Delft University of Technology

Deep Learning
CS4240

Reproducibility Project Blog

Authors:
Mark Erik Lukacs (5378559, mlukacs)
Stefan Petrescu (5352150, stefanpetrescu)

April 19, 2021

Context

Convolutional neural networks (CNNs) have proven to be a dominant force in computer vision tasks, and have a broad range of applications, for e.g. in image and video recognition, image classification, image segmentation, etc. CNNs are designed to automatically and adaptively learn spatial hierarchies of features through back-propagation by using multiple building blocks, such as convolutional layers, pooling layers, and fully connected layers [8]. An interesting property of CNNs is the fact that, by design, a CNN is translation equivariant i.e. the translation of input features results in an equivalent translation of outputs. As an intuitive example of what this means, if a CNN has been trained to detect the presence of a cat in an image, due to its translation equivariance property, it will be able to recognize the cat, regardless of where the cat appears in the image. This is achieved by weight sharing across the receptive field of the neurons in the convolutional layers. Thus, in Figure 1, if a cat image is shifted, a CNN will still be able to detect the cat.

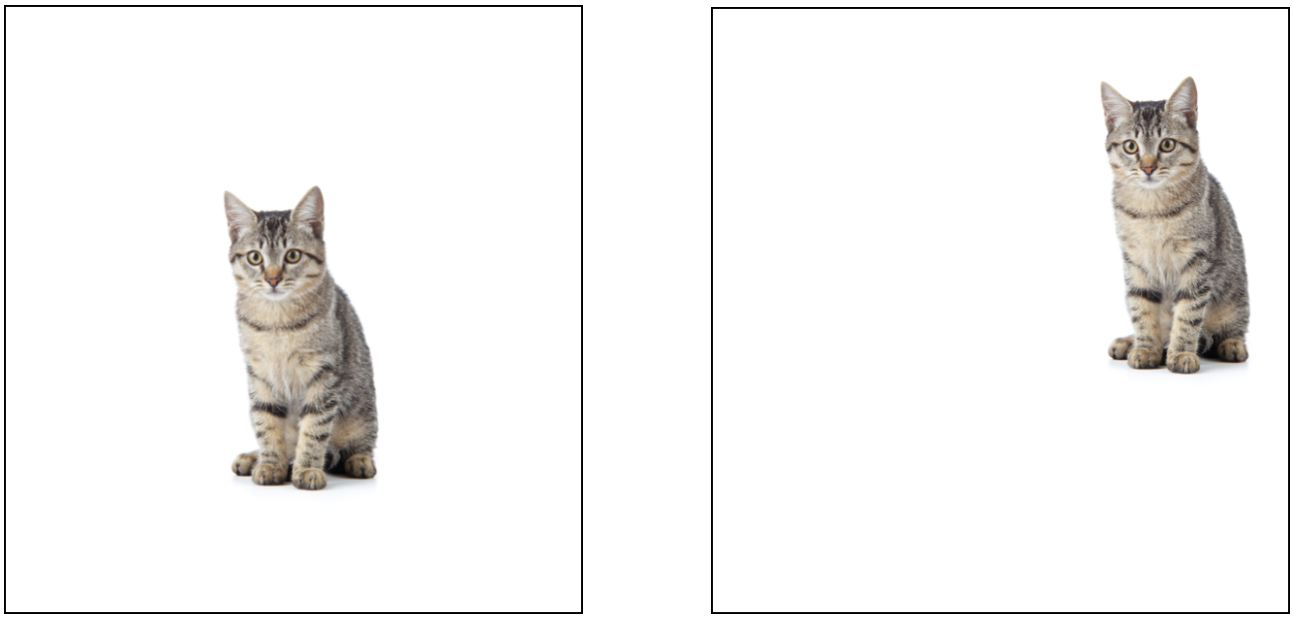

Figure 1: Example of the translation equivariance property of CNNs; the ability to recognize the cat, regardless of how the image is shifted.

Now, what happens when the same objects appear at different scales in images? For example, what would happen if cats would appear at different scales? Would a regular CNN still be able to recognize the cats? Unfortunately, although CNNs have very powerful and interesting properties, they are not designed to be equivariant to neither rotations nor scale changes of the input. This becomes a problem right? Because, in a real life application, these types of input transformations happen all the time - for e.g. think of images processed by an autonomous vehicle - it is desirable to detect pedestrians, regardless of the scale they may appear at.

Figure 2: Example of scaled(-down) objects; CNNs, are not inherently designed to recognize the same objects at different scales.

So... how can this problem be tackled? In 2019, a very interesting solution was proposed by Ivan Sosnovik, Michał Szmaja, and Arnold Smeulders. Their group published a paper named "Scale-Equivariant Steerable Networks" that solved this issue. Not only did they fix this issue, but they managed to provide a computationally expensive comparable solution to 'vanilla' CNNs, obtaining state of the art results for the MNIST and STL-10 datasets. Therefore, as we found the paper to be very interesting & exciting, in the following sections, we analyze their contributions, provide some intuition behind their method, and, last but not least, present our attempt at replicating their results.

What is scale-equivariance?

As previously mentioned, one of the most important reasons for which CNNs excel in computer vision tasks is that convolutional layers are translation equivariant. This means that if we shift an input image by (x', y') pixels, the output of the layer also shifts.

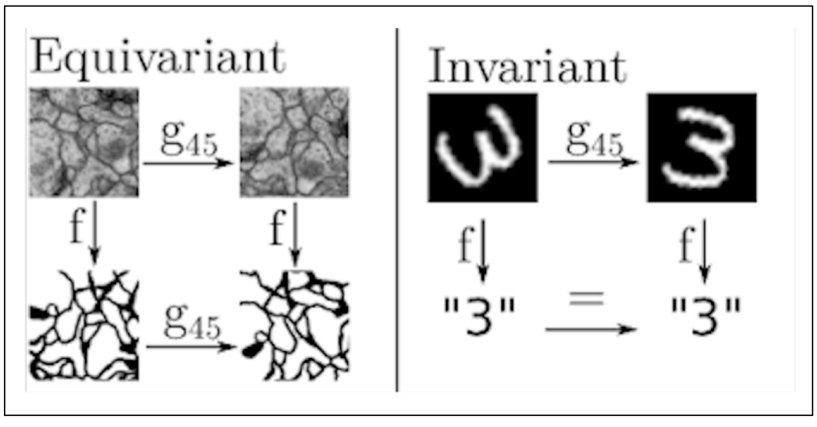

Figure 3: Example of the translation equivariance property of CNNs; the ability to recognize the cat, regardless of how the image is shifted.

A special case of equivariance, is invariance. Invariance means that no matter how we transform the input, the output remains the same. The transition from equivariance to invariance in CNNs is happening in the pooling layers. For example, if the biggest value in a 3x3 pooling block is in the center, an input shift of 1 doesn't change the output of that block. However, an important remark has to be made, pooling is only quasi-invariant, and

equivariance is limited by edge-effects in CNNs.
Now, let's imagine the same for scaling an image. If an input image is scaled-up/down, the output should be also scaled-up/down. As previously mentioned, we know that, by default, a convolutional layer doesn't have this property. To tackle this issue, scale-equivariant layers have to be defined. Scale-equivariant layers will be able to respond to scale differences in the same manner as a regular convolutional layer responds to input shifts. Scale-equivariance is derived from a mathematical concept: group-equivariance. Roughly speaking, group equivariant transformations mean that if the input of the layer is transformed by g, the output is also transformed by g. And g can be any homomorphism, for example: translation, rotation [2], scale [7], or the combination of these. By designing G-equivariant layers, we can further increase the weight sharing in a meaningful way.

How does the paper approach the problem?

Mathematical background

Reading the paper once, we see that the GitHub repository is public, we are happy that we can steal (I mean, reuse) the code, and have a free lunch... Well, before doing that, let's understand what's happening on a mathematical and algorithmic level. [1]

For starters, let's understand what is a steerable filter: A steerable filter is a type of kernel, where the scale of the kernel can be changed easily through a parameter. The mathematical definition provided by the paper is:

$$\psi_\sigma(x) = \sigma^{-1}\psi(\sigma^{-1}x) \tag{1}$$

The inner σ^{-1} is the scaling the filter, while the outer σ^{-1} is normalizing the filter. In this way we can rescale an arbitrary $\psi(x)$ filter by parameter σ – Figure 4 provides an intuitive visualisation.

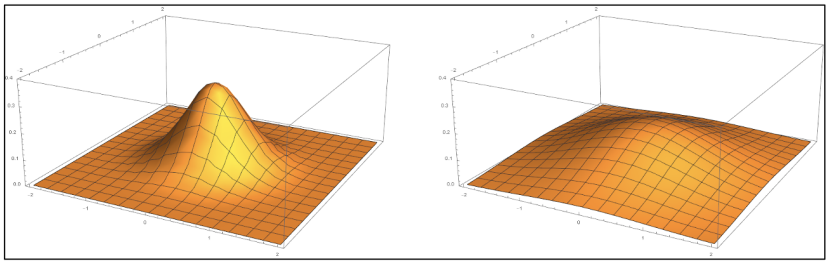

Figure 4: The same $\psi(x, y)$ gaussian 2D filter with $\sigma = 0.5$ on the left and $\sigma = 0.1$ on the right.

The second thing we should understand is the scale-translation group. The group is denoted by H and defined as a scaling operation followed by a translation operation. It is composed of two sub-groups, namely S and T.

The scaling group is denoted by S. The group operation is scaling, which is represented as multiplication by s, and the inverse for element s is $s^{-1} = \frac{1}{s}$. The unit element is $s * \frac{1}{s} = 1$ However, S is defined as a discrete scale group (to be more manageable mathematically at the Haar integral), consisting of the elements $[\dots a^{-2}, a{-}1, 1, a^1, a^2, \dots]$ where $a \in \mathbb{R}$ is a parameter of the model. In that case the inverse of a^n is a^{-n}.

The translation group is denoted by T. The group operation is obviously translation, and represented as addition of t. The inverse of the element of t is $-t$. The unit element is $t + (-t) = 0$. The translation group is left continuous, instead of being discretized to multiples of 1 pixel, because the continuous convolution on T is mathematically well defined (and it could be scaled afterwise).

To perform the group operation (i.e. scaling or addition), we have to apply the group element to the variable of the input function via the group operation (i.e. multiply or add to x respectively). A nice property of these two groups is that their semidirect product can be easily defined. A direct product is the

[1] The authors of the paper define everything for 1 dimensional signals, and "The generalization to higher-dimensional cases is straightforward". We think that an intuitive explanation isn't necessarily one dimensional, therefore our explanation won't stay there but we promise to keep the number of dimensions low.

group-theory equivalent of a cartesian product; a semi-direct product is just the generalization of the direct product. This operation can be imagined as an outer product of 2 vectors. The formal definition is $H = (s,t)|s \in S, t \in T$, which means that transformation h is a scaling s followed by a translation t. The group-operation is $(s_2,t_2)\cdot(s_1,t_1) = (s_2 s_1, s_2 t_1 + t_2)$. Transforming equation $(s_2,t_2)^{-1}\cdot(s_1,t_1) = (s_2^{-1}s_1, s_2^{-1}(t_1 - t_2))$ we can find the inverse element, which is $(s^{-1}, s^{-1}(-t))$. The unit element is $(1,0)$.

By defining group H we transformed the problem of finding a scale-equivariant convolution to finding a group-equivarient convolution. We transformed the problem from a specific one to a general one, in hope that we can find a solution for that. Luckily we followed the citations of the paper and found the definition of group-equivariant convolution:

$$[f \star_G \psi](g) = \int_G f(g') L_g[\psi](g') d\mu(g') = \int_G f(g')\psi(g^{-1}g')d\mu(g') \tag{2}$$

The mathematical details are insanely high, as multidimensional calculus and group theory are the minimum to have an idea what is going on. Therefore, we wouldn't recommend looking them up, but, if you chose to do so, we found [4, 3] to be a great source.
In the group-equivariant convolution $f(g')$ denotes our input signal which corresponds to the image, or the scale equivariant input at later layers. $L_g[\psi](g')$ (or $\psi(g^{-1}g')$ after the transformation) is the filter, and $\mu(g')$ denotes the Haar measure. After a bunch of mathematical transformation, we arrive at:

$$[f \star_H \psi_\sigma](s,t) = \sum_{s'} [f(s',\cdot) \star \psi_{s\sigma}(s^{-1}s',\cdot)](t) \tag{3}$$

One thing we haven't talked about yet are the channels. Regular convolutional layers sum over the input channels, and have a different filter for each input-output channel pair. Then, let's make our formula do exactly the same. In the equation below, C_{in} and C_{out} are the number of input and output channels respectively.

$$[f \star_H \psi_\sigma]_m(s,t) = \sum_{n=1}^{C_{in}} \sum_{s'} [f_n(s',\cdot) \star \psi_{n,m,s\sigma}(s^{-1}s',\cdot)](t), \quad m = 1...C_{out} \tag{4}$$

Algorithm

The aforementioned equation can't be implemented directly in code. Firstly, in the filter ψ the S-group is infinite; that needs to be further limited. Let N_S (later also denoted as S) be this limit, so group S becomes $[a, a^{-1}, \dots a^{-N_S}]$. Secondly, because the input image is defined on $\mathbb{Z}^2$ instead of $\mathbb{R}^2$, the group T has to be discretized. Following the author's choice, instead of defining each pixel in the kernel as a weight, each filter will be composed as a linear combination of a complete basis. While constructing functions this way, the dimension of the basis is often infinite (for example Taylor or Fourier series). Thus, we limit the algorithm to have an N_b dimensional basis. This way we can set the weights of the network to be the coefficients of the linear combination. Mathematically speaking: $\psi = \sum_i w_i \psi_i$, where w_i is a weight, and ψ_i is a basis vector. After this trick we can implement the magical equivariant convolution[2] [3].

Now, talking about the implementation side for each filter, we have a weight vector of length N_b (the coefficients of the linear combination), and the prescaled filter basis, of a shape $[N_b, S, V, V]$, where N_b is the number of bases, S is the number of scalings, and V is the spatial (x,y) size of the filter. When we have every C_{out}–C_{in} pair, this can be implemented efficiently, if the weights tensor have a shape of $[C_o ut, C_i n, N_b]$, multiplied by the precalculated bases of shape $[N_b, S, V, V]$. For the multiplication, we sum over the N_b dimension. For example by using the `torch.einsum('ijk, klmn -> ijlmn'), weights, bases)` function. For the output of this operation, we will have the filters in a shape of $[C_o ut, C_i n, S, V, V]$, denoted as κ. This is visualized in the next figure:

[2] Note 1: According to the paper, a basis of 2D Hermite polynomials with 2D Gaussian envelope works good enough

[3] This way, especially at larger filter sizes, the weight sharing is intensified. For example, a 7 by 7 convolutional kernel with 4th-order hermite polynomials takes only 10 parameters instead of the regular 49. While we can't reason logically if this weight sharing makes sense or not, our intuition tells us, that it does; this extra weight sharing could be another source for the increased accuracy alongside the scale-equivariant layers.

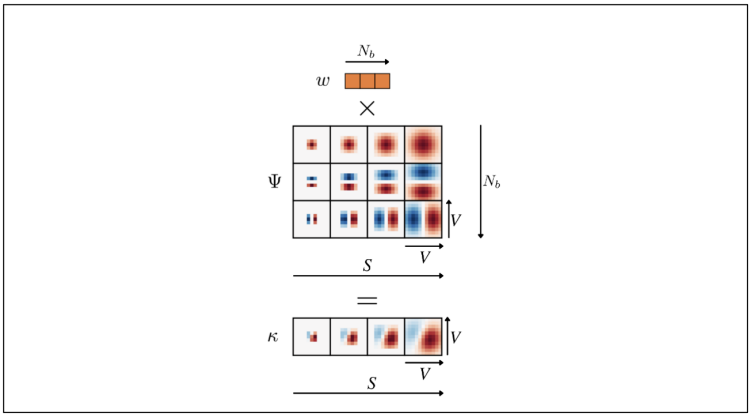

Figure 5: Visualization of the filter basis for a singular $C_{in} \to C_{out}$ channel.

Using the scale-translation invariant convolution's equation, we can define 2 scenarios. We tried to follow the author's notation: we refer as T→H to cases when the input of the layer has a scale dimension of 1, (also known as "image"), and the filter has multiple scale dimensions, and we refer as H→H to cases when both the input of the layer and the filter has multiple scale dimensions.
In the case of T→H, "the summation over S degenerates" was mentioned. We felt this line to be a little bit too ambiguous, therefore, we provide an alternative explanation: For each s in the scale dimension we perform a regular convolution between $\kappa[:, :, s, :, :]$ and the input image. The results of convolution is stored as an array of images, thereby producing an operation leading from group T to H. To leverage the already optimized PyTorch libraries, the T→H this can be implemented in the following form:

$$\texttt{convTH(f, w, } \psi\texttt{) = squeeze(conv2d(f, expand(w } \times \psi\texttt{)))}$$

In this case, the input κ filters are expanded from the shape of $[C_{out}, C_{in}, S, V, V]$ to the shape of $[C_{out}S, C_{in}, V, V]$. After this, the expanded filter base is convolved with the input image (which has the shape of $[C_{in}, U, U]$, where U is the size of the image). The output of this convolution yields a tensor of shape $[C_{out}S, U, U]$, which squeezed the dimensions $[C_{out}, S, U, U]$. While we have implemented this layer from scratch, we have not used it; due to time constraints, we were forced to chose another approach in order to meet the project's deadline [4] [5] [6].

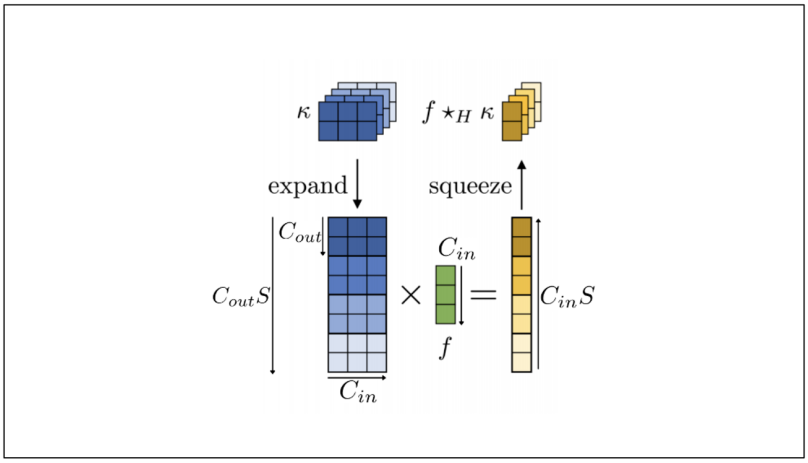

Figure 6: Visualisation of convolution $T \to H$. Spatial components are hidden for simplicity.

The case of $H \to H$ can be imagined as doing a convolution in the direction of x, y, and s, where the direction s. However, as this would extend the mathematical presentation even further, we would like to think that we have provided enough intuition for $T \to H$ in order for one to be able to understand $H \to H$ on his own.

[4] Note 1: For the shape of the expanded κ the paper defines the size $[C_{out}, C_{in}S, V, V]$ instead of $[C_{out}S, C_{in}, V, V]$. We believe the authors made quite a painful typo here.

[5] Note 2: The input image has a size of $[U, U]$ because the datasets used for benchmarking were using square images. Nothing restricts this to have the shape of $[U_1, U_2]$, but we've decided to follow the author's notation.

[6] Note 3: When applying the convolution the output doesn't always have the size of $[U, U]$, it is modified by padding, kernel size, and stride in the normal way.

Data preprocessing and augmentation

In regards to hand-down experiments, we had to process and augment data for two datasets, namely MNIST and STL-10. For each, as mentioned in the paper, we followed the specific steps.

MNIST

We rescaled the MNIST [5] images using a uniformly sampled factor between 0.3 - 1 and padded the images with zeros to retain the resolution of the initial images.

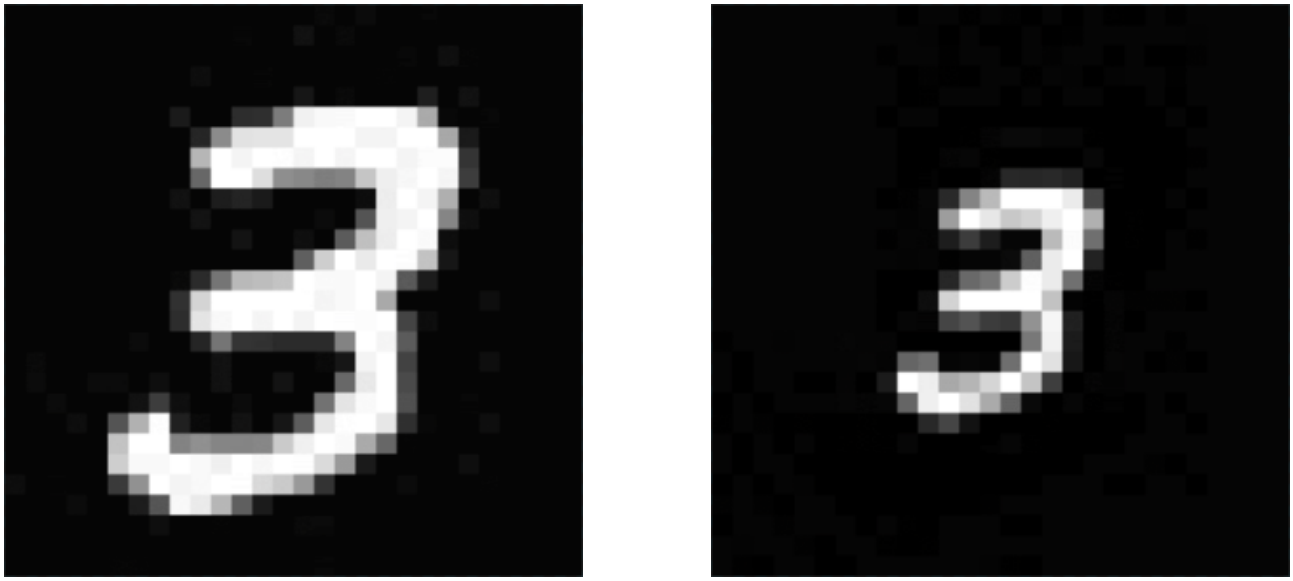

Figure 7: Example of rescaling MNIST image.

Furthermore, we generated 6 realizations of this dataset. 10.000 for training, 2.000 for evaluation, and 48.000 for testing.

STL-10

Similarly, for the STL10 [1] dataset, we followed the paper's instructions for getting data ready for the experiments. We normalized the images subtracting the per-channel mean and dividing by the per-channel standard deviation, augmented by applying 12 pixel zero padding and randomly cropping back to the 96x96px dimensions. Also, we used the horizontal random flips (50% probability) and cutout of 1 hole of 32 pixels.

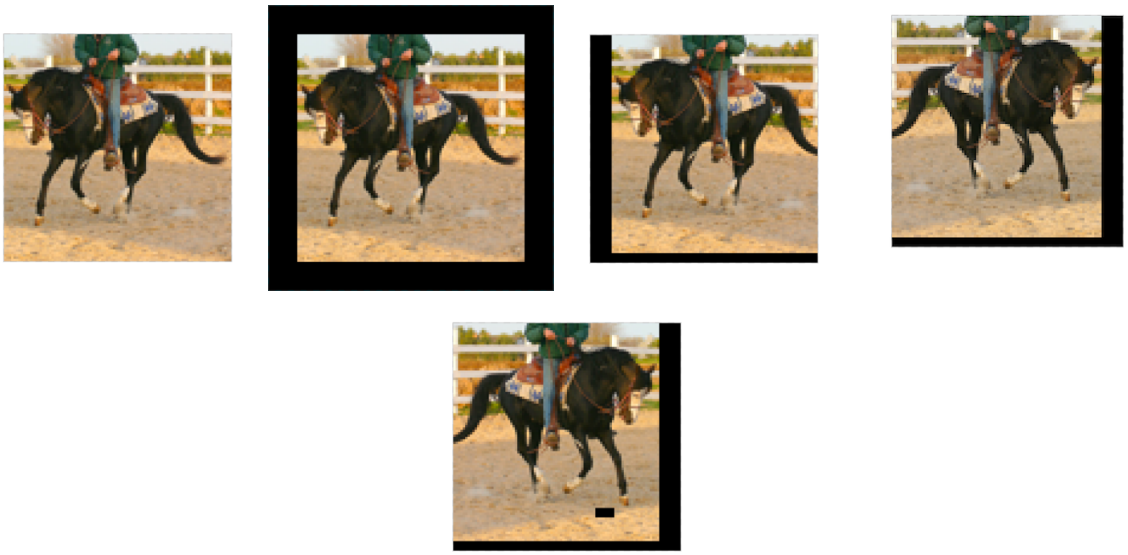

Figure 8: Data processing on an STL–10 image (normalization step not included in the figure).

Results

Running the code [6] we were able to replicate the experiments for the MNIST dataset. Although the code for STL-10 was available, we were unable to replicate the results, due to the fact that we did not have enough computational resources to run the experiments. After replicating the modified MNIST dataset (presented in the previous section), we were able to reproduce the results. These are somewhat similar, with small differences. For the MNIST dataset, we ran the experiments on the following models:

1. mnist_ses_scalar_28
2. mnist_ses_scalar_56
3. mnist_ses_vector_28
4. mnist_ses_vector_56
5. mnist_ses_scalar_28p
6. mnist_ses_scalar_56p
7. mnist_ses_vector_28p
8. mnist_ses_vector_56p

After replicating the modified MNIST dataset, we were able to reproduce the results. For each of the models mentioned at the bottom of the previous section, 2 experiments were conducted: one for which the scaling factor was set to 1 and one for which the scaling factor was set to 0.5. Therefore, for each model, we obtained 12 results (6 for each realization). The results were stored in "results.yml" which was further processed using a python script written by us. We reproduced results similar to the author's'. The results are displayed in the table below. The STL10 models defined by the authors had 11M parameters. We tried to run these with

Method	**28 x 28**	**28 x 28 +**	**56 x 56**	**56 x 56 +**
SESN Scalar (ours)	1.95 ± 0.17	1.98 ± 0.14	1.68 ± 0.12	1.67 ± 0.18
SESN Scalar (paper's)	2.10 ± 0.10	1.79 ± 0.09	1.74 ± 0.09	1.50 ± 0.07
SESN Vector (ours)	2.00 ± 0.2	2.00 ± 0.21	1.66 ± 0.17	1.59 ± 0.16
SESN Vector (paper's)	2.08 ± 0.09	1.76 ± 0.08	1.68 ± 0.06	1.42 ± 0.07

Table 1: Replicated results for the MNIST dataset. The '+' denotes scaling data augmentation.

a reduced batch size (to fit into our GPU limit of 12GB), but the model was training too slowly in order to reproduce meaningful results. Subsequently, these were not included.

Conclusion

This blog was created in the context of TU Delft's CS4240 Deep Learning course, as a group project. By reading this, we hope that you now have a better understanding of what scale-equivariance is, and why designing scale-equivariant CNNs is valuable. Although, at the first glance, the paper looked really "mathy", we can say that after spending countless hours on Wikipedia articles trying to extend our knowledge in group theory, we kind of understood the authors' intent. We believe that G–equivariant convolutions (such as scale–equivariant) are a promising direction for CNNs, as their applicability is undeniable.

Acknoledgements & Links

We would like to thank Nergis Tömen and Tomasz Motyka for their advice and helpful insights.

Link to **paper**: https://arxiv.org/abs/1910.11093
Link to this **project's website**: https://spetrescu.github.io
Link to our ***GitHub* repository**: https://github.com/vioSpark/reproduction-project-DL2021
Link to our ***Medium* blog post**: https://lukacs-mark.medium.com/sesn-cec766026179

References

[1] Adam Coates, Andrew Ng, and Honglak Lee. "An Analysis of Single-Layer Networks in Unsupervised Feature Learning". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Geoffrey Gordon, David Dunson, and Miroslav Dudík. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, Nov. 2011, pp. 215–223. URL: http://proceedings.mlr.press/v15/coates11a.html.

[2] Taco Cohen and Max Welling. "Group equivariant convolutional networks". In: *International conference on machine learning*. PMLR. 2016, pp. 2990–2999.

[3] M.L. Eaton. *Group Invariance Applications in Statistics*. Regional conference series in probability and statistics. [Online; accessed 16-April-2021]. Institute of Mathematical Statistics, 1989. ISBN: 9780940600157.

[4] Gerald B. Folland. *A Course in Abstract Harmonic Analysis*. [Online; accessed 16-April-2021]. 1995.

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

[6] Stefan Petrescu and Mark Erik Lukacs. *Project's GitHub repository*. https://github.com/vioSpark/reproduction-project-DL2021. [Online; accessed 16-April-2021]. 2021.

[7] Ivan Sosnovik, Michał Szmaja, and Arnold Smeulders. "Scale-equivariant steerable networks". In: *arXiv preprint arXiv:1910.11093* (2019).

[8] Rikiya Yamashita et al. "Convolutional neural networks: an overview and application in radiology". In: *Insights into Imaging* 9.4 (Aug. 2018), pp. 611–629.