CONVOLUTION NEURAL NETWORKS FOR FACE RECOGNITION AND FEATURE EXTRACTION

[1]Fareena, [2]Divya Ravi. N
Dept. of Information Science and Engineering, Alva's Institute of Engineering and Technology

Abstract
A robust human identity authentication system is vital nowadays due to the increasing number of crime and losses through identity fraud. And thus, facial recognition for verification and validation has been one of the major evolutions through neural networks. However, it still remains one of the challenging problems. The main challenge is how to improve the recognition performance when affected by the variability of non-linear effects that include illumination variances, poses, facial expressions, occlusions, etc.

In the paper, robust 4-layer Convolutional Neural Network (CNN) architecture is proposed for the face recognition problem, with a solution that is capable of handling facial images.

We will outline the most important existing approaches to facial image analysis and present novel methods based on Convolutional Neural Networks (CNN) to detect, normalize and recognize faces and facial features. CNN is inspired by visual mammalian cortex of simple and complex cells. It consists of 4- 8 layers with image processing tasks incorporated into the design. CNN applies three architectural concepts in its architecture namely shared weights, local receptive field and sub sampling. They show to be a powerful and flexible feature extraction and classification technique which has been successfully applied in other contexts, i.e. hand-written character recognition, and which is very appropriate for face analysis problems.

INTRODUCTION

Face detection is a well studied problem in computer vision. Modern face detectors can easily detect near frontal faces. Recent research in this area focuses more on the uncontrolled face detection problem, where a number of factors such as pose changes, exaggerated expressions and extreme illuminations can lead to large visual variations in face appearance, and can severely degrade the robustness of the face detector. The difficulties in face detection mainly come from two aspects: 1) The large visual variations of human faces in the cluttered backgrounds 2) The large search space of possible face positions and face sizes. The former one requires the face detector to accurately address a binary classification problem while the latter one further imposes a time efficiency requirement. In the paper, we propose to apply the Convolutional Neural Network (CNN) for face detection. Compared with the previous hand-crafted features, CNN can automatically learn features to capture complex visual variations by leveraging a large amount of training data and its testing phase can be easily parallelized on GPU cores for acceleration. Convolutional nets can be used to classify images (name what they see), cluster them by similarity (photo search), and perform object recognition within scenes. They can identify faces, individuals, street signs, eggplants, platypuses and many other aspects of visual data..

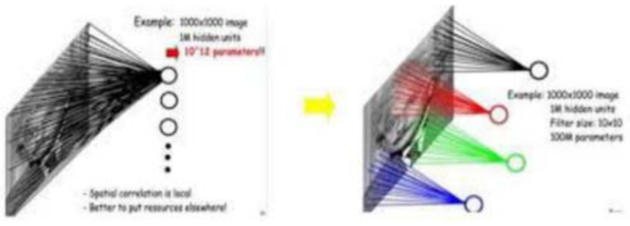
I. PRINCIPLES OF CNN

Convolution is the integral measuring how much two functions overlap as one passes over the other. Think of a convolution as a way of mixing two functions by multiplying them. With image analysis, the static, underlying function (the equivalent of the immobile bell curve) is the input image being analyzed, and the second, mobile function is known as the filter, because it picks up a signal or feature in the image. The two functions relate through multiplication. The next thing to understand about convolutional nets is that they are passing *many* filters over a single image, each one picking up a different signal. At a fairly early layer, you could imagine them as passing a horizontal line filter, a vertical line filter, and a diagonal line filter to create a map of the edges in the image. Convolutional networks take those filters, slices of the image's feature space, and map them one by one; that is, they create a map of each place that feature occurs. By learning different portions of a feature space, convolutional nets allow for easily scalable and robust feature engineering. (Note that convolutional nets analyze images differently than RBMs. While RBMs learn to reconstruct and identify the features of each image as a whole, convolutional nets learn images in pieces that we call feature maps.) So convolutional networks perform a sort of search. Picture a small magnifying glass sliding left to right across a larger image, and recommencing at the left once it reaches the end of one pass (like typewriters do). That moving window is capable recognizing only one thing, say, a short vertical line. Three dark pixels stacked atop one another. It moves that vertical-line-recognizing filter over the actual pixels of the image, looking for matches.

Each time a match is found, it is mapped onto a feature space particular to that visual element. In that space, the location of each vertical line match is recorded, a bit like birdwatchers leave pins in a map to mark where they last saw a great blue heron. A convolutional net runs many, many searches over a single image – horizontal lines, diagonal ones, as many as there are visual elements to be sought

II. METHODOLOGY

CNN algorithm has two main processes: convolution and sampling . Convolution process: Uses a trainable filter Fx, deconvolution of the input image (the first stage is the input image, the input of the after convolution is the feature image of each layer, namely Feature Map), then add a bias bx, we can get convolution layer Cx. A sampling process: n pixels of each neighborhood through pooling steps, become a pixel, and then by scalar weighting Wx + 1 weighted, add bias bx + 1, and then by an activation function, produce a narrow n times feature map Sx + 1.

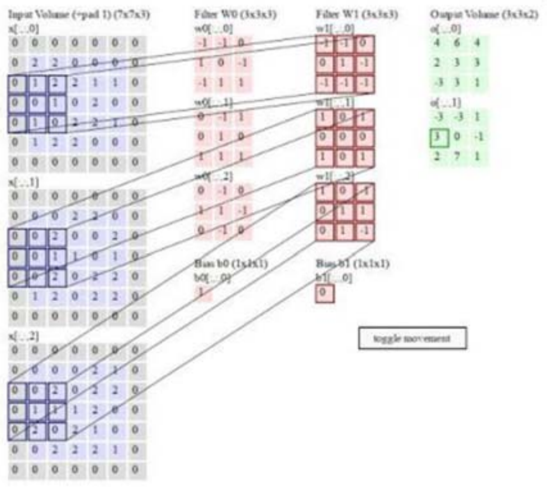


.

III. WORKING

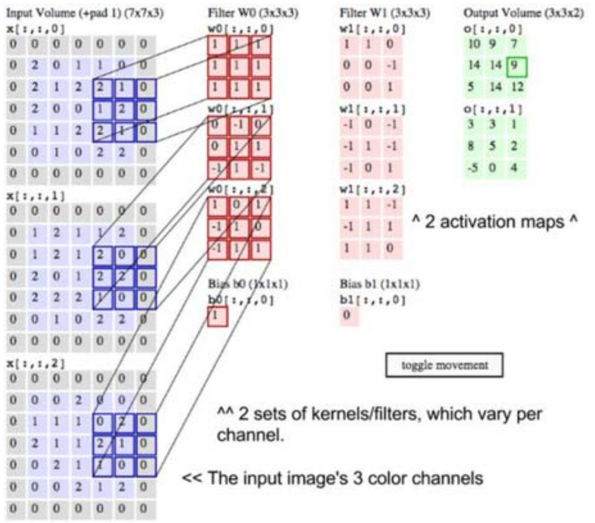
Convolutional networks perceive images as volumes; i.e. three-dimensional objects, rather than flat canvases to be measured only by width and height. That's because digital color images have a red-blue-green (RGB) encoding, mixing those three colors to produce the color spectrum humans perceive. A convolutional network ingests such images as three separate strata of color stacked one on top of the other. So a convolutional network receives a normal color image as a rectangular box whose width and height are measured by the number of pixels along those dimensions, and whose depth is three layers deep, one for each letter in RGB. Those depth layers are referred to as channels. As images move through a convolutional network, we will describe them in terms of input and output volumes, expressing them mathematically as matrices of multiple dimensions in this form: 30x30x3. From layer to layer, their dimensions change for reasons that will be explained below. Pay close attention to the precise measures of each dimension of the image volume, because they are the foundation of the linear algebra operations used to process images.

Now, for each pixel of an image, the intensity of R, G and B will be expressed by a number, and that number will be an element in one of the three, stacked two-dimensional matrices, which together form the image volume. Those numbers are the initial, raw, sensory features being fed into the convolutional network, and the ConvNets purpose is to find which of those numbers are significant signals that actually help it classify images more accurately. (Just like other feedforward networks we have discussed.) Rather than focus on one pixel at a time, a convolutional net takes in square patches of pixels and passes them through a filter. That filter is also a square matrix smaller than the image itself, and equal in size to the patch. It is also called a *kernel*, which will ring a bell for those familiar with support-vector machines, and the job of the filter is to find patterns in the pixels.



Imagine two matrices. One is 30x30, and another is 3x3. That is, the filter covers one-tenth of one image channel's surface area. We are going to take the dot product of the filter with this patch of the image channel. If the two matrices have high values in the same positions, the dot product's output will be high. If they don't, it will be low. In this way, a single value – the output of the dot product – can tell us whether the pixel pattern in the underlying image matches the pixel pattern expressed by our filter.

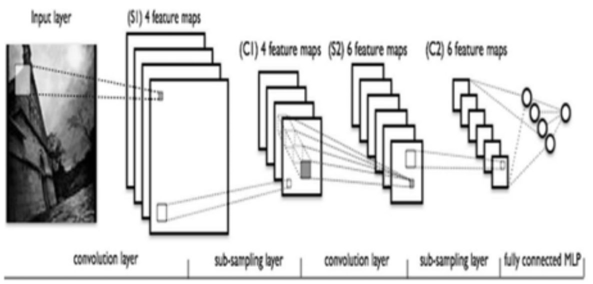
Let's imagine that our filter expresses a horizontal line, with high values along its second row and low values in the first and third rows. Now picture that we start in the upper left hand corner of the underlying image, and we move the filter across the image step by step until it reaches the upper right-hand corner. The size of the step is known as *stride*. At each step, take another dot product, and you place the results of that dot product in a third matrix known as an *activation map*. The width, or number of columns, of the activation map is equal to the number of steps the filter takes to traverse the underlying image. Since larger strides lead to fewer steps, a big stride will produce a smaller activation map. This is important, because the size of the matrices that convolutional networks process and produce at each layer is directly proportional to how computationally expensive they are and how much time they take to train.



A larger stride means less time and compute. A filter superimposed on the first three rows will slide across them and then begin again with rows 4-6 of the same image. Because images have lines going in many directions, and contain many different kinds of shapes and pixel patterns, you will want to slide other filters across the underlying image in search of those patterns. For example, look for 96 different patterns in the pixels. Those 96 patterns will create a stack of 96 activation maps, resulting in a new volume that is 10x10x96. In the diagram below, we've relabeled the input image, the kernels and the output activation maps to make sure we're clear.

IV. CNN ARCHITECTURE

CNNs are comprised of three types of layers. These are convolutional layers, pooling layers and fully-connected layers. When these layers are stacked, a CNN architecture has been formed. A simplified CNN architecture for MNIST classification is illustrated in following figure.



The basic functionality of the example CNN above can be broken down into four key areas. 1. As found in other forms of ANN, the input layer will hold the pixel values of the image. 2. The convolutional layer will determine the output of neurons of which are connected to local regions of the input through the calculation of the scalar product between their weights and the region connected to the input volume. The rectified linear unit (commonly shortened to ReLu) aims to apply an 'elementwise' activation function such as sigmoid to the output of the activation produced by the previous layer. 3. The pooling layer will then simply perform downsampling along the spatial dimensionality of the given input, further reducing the number of parameters within that activation. 4. The fully-connected layers will then perform the same duties found in standard ANNs and attempt to produce class scores from the activations, to be used for classification. It is also suggested that ReLu may be used between these layers, as to improve performance.

Through this simple method of transformation, CNNs are able to transform the original input layer by layer using convolutional and downsampling tech-niques to produce class scores.

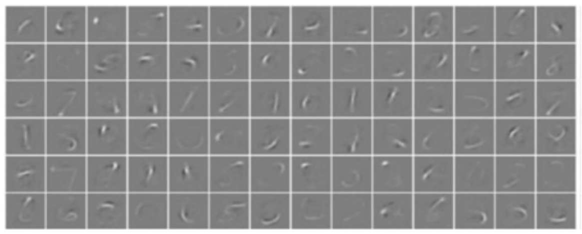
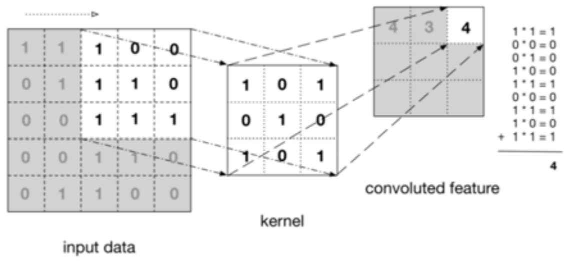


Fig: Real convolution image

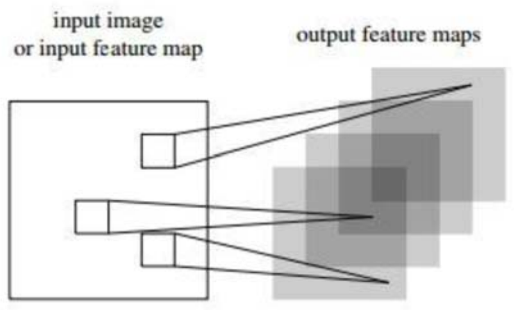
The creation and optimization of these models can take quite some time, and can be quite confusing. We will now explore in detail the individual layers, detailing their hyperparameters and connectivities.

A. Convolution layer

The convolutional layer plays a vital role in how CNNs operate. The layers parameters focus around the use of learnable kernels. These kernels are usually small in spatial dimensionality, but spreads along the entirety of the depth of the input. When the data hits a convolutional layer, the layer convolves each filter across the spatial dimensionality of the input to produce a 2D activation map. As we glide through the input, the scalar product is calculated for each value in that kernel. (Figure 4) From this the network will learn kernels that 'fire' when they see a specific feature at a given spatial position of the input. These are commonly known as activations.



The centre element of the kernel is placed over the input vector, of which is then calculated and replaced with a weighted sum of itself and any nearby pixels. Every kernel will have a corresponding activation map, of which will be stacked along the depth dimension to form the full output volume from the convolutional layer.



To calculate this, you can make use of the following formula: (V − R) + 2Z S + 1 Where V represents the input volume size (height× width×depth), R represents the receptive field size, Z is the amount of zero padding set and S referring to the stride.

B. Pooling layer

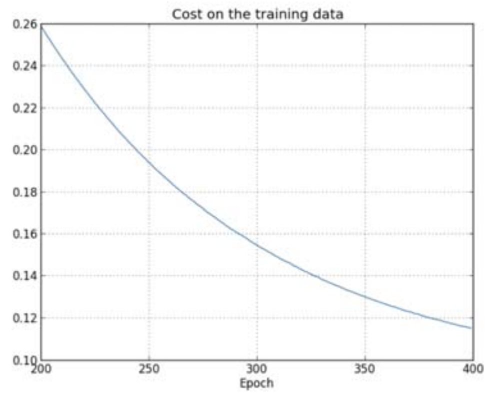
Pooling layers aim to gradually reduce the dimensionality of the representation, and thus further reduce the number of parameters and the computational complexity of the model.

The pooling layer operates over each activation map in the input, and scales its dimensionality using the "MAX" function. In most CNNs, these come in the form of max-pooling layers with kernels of a

dimensionality of 2 × 2 applied with a stride of 2 along the spatial dimensions of the input. This scales the activation map down to 25% of the original size - whilst maintaining the depth volume to its standard size. Due to the destructive nature of the pooling layer, there are only two generally observed methods of max-pooling. Usually, the stride and filters of the pooling layers are both set to 2 × 2, which will allow the layer to extend through the entirety of the spatial dimensionality of the input. Furthermore overlapping pooling may be utilized, where the stride is set to 2 with a kernel size set to 3. Due to the destructive nature of pooling, having a kernel size above 3 will usually greatly decrease the performance of the model. It is also important to understand that beyond max-pooling, CNN architectures may contain general-pooling. General pooling layers are comprised of pooling neurons that are able to perform a multitude of common operations including L1/L2-normalisation, and average pooling.

V. COMPUTATION

The convolution operation is defined as a multiplication operation between a filter vector u ∈ Rmd and a concatenation vector representation xi:i+m−1 given by: xi:i+m−1 = xi ⊕ xi+1 ⊕ · · · ⊕ xi+m−1 (1) where xi:i+m−1 represents a window of m continuous time steps starting from the i-th time step. In addition, a bias term b is also added into the convolution operation, so that the final operation is given as: ci = g(u T xi:i+m−1 + b) (2) where ∗ T denotes the transpose of a matrix ∗ and g is a non-linear activation function that is set to Rectified Linear Units(ReLu)in our model [43]. Each vector u can be regarded as a filter, and the single value ci can be regarded as the activation of the window. Max-pooling: The convolution operation over the whole sequence is applied by sliding the filtering window from the beginning time step to the ending time step. It is easily shown that a feature map is a vector denoted as follows: cj = [c1, c2, . . . , cl−m+1] (3) where the index j denotes the j-th filter. It corresponds to multi-windows as {x1:m, x2:m+1, . . . , xl−m+1:l}. The pooling layer is able to reduce the length of the feature map, which can further minimize the number of model parameters. The hyper-parameter of pooling layer is the pooling length denoted as s. The max operation is taking a max over the s consecutive values in feature map cj . Then, the compressed feature vector can be obtained as: h = h h1, h2, . . . , h l−m s +1 i (4) where hj = max(c(j−1)s , c(j−1)s+1 , . . . , cjs−1). Generally, multiple filters are applied with different initialized weights to derive the output of the CNN layer. Generally, the size of the input sequence in the CNN layer is n × l × d, and n is the number of data samples. The size of the corresponding outputs is n × (l−m s + 1) × k. It is easily shown that after the convolutional and pooling operation, the length of sequence data can be compressed from l to (l−m s + 1). Compared to the original representation is raw sensory data with a dimensionality of d that is usually the number of sensors in each time step; more abstract and informative representation can be learned after CNN, and the corresponding dimensionality is k, which is the number of filters. The cost on training data is as follows



Layer	Type	Units	x	y	Receptive field x	Receptive field y	Connection Percentage
1	Convolutional	20	21	26	3	3	100
2	Subsampling	20	11	13	2	2	–
3	Convolutional	25	9	11	3	3	30
4	Subsampling	25	5	6	2	2	–
5	Fully connected	40	1	1	5	6	100

Fig:Dimensions for the convolutional network. The connection percentage refers to the percentage of nodes in the previous layer which each node in the current layer is connected to – a value less than 100% reduces the total number of weights in the network and may improve generalization. The connection strategy used here is similar to that used by Le Cun et al. [24] for character recognition.

The visualization of the feature activations across the convolutional layers allows evaluating the effect of filter size as well as filtering placement. For example, by analyzing the feature activations of the first and second layer, the authors 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 .

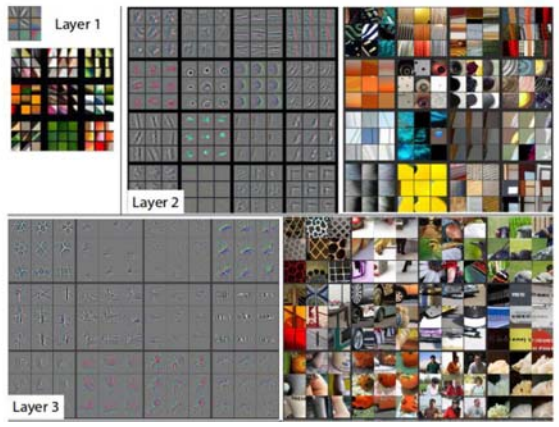


Figure shows a selection of features across several layers of a fully trained convolutional network using the visualization technique

CONCLUSION

In the course of the paper we discussed the basic notions of both neural networks in general and the multilayer perceptron in particular. We introduced convolutional neural networks by discussing the different types of layers used in recent implementations: the convolutional layer; the non-linearity layer; the rectification layer; the local contrast normalization layer; and the pooling and subsampling layer. Based on these basic building blocks, we discussed the traditional convolutional neural networks. Nevertheless, convolutional neural networks and deep learning in general is an active area of research. Although the difficulty of deep learning seems to be understood learning feature hierarchies is considered very hard. Here, the possibility of unsupervised pre-training had a huge impact and allows to train deep architectures in reasonable time. Nonetheless, the reason for the good performance of deep neural networks is still not answered fully channel through virtualization techniques.

REFERENCES

1. Behnke, 2003] Sven Behnke. *Hierarchical Neural Network for Image Interpretation*, volume 2766 of *Lecture Notes in Computer Science*. Springer, 2003.

2. [Boureau *et al.*, 2010] Y-Lan Boureau, Jean Ponce, and Yann LeCun. A Theoretical Analysis of Feature Pooling inVisual Recognition. In *International Conference on Machine Learning*, 2010.

3. [Chellapilla *et al.*, 2006] Kumar Chellapilla, Sidd Puri, and Patrice Simard. High performance convolutional neural networks for document processing. In *International Workshop on Frontiers in Handwriting Recognition*, 2006

4. [Ciresan *et al.*, 2010] Dan C. Ciresan, Ueli Meier, Luca M.

5. Gambardella, and J¨urgen Schmidhuber. Deep big simple neural nets for handwritten digit recognition. *Neural Computation*,22(12):3207–3220, 2010.

6. [Coates *et al.*, 2010] Adam Coates, Honglak Lee, and Andrew Ng. An analysis of single-layer networks in unsupervised feature learning. In *Advances in Neural Information Processing Systems*, 2010.

7. [Fukushima, 1980] Kunihiko Fukushima. Neocognitron: A self-organizing neural network for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*,

8. 36(4):193–202, 1980 [Fukushima, 2003] Kunihiko Fukushima. Neocognitron for handwritten digit recognition. *Neurocomputing*, 51:161 180, 2003.

9. [Hoyer and Hyv¨arinen, 2000] Patrik O. Hoyer and Aapo Hyv¨arinen. Independent component analysis applied to feature extraction from colour and stero images. *Network: Computation in Neural Systems*, 11(3):191–210, 2000.

10. [Jarrett *et al.*, 2009] Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. What is the best multi-stage architecture for object recognition? In *Proc.International Conference on Computer Vision*, 2009.