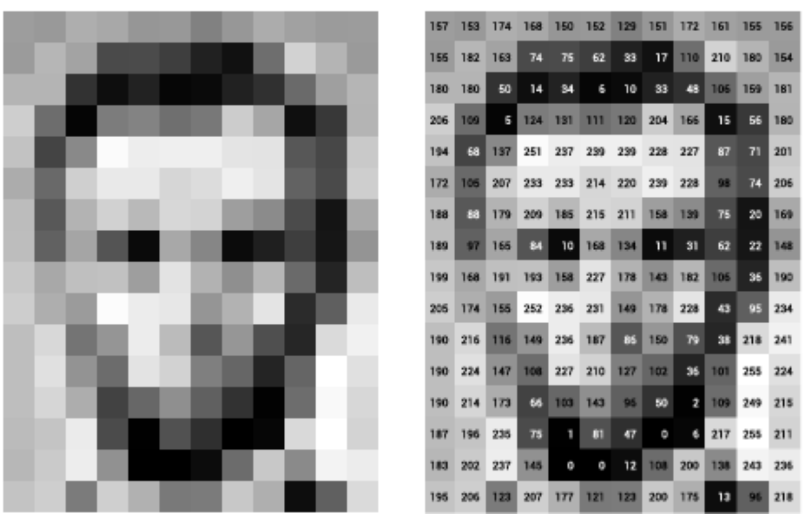
CONVOLUTION NUERAL NETWORK

Convolutional Neural Networks (ConvNets or CNNs) are a category of Neural Networks that have proven very effective in areas such as image recognition and classification. ConvNets have been successful in identifying faces, objects and traffic signs apart from powering vision in robots and self driving cars.

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 article we will discuss the architecture of a CNN and the back propagation algorithm to compute the gradient with respect to the parameters of the model in order to use gradient based optimization.



157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

Why convolution?

It is a common practice nowadays to construct deep neural networks with a set of convolution layers. However, it was not always like this, earlier neural networks and other machine learning frameworks didn’t employ convolutions. Feature extraction and learning were two separate fields of study until recently. This is why it is important to understand how Convolution works and why it took such an important place in deep learning architectures.

SPATIAL INVARIANCE or LOSS IN FEATURES

The spatial features of a 2D image are lost when it is flattened to a 1D vector input. Before feeding an image to the hidden layers of an MLP, we must flatten the image matrix to a 1D vector. This implies that all of the image's 2D information is discarded.

Sample Image

0	0	0	5	0	0	0
0	5	18	32	18	5	0
0	18	64	100	64	18	0
5	32	100	100	100	32	5
0	18	64	100	64	18	0
0	5	18	32	18	5	0
0	0	0	5	0	0	0

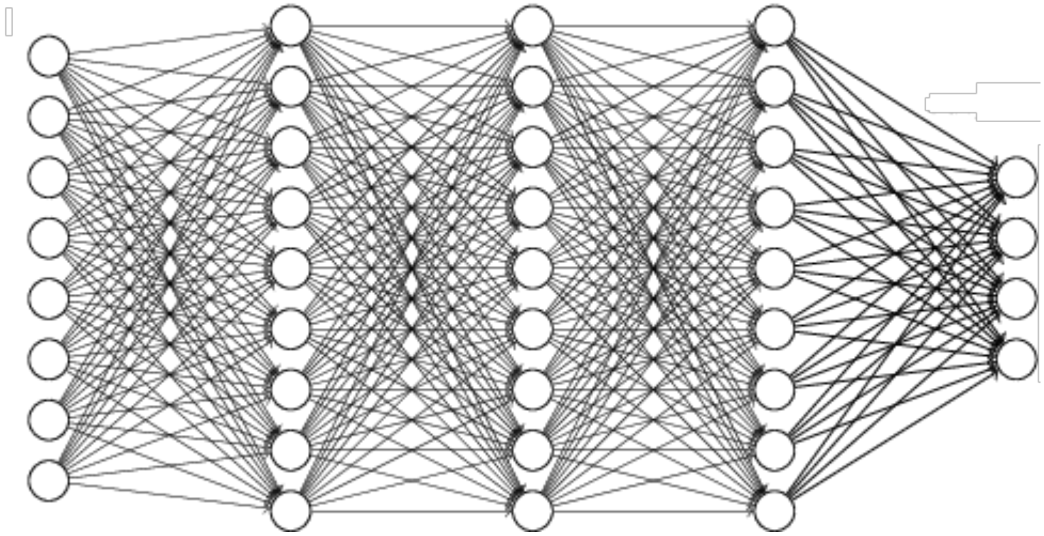
Increase in Parameter Issue

While increase in Parameter Issue is not a big problem for the MNIST dataset because the images are really small in size (28 × 28), what happens when we try to process larger images?

For example, if we have an image with dimensions 1,000 × 1,000, it will yield 1 million parameters for each node in the first hidden layer.

- So if the first hidden layer has 1,000 neurons, this will yield 1 billion parameters even in such a small network. You can imagine the computational complexity of optimizing 1 billion parameters after only the first layer.

Fully Connected Neural Net

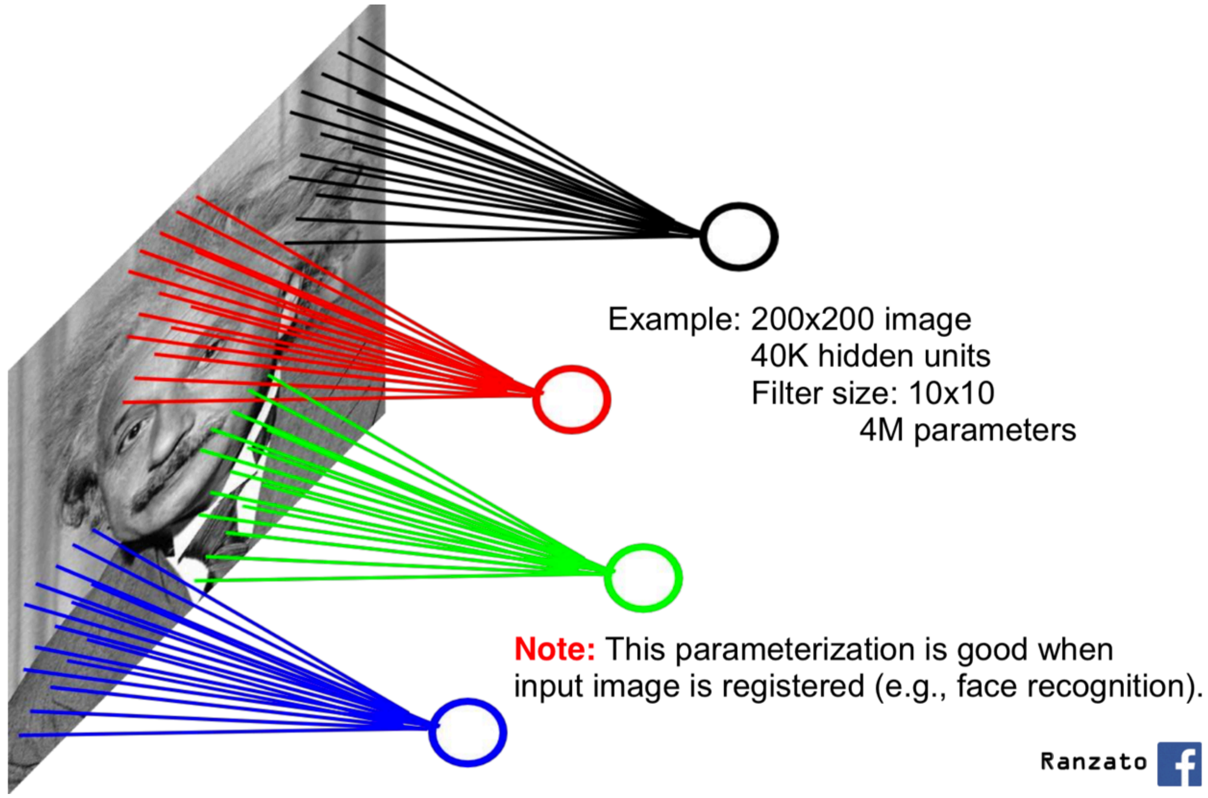


A fully connected neural network consists of a series of fully connected layers that connect every neuron in one layer to every neuron in the other layer.

Local Connected Neural Net

Before we dive into the details of building a locally connected neural network, let’s first define what it is. As mentioned earlier, a locally connected neural network is a type of convolutional neural network (CNN) that has a specific topology. In a traditional CNN, each neuron in a layer is connected to a small, local region of the previous layer using a set of shared weights. This allows the network to learn spatial or temporal features in the data.

In a locally connected neural network, each neuron in a layer is only connected to a small, local region of the previous layer, but instead of using shared weights, each neuron has its own set of weights. This can be useful when dealing with data that has a more complex spatial or temporal structure, as it allows the network to learn more specific features in each local region.



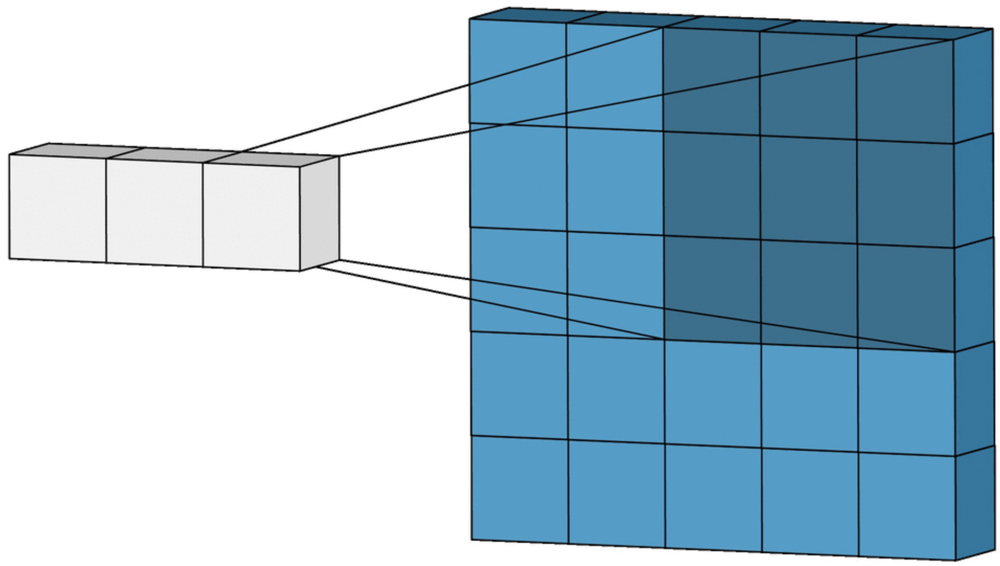
Source

Guide for design of a neural network architecture suitable for computer vision

- In the earliest layers, our network should respond similarly to the same patch, regardless of where it appears in the image. This principle is called translation invariance.
- The earliest layers of the network should focus on local regions, without regard for the contents of the image in distant regions. This is the locality principle. Eventually, these local representations can be aggregated to make predictions at the whole image level.

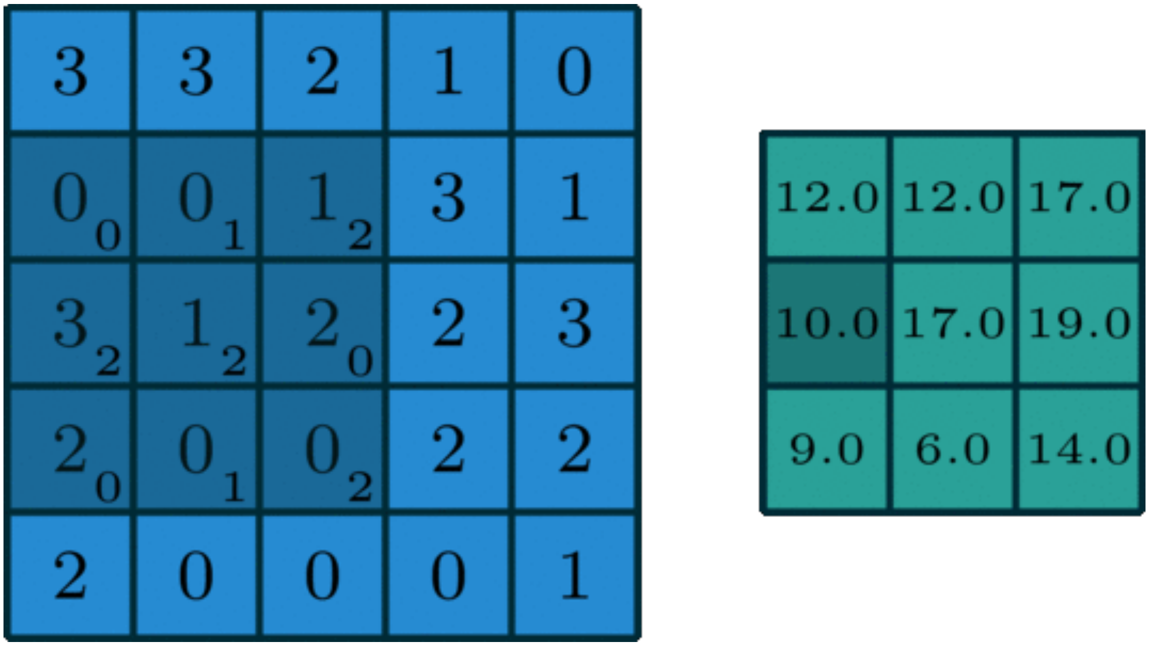
Visualizing the Process

Simple Convolution



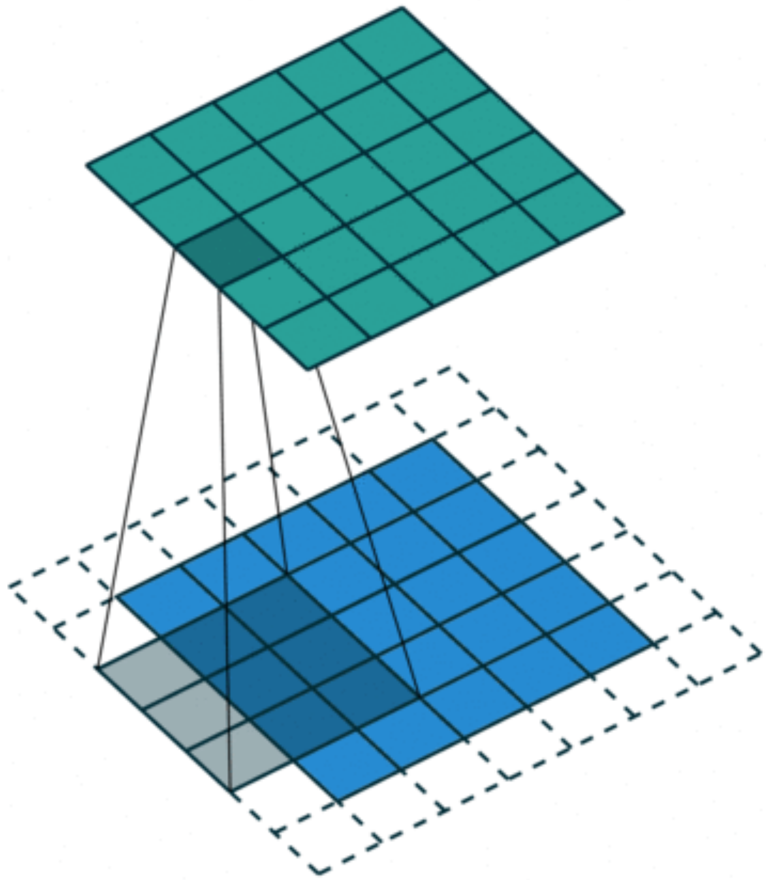
Matrix Calculation

In mathematics, particularly in linear algebra, matrix multiplication is a binary operation that produces a matrix from two matrices. For matrix multiplication, the number of columns in the first matrix must be equal to the number of rows in the second matrix



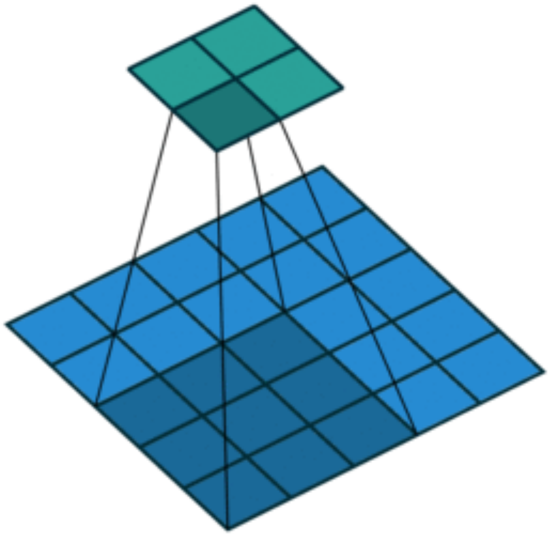
Padding Concept

Padding is a term relevant to convolutional neural networks as it refers to the amount of pixels added to an image when it is being processed by the kernel of a CNN. For example, if the padding in a CNN is set to zero, then every pixel value that is added will be of value zero.



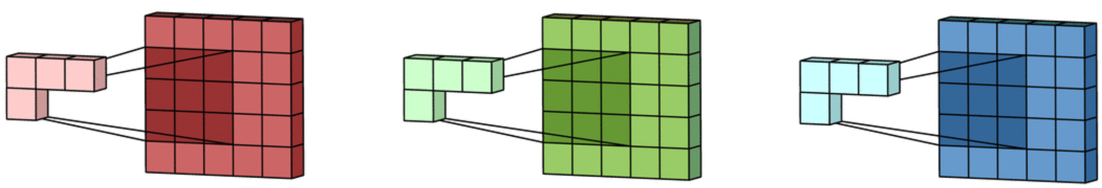
Stride Concept

The stride indicates how many steps we take in each convolutional step.It is always one. We can see that the output is less than the input in size. We employ padding to keep the output's dimension the same as the input's. The method of padding involves symmetrically adding zeros to the input matrix.



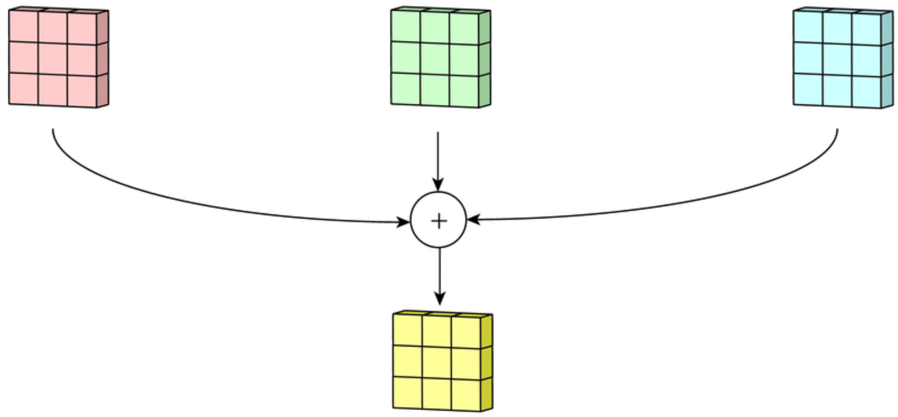
Feature Accumulation

The feature maps in Convolutional Neural Networks (CNNs) can differ significantly for different types of input data, such as text, image, and audio. In image processing tasks, the feature maps in CNNs represent visual patterns and features such as edges, corners, shapes, and textures in the input image.



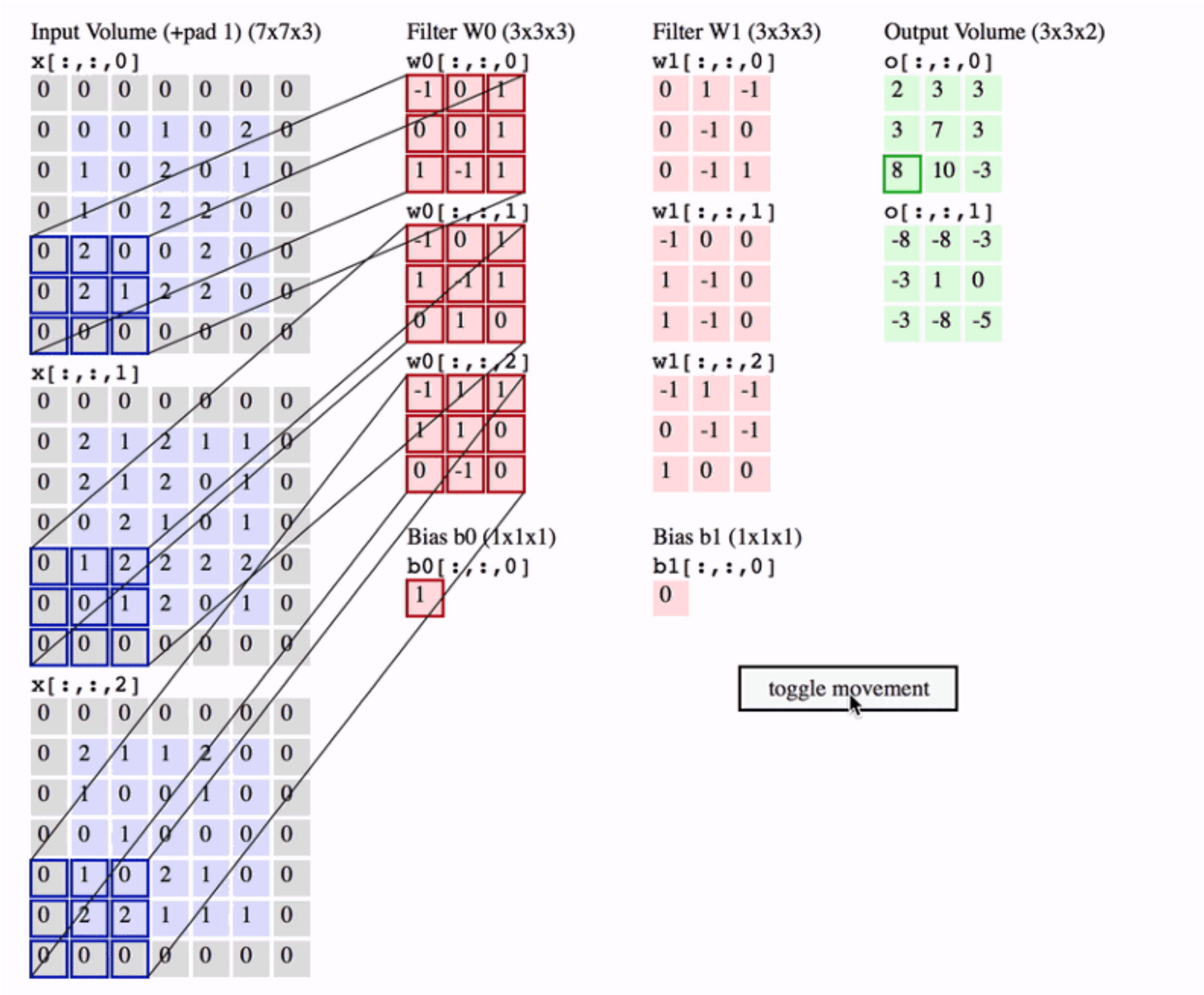
Feature Aggregation

Inside a computational neuron, the weights and the inputs to the neurons are interacted and aggregated into a single value. The way we gather the input from the other previous neurons are called aggregation function.



Convolution Operation

Convolutional Operation means for a given input we re-estimate it as the weighted average of all the inputs around it. We have some weights assigned to the neighbor values and we take the weighted sum of the neighbor values to estimate the value of the current input/pixel.

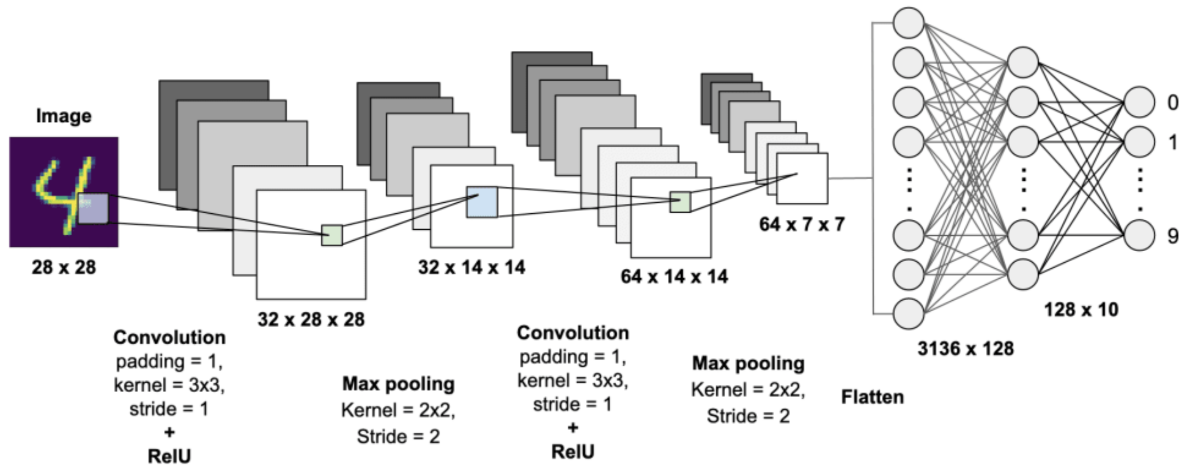


Source

Source

The CNN Complete Network Overview

A CNN typically has three layers: a convolutional layer, a pooling layer, and a fully connected layer.



Convolution Layer

The convolution layer is the core building block of the CNN. It carries the main portion of the network's computational load.

This layer performs a dot product between two matrices, where one matrix is the set of learnable parameters otherwise known as a kernel, and the other matrix is the restricted portion of the receptive field. The kernel is spatially smaller than an image but is more in-depth. This means that, if the image is composed of three (RGB) channels, the kernel height and width will be spatially small, but the depth extends up to all three channels.

During the forward pass, the kernel slides across the height and width of the image-producing the image representation of that receptive region. This produces a two-dimensional representation of the image known as an activation map that gives the response of the kernel at each spatial position of the image. The sliding size of the kernel is called a stride.

Pooling Layer

The pooling layer replaces the output of the network at certain locations by deriving a summary statistic of the nearby outputs. This helps in reducing the spatial size of the representation, which decreases the required amount of computation and weights. The pooling operation is processed on every slice of the representation individually.

There are several pooling functions such as the average of the rectangular neighborhood, L2 norm of the rectangular neighborhood, and a weighted average based on the distance from the central pixel. However, the most popular process is max pooling, which reports the maximum output from the neighborhood.

Fully Connected Layer

Neurons in this layer have full connectivity with all neurons in the preceding and succeeding layer as seen in regular FCNN. This is why it can be computed as usual by a matrix multiplication followed by a bias effect.

The FC layer helps to map the representation between the input and the output.

In [ ]:

Practical Implementation

In [2]:

```
from tensorflow import keras
from keras.datasets import cifar10

# load the pre-shuffled train and test data
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.g
z
170498071/170498071 [==============================] - 2s 0us/step
```

In [3]:

```
x_train.shape
```

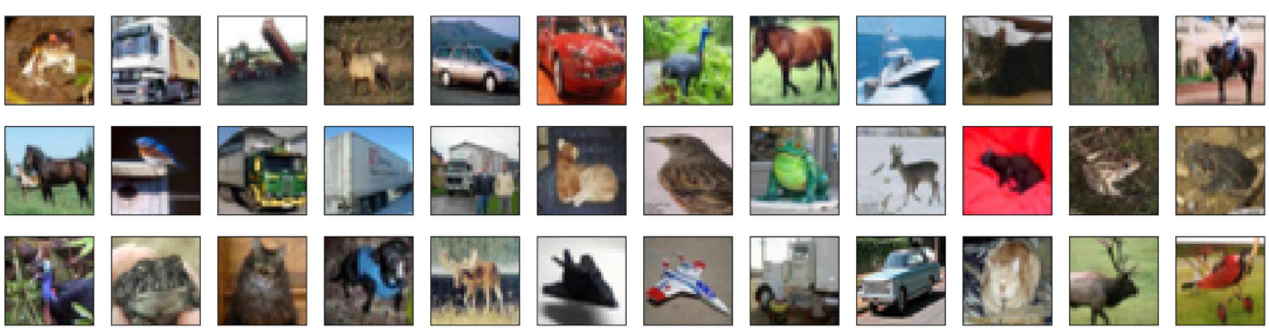
Out[3]:

```
(50000, 32, 32, 3)
```

In [4]:

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

fig = plt.figure(figsize=(20,5))
for i in range(36):
    ax = fig.add_subplot(3, 12, i + 1, xticks=[], yticks=[])
    ax.imshow(np.squeeze(x_train[i]))
```



Rescale the Images by Dividing Every Pixel in Every Image by 255

In fact, the cost function has the shape of a bowl, but it can be an elongated bowl if the features have very different scales. Figure below shows Gradient Descent on a training set where features 1 and 2 have the same scale (on the left), and on a training set where feature 1 has much smaller values than feature 2 (on the right).

Tip: When using Gradient Descent, you should ensure that all features have a similar scale to speed up training or else it will take much longer to converge.

In [5]:
```
x_train
```

Out[5]:
```
array([[[[ 59,  62,  63],
         [ 43,  46,  45],
         [ 50,  48,  43],
         ...,
         [158, 132, 108],
         [152, 125, 102],
         [148, 124, 103]],

        [[ 16,  20,  20],
         [  0,   0,   0],
         [ 18,   8,   0],
         ...,
         [123,  88,  55],
         [119,  83,  50],
         [122,  87,  57]],

        [[ 25,  24,  21],
         [ 16,   7,   0],
         [ 49,  27,   8],
         ...,
         [118,  84,  50],
         [120,  84,  50],
         [109,  73,  42]],

        ...,

        [[208, 170,  96],
         [201, 153,  34],
         [198, 161,  26],
         ...,
         [160, 133,  70],
         [ 56,  31,   7],
         [ 53,  34,  20]],

        [[180, 139,  96],
         [173, 123,  42],
         [186, 144,  30],
         ...,
```

```
         [184, 148,  94],
         [ 97,  62,  34],
         [ 83,  53,  34]],

        [[177, 144, 116],
         [168, 129,  94],
         [179, 142,  87],
         ...,
         [216, 184, 140],
         [151, 118,  84],
         [123,  92,  72]]],


       [[[154, 177, 187],
         [126, 137, 136],
         [105, 104,  95],
         ...,
         [ 91,  95,  71],
         [ 87,  90,  71],
         [ 79,  81,  70]],

        [[140, 160, 169],
         [145, 153, 154],
         [125, 125, 118],
         ...,
         [ 96,  99,  78],
         [ 77,  80,  62],
         [ 71,  73,  61]],

        [[140, 155, 164],
         [139, 146, 149],
         [115, 115, 112],
         ...,
         [ 79,  82,  64],
         [ 68,  70,  55],
         [ 67,  69,  55]],

        ...,

        [[175, 167, 166],
         [156, 154, 160],
         [154, 160, 170],
         ...,
         [ 42,  34,  36],
         [ 61,  53,  57],
         [ 93,  83,  91]],

        [[165, 154, 128],
         [156, 152, 130],
         [159, 161, 142],
         ...,
         [103,  93,  96],
         [123, 114, 120],
```

```
         [131, 121, 131]],

        [[163, 148, 120],
         [158, 148, 122],
         [163, 156, 133],
         ...,
         [143, 133, 139],
         [143, 134, 142],
         [143, 133, 144]]],


       [[[255, 255, 255],
         [253, 253, 253],
         [253, 253, 253],
         ...,
         [253, 253, 253],
         [253, 253, 253],
         [253, 253, 253]],

        [[255, 255, 255],
         [255, 255, 255],
         [255, 255, 255],
         ...,
         [255, 255, 255],
         [255, 255, 255],
         [255, 255, 255]],

        [[255, 255, 255],
         [254, 254, 254],
         [254, 254, 254],
         ...,
         [254, 254, 254],
         [254, 254, 254],
         [254, 254, 254]],

        ...,

        [[113, 120, 112],
         [111, 118, 111],
         [105, 112, 106],
         ...,
         [ 72,  81,  80],
         [ 72,  80,  79],
         [ 72,  80,  79]],

        [[111, 118, 110],
         [104, 111, 104],
         [ 99, 106,  98],
         ...,
         [ 68,  75,  73],
         [ 70,  76,  75],
         [ 78,  84,  82]],
```

```
        [[106, 113, 105],
         [ 99, 106,  98],
         [ 95, 102,  94],
         ...,
         [ 78,  85,  83],
         [ 79,  85,  83],
         [ 80,  86,  84]]],


       ...,


       [[[ 35, 178, 235],
         [ 40, 176, 239],
         [ 42, 176, 241],
         ...,
         [ 99, 177, 219],
         [ 79, 147, 197],
         [ 89, 148, 189]],

        [[ 57, 182, 234],
         [ 44, 184, 250],
         [ 50, 183, 240],
         ...,
         [156, 182, 200],
         [141, 177, 206],
         [116, 149, 175]],

        [[ 98, 197, 237],
         [ 64, 189, 252],
         [ 69, 192, 245],
         ...,
         [188, 195, 206],
         [119, 135, 147],
         [ 61,  79,  90]],

        ...,

        [[ 73,  79,  77],
         [ 53,  63,  68],
         [ 54,  68,  80],
         ...,
         [ 17,  40,  64],
         [ 21,  36,  51],
         [ 33,  48,  49]],

        [[ 61,  68,  75],
         [ 55,  70,  86],
         [ 57,  79, 103],
         ...,
         [ 24,  48,  72],
         [ 17,  35,  53],
         [  7,  23,  32]],
```

```
 [[ 44,  56,  73],
  [ 46,  66,  88],
  [ 49,  77, 105],
  ...,
  [ 27,  52,  77],
  [ 21,  43,  66],
  [ 12,  31,  50]]],


[[[189, 211, 240],
  [186, 208, 236],
  [185, 207, 235],
  ...,
  [175, 195, 224],
  [172, 194, 222],
  [169, 194, 220]],

 [[194, 210, 239],
  [191, 207, 236],
  [190, 206, 235],
  ...,
  [173, 192, 220],
  [171, 191, 218],
  [167, 190, 216]],

 [[208, 219, 244],
  [205, 216, 240],
  [204, 215, 239],
  ...,
  [175, 191, 217],
  [172, 190, 216],
  [169, 191, 215]],

 ...,

 [[207, 199, 181],
  [203, 195, 175],
  [203, 196, 173],
  ...,
  [135, 132, 127],
  [162, 158, 150],
  [168, 163, 151]],

 [[198, 190, 170],
  [189, 181, 159],
  [180, 172, 147],
  ...,
  [178, 171, 160],
  [175, 169, 156],
  [175, 169, 154]],

 [[198, 189, 173],
```

```
         [189, 181, 162],
         [178, 170, 149],
         ...,
         [195, 184, 169],
         [196, 189, 171],
         [195, 190, 171]]],


       [[[229, 229, 239],
         [236, 237, 247],
         [234, 236, 247],
         ...,
         [217, 219, 233],
         [221, 223, 234],
         [222, 223, 233]],

        [[222, 221, 229],
         [239, 239, 249],
         [233, 234, 246],
         ...,
         [223, 223, 236],
         [227, 228, 238],
         [210, 211, 220]],

        [[213, 206, 211],
         [234, 232, 239],
         [231, 233, 244],
         ...,
         [220, 220, 232],
         [220, 219, 232],
         [202, 203, 215]],

        ...,

        [[150, 143, 135],
         [140, 135, 127],
         [132, 127, 120],
         ...,
         [224, 222, 218],
         [230, 228, 225],
         [241, 241, 238]],

        [[137, 132, 126],
         [130, 127, 120],
         [125, 121, 115],
         ...,
         [181, 180, 178],
         [202, 201, 198],
         [212, 211, 207]],

        [[122, 119, 114],
         [118, 116, 110],
         [120, 116, 111],
```

```
        ...,
        [179, 177, 173],
        [164, 164, 162],
        [163, 163, 161]]]], dtype=uint8)
```

In [6]:
```
x_train = x_train.astype('float32')/255
x_test = x_test.astype('float32')/255
```

In [7]:
```
from keras.utils import np_utils
from tensorflow import keras

# one-hot encode the labels
num_classes = len(np.unique(y_train))
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

# break training set into training and validation sets
(x_train, x_valid) = x_train[5000:], x_train[:5000]
(y_train, y_valid) = y_train[5000:], y_train[:5000]

# print shape of training set
print('x_train shape:', x_train.shape)

# print number of training, validation, and test images
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
print(x_valid.shape[0], 'validation samples')
```

```
x_train shape: (45000, 32, 32, 3)
45000 train samples
10000 test samples
5000 validation samples
```

In [8]:
```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

model = Sequential()
model.add(Conv2D(filters=16, kernel_size=2, padding='same', activation='relu
                        input_shape=(32, 32, 3)))
model.add(MaxPooling2D(pool_size=2))
model.add(Conv2D(filters=32, kernel_size=2, padding='same', activation='relu
model.add(MaxPooling2D(pool_size=2))
model.add(Conv2D(filters=64, kernel_size=2, padding='same', activation='relu
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.3))
model.add(Flatten())
model.add(Dense(500, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(10, activation='softmax'))

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 16)        208

 max_pooling2d (MaxPooling2D  (None, 16, 16, 16)       0
 )

 conv2d_1 (Conv2D)           (None, 16, 16, 32)        2080

 max_pooling2d_1 (MaxPooling  (None, 8, 8, 32)         0
 2D)

 conv2d_2 (Conv2D)           (None, 8, 8, 64)          8256

 max_pooling2d_2 (MaxPooling  (None, 4, 4, 64)         0
 2D)

 dropout (Dropout)           (None, 4, 4, 64)          0

 flatten (Flatten)           (None, 1024)              0

 dense (Dense)               (None, 500)               512500

 dropout_1 (Dropout)         (None, 500)               0

 dense_1 (Dense)             (None, 10)                5010

=================================================================
Total params: 528,054
Trainable params: 528,054
Non-trainable params: 0
_________________________________________________________________
```

In [9]:

```
model.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=
```

In [10]:

```
from keras.callbacks import ModelCheckpoint

# train the model
checkpointer = ModelCheckpoint(filepath='model.weights.best.hdf5', verbose=3

hist = model.fit(x_train, y_train, batch_size=32, epochs=5,
          validation_data=(x_valid, y_valid), callbacks=[checkpointer],
          verbose=3, shuffle=True)
```

```
Epoch 1/5

Epoch 1: val_loss improved from inf to 1.39245, saving model to model.weight
s.best.hdf5
Epoch 2/5

Epoch 2: val_loss improved from 1.39245 to 1.29743, saving model to model.we
ights.best.hdf5
Epoch 3/5

Epoch 3: val_loss improved from 1.29743 to 1.24308, saving model to model.we
ights.best.hdf5
Epoch 4/5

Epoch 4: val_loss improved from 1.24308 to 1.05789, saving model to model.we
ights.best.hdf5
Epoch 5/5

Epoch 5: val_loss improved from 1.05789 to 0.96741, saving model to model.we
ights.best.hdf5
```

In [11]:

```
model.load_weights('model.weights.best.hdf5')
```

In [12]:

```
# get predictions on the test set
y_hat = model.predict(x_test)

# define text labels (source: https://www.cs.toronto.edu/~kriz/cifar.html)
cifar10_labels = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'f
```

```
313/313 [==============================] - 1s 2ms/step
```

In [13]:

```
# plot a random sample of test images, their predicted labels, and ground tr
fig = plt.figure(figsize=(20, 8))
for i, idx in enumerate(np.random.choice(x_test.shape[0], size=32, replace=F
    ax = fig.add_subplot(4, 8, i + 1, xticks=[], yticks=[])
    ax.imshow(np.squeeze(x_test[idx]))
    pred_idx = np.argmax(y_hat[idx])
    true_idx = np.argmax(y_test[idx])
    ax.set_title("{} ({})".format(cifar10_labels[pred_idx], cifar10_labels[t
                 color=("blue" if pred_idx == true_idx else "red"))
```

In [14]:
```
# evaluate test accuracy
score = model.evaluate(x_test, y_test, verbose=0)
accuracy = 100*score[1]

# print test accuracy
print('Test accuracy: %.4f%%' % accuracy)
```

```
Test accuracy: 64.8500%
```

In [ ]: