Intelligent Systems: Reasoning and Recognition

James L. Crowley

MOSIG M1 Winter Semester 2019-2020
Lessons 9 10 March 2020

Convolutional Neural Networks

Outline

Notation

x_d	A feature. An observed or measured value.
$\vec{X}$	A vector of D features.
D	The number of dimensions for the vector $\vec{X}$
$\{\vec{X}_m\}\ \{y_m\}$	Training samples for learning.
M	The number of training samples.
$a_j^{(l)}$	The activation output of the *j*th neuron of the *l*th layer.
$w_{ij}^{(l)}$	The weight from unit *i* of layer *l–1* to the unit *j* of layer *l*.
b_j^l	The bias for unit *j* of layer *l*.
η	A learning rate. Typically very small (0.01). Can be variable.
L	The number of layers in the network.
$\delta_m^{out} = \left(a_m^{(L)} - y_m\right)$	Output Error of the network for the *m*th training sample
$\delta_{j,m}^{(l)}$	Error for the jth neuron of layer *l*, for the *m*th training sample.
$\Delta w_{ij}^{(l)} = a_i^{(l-1)}\delta_j^{(l)}$	Update for weight from unit *i* of layer *l–1* to the unit *j* of layer *l*.
$\Delta b_j^{(l)} = \delta_j^{(l)}$	Update for bias for unit *j* of layer *l*.

Introduction

Key Equations

Feed Forward from Layer i to j:

$$a_j^{(l)} = f\left(\sum_{i=1}^{N^{(l-1)}} w_{ij}^{(l)} a_i^{(l-1)} + b_j^{(l)} \right)$$

Feed Forward from Layer j to k:

$$a_k^{(l+1)} = f\left(\sum_{j=1}^{N^{(l)}} w_{jk}^{(l+1)} a_j^{(l)} + b_k^{(l+1)} \right)$$

Output Error:

$$\delta_m^{out} = \left(a_m^{(L)} - y_m \right)$$

Back Propagation from Layer j to i:

$$\delta_{i,m}^{(l-1)} = \frac{\partial f(z_i^{(l-1)})}{\partial z_i^{(l-1)}} \sum_{j=1}^{N^{(l)}} w_{ij}^{(l)} \delta_{j,m}^{(l)}$$

Back Propagation from Layer k to j:

$$\delta_{j,m}^{(l)} = \frac{\partial f(z_j^{(l)})}{\partial z_j^{(l)}} \sum_{k=1}^{N^{(l+1)}} w_{jk}^{(l+1)} \delta_{k,m}^{(l+1)}$$

Weight and Bias Corrections for layer j:

$$\Delta w_{ij,m}^{(l)} = -a_i^{(l-1)} \delta_{j,m}^{(l)}$$

$$\Delta b_{j,m}^{(l)} = -\ \delta_{j,m}^{(l)}$$

Network Update Formulas:

$$w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} + \eta \cdot \Delta w_{ij,m}^{(l)}$$

$$b_j^{(l)} \leftarrow b_j^{(l)} + \eta \cdot \Delta b_{j,m}^{(l)}$$

Convolutional Neural Networks.

The classic approach to image analysis is to use mathematical properties of signal processing operations to design filters for image description. For example, an image derivative filter can be constructed by samplng a Gaussian derivative function $G_x(x,y,\sigma)$ over a finite range of integer values of x and y.

$$G_x(x,y,\sigma) = \frac{\partial G(x,y,\sigma)}{\partial x} = -\left(\frac{x}{\sigma^2}\right)\cdot e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

Typically x, y are integer values over a range of $R \le x, y \le R$ where $R = 3\sigma$.
An image derivative can then be computed by convolution of the filter $G_x(x,y,\sigma)$ with an image using the formula:

$$P_x(x,y) = P * G_{\mathbf{x}}(x,y,\sigma) = \sum_{u=-R}^{R}\sum_{v=-R}^{R} P(x-u,y-v)G_x(u,v,\sigma)$$

The filter $G_x(x,y,\sigma)$ is referred to as a “receptive field”. Convolutional Neural Networks (CNNs) replace the mathematical derivation of the receptive fields with filters learned from training data using back-propagation. The filters are learned locally, by considering every possible *NxN* image window as input to a network.

Fully connected Networks

A fully connected network is a network where each unit at layer *l+1* receives activations from all units at layer *l*.

If there are $N^{(l)}$ units at layer *l* and $N^{(l+1)}$ units are layer *l+1* then a fully connected network requires learning $N^{(l)}\cdot N^{(l+1)}$ parameters for layer *l*. Reliable learning requires that the number of data samples exceed the number of parameters. While this may be tractable for small examples, it quickly becomes excessive for practical problems, as found in computer vision or speech recognition.

For example, a typical image may have 1024 x 2048 = 2^{21} pixels. If we assume, say a 512 x 512 =2^{18} hidden units we have 2^{39} parameters to learn for a single class of image pattern, requiring more than 2^{39} training images. Clearly this is not practical (and, in any case not necessary).

A common solution is to perform learning using a limited size window, and to use all possible windows as training data. This leads to a technique where all possible, overlapping, image windows of size NxN provide training data to train the network.

We then use the same learned weights with every hidden cell. The resulting operation is equivalent to a "convolution" of the learned weights with the input signal and the learned weights are referred to as "receptive fields" in the neural network literature.

Local and Stationary Signals

Convolutional Neural Networks (CNNs) are used to interpret image and speech signals because both images and speech signals have two interesting theoretical properties: They are local and stationary.

1) Local. Local means that (most of) the information can be found within a limited sized neighborhood of the signal. In fact, image information tends to be multi-scale, but this can be easily accommodated by projecting the image onto a multi-scale pyramid. Such a representation is "local" at multiple scales, with low-resolution scales providing context for higher resolution. This can be referred to as "multi-local".

2) Stationary. A stationary signal is a random (unknown) signal whose joint probability density function does not change when shifted in time (speech) or space (image). Image and Speech signals tend to have stationary statistics. Thus the same processing can be applied to every possible (overlapping) window.

There are exceptions to both rules, but these can be handled with established techniques.

What Window Size?

What window size should be used for a feature in a Convolutional Neural Network? It is common for authors to use 3x3 or 5x5. There is no clear theory (so far) to explain this phenomena or predict window size as a function of number of layers. Most early researchers simply set up a python script to test a range of sizes and layers and experimentally discover which works best for a given data set. Most programmers used an existing network discovered with such a method.

Convolutional Neural Network for images

The Convolution Equation.

For a digital signal, *s(n)*, the equation for convolution of a digital filter, *g(n)* composed of *N* coefficients is:

$$(s * g)(n) = \sum_{m=1}^{N} g(m)s(n-m)$$

For image processing, the signal and filter are generally 2D: To avoid overloading the symbols x and y, we will refer to the image columns and rows as i and j. Thus the image is *P(i, j)*. We will extract a *WxH* window *s(u,v)* at each image position *(i, j)*

$$s(u,v) = P(i-u, j-v)$$

where *s(u, v)* is the image window and *g(u,v)* is the receptive field).
The formula for 2D convolution is then:

$$g * P(i,j) = \sum_{v=1}^{H}\sum_{u=1}^{W} g(u,v)P(i-u,i-v) = \sum_{v=1}^{H}\sum_{u=1}^{W} g(u,v)s(u,v)$$

Thus the value at each position *i,j* is the sum of the product of *g(u,v)* with a window placed at *i,j*. Note that a 2D convolution can easily be re-expressed as a 1D convolution by mapping successive rows of *g(u,v)* into 1 long column, *g(n)* with: $n = (v-1)\cdot H + u$

Convolutional Neural Networks

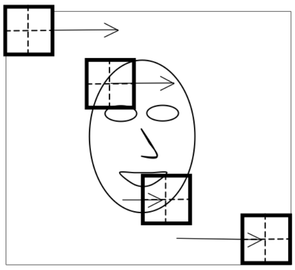
Convolutional Neural Networks (CNN) almost always out-perform systems using handcrafted descriptors for domain specific problems where large amounts of labeled training data are available.

At each layer of the network, a CNN describes neighborhood around each pixel as a vector of K features, computing by as a product with K receptive fields. The number of descriptors, K, is referred to as the "Depth" of the layer (number of channels or number of features at each sample). For example, an RGB image is said to have a depth of 3. The 2D array of descriptors is sometimes called a feature "map".

Each descriptor is computed as a weighted sum of the pixels within an N x N window followed by a non-linear activation function.

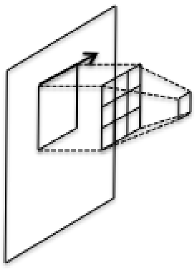
Let *P(i, j)* be a color image with C columns and R rows $\vec{P}(i,j) = \begin{pmatrix} R \\ G \\ B \end{pmatrix}(i,j)$

The first layer of the CNN will describe the image using a scanning-window technique, extracting a small *NxN* window for each pixel.



$X_{i,j}(u, v) = P(i\text{-}u, j\text{-}v)$

(assume pixels outside the image return as a zero). The use of *i–u* and *j–v* is rather than *i+u* and *j+v* is purely to assure equivalence with the classical signal processing operation of "convolution" in which the filter is "flipped" around x, y. In reality most implementations extract the window with *i+u* and *j+v*. Technically, in signal processing, this would be called a "cross correlation".

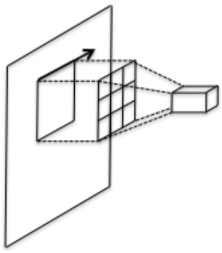


$$a(i,j) = f(z(i,j)) = f\left(\sum_{u,v} W(u,v)P(i-u,j-v) + b_k\right)$$

Multiple Receptive Fields at each Layer

A second innovation used in convolutional neural networks is to learn multiple *NxN* receptive fields at each layer. This is similar to common practice in computer vision.

The number of receptive fields, *k=1, K* is called the "depth" at that layer.



$$a_k(i,j) = f(z_k(i,j)) = f\left(\sum_{u,v} W_k\,(u,v)P(i-u,j-v) + b_k\right)$$

Here we show the activations for the first layer, but this is generalized for multiple layers.

For each *NxN* window, the CNN will compute the product with a vector of *K* receptive fields, $W_k(u,v)$ with a bias b_k.

$$z_k = \sum_{u,v} W_k(u,v)X_{i,j}(u,v) + b_k = \sum_{u,v} W_k(u,v)P(i-u, j-v) + b_k$$

The weighted sum is then processed with a non-linear activation function, *f()*, typically a sigmoid or a tanh or a relu of the sum of the product.

$$a_k = f(z_k) = f\left(\sum_{u,v} W_k(u,v)X_{i,j}(u,v) + b_k\right)$$

Because a vector of activations $\vec{a}_k = \begin{pmatrix} a_1 \\ \vdots \\ a_K \end{pmatrix}$ is computed for each image position, this should properly be written as $a_k(i,j) = f(z_k) = f\left(\sum_{u,v} W_k(u,v)X_{i,j}(u,v) + b_k\right)$

The result is a "feature map" of k features at each position $a_k(i,j)$, with k values at each image position *(i,j)*.

The receptive fields, $W_k(u,v)$ can be learned using back-propagation, from a training set where each window is labeled with a target class, using an "indicator" image *y(i,j)*. For multiple target classes, the indicator image can be represented as a vector image, $\vec{y}(i,j)$. More classically, *y(i, j)* is a binary image with 1 at each location that contains the target class and 0 elsewhere.

Hyper-parameters

CNNs are typically configured with a number of “hyper-parameters”:

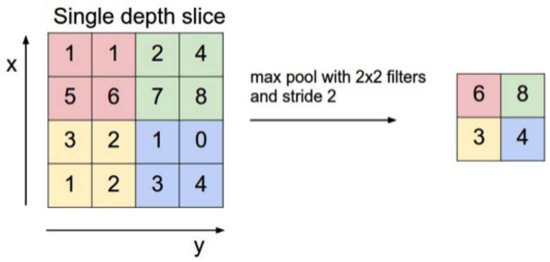
Depth: This is the number D of receptive fields for each position in the feature map. For a color image, the first layer depth at layer 0 would be D=3. If described with 32 image descriptors, the depth would be D=32 at layer 1.

Stride: Stride is the step size, S, between window positions. By default it generally 1, but for larger windows, it is possible define larger step sizes.

Spatial Extent: This is the size of the filter, NxN.

Zero-Padding: Size of region at the border of the feature map that is filled with zeros in order to preserve the image size (typically N).

Pooling



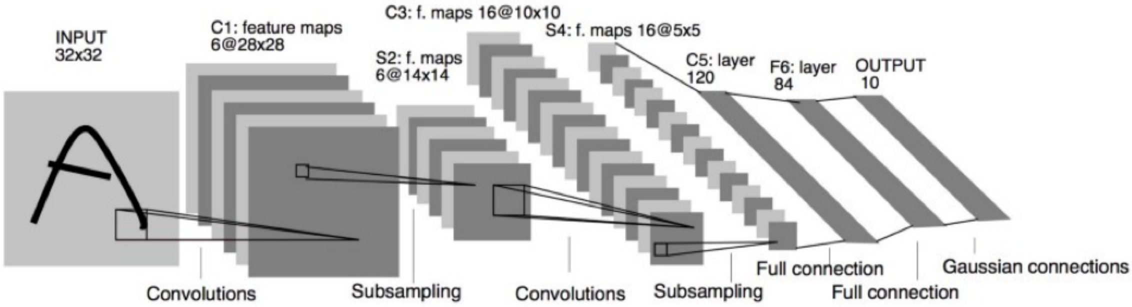
Pooling is a form of down-sampling that partitions the image into non-overlapping regions and computes a representative value for each region. The feature map is partitioned into small non-overlapping rectangles, typically of size 2x2 or 4x4, and a single value it determined for each rectangle. The most common pooling operators are average and max. Median is also sometimes used. The earliest architectures used average, creating a form of multi-resolution pyramid. Max pooling was soon shown to work better.

Classic CNN Architectures

CNN network architectures continues to be a very popular area of research with innovations published nearly every month. Very often, researchers will run a script to automatically compare results for variations in hyper-parameters for a benchmark data set. The winning set of hyper-parameters (number of layers, depth etc) define a new architecture ! Here are some of the earliest and best known architectures.

LeNet5

One of the very deep learning architectures to outperform classical machine learning was LeNet5. Towards the end of the first wave of popularity of Neural Networks in the late 80s, several researchers began experimenting with networks composed of more than 3 layers. From 1988, Yann LeCunn began experimenting with a series of such architectures, referred to as LeNet, for the task of recognizing handwritten characters.



The LeNet5 architecture (1994)

In 1994 Yann LeCunn showed that LeNet5 provided the best performance for written character recognition. Because processing power, memory and training data were very limited at that time, many of the innovations in LeNet5 concerned methods to reduce parameters and computing without degrading performance.

Many of the insights of LeNet5 continued to be relevant as more training data, and additional computing power enabled larger and deeper networks, because they allowed more effective performance for a given amount of training data and parameters.

Recall that, generally, the amount of training required for a network depends on the number of parameters to be trained. In addition, for a given network, performance increases with additional training. Thus any technique that gives equivalent performance with fewer parameters will scale to larger networks.

LeNet5 is composed of multiple repetitions of 3 operations: Convolution, Pooling, Non-linearity. Convolution windows were of size 5x5 with a stride of 1, no zero padding and a depth of 6. That is 6 receptive fields are learned for each pixel in the first layer. Using 5x5 filters without zero padding reduced the input window of 32 x 32 pixels to a layer of composed of 6 sets of 28 x 28 units. A Sigmoid was used for the activation function. Pooling was performed as a spatial averaging over 2x2 windows giving a second layer of 6 x 14 x 14.

This was then convolved with 16 5x5 receptive field, yielding a layer with 16 x 10x10 units. Average pooling over 2x2 windows reduced this to a layer of 16x5x5 units. These were then fed to two fully connected layers and then smoothed with a Gaussian filter to produce 10 output units, one for each possible digit.

Despite the experimental success, LeCunn found it very difficult to publish his results in the computer vision and machine learning literatures, which were more concerned with multi-camera geometry and Bayesian approaches to recognition. The situation began to change around 2010, driven by the availability of GPUs, and planetary scale data (continued exponential growth of the World Wide Web). In addition computer vision and machine learning were increasingly organized around open competitions for Performance Evaluation on benchmark data sets.

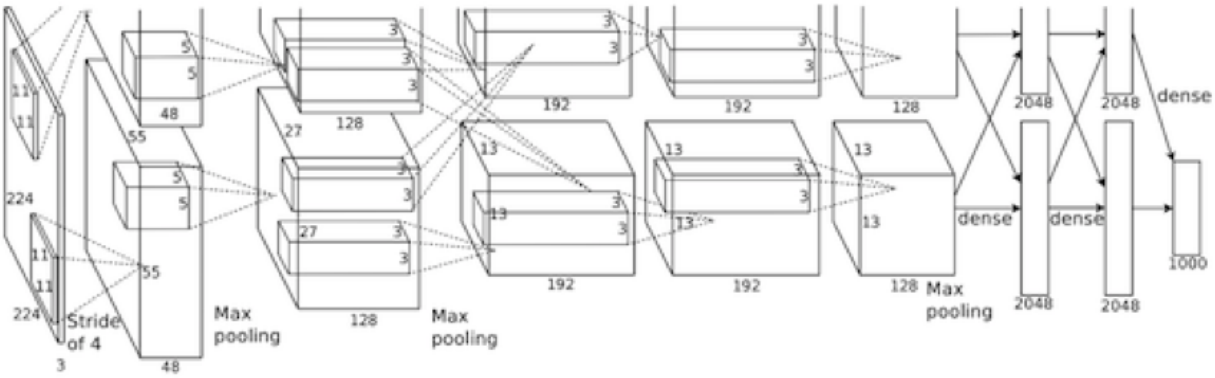
AlexNet

A classic, hard challenge at the time was the ImageNet competition. ImageNet is a large visual database designed for use in visual object recognition software research. The database was presented for the first time as a poster at the 2009 CVPR by researchers from Princeton University. Since 2010, the ImageNet project runs an annual software contest, the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), where software programs compete to correctly classify and detect objects and scenes.

ImageNet crowd-sources the annotation process. Currently more than 14 million images have been hand-annotated by the project to indicate what objects are pictured and in at least one million of the images, bounding boxes are also provided. Image-layer annotations indicate the presence or absence of an object class in an image. Object-layer annotations provide a bounding box around the (visible part of the) indicated object.

Initial champions were statistical recognition techniques using techniques such as SIFT and HoG. However, in 2012, Alex Krizhevsky won the competition by dramatically large margins, using a technique named AlexNet.

AlexNet, is a deeper and larger variation of LeNet5.



AlexNet Architecture (2010)

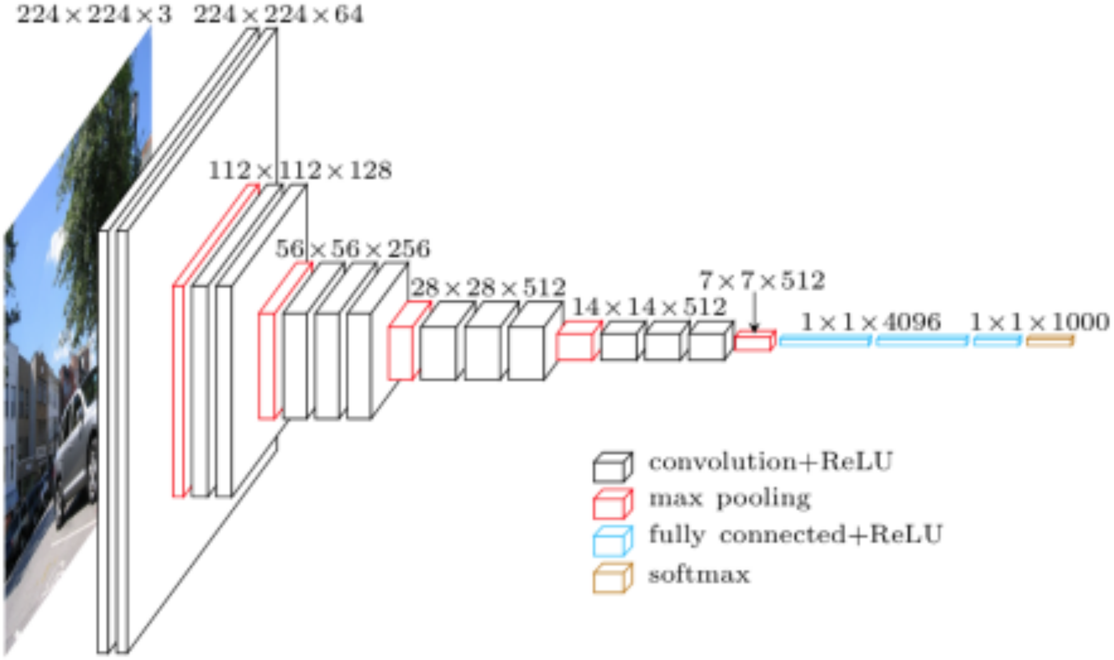
Innovations in AlexNEt include:

1. The use of relu instead of sigmoid or tanh. Relus provided a 6 times speed up with the same accuracy, allowing more training.
2. A technique called “dropout” in which randomly chosen units are temporarily removed during learning. This regularizes the network preventing over-fitting to training data.
3. Overlap pooling, in which pooling is performed on overlapping windows.

The architecture is composed of 5 convolutional layers followed by 3 fully connected layers. Relu is used after each convolution and in each fully connected layer. The input image size of 224 x 224 is dictated by the number of layers in the architecture. Larger images are generally texture mapped to this size.

A good implementation can be found in PyTorch. The network has 62.3 million parameters, and needs 1.1 billion computations in a forward pass. The convolution layers account for 6% of all the parameters, and consume 95% of the computation. The network is commonly trainined in 90 epochs, with a learning rate 0.01, momentum 0.9 and weight decay 0.0005. The learning rate is divided by 10 once the accuracy reaches a plateau.

VGG - Visual Geometry Group



The VGG Architecture (2014)

In 2014, Karen Simonyan and Andrew Zisserman of the Visual Geometry Group at the Univ of Oxford demonstrated a series of networks referred to as VGG. An important innovation was the use of very many small (3x3) convolutional receptive fields. The also introduced the idea of a 1x1 convolutional filter.

For a layer with a depth of D receptive fields, a 1x1 convolution performs a weighted sum of the D features, followed by non-linear activation. The weights can be learned with back-propagation.

A stack of convolutional layers is followed by three Fully-Connected layers: the first two have 4096 channels each, the third performs classification and thus contains one channel for each class (1000 channels for ILSVRC). The final layer is the soft-max layer. The configuration of the fully connected layers is the same in all networks. All layers use Relu activation.