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 CS4240 Deep Learning Course.

The project represents our attempt at reproducing the results of a scientific paper, namely Scale-Equivariant Steerable Networks (SESN) by Ivan Sosnovik, Michał Szmaja and Arnold Smeulders.

One of the main reasons for which CNNs excel in image recognition & classification is their underlying structure, equivariant to translations i.e. the translation of input features results in an equivalent translation of outputs. This is being achieved by weight sharing across the receptive field of the neurons in particular layers. However, CNNs are not equivariant to scale changes of the input. In real-world scenarios, it is desirable to recognize objects at different scales, as this can be applied and used in many domains. For example, this can be applied in computer vision for autonomous vehicles. Thus, we find that this paper solves a very important & exciting problem, that of constructing CNNs being equivariant to scale changes of the input.

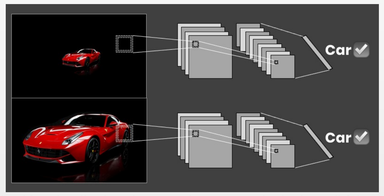

Figure: Intuitive visualization, the network can recognize both small & large scale objects.

Firstly, regarding the project's workflow, we analyzed the math and the algorithm presented in the paper. Here we saw how the scale-translation equivariant convolution is designed. Furthermore, we took a look at the algorithm representing these convolutions. Last but not least, we went through the data pre-processing steps and reproduced the paper's results, running the MNIST experiments on our own, using the paper's available code. We were not able to reproduce the STL10 results, as this would have required access to computational power inaccessible to us at the moment.
Before presenting the actual method, we consider important mentioning the difference between equivariance and invariance. The latter is a special case of the prior – achieved by the feature-extractor pooling layers present in CNNs' architecture.
In comparison to the already mentioned translation equivariance for which if the input is shifted the output is shifted as well, 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 in the figure below 1 rotation equivariance and invariance are visualized. However, in our case, the authors tackled the problem of combining translation and scale – producing a scale-translation equivariant/invariant neural network.

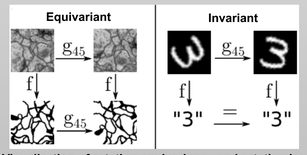

Figure: Visualization of rotation equivariance and rotation invariance.

2 *Method*

In the paper the authors start from a mathematical background. They first define scale-translation equivariant convolution and show an efficient algorithm for it.

Scale-equivariance is derived from the mathematical concept of 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 homomorphic mapping, like translation, rotation, mirroring, or scale. The authors defined the equivariant group *H*, as a translation followed by a scaling. They have applied group convolution (a generalization of convolution), and transformed it into equation 7:

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

Figure: The definition of scale-translation equivariant convolution. Here C_{in} is the number of input channels, C_{out} is the number of output channels, $\psi_{n,m,s\sigma}$ is the corresponding scaled-filter (with learnable weights), and f_n is the input image.

Implementation of equation 7 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 basis (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 were unable to use it in our experiments.

In the case of the following layers, the scale axis contains 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 is be calculated by convolving (with *convHH*) the input tensor, and w individually for each scale, and summing it afterwards. The figure below summarizes *convTH* and *convHH*.

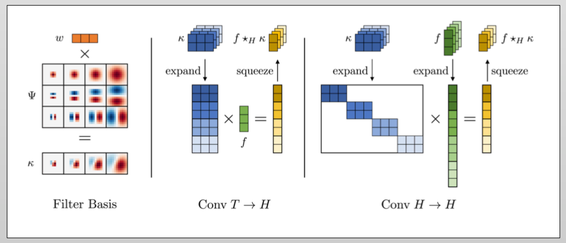

Figure: Visual representation of the scale-translation equivariant convolutions (spatial components are hidden).

The invariance of the network is coming from the pooling layers, where max-pooling is applied to both the scale and translational dimensions (similarly applied to translation dimensions in regular CNNs).

3 *Experiments*

For the experiments, two datasets were considered: MNIST and STL-10. For both, we pre-processed the data, following the paper's specific 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 realizations were generated. Each realization consisted of 60000 images, split into 10000 for training, 2000 for evaluation, and 48000 for testing. Furthermore, depending on the model (next section), we upscaled the images using bi-linear interpolation.

Figure: Example of image rescaling for MNIST; (left) Image before and (right) after rescaling.

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

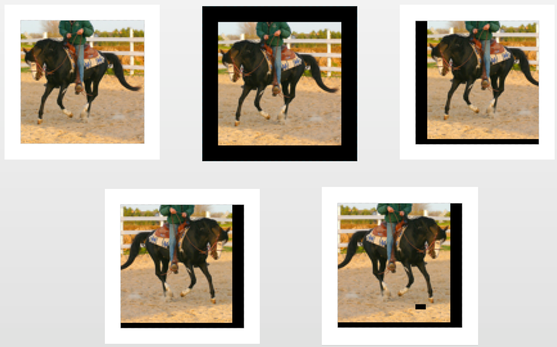

Figure: STL-10 pre-processing pipeline (for better visualization purposes normalization by mean and standard deviation were not included in the figures); Initial image (top left), followed by 12px zero padding (middle), random cropping to the initial 96x96px dimension, random horizontal flipping with 50% probability (bottom left), and cutout of 32px (bottom right).

Although we tried ourselves to implement the method, we encountered some difficulties. Thus, in order to run the experiments, we used the available code, both for MNIST and STL-10.
For the MNIST dataset, we experimented with 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

The experiments took approximately 14 hours, and ran on Google Collaboratory (using a NVIDA Tesla K80).

5 *Results*

After replicating the modified MNIST dataset, we were able to reproduce the results.
For each of model 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 reproduces results similar to the authors'. The results are displayed in the table below.

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

The STL10 models defined by the authors had 11M parameters. We tried to run these with reduced batch size (to fit into our GPU limit of 12GB), but the model was training too slowly to reproduce meaningful results. Subsequently, these were not included.

6 *Conclusion*

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.

In conclusion, for this project, we have reproduced the data augmentation steps, re-run the MNIST experiments, and produced quasi-equivalent results. We tried to run the STL–10 experiments but the lack of computational resources got the better of us. We tried to implement the layers themselves and, although we weren't able to test it, we believe that we have successfully implemented the T→H layer.

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