Understanding Convolution Neural Network

M.Eng Information Technology
Winter Semester 2019/20
Computational Intelligence

Seminar Paper :

Convolutional Neural Network

Yash Vyas 1266490
vyas@stud.fra-uas.de

Guide: Prof Dr. Andreas Pech

Abstract

This seminar paper focusses on convolutional neural networks and a visualization technique allowing
Further insights into their internal operation. After giving a brief introduction to neural,
We discuss convolutional neural networks in detail.
In addition, we discuss several approaches to regularization.
The second section introduces the different types of layers present in recent convolutional neural networks.
Based on these basic building blocks, we discuss the architecture of the traditional convolutional
Neural network as proposed in traditional papers. As well as the architecture of recent implementations.
The third section focusses on a technique to visualize feature activations of higher layers by back projecting
Them to the image plane. This allows to get deeper insights into the internal working of convolutional
Neural networks such that recent architectures can be evaluated and improved even further.

CONTENT

I. INTRODUCTION

A neural network is a system of interconnected artificial "neurons" that exchange messages between each other. The connections have numeric weights that are tuned during the training process, so that a properly trained network will respond correctly when presented with an image or pattern to recognize. The network consists of multiple layers of feature-detecting "neurons". Each layer has many neurons that respond to different combinations of inputs from the previous layers. As shown in [Figure 1.0], the layers are built up so that the first layer detects a set of primitive patterns in the input, the second layer detects patterns of patterns, the third layer detects patterns of those patterns, and so on. Typical CNNs use 5 to 25 distinct layers of pattern recognition. Training is performed using a "labelled" dataset of inputs in a wide assortment of representative input patterns that are tagged with their intended output response. Training uses general-purpose methods to iteratively determine the weights for intermediate and final feature neurons.

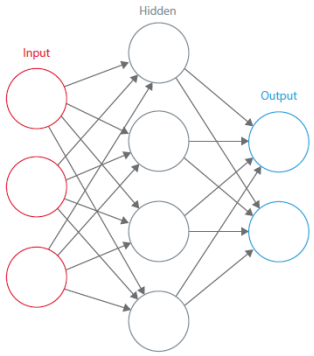

Figure 1.0: Artificial neural network (1)

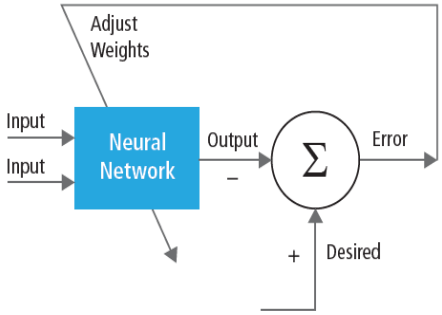

Figure 1.1: Training of neural network

[Figure 1.1] demonstrates the training process at a block level. Neural networks are inspired by biological neural systems. The basic computational unit of the brain is a neuron and they are connected with synapses. [Figure 1.2] compares a biological neuron with a basic mathematical model [Fig 1.3] [2].In a real animal neural system, a neuron is perceived to be receiving input signals from its dendrites and producing output signals along its axon. The axon branches out and connects via synapses to dendrites of other neurons. When the combination of input signals reaches some threshold condition among its input dendrites, the neuron is triggered and its activation is communicated to successor neurons. In the neural network computational model, the signals that travel along the axons (e.g., x_0) interact multiplicatively (e.g., w_0x_0) with the dendrites of the other neuron based on the synaptic strength at that synapse (e.g., w_0). Synaptic weights are learnable and control the influence of one neuron or another. The dendrites carry the signal to the cell body, where they all are summed. If the final sum is above a specified threshold, the neuron fires, sending a spike along its axon. In the computational model, it is assumed that the precise timings of the firing do not matter and only the frequency of the firing communicates information. Based on the rate code interpretation, the firing rate of the neuron is modelled with an activation function ***f*** that represents the frequency of the spikes along the axon. A common choice of activation function is sigmoid. In summary, each neuron calculates the dot product of inputs and weights, adds the bias, and applies non-linearity as a trigger function (for example, following a sigmoid response function). A CNN is a special case of the neural network described above.

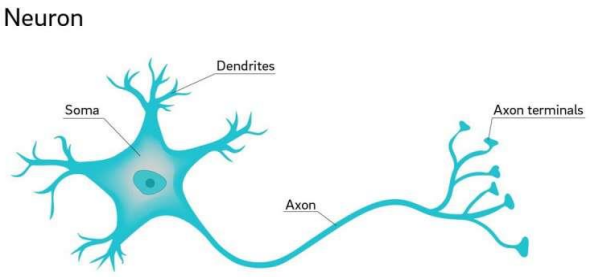

Figure 1.2: Biological neural system

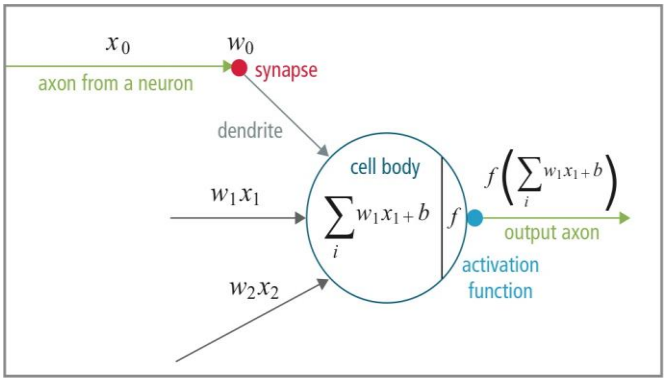

Figure 1.3: Biological neural system

II. WHAT IS CONVOLUTIONAL NEURAL NETWORK

A convolutional neural network (CNN or ConvNet) is one of the most popular algorithms for deep learning , a variant of machine learning in which a model learns to perform classification tasks directly from images, video data, texts or acoustic data. CNNs are especially useful for finding patterns in pictures and thus recognizing objects, faces and scenes. CNNs learn directly from image data. They use patterns to classify images and make manual extraction of features unnecessary. Applications that require object recognition and computer vision - such as self-driving vehicles and face recognition applications - make intensive use of CNNs. Depending on your application, you can create a CNN from scratch or use a pre-trained model with your data set. [Fig 2.0] shows the typical block diagram of CNN. The network consists of one or more convolutional layers, often with a subsampling layer, which are followed by one or more fully connected layers as in a standard neural network. The design of a CNN is motivated by the discovery of a visual mechanism, the visual cortex, in the brain. The visual cortex contains a lot of cells that are responsible for detecting light in small, overlapping sub-regions of the visual field, which are called receptive fields. These cells act as local filters over the input space, and the more complex cells have larger receptive fields. The convolution layer in a CNN performs the function that is performed by the cells in the visual cortex. A typical CNN for recognizing digits is shown in [Figure 2.0]. Each feature of a layer receives inputs from a set of features located in a small neighbourhood in the previous layer called a local receptive field. With local receptive fields, features can extract elementary visual features, such as oriented edges, end-points, corners, etc., which are then combined by the higher layers. In the traditional model of pattern/image recognition, a hand-designed feature extractor gathers relevant information from the input and eliminates irrelevant variabilities. The extractor is followed by a trainable classifier, a standard neural network that classifies feature vectors into classes. In a CNN, convolution layers play the role of feature extractor. But they are not hand designed.

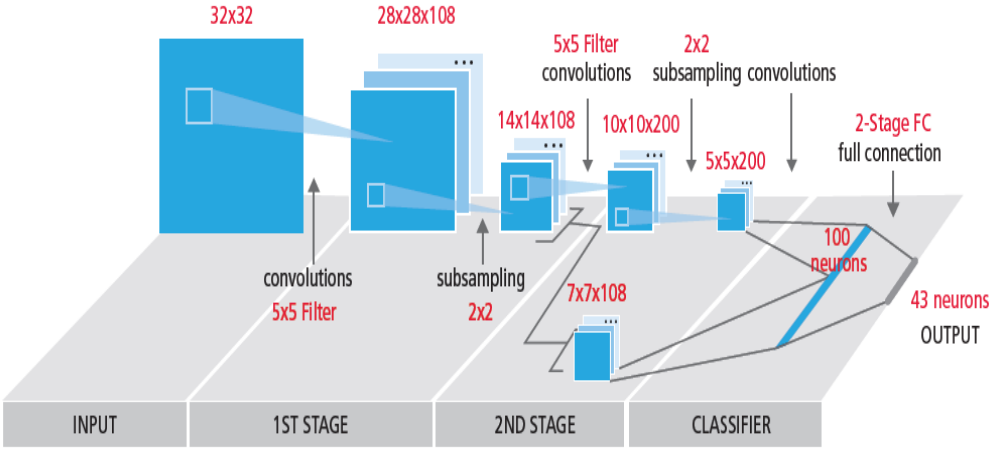

Figure 2.0: Typical Block diagram of Cnn

Convolution filter kernel weights are decided on as part of the training process. Convolutional layers are able to extract the local features because they restrict the receptive fields of the hidden layers to be local. Although neural networks can be applied to computer vision tasks, to get good generalization performance, it is beneficial to incorporate prior knowledge into the network architecture [3]. Convolutional neural networks aim to use spatial information between the pixels of an image. Therefore, they are based on discrete convolution. After introducing discrete convolution, we discuss the basic components of convolutional neural networks as described in [4] and [5].

2.1) ***Convolution***

For simplicity we assume a grayscale image to be defined by a function

$$I: \{1, \dots, n_1\} \times \{1, \dots, n_2\} \to \mathrm{W} \subseteq \mathbb{R}, (i, j) \vdash\succ I_{i,j}$$

such that the image I can be represented by an array of size $n_1 \times n_2$. Given the filter $\boldsymbol{K} \in \mathbb{R}^{2h_1+1 \times 2h_2+1}$, the discrete convolution of the image I with filter K is given by

$$(I * K)_{r,s} := \sum_{u=-h_1}^{h_1} \sum_{v=-h_2}^{h_2} \mathrm{K}_{u,v} I_{r+u,s+v}$$

where the filter $\boldsymbol{K}$ is given by

$$K = \begin{matrix} K_{-h_1,-h_2} & \dots & K_{-h1,-h_2} \\ \vdots & K_{0,0} & \vdots \\ K_{h_1,h_2} & \dots & K_{h_1,h_2} \end{matrix}$$

Note that the behaviour of this operation towards the borders of the image needs to be defined properly. A commonly used filter for smoothing is the discrete Gaussian filter $\mathrm{K}_{G(s)}$ [FP02] which is defined by

$$\left(K_{G(\sigma)}\right)_{r,s} = \frac{1}{\sqrt{2\pi\sigma^2}} exp\left(\frac{r^2 + s^2}{2\sigma^2}\right)$$

where s is the standard deviation of the Gaussian distribution [6].

2.2)Layers of Convolutional Neural Network

By stacking multiple and different layers in a CNN, complex architectures are built for classification problems. Four types of layers are most common: convolution layers, pooling/subsampling layers, non-linear layers, and fully-Connected layers.

a) Convolutional Layers

The convolution operation extracts different features of the input. The first convolution layer extracts low-level features like edges, lines, and corners. Higher-level layers extract higher-level features. [Figure 2.1] illustrates the process of 3D convolution used in CNNs. The input is of size $N \; x \; N \; x \; D$ and is convolved with H kernels, each of size $k \; x \; k \; x \; D$ separately. Convolution of an input with one kernel produces one output feature, and with H kernels independently produces H features. Starting from top-left corner of the input, each kernel is moved from left to right, one element at a time. Once the top-right corner is reached the kernel is moved one element in a downward direction, and again the kernel is moved from left to right, one element at a time. This process is repeated until the kernel reaches the bottom-right corner. For the case when $N = 32$ and $k = 5$, there are 28 unique positions from left to right and 28 unique positions from top to bottom that the kernel can take. Corresponding to these positions, each feature in the output will contain $28x28$ (i.e., (N-k+1) x (N-k+1)) elements. For each position of the kernel in a sliding window process, $k \; x \; k \; x \; D$ elements of input *and* $k \; x \; k \; x \; D$ elements of kernel are element-byelement multiplied and accumulated. So to create one element of one output feature, $k \; x \; k \; x \; D$ multiply-accumulate operations are required.

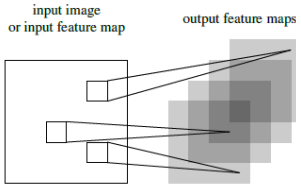

Figure 2.1. Illustration of a single convolutional layer. If layer l is a convolutional layer, the input image (if l = 1) or a feature map of the previous layer is convolved by different filters to yield the output feature maps of layer l.

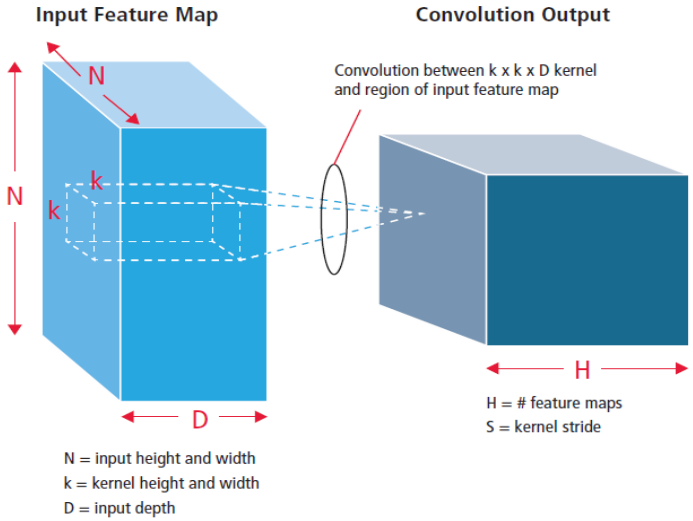

Figure 2.2 Pictorial representation of convolution process

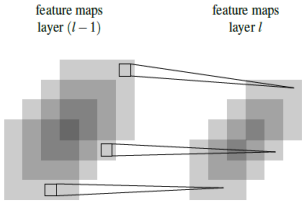

Figure 2.3 Illustration of pooling and subsampling layer

b) Pooling and SubSampling Layers

The pooling/subsampling layer reduces the resolution of the features. It makes the features robust against noise and distortion. There are two ways to do pooling: max pooling and average pooling. In both cases, the input is divided into non-overlapping two-dimensional spaces. Each input feature is $28x28$ and is divided into $14x14$ regions of size $2x2$. For average pooling, the average of the four values in the region are calculated. For max pooling, the maximum value of the four values is selected. [Figure 2.3] elaborates the pooling process further. The input is of size $4x4$. For $2x2$ subsampling, a $4x4$ image is divided into four non-overlapping matrices of size $2x2$. In the case of max pooling, the maximum value of the four values in the $2x2$ matrix is the output. In case of average pooling, the average of the four values is the output. Please note that for the output with index (2,2), the result of averaging is a fraction that has been rounded to nearest integer.

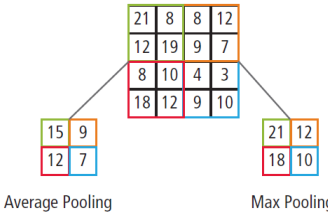

Figure 2.4 Pictorial representation of Max pooling and Average Pooling

c) Non-Linear Layers

Neural networks in general and CNNs in particular rely on a non-linear "trigger" function to signal distinct identification of likely features on each hidden layer. CNNs may use a variety of specific functions —such as rectified linear units (ReLUs) and continuous trigger (non-linear) functions—to efficiently implement this non-linear triggering.

c.1) RELU

A ReLU implements the function y = max(x,0), so the input and output sizes of this layer are the same. It increases the nonlinear properties of the decision function and of the overall network without affecting the receptive fields of the convolution layer. In comparison to the other non-linear functions used in CNNs (e.g., hyperbolic tangent, absolute of hyperbolic tangent, and sigmoid), the advantage of a ReLU is that the network trains many times faster. ReLU functionality is illustrated in Figure 2.4, with its transfer function plotted above the arrow.

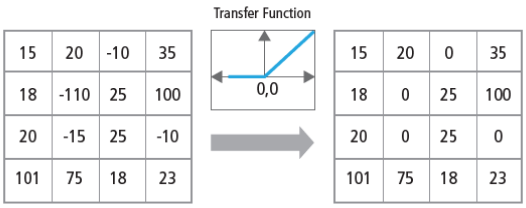

Figure 2.4 ReLU functionality

c.2)Continuous Trigger (non-liner) function

The non-linear layer operates element by element in each feature. A continuous trigger function can be hyperbolic tangent, absolute of hyperbolic tangent, or sigmoid (Figure 2.5 a, b, c, d). (Figure 2.5 - d) demonstrates how non-linearity gets applied element by element.

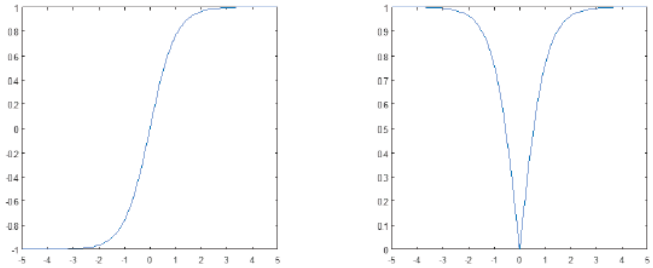

Figure 2.5: (a) Hyperbolic tangent function (b) Absolute hyperbolic tangent function

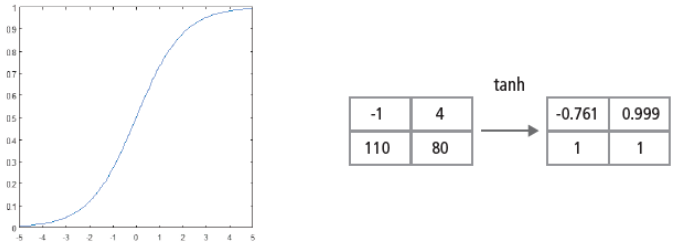

Figure 1.5: (c) Sigmoid function *(d) tanh processing*

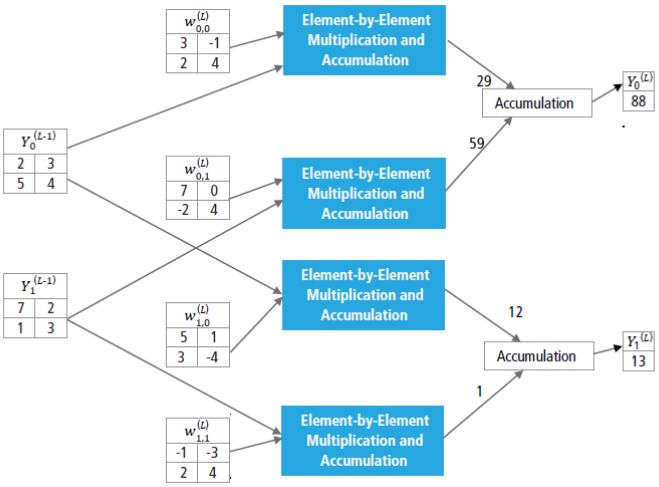

Figure 2.6: Processing of fully connected layer

d) Fully Connected Layers

Fully connected layers are often used as the final layers of a CNN. These layers mathematically sum a weighting of the previous layer of features, indicating the precise mix of "ingredients" to determine a specific target output result. In case of a fully connected layer, all the elements of all the features of the previous layer get used in the calculation of each element of each output feature. Figure 13 explains the fully connected layer ℓ. Layer ℓ -1 has two features, each of which is 2x2, i.e., has four elements. Layer ℓ has two features, each having a single element. Let layer ℓ be a fully connected layer. If layer (ℓ - 1) is a fully connected layer, as well, we may apply equation of multilayer perceptron. Otherwise, layer ℓ m_1^{l-1}feature maps of size $m_2^{l-1} \times m_3^{l-1}$ as input and the ι^{th} unit In layer ℓ computes:

$$y_i^l = f(z_i^l) \qquad with \qquad z_i^l = \sum_{j=1}^{m_1^{l-1}} \sum_{j=1}^{m_2^{l-1}} \sum_{s=1}^{m_3^{l-1}} w_{i,j,r,s}^l \, (Y_j^{l-1})_{r,s}$$

where $w_{i,j,r,s}^l$ denotes the weight connecting the unit at position (r, s) in the j^{th} feature map of layer $l-1$ and the i^{th} unit in layer$l$. In practice, convolutional layers are used to learn a feature hierarchy and one or more fully connected layers are used for classification purposes based on the computed features (6).Note that a fully-connected layer already includes the non-linearities while for a convolutional layer the non-linearities are separated in their own layer.

*2.*4) Architecture

We discuss both the traditional convolutional neural network as proposed in [6] as well as a modern variant as used in [7].

a)Traditional Convolutional Neural Network

In [8] the basic building block of traditional neural networks is $F_{CSG} - P_A$ while in [6] , the subsampling is accomplished within the convolutional layers and there are no gain coefficients used. In [Figure 2.7]: The architecture of the original convolutional neural network, as introduced in [6], alternates between convolutional layers including hyperbolic tangent non-linarites and subsampling layers. In this illustration, the convolutional layers already include non-linearities and, thus, a convolutional layer actually represents two layers. The feature maps of the final subsampling layer are then fed into the actual classifier consisting

of an arbitrary number of fully connected layers. The output layer usually uses softmax activation functions. General, the unique characteristic of traditional convolutional neural networks lies in the hyperbolic tangent non-linarites and the weight sharing .This is illustrated in [Fig 2.7] where the non-linearities are included within the convolutional layers.

b)Modern Convolutional Neural Network

As example of a modern convolutional neural network we explore the architecture used [9] in which gives excellent performance on the ImageNet Dataset [10] .The architecture comprises five convolutional layers each followed by a rectified linear unit non-linearity layer, brightness normalization and overlapping pooling. Classification is done using three additional fully-connected layers. To avoid overfitting [9] uses dropout as regularization technique. Such a network can be specified by $F_{CR} - N_B - P$ where F_{CR} denotes a convolutional layer followed by a non-linearity layer with rectified linear units. Details can be found in [9]. In [11] the authors combine several deep convolutional neural networks which have a similar architecture as described above and average their classification/prediction result. This architecture is referred to as
Multi-column deep convolutional neural network.

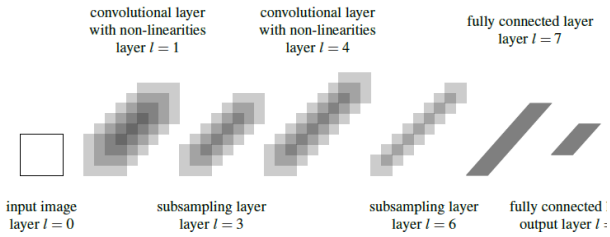

Figure 2.7: Architecture of CNN

III. UNDERSTANDING CONVOLUTIONAL NEURAL NETWORK

Although convolutional neural networks have been used with success for a variety of computer vision tasks, their internal operation is not well understood. While back projection of feature activations from the first convolutional layer is possible, subsequent pooling and rectification layers hinder us from understanding higher layers as well. As stated in [12], this is highly unsatisfactory when aiming to improve convolutional neural networks. Thus, in [12], a visualization technique is proposed which allows us to visualize the activations from higher layers. This technique is based on an additional model for unsupervised learning of feature hierarchies: the deconvolutional neural network as introduced in [13] .

3.1) Deconvolutional Neural Networks

Similar to convolutional neural networks, deconvolutional neural networks are based upon the idea of generating feature hierarchies by convolving the input image by a set of filters at each layer [13]. However, deconvolutional neural networks are unsupervised by definition. In addition, deconvolutional neural networks are based on a top-down approach. This means, the goal is to reconstruct the network input from its activations and filters [13].

a) Deconvolutional Layer

Let layer l be a deconvolutional layer. The input is composed of m_1^{l-1} feature maps of size $m_2^{l-1} \times m_3^{l-1}$. Each such feature map Y_i^{l-1} is represented as sum over m_1^l feature maps convolved with filters $\boldsymbol{K}_{j,i}^{l}$:

$$\sum_{j=1}^{m_1^l} K_{j,i}^l * Y_j^l = Y_i^{l-1}$$

As with an auto-encoder, it is easy for the layer to learn the identity, if there are enough degrees of freedom. Therefore, (13) introduces a sparsity constraint for the feature maps Y_j^l , and the error measure for training layer l is given by

$$E^l(w) = \sum_{i=1}^{m_1^{l-1}} \| \sum_{i=1}^{m_1^l} K_{j,i}^l * Y_j^l - Y_i^{l-1} \|_2^2 + \sum_{i=1}^{m_1^l} \| Y_i^l \|_p^p$$

where $\|\cdot\|_p$ is the vectorised p-norm and can be interpreted as L_p-regularization. The difference between a convolutional layer and a deconvolutional layer is illustrated in figure 9. Note that the error measure E_l is specific for layer l . This implies that a deconvolutional neural network with multiple deconvolutional layers is trained layer-wise.

b) Unsurpervised Training

Similar to unsupervised training discussed in section 2.4, training is performed layer-wise. Therefore, above equation is optimized by alternately optimizing with respect to the feature maps Y_i^l given the filters $K_{j,i}^l$ and the feature maps Y_i^{l-1} of the previous layer and with respect to the filters $K_{j,i}^l$ (13). Here, the optimization with respect to the feature maps Y_i^l causes some problems. For example when using $p = 1$, the optimization problem is poorly conditioned [13] and therefore usual gradient descent optimization fails. An alternative optimization scheme is discussed in detail in [13], however, as we do not need to train deconvolutional neural networks, this is left to the reader.

3.2) Visualizing Convolutional Neural Networks

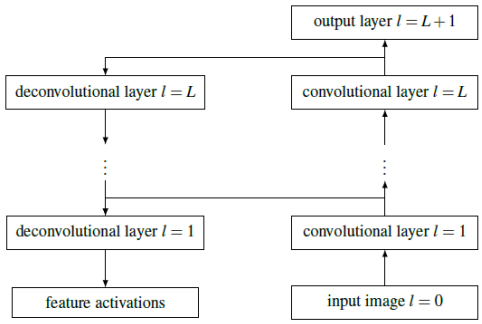

Figure 3.0: After each convolutional layer, the feature activations of the previous layer are reconstructed using an attached deconvolutional layer. For l > 1 the process of reconstruction is iterated until the feature

To visualize and understand the internal operations of a convolutional neural network, a single deconvolutional layer is attached to each convolutional layer. Given input feature maps for layer l, the output feature maps Y_i^l are fed back into the corresponding deconvolutional layer at level l. The deconvolutional layer reconstructs the feature maps Y_i^{l-1} that gave rise to the activations in layer l [12]. This process is iterated until layer $l = 0$ is reached resulting in the activations of layer l being back projected onto the image plane. The general idea is illustrated in [Fig 3.0]. Note that the deconvolutional layers do not need to be trained as the filters are already given by the trained convolutional layers and merely have to be transposed. More complex convolutional neural networks may include non-linearity layers, rectification layers as well as pooling layers. While we assume the used non-linarites to be invertible, the use of rectification layers and pooling layers cause some problems.

a) Pooling Layers

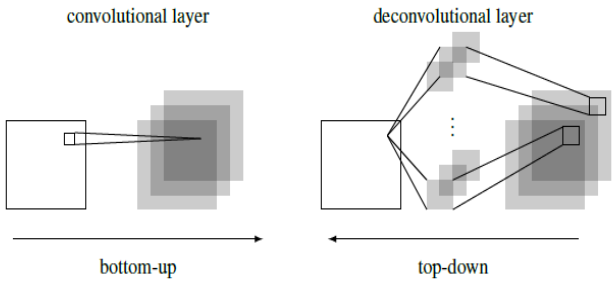

Figure 3.0: An illustration of the difference between the bottom-up approach of convolutional layers and the

Let layer l be a max pooling layer, then the operation of layer l is not invertible. We need to remember which positions within the input feature map Y_i^l gave rise to the maximum value to get an approximate inverse [12]. Therefore, as discussed in [12], switch variables are introduced.

b) Rectification Layers

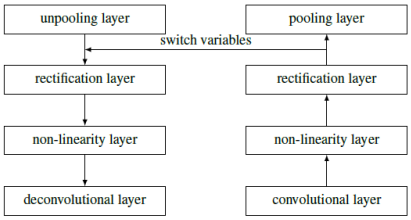

Figure 3.2 While the approach described in section 4.2 can easily be applied to convolutional neural networks including non-linearity layers, the usage of pooling and rectification layers imposes some problems. The max pooling operation is not invertible. Therefore, for each unit in the pooling layer, we remember the position in the corresponding feature map which gave rise to the unit's output value. To accomplish this, so called switch variables are introduced [12]. Rectification layers can simply be inverted by prepending a rectification layer to the deconvolutional layer.

The convolutional layer may use rectification layers to obtain positive feature maps after each non-linearity layer. To cope with this, a rectification layer is added to each deconvolutional layer to obtain positive reconstructions of the feature maps, as well [12]. Both the incorporation of pooling layers and rectification layers is illustrated in [Fig 3.2].

3.3) Convolutional Neural Network Visualization

The above visualization technique can be used to discuss several aspects of convolutional neural networks. We follow the discussion in [12] which refers to the architecture described in section 2.4.b.

a) Filters and Features

Back projecting the feature activations allows close analysis of the hierarchical nature of the features within the convolutional neural network. Studies in [12], shows the activations for three layers with corresponding input images. While the first and second layer comprise filters for edge and corner detection, the filters tend to get more complex and abstract with higher layers. For example when considering layer 3, the feature activations reflect specific structures within the images: the patterns used in layer 3, row 1, column 1; human contours in layer 3 row3, column 3. Higher levels show strong invariances to translation and rotation [12]. Such transformations usually have high impact on low-level features. In addition, as stated in [12], it is important to train the convolutional neural network until convergence as the higher levels usually need more time to converge.

b) Architecture Evaluation

The visualization of the feature activations across the convolutional layers allows to evaluate the effect of filter size as well as filter placement. For example, by analysing the feature activations of the first and second layer, the authors of [12] observed that the first layer does only capture high frequency and low frequency information and the feature activations of the second layer show aliasing artifacts. By adapting the filter size of the first layer and the skipping factor used within the second layer, performance could be improved. In addition, the visualization shows the advantage of deep architectures as higher layers are able to learn more complex features invariant to low-level distortions and translations [12].

IV. WHY CNNS?

While neural networks and other pattern detection methods have been around for the past 50 years, there has been significant development in the area of convolutional neural networks in the recent past. This section covers the advantages of using CNN for image recognition.

4.1) Ruggedness to shifts and distortion in the image

Detection using CNN is rugged to distortions such as change in shape due to camera lens, different lighting conditions, different poses, presence of partial occlusions, horizontal and vertical shifts, etc. However, CNNs are shift invariant since the same weight configuration is used across space. In theory, we also can achieve shift invariantness using fully connected layers. But the outcome of training in this case is multiple units with identical weight patterns at different locations of the input. To learn these weight configurations, a large number of training instances would be required to cover the space of possible variations.

4.2) Fewer memory requirements

In this same hypothetical case where we use a fully connected layer to extract the features, the input image of size 32x32 and a hidden layer having 1000 features will require an order of 10^6 coefficients, a huge memory requirement. In the convolutional layer, the same coefficients are used across different locations in the space, so the memory requirement is drastically reduced.

4.3) Easier and Better Training

Again using the standard neural network that would be equivalent to a CNN, because the number of parameters would be much higher, the training time would also increase proportionately. In a CNN, since the number of parameters is drastically reduced, training time is proportionately reduced. Also, assuming perfect training, we can design a standard neural network whose performance would be same as a CNN. But in practical training, a standard neural network equivalent to CNN would have more parameters, which would lead to more noise addition during the training process. Hence, the performance of a standard neural network equivalent to a CNN will always be poorer.

V. FUTURE OF CNNS

Among the promising areas of neural networks research are recurrent neural networks (RNNs) using long short term memory (LSTM). These areas are delivering the current state of the art in time-series recognition tasks like speech recognition and handwriting recognition. RNN/auto encoders are also capable of generating handwriting/ speech/images with some known distribution [17], [18], [19], [20], [21]. Deep belief networks, another promising type of network utilizing restricted Boltzman machines (RMBs)/auto encoders, are capable of being trained greedily, one layer at a time, and hence are more easily trainable for very deep networks [22],[23]

VI. USING CNN FOR CLASSIFICATION

Convolutional neural networks are essential tools for deep learning, and are especially suited for image recognition. The following MatLab program is used to identify digits by creating model from training labelled dataset and using the model for classification.

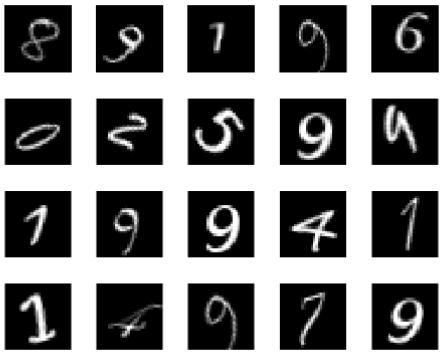

Figure 6.0 Sample from dataset

Label	Count
0	1000
1	1000
2	1000
3	1000
4	1000
5	1000
6	1000
7	1000
8	1000
9	1000

```
accuracy =

    0.9908
```

Figure 6.1 Accuracy of Cnn described on the Side Fig.

```
% Load and Explore Image Data
digitDatasetPath = fullfile(matlabroot,'toolbox','nnet','nndemos', ...
    'nndatasets','DigitDataset');
imds = imageDatastore(digitDatasetPath, ...
    'IncludeSubfolders',true,'LabelSource','foldernames');

%Display some of the images in the datastore.
    figure;
perm = randperm(10000,20);
for i = 1:20
    subplot(4,5,i);
    imshow(imds.Files{perm(i)});
end

%Calculate the number of images in each category
labelCount = countEachLabel(imds)

%Specify Training and Validation Sets
numTrainFiles = 750;
[imdsTrain,imdsValidation] = splitEachLabel(imds,numTrainFiles,'randomize');
```

Figure 6.2 MatLab code for Image classification (a)

```
%Define the convolutional neural network architecture.
layers = [
    imageInputLayer([28 28 1])

    convolution2dLayer(3,8,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)

    convolution2dLayer(3,16,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)

    convolution2dLayer(3,32,'Padding','same')
    batchNormalizationLayer
    reluLayer

    fullyConnectedLayer(10)
    softmaxLayer
    classificationLayer];
```

Figure 6.2 MatLab code for Image classification (b)

```
% Load and Explore Image Data
digitDatasetPath = fullfile(matlabroot,'toolbox','nnet','nndemos', ...
    'nndatasets','DigitDataset');
imds = imageDatastore(digitDatasetPath, ...
    'IncludeSubfolders',true,'LabelSource','foldernames');

%Display some of the images in the datastore.
    figure;
perm = randperm(10000,20);
for i = 1:20
    subplot(4,5,i);
    imshow(imds.Files{perm(i)});
end

%Calculate the number of images in each category
labelCount = countEachLabel(imds)

%Specify Training and Validation Sets
numTrainFiles = 750;
[imdsTrain,imdsValidation] = splitEachLabel(imds,numTrainFiles,'randomize');
```

Figure 6.2 MatLab code for Image classification (c)

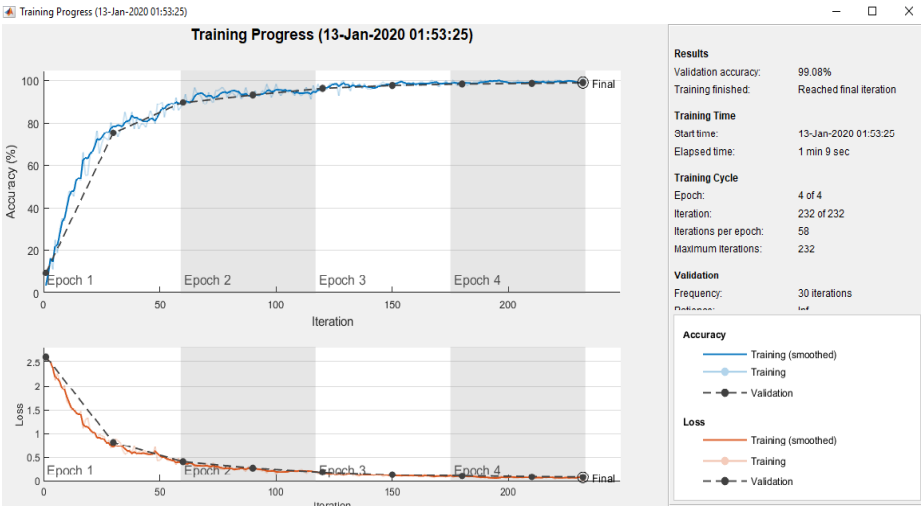

Figure 6.3 Training details with accuracy and loss data

VII. References

[1]. K. Hornik, M. Stinchcombe, and H. White. *Multilayer feedforward networks are universal.* San Diego : Pcrgamon Press plc, 1989.

[2]. Karpathy, Andrej. *Neural Networks for Visual Recognition,.* 2015.

[3]. Convolutional neural network. *Wikipedia.* [Online] https://en.wikipedia.org/wiki/Convolutional_neural_network.

[4]. *Convolutional networks and applications in vision.* Y. LeCun, K. Kavukvuoglu, and C. Farabet. 2010.

[5]. Artificial Neural Network. *WIkipedia.* [Online] https://en.wikipedia.org/wiki/Artificial_neural_network.

[6]. *Computer Vision, International Conference on,.* K. Jarrett, K. Kavukcuogl, M. Ranzato, and Y. LeCun. 2009.

[7]. *Advances in Neural Information Processing Systems,.* A. Krizhevsky, I. Sutskever, and G. E. Hinton. 2012.

[8]. *In Computer Vision, International Conference.* K. Jarrett, K. Kavukcuogl, M. Ranzato, and Y. LeCun. 2009.

[10]. *Visualizing and understanding convolutional networks.* Fergus, M. D. Zeiler and R. 2013.

[11]. *Computing Research Repository.* D. C. Ciresan, U. Meier, and J. Schmidhuber. 2012. : s.n.

[12]. *Visualizing and understanding convolutional networks. Computing.* Fergus., M. D. Zeiler and R. 2013.

[13]. *Deconvolutional networks. In Computer Vision and Pattern Recognition, Conference.* M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus. 2010.

[14]. *Generalization and network design strategies, In Connectionism in Perspective.* LeCun., Y. 1989.

[15]. *Computer Vision: A Modern Approach. Prentice Hall Professional.* Ponce, D. Forsyth and J. New Jersey : s.n., 2002.

[16]. *Backpropagation applied to handwritten zip code recognition. Neural Computation.* Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard,W. Hubbard, and L. D. Jackel. 1989.

[17]. *Man Vs. Computer: Benchmarking Machine Learning Algorithms For Traffic Sign Application.".* Stallkamp, J., M. Schlipsing, J. Salmen, and C. Igel. 2011.

[18]. *Long Short-Term Memory." Neural Computation,.* Hochreiter, Sepp, and Jürgen Schmidhuber. 1997. 1997.

[19]. *Generating Sequences With Recurrent Neural Networks.* Graves, Alex. 2014.

[20]. *Recurrent Neural Networks.* Schmidhuber, Jurgen. 2015.

[21]. *Emergence of simple-cell receptive field properties by learning.* Olshausen, Bruno A., and David J. Field. 1996.

[22]. *"Reducing the dimensionality of data with neural networks.".* Hinton, G. E. and Salakhutdinov, R. R. 2006.

[23]. *Deep belief networks.* Hinton, Geoffrey E. 2009.