International Journal of
Intelligent Systems and Applications in Engineering

ISSN:2147-6799 www.ijisae.org Original Research Paper

Rolling in the Deep Convolutional Neural Networks

Derya Soydaner*[1]

Submitted: 15/11/2019 Accepted : 12/12/2019

Abstract: *Over the past years, convolutional neural networks (CNNs) have achieved remarkable success in deep learning. The performance of CNN-based models has caused major advances in a wide range of tasks from computer vision to natural language processing. However, the exposition of the theoretical calculations behind the convolution operation is rarely emphasized. This study aims to provide better understanding the convolution operation entirely by means of diving into the theory of how backpropagation algorithm works for CNNs. In order to explain the training of CNNs clearly, the convolution operation on images is explained in detail and backpropagation in CNNs is highlighted. Besides, Labeled Faces in the Wild (LFW) dataset which is frequently used in face recognition applications is used to visualize what CNNs learn. The intermediate activations of a CNN trained on the LFW dataset are visualized to gain an insight about how CNNs perceive the world. Thus, the feature maps are interpreted visually as well, alongside the training process.*

Keywords: *Convolutional Neural Networks, Deep Learning, Image Processing*

1. Introduction

Convolutional neural networks (CNNs) are a specialized kind of neural network for processing data that has a known grid-like topology [1]. They have been used in image recognition since the 1980s. Over the years, with the aid of the increase in computational power and the amount of available training data CNNs have achieved significant performance on some complex tasks such as visual perception, voice recognition and natural language processing [2]. Since the early 2000s, CNNs have been applied with great success to the detection, segmentation and recognition of objects and regions in images [3]. During this period, several architectures have been proposed such as LeNet [4], AlexNet [5], DenseNet [6], ResNet [7], VGG [8], Inception and GoogLeNet [9]. The aim of this study is to shed light on the common working principles and calculations behind the convolutional layers of these successful architectures. Because it is often said that deep learning models are *black boxes*. Although this is generally true for certain types of deep learning models, it is definitely not true for CNNs [10]. The convolution operation and backpropagation in CNNs can be expressed clearly. Besides, the representations learned by CNNs can be extracted and displayed visually.

In deep learning literature, there exists a genuine interest in understanding and visualizing CNNs. For example, a comprehensive survey of several representative CNN visualization methods is provided [11]. In another study, a novel visualization technique that gives insight into the function of intermediate feature layers and the operation of the classifier is introduced [12]. They use these visualizations to find model architectures that outperform AlexNet on the ImageNet classification benchmark. Also, three visualization methods namely inversion, activation maximization and caricaturization are studied [13]. Two visual tools are introduced to interpret neural networks [14]. Besides, in order to classify knowledge representations in high convolutional layers a new method to modify a traditional CNN into an interpretable CNN is proposed [15]. By using three different methods, a comparison of heatmaps on three datasets is demonstrated [16]. More recently, existing activation maximization methods are reviewed and a probabilistic interpretation for these methods are discussed [17]. In comparison with these recent studies, this work is focused on understanding CNN training process in addition to the meaning of convolutional layers. CNNs are examined on a face recognition task to make them more interpretable.

In this work, a brief background information about CNNs is given in Section 2. Additionally, convolution operation on images is explained in detail as well as how the backpropagation algorithm works for CNNs. In Section 3, the interpretation of convolutional layers is examined by visualizing intermediate activations of convolutional layers. Finally, in Section 4, conclusions are drawn.

2. Convolutional Neural Networks

In 2012, a special CNN architecture called AlexNet won the ImageNet object recognition challenge [5]. This study is accepted as a breakthrough in deep learning literature. On the other side, the underlying idea of CNNs dates back to Cognitron [18] and Neocognitron [19]. Also, if one looks further back into the history, the idea of the structure of convolutional layers is inspired from the discoveries about the mammalian vision system [20-22]. Neurophysiologists David Hubel and Torsten Wiesel showed that certain neurons in the mammalian visual cortex responded selectively to images and parts of images of specific shapes. In their experiments, they found that certain neurons fired rapidly when a cat was shown images small lines at one angle and that other neurons fired rapidly in response to small lines at another angle. Later work revealed that other neurons were specialized to respond to images containing more complex shapes such as corners, longer lines, and large edges [23].

[1] *Statistics, Mimar Sinan Fine Arts University, İstanbul – 34380, TURKEY*
ORCID ID : 0000-0002-3212-6711
** Corresponding Author Email: derya.soydaner@msgsu.edu.tr*

Similarly, CNNs learn features hierarchically from images [24]. They classify an image by combining simpler definitions such as corners and edges. Because it is difficult for a computer to understand the meaning of an image represented as a collection of pixel values. Deep learning resolves this difficulty by breaking the desired complicated mapping into a series of nested simple mappings, each described by a different layer of a model. In a CNN architecture, the image defined by pixels is in the input layer. Then a series of hidden layers extracts increasingly abstract features from the image [1]. This is the main idea of CNNs.

A typical CNN architecture is composed of convolutional and pooling layers followed by several fully-connected layers. The well-known LeNet-5 is shown in Fig. 1 as an example of the general CNN architecture. While convolutional layers extract feature maps, pooling layers reduce the spatial size of representations. Thus, the pooling layer shrinks the size of feature maps to improve statistical efficiency and reduces memory requirements for storing the parameters [1].

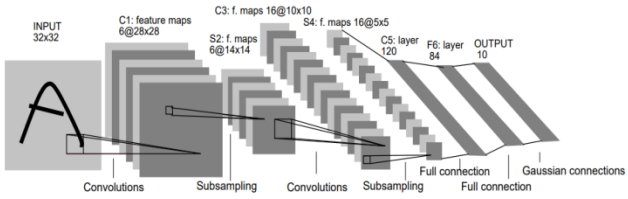

Fig. 1. LeNet-5 architecture [4]

In the convolutional layers the units are organized into planes, each of which is called a *feature map*. Units in a feature map each take inputs only from a small subregion of the image, and all of the units in a feature map are constrained to share the same weight values. If the units are considered as feature detectors, then all of the units in a feature map detect the same pattern but at different locations in the input image. Due to the weight sharing, the evaluation of the activations of these units is equivalent to a convolution of the image pixel intensities with a kernel comprising the weight parameters. As detecting multiple features is essential in order to build an effective model, there will generally be multiple feature maps in the convolutional layer, each having its own set of weight and bias parameters [25].

2.1. Convolution Operation

Convolution is a specialized kind of linear operation. CNNs are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers [1]. Convolving a 3x3 kernel over a 4x4 input using 1x1 strides is shown in Fig. 2. In this example, a 2x2 output is produced as a result of the convolution operation completed in four steps. Convolution is performed by multiplying the elements of the input corresponding to each element of the kernel. The first element of the 2x2 output is computed by taking summation of the results of this multiplication. The kernel starts on the leftmost part of the input feature map and slides by steps of one until it touches the right side of the input. Then, the kernel slides down, goes to the leftmost part and repeat the same process again. Thus, the kernel passes over the all input feature map [26]. For each convolutional layer, this operation is repeated as the number of filters. Thus, the forward pass is completed in CNNs.

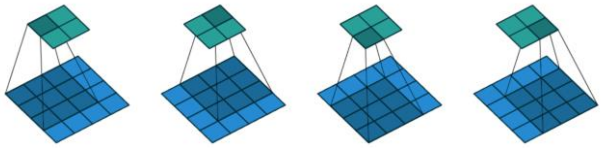

Fig. 2. A convolution operation example [27]

2.2. Convolution Operation on RGB Images

In CNNs, convolution operation can be performed in two different situations: An input image or a feature map produced by another convolutional layer may be convolved with a kernel. In both cases, the number of channels of the kernel and its input must match. Consider an RGB image as input. As RGB images have 3 channels, in order to convolve a kernel with an RGB image, the kernel must have 3 channels as well. Convolving a 2x2x3 kernel over a 3x3x3 input is shown in Fig. 3. The difference between this operation and the example given in previous subsection is that convolution is performed multiplying the elements of each kernel channel corresponding numbers from the red, green and blue channels of input. The elements of the 2x2 output are computed by taking summation of these multiplied results.

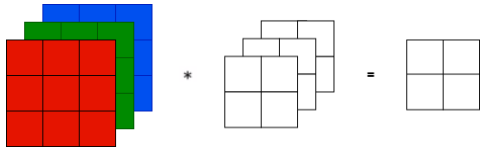

Fig. 3. A convolution operation example on an RGB image

In general, more than one filter is used in convolutional layers. In that case, a feature map is obtained by convolution operation for each filter. Then, these feature maps are concatenated to constitute the output. For example, in Fig. 3, if there are 5 filters instead of 1, the output will be 2x2x5. The number of filters determines the number of output channels.

2.3. Backpropagation in Convolutional Neural Networks

The backpropagation procedure [28] to compute the gradient of an objective function with respect to the weights of a multilayer stack of modules is nothing more than a practical application of the chain rule for derivatives. The key insight is that the gradient of the objective with respect to the input of a module can be computed by working backwards from the gradient with respect to the output of that module (or the input of the subsequent module) [3]. CNNs are some of the first working deep networks trained with backpropagation [1]. In CNNs, in addition to the forward pass, the backward pass is also performed by convolutions. When the forward pass is completed, the loss gradient from the previous layer must be calculated to move the loss backwards. The key insight is that this calculation is performed by convolution operation. In order to explain the backpropagation for convolutional layers, convolving a 2x2 kernel *(F)* over a 3x3 input *(X)* is shown in Fig. 4.

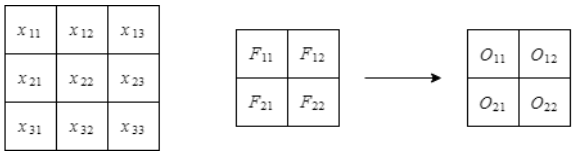

Fig. 4. Convolving a 2x2 kernel *(F)* over a 3x3 input *(X)*

In this example, a 2x2 output *(O)* is produced. The calculations of the elements of this output obtained by the forward pass are given in the following equations:

$$O_{11} = X_{11}F_{11} + X_{12}F_{12} + X_{21}F_{21} + X_{22}F_{22} \quad (1)$$

$$O_{12} = X_{12}F_{11} + X_{13}F_{12} + X_{22}F_{21} + X_{23}F_{22} \quad (2)$$

$$O_{21} = X_{21}F_{11} + X_{22}F_{12} + X_{31}F_{21} + X_{32}F_{22} \quad (3)$$

$$O_{22} = X_{22}F_{11} + X_{23}F_{12} + X_{32}F_{21} + X_{33}F_{22} \quad (4)$$

At the end of the forward pass, a loss value *(L)* is computed to backpropagate from the output back to the input by using a loss function. To perform the backward pass, $\partial L/\partial X$ and $\partial L/\partial F$ must be computed using the chain rule. Thus, the loss can be backpropagated to the other layers. $\partial L/\partial F$ is used to update filter *F*. On the other side, $\partial L/\partial X$ becomes the loss gradient for the previous layer when *X* is the output of the previous layer [29].

$\partial L/\partial F$ is computed in two steps. The first step is to find the local gradient, $\partial O/\partial F$. The calculations for the local gradients for O_{11} for the example in Fig. 4 are given in (5). As seen from the equations, the first element of the output is differentiated with respect to the elements of *F*. Similarly, the local gradients can be computed for all the output elements. After finding the gradients, for every element of *F*, a general rule to compute $\partial L/\partial F$ by using the chain rule is shown in (6).

$$\frac{\partial O_{11}}{\partial F_{11}} = X_{11},\ \frac{\partial O_{11}}{\partial F_{12}} = X_{12},\ \frac{\partial O_{11}}{\partial F_{21}} = X_{21},\ \frac{\partial O_{11}}{\partial F_{22}} = X_{22} \quad (5)$$

$$\frac{\partial L}{\partial F} = \frac{\partial L}{\partial O} * \frac{\partial O}{\partial F} \quad (6)$$

The expansion of (6) is given between equations (7) and (10). In these expansions, if the results obtained in (5) are substituted to $\partial O/\partial F$, it is seen that (6) is equivalent to a convolution operation between input *X* and the loss gradient $\partial L/\partial O$.

$$\frac{\partial L}{\partial F_{11}} = \frac{\partial L}{\partial O_{11}} * \frac{\partial O_{11}}{\partial F_{11}} + \frac{\partial L}{\partial O_{12}} * \frac{\partial O_{12}}{\partial F_{11}} + \frac{\partial L}{\partial O_{21}} * \frac{\partial O_{21}}{\partial F_{11}} + \frac{\partial L}{\partial O_{22}} * \frac{\partial O_{22}}{\partial F_{11}} \quad (7)$$

$$\frac{\partial L}{\partial F_{12}} = \frac{\partial L}{\partial O_{11}} * \frac{\partial O_{11}}{\partial F_{12}} + \frac{\partial L}{\partial O_{12}} * \frac{\partial O_{12}}{\partial F_{12}} + \frac{\partial L}{\partial O_{21}} * \frac{\partial O_{21}}{\partial F_{12}} + \frac{\partial L}{\partial O_{22}} * \frac{\partial O_{22}}{\partial F_{12}} \quad (8)$$

$$\frac{\partial L}{\partial F_{21}} = \frac{\partial L}{\partial O_{11}} * \frac{\partial O_{11}}{\partial F_{21}} + \frac{\partial L}{\partial O_{12}} * \frac{\partial O_{12}}{\partial F_{21}} + \frac{\partial L}{\partial O_{21}} * \frac{\partial O_{21}}{\partial F_{21}} + \frac{\partial L}{\partial O_{22}} * \frac{\partial O_{22}}{\partial F_{21}} \quad (9)$$

$$\frac{\partial L}{\partial F_{22}} = \frac{\partial L}{\partial O_{11}} * \frac{\partial O_{11}}{\partial F_{22}} + \frac{\partial L}{\partial O_{12}} * \frac{\partial O_{12}}{\partial F_{22}} + \frac{\partial L}{\partial O_{21}} * \frac{\partial O_{21}}{\partial F_{22}} + \frac{\partial L}{\partial O_{22}} * \frac{\partial O_{22}}{\partial F_{22}} \quad (10)$$

$\partial L/\partial X$ is also computed in two steps. The first step is to find the local gradient $\partial O/\partial X$. For the example in Fig. 4, the calculations for the local gradients for O_{11} are given in (11). Here, the first element of output is differentiated with respect to the elements of *X*. Similarly, the local gradients can be computed for the all output elements. After finding the gradients, for every element of *X*, a general rule to compute $\partial L/\partial X$ by using the chain rule is shown in (12).

$$\frac{\partial O_{11}}{\partial X_{11}} = F_{11},\ \frac{\partial O_{11}}{\partial X_{12}} = F_{12},\ \frac{\partial O_{11}}{\partial X_{21}} = F_{21},\ \frac{\partial O_{11}}{\partial X_{22}} = F_{22} \quad (11)$$

$$\frac{\partial L}{\partial X} = \frac{\partial L}{\partial O} * \frac{\partial O}{\partial X} \quad (12)$$

The expansion of (12) including the substitution of results from (11) is given as follows:

$$\frac{\partial L}{\partial X_{11}} = \frac{\partial L}{\partial O_{11}} * F_{11} \quad (13)$$

$$\frac{\partial L}{\partial X_{12}} = \frac{\partial L}{\partial O_{11}} * F_{12} + \frac{\partial L}{\partial O_{12}} * F_{11} \quad (14)$$

$$\frac{\partial L}{\partial X_{13}} = \frac{\partial L}{\partial O_{12}} * F_{12} \quad (15)$$

$$\frac{\partial L}{\partial X_{21}} = \frac{\partial L}{\partial O_{11}} * F_{21} + \frac{\partial L}{\partial O_{21}} * F_{11} \quad (16)$$

$$\frac{\partial L}{\partial X_{22}} = \frac{\partial L}{\partial O_{11}} * F_{22} + \frac{\partial L}{\partial O_{12}} * F_{21} + \frac{\partial L}{\partial O_{21}} * F_{12} + \frac{\partial L}{\partial O_{22}} * F_{11} \quad (17)$$

$$\frac{\partial L}{\partial X_{23}} = \frac{\partial L}{\partial O_{12}} * F_{22} + \frac{\partial L}{\partial O_{22}} * F_{12} \quad (18)$$

$$\frac{\partial L}{\partial X_{31}} = \frac{\partial L}{\partial O_{21}} * F_{21} \quad (19)$$

$$\frac{\partial L}{\partial X_{32}} = \frac{\partial L}{\partial O_{21}} * F_{22} + \frac{\partial L}{\partial O_{22}} * F_{21} \quad (20)$$

$$\frac{\partial L}{\partial X_{33}} = \frac{\partial L}{\partial O_{22}} * F_{22} \quad (21)$$

The calculations between equations (13) and (21) are equivalent to a full convolution operation between the loss gradient $\partial L/\partial O$ and a 180-degree rotated filter [29]. After these calculations, the kernel is updated as shown in (22). Here, the learning rate is represented as α.

$$F_{new} = F_{old} - \alpha \frac{\partial L}{\partial F} \quad (22)$$

Convolutional layer is the core building block of the CNN architecture. Similar to this layer, the gradients of the pooling layer can be calculated by following the similar procedure of using chain rule.

3. Experimental Analysis

3.1. Dataset

One of the foremost success of CNNs is face recognition [30, 31]. Therefore, the well-known Labeled Faces in the Wild (LFW) dataset [32] is chosen to perform the visualizing experiments. LFW is composed of 13.233 face images belonging to 5749 people. They are RGB images in different sizes. In this study, these images are scaled to a size of 64x64. Also, LFW classes that have at least 30 images are chosen for training. Hence, 1777 face images belonging to 34 people are used for classification task. They are randomly divided into two subsets as 0.75 for training and 0.25 for testing.

3.2. Implementation Details

To understand CNNs better, the intermediate activations of a CNN architecture is visualized. Thus, the convolutional layers are interpreted visually in addition to the exposition of backpropagation algorithm for convolutional layers. The CNN architecture is designed for the visualizing experiments as follows: It includes three convolutional layers each one is followed by a 2x2 max-pooling layer. The convolutional layers include 32, 64 and 128 filters with 3x3 kernels, respectively. These layers are followed by two fully-connected layers including 128 and 256 hidden units. Lastly, the output layer is fully-connected including 34 hidden units with softmax activation function. The rest of the layers use ReLU. Dropout in ratio 0.50 is applied to the fully-connected layers preceding the output layer. This CNN architecture is trained using Adam [33] which is one of the widely used optimization algorithm in deep learning. The learning rate is 0.001. This network is trained 100 epochs with a minibatch size of 128. The loss function is categorical cross entropy.

3.3. Visualizing Intermediate Activations

The intermediate activations are visualized to get an idea about how CNNs see and interpret the world. To this end, firstly, the CNN architecture described in the previous subsection is trained 100 epochs. The course of training process is shown in Fig. 5. At the end of training, it achieves a classification accuracy of 83.47 on the test data.

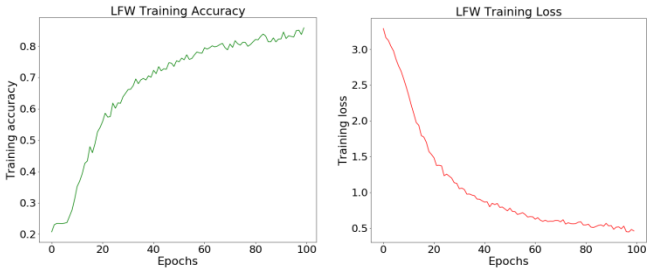

Fig. 5. *(Left)* Training accuracy. *(Right)* Training loss.

Then, the activations of all convolutional and pooling layers are examined using an example from the test data which is the network is not trained on. The picture below is the input image to examine the intermediate activations:

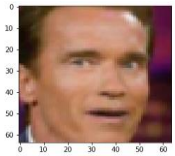

Fig. 6. The test image from LFW

As the representations learned by the CNN are simply representations of visual concepts, this experiment can provide better understanding about how successive layers transform their input. Additionally, this gives a view into how an input is decomposed into the different filters learned by the network [10]. The CNN architecture includes three convolutional layers, each one is followed by a max-pooling layer. Therefore, every channel in each of activation maps obtained from these six layers is visualized as a 2D image. The visualizations of feature maps are stacked side by side for each convolutional and pooling layer and they are shown below, respectively:

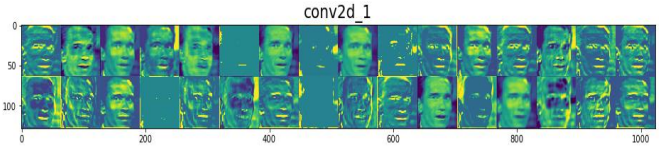

Fig. 7. The feature maps extracted from the first convolutional layer

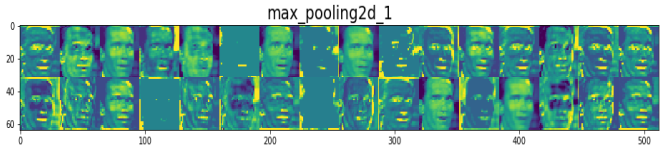

Fig. 8. The feature maps extracted from the first max-pooling layer

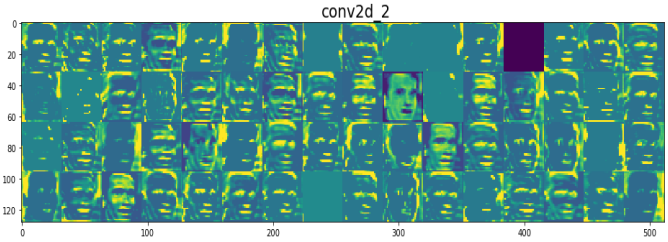

Fig. 9. The feature maps extracted from the second convolutional layer

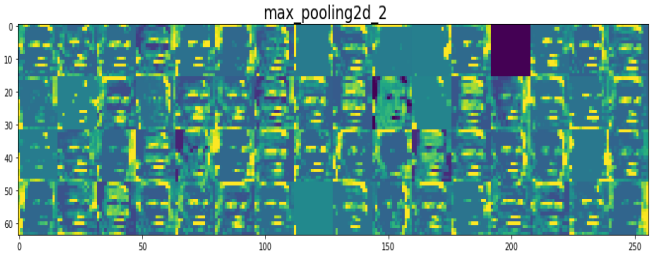

Fig. 10. The feature maps extracted from the second max-pooling layer

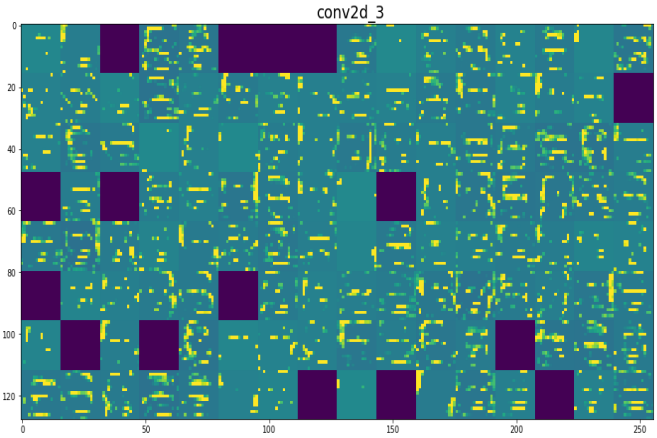

Fig. 11. The feature maps extracted from the third convolutional layer

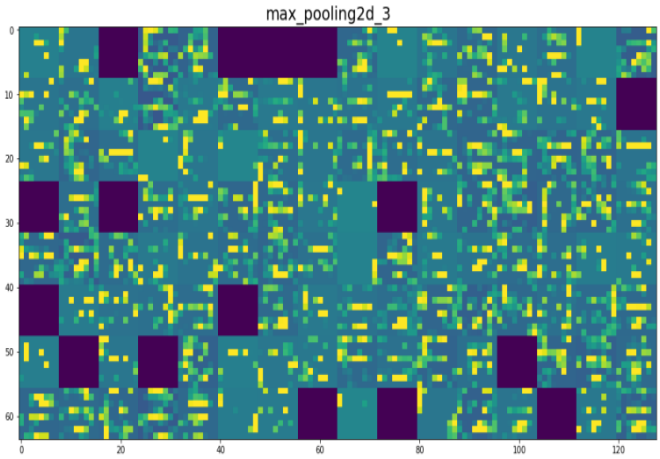

Fig. 12. The feature maps extracted from the third max-pooling layer

As seen from the visualizations, the neural network extracts increasingly abstract features from the test image. In the deeper layers, it becomes difficult to interpret the activations in visual. Additionally, the sparsity of the activations increases. In the first layer, all filters are activated by the input image. However, the number of blank activations increases in the following layers. This means that the pattern encoded by the filter is not found in the input image [10]. In the first convolutional layer, the activations keep almost all of the information in the test image. As the activations go through deeper layers, they start encoding higher-level features such as “mouth” or “eyes”. At the end, it classifies the image by combining simpler definitions. It can be seen clearly that the CNN learns features hierarchically from the face image.

4. Conclusion

In this work, the calculations behind convolutional layers are explored. Thus, the backpropagation for CNNs is clarified. Besides, it is investigated how CNNs learn and perceive the

world. Examining these issues is as important as building and training new CNN architectures on various tasks. Because deep CNNs have achieved impressive results and it is obvious that they will have many more in the near future. To this end, understanding and interpreting the learning process and internal representations can provide an insight into how the model works in addition to give hints to develop new architectures and algorithms.

References

[1] I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*. The MIT Press, 2016.

[2] A. Geron, *Hands-on Machine Learning with Scikit-Learn and Tensorflow: concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, Inc., 2017.

[3] Y. LeCun, Y. Bengio and G. Hinton, "Deep learning," *Nature*, vol. 512, no. 7553, 2015.

[4] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324, 1998.

[5] A. Krizhevsky, I. Sutskever and G. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in Neural Information Processing Systems*, pp. 1097-1105, 2012.

[6] G. Huang, Z. Liu, L. Maaten and K. Weinberger, "Densely connected convolutional networks," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4700-4708, 2017.

[7] K. He, X. Zhang, S. Ren and J. Sun, "Deep residual learning for image recognition," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770-778, 2016.

[8] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[9] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, "Going deeper with convolutions," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp.1-9, 2015.

[10] F. Chollet, *Deep Learning with Python*. Manning Publications Co., 2018.

[11] Z. Qin, F. Yu, C. Lıu and X. Chen, "How convolutional neural network see the world – A survey of convolutional neural network visualization methods," *Mathematical Foundations of Computing*, vol. 1, no. 2, 2018.

[12] M.D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," *European Conference on Computer Vision*, Springer, Cham, pp. 818-833, 2014.

[13] A. Mahendran and A. Vedaldi, "Visualizing deep convolutional neural networks using natural pre-images," *International Journal of Computer Vision*, vol.120, no. 3, pp. 233-255, 2016.

[14] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs and H. Lipson, "Understanding neural networks through deep visualization," *arXiv preprint arXiv:1506.06579*, 2015.

[15] Q. Zhang, Y.N. Wu and S. Zhu, "Interpretable convolutional neural networks," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8827-8836, 2018.

[16] W. Samek, A. Binder, G. Montavon, S. Lapuschkin and K-R. Müller, "Evaluating the visualization of what a deep neural network has learned," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no.11, pp. 2660-2673, 2016.

[17] A. Nguyen, J. Yosinski and J. Clune, "Understanding neural networks via feature visualization: A survey," *arXiv preprint arXiv:1904.08939*, 2019.

[18] K. Fukushima, "Cognitron: A self-organizing multilayered neural network," *Biological Cybernetics*, vol. 20, pp. 121-136, 1975.

[19] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, no. 4, pp. 193-202, 1980.

[20] D. Hubel and T. Wiesel, "Receptive fields of single neurons in the cat's striate cortex," *The Journal of Physiology*, vol. 148, no. 3, pp. 574-591, 1959.

[21] D. Hubel and T. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *The Journal of Physiology*, vol. 160, no. 1, pp. 106-154, 1962.

[22] D. Hubel and T. Wiesel, "Receptive fields and functional architecture of monkey striate cortex," *The Journal of Physiology*, vol. 195, no. 1, pp. 215-243, 1968.

[23] N. Nilsson, *The Quest for Artificial Intelligence*. Cambridge University Press, 2009.

[24] G. Hinton, S. Osindero and Y. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation*, vol. 18, no. 7, pp. 1527-1554, 2006.

[25] C.M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006.

[26] D. Soydaner, "Training deep neural network based hyper autoencoders with machine learning methods," Ph.D. dissertation, Dept. Statistics, Mimar Sinan Fine Arts University, İstanbul, Turkey, 2018.

[27] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," *arXiv preprint arXiv:1603.07285*, 2016.

[28] D. Rumelhart, G. Hinton and R. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533-536, 1986.

[29] P. Solai, "Convolutions and backpropagations," Available: https://medium.com/@pavisj/convolutions-and-backpropagations-46026a8f5d2c, 2018.

[30] Y. Taigman, M. Yang, M. Ranzato and L. Wolf, "DeepFace: Closing the gap to human-level performance in face verification," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1701-1708, 2014.

[31] C. Garcia and M. Delakis, "Convolutional face finder: A neural architecture for fast and robust face detection," *IEEE Transactions Pattern Analysis and Machine Intelligence*, vol. 26, no. 11, pp. 1408-1423, 2004.

[32] G. Huang, M. Ramesh, T. Berg and E. Learned-Miller, "Labeled faces in the wild: A database for studying face recognition in unconstrained environments," *University of Massachusetts, Amherst, Technical Report*, pp. 07-49, 2007.

[33] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.