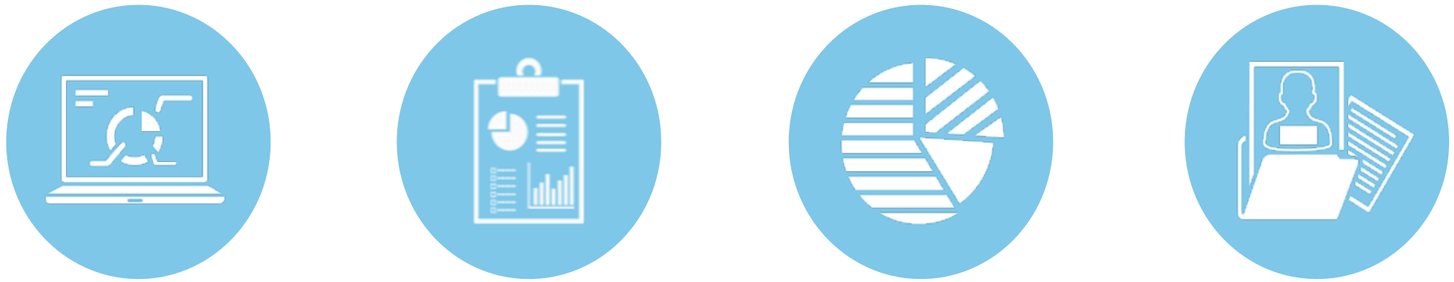
Deep Learning

Lesson 7—Convolutional Neural Network



Learning Objectives

- Describe what CNNs are and their applications
- Explain how CNNs differ from ANNs
- Discuss the process of convolution and how it work for image processing or other tasks
- Illustrate how zero padding works with variations in kernel weights
- Explain how to calculate the weighted inputs for all the feature maps stacked together
- Elaborate the pooling concepts in CNNs
- Learn how to implement CNNs within TensorFlow

Convolutional Neural Network

Topic 1—CNNs and Their Uses

Convolutional Neural Network (CNN)

Convolutional Neural Network

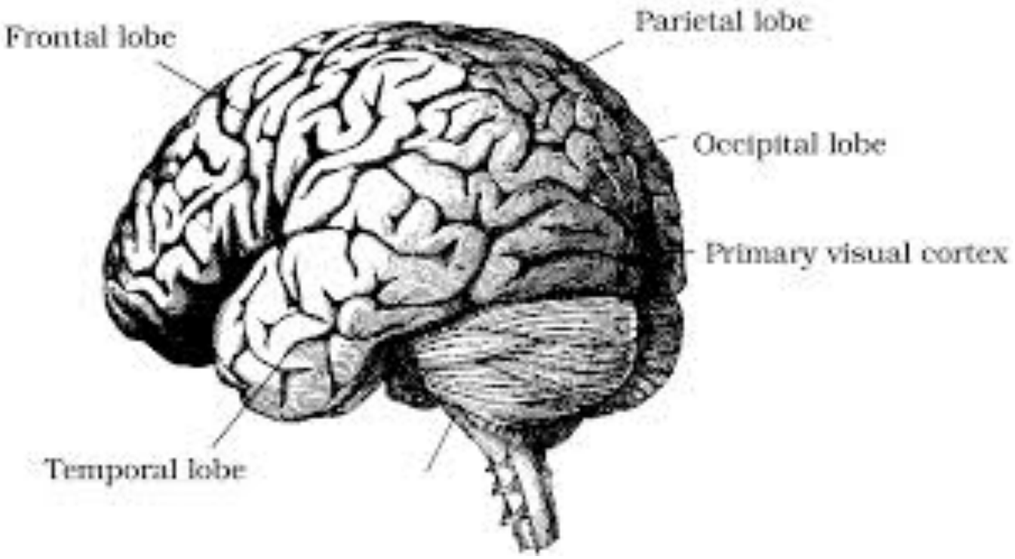
- Convolutional Neural Networks (CNN) are neural networks mainly used for image processing and classification.

Until quite recently, computers were not good at tasks like recognizing puppy in a picture or recognizing spoken words, which humans excel at.

Convolutional Neural Network (Contd.)

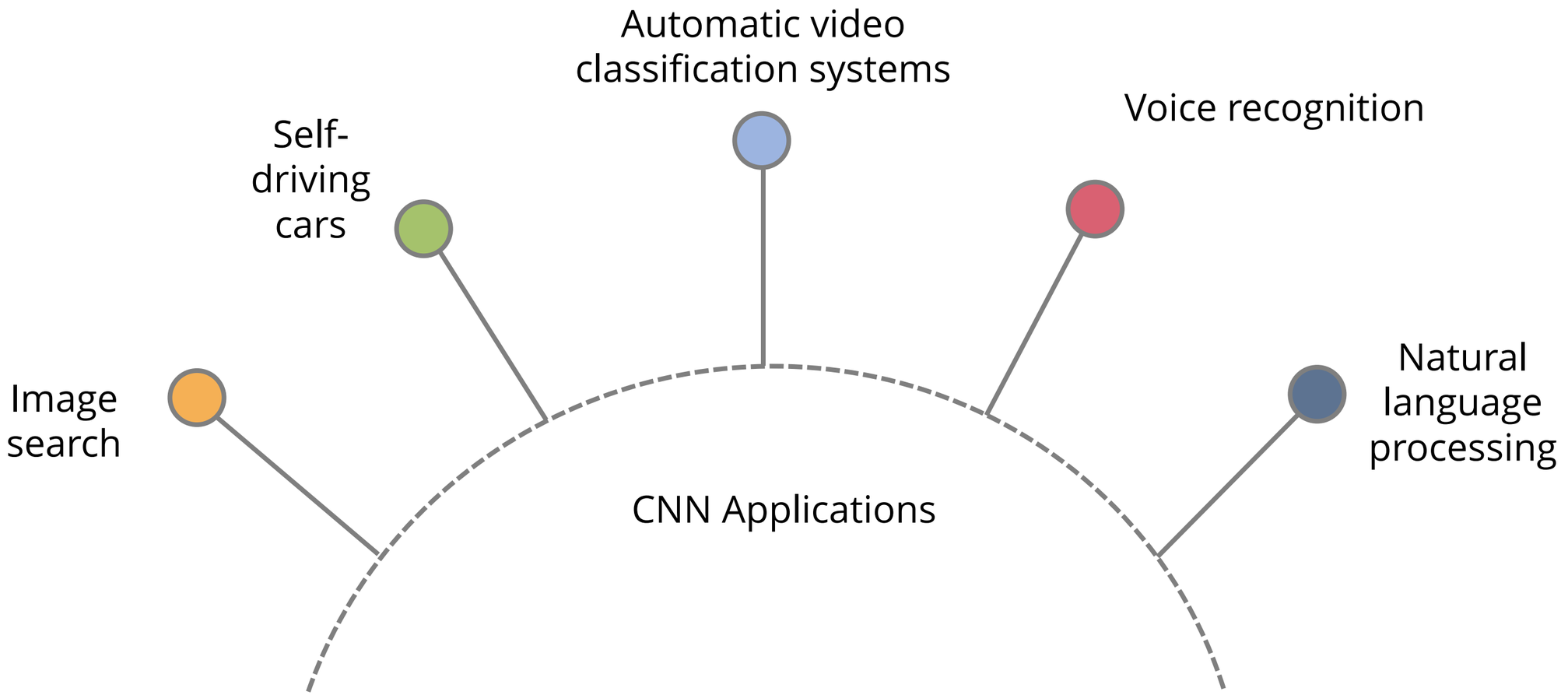
- The human brain uses special visual, auditory or other sensory modules to process tasks even before sensory information reaches consciousness.

- CNNs are based on brain's visual cortex function.



> Convolutional Neural Networks are also popularly known as ConvNets/CovNets.

Applications of CNN

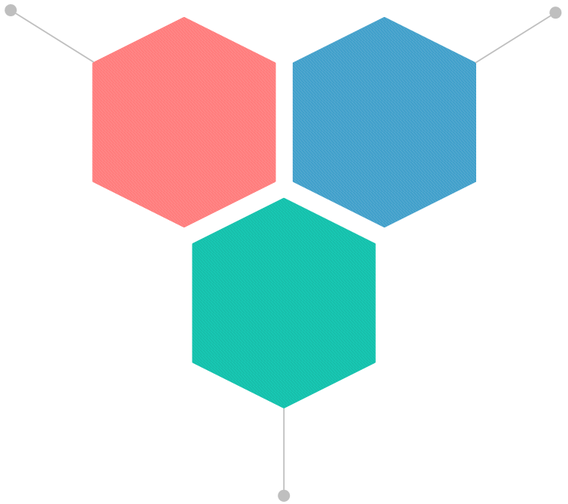


Applications of CNN (Contd.)

- There has been a lot of progress with CNNs in various image tasks:

Object detection

Detecting if an image has a person, animal or vehicle etc



Object localization

Zeroing in on the object of interest in the image. For example, detecting the person in the image and drawing a boundary around it

Image segmentation

Here the network outputs an image where each pixel indicates the class of the object to which the corresponding input pixel belongs

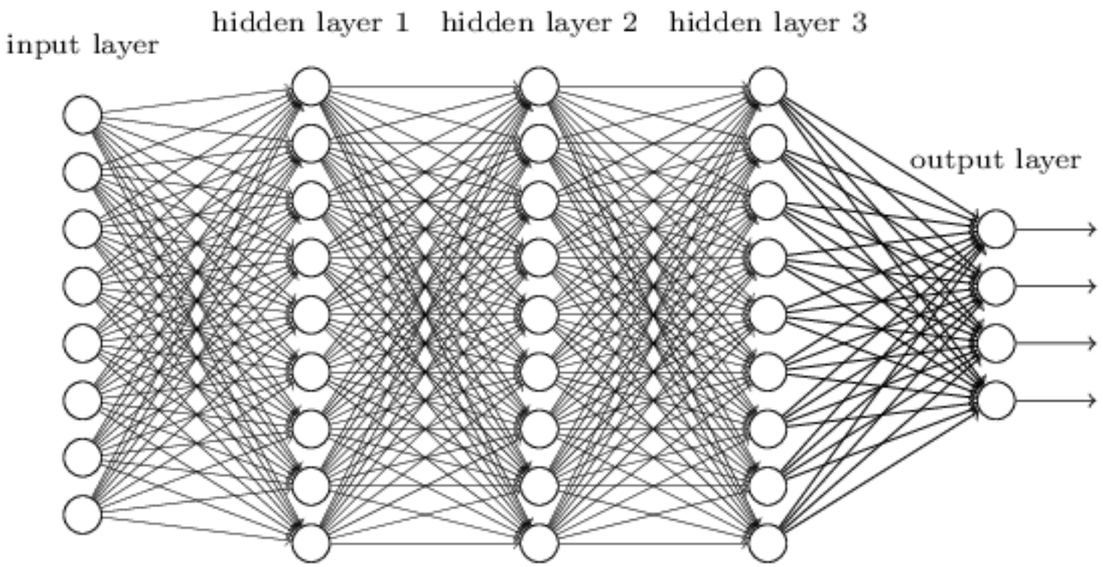
NOTE: In Object localization, the network outputs bounding boxes around various objects. This typically needs a CNN followed by an RNN scheme.

Convolutional Neural Network

Topic 2— How CNNs Differ From ANNs

Difference between ANN and CNN

- In an ANN, each neuron in the network is connected to every other neuron in the adjacent hidden layers.



- In a CNN, each neuron in the hidden layer is connected to a small region of the input neurons.

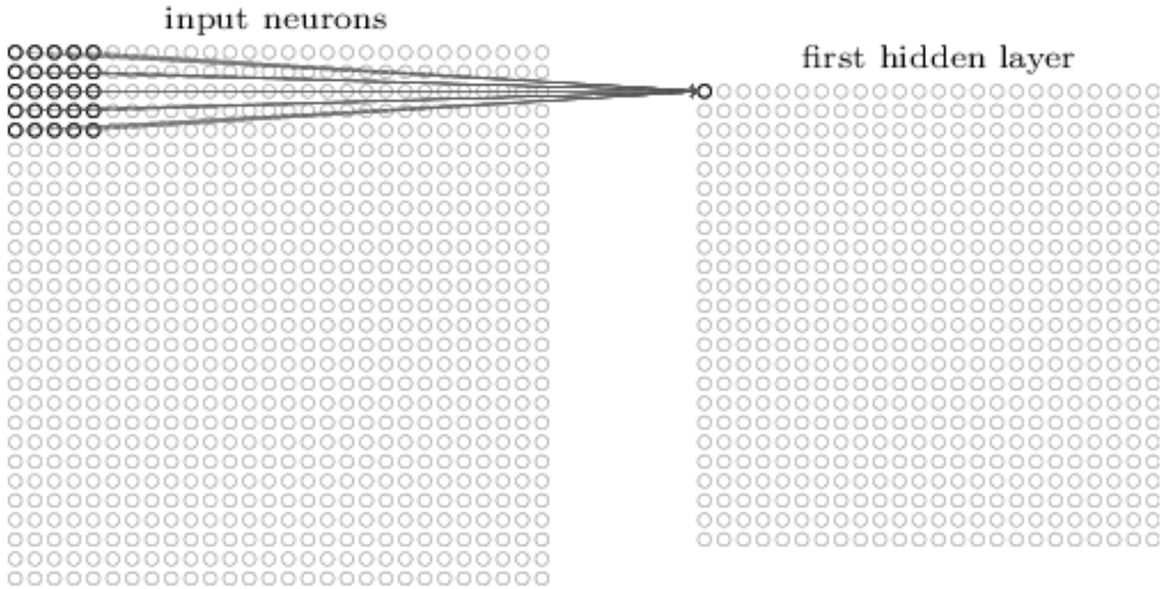


Image Source : "Neural Networks and Deep Learning" by Micheal Nielson

Difference between ANN and CNN (Contd.)

Why not use regular ANNs for image tasks ?

For small images it might work, but for large images the no of pixels are so many leading to millions of connections between neurons, leading to intractable solutions.

CNN Architecture

- CNN architecture has the following configuration:
  - Convolutional Layer
  - Pooling Layer
  - Fully connected Layer
- Most normal configuration is convolutional layer, followed by ReLU layer, followed by a pooling layer. The sets then keep repeating.
- The final layer is the fully connected layer, which precedes the final image classification.

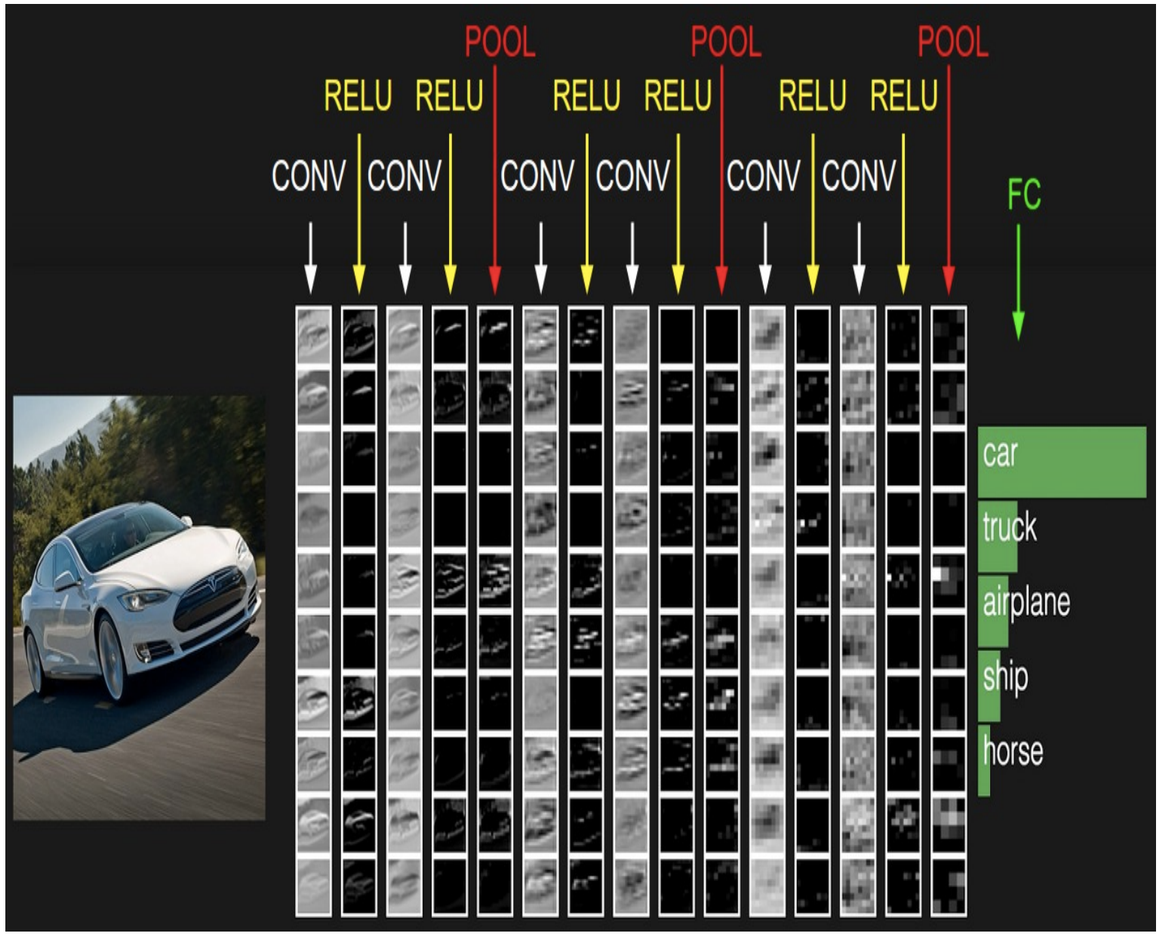


Image Source: http://cs231n.github.io

Other Architectures of CNN

- Over the years, many variants of this basic architecture have been developed such as:

LeNet-5 (1998)

AlexNet (2012)

ZFNet (2013)

GoogLeNet (2014)

VGGNet(2014)

ResNet (2015)

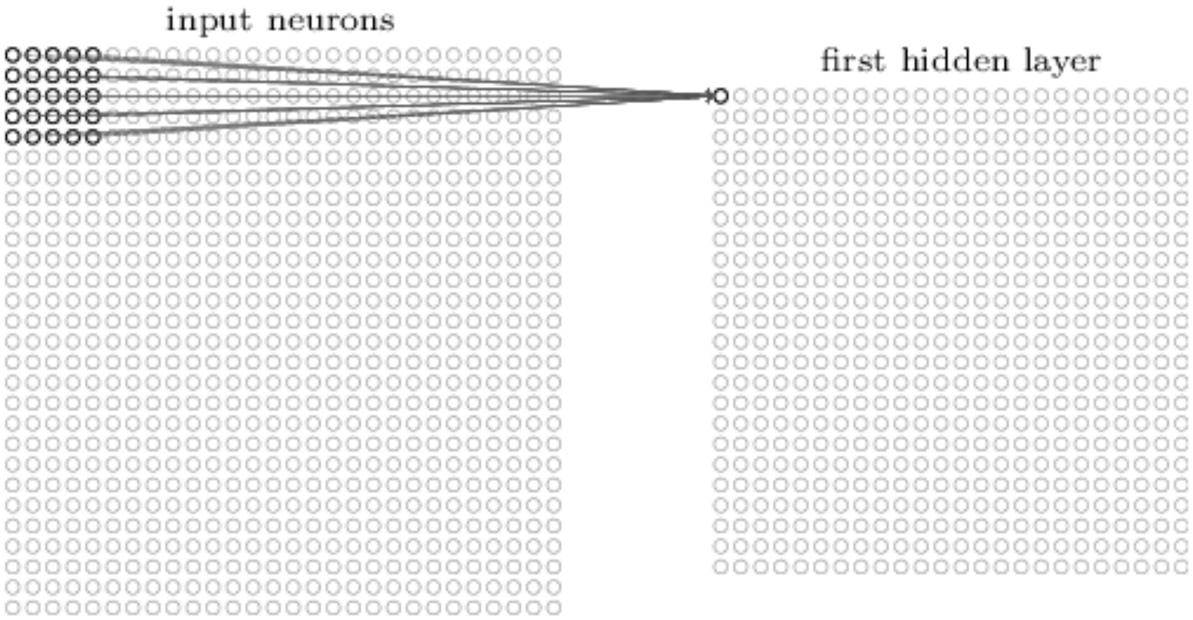
- These models can be used in applications by developers.

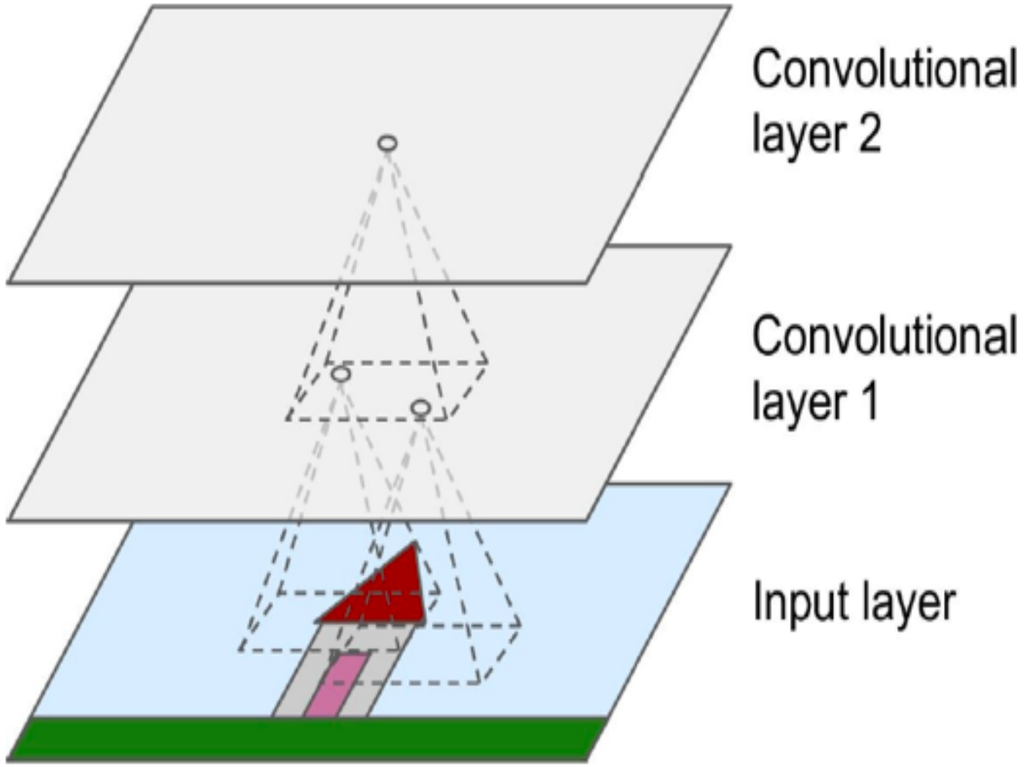
Convolutional Neural Network

Topic 3—Convolution process

Convolutional Layer

- Here neurons in the first hidden/convolutional layer are connected only to a subset of neurons in the prior layer, and so on in the next convolutional layer. This area in the input image is called **Receptive Field**.





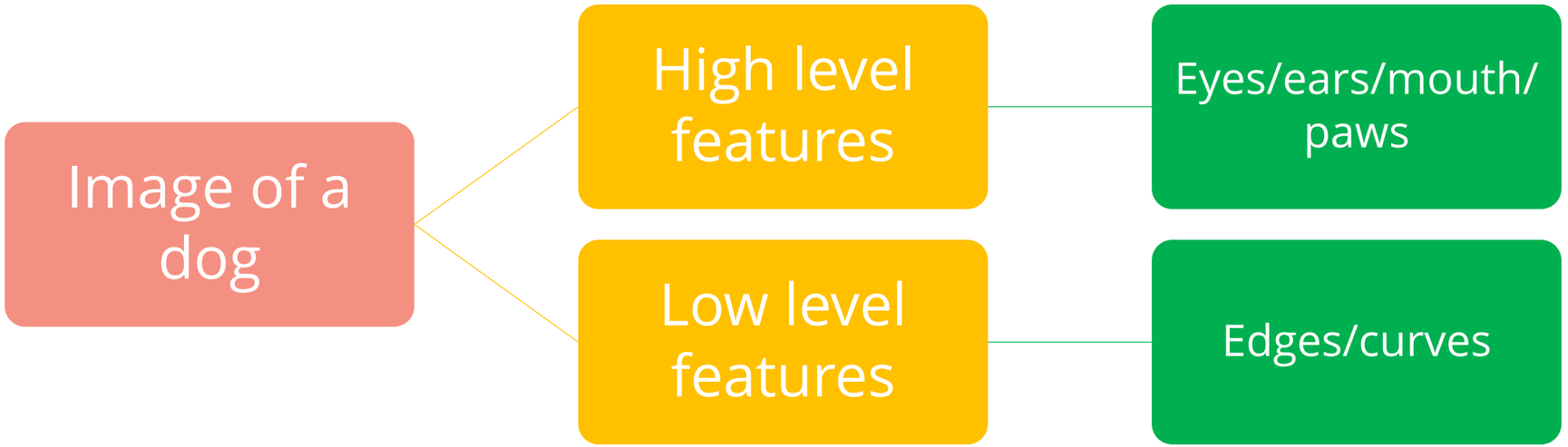
CNN layers with rectangular local receptive fields

Image Source: "Hands-on Machine Learning with Scikit-Learn and TensorFlow " by Aurelien Geron

Convolutional Layer

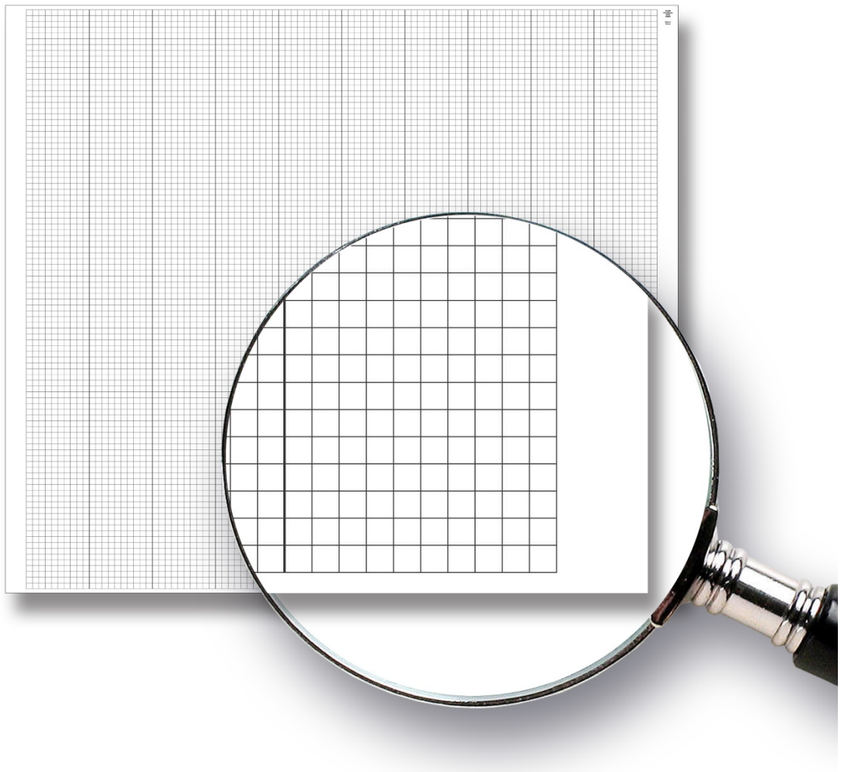
HIGH AND LOW LEVEL FEATURES

First hidden layer focuses on lower-level features and the later layers assemble them into higher-level features.



Process of Convolution

- Imagine a small patch being slid across the input image. This sliding is called **convolving**.
- It is similar to a flashlight moving from the top left end progressively scanning the entire image. This patch is called the **filter/kernel**. The area under the filter is the receptive field.
- The idea is to detect local features in a smaller section of the input space, section by section to eventually cover the entire image.
- In other words, the CNN layer neurons depends only on nearby neurons from the previous layer. This has the impact of discovering the features in a certain limited area of the input feature map.



Process of Convolution

EXAMPLE

- Assume the filter/kernel is a weight matrix "w_k". For example, let's assume a 3X3 weighted matrix.

0	1	1
1	0	0
1	0	1

- The weight matrix is a filter to extract some particular features from the original image. It could be for extracting curves, identifying a specific color, or recognizing a particular voice.

- Assume the input to be a 6X6 image.

81	2	209	44	71	58
24	56	108	98	12	112
91	0	189	65	79	232
12	0	0	5	1	71
2	32	23	58	8	209
49	98	81	112	54	9

Process of Convolution (Contd.)

EXAMPLE

- As the filter/kernel is slided across the input layer, the convolved layer is obtained by adding the values obtained by element wise multiplication of the weight matrix.

81	2	209	44	71	58
24	56	108	98	12	112
91	0	189	65	79	232
12	0	0	5	1	71
2	32	23	58	8	209
49	98	81	112	54	9

Input layer

0	1	1
1	0	0
1	0	1

Filter/Kernel

(Weighted matrix)

515			

Output

- For example, when the weighted matrix starts from the top left corner of the input layer, the output value is calculated as:

$$(81x0+2x1+209x1)+(24x1+56x0+108X0)+(91x1+0x0+189x1) = 515$$

Process of Convolution (Contd.)

EXAMPLE

- The filter then moves by 1 pixel to the next receptive field and the process is repeated. The output layer obtained after the filter slides over the entire image would be a 4X4 matrix.
- This is called an **activation map/ feature map**.

81	2	209	44	71	58
24	56	108	98	12	112
91	0	189	65	79	232
12	0	0	5	1	71
2	32	23	58	8	209
49	98	81	112	54	9

Input layer

0	1	1
1	0	0
1	0	1

Filter/Kernel

(Weighted matrix)

515	374		

Output

(Activation/Feature Map)

- The distance between two consecutive receptive fields is called the **stride**.
- In this example stride is 1 since the receptive field was moved by 1 pixel at a time.

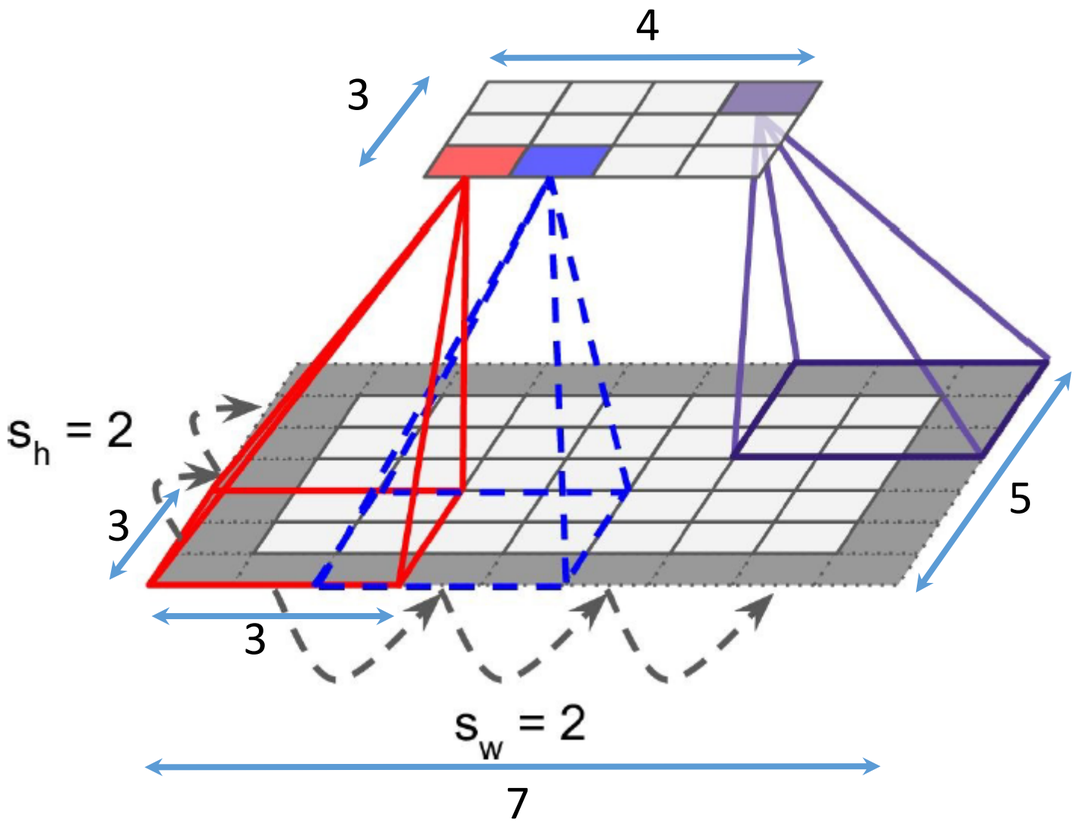
Convolutional Layer

SPACING OF STRIDES

- It is also possible to connect a large input layer to a much smaller layer by spacing out the receptive fields, i.e., increasing the stride.

- In the diagram :
  - Input layer is 5x7
  - Output layer is 3x4
  - Receptive field is 3x3
  - Stride is 2

- Here stride is same across two dimensions, but in general it can be different across height "s_h" and width "s_w".



Reducing dimensionality using a stride

Image Source: "Hands-on Machine Learning with Scikit-Learn and TensorFlow " by Aurelien Geron

Convolutional Layer

VERTICAL AND HORIZONTAL FILTERS

- The image shows two kernels – vertical and horizontal filters. Each is a 5x5 matrix with all 0s, except 1 in vertical line for vertical filter and 1 in horizontal line in horizontal filter.

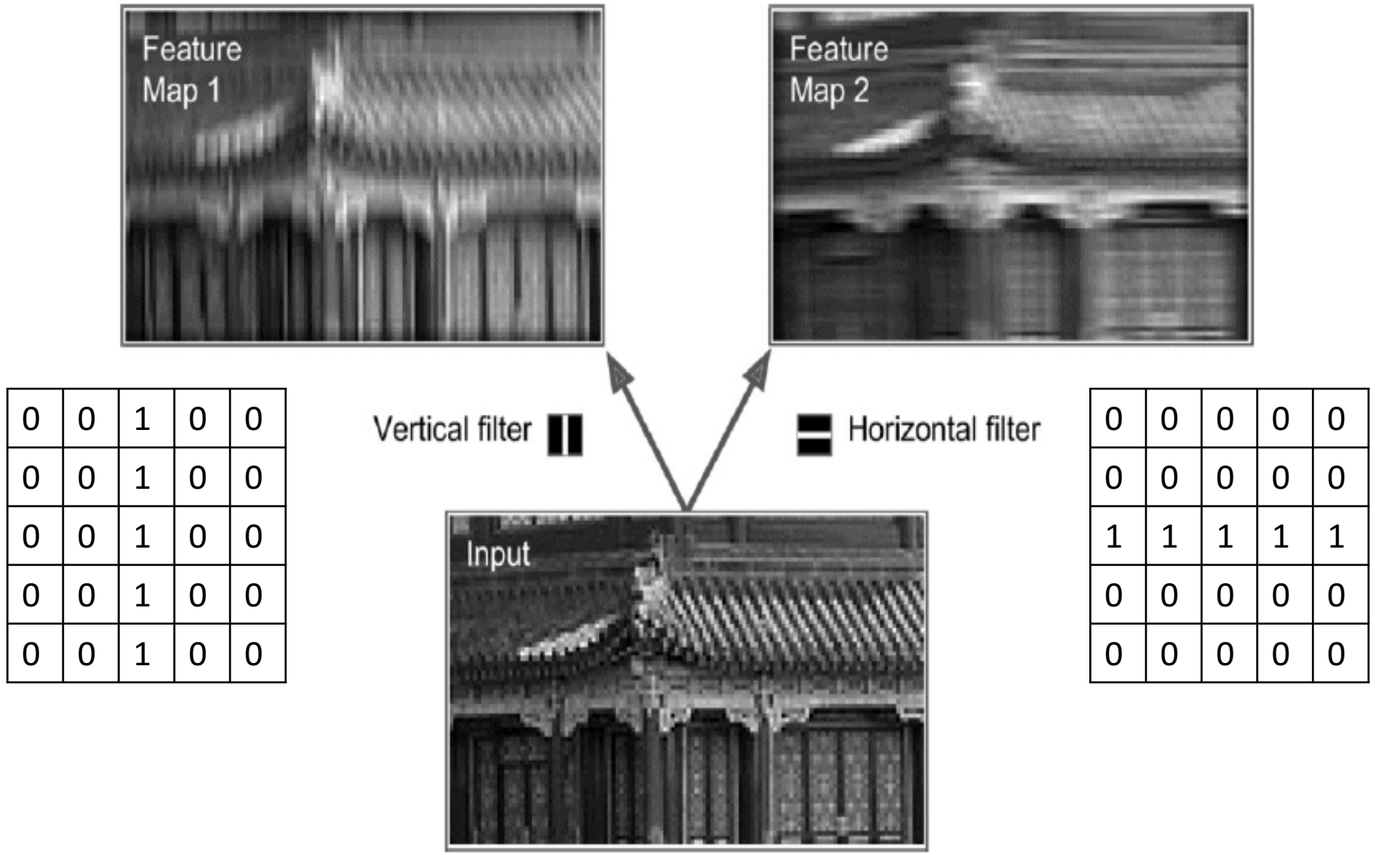


Image Source : "Hands-on Machine Learning with Scikit-Learn and TensorFlow " by Aurelien Geron

Convolutional Layer (Contd.)

VERTICAL AND HORIZONTAL FILTERS

- The effect of multiplying with vertical kernel filter is that all pixels except the vertical lines get subdued. Similarly with horizontal kernel filter, it accentuates the horizontal lines.
- The output image has a feature map, which highlights the areas in the image that are most similar to the filter.

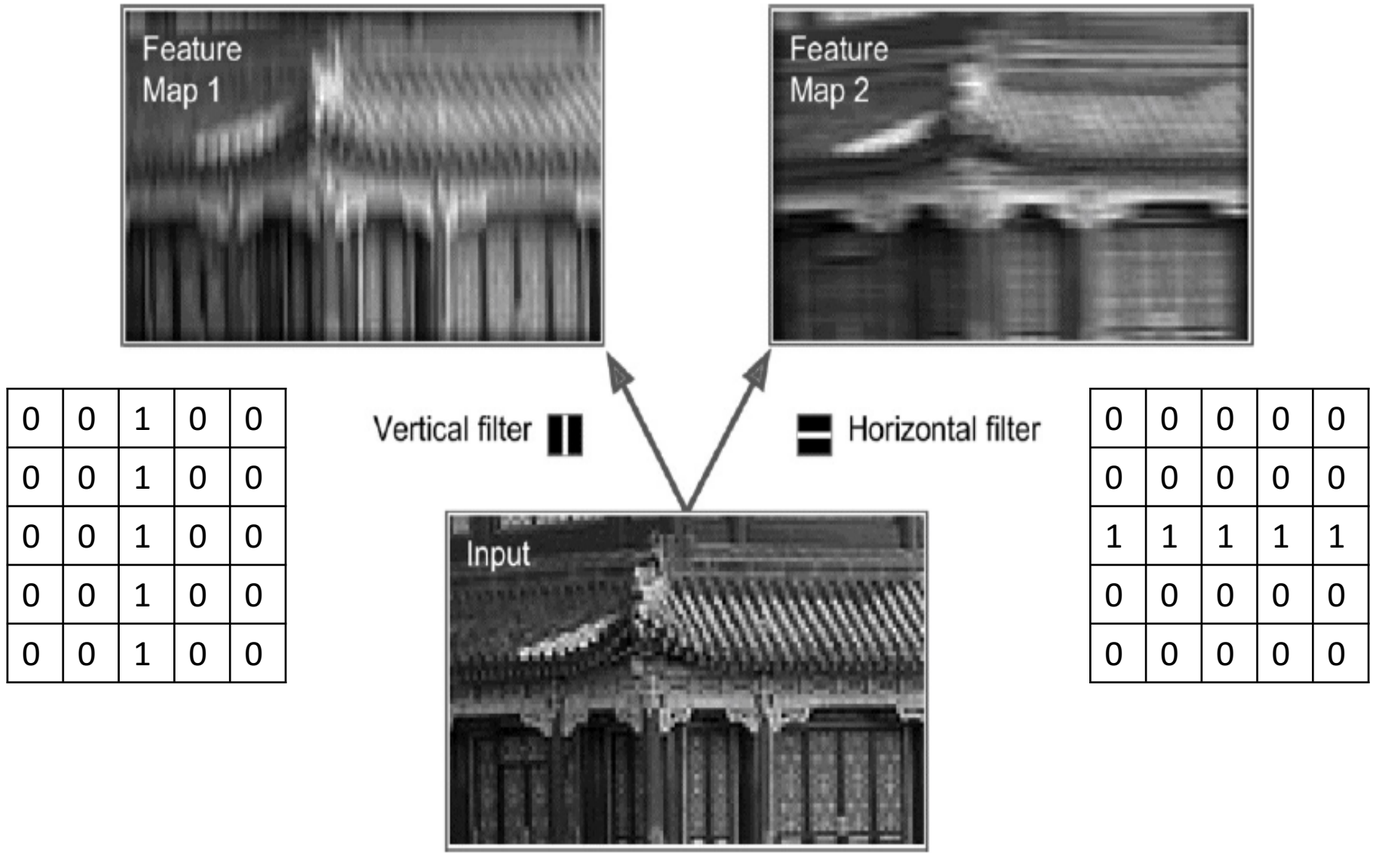


Image Source : "Hands-on Machine Learning with Scikit-Learn and TensorFlow " by Aurelien Geron

Convolutional Layer (Contd.)

VERTICAL AND HORIZONTAL FILTERS

- In this fashion, a CNN finds low level features first and then combines them in higher layers to detect complex features at points where both filters are active.

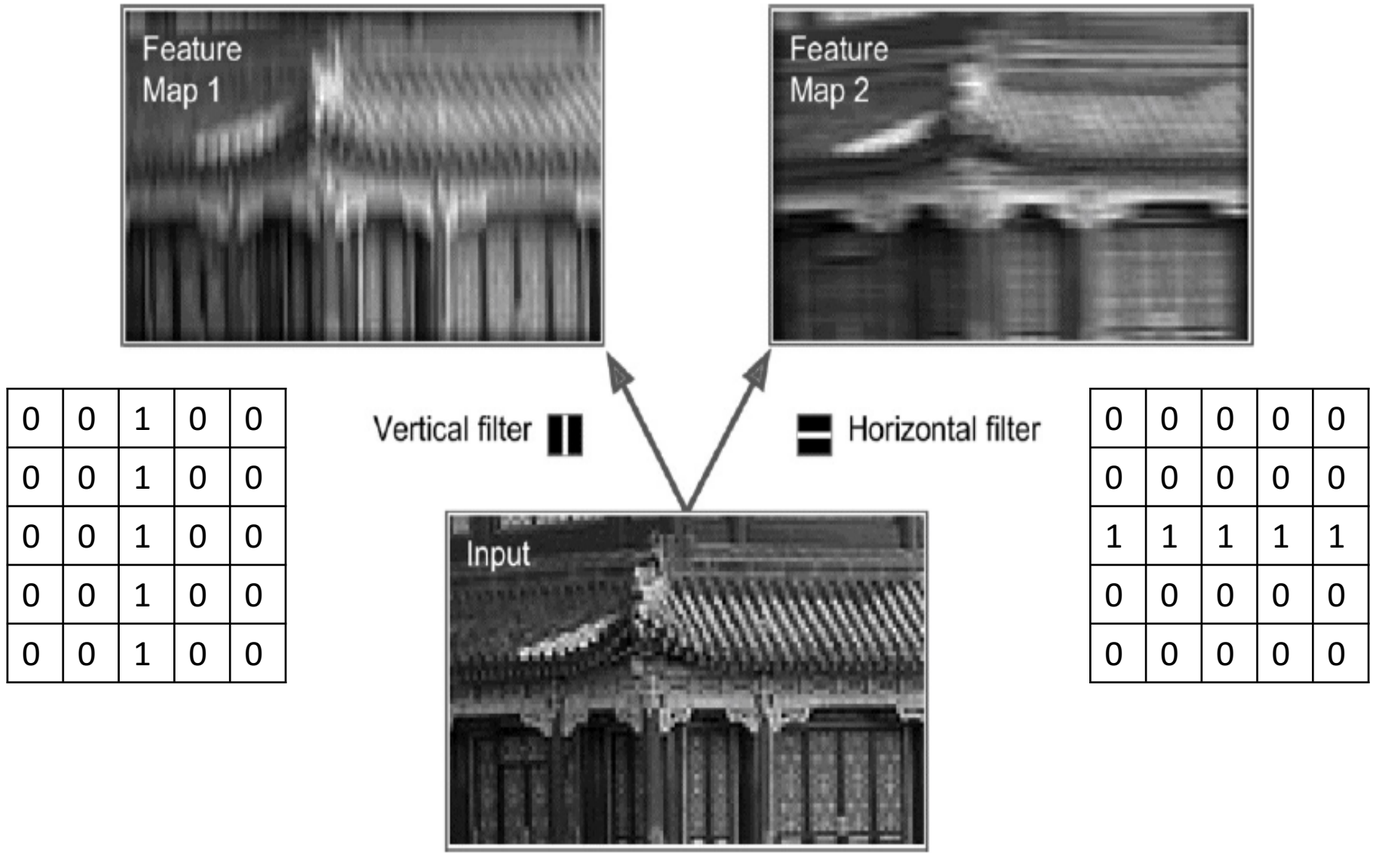


Image Source : "Hands-on Machine Learning with Scikit-Learn and TensorFlow " by Aurelien Geron

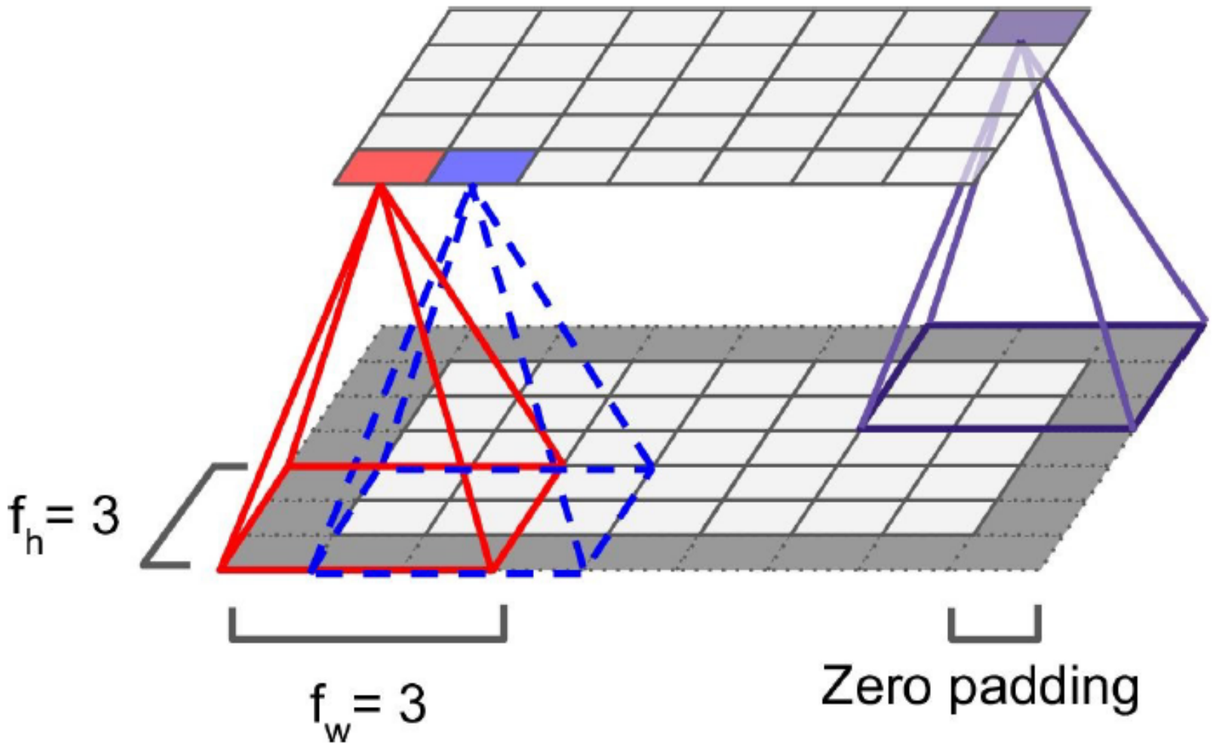
Convolutional Neural Network

Topic 4—Zero Padding

Zero Padding

- A neuron located in row i, column j of a given layer is connected to neurons in the previous layer located in rows i to $i+f_h-1$, columns j to $j+f_w-1$, where f_h and f_w are the height and width of the receptive field.

- To maintain height and width dimensions of convolutional layer same as previous layer, one zero-pads the input layer.



CNN layer with zero padding

Image Source: "Hands-on Machine Learning with Scikit-Learn and TensorFlow " by Aurelien Geron

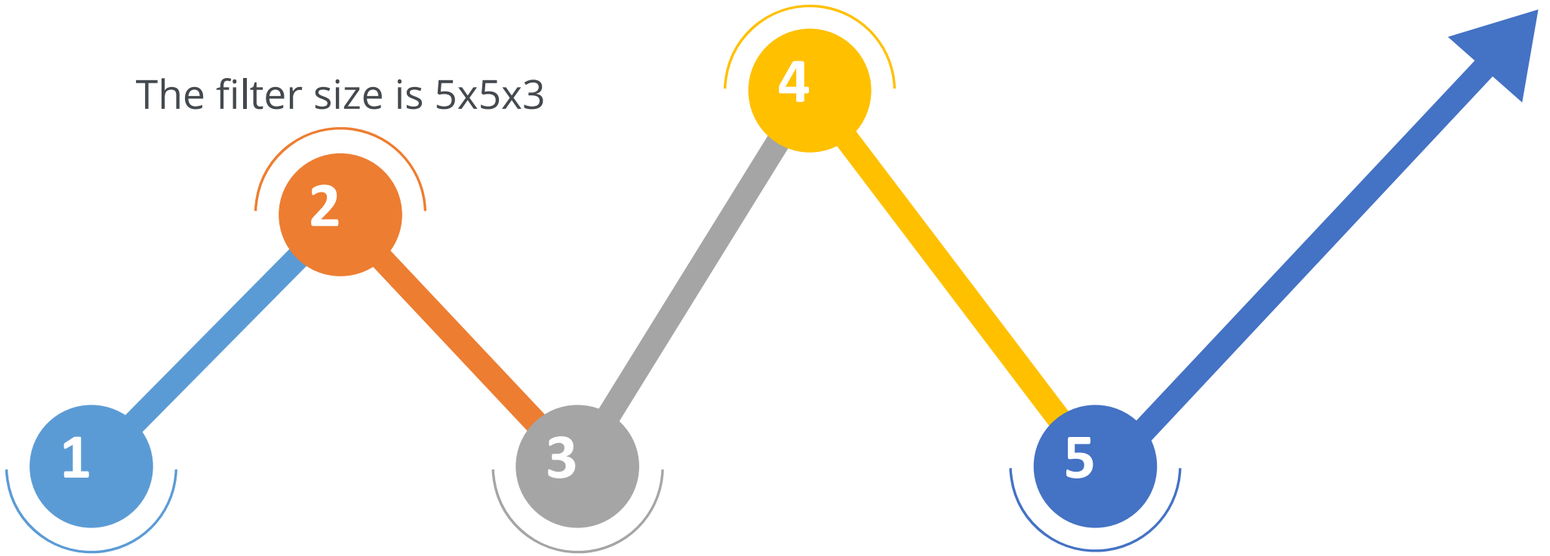
Zero Padding

CALCULATION

$$\text{Zero Padding} = \frac{(K-1)}{2}$$

where K= filter size

Therefore the input is zero padded with value of 2 resulting in a matrix of 36x36x3



The filter size is 5x5x3

Assume the input image to be of size 32x32x3

Required output image is 32x32x3

$$\text{Zero Padding} = \frac{(5-1)}{2}$$

$$= 2$$

Zero Padding

OUTPUT SIZE

- The output size can be calculated as:

$$O = \frac{(W - K + 2P)}{S} + 1$$

- Here W=32, K=5, P=2, S=1.
- Therefore the output size will be ,

$$\mathbf{O} = \frac{(\mathbf{32 - 5 + 2x2})}{\mathbf{1}} + \mathbf{1}$$

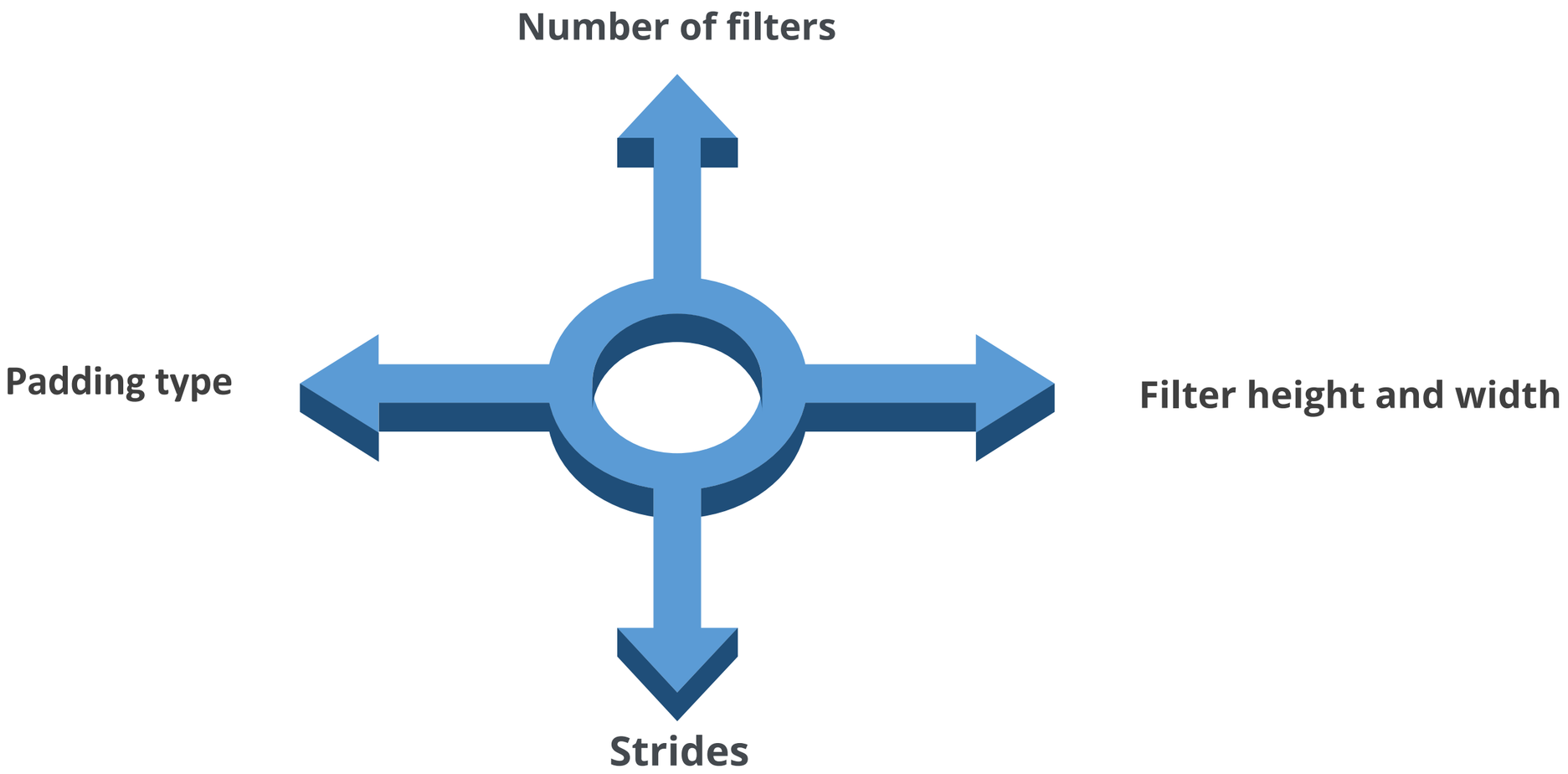
$$= 32$$

- Hence the output size will be the same as input i.e, 32x32x3.

Convolutional Layer

HYPERPARAMETERS

- The hyperparameters of CNN are:

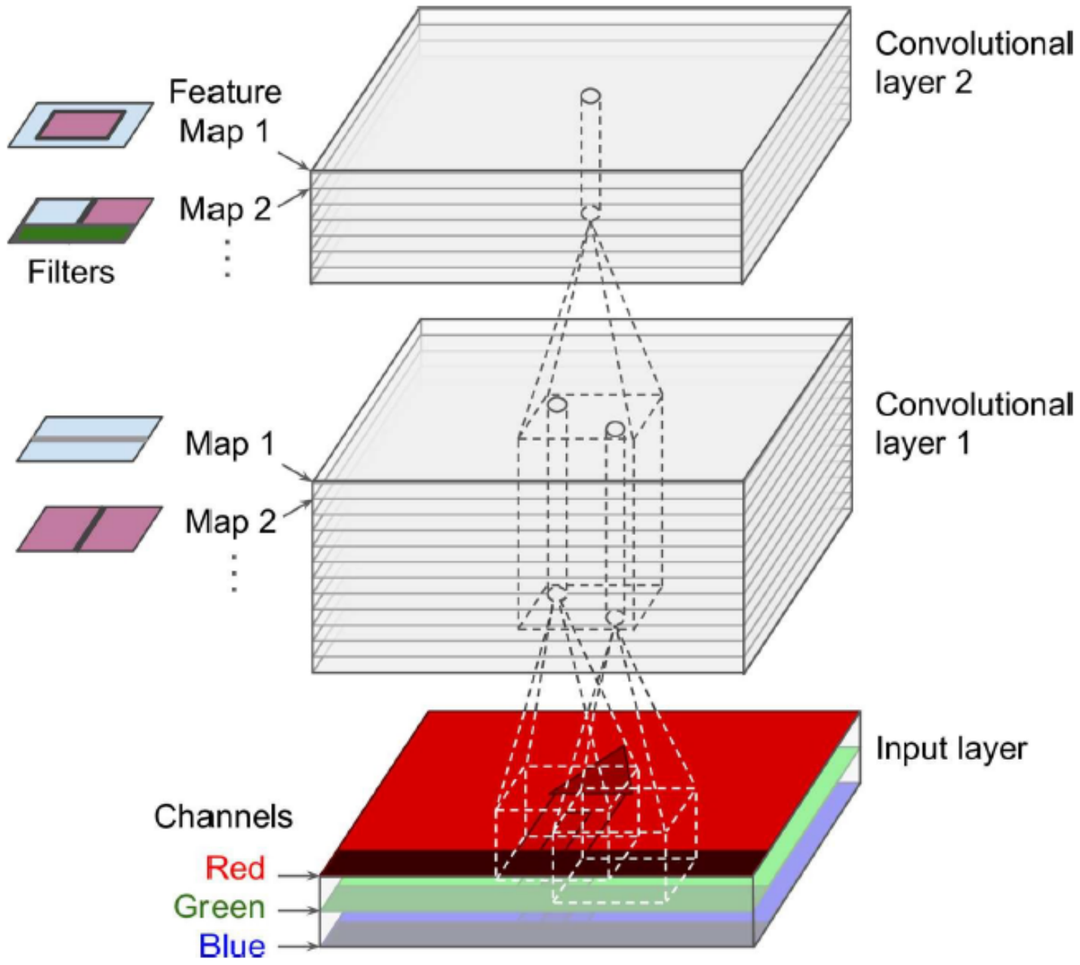


Convolutional Neural Network

Topic 5—Stacking Multiple Feature Maps

Stacking Multiple Feature Maps

- A single feature map has same parameters as a single kernel has the same weights/parameters as it moves across the image; however, different feature maps can have different parameters (weights + biases).
- The receptive field of previous layer extends across all of its feature maps.
- In summary, a convolutional layer applies multiple filters to the input image, making it capable of detecting multiple features in the input.

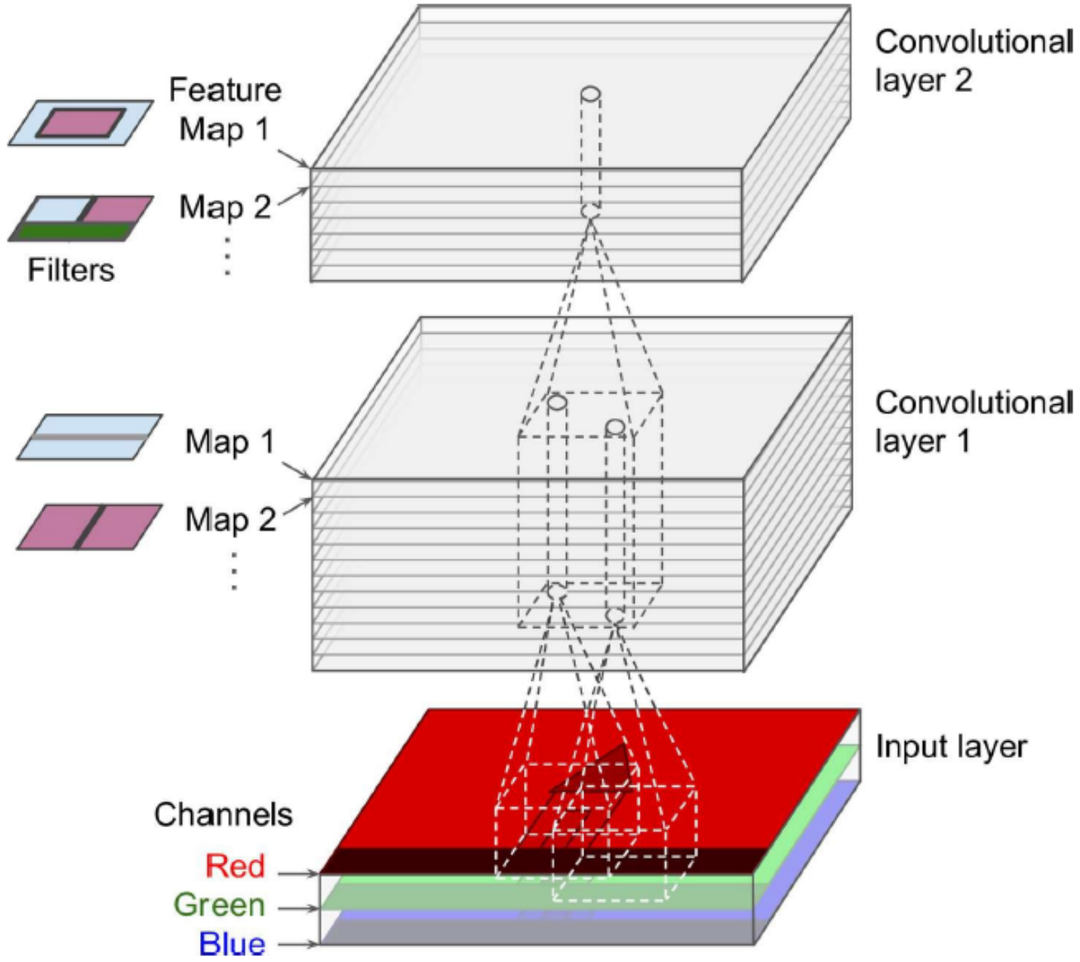


CNN layers with multiple feature maps

Image Source: "Hands-on Machine Learning with Scikit-Learn and TensorFlow " by Aurelien Geron

Stacking Multiple Feature Maps (Contd.)

- Images that are grayscale have just one channel. So it needs just 1 sublayer. Colored images have three channels – Red, Green and Blue. So it needs 3 sublayers.
- Satellite imagery that capture extra light frequencies (eg infrared) can have more channels.



CNN layers with multiple feature maps

Image Source: "Hands-on Machine Learning with Scikit-Learn and TensorFlow " by Aurelien Geron

Stacking Multiple Feature Maps (Contd.)

- The fact that all neurons in a feature map has just one set of parameters dramatically reduces the no of parameters needed.
- This also means that once a CNN has learned to recognize a pattern in one location, it can recognize it in any other location. This is known as location invariance.
- In contrast, if a regular DNN has learned to recognize a pattern in one location, it can recognize it only in that location.

Image Source: "Hands-on Machine Learning with Scikit-Learn and TensorFlow " by Aurelien Geron

Stacking Multiple Feature Maps

EQUATION

- The equation below shows the results of applying multiple feature maps.

- This equation specifically calculates the output of one cell in the convolutional layer after applying multiple feature maps to the preceding layer. In simpler terms, this calculation sums up the weighted inputs for all the feature maps stacked together.

$$z_{i,j,k} = b_k + \sum_{u=1}^{f_h} \sum_{v=1}^{f_w} \sum_{k=1}^{f_n} x_{i',j',k'} \cdot w_{u,v,k',k}$$

Stacking Multiple Feature Maps (Contd.)

EQUATION

b_k is the bias term for feature map *k* in layer *l*

$x_{i',j',k'}$ is the output of the neuron located in row *l*-1, row i',column j' in feature map k'

$$z_{i,j,k} = b_k + \sum_{u=1}^{f_h} \sum_{v=1}^{f_w} \sum_{k=1}^{f_n} x_{i',j',k'} \cdot w_{u,v,k',k}$$

where $i = u.s_h + f_h - 1$

and $j = v.s_w + f_w - 1$

$z_{i,j,k}$ is the output of the neuron located in row i, column j in feature map k of the convolutional layer (layer *l*)

s_h and s_w are the vertical and horizontal strides

f_h and f_w are the height and width of the receptive field. f_n is the number of feature maps in the previous layer, *l*-1

ReLU Layer

After every conventional layer, a non linearity is applied to the neural network.

In ANN, non linearity functions like sigmoid is used.

Research has found that ReLU layers can train the neural networks much faster than other activation functions.

ReLU layers apply an activation function, **f(x)= max(0,x),** which essentially drops all negative units.

Convolutional Neural Network

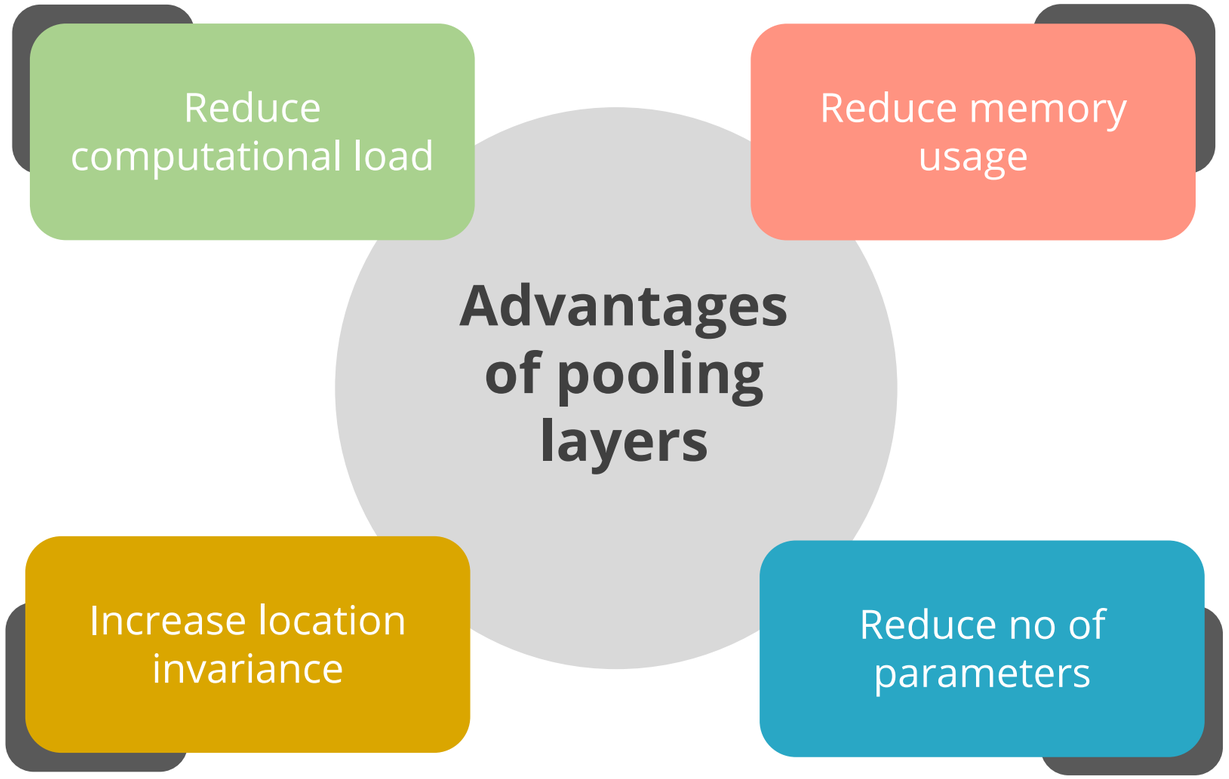
Topic 6—Pooling Layer and Fully Connected Layer

Pooling Layer

- A pooling layer is used to sub-sample (i.e., shrink) the input image.
- Like a convolutional layer, a pooling layer is connected to a small set of neurons in input image which fall within a receptive field.
- In detail, a pooling layer takes each feature map output from the convolutional layer and prepares a condensed feature map.
- This layer by itself does not have any weights.
- Its function is to progressively reduce the spatial size of the representation to reduce the weight of the parameters in the network. This also controls overfitting.
- A pooling layer has size, stride and padding type, as in case of a convolutional layer.

Pooling Layer

ADVANTAGES



Pooling Layer

MAX POOLING

- The figure shows a max pooling layer with 2x2 kernel, stride 2 and no padding.
- Max-pooling involves taking the maximum value out of a group of input values.
- Essentially it boils down to giving weightage to the largest of the values from among a group of values.
- Instead of height and weight pooling, one can pool over the depth dimension also, in which case the no of channels will reduce.

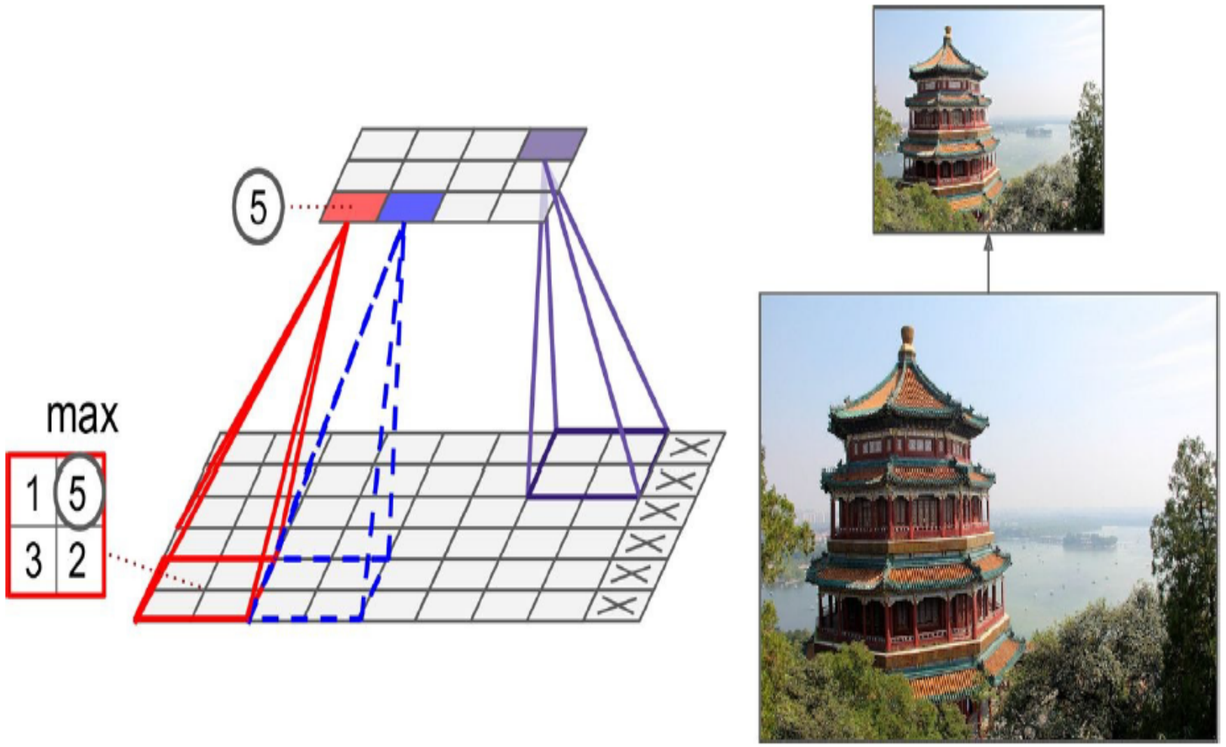


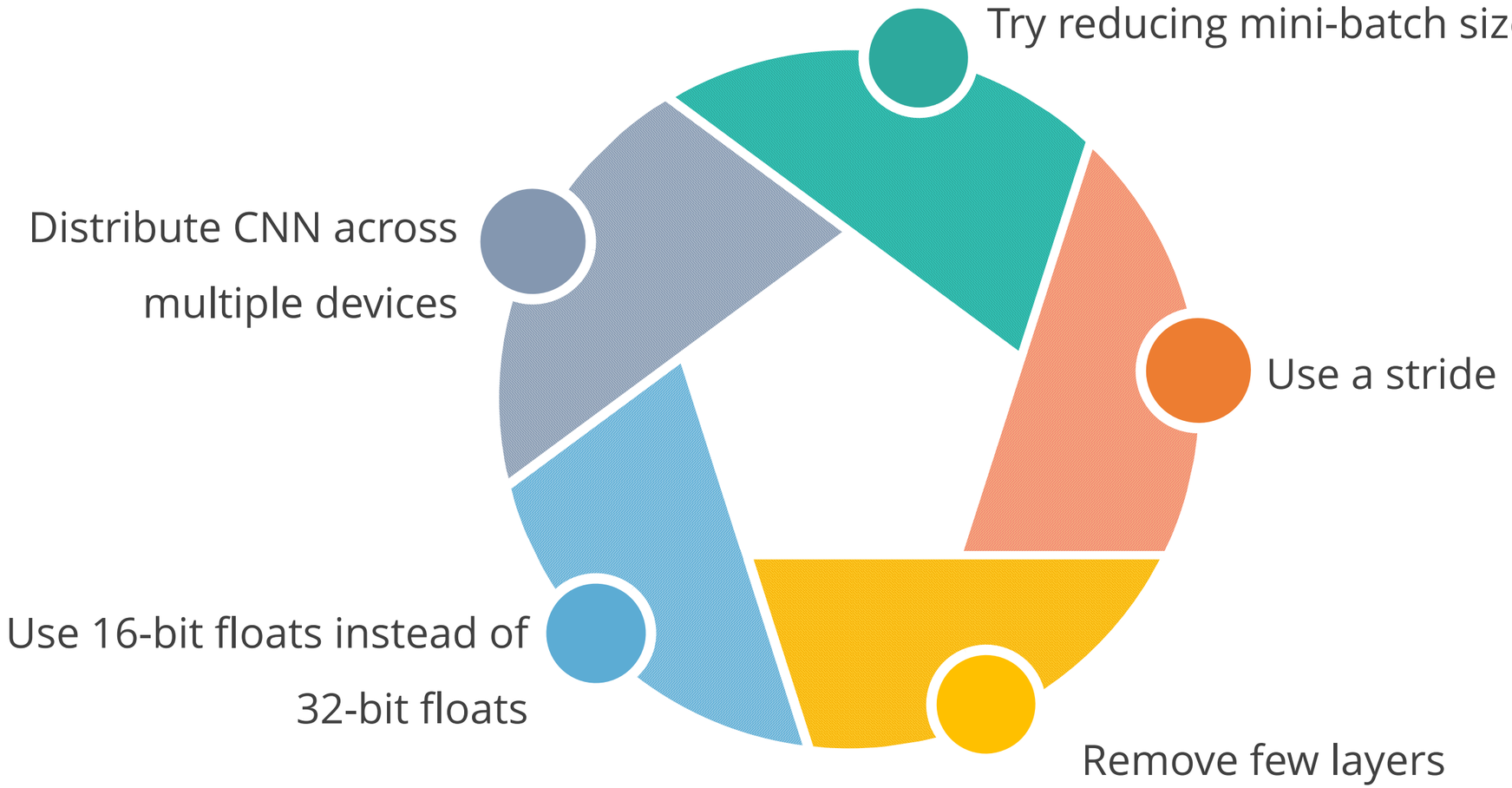
Image Source: "Hands-on Machine Learning with Scikit-Learn and TensorFlow " by Aurelien Geron

Fully Connected layer

- Typically CNN networks have a fully connected layer in the end to assemble all the information processed by the various CNN layers so far.
- The fully connected layer is usually followed by Softmax classifier, in case of classification functions e.g. to classify an input image as containing a person, animal or a vehicle.

Issues in CNN

- A major issue with CNNs is that they need a huge amount of RAM, particularly during backpropagation, where all computations made during forward pass have to be preserved.



Convolutional Neural Network

Topic 7—CNNs in TensorFlow

CNNs in TensorFlow

- In Tensorflow, each input image is a tensor of shape [height, width, channels].
- A mini-batch is represented as a 4D tensor of shape [mini-batch size, height, width, channels].

A mini-batch is the subset of the input training data. It is used to train the model in smaller batches rather than feeding the entire training data together. It provides a nice compromise between training with individual data sample one at a time vs training with all the data samples together in one go.

- The weights of a convolutional layer are represented as a 4D tensor of shape [f_h , f_w , f_n , $f_{n'}$]. The bias terms of a convolutional layer are simply represented as a 1D tensor of shape [f_n].

CNNs in TensorFlow

CODING

- This code loads two color images, one of a Chinese temple and other of a flower.
- Then it creates two 7x7 filters, one with horizontal white line and other with vertical white line.
- TF *conv2d* is a convolutional layer, with zero padding (add padding if needed) and stride of 2 in this case.
- *strides* is a 4 element 1D array, where two centre elements represent s_h and s_w.

```
# Load sample images
dataset = np.array(load_sample_images().images, dtype=np.float32)
batch_size, height, width, channels = dataset.shape

# Create 2 filters
filters_test = np.zeros(shape=(7, 7, channels, 2), dtype=np.float32)
filters_test[:, 3, :, 0] = 1  # vertical line
filters_test[3, :, :, 1] = 1  # horizontal line

# Create a graph with input X plus a convolutional layer applying the 2 filters
X = tf.placeholder(tf.float32, shape=(None, height, width, channels))
convolution = tf.nn.conv2d(X, filters, strides=[1,2,2,1], padding="SAME")

with tf.Session() as sess:
    output = sess.run(convolution, feed_dict={X: dataset})

plt.imshow(output[0, :, :, 1])  # plot 1st image's 2nd feature map
plt.show()
```

CNNs in TensorFlow

POOLING LAYER

- The following code creates a max pooling layer using a 2 × 2 kernel, stride 2, and no padding, then applies it to all the images in the dataset:

```
[...] # load the image dataset, just like above
# Create a graph with input X plus a max pooling layer
X = tf.placeholder(tf.float32, shape=(None, height, width, channels))
max_pool = tf.nn.max_pool(X, ksize=[1,2,2,1], strides=[1,2,2,1],padding="VALID")

with tf.Session() as sess:
    output = sess.run(max_pool, feed_dict={X: dataset})

plt.imshow(output[0].astype(np.uint8))  # plot the output for the 1st image
plt.show()
```

Padding=“SAME” means apply padding if needed. Padding=“VALID” means do not apply padding and some rows / columns may be ignored if needed.

CNNs in TensorFlow (Contd.)

POOLING LAYER

```
[...] # load the image dataset, just like above
# Create a graph with input X plus a max pooling layer
X = tf.placeholder(tf.float32, shape=(None, height, width, channels))
max_pool = tf.nn.max_pool(X, ksize=[1,2,2,1], strides=[1,2,2,1],padding="VALID")

with tf.Session() as sess:
    output = sess.run(max_pool, feed_dict={X: dataset})

plt.imshow(output[0].astype(np.uint8))  # plot the output for the 1st image
plt.show()
```

- The ksize argument contains the kernel shape along all four dimensions of the input tensor: [mini-batch size, height, width, channels].
- Either height and weight are 1 or channels is 1, depending on whether the pooling is across height and weight or if the pooling is across channels only.
- To create an average pooling layer, just use the avg_pool() function instead of max_pool().

CNNs in TensorFlow

OTHER FORMS

- Apart from conv2d(), TensorFlow offers other kinds of convolutional layers also, some of which are:

  - conv1d() creates a convolutional layer for 1D inputs. This is useful, for example, in natural language processing, where a sentence may be represented as a 1D array of words, and the receptive field covers a few neighbouring words.
  - conv3d() creates a convolutional layer for 3D inputs, such as 3D PET scan.

Image Source: "Hands-on Machine Learning with Scikit-Learn and TensorFlow " by Aurelien Geron

Demo 1

Demonstrate how to create CNNs in TensorFlow

- **Objective**: Demonstrate various aspects of CNN with TensorFlow.
- **Steps:**
  1. Load china.jpg and flower.jpg images from Python dataset module.
  2. Develop a CNN to apply a vertical and horizontal 7x7 kernel filter to highlight vertical and horizontal lines respectively in the images loaded with SAME padding. Plot the output for the two images.
  3. Try using the higher level TensorFlow layers API for convolution and check the output image.
  4. Using a dataset of numbers between 1 and 13, try to apply a prime number filter first with SAME and then with VALID padding. Check the output of this convolutional layer in the two cases.
  5. Create a max pooling layer using a 2x2 kernel, stride 2 and no padding. Apply this to the images in the china image dataset. Plot the output of the pooling layer.
- **Dataset used:** China and Flower images from Python dataset module.
- **Skills required:** In-depth understanding of CNNs and all their features like kernel, stride, padding, pooling etc.

Demo 2

CNNs for MNIST image classifcation

- **Objective**: Use CNNs for MNIST image classification.

- **Steps:**
  1. Apply CNNs for MNIST image classification. Print training and test accuracy of the trained model.

- **Dataset used:** MNIST dataset to test a CNN-based classification network.
- **Skills required:** In-depth understanding of CNNs and all their features like kernel, stride, padding, pooling etc.

Key Takeaways

- CNNs are neural networks that contain at least 1 convolutional layer. They are majorly used for image classification.
- The convolution operation involves detecting local features in a small receptive field in the data space by means of multiplying the values in the receptive field by a kernel filter (weight matrix).
- Zero padding is done when it is required to maintain the same dimensions of the layer as the previous convolutional layers.
- CNNs can detect multiple features in the input by use of stacked feature maps.
- Pooling allows CNNs to shrink the input space and zero in on the most important features of the input.
- Typical CNN architectures involve multiple CNN layers to detect low and high level features going from input to output. Each CNN layer is typically followed by a ReLU and pooling layer.

This concludes “Convolutional Layer Network”

The next lesson is “Other Forms Of Deep Learning”