Scale-Equivariant Steerable Networks Reproducibility Project – CS4240 Deep Learning

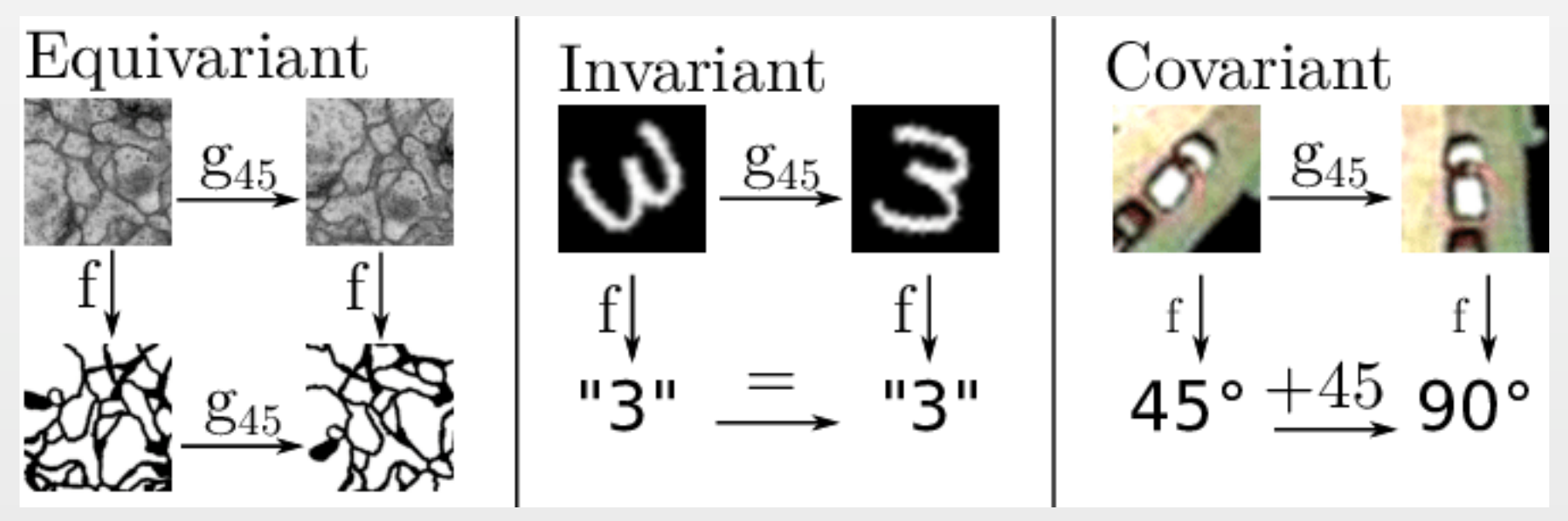
Mark Erik Lukacs, Stefan Petrescu

1 *Introduction*

This poster provides an overview of the reproducibility project for the Deep Learning Course. We reproduced the paper's results, running the experiments on our own, using the available code.

This poster provides an overview of the reproducibility project for the Deep Learning Course. We reproduced the paper's results, running the MNIST experiments on our own, using the available code.

The reason why CNNs excel in image processing is that convolutional layers are translation equivariant. This was achieved by excessive weight-sharing.
A special case of equivariance (which is valuable in image recognition tasks) is invariance. It is achieved by the feature-extractor pooling layers.
Translation-equivariance means that if the input is shifted, the output is shifted as well, while translation invariance means that the output is the same, no matter how the input is shifted.
Equivariance and invariance can be defined on other transformations as well, for example figure 1 demonstrates rotation equivariance and invariance. However, in the case of our paper the authors tackled the question of combining translation and scale, and producing scale-translation equivariant/invariant neural network.



2 *Method*

The mathematical background, they define group-convolution, and implement an efficient algorithm.

Scale-equivariance is derived from a mathematical concept: group-equivariance. A group equivariant transformation means that if the input of the layer is transformed by transformation *g*, the output is also transformed by *g*. Here *g* can be any diffeomorphism, like rotation, mirroring , scale, or something less meaningful, like inversion.

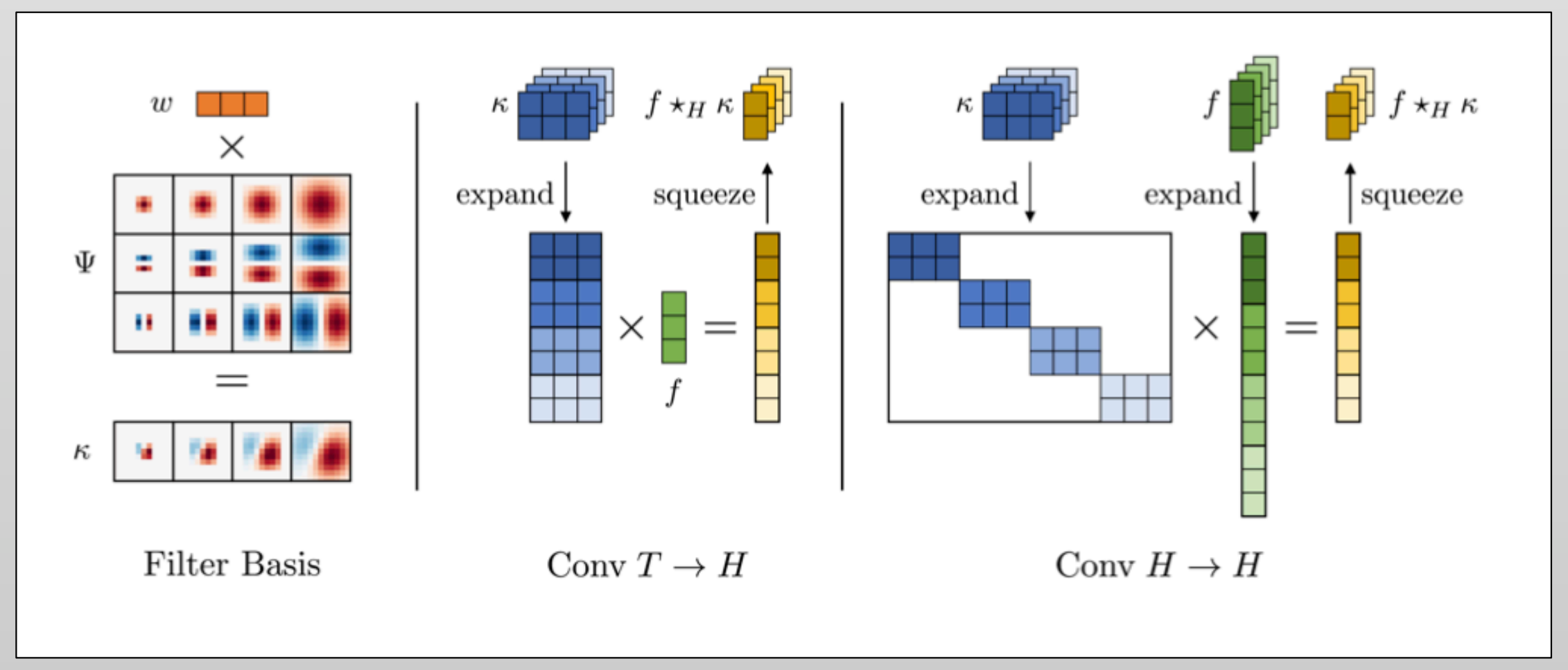
The authors defined group H, as a combination of a translation followed by a scaling. They have applied group convolution (a generalization of convolution), and transformed it into equation 1:

$$[f \star_H \psi_\sigma]_m(s,t) = \sum_{n=1}^{C_{\text{in}}} \sum_{s'} [f_n(s',\cdot) \star \psi_{n,m,s\sigma}(s^{-1}s',\cdot)](t), \quad m = 1 \dots C_{\text{out}} \qquad (7)$$

Implementation of \psi_{n,m,s\sigma} directly is not possible, therefore the authors decided to limit the number of scales to N_S, and represented each filter as a *steerable filter*. This means that each filter is a linear combination of N_b bases (each base is a 2D Hermite polynomial with a gaussian envelope, repeated N_S times at different scales), where the learnable parameters are the values of the linear combination.

Since the first layer has no scale-information embedded, this layer differs from the other layers. Using equation 7, the summation over S degenerates, and the T->H convolution can be defined as:
convTH(f, w, Ψ) = squeeze(conv2d(f, expand(w × Ψ)))
We have implemented this type of layer from scratch, however we weren't able to use it in our experiments.

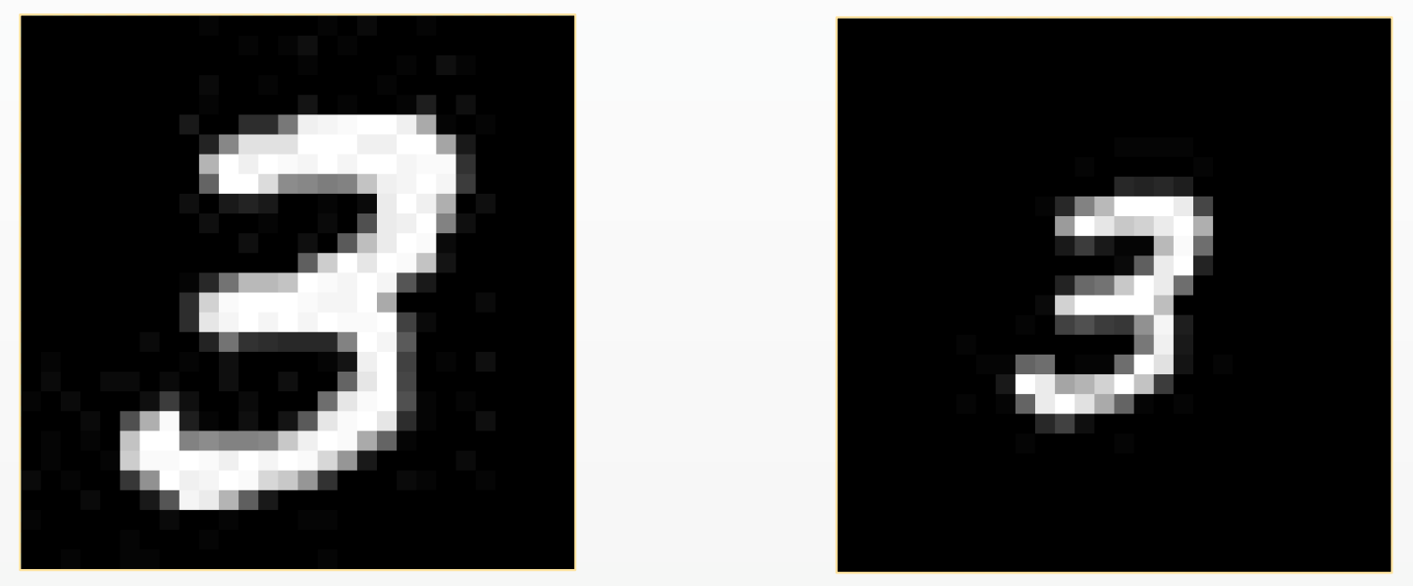
In the case of the following layers the scale axis has information too. If the filters have only one scale, equation 7 degenerates the same way, and the H->H layer can be defined as:
convHH(f, w, Ψ) = squeeze(conv2d(expand(f), expand(w × Ψ)))
If the filters have different scales, inter-scale interaction happens, and the output should be calculated by convolving (with convHH) the input tensor, and w individually for each scale, and summing it afterwards. Figure 2 sumarizes convTH and convHH.



The invariance is coming from the pooling layers, where max-pooling is applied to both the scale and translational dimension in a similar manner it is applied to translational dimensions in a regular convolutional layer.

3 *Experiments*

For the experiments, two datasets were considered: MNIST [REF] and STL-10 [REF]. For both we had to process data, following the specific paper guidelines. For the MNIST, the data processing step consisted of rescaling the images, based on a randomly sampled uniform factor, between 0.3-1. Following this procedure, 6 different such realizations were generated. Each consisted of 60000 images, each split into 10000 for training, 2000 for evaluation, and 48000 for testing.
To realize this, we created some python scripts, which can be found on our project's GitHub repository.



For the STL-10 dataset, data augmentation was also applied. Thus, in this case the images were normalized, padded with a 12px border (2) and then randomly cropped (3) to their initial 96x96px size. Furthermore, random horizontal flips (4) with a probability of 50% were applied & cutout of 32px (5).

5 *Results*

Being able to replicate the MNIST experiments, we were able to reproduce the results. These are somewhat similar, with some small differences.
For the MNIST dataset, we ran the experiments on the following models:

- mnist_ses_scalar_28
- mnist_ses_scalar_56
- mnist_ses_vector_28
- mnist_ses_vector_56
- mnist_ses_scalar_28p
- mnist_ses_scalar_56p
- mnist_ses_vector_28p
- mnist_ses_vector_56p

For each of these, 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 later processed using a python script. These can be found in the table below.

Method	28 x 28	28 x 28 +	56 x 56	56 x 56 +
SESN Scalar	1.95 ± 0.17	1.98 ± 0.14	1.68 ± 0.12	1.67 ± 0.18
SESN Vector	2.00 ± 0.2	2.00 ± 0.21	1.66 ± 0.17	1.59 ± 0.16

Acknowledgements & Links

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

Link to paper:
https://arxiv.org/abs/1910.11093

Link to our project's website:
https://spetrescu.github.io

Link to our project's GitHub repo:
https://github.com/vioSpark/reproduction-project-DL2021