Fundamentals of Convolutional Neural Networks

Anthony Shara
Aditya Ganapathi
DoHyun Cheon
Larry Yan
Richard Shuai

December 11, 2020

1 Convolutional Neural Networks

Convolutional neural networks are a type of neural network specifically designed to operate on images, requiring far fewer weights than fully-connected networks. Like other neural networks, CNNs takes input and processes it through layers of neurons with weighted dot products and nonlinearities. However, CNNs use two additional types of layers—convolutional and pooling—that significantly reduces the weights and computation required by fully connected layers.

1.1 Motivation for CNNs

In normal neural networks, every neuron in a layer is connected with a learnable weight to every neuron in the next layer. These fully connected layers work well for smaller input sizes, but as the size of the input, and therefore the number of neurons, increases, the number of weights increases quadratically. A color image of size $w \times h$ would require $w \times h \times 3$ weights for each neuron in the first layer (the input consists of wh pixels, each with 3 color channels). For a reasonably sized color image with dimensions 128 by 128, this would result in 49152 weights per neuron. Training and classifying with these weights would be resource and time-intensive and prone to overfitting training data due to the excess of weights.

CNNs resolve this problem by making two main assumptions about classifying images: low-level features can be extracted from a localized region of an input, and weights useful for a feature in one part of an image are useful in another part. These assumptions enable the use of the convolutional layer and CNNs themselves.

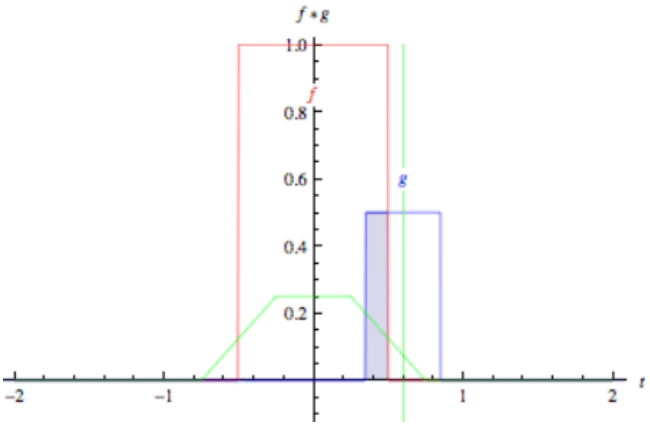


Figure 1: A convolution of functions f and g. A (flipped) copy of g in blue is moved across f in red, and their overlapping integral area at each position (vertical green line) producing the resulting convolution in green. Image adapted from Wolfram MathWorld [9].

2 Convolutions

A convolution is a mathematical operation that combines two functions into a third function that measures the degree of overlap between them. For continuous functions f and g, a convolution (represented with $*$) is defined as

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)\,d\tau$$

In the discrete case, a summation is used instead:

$$(f * g)[t] = \sum_{\tau=-\infty}^{\infty} f[\tau]g[t-\tau]$$

A convolution may be interpreted as taking one function, flipping it and shifting it across the other while measuring the amount of overlap (integral of their product / dot product) between them at each position. Recall cross correlation from 16A—a convolution is a closely related operation, with the only difference being that cross correlation does not flip a function before the product, while convolutions do.

Among other applications, convolutions are used to determine the response of Linear Time Invariant systems to an input signal, which will be elaborated upon in 16B.

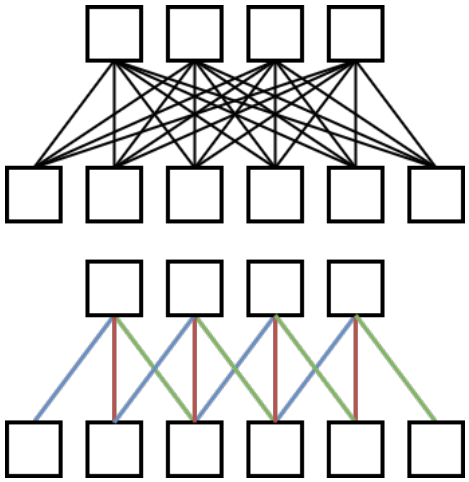


Figure 2: Top: a 1D fully connected layer with weights from every input neuron. Bottom: a 1D convolutional layer with filter size 3. Each neruon is only connected to 3 inputs and the same colored weights are all the same.

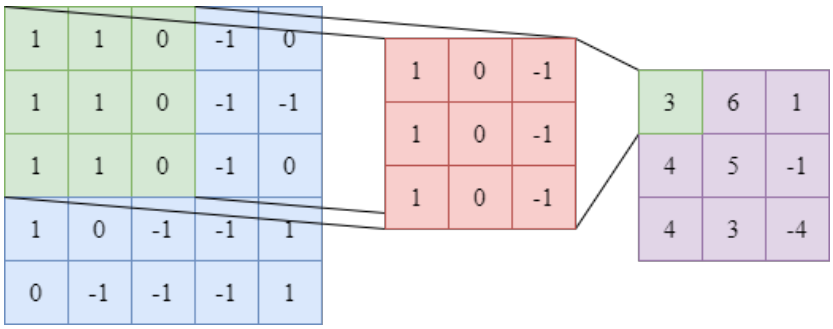


Figure 3: A convolution between part of the input (in green) and the filter (in red). Note that the filter produces a large positive output on areas similar to itself and large negatives on its inverse.

3 Convolutional layers

A convolutional layer consists of a set of learned filters (also known as a kernel) that are moved across the input. The filters are usually small spatially but extend over the full depth of the input (e.g. $3 \times 3 \times d$, $5 \times 5 \times d$). At every position, an elementwise dot product between the filter and the part of the input it's over is computed (a discrete convolution).[1] By moving the filter across the input, the activation of the filter over the input can be mapped. These filters extract features from the input, such as edges and color combinations at low levels, or faces and text at higher levels.

By assuming that features can be found locally, we can derive features from a relatively small filter, resulting in only the closest few input neurons affecting an output neuron. This reduces the number of weights for each output neuron from the number of neurons in the previous layer to just the size of the filter. Furthermore, convolutional layers use weight sharing—all the output neurons use the same set of weights for their corresponding inputs.[2] This allows for the use of a single filter across the entire input that can extract a feature from anywhere in the input. A convolutional layer usually has multiple filters that learn different features. The number of filters (and their output neurons) is also known as the depth of the layer.

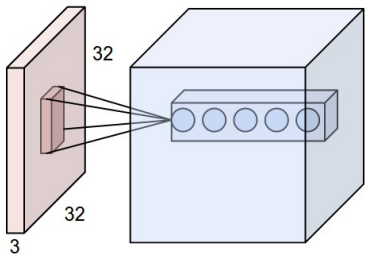


Figure 4: An example of a convolutional layer. Each neuron in the convolutional layer is connected to a localized area of the input. Note that the filter region extends through all 3 channels of the input. The convolutional layer has a depth of 5, with each neuron using a different filter on the same area of input. Image from Stanford CS231n [2].

[1] Note that in practice, most implementations of CNNs do not bother to flip the filters, making this operation closer to cross correlation than convolution. This note follows that precedent as well. Since the filters are learned, there is no practical difference between the two implementations. See if you can convince yourself why.

[2] There are also locally connected layers that do not use weight sharing. These are occasionally used when it is expected that a feature will be local and only found in a particular location, such as in centered facial recognition

3.1 Hyperparameters

Filter size (receptive field): The size of the filter, or how many input neurons are connected to each output neuron. Larger connectivities allow for more complex features to be extracted in that layer, but result in more computational complexity. If a layer does not use zero-padding, a filter size greater than 1. Even though a neuron may only look at a relatively small section of the preceding layer, multiple convolutional (and pooling) layers can have a large effective receptive field over the input. For example, two stacked 3×3 convolutional layers will have an effective receptive field of 5×5. This effect increases further with every subsequent layer or for dilated filters, which places spaces within the filter so it encompasses a greater area with few weights.

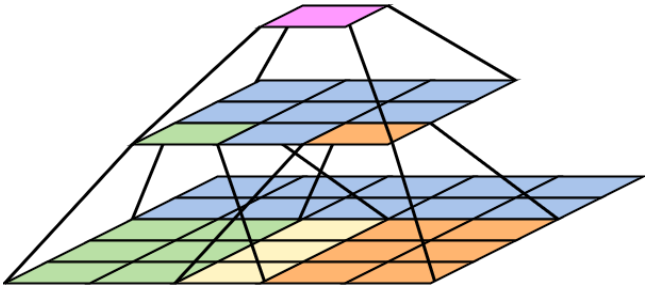


Figure 5: The neuron in the second layer is connected directly to a 3 by 3 section of the first layer. Each of those neurons is connected to a 3 by 3 section of the input, resulting in the second layer neuron having an effective receptive field of 5 by 5.

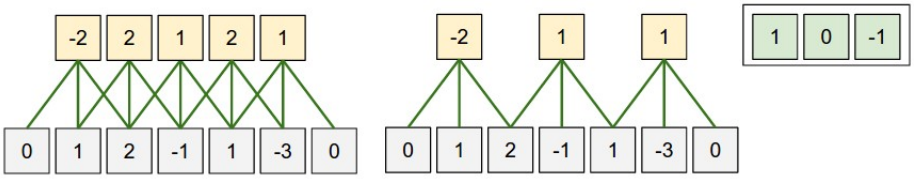


Figure 6: At left, a convolutional layer with filter size 3 and a stride of 1, with zero-padding of 1. In the center, the filter size and zero-padding are the same, but the stride is 2. The filter used for the convolution is at right. Image from Stanford CS231n [2].

Stride: How far the filter is moved across the input. A stride of 1 moves the filter one unit between every measurement, while a stride of 2 moves the filter by two units, and so on. A stride above 1 will reduce the size of the output layer, at the cost of possible information loss, as potential features may be present and

skipped by the large stride (because of this, strides tend to be 1 or 2). Usually, a larger stride is used with larger filter sizes to balance computational costs.

Zero-padding: The number of zeros added to the borders of the input. Without zero-padding, a filter with size greater than 1 will result in a smaller output size, as the filter is unable to move past the edge of the input. Zero-padding allows for computations at the edges of the filter to occur and provides extra rows so that a stride can fit over the input evenly. The equation $(W - F + 2P)/S + 1$, where F is the filter size, P is the padding on each edge, S is the stride, and W is the width (same must hold for the height) must hold so that the output has integer dimensions.

Depth: The number of different filters attempting to learn features. A larger depth enables more information to be learned, but requires another set of weights to be learned and provides more input weights for the next layer, incurring computational costs.

4 Pooling layers

A pooling layer reduces the size of the input layer, producing an output with fewer neurons and computations. Like a convolutional layer, a pooling layer also has a filter size and a stride, but instead of performing a dot product, a pooling layer aggregates its inputs and outputs with either an average or, more commonly, max value. This allows for most information to be transferred while reducing the size of the layer. Pooling layers usually use a stride of 2 and size of 2, as larger filters lose too much information to be useful and a smaller stride or filter wouldn't decrease the size of the next layer. Through successive pooling layers interspersed among the convolutional layers, the size of the input becomes much more manageable.[3]

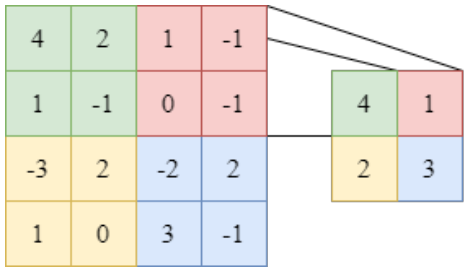


Figure 7: A max pooling layer with stride 2 and filter size 2.

[3]In addition to this spatial pooling, there is also cross-channel pooling, which samples from different depths and pools them, resulting in a shallower output.

5 Fully connected layers:

At the end of their computation, CNNs usually have one or more fully-connected layers[4] that use the features from previous layers to classify the input images. These function in the same way as those in other neural networks. Just like other neural networks, CNNs use backpropagation to compute gradients for each layer, allowing them to learn weights efficiently.

Layer (type)	Output Shape	Param #
CONV1 (Conv2D)	(None, 32, 32, 32)	7808
POOL1 (MaxPooling2D)	(None, 16, 16, 32)	0
CONV2 (Conv2D)	(None, 16, 16, 32)	9248
POOL2 (MaxPooling2D)	(None, 8, 8, 32)	0
flatten_11 (Flatten)	(None, 2048)	0
FC1 (Dense)	(None, 128)	262272
FC2 (Dense)	(None, 10)	1290
SOFTMAX (Activation)	(None, 10)	0

Total params: 280,618
Trainable params: 280,618
Non-trainable params: 0

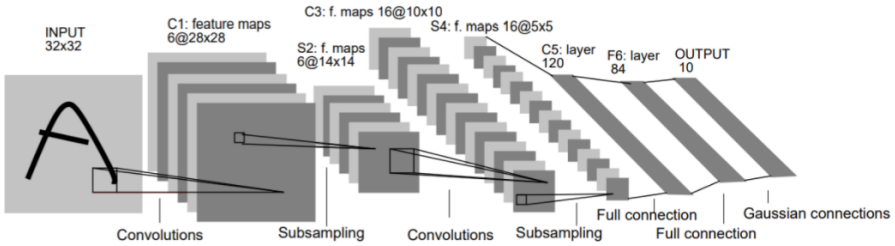
An important note is that most of the parameters in a CNN is contained in the first fully connected layer. Due to the large number of parameters FC layers require, many modern CNN's replace these layers.

6 CNN architectures

There are a few significant architectures that were developed. All of these besides LeNet-5 were submitted to the ImageNet challenge, a competition on image classification. Each contributed a significant development usually leading to a 1st place finish.

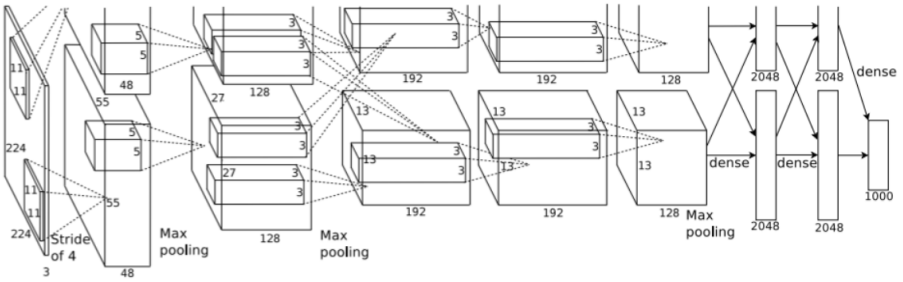
[4]Some modern architectures have replaced fully connected layers with convolutional layers that perform the same operations.

6.1 LeNet-5 (LeCun et al, 1998)



LeNet-5 is one of the earliest Convolutional Neural Networks. It consists of 2 convolutional and pooling layers, followed by 3 fully connected layers, hence the name LeNet-5. It was developed to identify handwritten digits.

6.2 AlexNet (Krizhevsky et al, 2012)

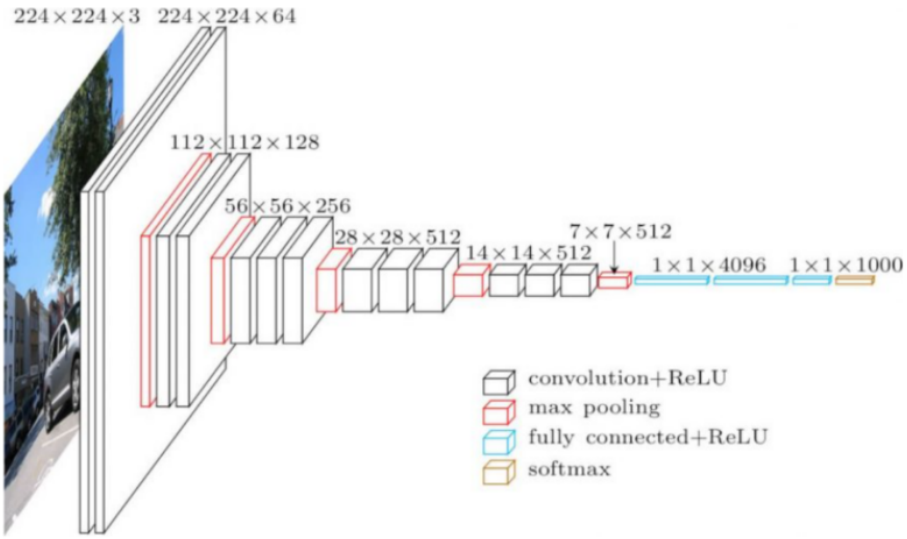


The ImageNet 2012 winner AlexNet popularized Convolutional Neural Networks in Computer Vision. The network was deeper and bigger than LeNet-5, and consisted of multiple convolutional layers stacked on each other, although it was common for networks to immediately have a pooling layer after a convolutional layer.

Key Points:

- Usage of ReLU, a non-saturating function, over the usage, of at then common, tanh and sigmoid activation functions.
- Trained over 2 GPU's.
- Data Augmentation: Extracted 224x224 patches, which resulted in 2048x more training points. During testing, extracted the 4 corner patches and the center patch, including their reflections, resulting in total 10 patches.
- Data Augmentation: Altered the intensity of the RGB color channels
- Dropout in the first two fully connected layers to prevent overfitting, which also halves the number of iterations required to converge.

6.3 VGGNet (Simonyan & Zisserman, 2014)



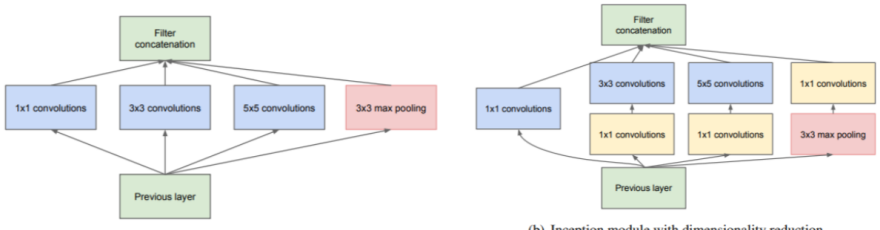
VGG16 Architecture

The ImageNet 2014 runner-up VGGNet only utilized 3x3 convolution filters.

Reasons to use a 3x3 filter:

- A 3x3 filter still captures the notion of left, right, up, down, and center.
- Applying 3 3x3 filters is equivalent to applying 1 7x7 filter. Applying 1 3x3 filter to a 7x7 image results in a 5x5 matrix. Applying 2 more 3x3 filters to a 5x5 matrix results in a 1x1 matrix.
- 3 3x3 filters requires 27 weights while 1 7x7 filter requires 49 weights. Using only 3x3 filters uses 45% fewer weights.
- Using small filters results in more ReLU layers, which makes the network more discriminative.

6.4 GoogLeNet ("Inception") (Szegedy et al, 2014)



(a) Inception module, naïve version

(b) Inception module with dimensionality reduction

Figure 2: Inception module

The winner of the 2014 ImageNet Contest, GoogLeNet included the development

of the Inception Module, dramatically reducing the number of parameters (6.6 million compared to AlexNet's 60 million).

Key Points:

- Consists of 22 layers, deeper than previous networks.
- Built without any fully connected layers, making it more computationally efficient.
- Inception Modules: The naive implementation runs convolutional layers in parallel then combines all of them. However, this method is computationally inefficient. The resulting inception module includes a dimension reduction, allowing networks to stack inception modules without the cost of computational efficiency.
- Applies average pooling instead of fully connected layers, eliminating a large amount of parameters.

6.5 ResNet (He et al, 2015)

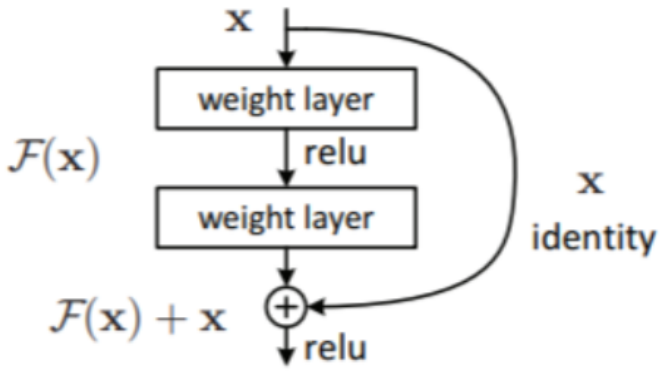


Figure 2. Residual learning: a building block.

The winner of the 2015 ImageNet Contest, ResNet introduces Residual Blocks, allowing for networks to reach new depths (ResNet is 152 layers deep) while still converging.

Network Bottleneck: Shallow vs Deep Networks:

A common problem networks faced was depth: We know that neural networks can approximate any function given enough layers, but at some depth, networks become too inaccurate. Additionally, due to some problems such as vanishing gradient (the effect on gradients from previous layers get smaller as we go deeper in the network), deep networks can struggle to learn easy functions, such as the identity function, while shallow networks succeed in doing so.

Residual Blocks:

Residual Blocks, as seen above, tries to learn the true function but includes an identity connection from x (also known as a skip connection).

Key Points:

- If the desired learning function is H(x), the layers learn the residual F(x) := H(x) - x, which results in an output of F(x) + x = H(x)
- Research has observed that learning the residual is easier.
- By having the skip connection, networks can easily learn functions like the identity function by simply setting the residual equal to zero.
- By using back propagation, we see that these skip connections allow initial layers to affect later layers, ridding networks of the vanishing gradient problem.

In general, it is difficult to find the optimal number of layers. Including skip connections allows our networks to be dynamic: not all layers have to contribute to our outcome, which lets the network decide the number of layers.

References

[1] Listgarten, Jennifer & Yu, Stella (2019) Introduction to Machine Learning Note 27 https://www.eecs189.org/static/notes/n27.pdf

[2] Stanford CS231n Convolutional Neural Networks: Architectures, Convolution/Pooling Layers https://cs231n.github.io/convolutional-networks/

[3] LeCun, Yann et al (1998) Gradient-Based Learning Applied to Document Recognition http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf

[4] Krizhevsky, Alex et al (2012) ImageNet Classification with Deep Convolutional Neural Networks https://papers.nips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[5] Simonyan, Karen & Zisserman, Andrew (2014) Very Deep Convolutional Networks for Large-Scale Image Recognition https://arxiv.org/pdf/1409.1556.pdf

[6] Hassan, Muneeb ul (2018) VGG16 – Convolutional Network for Classification and Detection https://neurohive.io/en/popular-networks/vgg16/

[7] Szegedy, Christian et al (2014) Going Deeper With Convolutions https://arxiv.org/pdf/1409.4842v1.pdf

[8] He, Kaiming et al (2015) Deep Residual Learning for Image Recognition https://arxiv.org/pdf/1512.03385.pdf

[9] Weisstein, Eric W. "Convolution." From MathWorld—A Wolfram Web Resource. https://mathworld.wolfram.com/Convolution.html