JOURNAL OF APPLIED
COMPUTER SCIENCE
Vol. 24 No. 3 (2016), pp. 33-48

Geometric Transformations Embedded into Convolutional Neural Networks

Paweł Tarasiuk, Michał Pryczek

Lodz University of Technology
Institute of Information Technology
ul. Wolczanska 215, 90-924 Lodz, Poland
{pawel.tarasiuk,michal.pryczek}@p.lodz.pl

Abstract. *This paper presents a novel extension to convolutional neural networks. While CNNs are known for invariance to object translation, changes to the other parameters could make the image recognition tasks difficult – that includes rotations and scaling. Some improvement in this area could be achieved with embedded geometric transformations used inside the CNNs. In order to provide a practical solution, which allows fast propagation and learning of the modified networks, "fast geometric transformations" are introduced.*
Keywords: *artificial intelligence, machine learning, deep learning, convolutional neural networks, image processing, image recognition, geometric transformations.*

1. Introduction

Convolutional neural networks (CNNs) are a state-of-art solution to the image classification problems. Solutions based on CNNs are regular winners [1, 2, 3] of the image classification category of ImageNet Large Scale Visual Recogintion Challenge [4]. It is a great example of how effective a relatively simple solution can be – CNNs are easy to use, because the input is usually just a raw digital image,

with optional very basic preprocessing (scaling, normalization, etc.) [1]. What is more, CNNs are easy to understand, as the whole architecture is a modification of multilayer perceptron (MLP) with reduced connections between layers and extensive weights sharing [5]. The standard gradient-based backpropagation learning is commonly used for training the CNN-based classifiers.

Biologically inspired by the visual cortex of a cat [6], CNNs are indifferent to small changes in the recognized object, especially translations [7]. However, no indifference to significant scale or rotation changes is provided by design – while a network with sufficient number of neurons could simply learn multiple rotations or scales. This approach is actively researched and improved [8]. However, some specific methods of making rotation-invariant CNNs without excessive number of neurons are pursued as well. One promising approach it to use transformation-specific pooling mechanism (such as orientation-pooling introduced in [9]) and use siamese CNNs which process multiple rotations in parallel branches [10].

The general reason why support for different rotations poses a problem to CNN is related to the structure of the "visual fields". Each output of a hidden CNN layer is a result of processing a axis-aligned rectangular range of the previous outputs (down to the input image). As result, the commonly used CNNs always learn to recognize pixel patterns related to the Chebyshev metric. The relation between two pixels is recognized only when both intersect with the same visual field, which is simply a ball in Chebyshev space (fig. 1). In order to recognize another kinds of patterns (which could be invariant to rotation, scaling, or both), another pixel topology needs to be applied. Some of the recent research on that topic [11] is very general, which makes it impossible to take the full advantage of actual hardware. However, some custom topologies could be achieved by embedding additional transformations into CNN architecture, as it is proposed in this paper (fig. 2). This approach is more limited in terms of possible configurations, but leaves the further processing to the optimized operations which are typical for CNNs.

This paper describes a possible solution to embed topology-changing geometric transformations into CNNs. "Fast geometric transformations" are intended solution to performing the additional operations in a reliable time. Another advantage of the proposed mechanism is a compatibility with standard methods of CNN learning.

Figure 1: Classical CNN: output map, shape of a visual field.

Figure 2: Logarithmic-polar transform + CNN: different topology makes it possible to achieve custom visual field shapes.

2. Fast geometric transformations

Image transformations such as translation, scaling, rotation, and cropping are the classical examples of geometric transformations. This term could be easily generalized to any transformation, where value (color) at each point of the output is a convex combination of values at some input points. This approach easily corresponds with an intuition that "geometric transformation" should operate only on the shape of image. However, pixels of the output image should only use colors that were already present or transitions between them.

Using the direct formula for rotations or polar-logarithmic transform [12] is noticeably slow, as trigonometric and exponential functions are called for each pixel. As presented in this paper, this calculations could be simplified when a large number of images is processed, and input and output sizes of a transformation are fixed. As each pixel of the result is a linear combination of input pixels, the

whole transformation is a *linear operator*. Caching the coefficients of this operator reduces the requirement of specific calculations to the preparation stage only, making it possible to process the actual images with improved performance. Using coefficients other than 0 or 1 is natural when the preimage of a single output pixel intersects with multiple input pixels, but it is useful in the general case as well, as it makes it possible to describe anti-aliasing.

The direct formulas for some transformations (such as rotation) are likely to suggest that the preimage of some output pixels exceeds the boundaries of input image, so the result is undefined. In order to provide the best approximation for the undefined regions, the weighted sum of the closest existing pixels could be used instead. It guarantees the result to be smooth. This approach could be considered equivalent to the infinite repetition of the input image bounding pixels.

Following the assumptions described above, it is possible to define the approximate linear operators for: translation, rotation, cropping and superposition of any operators of such kind. What is more, for each of the operators mentioned above, approximate inverse operator could be introduced – one possible application is "inverse cropping", which is easily achieved by the repetition of boundary pixels. This is enough to suggest a formal definition of geometric transformation operators. Let S be a vector space over $\mathbb{R}$ (it applies e.g. to digital image in RGB or CIELAB). Any transformation of $n \times m$ into $p \times q$ map could be described as $f : S^{n \times m} \to S^{p \times q}$ such that:

$$f_T(A) = \left(\sum_{t=0}^{n-1} \sum_{u=0}^{m-1} T_{rstu} \cdot A_{tu} \right)_{r=0 \ldots p-1,\, s=0 \ldots q-1}, \tag{1}$$

where T_{rstu} is 4-dimensional "matrix" $[0; 1]$ ($T \in [0; 1]^{p \times q \times n \times m}$) which satisfies:

$$\forall_{r=0 \ldots p-1,\, s=0 \ldots q-1} \quad \left(\sum_{t=0}^{n-1} \sum_{u=0}^{m-1} T_{rstu} \right) = 1. \tag{2}$$

The property above states that each element of $f_T(A)$ is a convex combination of elements of A.

Should the "matrix" T be once known for fixed transformation and image sizes, the further calculation would involve only additions and multiplications, without any need to calculate trigonometrical, exponential or logarithmic functions numerically. What is more, since the preimage of each pixel has a surface which is much smaller than the whole input image, matrices T_{rs} for any $r = 0 \ldots p - 1$, $s =$

$0 \ldots q-1$ are likely to be sparse – only small percentage of elements are anything else than zero. As a result, the whole T_{rs} matrices could be effectively approximated by some finite number of it's elements with the highest coefficients. The selected coefficients could be further normalized to make a proper convex combination, in case the sum of coefficients becomes less than 1.

The process of calculating T which describes the geometric transform (where $g : X \to Y$, $[0;n] \times [0;m] \subseteq X \subseteq \mathbb{R}^2$, $[0;p] \times [0;q] \subseteq Y \subseteq \mathbb{R}^2$ is a homeomorphism from input image to output image) with anti-aliasing could be performed in the following way:

$$\forall_{r=0\ldots p-1,\, s=0\ldots q-1, t=0\ldots n-1, u=0\ldots m-1} \quad T_{rstu} = \frac{(d((t,u),\, g^{-1}(r,s)) + \varepsilon)^{-1}}{\left(\sum_{t'=0}^{n-1} \sum_{u'=0}^{m-1} (d((t',u'),\, g^{-1}(r,s)) + \varepsilon)^{-1}\right)}, \tag{3}$$

where $\varepsilon > 0$ is necessary to avoid division by 0 in case of a point which matches the center of preimage perfectly. We consider d as any metric on $\mathbb{R}^2$ (the results are clear and smooth for Minkowski space of power greater than 2, e.g. L_3).

Instead of condirering the whole input image ($t = 0 \ldots n-1, u = 0 \ldots n-1$), calculating T_{rs} could be limited to points (t,u) such that:

$$d_{Ch}((t,u),\, g^{-1}((r,s))) \leq \frac{1}{2} \quad \vee \quad d_{Ch}(g((t,u)),\, (r,s)) \leq \frac{1}{2}, \tag{4}$$

where d_{Ch} is a Chebyshev metric on $\mathbb{R}^2$. This condition is intended to ensure that:

- each input image pixel affects at least one pixel of the output image,
- each output image pixel has nonempty preimage under simplified operator.

Additionally, as it was mentioned above, the number of considered T_{rs} elements could be limited a priori with some constant number $M \geq 1$ of elements with the highest coefficients. All the other elements of T_{rs} are approximated as zeros.

For all the practically used geometric transformations (including translation, rotation, scaling, cropping and polar-logarithmic transformation), the homeomorphism g is given by a direct formula, so a corresponding transformation T is guaranteed to exist. Additional properties which can be stated include that:

- for each homeomorphism g which could be described by operator T, the inverse function g^{-1} could be used to define an approximate "inverse operator" T^{-1},

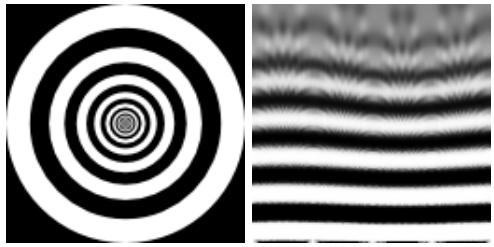

Figure 3: Rings of exponentially growing width are transformed into horizontal bard of equal height (limited resolution in the image center results in a noise).

- if T and U are linear operators, then so is their superposition TU.

2.1. Polar-logarithmic transform

Polar-logarithmic transform [12, 13] is a useful utility for image analysis, when invariance to rotation scale is pursued.

Each segment with an endpoint in the image center is transformed into a horizontal segment of the output map, while each centered circle is transformed into a vertical segment. As a result – columns are images of circular sectors (fig. 4), and rows of the input matrix are images of centered rings. Width of a preimage rings grows exponentially for the consequent rows (fig. 3).

Let us consider a polar-logarithmic transformation of $w_0 \times h_0$ image into a result with w_1 columns and h_1 rows. The concise formula is easy to present when you describe each (x, y) point on the plane as a complex number $z = x + iy$. The transformation could be defined by $g : \mathbb{C} \to \mathbb{C}$ with the following formula:

$$g(z) = \left(\frac{\ln\left(\|z - c\| + \varepsilon\right)}{\ln\left(\frac{\min\{w_0, h_0\}}{2} + \varepsilon\right)} \cdot (h_1 - 1) \right) + i \left(\frac{\mathrm{Arg}(z - c) + \pi}{2\pi} \cdot (w_1 - 1) \right), \tag{5}$$

where c is a center of the original image, given by $c = \frac{w_0 - 1}{2} + i\frac{h_0 - 1}{2}$ and ε is a small positive number (illustrations in this paper were generated for $\varepsilon = \frac{1}{4}$. Too big ε causes the central point to spill over notable amount of transformed image, and too small epsilon makes the images of boundary pixels take too much space.

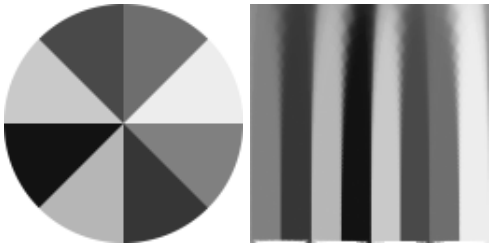

Figure 4: Circular sectors of equal angle are transofrmed into vertical bars of equal width.

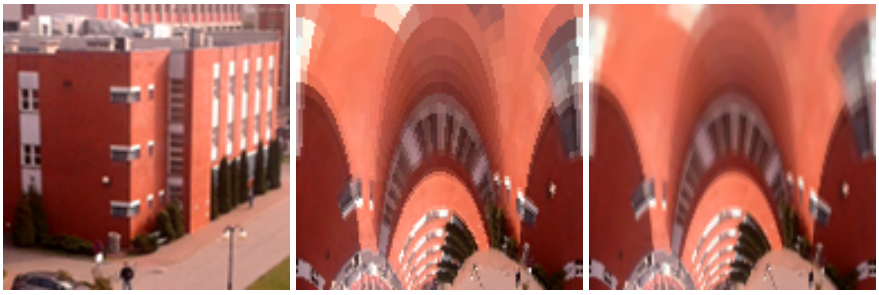

Figure 5: Sample 128×128 image and the results of polar-logarithmic transform for 1 (middle) and 100 (right) highest elements.

2.2. Benchmark results

The method described in the previous section was implemented in `Python` [14] with `numpy` [15]. The `numpy` module allows all the matrix operations be implemented with a single call which is performed on the low level – actually `numpy` arrays are equivalent to dynamic arrays in `C`. As the result we get a concise implementation without any typical disadvantages to performance which could be expected in case of scripting languages.

2.2.1. Polar-logarithmic transform

Transforming 100 images of 128×128 size into polar-logarithmic transforms (of 128×128 size as well) took:

- with direct function g formula used for each image: 88.40 s,

Figure 6: Sample image and the results of 30 degree rotation performed: using `Python Imaging Library` (middle) and using the cached linear operator (right).

- with linear operator T which was only calculated once: 0.39 s.

To sum up, using the solution presented above brought 200x improvement in calculation time in the example setup.

2.2.2. Image rotation: Python Imaging Library [16] vs proposed transformation description

Rotating 1000 images of size 128×128 by 30 degree:

- with `Python Imaging Library`: 1.2 s,

- with cached linear operator: 0.9 s.

The custom method with linear operators was slightly faster than the dedicated solution from `Python Imaging Library`. However, image rotation is a relatively simple operation. However, the proposed model of linear operators makes the calculation time independent from the original formula. Especially, performing a sequence of rotations, translations and cropping with `Python Imaging Library` would take time equal to the sum of times of each operation. The method presented in this paper makes it possible to reduce the superposition of the whole sequence to a single linear operator, which could be used with the same performance as a single rotation. The same remark applies to single geometric transformations with structurally complex formulas, such as polar-logarithmic transformations.

3. Extended CNN architectures

Performing a geometric transformation on input data could be performed outside of the CNN, at the data preparation stage. However, since CNN input is a raw image, the initial layers are known to perform very basic image filtering (Gabor-like features detection) [17], which is known to reduce the noise. What is more, the denoising achieved in convolutional neural networks is adjusted specifically to the learning objective, because the convolutional filters are adapted in the learning process.

In order to keep the advantage of specifically-trained initial CNN layers, the proposed architectures involve geometric transformations on the hidden layer outputs. That means that with each iteration transformation needs to be applied to all output channels. Repeating the same transformation so frequently is the reason why fast geometric transformations is a key requirement for this method. Direct approach to geometric transformations would be more complex than any standard CNN layer, while applying cached linear operator takes less operations than the convolutional layer. As result, the introduced method is unlikely to become a bottleneck in either propagation or learning process.

Another way how the proposed definition of fast geometric transformations makes them useful for CNNs is existence of approximate inverse transformations. This fact is used in back-propagation learning – applying an inverse transformation to the gradient data makes it possible to propagate the errors correctly to the previous layers.

The simplified application of the geometric transformations is described in fig. 7 (“variant I”). In order to avoid combining matrices from different topologies, the processing of different transforms could be separated – resulting in a siamese network denoted as “variant II” (fig. 8).

In both variants of the extended CNN architecture, different sets of geometric transformations could be considered. The results presented in this paper include: multiple rotations and polar-logarithmic transform.

3.1. Results

The experiments were performed on datasets based on Street View House Numbers [18]. This dataset consists of house number digits cropped from Google Street View images. As house numbers usually consist of multiple digits, only the digit in a center of the cropped region is considered in the related image classifica-

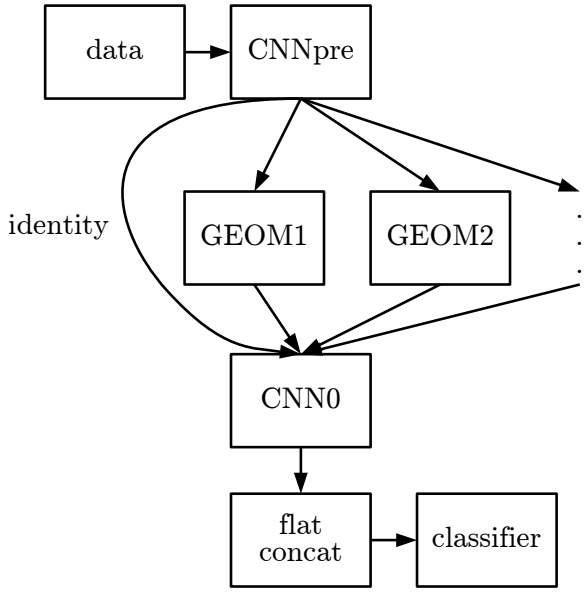

Figure 7: Embedding geometric transformations into CNN: "variant I" (simplified): results of geometric transformations form additional channels for the further convolutional layers.

tion task. Different colors and variable context make this problem difficult, so there is no known machine learning method that performs the classification without errors. The original version consists of 73257 training images and 26032 images in a test set. All the images have fixed size 32×32.

However, in order to analyze the suggested models in terms of rotations invariance, additional random rotations were added to both training and test subsets. The "$\pm 22.5°$" dataset was generated by replacing each image from SVHN by the result of it's three random rotations, with angle chosen uniformly from $[-22.5°, 22.5°]$ range. The other variant – "$\pm 45°$" – was generated in a similar way, but with twice wider range of possible rotations.

Diagrams presented on figures 7 and 8 use some common labels. The explanation and relation to the actual neural network model is presented below.

- `CNNpre` consists of convolutional layers used in the initial processing of a raw image. In all the presented experiments `CNNpre` was a single convolutional layer with 5×5 filters and 200 output channels.

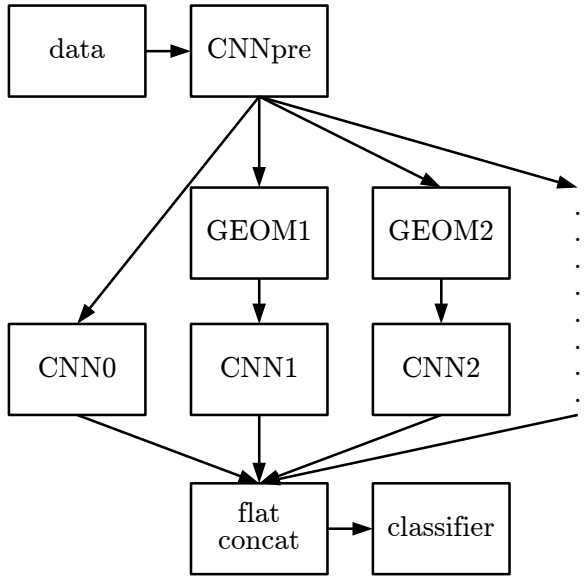

Figure 8: Embedding geometric transformations into CNN: "variant II" (siamese): parallel branches of the network process results of different transformations, in order to be used together as a classifier input.

- `GEOMk` are geometrical transformations. The experiments involve 0-2 units of this kind embedded into the network.
- `CNNk` are sequences of convolutional layers used for further processing, typical for convolutional neural networks. In the performed experiments each unit labeled this way consists of 5 consequent convolutional layers with 5×5 filters and 100 output channels each.
- `flat concat` is simply a vector concatenation. Numbers from different bottom layers can be stored together in order to be propagated further.
- `classifier` is just a multi-layer perceptron. It contains a hiddent fully-connected layer with 2000 neurons and uses Dropout [17] with coefficient $p = 0.5$. The next layer is simply output – a layer with 10 neurons, where `Softmax` function can be used to determine the classification results.

For each dataset the tests were performed on a few network architectures:

- "basic" – just a plain CNN. It actually fits both "variant I" and "variant II" when the number of geometric transformations is 0.

- "variant I/II + rotations" – for each variant, the network architecture with two rotations (`GEOM1` and `GEOM2`) was tested. Rotations angles are α and $-\alpha$, where α is adjusted to the dataset (15° for "±22.5°" dataset and 30° for "±45°" dataset). Since the identity (0° rotation) is present in both variants, the selected values guarantee that result of some of the three outputs will be close to the unrotated sample (the maximal possible difference is $\alpha/2$).

- "variant I/II + logpolar" – the only geometric transformation involved is `GEOM1` – polar-logarithmic transform.

The results are presented in table 1. In case of ±22.5° dataset, both variants worked quite well with embedded rotations – considering three rotations seems to be sufficient to cause a notable improvement in case of this range. Using polar-logarithmic transform in case of this dataset resulted in some improvement when compared to the basic CNN, but the results were not as good as in case of rotations.

When the rotations range was increased to ±45°, some more interesting results were achieved. Lack of rotation invariance of CNNs is clearly visible, as the results are generally much worse than in ±22.5° case. In order to make the comparison between datasets honest, the number of neurons was not increased for the second dataset – doing so (and accepting multiple times slower propagation and learning) would possibly lead to some better results.

The most interesting part is, however, comparison between the presented methods. Each of the proposed variants resulted in an improvement when compared to "basic" CNN, so all those ways of increasing the rotation invariance are useful to some extent. Considering only three rotations for such a large range wasn't enough to provide the best results in this category. The results of "variant I + logpolar" architecture was pretty similar to those with rotations. That is already interesting, since polar-logarithmic transformation changes the topology completely. Using the resulting channels together with original ones is unlikely to make much sense – this variant was tested only for the sake of completeness of the results. The sensible option, which is "variant II + logpolar" was, however, significantly the best solution for ±45° dataset. When the range of rotations is wide, a transformation which represents rotations as translations (which CNNs naturally recognize well) makes the classification notably easier. While using larger number of rotations can have a positive impact on the results, it is only possible at the cost of additional

Table 1: Summary of results.

Method	Dataset	Top-1 accuracy	Top-3 accuracy
basic	±22.5°	83.767%	94.801%
I + rotations$_{15}$	±22.5°	86.529%	95.911%
II + rotations$_{15}$	±22.5°	86.115%	95.846%
I + logpolar	±22.5°	84.438%	94.954%
II + logpolar	±22.5°	84.870%	95.274%
basic	±45°	39.185%	69.964%
I + rotations$_{30}$	±45°	43.923%	71.384%
II + rotations$_{30}$	±45°	42.925%	71.086%
I + logpolar	±45°	43.937%	71.986%
II + logpolar	±45°	49.111%	73.784%

calculations. Remarkably, "logpolar" variants were actually faster than "rotations", since only one geometric transformation / branch was used instead of two.

4. Conclusions

Convolutional neural networks, successful as they are for general image classification tasks such as ILSVRC [4], can be improved when specific tasks are considered. One specific case is recognition of objects from the same class in different rotations. Since the recognized patterns are closely related to the Chebyshev metric, additional flexibility – such as invariance to rotation or scale – could be achieved by processing the images in a different topology. In some cases this could be achieved by embedding geometric transformations into the network architecture.

In order to make the extended architecture practical, geometric transformations need to be calculated quickly. Since the parameters of the geometric transformation (including input and output size) are likely to be constant, it could be approximated by a linear operator with a sparse matrix. Caching the operator drastically reduces the time needed to perform any further operations. What is more, after the linear operator is calculated, it does not matter what it represents – complex transformations (such as polar-logarithmic) and superpositions of multiple transformations perform equally fast as one simple transformation.

When geometric transformations are described as linear operators, the approximate inverse operations can be calculated. This provides a way to describe the

appropriate steps for back-propagation learning of CNNs with embedded transformations.

The achieved results show that additional channels which are not related to the standard behavior of CNN are likely to improve the rotation invariance. Considering multiple image rotations is a good idea, but small number of transformations is likely to become insufficient when the range of possible rotations is large. One possible solution to process the rotation-invariant patterns is to use embedded polar-logarithmic transform.

In the further research it would be useful to compare the results of extending the CNN by geometric transformation to simply increasing the number of neurons. Another case worth testing is to propose a mixed architecture that uses both embedded rotations and polar-logarithmic architecture. Last but not least, the properties of fast geometric transformations could be used in even more extensive way when superpositions of multiple transformations are considered. The last idea could be used to propose a single model which combines advantages of "variant I" (using CNN layers on maps of features related to different topologies) with "variant II" (multiple dedicated CNN layers for each custom topology).

References

[1] Krizhevsky, A., Sutskever, I., and Hinton, G. E., *ImageNet Classification with Deep Convolutional Neural Networks*, In: Advances in Neural Information Processing Systems 25, edited by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Curran Associates, Inc., 2012, pp. 1097–1105.

[2] Zeiler, M. D. and Fergus, R., *Visualizing and Understanding Convolutional Networks*, CoRR, Vol. abs/1311.2901, 2013.

[3] Nguyen, T. V., Lu, C., Sepulveda, J., and Yan, S., *Adaptive Nonparametric Image Parsing*, CoRR, Vol. abs/1505.01560, 2015.

[4] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L., *ImageNet: A Large-Scale Hierarchical Image Database*, In: CVPR09, 2009.

[5] Fukushima, K., *Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position*, Biological Cybernetics, Vol. 36, 1980, pp. 193–202.

[6] Hubel, D. H. and Wiesel, T. N., *Receptive Fields and Functional Architecture in Two Nonstriate Visual Areas (18 and 19) of the Cat*, Journal of Neurophysiology, Vol. 28, 1965, pp. 229–289.

[7] LeCun, Y. and Bengio, Y., *Convolutional Networks for Images, Speech, and Time-Series*, In: The Handbook of Brain Theory and Neural Networks, edited by M. A. Arbib, MIT Press, 1995.

[8] Cheng, G., Zhou, P., and Han, J., *Learning Rotation-Invariant Convolutional Neural Networks for Object Detection in VHR Optical Remote Sensing Images*, IEEE Transactions on Geoscience and Remote Sensing, Vol. 54, No. 12, Dec 2016, pp. 7405–7415.

[9] Gonzalez, D. M., Volpi, M., and Tuia, D., *Learning rotation invariant convolutional filters for texture classification*, CoRR, Vol. abs/1604.06720, 2016.

[10] Laptev, D., Savinov, N., Buhmann, J. M., and Pollefeys, M., *TI-POOLING: transformation-invariant pooling for feature learning in Convolutional Neural Networks*, CoRR, Vol. abs/1604.06318, 2016.

[11] Vialatte, J., Gripon, V., and Mercier, G., *Generalizing the Convolution Operator to Extend CNNs to Irregular Domains*, CoRR, Vol. abs/1606.01166, 2016.

[12] Weiman, C. F. R. and Chaikin, G., *Logarithmic Spiral Grids for Image Processing and Display*, Computer Graphics and Image Processing, Vol. 11, No. 3, November 1979, pp. 197–226.

[13] Tomczyk, A., Szczepaniak, P. S., and Lis, B., *Generalized Multi-layer Kohonen Network and Its Application to Texture Reognition*, In: Proceedings of the Lecture Notes in Artificial Intelligence, No. 3070, 2004, pp. 760–767.

[14] Foundation, P. S., *The Python Language Reference*, 1990–2016.

[15] Ascher, D., Dubois, P. F., Hinsen, K., Hugunin, J., and Oliphant, T., *Numerical Python*, Lawrence Livermore National Laboratory, Livermore, CA, ucrl-ma-128569 ed., 1999.

[16] PythonWare, *Python Imaging Library (PIL)*, 2009–2016.

[17] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R., *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*, J. Mach. Learn. Res., Vol. 15, No. 1, Jan. 2014, pp. 1929–1958.

[18] Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y., *Reading Digits in Natural Images with Unsupervised Feature Learning*, In: NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011, 2011.