

# Intelligent Systems: Reasoning and Recognition

James L. Crowley

Ensimag 2  
Lesson 11

Winter Semester 2021  
17 March 2021

## Convolutional Neural Networks

### Outline

|   |    |
|---|----|
| Notation .....                                    | 2  |
| The Mammalian Visual Cortex .....                 | 3  |
| Receptive Fields in the Visual Cortex .....       | 3  |
| Convolutional Neural Networks .....               | 6  |
| Fully-Connected Networks .....                    | 6  |
| Early Convolutional Neural Networks: LeNet5 ..... | 6  |
| Multiple Receptive Fields at each Layer .....     | 9  |
| Classic CNN Architectures .....                   | 10 |
| Benchmark Data Sets for Object Detection .....    | 10 |
| AlexNet .....                                     | 12 |
| CNN Hyper-parameters .....                        | 13 |
| Pooling .....                                     | 13 |
| VGG - Visual Geometry Group .....                 | 14 |
| A Keras example of a simple CNN .....             | 15 |

## 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 $j^{\text{th}}$ neuron of the $l^{\text{th}}$ 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 $l$ .   |
| $\eta$   | A learning rate. Typically very small (0.01). Can be variable.                               |
| $L$  | The number of layers in the network.   |
| $\delta_m^{\text{out}} = (a_m^{(L)} - y_m)$        | Output Error of the network for the $m^{\text{th}}$ training sample                          |
| $\delta_j^{(l)}$                                   | Error for the $j^{\text{th}}$ neuron of layer $l$ , for the $m^{\text{th}}$ 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 $l$ .  |

Convolution Equation: Convolution of a digital signal,  $s(n)$  with a digital filter,  $f(n)$  composed of  $N$  coefficients. Note that the "\*" symbol is reserved for convolution and is not used for multiplication.

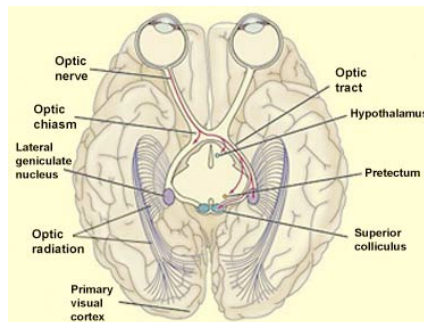
$$(f * s)(n) = \sum_{m=1}^N f(m)s(n-m)$$

2D convolution of a

$$f * P(i, j) = \sum_{v=1}^N \sum_{u=1}^N f(u, v)P(i-u, j-v)$$

## The Mammalian Visual Cortex

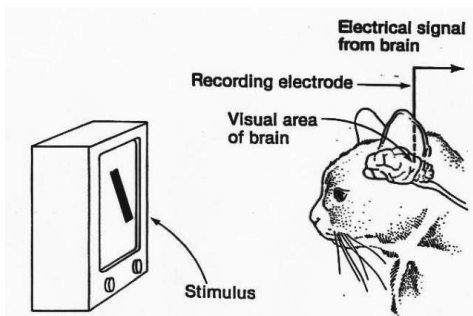
The Visual Cortex of mammals is composed of multiple layers of retinotopic maps.



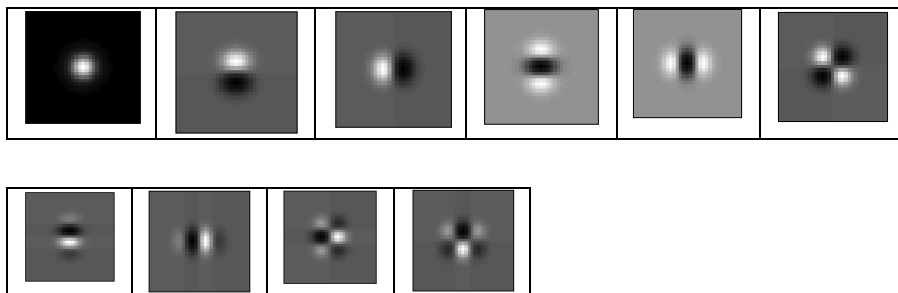
Each map is an image of the retina projected onto (convolved with) a receptive field at different scales and different orientations.

### Receptive Fields in the Visual Cortex

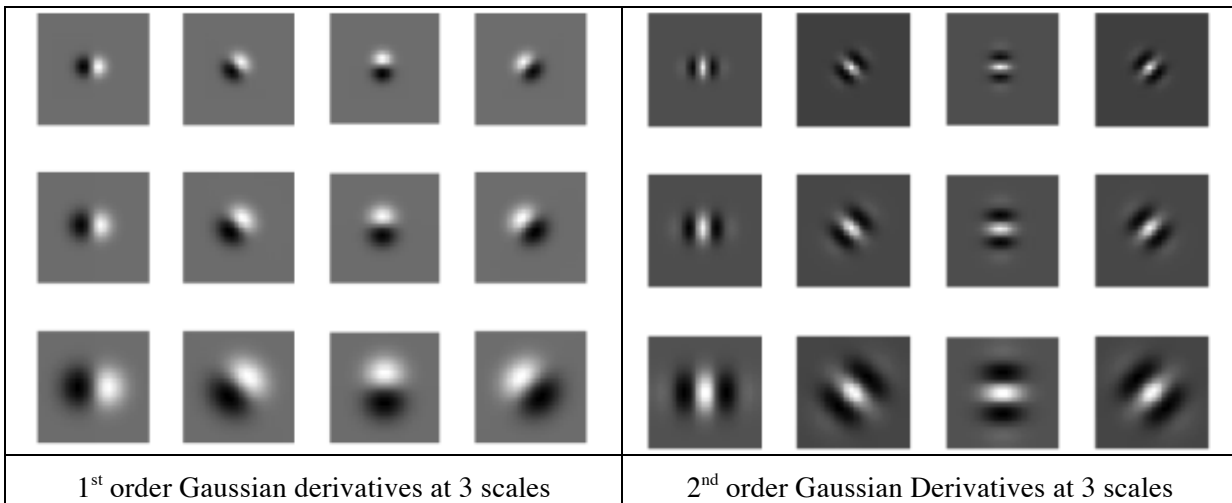
In 1968, Hubel and Wiesel probed the visual cortex of a cat with electrodes and found layers of cells that responded to local patterns of stimulation. They discovered that the visual cortex is composed of a series of layers. Each layer is a map of the retina filtered by a “receptive field” that respond to a certain pattern over a narrow range of sizes and orientations.



The patterns at the lowest level look like these:



Each layer is a specific pattern at a specific orientation and scale. It was discovered that these patterns could be modeled as Gabor function at different scales and orientations.



This figure shows first and second order Gaussian derivative features from 3 scales, computed using sums and differences of adjacent samples in a half-octave Gaussian pyramid.

The first level receptive fields were found to be local filters that pass a narrow range of spatial frequencies. The first layer receptive fields were found to be modeled by Gabor functions (Gaussian modulated by a Cosine plus an imaginary Sin).

The first layer receptive fields can also be modeled multi-scale derivatives of Gaussians functions. These two representations are very similar. However, Gabor functions can be very expensive to convolve with an image. Multi-scale Gaussian derivatives can convolved very efficiently because of a number mathematical properties that are beyond the scope of this course.

As they moved up the visual cortex, Hubel and Wiesel found that these patters were combined to form more complex patterns, such as corners, bars, crosses, etc. These were named "complex" receptive field, and are similar to receptive field patterns learned by convolutional neural networks.

Inspired by the results of Hubel and Wiesel, computer vision researchers have explored the use of image description with receptive fields since the 1980's, although this was never a dominant paradigm in computer vision, in part the computational cost exceeded the computing power of early computers. Most such research explored image description using image derivatives computed with receptive fields.

A Receptive Field for computing image derivative filter can be constructed by sampling 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  $-R \leq x, y \leq R$  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:

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

Convolutional Neural Networks (CNNs) replace the mathematical derivation of receptive fields with filters learned from training data using back-propagation. The filters are learned locally, by considering every possible  $N \times N$  image window as input to a network.

## Convolutional Neural Networks.

### Fully-Connected Networks

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. Most experiments explored fully connected networks, where each unit at layer  $l+1$  receives activations from all units at layer  $l$ . The result is a very rapid growth in the number of parameters to learn, even for simple problems.

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  $1024 \times 2048 = 2^{21}$  pixels. If we assume, say a  $512 \times 512 = 2^{18}$  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).

### Early Convolutional Neural Networks: LeNet5

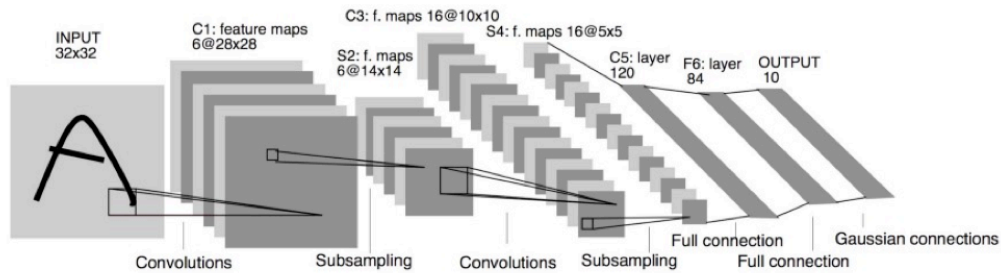
From 1988, Yann LeCunn began experimenting with a series of multi-layer architectures, referred to as LeNet, for the task of recognizing handwritten characters.

LeCunn's first insight was to limit each neural unit to a connection to small window of units in the previous level, and to learn the same weights for all units. This leads to a technique where all possible, overlapping, image windows of size  $N \times N$  provide training data to train a small number of parameters for a receptive fields network. The network then uses the same learned weights with every hidden cell. Recall that, generally, the amount of training required for a network depends on the number of parameters to be trained. Thus any technique that gives equivalent performance with fewer parameters will scale to larger networks.

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.

A second insight was to use several convolutional units in parallel to describe each window. This led to a map of features for each pixel with the number of units referred to as "depth".

A third insight was to reduce the resolution of the image by resampling while increasing the number of parallel receptive fields (depth) at each level. This can be illustrated with the LeNet5 architecture shown here:



The LeNet5 architecture (1994)

In 1994 Yann LeCun 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.

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. The output was then convolved with 16 5x5 receptive fields, 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, LeCun 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) and the emergence of challenged based research in computer vision. During this period, computer vision

and machine learning were increasingly organized around open competitions for Performance Evaluation on benchmark data sets.

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.

### The Convolution Equation

For a digital signal,  $s(n)$ , the equation for convolution of a Finite Impulse Response (FIR) digital filter,  $w(n)$  composed of  $N$  coefficients is:

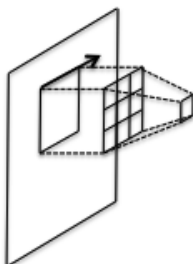
$$(w * s)(n) = \sum_{m=1}^N w(m)s(n - m)$$

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  $P(i, j)$ . The formula for 2D convolution of an  $N \times N$  filter  $w(i, j)$  with an image is:

$$w * P(i, j) = \sum_{v=1}^N \sum_{u=1}^N w(u, v)P(i - u, j - v)$$

The value at each position  $i, j$  is the sum of the product of a filter (kernel, or receptive field)  $w(u, v)$  with a neighborhood of the image placed at  $i, j$ . Note that a 2D convolution can easily be re-expressed as a 1D convolution by mapping successive rows of the  $N \times N$  filter  $w(u, v)$  into 1 long column with  $N^2$  coefficients,  $f(n)$ , using:  $n = (v - 1) \cdot N + u$

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 convolution, the filter is “flipped” around the center pixel. In reality, many implementations simply use  $i + u$  and  $j + v$ . Technically, in signal processing, this would be called a “cross-correlation”.

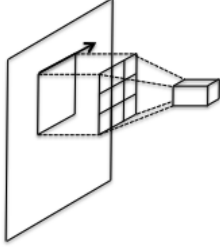


$$a(i, j) = f(z(i, j)) = f\left(\sum_{u, v}^N w(u, v)P(i - u, j - v) + b_k\right)$$



### Multiple Receptive Fields at each Layer

A second innovation was to learn multiple  $N \times N$  receptive fields at each layer, as was observed by David Hubel and Torsten Wiesel and used in Computer Vision. The number of receptive fields is called the “depth” at that layer. We will use the symbol  $d$  from 1 to  $D$  as an index for the depth (number of receptive fields) at each level.



$$a_d(i, j) = f(z_d(i, j)) = f\left(\sum_{u, v} W_d(u, v)P(i-u, j-v) + b_d\right)$$

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

$$z_d = \sum_{u, v} W_d(u, v)X_{i, j}(u, v) + b_d = \sum_{u, v} W_d(u, v)P(i-u, j-v) + b_d$$

The weighted sum is then processed with a non-linear activation function,  $f()$ , typically a relu or sigmoid 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  $\vec{a}_d = \begin{pmatrix} a_1 \\ \vdots \\ a_D \end{pmatrix}$  is computed for each image position, this

should properly be written as  $a_d(i, j) = f(z_d) = f\left(\sum_{u, v} W_d(u, v)X_{i, j}(u, v) + b_d\right)$

The result is a “feature map” of  $d$  features at each position  $a_d(i, j)$ , with  $d$  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.

## **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!

The emergence of the internet and the world-wide web made it possible to assemble massively large data sets of training data, and to issue global challenges for computer vision techniques to compete on these challenges. Many of the most famous CNN architectures have been established by winning large scale image classification challenges. The parameters of the challenge often explain the choice of parameters for the network, such as the size of the input image and the number of output categories.

The following are several key data sets that have influenced the evolution of the domain. Many of the popular architectures were designed specifically to address research challenges based on these data sets.

### **Benchmark Data Sets for Object Detection**

#### **Pascal VOC (Visual Object Challenge)**

PASCAL (Pattern Analysis, Statistical Modelling and Computational Learning) was a European Network of Excellence launched in 2003. In 2005, the network launched its first Visual Object Challenge (VOC), challenging research groups to recognize 4 classes (bicycles, cars, motorbikes, people) using a data set of 1578 images containing 2209 annotated objects. In 2006 the challenge was extended to 10 classes (bicycle, bus, car, cat, cow, dog, horse, motorbike, person, sheep) in 2618 images containing 4754 annotated objects, but including the images from flickr and from Microsoft Research Cambridge (MSRC) dataset. In 2007 this was extended 20 classes organized as a 2 level hierarchy in 9,963 images containing 24,640 annotated objects. The competition continued annually, with refinements to performance metrics and increasing number of labeled images. By 2012 the data set had 11,530 images containing 27,450 ROI annotated objects and 6,929 segmentations.

The Pascal VOC helped establish the popularity of such techniques as SIFT, HoG, SVM, Kernel Methods, and GMM for computer vision prior.

Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J. and Zisserman, A., The PASCAL Visual Object Classes (VOC) Challenge, *International Journal of Computer Vision*, 88(2), 303-338, 2010

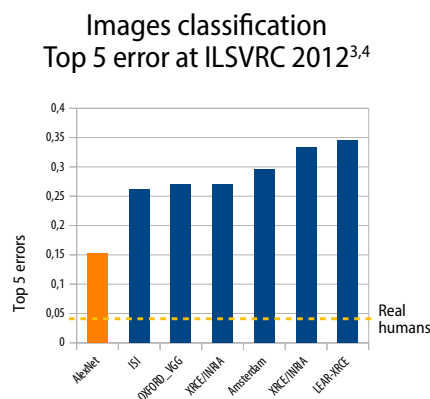
## ImageNet

ImageNet is an image database organized according to the nouns in the WordNet hierarchy. Each node of the WordNet hierarchy is depicted by hundreds of images in ImageNet. In 2006, Fei-Fei Li began working on the idea for ImageNet based on the word-database of WordNet and using many of its features. In 2007, Li was appointed as assistant professor at Princeton, where she assembled a team of researchers to work on ImageNet. They used the Amazon Mechanical Turk to help with the classification of images. They presented their database for the first time as a poster at the 2009 Conference on Computer Vision and Pattern Recognition (CVPR) in Florida.

Most state-of-the-art object detection networks pre-train on ImageNet and then rely on transfer learning to adapt the learned recognition system to a specific domain.

## ILSVRC

In 2010 Fei-Fei Li joined with the PASCAL VOC team to create a joint research challenge where research teams compete to achieve higher accuracy on several visual recognition tasks. The resulting annual competition is known as the ImageNet Large Scale Visual Recognition Challenge (ILSVRC). The ILSVRC uses a "trimmed" list of only 1000 image categories or "classes", including 90 of the 120 dog breeds classified by the full ImageNet schema. In 2010 and 2011, a good score for the ILSVRC top-5 classification error rate was 25%.

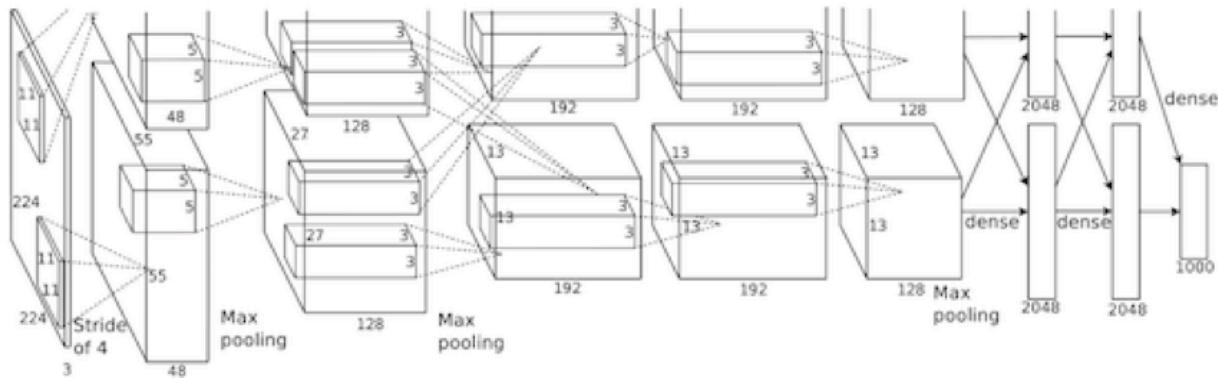


Initial champions were statistical recognition techniques using techniques such as SIFT and HoG. However, in 2012, Alex Krizhevsky won the competition with a deep convolutional neural net based on LeNet5 called AlexNet. AlexNet achieved an error rate of 16% (accuracy of 84%). This dramatic quantitative improvement marked

the start of the rapid shift to techniques based on Deep Learning using Neural Networks. By 2014, more than fifty institutions participated in the ILSVRC, almost exclusively with different forms of Network Architectures. In 2017, 29 of 38 competing teams demonstrated error rates less than 5% (better than 95% accuracy).

### 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 trained 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.

## CNN Hyper-parameters

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

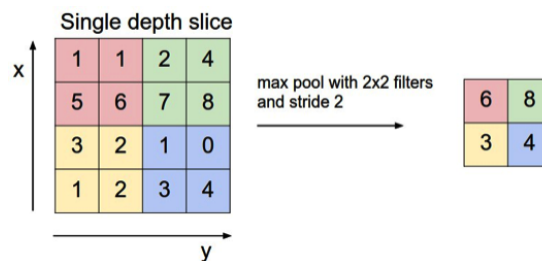
Spatial Extent: This is the size of the filter,  $N \times N$ . Early networks followed computer vision theory and used  $11 \times 11$  or  $9 \times 9$  filters. Experimentation has shown that  $3 \times 3$  filters can work well with multi-layer networks.

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. Some networks will use  $N \times N \times D$  receptive fields, including  $1 \times 1 \times D$ .

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.

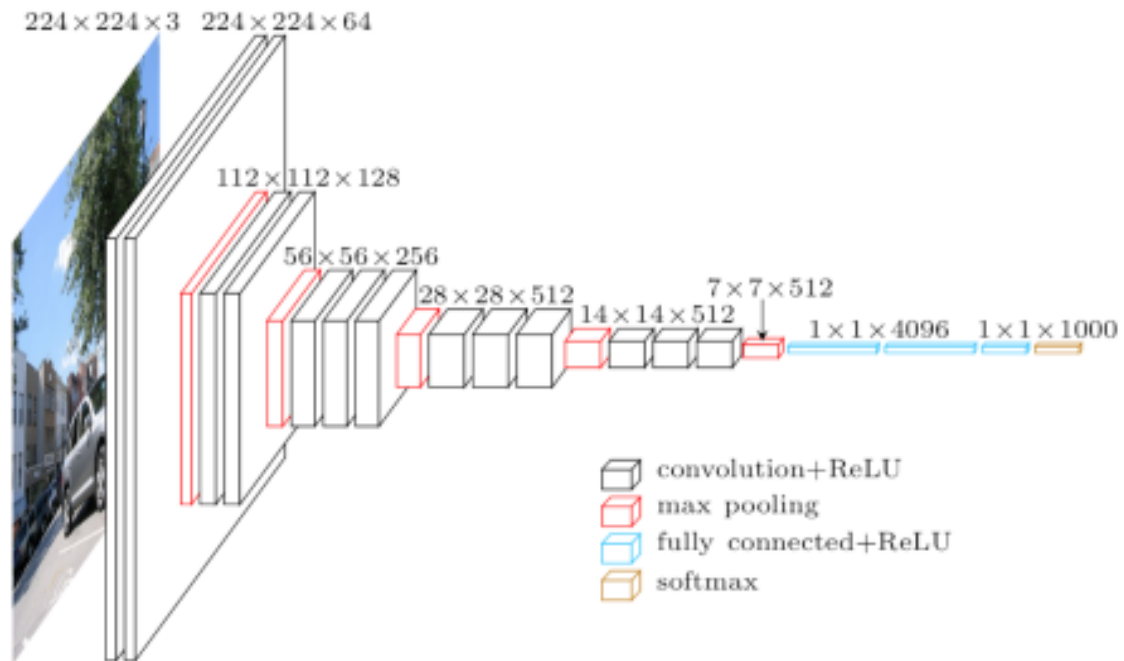
Zero-Padding: Size of region at the border of the feature map that is filled with zeros in order to preserve the image size (typically  $N$ ).

## 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  $2 \times 2$  or  $4 \times 4$ , and a single value is 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.

## 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. They 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.

## A Keras example of a simple CNN

The MNIST (Modified National Institute of Standards and Technology) database is a large collection of handwritten digits. The MNIST database contains 60,000 training images and 10,000 testing images. The database was created by "re-mixing" samples of digits from NIST's original datasets taken from American Census Bureau employees and American high school students. The black and white images from NIST were normalized to fit into a 28x28 pixel bounding box and anti-aliased, which introduced gray-scale levels.

The following is a simple Keras example of to detect MNIST digits, provided by Frank Cholet of Google. This example processes 28x28 pixel imaggettes with a convolutional layer of 32 3x3 filters using relu, followed by 2x2 max pooling, a convolutional layer of 64 3x3 filters, using relu, followed by 2x2 max pooling, a flatten layer, dropout of 0.5 and a fully connected layer.

```
model = keras.Sequential(  
    [  
        keras.Input(shape=input_shape),  
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),  
        layers.MaxPooling2D(pool_size=(2, 2)),  
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),  
        layers.MaxPooling2D(pool_size=(2, 2)),  
        layers.Flatten(),  
        layers.Dropout(0.5),  
        layers.Dense(num_classes, activation="softmax"),  
    ]  
)
```

|                                |                    |       |
|--------------------------------|--------------------|-------|
| conv2d (Conv2D)                | (None, 26, 26, 32) | 320   |
| max_pooling2d (MaxPooling2D)   | (None, 13, 13, 32) | 0     |
| conv2d_1 (Conv2D)              | (None, 11, 11, 64) | 18496 |
| max_pooling2d_1 (MaxPooling2D) | (None, 5, 5, 64)   | 0     |
| flatten (Flatten)              | (None, 1600)       | 0     |
| dropout (Dropout)              | (None, 1600)       | 0     |
| dense (Dense)                  | (None, 10)         | 16010 |