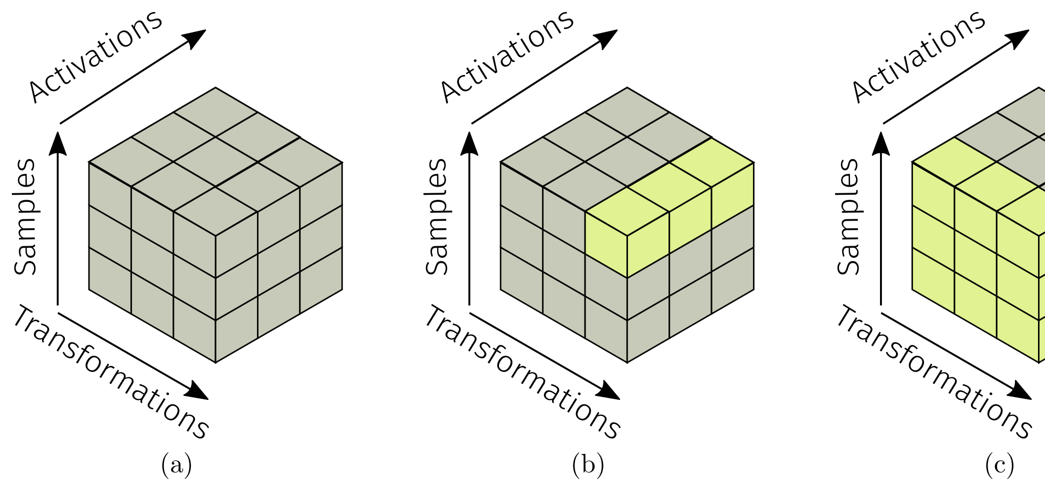


Figure 4: a) Cube of activation values for a network. b) The result of a forward pass for a single sample and transformation. c) Slice of the activations cube that corresponds to the **ST** a single output.

calculating gradients. For a given batch of inputs, we can also obtain all the activations of the network in a single pass. However, these activations represent only a small subset of each of the values of each of the k **ST** matrices. Therefore, there is a mismatch between the way neural network activations are usually computed and the order we require to efficiently compute the measures. In terms of the activation cube (Figure 4), the forward pass naturally iterates first over the activations dimension, while computation of the measure of each activation naturally requires iterating first over the samples and transformations dimensions.

It is possible to compute the **ST** matrix corresponding to a *single* activation a by looping over all $n \times m$ transformed examples in a batched fashion and discarding all other activations values. This would correspond to computing a slice of the **ST** cube (Figure 4), choosing a single index in the Activations dimension.

However, if all **ST** matrices are needed repeating this process over all k activations is prohibitive. In a network with k activations, there are k corresponding **ST** matrices and k can be very large. For example, ResNet18 (the smallest of the ResNet model family) produces on the order of 3 million activations when using images of 224×224 resolution as input [17].

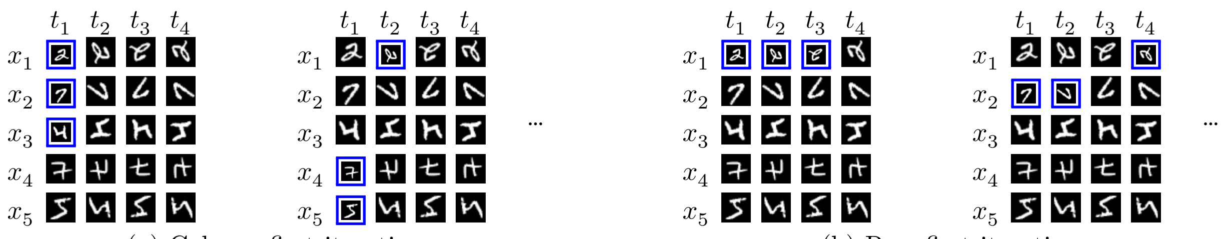
Another possibility is to generate and store all k **ST** matrices without repeating forward passes, that is, the entire **ST** cube (Figure 4). While m, the number of transformations, is usually small, the number of samples n can be large as well. Therefore, storing all k **ST** matrices can be limiting or even impossible, requiring memory for $n \times m \times k$ floating-point scalar numbers. For example, using the same ResNet18 model mentioned before, $n = 1000$ samples ([18] presents evidence on the need for this number of samples), $m = 16$ transformations and single-precision floating point numbers to store the activations, all the **ST** matrices would require $\frac{k \times m \times n \times 4}{2^{20}} = 178$ GB of storage, which is difficult to fit in RAM, let alone a GPU. Storing the activations in disk and then analyzing them is also possible but very demanding in terms of storage, specially when performing various experiments.

Alternatively, all measure computations that use **ST** matrices must be implemented in a running or online fashion, looping over the samples and transformations either in a row first (transformations first) or column first (samples first) fashion, as shown in Figure 5. With this strategy, we can calculate activations as shown in Figure 6, and compute the measure for all the activations in a parallel fashion. In this way, the memory requirements are kept constant while utilizing the full efficiency of the forward pass.

Given the nature of most neural networks and tensor computation models and frameworks, looping over the *st* matrices to implement a measure is not straightforward. Also, the computation of the forward pass must be done in batches, which brings a further complication. This makes it difficult to implement new measures. Therefore, we have developed an open-source library[1] that given a model, a set of samples (a dataset) and a set transformations, allows looping over these matrices in an efficient fashion. The library simplifies implementation of the measures, and also includes implementations of the measures described below.

In the following subsections, we define the equivariance measures based on the **ST** matrix. We define three types of measures, each based on different concepts. The ANOVA measure (section 2.3) uses the traditional analysis of variance procedure to determine if an activation is invariant or not assuming the various transformations are similar to treatments in an ANOVA setting. Variance-based measures (section

[1]Transformational Measures library: `https://github.com/facundoq/transformational_measures`



(a) Column-first iteration (b) Row-first iteration

Figure 5: Inputs for the batched iteration over the **ST** matrix, using *batchsize* = 3, $n = 5$ samples and $m = 4$ transformations.

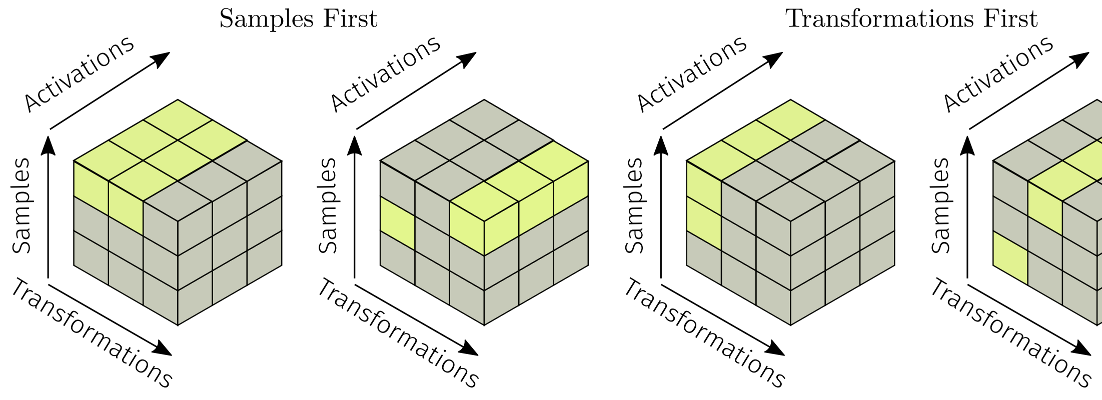


Figure 6: Outputs for the batched iteration over the **ST** matrix, using *batchsize* = 2, $n = 3$ samples and $m = 3$ transformations, for a network with $k = 3$ activations. The first two images correspond to a samples first iteration, while the others represent a transformation first iteration.

2.4) use the common sample variance or standard deviation of each activation to quantify its equivariance. Distance-based measures (section 2.5) compare the distance between activations to quantify how much they change under transformation of the inputs.

2.3 ANOVA Measure

The Analysis of Variance (ANOVA) is a statistical hypothesis testing method [19]. It is used to analyze samples of different groups. ANOVA can establish if the means for different groups of samples, called treatments, are statistically similar. While ANOVA is a parametric method, it has mild assumptions and is robust to violations of normality, specially with large sample sizes such as those available for machine learning datasets [19].

The matrix of observations used in one-way ANOVA contain n rows and m columns; each row corresponds to a sample to which m treatments have been applied independently. Given that the **ST** matrix is very similar to the matrix of observations in one-way ANOVA, we can adapt the interpretation of the method for invariance testing. The null hypothesis in ANOVA is that all means are the same for the different groups. If the different groups correspond to different transformations, the null hypothesis is equivalent to invariance in a statistical interpretation. If the null hypothesis is rejected, then the activation is not invariant.

We define the **ANOVA** measure (AM) simply as the application of the one-way ANOVA procedure to the **ST** matrix of each activation, independently. Therefore, the only parameter of the measure is the significance value of the test, α. As mentioned in Section 2.2 the number of activations in a neural network is quite large. Therefore, we must apply a Bonferroni correction [19] to α to account for the large number of corresponding hypothesis tests.

The choice of invariance as null hypothesis can be considered strange, since in general it is a property models are not assumed to have a priori. However, in many cases the models that are being measured have been trained or designed for invariance. Therefore we argue that while both approaches have their merits, using invariance as the null hypothesis can be more appropriate in many cases.

The ANOVA measure makes the assumption that, for each transformation, the corresponding activation of all transformed samples is unimodal. Therefore, it expects that each activation responds in a similar fashion for different samples, even when these are very different, even of different classes. This limitation

$$\begin{array}{cc} & \rightarrow \text{(1) Var} \\ \text{(2) Mean} \downarrow & \begin{bmatrix} a(t_1(x_1)) & a(t_2(x_1)) & a(t_3(x_1)) & a(t_4(x_1)) \\ a(t_1(x_2)) & a(t_2(x_2)) & a(t_3(x_2)) & a(t_4(x_2)) \\ a(t_1(x_3)) & a(t_2(x_3)) & a(t_3(x_3)) & a(t_4(x_3)) \\ a(t_1(x_4)) & a(t_2(x_4)) & a(t_3(x_4)) & a(t_4(x_4)) \\ a(t_1(x_5)) & a(t_2(x_5)) & a(t_3(x_5)) & a(t_4(x_5)) \end{bmatrix} \end{array} \implies Mean\left(\begin{bmatrix} Var(\begin{bmatrix} a(t_1(x_1)) & a(t_2(x_1)) & a(t_3(x_1)) & a(t_4(x_1)) \end{bmatrix}) \\ Var(\begin{bmatrix} a(t_1(x_2)) & a(t_2(x_2)) & a(t_3(x_2)) & a(t_4(x_2)) \end{bmatrix}) \\ Var(\begin{bmatrix} a(t_1(x_3)) & a(t_2(x_3)) & a(t_3(x_3)) & a(t_4(x_3)) \end{bmatrix}) \\ Var(\begin{bmatrix} a(t_1(x_4)) & a(t_2(x_4)) & a(t_3(x_4)) & a(t_4(x_4)) \end{bmatrix}) \\ Var(\begin{bmatrix} a(t_1(x_5)) & a(t_2(x_5)) & a(t_3(x_5)) & a(t_4(x_5)) \end{bmatrix}) \end{bmatrix}\right)$$

Figure 7: Calculation of the Transformation Variance measure for $n = 5$ samples and $m = 4$ transformations. First, (1) the variance of each row (over transformations) is calculated; then, (2) the mean of each column (over samples).

may reduce the utility of the measure by rejecting the null hypothesis unnecessarily.

The computation of the ANOVA requires two iterations over the **ST** matrix. The first iteration computes the means for each treatment/transformation; the second, the within groups sum of squares. Both iterations are performed iterating first over transformations and then over samples, that is, iterating over the rows of the **ST** matrix.

As will be seen in the experiments (Section 3), the ANOVA measure is very sensitive to violations of invariance and therefore not very useful for measuring that property. However, it does serve as a baseline and first approach to measuring invariance.

2.4 Variance-based Invariance Measures

Variance-based invariance measures are based on calculating the variance of each activation. The variance Var is a function with range $[0, \infty]$. Therefore, we can consider an activation as invariant if its variance is 0.Values greater than 0 indicate different degrees of lack of invariance. We measure two different sources of variance, Transformation Variance and Sample Variance, each computed by varying the transformations and samples, respectively. Afterwards, these two variances are combined to obtain a Normalized Variance measure. The following section describes the three measures and their relationship.

2.4.1 Transformation Variance Measure

The **Transformation Variance** (TV) of an activation a is defined as the mean of the variance of each row in the **ST** matrix (Equation [2]).

$$TV(a) = Mean\left(\begin{bmatrix} Var(\mathbf{ST}(a)[1,:]) \\ \cdots \\ Var(\mathbf{ST}(a)[n,:]) \end{bmatrix}\right) \qquad [2]$$

Where:

- $\mathbf{ST}(a)[i,:] = \begin{bmatrix} \mathbf{ST}(a)[i,1] & \cdots & \mathbf{ST}(a)[i,m] \end{bmatrix}$ is a vector containing row i of $\mathbf{ST}(a)$.
- $Var(\begin{bmatrix} x_1 & \cdots & x_n \end{bmatrix}) = \frac{\sum_{i=1}^{n} x_i - \bar{x}}{n-1}$ is the standard sample variance defined over a vector of observations $\begin{bmatrix} x_1 & \cdots & x_n \end{bmatrix}$.
- $\bar{x} = Mean(\begin{bmatrix} x_1 & \cdots & x_n \end{bmatrix}) = \frac{\sum_{i=1}^{n} x_i}{n}$ is the standard sample mean.

Each row i of the **ST** matrix contains the activations for sample x_i and all transformations (Equation [1] and Figure 3). Therefore, the variance is computed over the activations for different transformations; and the mean over samples. Should the activation be completely invariant to T, all the values in each row would be equal and therefore the variance of each row would be 0. In this way, if an activation is invariant to transformations then its *transformational* variance is 0.

Figure 8 (a) shows the results of calculating the Transformation Variance as a heatmap. Each column of the heatmap corresponds to a layer. Within each layer/column, each cell corresponds to the measure value for a different activation. Each color corresponds to a different level of invariance. Values in green indicate outliers. Note that in general since the layers are arbitrary there is no guaranteed row-structure in the image; each layer/column can have a different number of activations and therefore rows. Furthermore, the vertical distance between two values of different columns does not carry any information. Nonetheless, layers corresponding to activation functions, for example, do have the same structure as the previous layer.

For layers whose output is a set of feature maps (Convolutional, MaxPooling, etc), we show only one value for each feature map. That value corresponds to the mean measure over the spatial dimensions.

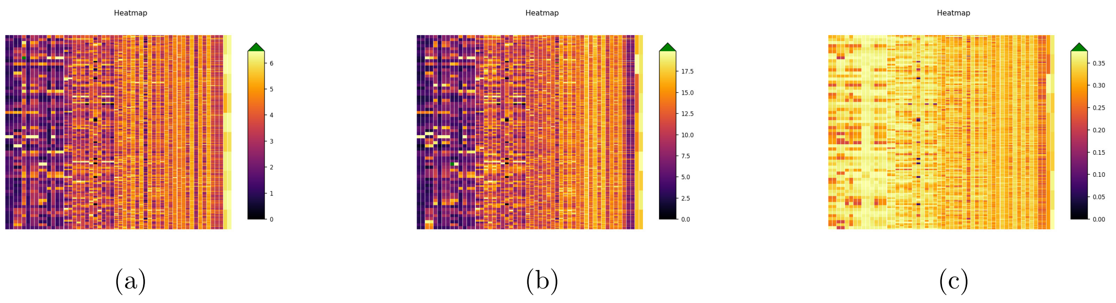


Figure 8: (a) Transformation Variance, (b) Sample Variance and (c) Normalized Variance of each activation of the ResNet model. The transformation set for this example is a set of 16 rotations distributed uniformly between 0° and 360°, and the dataset is the test set of CIFAR10.

$$\begin{array}{cc} & \rightarrow \text{(2) Mean} \\ \text{(1) Var} \downarrow & \begin{bmatrix} a(t_1(x_1)) & a(t_2(x_1)) & a(t_3(x_1)) & a(t_4(x_1)) \\ a(t_1(x_2)) & a(t_2(x_2)) & a(t_3(x_2)) & a(t_4(x_2)) \\ a(t_1(x_3)) & a(t_2(x_3)) & a(t_3(x_3)) & a(t_4(x_3)) \\ a(t_1(x_4)) & a(t_2(x_4)) & a(t_3(x_4)) & a(t_4(x_4)) \\ a(t_1(x_5)) & a(t_2(x_5)) & a(t_3(x_5)) & a(t_4(x_5)) \end{bmatrix} \end{array} \Longrightarrow Mean\left(\left[Var\left(\begin{bmatrix} a(t_1(x_1)) \\ a(t_1(x_2)) \\ a(t_1(x_3)) \\ a(t_1(x_4)) \\ a(t_1(x_5)) \end{bmatrix}\right), Var\left(\begin{bmatrix} a(t_2(x_1)) \\ a(t_2(x_2)) \\ a(t_2(x_3)) \\ a(t_2(x_4)) \\ a(t_2(x_5)) \end{bmatrix}\right), Var\left(\begin{bmatrix} a(t_3(x_1)) \\ a(t_3(x_2)) \\ a(t_3(x_3)) \\ a(t_3(x_4)) \\ a(t_3(x_5)) \end{bmatrix}\right), Var\left(\begin{bmatrix} a(t_4(x_1)) \\ a(t_4(x_2)) \\ a(t_4(x_3)) \\ a(t_4(x_4)) \\ a(t_4(x_5)) \end{bmatrix}\right)\right]\right)$$

Figure 9: Calculation of the Sample Variance measure for $n = 5$ samples and $m = 4$ transformations. First, the variance of each column (over samples) is calculated; then, the mean of each row (over transformations).

Using Welford's running mean and variance equations [20], computing the Transformation Variance requires a single iteration through the rows of **ST**, and therefore the running time is $\mathcal{O}(k \times n \times m)$.

The Transformation Variance is measured in units of the activation a. When $TV(a) = 0$ the activation is invariant to transformations. However, if $TV(a) > 0$ there is no clear interpretation for that value, since the unit of the activation depends on both the samples employed and the parameters of the model, which may vary overmuch as can be observed in Figure 8 (a). To neutralize this unwanted source of variance, we can normalize the Transformation Variance values. We propose using the Sample Variance, as defined below, to divide the Transformation Variance and obtain a Normalized Variance measure.

2.4.2 Sample Variance Measure

The **Sample Variance** (SV) is the conceptual transpose of the Transformation Variance. It is equivalent to computing the Transformation Variance on the transpose of the **ST** matrix. Therefore, instead of computing the variance over rows and then the mean of the result (a column vector), the Sample Variance is obtained by first computing the variance over the rows, and then the mean over the resulting column vector (9).

Equation 3 shows the formal definition of the Transformation Variance for an activation a in terms of its $\mathbf{ST}(a)$ matrix.

$$SV = Mean\left(\left[Var(\mathbf{ST}[:,1]) \quad \cdots \quad Var(\,\mathbf{ST}(a)[:,m]\,)\right]\right) \tag{3}$$

While the Transformation Variance measures the variance due to the transformations of the samples, the Sample Variance measures the variance due to the natural variability of the domain. Figure 8 (b) shows the results of calculating the Sample Variance as a heatmap. Note that the order of magnitude of the values of the Sample Variance is similar to that of the Transformation Variance.

Using running mean and variance equations, computing the Sample Variance requires a single iteration through the columns of **ST**, and therefore the running time is also $\mathcal{O}(k \times n \times m)$.

2.4.3 Normalized Variance Measure

The **Normalized Variance** (*NV*) is simply the ratio between the Transformation Variance (equation 2) and the Sample Variance (equation 3) [2]:

$$NV(a) = \frac{TV(a)}{SV(a)} = \frac{\frac{1}{n}\sum_{i=1}^{n} Var(\mathbf{ST}(a)[i,:])}{\frac{1}{m}\sum_{i=1}^{m} Var(\mathbf{ST}(a)[:,i])} \tag{4}$$

[2]Note that the Normalized Variance measure corresponds to the V measure previously described by the authors in [15]

The Normalized Variance is therefore a ratio that weights the variability due to the transformation with the sample variability. Since both have the same unit, the result is a dimensionless value. Figure 8 (c) shows the result of the Normalized Variance measure.

The computation of NV requires only two loops over the **ST** matrix, a transformation-first loop to compute the Transformation Variance, and a samples-first loop to compute the Sample Variance. Therefore, its running time is also $\mathcal{O}(k \times n \times m)$.

Special cases in the computation of the Normalized Variance measure In cases where both $TV(a) = 0$ and $SV(a) = 0$, we set $NV(a) = 1$. the very definition of dead activations which don't respond to any pattern and have no use in the network, and so we set $NV = 1$ as a compromise.

For the case $TV(a) > 0$ and $SV(a) = 0$, we set $NV(a) = +\infty$. The first case is The second case is similar case but the transformations make the activation vary, possibly because they generate samples outside of the original distribution. Both cases can occur in common datasets, specially if they are synthetic or have been heavily preprocessed. For example, if all images in a dataset contain a black background and centered objects, it is likely that activations corresponding to the borders of feature maps, specially in the first layers, correspond to dead activations. However, transformations such as scale or translation can cause the borders of the feature maps to have non-zero activations, giving rise to the second case.

Values of the Normalized Variance We can analyse the possible values of NV as follows:

- If $NV(a) = 0$, then $TV(a) = 0$ and the activation is clearly invariant.
- If $NV(a) < 1$, the variance due to the transformations is less than that due to the samples, and so we can consider the activation to be approximately invariant.
- If $NV(a) > 1$ the same reasoning applies, but with the opposite conclusion.
- If $NV(a) = 1$, then both variances are in equilibrium, and there's no distinction between sample and transformation variability. In this case, it is possible that the dataset/domain naturally contains transformed samples, or simply that the model was trained in such a way that these values are similar.

However, for values $0 < NV(a)$ there is no clear interpretation in terms of absolute invariance. We can only interpret them in terms of relative invariance. For example, if $NV(a) = 0.5$, then the sample variance is twice the transformation variance.

2.5 Distance-based

Variance-based measures assume a unimodal distribution of the activations, since the variance quantifies deviations from the mean. However, in some cases that assumption may be incorrect.

Distance-based measures are similar to variance methods, but instead of calculating the variance, they employ a distance function between activations for different transformations or samples. By computing the distance between all pairs of activations, there are no assumptions of unimodality, and an appropriate distance function can be employed for different types of activations.

Analogously to the variance measures, we define the $TransformationDistance$ (TD), $SampleDistance$ (SD) and $NormalizedDistance$ (ND) measures using distance functions as follows (Equation 5)

$$\begin{aligned} TD(a) &= \text{Mean}\left(\left[\, \text{AverageDistance}(\mathbf{ST}[1,:]) \cdots \text{AverageDistance}(\mathbf{ST}[n,:]) \,\right]\right) \\ SD(a) &= \text{Mean}\left(\left[\, \text{AverageDistance}(\mathbf{ST}[:,1]) \cdots \text{AverageDistance}(\mathbf{ST}[:,m]) \,\right]\right) \\ ND(a) &= \frac{TD(a)}{SD(a)} \end{aligned} \qquad [5]$$

Where AverageDistance computes the mean distances between all values of a vector, and $d : R^2 \to R$ is any distance function:

$$\text{AverageDistance}([\, x_1 \cdots x_n \,]) = \frac{\sum_{i=1}^{n} \sum_{j=1}^{n} d(x_i, x_j)}{n^2} \qquad [6]$$

The distance measures calculate the average pairwise distances between activations either row-wise (Transformation Distance) or column-wise (Sample Distance). If an activation is completely invariant to a transformation, the mean distance between all transformations of a sample ($AverageDistance(\mathbf{ST}[i,:])$) will be 0. Therefore the Transformation Distance will be 0, and so will the Normalized Distance.

If the activation is approximately invariant, the mean distance between transformed samples will quantify this, and the mean distance between samples will quantify the degree of invariance.

Figure 10: Blocks of the distance matrix sampled for the approximation.

Approximation of the average distance The full computation of all distances between transformations and samples can be prohibitive. Such distance matrix would have size $n \times n$ for the sample variance and $m \times m$ for the transformation variance. While the mean distance does not require storing all distance values, it does require storing all k activations. As discussed in 2.2.1, the computation of the **ST** matrix must be done online. Therefore, an approximation of the mean distances must be employed. Since looping over the **ST** matrix is done by batches, it is straightforward to only compute distances for samples in the same batch. Therefore, we are approximating the mean of the full distance matrix by only computing the mean of the distances between blocks of the distance matrix.

A more principled approach would involve computing a low-rank approximation of the full euclidean distance matrix and then computing the mean distances [21]. However, the current best randomized algorithms for a single matrix are $\mathcal{O}(n+m)$ [22], which would render the computation of the measure impractical given a large number k of activations.

Relationship between variance and squared euclidean distance In the case of the squared euclidean distance measure, the variance and distance measures are equivalent, as per equation 7. In this particular case, we can avoid the approximation introduced by sparsely sampling the distance matrix by simply computing variance-based measures.

$$\begin{aligned}
\text{AverageDistance}([\,x_1 \cdots x_n\,]) &= \frac{\sum_{i=1}^{n}\sum_{j=1}^{n}(x_i - x_j)^2}{n^2} \\
&= 2n\frac{\sum_{i=1}^{n}(x_i - \text{Mean}([\,x_1 \cdots x_n\,]))^2}{n^2} \\
&= 2\frac{\sum_{i=1}^{n}(x_i - \text{Mean}([\,x_1 \cdots x_n\,]))^2}{n} \\
&= 2\,\delta([\,x_1 \cdots x_n\,])
\end{aligned} \qquad [7]$$

2.6 Same-Equivariance

Determining equivariances in general is more difficult than in the special case of invariances. As defined in section 1, a function f is equivariant to transformation t if there exists g such that $f(t(x)) = g(f(x))\forall x$. Analyzing the equivariance of a network implies a) obtaining information about the structure of g and b) quantifying this structure to obtain useful information.

In specific cases where f and t have certain structure, the function g can be univocally or approximately determined. For example, in the case of the TI-pooling model [10] for rotations, a rotation of the input corresponds approximately to a permutation of the feature maps. Without any a-priori information about f, which is generally the case in vanilla networks trained with data augmentation, estimating the equivariance requires a universal approximator function to learn the mapping g.

A special case of equivariance is same-equivariance, where $g = t$, and so $f(t(x)) = t(f(x))\forall x$. In this case, the transformation acts in the same way for both the input and the feature. As with invariance, same-equivariance assumes a particular form for g that allows to quantify this type of equivariance.

In this section we propose a method to compute the approximate Same-Equivariance of the activations. Same equivariance requires both input and feature to be in the domain of t, and therefore to share structure. For example, if the transformation t is defined on images, then $f(x)$ must be an image as well. Therefore, we will define same-equivariance not on a single activation a, but on a set of activations $A = [\,a_1 \cdots a_p\,]$. Note that A is a set with structure; for example, it could be a $3D$ tensor for RGB images. This set of activations must have the same structure as the input x.

We can estimate the degree of same-equivariance of a set of activations A to a single transformation t by measuring the average value of $||A(t(x)) - t(A(x))||$ over all samples, where $||\cdot||$ is the euclidean or other norm. If f is same-equivariant, then $A(t(x)) = t(A(x))$ and so $||A(t(x)) - t(A(x))||$. In the case where $T = [\,t_1 \cdots t_m\,]$, we can compute the average over all transformations.

A problem with the previous formulation, however, is that it cannot characterize the interactions between the representations of different transformations. Instead, if we require T to be a set of invertible transformations, we can instead compare the values $[\,t_1^{-1}(A(t_1(x))) \cdots t_m^{-1}(A(t_m(x)))\,]$. If A is same-equivariant, these values should all be equal, since $t_i^{-1}(A(t_i(x))) = t_i^{-1}(t_i(f(x))) = a(x)$.

We can therefore define the Same-Equivariance (SE) measure (equation 8) of a set of activations A in a similar fashion to the Transformation Distance measure in section 2.5.

$$\begin{aligned} SE(A) &= \text{Mean}\left(\left[\, \text{AverageDistance}(\mathbf{ST'}[1,:]) \cdots \text{AverageDistance}(\mathbf{ST'}[n,:]) \,\right]\right) \\ \mathbf{ST'}(A)[i,j] &= t_j^{-1} A(t_j(x_i)) \end{aligned} \quad [8]$$

Equation 8 employs a modified **ST** matrix denoted by $\mathbf{ST'}$, where the activations or features are inverse-transformed in order to measure the equivariance. Analogously to the invariance measures $TransformationVariance$ and $TransformationDistance$, a variance-based same-equivariance measure can be defined by replacing the distance measure with a variance calculation, which corresponds to the squared euclidean distance.

For this measure we have chosen not to normalize with the sample equivariance. Depending on the task, models are expected to be invariant to sample variations; indeed this is the principle of generalization [3]. However, same-equivariance of *samples* is not a natural prior for classification problems, and the obtained values would be generally much greater than those of the same-equivariances due to transformations. Therefore, to obtain normalized values we interpret the set of activations A as a vector and normalize it to unit norm, dividing by the norm induced by the distance measure AverageDistance.

We emphasize that this method is limited to set of activations with the same structure as x. For example, in the case of CNNs used for image analysis, this method can only be applied to feature maps. The output of fully connected layers cannot be measured for same equivariance, since most transformations for $2D$ images are not defined on $1D$ vectors. However, many modern network architectures for images rely mostly on convolutional layers, and employ only one or two fully connected to produce the final output. Therefore, this represents a minor limitation.

3 Analysis of Equivariance Measures and Models

We perform several experiments to 1) validate the measures 2) study their behavior and 3) analyze existing neural network models in terms of their invariances and equivariances[3]. To validate the measures, we perform qualitative and quantitative experiments to determine if they are indeed capable of measuring the desired properties. Afterwards, we study the general behavior of the measures in terms of different variables: random weight initialization, dataset size, dataset subset (train or test) and use of batch normalization. We also compare the measures with their stratified version, and the different convolutional aggregation strategies. Finally, we compare several popular CNN models in terms of their invariance.

While the measures can be employed to analyze any type of model, in this work we focus on image classification problems to characterize the measures and evaluate models.

Given space constraints for this summary, we present the results of only some validation experiments. The results of all experiments can be found in [18]. Nonetheless, we also include a conclusion section based on the results of all experiments.

3.1 Setup

The general framework of our experiments consists of training a model with a given dataset and set of transformations used for data augmentation. The data augmentation consists of applying to each example a transformation randomly selected from the set of transformation.

Then, we evaluate measures using the trained model and the **same** set of transformation, unless otherwise stated. In the following, we describe the datasets, transformations and models used in our experiments.

3.1.1 Datasets

All of our experiments use either the MNIST or the CIFAR10 datasets. Both are well known and we expect any analysis performed on them is easier to understand and relate to existing methods. Also, both are small datasets to ease the computational burden. While MNIST is somewhat toy-like, it provides more interpretable results. Since all models obtain an accuracy near 100 for the test set on MNIST, this dataset allows to evaluate the results of the measure in a near perfect accuracy setting. CIFAR10, on the other hand, consists of more complex natural images that complement the analysis.

[3]All experiment code available at `https://github.com/facundoq/transformational_measures_experiments`

3.1.2 Transformation sets

To simplify our experimental setup, we define four common transformation sets. These sets represent common affine transformations, and therefore provide a wide range diversity in order to establish properties of the measures independently of the specific transformations used.

1. Rotation (16 transformations), of discretized into 16 distinct angles. Rotations are always with respect to the center of the image.
2. Scaling (11 transformations). We used scale coefficients $[\,0.5\ 0.6\ 0.7\ 0.8\ 0.9\ 1.0\ 1.05\ 1.10\ 1.15\ 1.20\ 1.25\,]$, and kept the aspect ratio fixed (same coefficient for height and width). We downscale more than upscale because upscaling to more than 125% of the image made the objects no longer recognizable on both datasets. Note that we scale the contents of the image but the size is kept constant; when downscaling, we fill the borders with black pixels.
3. Translation (24 transformations): We generate all translations of $d = 2^i$ pixels with the following scheme $[\,(-d,-d)\ (-d,d)\ (d,-d)\ (d,d)\ (0,d)\ (d,0)\ (0,-d)\ (-d,0)\,]$, with $i = [\,0,1,2\,]$, for a total of $8*3 = 24$ translation transformations. Note that in the case of MNIST and CIFAR this amounts to a translation of up to 15% of the image in each direction.
4. Combined (4224 transformations): All the possible transformation combinations of the previous three sets, with $16 \times 11 \times 24 = 4224$ transformations.

We will refer to these simply as the rotation, scale, translation and combined sets. Figure 11 shows examples of all sets for MNIST and CIFAR10.

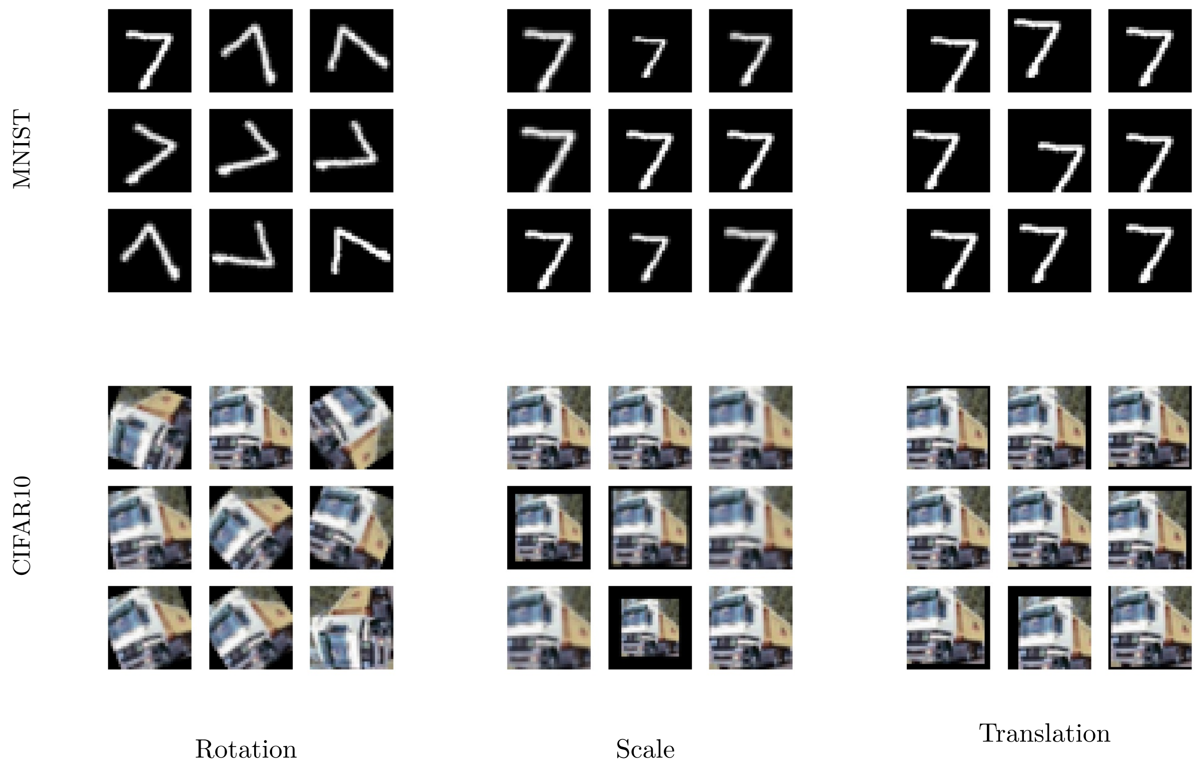


Figure 11: Samples of MNIST (top row) and CIFAR10 (bottom row) for each set of transformations.

Since the combined transformation set is very large, it is computationally very expensive to evaluate. Therefore we have only used it in some experiments.

3.1.3 Models

For most of the experiments we use the **SimpleConv** model, shown in Figure 12. It is a simple model consisting of traditional Convolution (Conv), MaxPooling (MaxPool) and Fully Connected (FC) layers. The

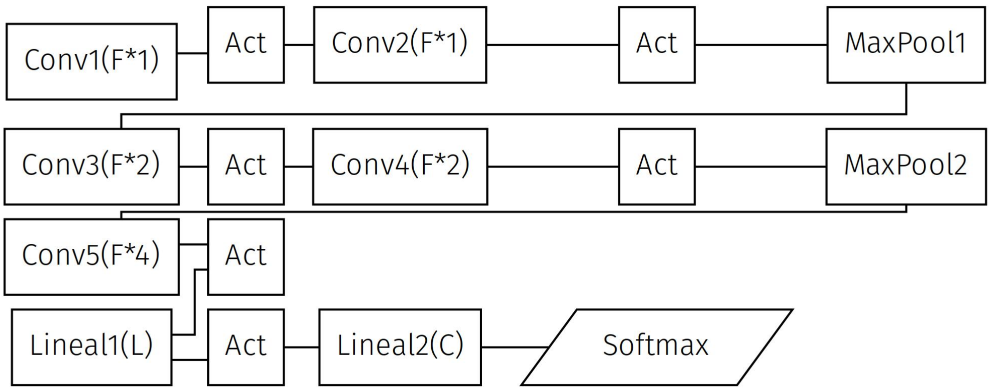


Figure 12: Architecture of the SimpleConv model. The model is a typical CNN with Convolutional, Max-Pooling and Fully Connected layers.

activation functions are all ELU, except for the final $Softmax$. All convolutions have $stride = 1$ and $kernelsize = 3 \times 3$. MaxPooling layers are 2D and use $stride = 2$ and $kernelsize = 2 \times 2$. SimpleConv was the simplest possible model with only those three layers that obtains 80% accuracy in CIFAR, similarly to the rest of the models. Limiting the design to only these layers applied in a feedforward fashion also facilitates the analysis.

Since our goal is not to obtain state of the art accuracies, to prioritize consistency and simplicity we employed the AdamW optimizer [23] with a learning rate of $1 - e4$ to train the models. The number of epochs used to train each model was determined separately for every dataset and set of transformations to ensure the model converges. To this effect, a base number of epochs was chosen for every model/dataset combination. To account for the difficulty of adding more transformations to data augmentation of the training set, that number of epochs was multiplied by $log(m)$, where m is the size of the transformation set.

Given that CIFAR10 is a more challenging dataset than MNIST, the models for this last dataset have been modified to use half the number of filters/features than for CIFAR10 in all layers. Since effective invariance cannot be separated from accuracy, we verified that in all cases the accuracy of the models was superior to 95% on MNIST, and to 75% on CIFAR10.

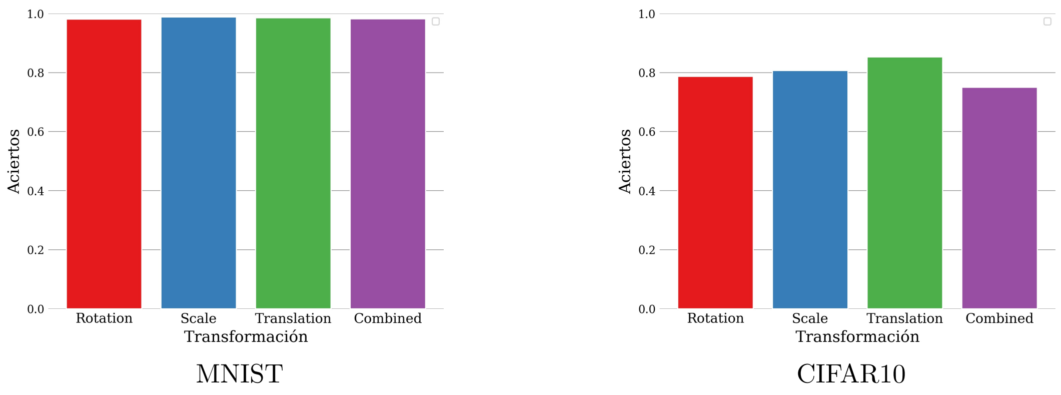


Figure 13: Accuracies for the SimpleConv model on MNIST and CIFAR10 for the 4 sets of transformations.

3.2 Results

We validate the measures in terms of their ability to detect invariances in the models, and their sensitivity to random initializations of the models' weights. To that effect, we present the results of three experiments:

- A comparison of between the measures with models trained with data augmentation (invariant) and without it, to validate their sensitivity.

- An analysis of the impact of randomly initializing networks on the final pattern of invariance or same-equivariance of its layers.
- An analysis of the pattern of invariance or same-equivariance of the layers of a network before it has been trained, ie, with random weights.

In order to better understand the information provided by the presented measures, we compare and analyse the results of the various measures. To simplify the comparison, we present the results of the measures aggregated by layers; we compute the mean value of the measure for all activation of each layer, and plot the resulting means.

Finally, we compare the invariance of models trained with and without data augmentation. Models trained without data augmentation have very low accuracy when evaluated on transformed samples [15]. Models trained *with* data augmentation, on the other hand, recover the lost performance. Therefore, we expect the latter models to possess more invariance, at least in the final Softmax layer. In this way we can determine if the measures are actually representative of the invariance.

3.2.1 Transformation and Sample Measures

Figure 14 shows the distribution of the Transformation Variance and Sample Variance measures (numerator and denominator of the Normalized Variance measure, respectively) for the MNIST and CIFAR10 datasets.

First, we note that the magnitude of both measures is in the same order, but it is quite different between datasets and mildly different between transformations. We note as well that the magnitudes vary significantly across layers. The units of the activations of convolutional layers are significantly lower than those of fully connected layers, hence the lower variance. The variance for models on MNIST is also much lower than the variance for models trained on CIFAR10. Finally, in the case of MNIST the variance for rotation transformations is significantly lower than those of other transformations. This confirms our claims in Section 2 on the importance of normalizing the Transformation Variance to obtain values that are interpretable across layers, datasets and transformations.

Second, models trained with data augmentation have less variant activations, as expected. This indicates that the measure is indeed reflecting the invariance of the model. It is interesting to note, however, that in some occasions models trained without data augmentation actually have less average variance than those trained with it, except for the final layers. While this could be possible in principle because of different coding schemes between the models, note that when this occurs it affects both the Transformation Variance and Sample Variance measure, which is another reason to require normalization of the measure. The final layers are always more invariant in models trained with data augmentation, as expected since the loss function indirectly optimizes for invariance.

The fact that the Transformation Variance of each layer is almost always less than the Sample Variance indicates that the invariance is a global property, that is, it is encoded in the whole network and not in a specialized set of layers.

Figure 15 shows the distribution of the Transformation Distance and Sample Distance measures (numerator and denominator of the Distance measure, respectively) for the MNIST and CIFAR10 datasets. We note here as well the need for normalizing the Transformation measure to obtain interpretable results. Note also the correlation between the Transformation Variance and Transformation Distance measures, as well as the Sample Variance and Sample Distance.

3.2.2 Normalized measures

Figure 16 shows the results with the same configuration as before of the normalized invariance measures Normalized Variance (NV), Normalized Distance (ND), Goodfellow (GF) and ANOVA.

The ANOVA measure is very insensitive since it rejects the null hypothesis in all cases and therefore considers all activations as variant. This indicates the ANOVA measure is not well suited to measure invariance as is. The Variance and Distance measures, however, can detect the relative invariances of each layer. Both measures are correlated, as expected (Section 2).

We include the Goodfellow measure [24], which also measures invariance but with a different approach. In order to calculate the measure in a reasonable time, we adapted the original algorithm to determine the threshold t so that instead of calculating the 1% percentile, we calculate the value z for which a normal distribution satisfies $P(f \leq z) = 0.01$. Afterwards, the employ such value z as a threshold t. This way, we avoid having to store all activations to calculate the percentile.

For the Goodfellow measure, lower values also indicate more invariance. The measure seems to detect the greater invariance of the models trained with data augmentation, specially in linear layers. However, it

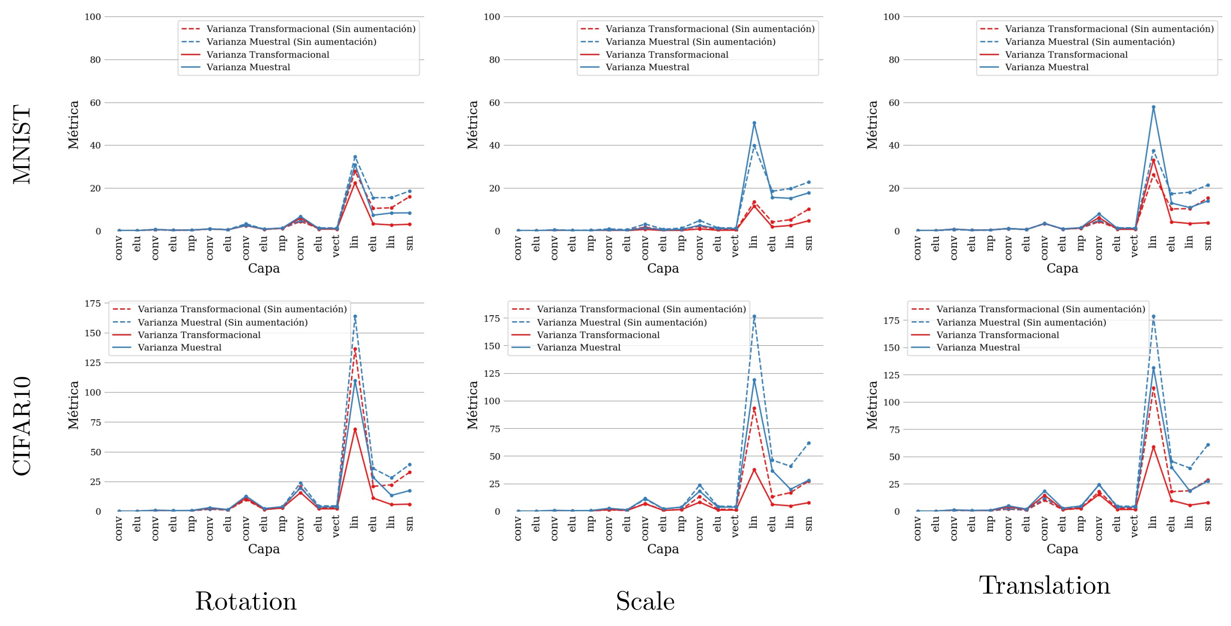


Figure 14: Comparison of Transformation Variance (TV) and Sample Variance (SV) measures for various transformation sets and the SimpleConv model. Dotted lines indicate models trained without data augmentation.

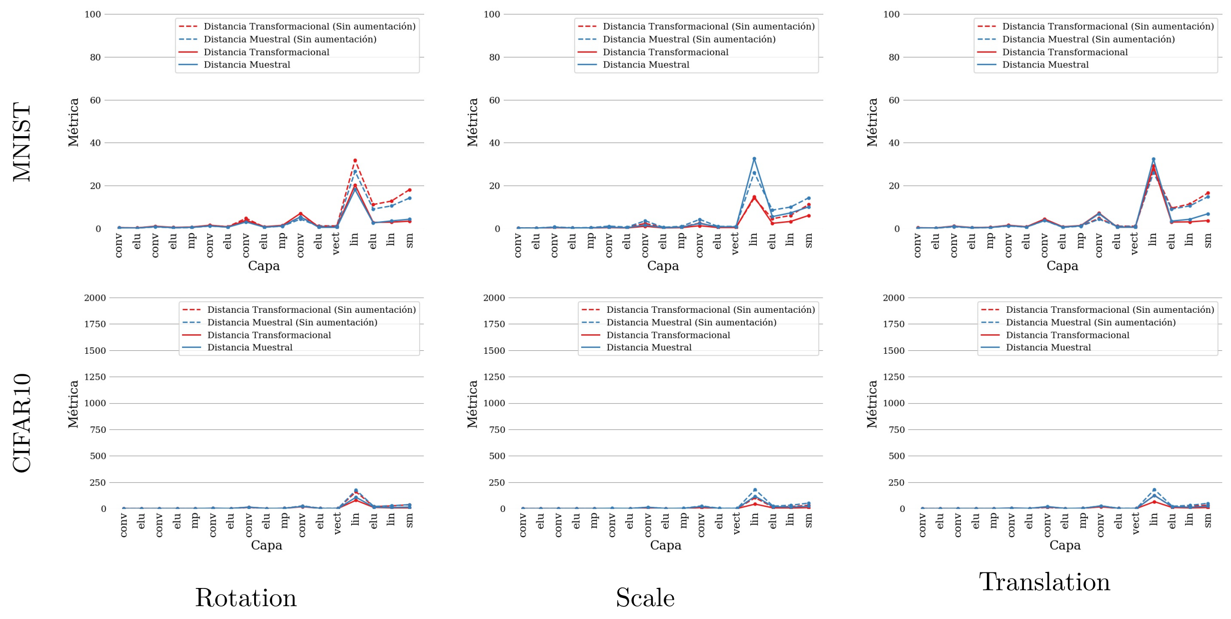


Figure 15: Comparison of Transformation Distance (TD) and Sample Distance (SD) measures for the SimpleConv model. Dotted lines indicate models trained without data augmentation.

presents low peaks for convolutional layers, which contradict the fact that activation functions such as ELU never increase the variance, as show in [18].

The Normalized Variance and Normalized Distance measures both detect the greater invariance of models trained with data augmentation. Both measures are also correlated, as expected. The Normalized Variance measure shows slightly lower values than the Normalized Distance measure, a difference which is expected due to the approximation, but their structure is quite similar.

3.2.3 Effect of random initialization of the model on the measures.

To study the effect of the random initialization of the model on the measure, we trained 8 instances of each model using the same architecture, dataset and set of transformations, until convergence. Each model

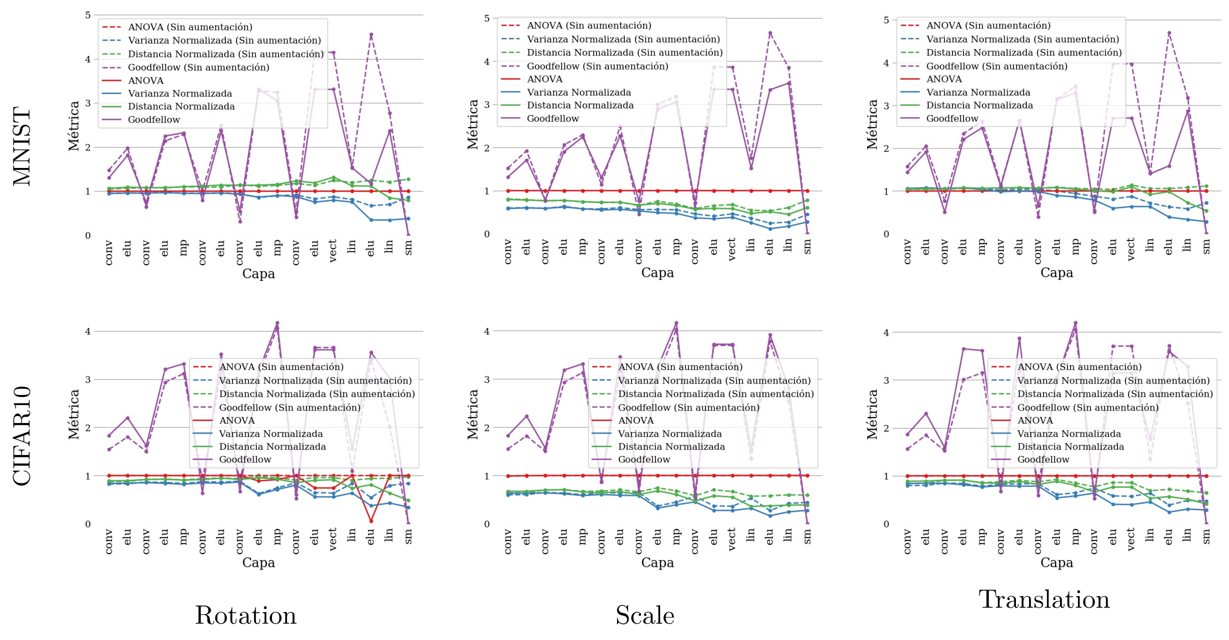


Figure 16: Comparison of normalized measures Normalized Variance (NV), Normalized Distance (ND) and ANOVA for the SimpleConv model. Dotted lines indicate models trained without data augmentation.

was initialized with different random weights. Afterwards, each trained model was evaluated with the same measure.

Figure 17 shows the results for all the instances of the Normalized Variance measure, aggregated by layer, with separate experiments for different datasets/transformations.

The results suggest that that the Normalized Variance varies very slightly with respect to the initialization, although we have detected occasional outliers (Figure 17 (c)). Interestingly, it is an emergent property despite the random initialization of the network weights. Therefore, this pattern might mostly depend only on the model architecture, dataset and transformation. This suggests we can avoid performing several runs to compute an average value of a metric when the test set is sufficiently large, in the same way we think about accuracy and other measures.

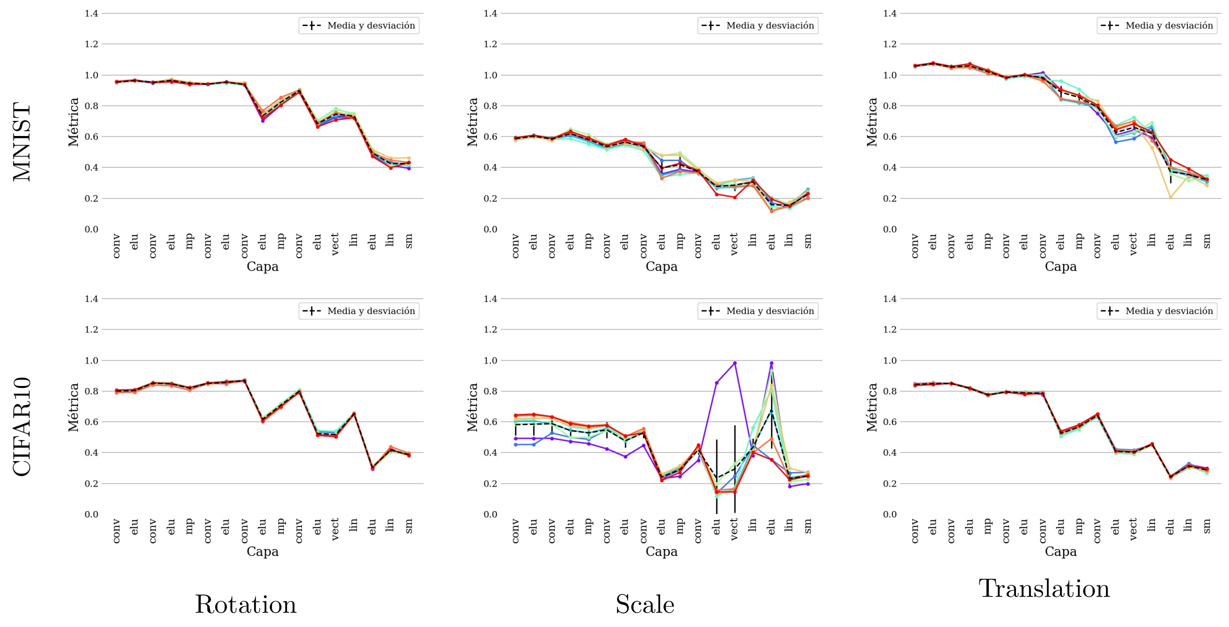


Figure 17: Stability of the Normalized Variance measure for the SimpleConv model. Each line in each plot corresponds to the invariance obtained by different models with the same architecture. The dashed line shows the mean values. Vertical bars indicate standard deviation.

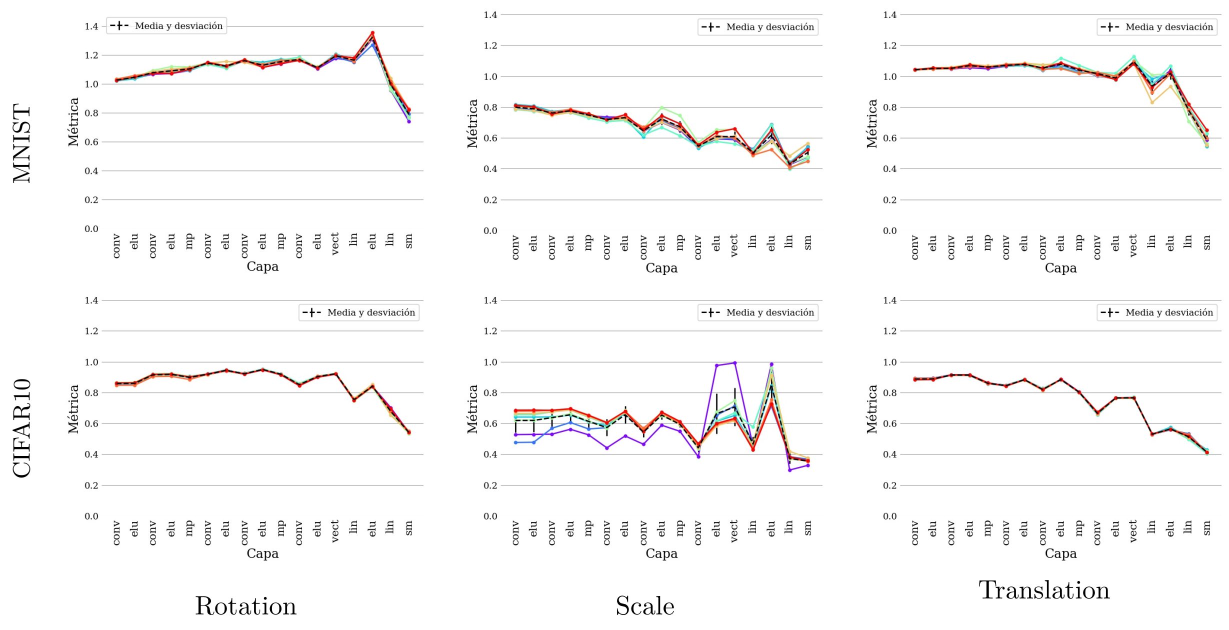


Figure 18: Stability of the Normalized Distance measure for the SimpleConv model. Each line in each plot corresponds to the invariance obtained by different models with the same architecture. The dashed line shows the mean values. Vertical bars indicate standard deviation.

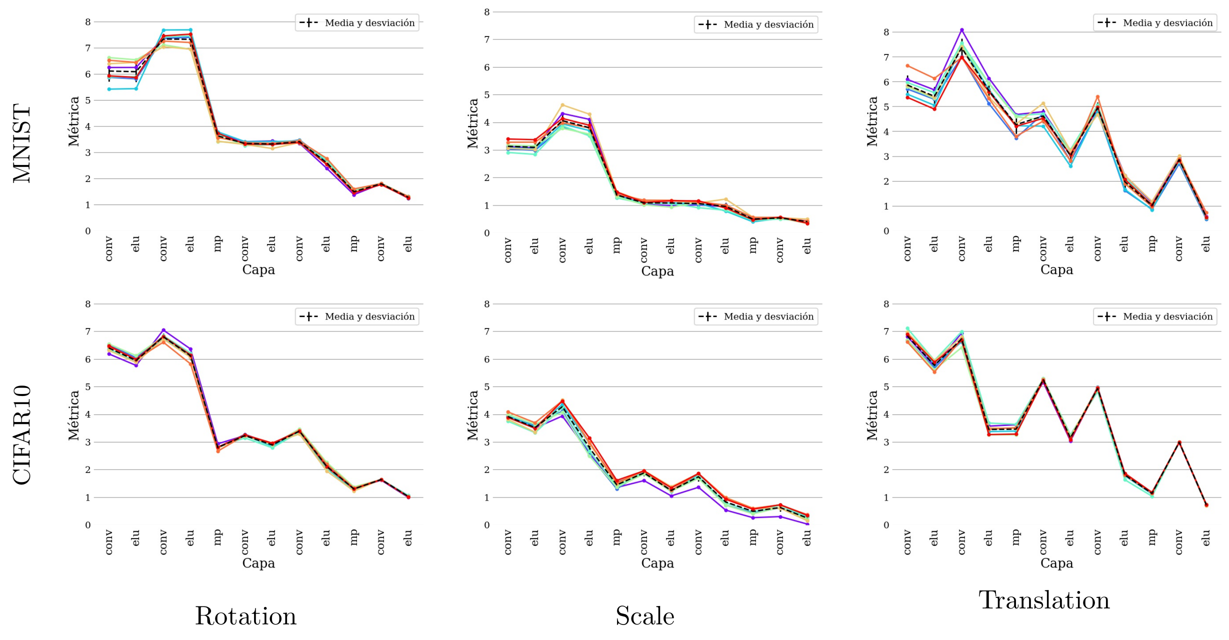


Figure 19: Stability of the Same-Equivariance measure for the SimpleConv model. Each line in each plot corresponds to the invariance obtained by different models with the same architecture. The dashed line shows the mean values. Vertical bars indicate standard deviation.

3.2.4 Networks with random weights

To further understand the origin of the invariance structure of the networks, we study the invariance and same-equivariance of untrained models, that is, models with random weights. Both convolutional and fully connected layers employ the Kaiming uniform initialization [25], and we measured 8 random models for each case, given the random initialization.

Figure 20 shows the results for the Normalized Variance measure. We can observe that the invariance depends highly on the transformation set and dataset, but not on the set of random weights of the network.

This suggests that given a controlled initialization, invariance is mostly a property of the network architecture rather than the specific values of the weights. Figure 21 shows a similar situation for the Normalized Distance measure. Coupled with the results in Section 3.2.3, this indicates that the invariance structure is mostly fixed for a model, given a fixed dataset and a set of transformations.

Another interesting finding is that the invariance of the networks without training has no trend, that is, it does not vary with respect to layer depth or type. As noted in Section 3.2, models trained with data augmentation show a negative trend of variance, with less variance in the first layers.

Given that when training the network its invariance stabilizes, we can conclude that the acquisition of invariance through training with data augmentation is not a process that yields a single set of weights as a solution, in the same way that when training a network we can obtain very similar accuracies with different sets of final weights [26]. This indicates that the local minima corresponding to optimal accuracies also corresponds in this case to a similar pattern of invariance.

In the case of the Distance Same-Equivariance (Figure 22)) we can observe that the same-equivariance does not vary significantly, and is very similar to that of trained models (Section 3.2.3). This indicates that this property, unlike invariance, depends only on the architecture of the network and not on its weights.

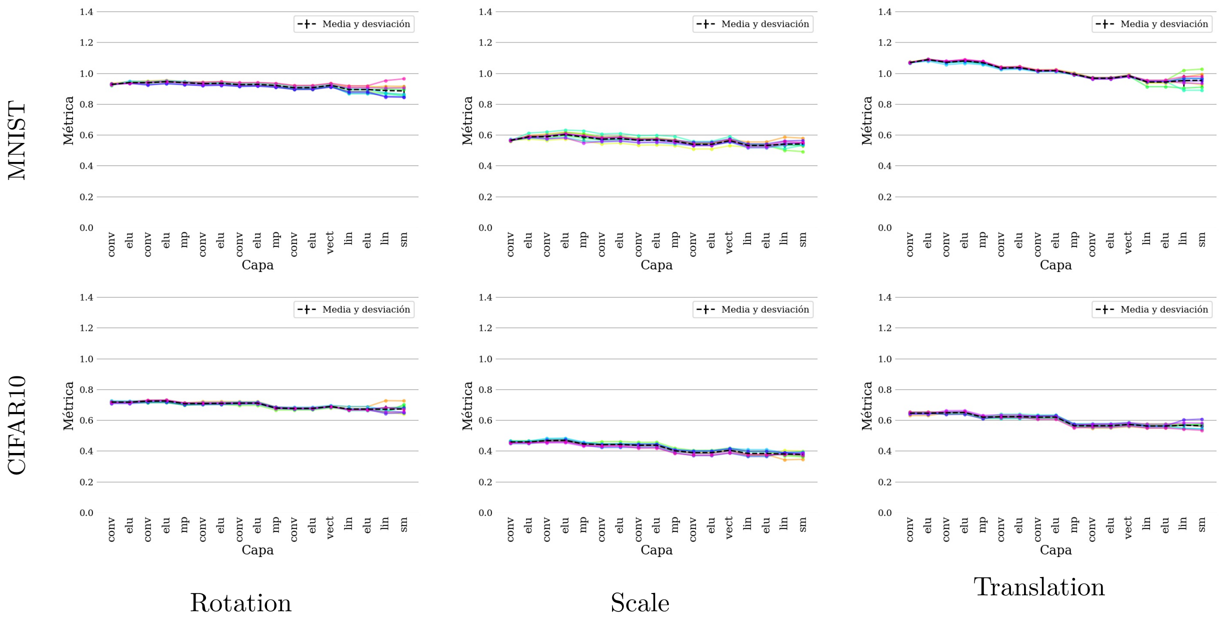


Figure 20: Normalized Variance for the SimpleConv model with random weights. Each plot corresponds to the measure calculated with different sets of random weights.

3.3 Summary of results

Using the measures defined in Section 2 we evaluated the invariance and same-equivariance of modern CNN models. Also, we validated and analyzed several aspects of the measures.

For these experiments, we used the well-known CIFAR10 and MNIST datasets. We employed 4 sets of affine transformations, which are of importance for several image processing applications: rotation, scaling and translation, and a combination of these. In most experiments, we used the SimpleConv model, which is a common model that represents the basic characteristics of modern CNN models. This combination of model, dataset and transformation allows the results to be interpreted and relevant for analysis.

We start by validating the measures with models trained with and without data augmentation. In this fashion, we note that the Normalized Variance and Normalized Distance measures are actually correlated with invariance, and that æcan also measure same-equivariance. We also compare the behavior of the measures with and without normalization, and found such schemes necessary for a simple interpretation and comparison of the measure results.

We also compare the measures with an existing technique [24]. In the case of the ANOVA measure, which we also proposed in Section 2, we found it does not have the necessary sensitivity for detecting invariance in neural networks, since it always rejects the null hypothesis, therefore considering all activations as invariant. The Normalized Variance and Normalized Distance measures, however, can measure fluctuations in invariance, with more sensitivity than the measure by Goodfellow. In the case of the Distance Same-Equivariance

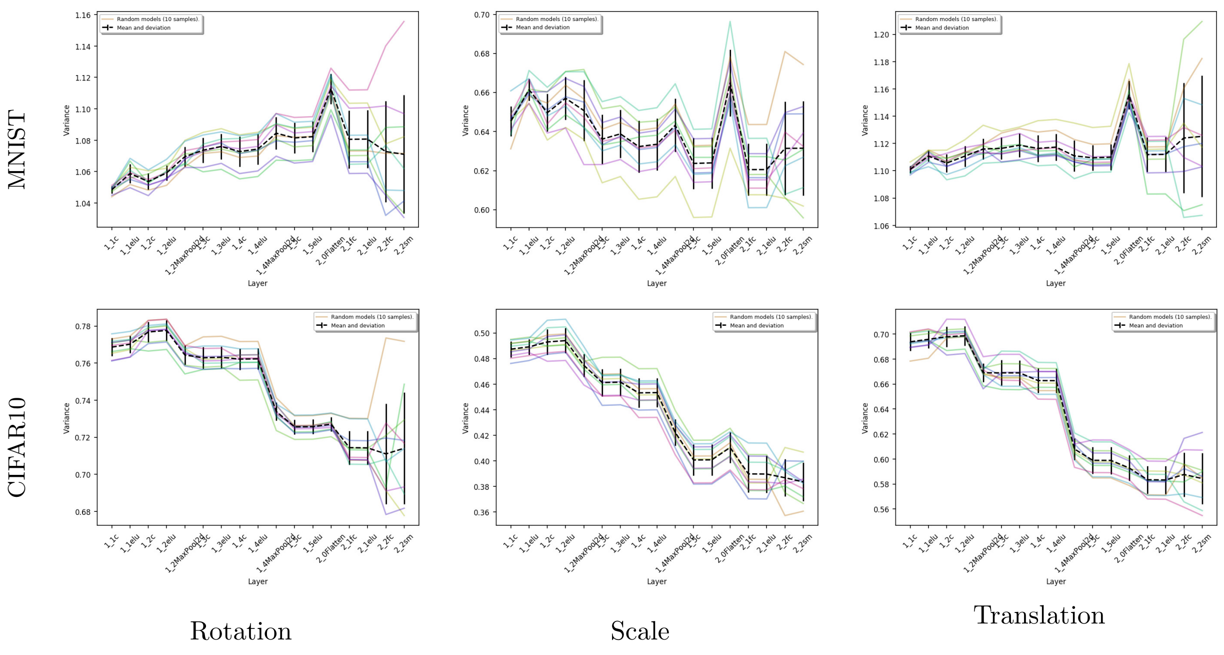


Figure 21: Normalized Distance for the SimpleConv model with random weights. Each plot corresponds to the measure calculated with different sets of random weights.

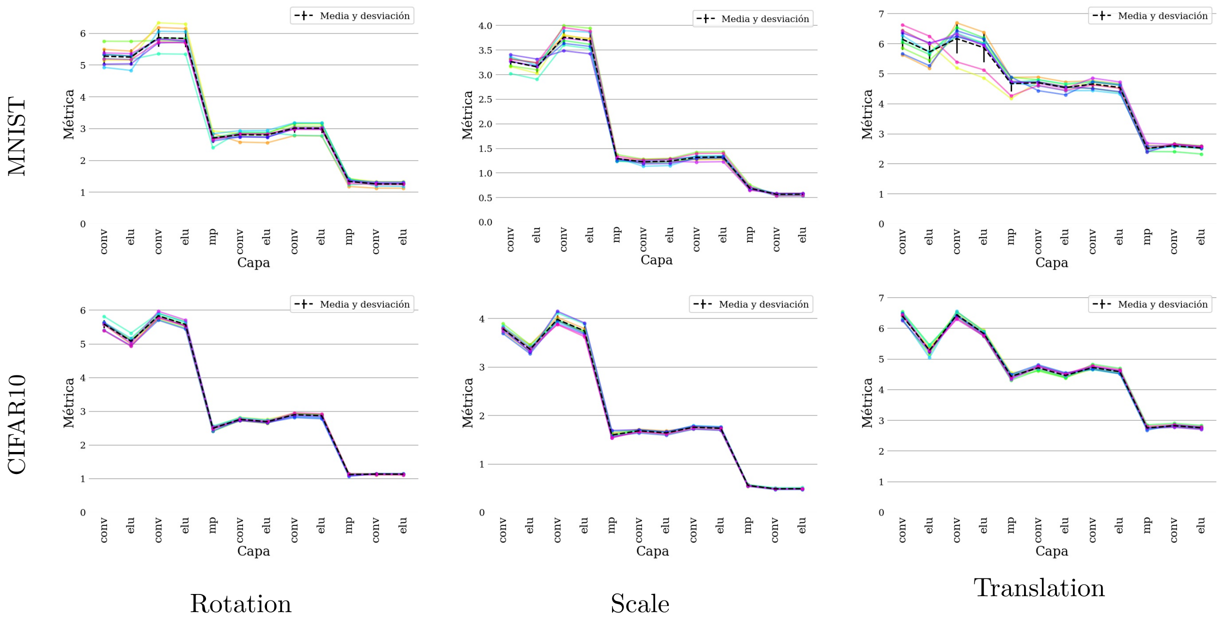


Figure 22: Same-Equivariance for the SimpleConv model with random weights. Each plot corresponds to the measure calculated with different sets of random weights.

measure, we did not perform comparisons with other measure since it is the first of its kind.

Based on our analysis, we found that measures require between 5000 and 10000 to evaluate correctly on MNIST and CIFAR10. In this case, they can be computed with either the test or training set, with little different in results. They are stable with respect to random initializations of the layer weights, since they converge to similar values in all cases.

Confirming previous results of other authors [24, 27], we find invariance in neural networks trained with data augmentation increases with depth for neural network. Studying the behavior of random networks, and of networks during their training, the results suggets that invariance is a property mostly learned during training, although its final structure is not determined by the weights. Instead, architecture, transformations and samples mostly determine the pattern of invariance of a network. The structure and magnitude of the invariance vary with the type and complexity of transformations, respectively. Results also suggests that invariance depends on both the dataset used evaluate the measure and the dataset used to train the model.

Same-equivariance, as well as invariance, also increases with network depth. Unlike invariance, it depends mostly on the network architecture, with much less influence from the training procedure. Experiments which evaluate same-equivariance with different datasets indicate that it does not depend either on the dataset used to train the network and the dataset used to evaluate the measure. On the other hand, it does not depend on the set of transformations used to train the model, but it does depend on the transformations used to evaluate the measure. As with invariance, its structure and magnitude depends on the type of transformation and complexity, respectively, of the transformations used to evaluate the measure.

Networks can also be analyzed in terms of architectural decisions such as the type of activation function and the use of Batch Normalization layers, which for our experiments seem not to affect same-equivariance. Experiments with kernel size indicate that it correlates negatively with same-equivariance. This is to be expected, since same-equivariance for networks that process images is measured on feature maps being output by convolutional layers, and these are very sensitive to this parameter. MaxPooling layers seem to increase same-equivariance with respect to simply using Convolutional layers with $stride = 2$, although the same does not occur with invariance. Finally, most modern CNN models such as ResNets [17] and Inception [28] seem to possess a similar pattern of same-equivariance and invariance.

To summarize, the methodology employed in these experiments allow the characterization of invariance and same-equivariance of neural network models, allowing for a new perspective on their inner workings.

4 Conclusions

Our main conclusion is that the encoding of equivariance in a network is a complex process that can be understood from different perspectives, and the equivariant measures we proposed enable important tools to investigate them.

For example, using the measures we show that the structure of the invariance of a network is a property that does not depend on the exact weights of the network, both for networks with random weights and for trained networks. However, this structure is dependent on the dataset and transformation set. On the other hand, same-equivariance is a property that does not depend on the weights nor the dataset used to measure it, and has only a mild dependence on the transformation set. This indicates that same-equivariance is determined by the network design to a larger degree than invariance. We also determine that features such as Batch Normalization layers do not affect the equivariance of the network, while the size of the kernel in Convolutional Layers does, regardless of the fact that networks with either of these characteristics achieve similar levels of accuracy.

Therefore, the set of measures and experimental methods we propose in this thesis allow a deep analysis of the encoding of equivariance in a network. We believe it is possible to learn more about Convolutional and Neural Networks by studying their equivariances and therefore improve existing models, enabling new applications of Computer Vision and Machine Learning.

4.1 Topics for future research

Regarding the acquisition of invariance via data augmentation, in principle we note the necessity of expanding the evaluation domains, considering other classification problems as well as segmentation, localization and regression. In this regard, domains where samples appear naturally rotated are of special importance. Furthermore, studies relating the complexity of the transformations to the corresponding model complexity are needed to understand the computational cost of invariance and equivariance. This relationship can be studied in terms of computing time or size of the model, as well as design complexity and transferability of features. Finally, techniques for transfer learning to acquire invariance must be advanced and formalized, given that in several domains transfer learning is the most direct and sometimes only choice to obtain models with good performance

Regarding the proposed equivariance measures, it would be interesting to endow them with the capacity to automatically detect equivariance structures in the network in terms of groups of activations. This would allow the analysis with intermediate granularities that lie between analyzing the whole network and individual activations. Furthermore, efficient extensions of the measures for the full equivariance case can be explored. The statistical characteristics of the measures have yet to be studied, in order to be able to perform hypothesis testing, for example, to compare two models in terms of invariance. Also, invariance analysis can be a complement to Adversarial Attacks and Defenses, since invariance to perturbations implies robustness to attacks. Finally, we have planned to implement support for Tensorflow and Tensorboard in the Transformational Measures library, so that the use of this techniques is available to a wider community of researchers. We will also continue with its development to improve its performance and ease of use, adding new features such as the ability to filter layers or activations, and to pre-process them.

Regarding the characterization of models via the measures, we believe it is necessary to explore more certain aspects such as the dependency between the number of filters or features and the acquired equivariance. Using the same scheme we employed for the data augmentation experiments, we will analyse the difference of the invariance structure between models trained for invariance from scratch from those that have been pre-trained in order to offer a complementary perspective on transfer learning via invariance and equivariance. Finally, we are also interested in widening the set of transformations and domains in which to apply the measure, ranging outside the set of affine transformations and image classification problems

Given the preceding arguments, we believe more can be learned about Neural Networks by studying their equivariances and invariances.

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[2] S. Dieleman, J. De Fauw, and K. Kavukcuoglu, “Exploiting Cyclic Symmetry in Convolutional Neural Networks,” *arXiv:1602.02660 [cs]*, Feb. 2016, arXiv: 1602.02660. [Online]. Available: http://arxiv.org/abs/1602.02660

[3] Q. Xie, Z. Dai, Y. Du, E. Hovy, and G. Neubig, “Controllable invariance through adversarial feature learning,” in *Advances in Neural Information Processing Systems*, 2017, pp. 585–596.

[4] W. Shang, K. Sohn, D. Almeida, and H. Lee, “Understanding and improving convolutional neural networks via concatenated rectified linear units,” in *International Conference on Machine Learning*, 2016, pp. 2217–2225.

[5] F. Bucci, F. Lillo, J.-P. Bouchaud, and M. Benzaquen, “Are trading invariants really invariant? trading costs matter,” 2019.

[6] F. Quiroga, F. Ronchetti, L. Lanzarini, and A. F. Bariviera, “Revisiting data augmentation for rotational invariance in convolutional neural networks,” in *International Conference on Modelling and Simulation in Management Sciences*. Springer, 2018, pp. 127–141.

[7] A. Azulay and Y. Weiss, “Why do deep convolutional networks generalize so poorly to small image transformations?” *CoRR*, vol. abs/1805.12177, 2018. [Online]. Available: http://arxiv.org/abs/1805.12177

[8] R. Gens and P. M. Domingos, “Deep Symmetry Networks,” in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 2537–2545. [Online]. Available: http://papers.nips.cc/paper/5424-deep-symmetry-networks.pdf

[9] T. S. Cohen and M. Welling, “Steerable CNNs,” *arXiv:1612.08498 [cs, stat]*, Dec. 2016, arXiv: 1612.08498. [Online]. Available: http://arxiv.org/abs/1612.08498

[10] D. Laptev, N. Savinov, J. M. Buhmann, and M. Pollefeys, “TI-POOLING: transformation-invariant pooling for feature learning in convolutional neural networks,” *CoRR*, vol. abs/1604.06318, 2016. [Online]. Available: http://arxiv.org/abs/1604.06318

[11] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu, “Spatial Transformer Networks,” *arXiv:1506.02025 [cs]*, Jun. 2015, arXiv: 1506.02025. [Online]. Available: http://arxiv.org/abs/1506.02025

[12] K. Lenc and A. Vedaldi, “Understanding image representations by measuring their equivariance and equivalence,” *arXiv:1411.5908 [cs]*, Nov. 2014, arXiv: 1411.5908. [Online]. Available: http://arxiv.org/abs/1411.5908

[13] M. Srivastava and K. Grill-Spector, “The effect of learning strategy versus inherent architecture properties on the ability of convolutional neural networks to develop transformation invariance,” *CoRR*, vol. abs/1810.13128, 2018. [Online]. Available: http://arxiv.org/abs/1810.13128

[14] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. A. Riedmiller, "Striving for simplicity: The all convolutional net," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings*, 2015. [Online]. Available: http://arxiv.org/abs/1412.6806

[15] F. Quiroga, J. Torrents-Barrena, L. Lanzarini, and D. Puig, "Measuring (in) variances in convolutional networks," in *Conference on Cloud Computing and Big Data*. Springer, 2019, pp. 98–109.

[16] A. C. Gilbert, Y. Zhang, K. Lee, Y. Zhang, and H. Lee, "Towards understanding the invertibility of convolutional neural networks," *Proceeding of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI)*, 2017.

[17] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[18] F. M. Quiroga, "Medidas de invarianza y equivarianza a transformaciones en redes neuronales convolucionales," Ph.D. dissertation, Universidad Nacional de La Plata, 2020.

[19] R. E. Kirk, "Experimental design," *Handbook of Psychology, Second Edition*, vol. 2, 2012.

[20] T. F. Chan, G. H. Golub, and R. J. LeVeque, "Algorithms for computing the sample variance: Analysis and recommendations," *The American Statistician*, vol. 37, no. 3, pp. 242–247, 1983.

[21] I. Dokmanic, R. Parhizkar, J. Ranieri, and M. Vetterli, "Euclidean distance matrices: essential theory, algorithms, and applications," *IEEE Signal Processing Magazine*, vol. 32, no. 6, pp. 12–30, 2015.

[22] P. Indyk, A. Vakilian, T. Wagner, and D. Woodruff, "Sample-optimal low-rank approximation of distance matrices," *arXiv preprint arXiv:1906.00339*, 2019.

[23] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=Bkg6RiCqY7

[24] I. Goodfellow, H. Lee, Q. V. Le, A. Saxe, and A. Y. Ng, "Measuring invariances in deep networks," in *Advances in neural information processing systems*, 2009, pp. 646–654.

[25] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[26] K. Kawaguchi, "Deep learning without poor local minima," in *Advances in neural information processing systems*, 2016, pp. 586–594.

[27] K. Lenc and A. Vedaldi, "Understanding image representations by measuring their equivariance and equivalence," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 991–999.

[28] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.