CSE301 Bio-computation week 6-7 Lab and Tutorial

An Introduction to Convolutional Neural Network (I)

Prepared by Chao Yan

1. Background

The success of machine vision task is heavily depended on learning-representation algorithms, while the performance of machine learning methods is dependent on the choice of data representation (or features) on which they are applied. A good representation method can entangle and hide more or less the different explanatory factors of variation behind the observed milieu of low level sensory data, however, mostly requiring specific domain knowledge in design. To expand the scope and ease of applicability of machine learning, the most important perspective on intelligent machine vision solution, is to make learning algorithms less dependent on human ingenuity and prior knowledge compensated feature. Some remarkable works in the area of representation-learning algorithm particularly focus on unsupervised feature learning and deep learning which covers advances in probabilistic models, auto-encoders, manifold learning, and deep learning.

In 2006, a breakthrough in deep learning was initiated by Geoff Hinton. In 2012, with the aid of graphic processing unit(GPU), Krizhevsky et al. proposed a large-scale deep convolutional network trained by standard back propagation, with breakthrough on the large-scale ImageNet object recognition dataset, attaining a significant gap compared with existing approaches that use shallow models, and bringing high impact to research in computer vision.

Convolutional Neural Networks (CNN), the first realization as neocognition, can be considered first proposed by Fukushima. The Neocognition, a biologically-inspired hierarchical and shift-invariant model for pattern recognition, was an early attempt at just such a process, however, lacked a supervised training algorithm. LeCun et al. showed that stochastic gradient descent via back propagation was effective for training convolutional neural networks (CNNs). They combine three architectural ideas (i)***local receptive fields***, (ii)***shared weights***, and (iii)***spatial or temporal sub-sampling***, to ensure some degree of shift, scale, and distortion invariance, and finally successfully applied CNN into document recognition. ***Local receptive fields*** mean the local connection of the receptive fields. Local connections have been used many times in neural models of visual learning. The concept "receptive field" (the filter definition field) originates from Hubel and Wiesel's study on cats' striate cortex. With local receptive fields neurons can extract elementary visual features such as oriented edges, endpoints, corners (or similar features in other signals such as speech spectrograms). These features are then combined by the subsequent layers in order to detect higher order features. ***Shared weights*** are known as local weight sharing topology in CNN which improves tolerance to local distortions. Additionally, the model complexity and the number of weights is efficiently reduced by weight sharing. ***Spatial or temporal sub-sampling*** performs a local averaging and a sub-sampling, thereby reducing the resolution of the feature map and reducing the sensitivity of the output to shifts and distortions.

CNNs saw heavy use in the 1990s, but then fell out of fashion due to (i) the rise of support vector machines (ii) computational constraints. Now, CNNs is back and being again heavily used as two reasons. First, in theory, CNN has the advantage of receiving raw image as input rather than hand-coded features. The model accomplishes feature extracting and classifying as a whole, and both building blocks are learned in a supervised procedure. Second, in practice, improvements on GPU hardware have enabled CNNs to scale to networks of millions of parameters for large amounts of training data, which has in turn led to significant improvements in image classification. Although convolutional neural networks (CNNs) are established as a powerful class of models for image recognition problems, the extension of CNN architecture to different visual tasks remains empirical evaluation including CNN structure, activation function, pooling methods, pre-training, regularization rules, learning strategy and etc.

2. Overview

A Convolutional Neural Network (CNN) is comprised of one or more convolutional layers (often with a subsampling step) and then followed by one or more fully connected layers as in a standard multilayer neural network. The architecture of a CNN is designed to take advantage of the 2D structure of an input image (or other 2D input such as a speech signal). This is achieved with local connections and tied weights followed by some form of pooling which results in translation invariant features. Another benefit of CNNs is that they are easier to train and have many fewer parameters than fully connected networks with the same number of hidden units. In this tutorial we will discuss the architecture of a CNN.

3. CNN building blocks

3.1 Convolution

Convolutional Neural Networks (CNN), a biologically-inspired variant of multi-layer perceptron(MLP), is also known as ”shared weight” neural networks introduced by LeCun et al. From Hubel and Wiesel’s early work on the cat’s visual cortex, it is known that the visual cortex contains a complex arrangement of cells. These cells are sensitive to small sub-regions of the visual field, called a receptive field. The sub-regions are tiled to cover the entire visual field. These cells act as local filters over the input space and are well-suited to exploit the strong spatially local correlation present in natural images.

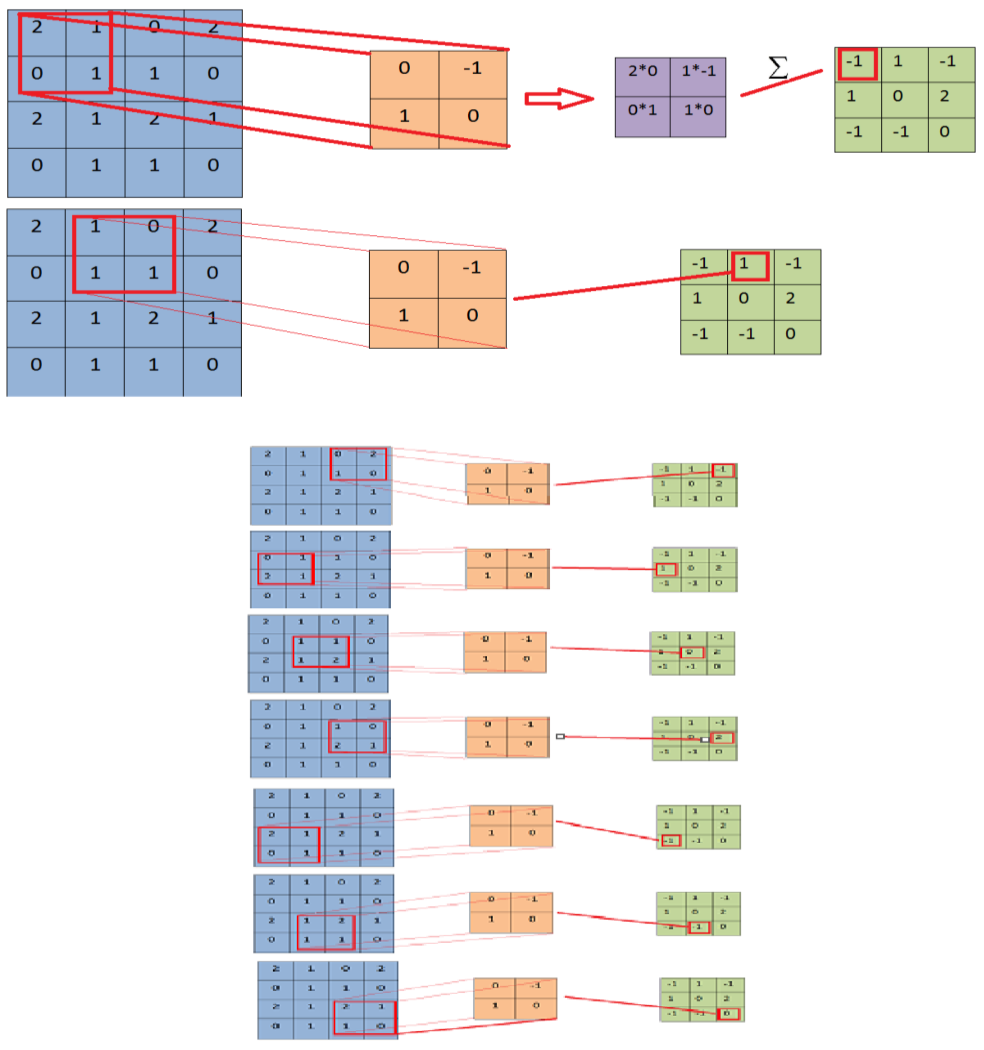
In the CNN architecture, the local receptive field (kernel or filter) is replicated across the entire visual field to form a feature map, which is known as convolution operation. The principle of weight sharing is known as these convolution operations share the same parameterization (weight vector and bias). The sharing of weights over convolution operation reduces the number of free variables, increasing the generalization performance of the network. Weights (kernels or filters) are initialized as random and will be learned to be edge, color, or specific patterns' detectors.

The convolution operation is expressed as

$$y^j = f_{acti}\left(b^j + \sum_i w^{ij} * x^i\right) \quad (1)$$

Where x^i and y^j are the i-th input feature map and the j-th output feature map, respectively. $w^{ij}$is the weights of the convolution filter. $*$denotes the convolution operation. $b^j$and $f_{acti}(\cdot)$ is the bias and activation function of the j-th output feature map, respectively. In addition, suppose the input feature map size is $i1 \times i2$, and filter size is $f1 \times f2$, the output feature map size is thus $o1 = i1 - f1 + 1$ and $o2 = i2 - f2 + 1$ due to the board effects.By Equ.1, A feature map is obtained by repeated application of the function across sub-regions of the entire input map, in other words, by convolution of the input map with a convolution filter, adding a bias term and then applying a non-linear activation function. The bias term is there only to ensure the predicted output will be unbiased. For example, if the input has a dynamic (range) that goes from -1 to $+1$ and the output is simply a translation of the input by $+3$, a neural net with a bias term will simply have the bias neuron with a non-zero weight while the others will be zero. The non-linear activation function $f_{acti}(\bullet)$ is regarded as a single layer as opposite to a function intergraded in convolution layer, and will be introduced later.

Convolution Demo.



Exercise 1

In given file directory ...\convolution_operation_exercise, you are required to calculate the convolution results of cat_input.jpg using six kernels separately. Then. visualizing the result using imagesc() function in MATLAB.

Step 1. Read the input image and format it into gray channel, double data type, and range of [0 1]

For example:

```
im = imread('cat_input.jpg');
im = rgb2gray(im);
im = double(im);
im = im/255;
```

Step2. Read the kernel image and format it into gray channel, double data type, range of [-1 1]

For example:

```
kernel = imread('kernel_1.jpg');
kernel = rgb2gray(kernel);
kernel = double(kernel);
kernel = (kernel/255-0.5)*2;
```

Step3. Calculate the convolution result using convn() function in MATLAB.

For example:

```
output = convn(input, kernel, 'valid');
```

Step4. Visualize the input matrix, kernel matrix, and the convolution output matrix usingimagesc() function in MATLAB.

For example:

```
imagesc(im);
figure;
imagesc(kernel);
figure;
imagesc(output);
```

Questions

1. Using the six different kernels, you can calculate six different convolution results, visualization the results and have a look at the differences.
2. Give an input matrix with size of [axb], and a kernel size of [mxn], what isthe size of convolution result matrix?

3.2 Pooling

Due to the replication of weights in a convolutional layer, the detected features may still sensitive to the precise positions of pattern, which is harmful to the performance if it is followed by a classifying procedure next. To solve such problem, a reasonable method is pooling the feature, which has three major advantage: (i)Pooling in an efficient form of dimensionality reduction, which decreases feature maps' resolution and reduces computation for upper layers in CNN. It throws away unnecessary information and only preserves the most critical information, (ii)Pooling makes activations in a neural network less sensitive to the specific structure of the neural network. And it makes a network less sensitive to the exact location of the pixels, which results a form of translation invariance, (iii) Pooling summarizes the output of multiple neurons from convolutional layers with the essence of taking nearby feature detectors and forming local or global 'bag of features'.

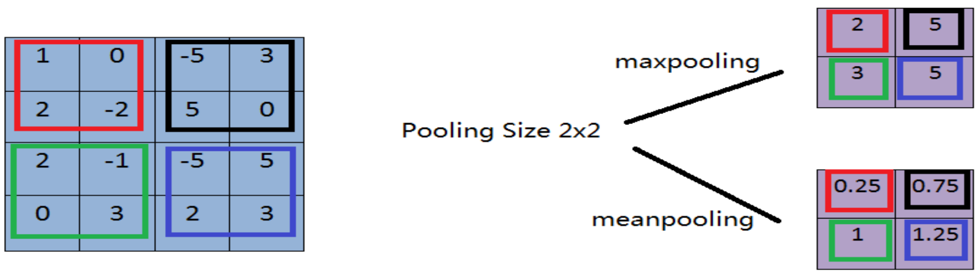
Typical pooling functions are average and maximum, generally with the name of average-pooling (subsampling, downsampling, meanpooling) and max-pooling layers, as in Equ.2 and Equ.3, respectively.

$$y^i{}_{m,n} = \frac{1}{s^2} \sum_{0\le\lambda,\mu<s} x^i{}_{m\cdot s+\lambda, n\cdot s+\mu} \tag{2}$$

$$y^i{}_{m,n} = \max_{0\le\lambda,\mu<s} x^i{}_{m\cdot s+\lambda, n\cdot s+\mu} \tag{3}$$

where each neuron in the i-th output map y^i pools over an $s \times s$ non-overlapping local region in the i-th input map x^i. Average pooling as the name suggest basically takes the arithmetic mean of the elements in each pooling region while Max-pooling selects the largest element from the input.

Pooling Demos



Exercise 2

In given file directory ...\pooling_operation_exercise, you are required to calculate the pooling results of head_input.jpg using maxpooling and meanpoolingseparately. Then. visualizing the result using imshow() function in MATLAB.

Step 1. Read the input image and format it into gray channel, double data type, and range of [0 1]

For example:

```
im = imread('head_input.jpg');
im = rgb2gray(im);
im = double(im);
im = im/255;
```

Step 2.Resize the input matrix and make sure its size can be divided exactly by scale. If the size of input is 321x426 and the scale is 2. Then we need to resize 321 to 320 or 322, so that it can be divided exactly by 2.

For example:

```
im = imresize(im, [320 426]);
```

Step 3. MaxPooling the input and show the output using imshow() in MATLAB

For example:

```
addpath…\pooling_operation_exercise\MaxPooling_function
[output,idx] = MaxPooling(input,[scale scale]);
imshow(output);
```

Step 4. MeanPooling the input and show the output using imshow() in MATLAB

For example:

```
addpath…\pooling_operation_exercise\MeanPooling_function
output = MeanPooling(input,scale);
imshow(output);
```

Questions

1. Calculate the pooling result using two pooling methods, and visualize the results and have a look at the differences.
2. Give an input matrix with size of [axb], and a pooling scale of m, what is the size of output matrix?