Contents

1 Theory

In this chapter will dicussed concepts of Image processing, Machine learning in general, Neural networks and finaly the Convolutional neural networks.

1.1 Image Processing

There was no wider adoption of that machine learning techniques in image processing for very long time, Even though they existed as field of study since 1950's. Reason being that machine learning algorithms were very simple and therefore unfit for generally very complex problems of image processing (e.g. object detection and classification).

Research in image processing was adopting predominately bottom up (feature-based) methodologies. Typical classification pipeline in computer vision would be following:

- Image capture - Image is captured (by camera or similar device) and digitized.
- Preprocessing - Digitized image is modified to emphasize important features (e.g. noise reduction, contrast normalization etc.).
- Segmentation - Selection of interesting features (edges, similar surfaces).

- Description - Extraction of radiometric, photometric descriptors and so on.

- Classification - Some means to classify the object.

In the first attempts to apply machine learning in image processing, it was used usually deployed as the classification model. In other words complex problems were reduced and simplified in order to utilize machine learning techniques. Consequently this approach favors simpler models.

This also brings the problem that any of these applications is not very versatile. Each application is usually only capable of solving very simple problem and any deviation from ideal circumstances can mean failure. Application can have problem with varying contrast, illumination, scaling, rotation etc.

Second problem is often the fact that because image has to be preprocessed several times before it is fed into machine learning model it demands extra time and resources. This can created problems in training phase but also during operation after deployment. This is less of a problem with technical innovation and today it can be solved by very fast DSP, but it still is not negligible effect and it can have negative affect on the price of the solution. This is where Deep learning models starts to exhibit significant advantage.

Deep Learning models are in theory capable of capturing complex (higher order) features of the input due to the fact they can learn very high level of abstraction from even very low level features. And what is even more important, is the fact that Deep learning model in theory doesn't need any kind (or very little) of preprocessing. Input image can be directly connected as input into Deep Network.

1.2 Machine Learning

It is necessary to properly define machine learning before we move forward. There are many valid definitions that may differ but in this text we will adhere to the following. Field of Machine learning is a subset of ai that is describing tools that can be used to solve problems.

In order to properly explain We will use a definition from [?]

1.2.1 Task

Loss function

Optimization

1.2.2 Supervised Learning

1.2.3 Unsupervised Learning

1.2.4 Generalization of Machine Learning

Machine learning approach to solving problmes can be generalized.

1.3 Neural Networks

Neural networks are definitely inspired by biology. First attempts to created model of neuron had multiple elements equivalent with neurons of human brain. As time progressed this equivalence sized to being as important and modern neural networks models correspond to h their biological counterparts only superficially.

In the first iterations of mlp evolution it generally had all neurons in hidden and output layer to be similar. Namely activation function was predominantly sigmoid. Activation function of hidden layers is to this day one of the most dynamically evolving components of Neural networks. Currently there is several options for but mainly used is Restricted Linear Unit (ReLU) which models following mathematical function

$$g(z) = \max\{0, z\} \tag{1}$$

img$_{\_2\_\text{relu}}$

1.3.1 Deep Learning in image processing

It was found that general **Fully Connected Neural Network** is not ideal for image processing needs. Even small images typically represents enormous amount of inputs (i.e. image of the size 64x64 pixels represents 4096 inputs). Since each of these inputs has to be connected to all neurons in following layer and weight of each connection has to be memorized, this represents enormous amount of parameters. Moreover because during the learning process update of these weights is computed via matrix multiplication for larger images this can be unresolvable problem, which exacerbate with the number of hidden layers.

The structure of FCNN has another deficiency for image processing application, which is that it doesn't capture geometric properties of information from input image. In other words because individual layers are fully connected (each output in lower layer is connected to each input in higher layer) networks are not capturing any information about relation of position of individual inputs (image pixels) to each other.

Third problem is that for higher depth of fcnn increases the likelihood of getting stuck in some local minima.

All of these problems were solved by the specific type of nn model called cnn [citation]

1.4 Convolutional Neural Networks

cnn are specialized type of nn that was originally used in image processing applications. They are arguably most successful models in ai inspired in biology. Even though they were guided by many different fields, the core design principles were drawn from neuroscience. Since their success in image processing, they were also very successfully deployed in natural language and video processing applications.

Aforementioned inspiration in biology was based on scientific work of David Hubel and Torsten Wiesel. Hubel and Wisel, who were neurophysiologist, investigated vision system of mammals from late 1950 for several years. In the experiment, that might be considered little gruesome for today's standards, they connected electrodes into brain of anesthetized cat and measured brain response to visual stimuli [Citation]. They discovered that reaction of neurons in visual cortex was triggered by very narrow line of light shined under specific angle on projection screen for cat to see. They determined that individual neurons from visual cortex are reacting only to very specific features of input image. Hubel and Wiesel were awarded the Nobel Prize in Physiology and Medicine in 1981 for their discovery and their finding inspired design of cnn.

There will be several suppositions made in order to simplify explanation of the concepts involved:

- It will be presumed that convolutional layer is working with rectangular input data (e.g. images). Even though the Convolutional networks can be also trained to use 1-dimensional input (e.g. sound signal) or 3-dimensional (e.g. mri images) etc.
- The complexity of multiple-channel inputs (i.e. colored images) will be ignored.
- Each layer requires rectangular input and produces rectangular output per one kernel.

1.4.1 Structure of CNN

Structure of Convolutional networks is typically composed of three different types of layers. Layer can be of Convolutional, Pooling and *Fully-connected* type. Each type of layer has different rules for forward and error backward signal propagation.

Convolutional layer As the name suggests this layer employs convolution operation. Input into this layer is simply called input. Convolution operation is performed on input with specific filter, which is called kernel. Output of convolution operation is typically called *feature map*.

Input into Convolutional layer is either image (in case of first network layer) or *feature map* from previous layer. Kernel is typically of square shape and its width can range from 3 to N pixels (typically 3, 5 or 7). *Feature map* is created by convolution of kernel over each specified element of input. Convolution is described in more detail in section describing training of CNN.

Depending on the size of kernel and layer's padding preferences the process of convolution can produce *feature map* of different size than input. When the size of output should be preserved it is necessary to employ *zero padding* on the edges of input. *Zero padding* in this case has to add necessary amount of zero elements around the edges of input. This amount is determined by

$$p = ((h-1)/2) \tag{2}$$

where h is width of used kernel. In opposite case the *feature map* is reduced by the $2 * p$. Decreasing of the *feature map* can be in some cases desirable.

Reduction of *feature map* can go even further in case of use of stride. Application of stride specifies by how many input points is traversed when moving to neighboring position in each step. When the stride is 1, kernel is moved by 1 on each step and the resulting size of *feature map* is not affected.

Each Convolutional layer is typically composition of several different kernels. In other words output of this layer is tensor containing *feature map* for each used kernel. Each of these is designed to underline different features of input image. In the first layers these features are typically edges. In following layers the higher the layer the more complex features are captured.

Each kernel that is used is applied to all inputs of the image to produce one *feature map* which basically means that neighboring layers are sharing the same weights. This might not be sufficient in some applications and therefore it is possible to use two other types of connections. *Locally connected* which basically means that applied kernel is of the same size as the input and *tiled convolution* which means alternation of more than one set of weights on entire input.

Tiled convolution is interesting because with clever combination with *max-pooling* explained bellow it allows to train specific feature from multiple angles (in other words invariant to rotation).

Each convolutional layer has non-linearity on its output that is sometimes also called the *detector stage.*

Pooling layer This layer typically (more details later) doesn't constitute any learning process but it is used to down-sample size of the input. The Principle is that input is divided into multiple not-overlapping rectangular elements and units within each element are used to create single unit of output. This decreases the size of output layer while preserving the most important information contained in input layer. In other words pooling layer compresses information contained within input.

Type of operation that is performed on each element determines a type of pooling layer. This operation can be averaging over units within element, selecting maximal value from element or alternatively learned linear combination of units within element. Learned linear combination introduces form of learning into the pooling layer, but it is not very prevalent.

Selecting of maximal value is most common type of pooling operation and in that case the layer is called *Max-Pooling* accordingly. Positive effect of Max-pooling down-sampling is that extracted features that are learned in convolution are invariant to small shift of input. *Max-Pooling* layer will be used to describe process of training of cnn.

As already mentioned another advantage of Max-pooling arises when combined with *Tiled convolution.* To create simple detector that is invariant to rotation it possible to use 4 different kernels that are rotated by 90 degrees among each other and when the *tiled convolution* is used to tile them in groups of 4, the Max-pooling makes sure that resulted *feature map* contains output from the kernel with strongest signal (i.e. the one trained for that specific rotation of the feature).

Fully-Connected layer Fully-Connected layer is formed from classical neurons that can be found in fcnn and it is always located at the end of the layer stack. In other words it is never followed by another Convolutional layer. Depending on the size of whole cnn it can have 1 to 3 *fully connected* layers (usually not more than that). Input of the first FC layer has inputs from all neurons from previous layer to all neurons of following layer (hence fully connected). All fully connected layers are together acting as fcnn.

1.4.2 Training of CNN

Training process of cnn is analogues to fcnn in that both are using *Forward Propagation* and *Backward Propagation* phases.

Situation with cnn is more complicated because network is composed of different types of layers and therefore training must accommodate for variability between different layers and also the individual convolution layers are sharing weights across all neurons in each layer.

First phase is the *Forward Propagation*, where the signal is propagated from inputs of the cnn to its output.

In the last layer the output is compared with desired values by *Error function E* and error is estimated.

Secondly in *Backward Propagation* phase the error is propagated backwards through the network and weights for individual layers are updated by its contribution on the error. Most commonly used algorithm for update of weights is *Gradient Descent*. It is not the only one used but in majority of cases the training algorithm is at least based on *Gradient descent*.

Forward Propagation

Convolution Layer Each convolutional layer has inputs. In case that the layer is first, it is network input (i.e individual pixels of image) in other cases, the inputs are outputs from neurons from previous layer (this is typically pooling layer).

Presuming that input of a layer is of size NxN units and kernel is of size mxm. Convolution is computed over $(N-m+1)x(N-m+1)$ units (presuming that there is no zero padding).

Computation of convolution output $x_{ij}^{(l)}$ is defined as

$$x_{ij}^{(l)} = \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} \omega_{ab} y_{(i+a)(j+b)}^{(l-1)} \tag{3}$$

where $i, j \in (0, N-m+1)$, l is index of current layer, ω_{ab} are weights of layer (kernel) and $y_{(i+a)(j+b)}^{(l-1)}$ is output of previous layer.

Output of convolutional layer $y_{ij}^{(l)}$ is computed by squashing of output of convolution operation $x_{ij}^{(l)}$ through non-linearity:

$$y_{ij}^{(l)} = \sigma(x_{ij}^{(l)}) \tag{4}$$

where σ represents this non-linear function. equation

Pooling layer (Max-Pooling) Feed forward operation of pooling layer is generally very simple and it constitutes in selecting of maximal value within subset pooling of multiple inputs into single output. Ratio is typically 4 to 1, which means that input matrix is divided into not-overlapping sub-matrices of size 2x2 and each of these produces 1 output. Size of sub-matrices can vary and is dependent on size of input, number of layers.

Fully Connected layer Signal is distributed through FC layer in similar fashion as in Convolutional layer. The main difference is that weights of individual neuron connections are not shared among all neurons in one layer.

Backward Propagation

Convolution Layer Following equasions were lifted from [?].

$$\frac{\partial E}{\partial \omega_{ab}} = \sum_{i=0}^{N-m} \sum_{j=0}^{N-m} \frac{\partial E}{\partial x_{ij}^{l}} \frac{\partial x_{ij}^{l}}{\partial \omega_{ab}} = \sum_{i=0}^{N-m} \sum_{j=0}^{N-m} \frac{\partial E}{\partial x_{ij}^{l}} y_{(i+a)(j+b)}^{l-1} \tag{5}$$

$$\frac{\partial E}{\partial x_{ij}^{(l)}} = \frac{\partial E}{\partial y_{ij}^{l}} \frac{\partial y_{ij}^{l}}{\partial x_{ij}^{l}} = \frac{\partial E}{\partial y_{ij}^{l}} \frac{\partial}{\partial x_{ij}^{l}} \left(\sigma\left(x_{ij}^{l}\right)\right) = \frac{\partial E}{\partial y_{ij}^{l}} \sigma'\left(x_{ij}^{l}\right) \tag{6}$$

$$\frac{\partial E}{\partial y_{ij}^{l-1}} = \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} \frac{\partial E}{\partial x_{(i-a)(j-b)}^{l}} \frac{\partial x_{(i-a)(j-b)}^{l}}{\partial y_{ij}^{l-1}} = \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} \frac{\partial E}{\partial x_{(i-a)(j-b)}^{l}} \omega_{ab} \tag{7}$$

Pooling layer (Max-Pooling) As mentioned in section for *forward propagation*, there is no explicit learning process happening in pooling layer. Error is propagated backwards depending on how the signal was propagated forward. In case of *Max-pooling* layer the error is propagated only to the unit with maximal output in *forward propagation* phase (in other words to the winner of pooling). The error is propagated very sparsely, as result.

In case of different pooling method it is adjusted accordingly (i.e. for *average pooling* the error is propagated according to contribution of individual neurons).

Fully connected layer Training mechanism for FC layer if following the same principles as in FCNN, which is not a subject of detailed discussed here. It is similar to one for convolution layers and from our perspective is only important that the first (last in the sense of *Backward Propagation*) FC layer propagates error gradient of each neuron in it, that is then send to all neurons in preceding (following in the sense of *Backward Propagation*) layer.

1.4.3 Advantages of CNN

To further highlight the difference between fcnn and cnn it is worth to compare the case of 2 neighboring layers. Lets have gray scale input image of size 32x32 pixels and following layer will be convolutional with 6 feature maps of size 28x28. Kernels used in this convolutional layer will have the size of 5x5. In this case we have totally $(5 * 5 + 1) * 6 = 156$ parameters between the two layers. If we would like to create equivalent connection between two layers of fcnn, then it would have mean $(32 * 32 + 1) * 28 * 28 = 803600$ connections (parameters). Which means that difference between the two is of ~5000 ratio. This difference would rise exponentially with larger images or with more color channels. When input size of the image changes to 64x64 and it has rgp color then fcnn would requires $(64*64*3+1)*28*28 = 9634576$ connections (parameters). In the same case the CNN only needs $(5 * 5 * 3 + 1) * 6 = 456$ parameters. Which is difference of ~20000 factor. Just to elaborate, in case that CNN would be used to process video. Analogically to previous examples in case of moving image in time the number of parameters raises linearly with number of images in analyzed video.