Modélisation et traitement d'image - Projet
Involution: Inverting the Inherence of Convolution for Visual Recognition

Hadrien Salem
hadrien.salem@centrale.centralelille.fr
Emilie Salem
emilie.salem@centrale.centralelille.fr
Capucine Garçon
capucine.garcon@centrale.centralelille.fr
Thomas Waldura
thomas.waldura@centrale.centralelille.fr

Abstract

The goal of the article Li et al. (2021) is to put forward a new operator for neural network building that serves as an alternative to convolution. As opposed to convolution, which is **spatial-agnostic** and **channel-specific**, the new involution operator proposed by the authors is **spatial-specific** and **channel-agnostic**, which means that behavior is shared between channels. Involution has two main advantages over convolution: (1) the weights prioritize important information of the image in an **adaptive way**, and (2) they capture interactions between pixels on a **larger scale**. The operator is tested on image classification, object detection and segmentation state-of-the-art benchmarks, and it is shown that it yields better results than convolution, both in **accuracy** and in **computational cost**. In this report, we summarise the main aspects of Li et al. (2021)'s article and conduct our own experimentations with involution layers on different image classification datasets. Our experiments are available on GitHub at `https://github.com/SnowHawkeye/mti-project-involution`.

1 Context and type of approach

The convolution operator, which is spatial-agnostic and channel-specific has enabled the rise of deep learning in Computer Vision. This operator is a pillar in constructing deep neural networks due to its compactness and its spatial-agnostic property. Indeed, since the creation of the VGGNet, the spatial span of convolution kernel can now be restricted to 3 x 3. However, the spatial-agnostic property deprives them of their ability to spatially detect different patterns.

We are now going to see the mathematical description of the convolution operator.

1.1 The convolution operator

Let us consider an image $X \in \mathbb{R}^{H \times W \times C_i}$ on which we want to perform convolution. H is its height, W its width and C_i its number of channels. Inside our image, each feature vector $X_{i,j} \in R^{C_i}$ is considered as a pixel.

Now, we can define the formulation of a group of C_o convolution filters $\mathcal{F} \in R^{C_o \times C_i \times K \times K}$ where each filter is $\mathcal{F}_k \in R^{C_i \times K \times K}$ with $k = 1, 2..., C_o$. Therefore, we have C_i kernels of size $K \times K$ that execute Multiply-Add operations on the input feature map in a sliding window. This operation is repeated C_o times in order to obtain an output feature map $Y \in R^{H \times W \times C_o}$:

35th Conference on Neural Information Processing Systems (NeurIPS 2021), Sydney, Australia.

$$Y_{i,j,k} = \sum_{c=1}^{C_i} \sum_{(u,v)\in\Delta_K} \mathcal{F}_{k,c,u+\lfloor K/2 \rfloor, v+\lfloor K/2 \rfloor} X_{i+u,j+v,k}$$

1.2 Limitations of convolution

Although convolution is a powerful operator in many deep neural networks, these networks are computationally demanding, especially with the convolutional layers that consume most of the processing time. This then restricts their deployment. Also, as we have seen before, their spatial-agnostic property limits them in capturing spatial interactions in a single shot.

In the next part, we will look at the involution operator, which is the heart of the article and which has enabled the overcoming of some of the limitations mentioned above.

2 The involution operator

The involution operator is very similar to the convolution operator, except for two main differences:

- Instead of using the same convolution kernel for each pixel, a specific involution kernel is created for each pixel using a kernel generation function (**spatial-specific**). It is the parameters of this kernel generation function that are learnt during training.
- The involution operation is shared across group of channels instead of being specific to each of them (**channel-agnostic**).

2.1 Mathematical description of involution

Let us consider an image $X \in \mathbb{R}^{H\times W\times C}$ on which we want to perform involution, H being its height, W its width and C its number of channels.

Let us then define the involution kernel $\mathcal{H} \in \mathbb{R}^{H\times W\times K\times K\times G}$, where H is the height of the image previously defined, W is its width, K is the size of the filter and G is the number of groups of channel that share the same involution kernel. In other words, we have $H \times W$ filters of size $K \times K$ for each of the G groups of channels.

For each pixel $X_{i,j} \in \mathbb{R}^C$, we tailor a specific involution kernel $\mathcal{H}_{i,j}$ using the kernel generation function ϕ defined as followed:

$$\mathcal{H}_{i,j} = \phi(X_{i,j}) = W_1\sigma(W_0X_{i,j})$$

where:

- $W_0 \in \mathbb{R}^{C\times C}$ and $W_1 \in \mathbb{R}^{K\times K\times G\times C}$ are two linear transformations.
- σ performs batch normalization and non-linear activation between the two linear transformations.

We can therefore consider ϕ as a "bottleneck" neural network, where the weights of the matrices W_0 and W_1 are the parameters to learn.

Once the involution kernel of a pixel is computed, the output of the involution operation is obtained by performing Multiply-Add operations on the input, similarly to convolution:

$$Y_{i,j,k} = \sum_{(u,v)\in\Delta_K} \mathcal{H}_{i,j,u+\lfloor K/2 \rfloor, \lceil kG/C \rceil} X_{i+u,j+v,k}$$

The whole process of involution is illustrated by Figure 1. Here are some important aspects of involution:

- All involution kernels are built from the same function ϕ whose parameters are learnt by back-propagation. This leads to a drastic reduction in the number of parameters for the layer, compared to convolution.
- Involution kernels are shared across channels. Nevertheless, channels can be separated into a number of groups G, which is a hyperparameter of involution layers.
- Just like convolution, a padding strategy needs to be defined for pixels on the edges of the image.

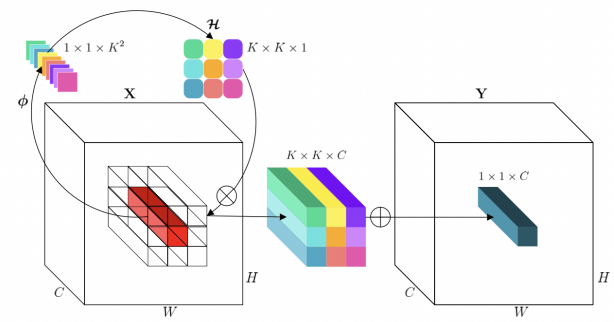


Figure 1: Schematic illustration of our proposed involution. The involution kernel $\boldsymbol{\mathcal{H}}_{i,j} \in \mathbb{R}^{K \times K \times 1}$ ($G = 1$ in this example for ease of demonstration) is yielded from the function $\boldsymbol{\phi}$ conditioned on a single pixel at (i, j), followed by a channel-to-space rearrangement. The Multiply-Add operation of involution is decomposed into two steps, with $\bigotimes$ indicating multiplication broadcast across C channels and $\bigoplus$ indicating summation aggregated within the $K \times K$ spatial neighborhood. Best viewed in color.

Figure 1: Illustration of the involution operation (Li et al. (2021))

2.2 Comparison to the attention mechanism

Involution can be seen as a generalization of the self-attention mechanism. Indeed, let us consider the output of the self-attention operation:

$$Y_{i,j,k} = \sum_{(p,k) \in \Omega} (QK)^T_{i,j,p,q,\lceil kH/C \rceil} V_{p,q,k}$$

where Q, K and V are respectively the query, key and value matrices, obtained by performing linear transformations on the input X. Their weights are what is learnt during training.

We can notice that the formula is very similar to the involution one, notably:

- Multiply-Add operations are performed, with a weighted sum on a neighborhood of pixels (noted Δ for involution and Ω for self-attention).
- $V_{p,q}$ can be seen as the equivalent of the pixel $X_{i,j}$.
- The product QK^T can be seen as the equivalent of the convolution kernel $\mathcal{H}$.

This similarity with self-attention serves as justification for good properties of involution. Indeed, self-attention is a relation-based mechanic whose goal is to define which parts of the input "deserve more attention", meaning that they are more important in the learning process. Despite some differences, the similarity between the two processes explains that involution is capable of retaining relevant information within an image.

2.3 Results on state-of-the-art benchmark

2.3.1 Classification results

The classification experiments provided by Li et al. (2021) were performed on the ImageNet dataset. On the same principle as the ResNet family, the authors have created a family of involution networks, called RedNet, by varying their depth.

The comparison was established between the RedNet family and convolution-based and self-attention-based models (ResNet, LR-Net, SAN). The results are the following:

- The RedNet family provides a better recognition accuracy for each depth with less computational demand and generally fewer parameters, as shown in Table 1 in the article Li et al. (2021).
- RedNet allows for a better balance of accuracy-parameters and accuracy-complexity compared to state-of-the-art networks based on self-attention like SAN and Axial ResNet, as shown in Figure 2 in the article Li et al. (2021).

2.3.2 Object detection and Instance segmentation

To test the ability of involution to capture objects on images, the authors (Li et al. (2021)) have adapted the RetinaNet, the Faster R-CNN and the Mask R-CNN networks with ResNet and RedNet backbones.

They have compared the different networks with the ResNet backbone, with convolution-based neck and head. The results are the following:

- The three different networks with the RedNet backbone show better results compared to their ResNet-based counterpart in terms of average precision, while being more parameter- and computation-conserving as shown in Table 3 in the article Li et al. (2021).
- When the convolution is changed to involution in the necks and then in the heads (RedNet backbone), we again notice an improvement in the results (in terms of bounding box average precision) and a reduction in the number of parameters and in the computation cost as shown in the Table 3 in the article Li et al. (2021).

3 Numerical illustrations

In this section, we compare convolution to involution in some experiments of our own. The goal is to try and find the results obtained in the article, but on different datasets and with different setups, so as to test some limitations of involution layers.

Our experiments use a TensorFlow implementation of the involution layer provided by Gosthipaty (2021). Even though we did not make use of it for this project, a PyTorch implementation of the operator also exists, provided by Reich et al. (2021).

For each experiment, we will be comparing the performance of two models of the exact same architecture, but one will be using traditional convolution layers while the other will be using involution layers.

This comparison will take into account:

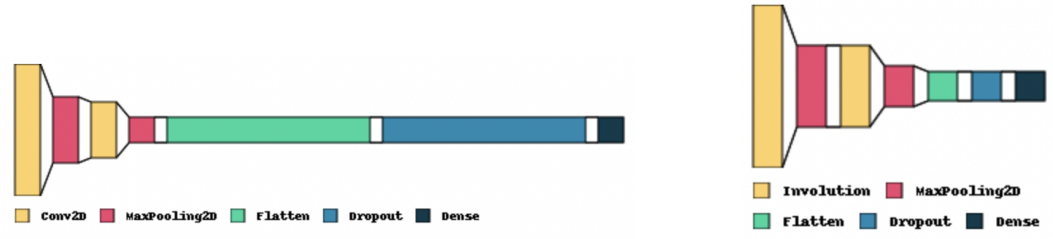
- The accuracy of the models on the validation set;
- The number of parameters of the models;
- The time required to train the models with identical setups on GPU and CPU.

In the following subsections, we will precise the experimental setup for different datasets (MNIST, Fashion-MNIST and CIFAR10). All results are regrouped in Table 1.

All of our experiments are condensed into a jupyter notebook available at `https://github.com/SnowHawkeye/mti-project-involution`. Some of the training parameters (batch size and number of epochs) may differ slightly from what is presented here, but the observed results are the same.

3.1 Experiment on MNIST

We conduct a first experiment on the well-known MNIST handwritten digits dataset, which has 60,000 training samples and 10,000 test samples of 28×28 greyscale images. **For convolution**, the network architecture we use is a simple succession of two 2D convolutions and max poolings, followed by a flattening layer, a dropout layer, and a fully connected layer with softmax activation. This architecture is inspired from LeCun et al. (2010). **For involution**, we use the exact same architecture, but we replace all convolution layers by involution layers. The two architectures are displayed in Figure 2. We train our networks on 15 epochs with batches of size 128. Results are displayed in Figure 3 and Table 1.



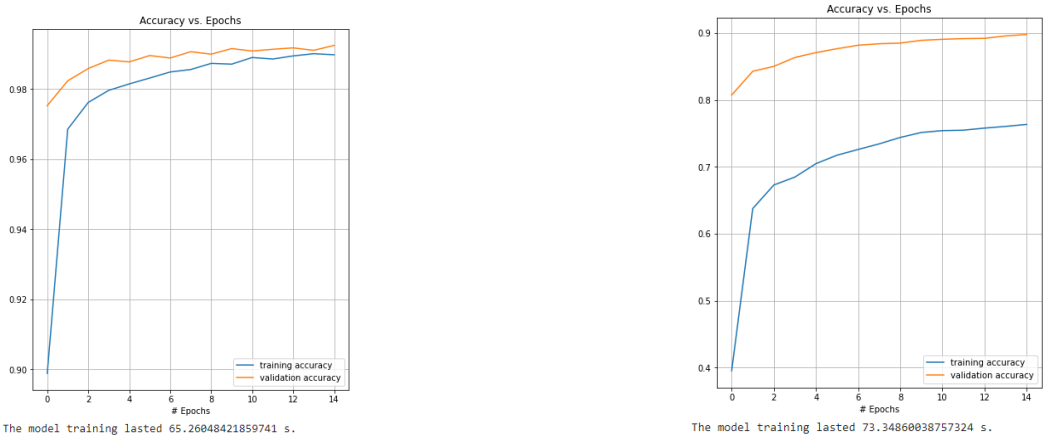
(a) Convolution (b) Involution

Figure 2: Architecture of the networks for the experiment on MNIST.

The first difference to notice is the large gap in terms of number of parameters: 34,826 for convolution, versus 544 for involution. There is however a compromise to compensate for this gain: the validation accuracy for involution reaches about 90%, whereas convolution quickly goes up to 99%. Nevertheless, the performance of involution remains relatively good, and could probably be further improved by a more suited architecture and a longer training time.

Although we did note a significant diminution in the number of parameters for involution, it was not reflected in training times, with convolution being faster to train on 15 epochs (65 seconds versus 73 seconds on a GPU). We suppose that the reason for that is that we used an optimised Keras implementation of convolution, and a non-optimised implementation of involution. To confirm this conjecture, we try executing the tests again on a CPU, and this time involution becomes faster (323 seconds, instead of 503 seconds for convolution). This demonstrates our intuitive assumption that at equal levels of optimisation, involution is faster to train due to having much less parameters.

It is also interesting to note that the accuracy on the training set is significantly worse for involution, which means it is able to generalise very well. This can be an advantage in cases where convolution would overfit.



(a) Convolution (b) Involution

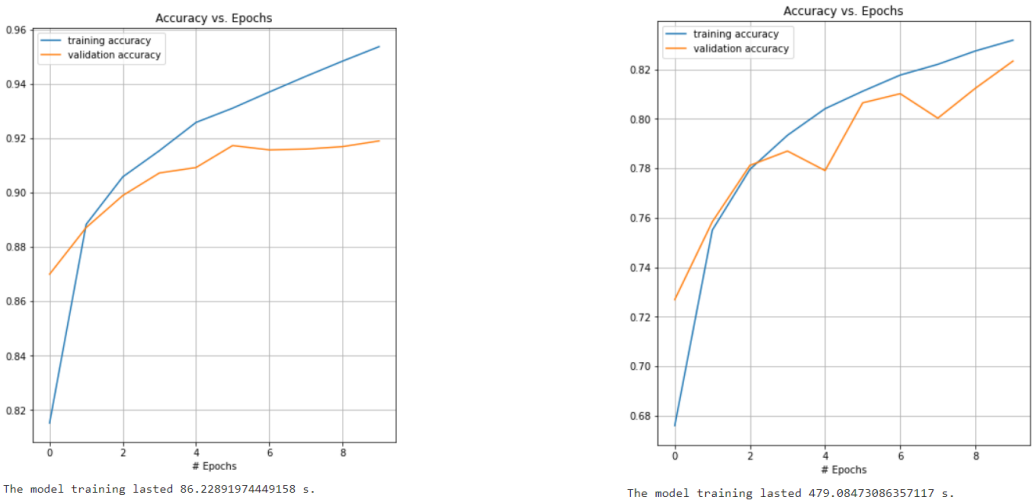
Figure 3: Results on MNIST dataset.

3.2 Experiment on Fashion MNIST

In a second experiment, we add an extra convolution (resp. involution) layer in the networks, and train them over the fashion MNIST dataset provided by Zalando. The training set contains 60,000 samples, and the test set contains 10,000 samples, all of which are 28×28 greyscale images. The two architectures are displayed in Figure 4, and results are displayed in Figure 5 and Table 1.

(a) Convolution (b) Involution

Figure 4: Architecture of the networks for the experiment on Fashion MNIST.



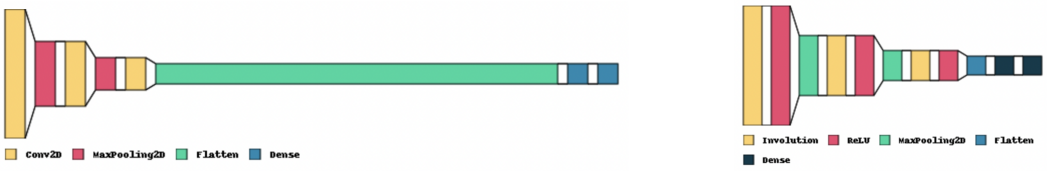
(a) Convolution (b) Involution

Figure 5: Results on Fashion MNIST dataset.

The observations we make are very similar: convolution has a lot more parameters to train, but provides better results in terms of accuracy. The gap in performance with involution is widened on this more complex dataset, with a difference of about 10% in validation accuracy.

3.3 Experiment on CIFAR10

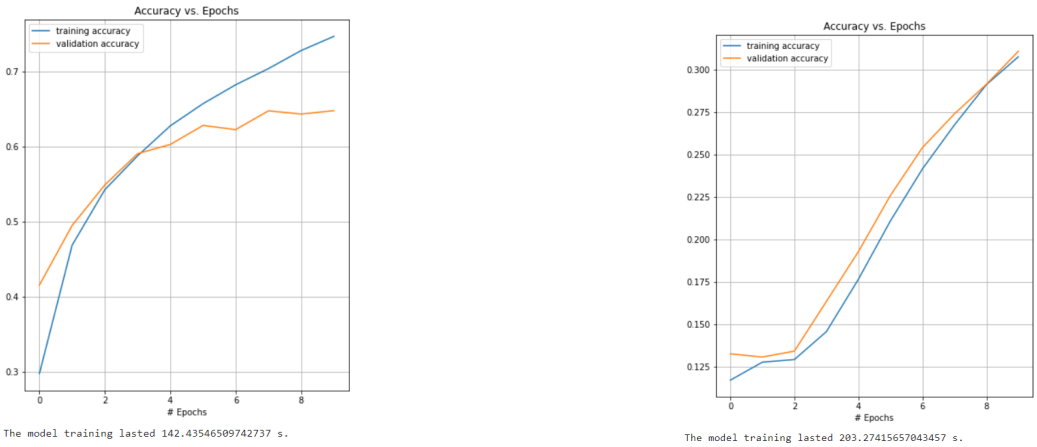
We conduct a last experiment on the CIFAR10 dataset, which contains 60,000 32×32 colored images belonging to ten different classes. Once again we use quite a simple network, composed of a succession of two 2D convolutions and max poolings, followed by a flattening layer and two fully-connected layers, the last one using softmax activation. The two architectures are displayed in Figure 6.



(a) Convolution (b) Involution

Figure 6: Architecture of the networks for the experiment on CIFAR10.

Since the CIFAR10 dataset is even more challenging for image classification than MNIST and Fashion MNIST, we don't expect to get a satisfying accuracy with such a simple network. However the purpose of the experiment being solely to compare convolution and involution, this is not an issue for us. Results are displayed in Figure 7 and Table 1.



(a) Convolution (b) Involution

Figure 7: Results on CIFAR dataset.

The results are once again very similar: the involution network has way less parameters to train (13,074, instead of 319,178), which has an impact on the training time on CPU (922,53s for involution, 1077,86s for convolution) but not on GPU (203,27s for involution, 142,44s for convolution) due to the unoptimized implementation. As for the validation accuracy, it is significantly lower for involution (31,10% on GPU) than for convolution (64,78% on GPU).

This last experiment further proved what we had already observed on Fashion MNIST: the performance gap between involution and convolution is significantly widened on more complex datasets. Indeed, the difference in validation reaches more than 30% this time, and involution's advantage in training speed is not as striking as on MNIST and Fashion MNIST (only 150s less than convolution).

Even though these results might partly be due to a poor architecture design on our part, they also show that involution might not be as advantageous for very complex datasets, and tends to shine on more simple tasks.

3.4 Conclusions on our results

The results of the previously described experiments are summed up in table 1. We can draw several conclusions from them:

- Convolution performs systematically better than involution in terms of validation accuracy, with the performance gap being larger on more complex datasets. However, our comparisons were conducted by replacing convolution layers by involution layers in architectures that were initially designed for convolution. A more refined design of involution networks could help bridge the performance gap, as shown by the authors of Li et al. (2021).
- One of the main advantages of involution is the drastic reduction in number of parameters, with involution networks having up to 65 times less parameters than equivalent convolution networks. This leads to a significant reduction in computing time on CPUs. Theoretically, the same could be observed on GPUs, but we were unable to reuse the authors' GPU implementation of convolution.

Table 1: Results of our experiments

		Convolution	**Involution**
Accuracy	*MNIST*	**98,97 %**	88,82 %
	Fashion-MNIST	**91,91 %**	82,34%
	CIFAR10	**64,78 %**	31,10 %
Number of parameters	*MNIST*	34 826	**544**
	Fashion-MNIST	257 162	**3 916**
	CIFAR10	319 178	**13 074**
Training time	*MNIST (GPU)*	**65,26 s**	73,35 s
	MNIST (CPU)	502,75 s	**323,23 s**
	Fashion-MNIST (GPU)	**86,23 s**	479,08s
	Fashion-MNIST (CPU)	982,03 s	**621,97 s**
	CIFAR10 (GPU)	**142,44 s**	203,27 s
	CIFAR10 (CPU)	1077,86 s	**922,53 s**

4 Conclusion

In conclusion, involution is a very promising operator: by learning a kernel generation function instead of individual kernels and by sharing kernels across channels, it allows for very light-weight networks. It performs slightly worse than convolution in terms of accuracy but this is compensated by its fast training time and good generalization capacity, especially on simpler problems. For the moment involution suffers from the lack of an easily-accessible CUDA implementation that prevents it from being widely used, but it definitely has the potential to become an interesting alternative to convolution in the future.

References

Gosthipaty, A. R. (2021). Tensorflow implementation of involution. `https://github.com/ariG23498/involution-tf`.

LeCun, Y., Kavukcuoglu, K., and Farabet, C. (2010). Convolutional networks and applications in vision. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pages 253–256.

Li, D., Hu, J., Wang, C., Li, X., She, Q., Zhu, L., Zhang, T., and Chen, Q. (2021). Involution: Inverting the inherence of convolution for visual recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12321–12330.

Reich, C., Memmel, M., and shikishima TasakiLab (2021). Pytorch implementation of involution. `https://github.com/ChristophReich1996/Involution`.