Convolutional Neural Network

Dr. Mongkol Ekpanyapong

CNN Example

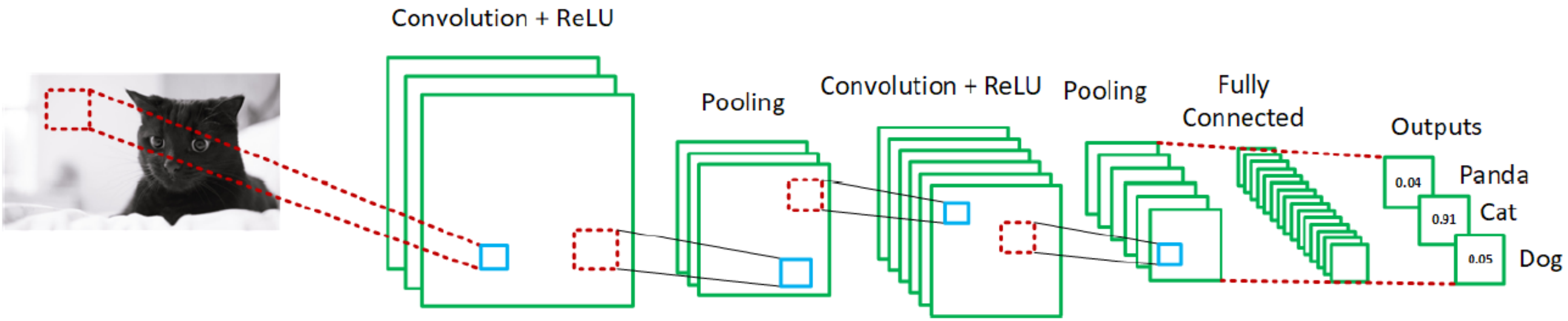
```
from tensorflow.keras.datasets import mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
test_images.shape
len(test_labels)
from tensorflow import keras
from tensorflow.keras import layers
img_rows = train_images[0].shape[0]
img_cols = train_images[0].shape[1]
model = keras.Sequential([
layers.Conv2D(64, (3, 3), activation='relu', input_shape=(28, 28,1)),
layers.MaxPooling2D((2, 2)),
layers.Conv2D(64, (3, 3), activation='relu'),
layers.MaxPooling2D((2, 2)),
layers.Flatten(),
layers.Dense(10, activation='softmax')
])

model.summary()
```

```
model.compile(optimizer="sgd",
loss="sparse_categorical_crossentropy",
metrics=["accuracy"])
train_images = train_images.reshape(train_images.shape[0], img_rows,
img_cols, 1)
test_images = test_images.reshape(test_images.shape[0], img_rows,
img_cols, 1)

train_images = train_images.astype("float32") / 255.0

test_images = test_images.astype("float32") / 255.0

print(train_images.shape)

model.fit(train_images, train_labels, epochs=5, batch_size=128)
```

Homework

- Modify CNN model in Keras on MNIST data set so that the accuracy is better than 98%

Layers of a CNN



Fully Connected Network (FCN)

- Fully connected layers that takes in a 256x256 images and maps to 10 output neural will have 256x256x10+1 = 655,360+1 parameters
- Hence, the FCN model is more complex
- FCN is tended to be more overfit

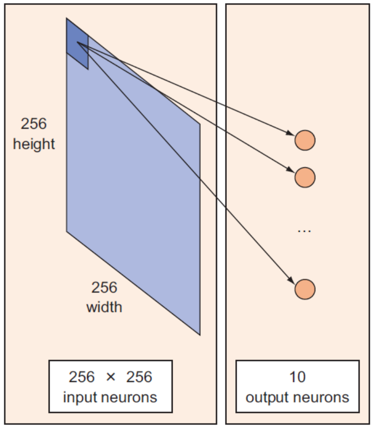


Image from Machine Learning with TensorFlow book

FCN/CNN layer Comparison

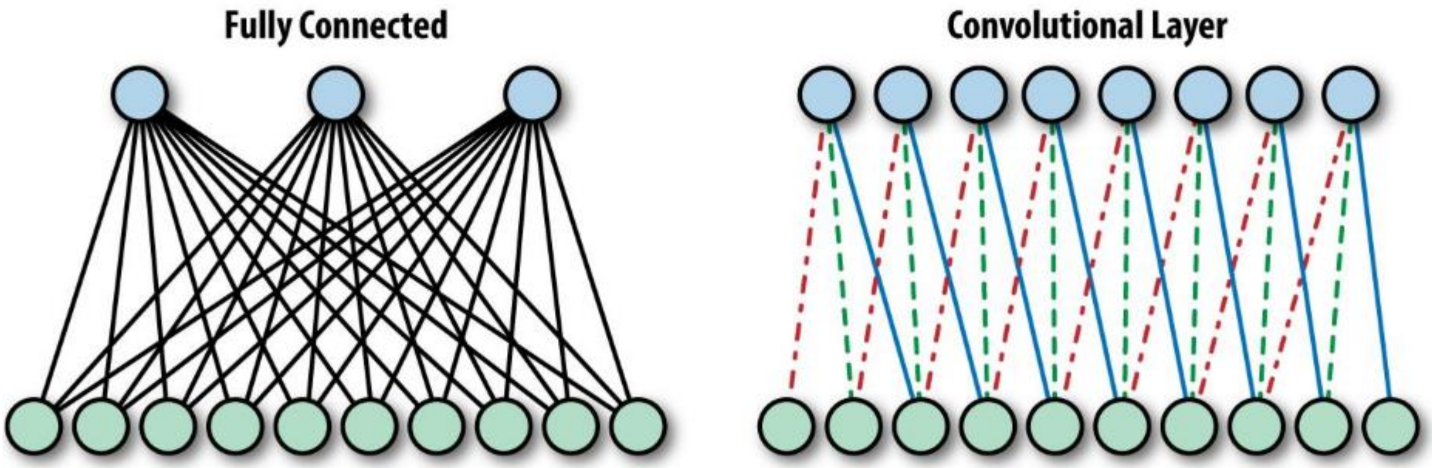


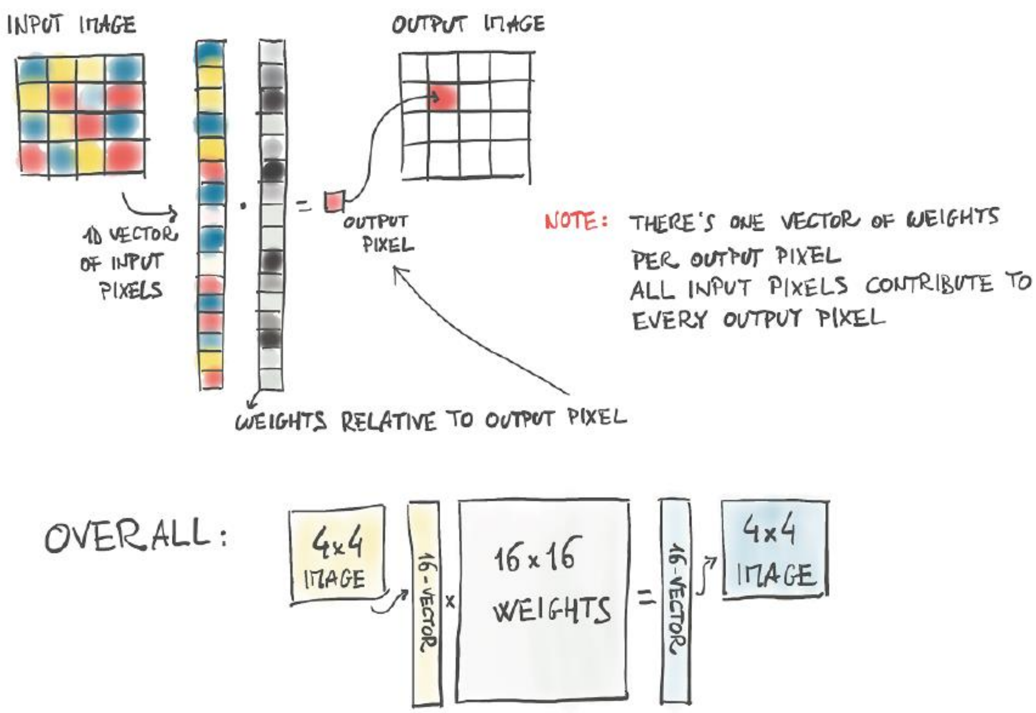
Image from Learning TensorFlow book

The Limit of Fully Connected

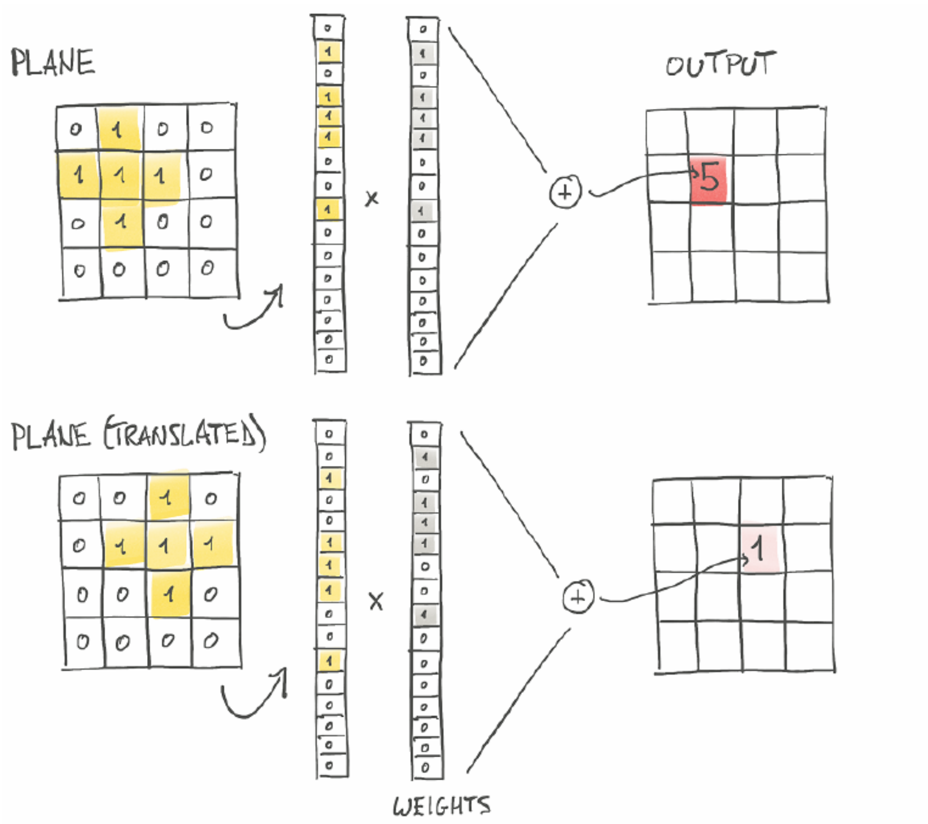
- Higher number of parameters
- We use information from far-away pixel during the prediction
- Non-translation invariant

The Limit of Fully Connected

- Every input pixel is combined with every other to produce the output results in large number of parameters



Not-Translation Invariant



CNN

- A convolution is a weighted sum of the pixel values of the image as the window slides across the whole image
- The difference between a fully connected layer and a convolution layer is that the fully connected layers learn global patterns whereas convolution layers learn local pattern
- Convolution operates over 3D tensor called feature map

Key Characteristic of CNN

- The patterns they learn are translation invariant (FCN is not)
- They can learn spatial hierarchies of patterns
- Also, it should learn about the filter automatically

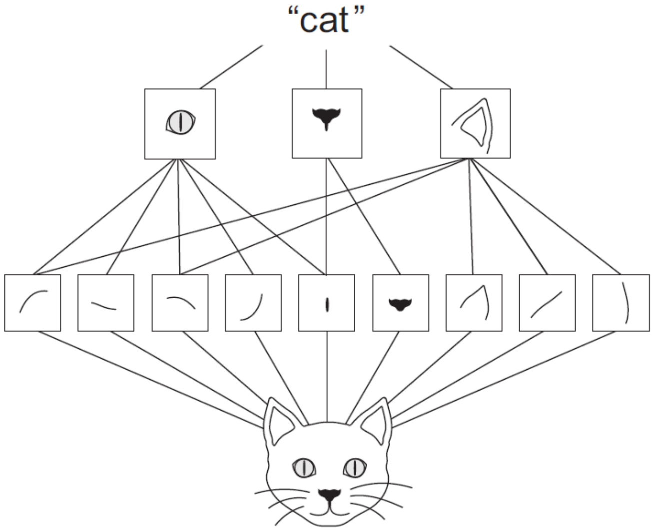
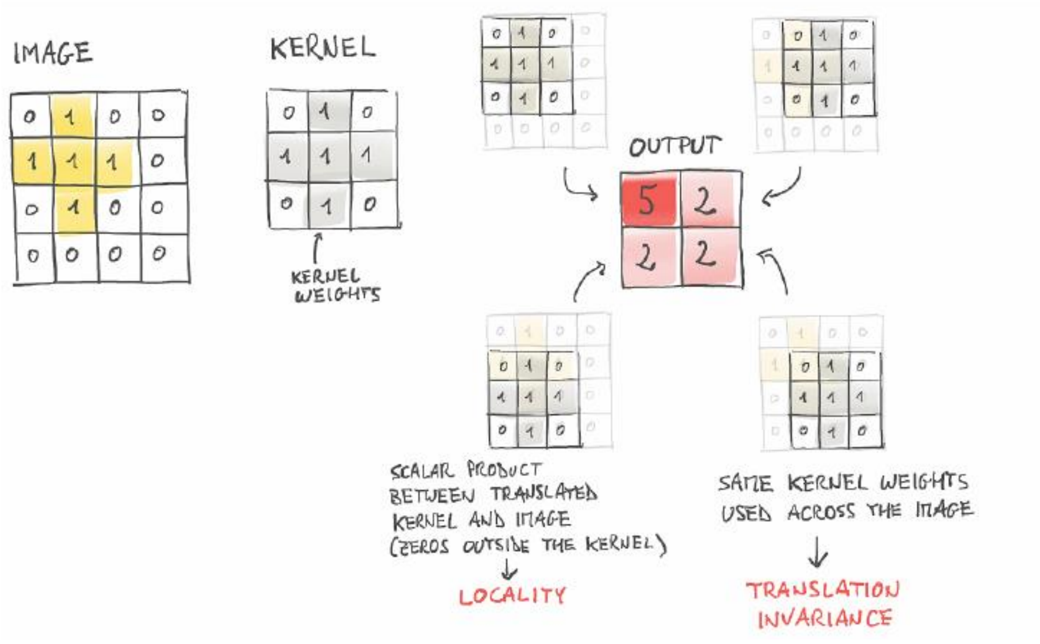


Image from Deep Learning with Python

CNN

- Locality and translation invariance



Convolutional Neural Networks

- CNN has just enough weights to look at a small patch of the image
- The number of parameters are reduced to just 5 x 5+1 = 25+1 parameters per node

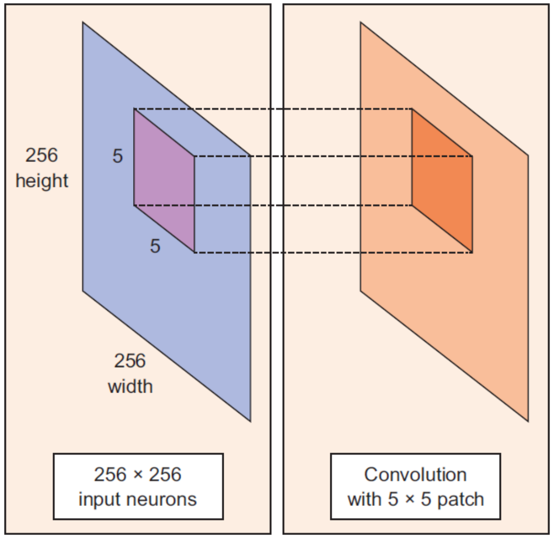


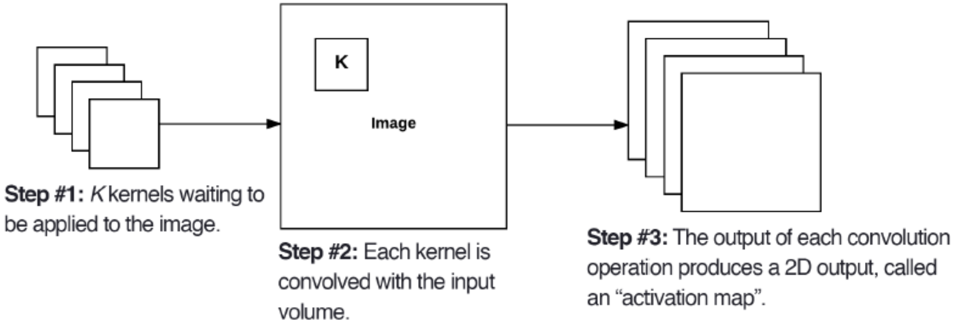
Image from Machine Learning with TensorFlow book

CNN Layer Types

- Convolutional (CONV)
- Activation (ACT) e.g., RELU or SOFTMAX
- Pooling (POOL)
- Fully-connected (FC)
- Batch Normalization (BN)
- DropOut (DO)

CONV layers

- Instead of fully connected layer, we can use convolutional layer instead
- Convolutional layer introduces the local connectivity and reduces the number of parameters for training



Example

- Given a grayscale image of size 256x256,

It connects to 25 output neurons, the number of parameters is

256x256x25+bias =

1,638,400+1 for a

fully connected layer

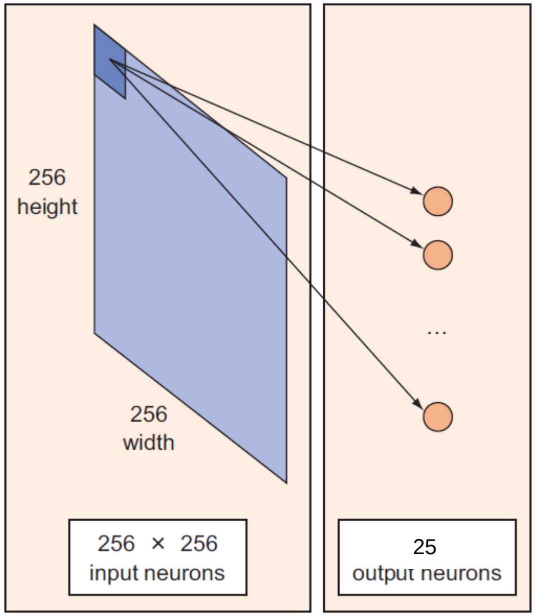


Image from Machine Learning TensorFlow book

Example with CONV

- Given a grayscale image of size 256x256,

It connects to 5x5 patch, the number of parameters is

5x5+1 = 25+1 for a convolutional layer

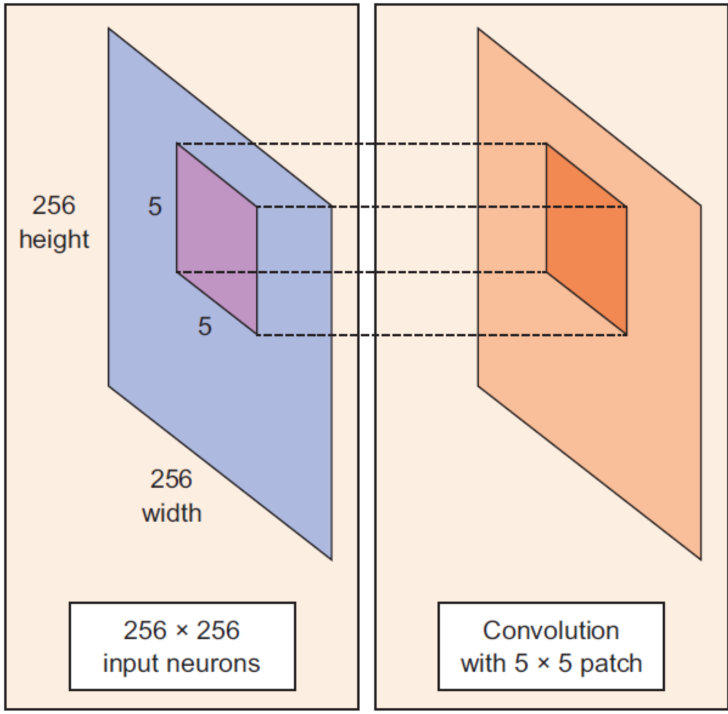


Image from Machine Learning TensorFlow book

Backpropagation

- We can consider the weight/parameter of each kernel similar to fully connected layer as shown below in which sigma is the activation function

$$\sigma\left(b + \sum_{l=0}^{4}\sum_{m=0}^{4} w_{l,m} a_{j+l,k+m}\right)$$

- Hence, the same back propagation can be applied in which now the convolution layer will learn the filters to be used

Multiple Feature map

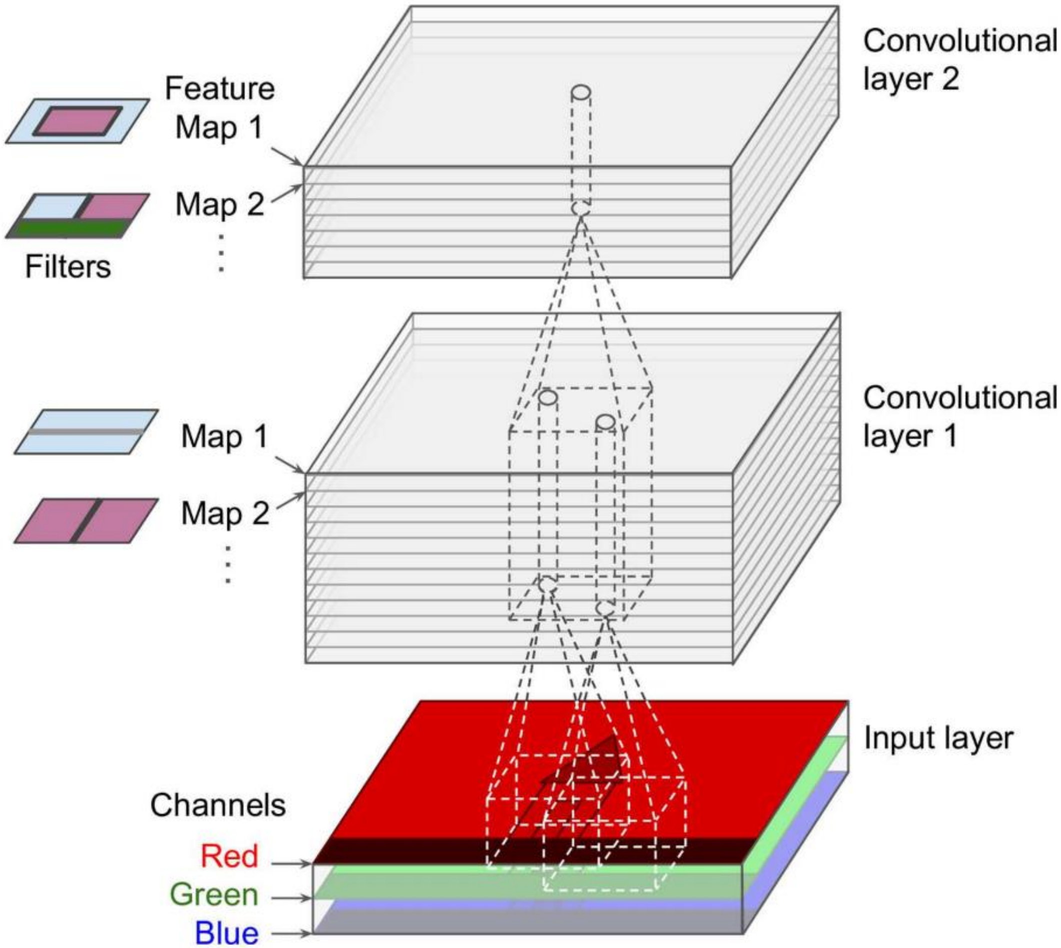


Image from Hands-on Machine Learning with SciKit-Learn and TensorFlow

Local Patterns in Image

- Image can be broken into local pattern such as edges, textures

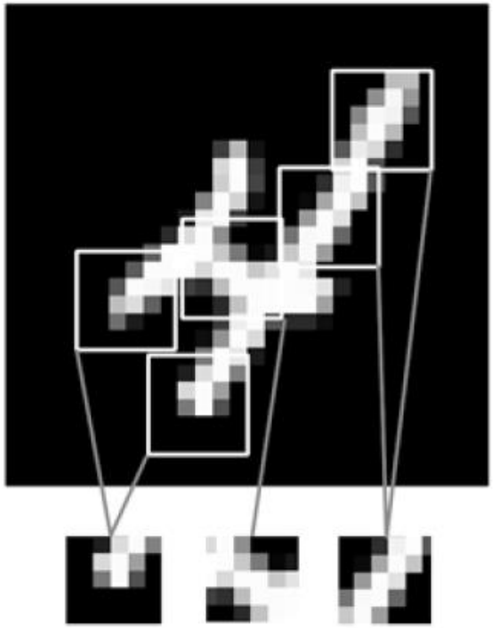


Image from Deep Learning with Python book

CNN Hyper-Parameters

- Number of convolution layers
- Convolution window size
- Convolution filter mask (Filter Depth=K)
- Number of stride
- Padding

Example of random initialized matrices for 32 filter

- The filters to be learnt by CNN

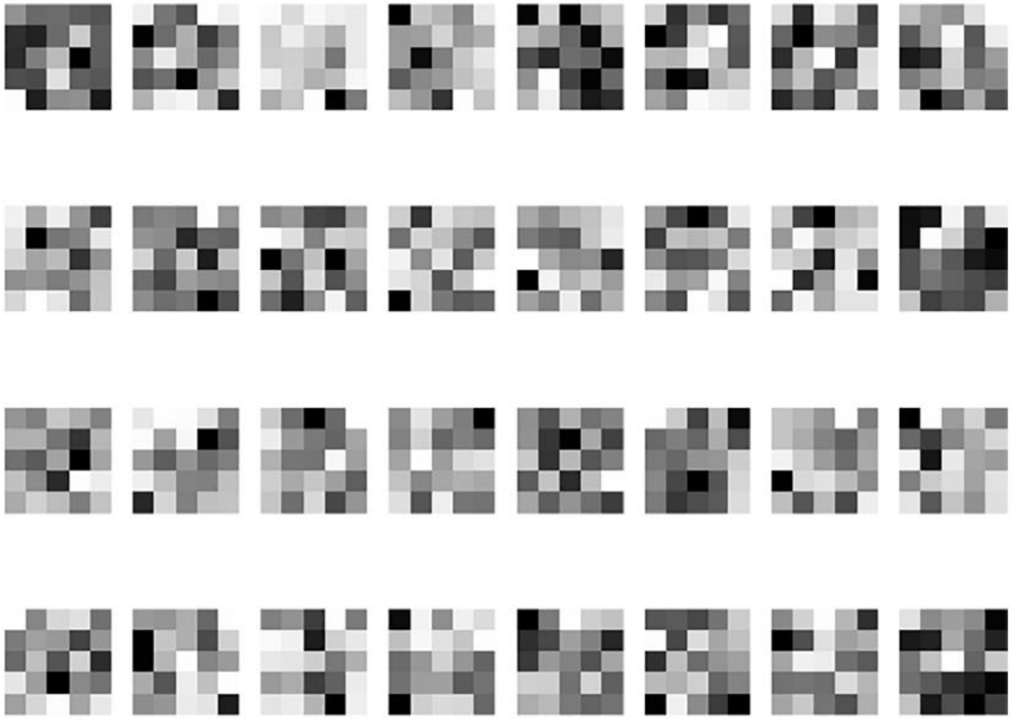
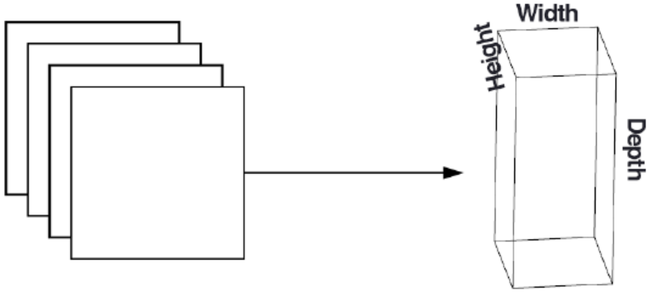


Image from Learning TensorFlow book

K filter/Activation Map

- After we apply K filter, we get the the volume of activation/feature map for the next layer
- Note that the depth can be more than K, as the input can have many channels



K

Filter Results

- We call it Feature Map or Activation Map

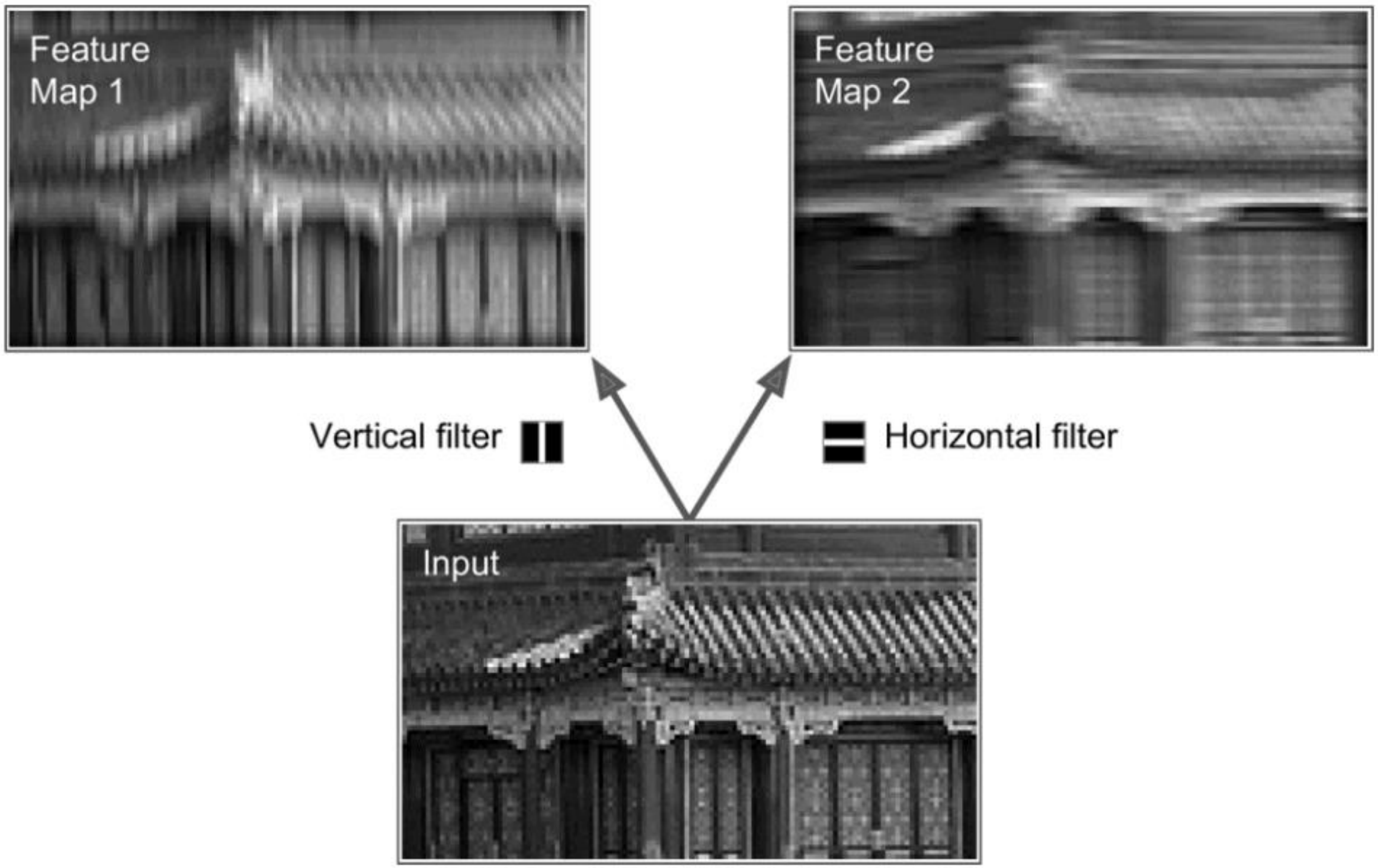
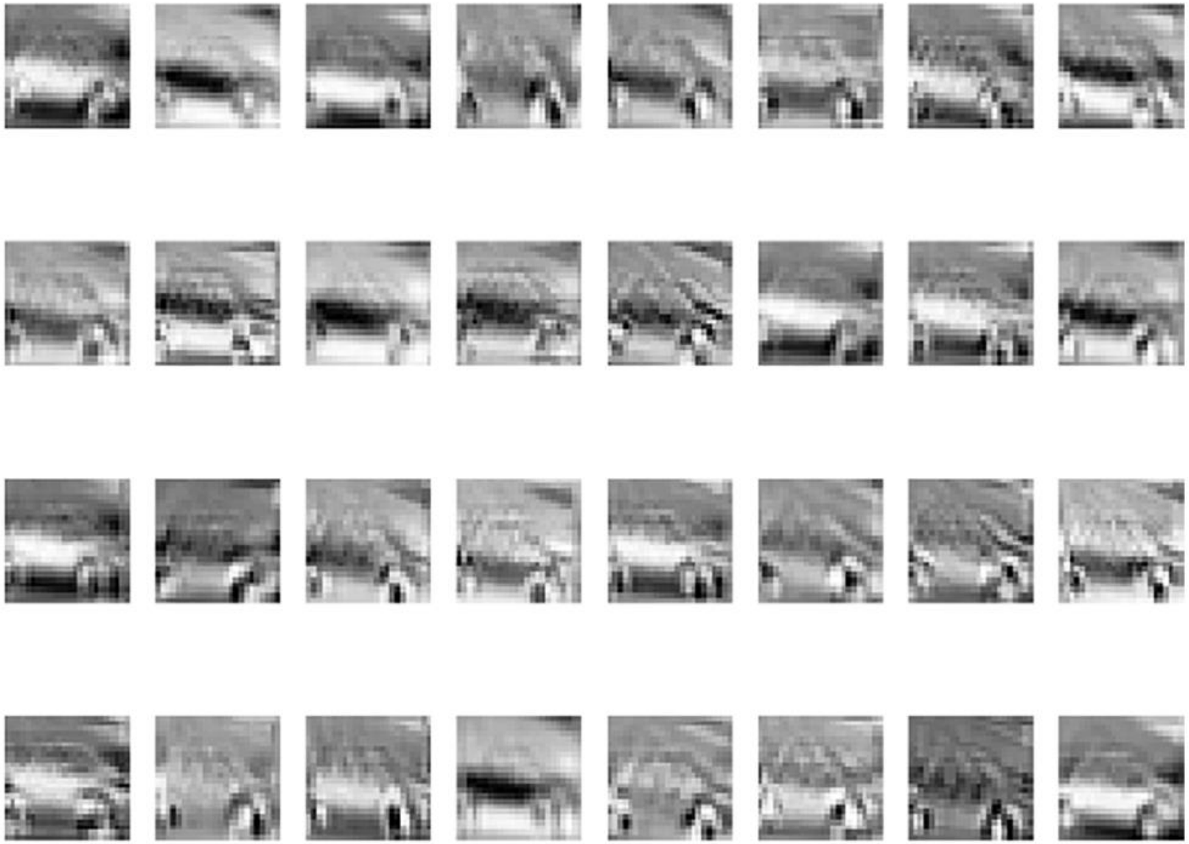
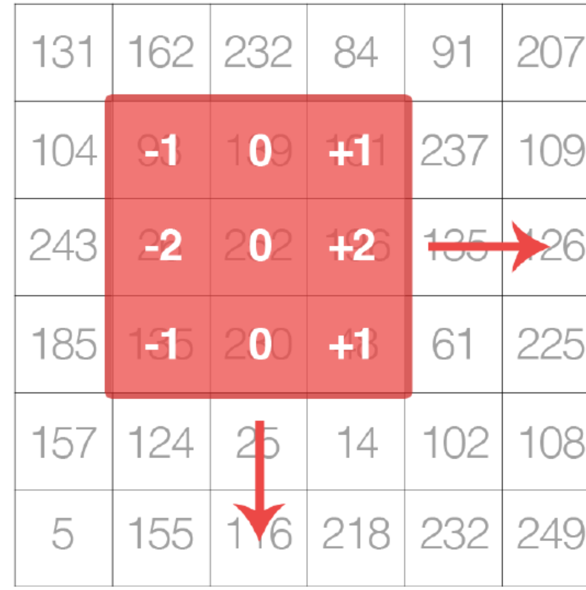


Image from Hands-on Machine Learning with SciKit-Learn and TensorFlow

Filter Result by applying on a car



Stride



- The stride is a step of sliding of small matrix over the image (big matrix)
- Example of image (left) and filter (right)

95	242	186	152	39
39	14	220	153	180
5	247	212	54	46
46	77	133	110	74
156	35	74	93	116

0	1	0
1	-4	1
0	1	0

- Output of a convolution with 1x1 stride (left) and 2x2 stride (right)

692	-315	-6
-680	-194	305
153	-59	-86

692	-6
153	-86

Padding

- Padding is a technique to retain the original image size when applying a convolution
- In Tensorflow framework, only zero padding is provided
- From the figure, top left is the kernel, top right is the image

692	-315	-6
-680	-194	305
153	-59	-86

0	0	0	0	0	0	0
0	95	242	186	152	39	0
0	39	14	220	153	180	0
0	5	247	212	54	46	0
0	46	77	133	110	74	0
0	156	35	74	93	116	0
0	0	0	0	0	0	0

-99	-673	-130	-230	176
-42	692	-315	-6	-482
312	-680	-194	305	124
54	153	-59	-86	-24
-543	167	-35	-72	-297

Padding Example

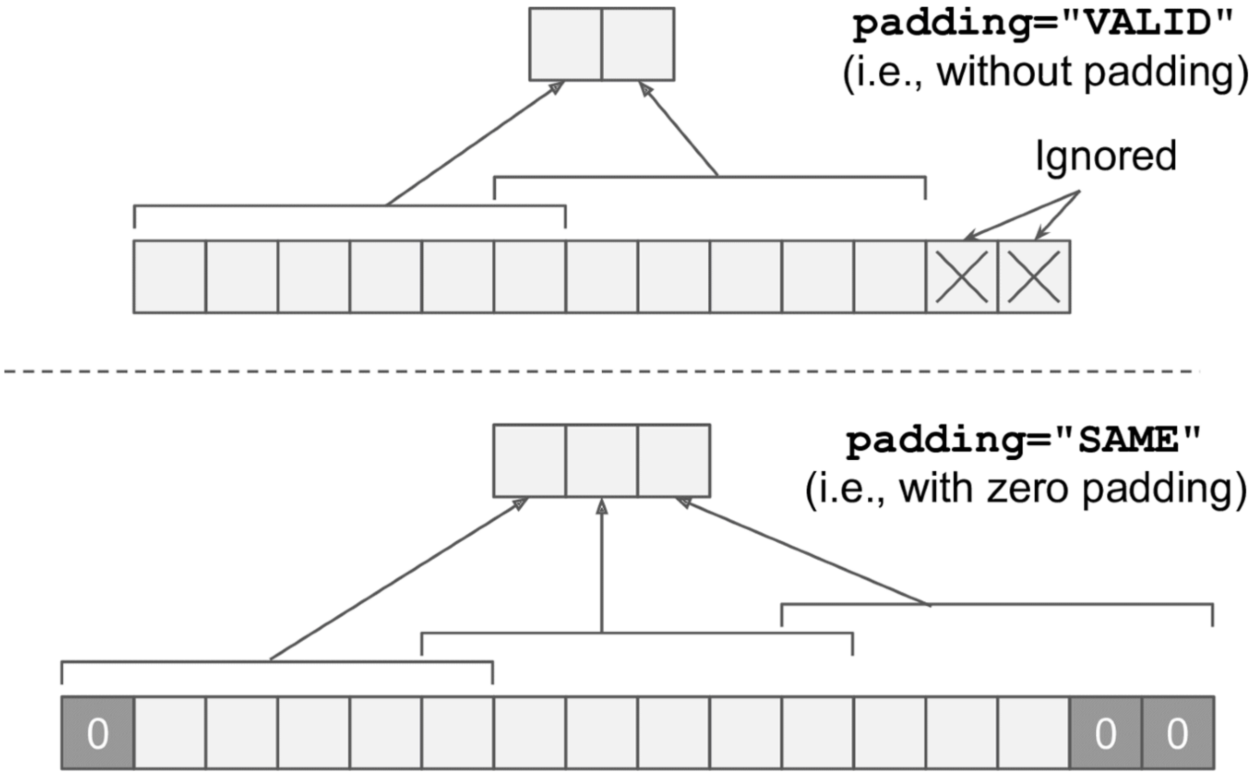
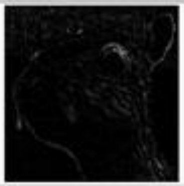


Image from Hands-on Machine Learning with SciKit-Learn and TensorFlow

Convolution Recap

- Convolution Filters or Kernels detect features in images

Detected features

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

What is the Output?

1	0	1	0	1
1	0	0	1	1
0	1	1	0	0
1	0	0	1	0
0	0	1	1	0

Input Image

*

0	1	0
1	0	-1
0	1	0

Filter or Kernel

Recap

$$(1x0) + (0x1) + (1x0)+(1x1) + (0x0) + (0x-1) + (0x0) +(1x1) + (1x0) = 2$$

1x0	0x1	1x0	0	1
1x1	0x0	0x-1	1	1
0x0	1x1	1x0	0	0
1	0	0	1	0
0	0	1	1	0

Input Image

*

0	1	0
1	0	-1
0	1	0

Filter or Kernel

=

2		

Output or Feature Map

$$(0x0) + (1x1) + (0x0)+(0x1) + (0x0)+(1x\text{-}1) + (1x0) + (1x1) + (0x0) = 1$$

1	0x0	1x1	0x0	1
1	0x1	0x0	1x-1	1
0	1x0	1x1	0x0	0
1	0	0	1	0
0	0	1	1	0

Input Image

*

0	1	0
1	0	-1
0	1	0

Filter or Kernel

=

2	1	

Output or Feature Map

$$\begin{bmatrix}1&0&1&0&1\\1&0&0&1&1\\0&1&1&0&0\\1&0&0&1&0\\0&0&1&1&0\end{bmatrix} * \begin{bmatrix}0&1&0\\1&0&-1\\0&1&0\end{bmatrix} = \begin{bmatrix}2&1&-1\\ & & \\ & & \end{bmatrix}$$

Input Image

Filter or Kernel

Output or Feature Map

1	0	1	0	1
1	0	0	1	1
0	1	1	0	0
1	0	0	1	0
0	0	1	1	0

Input Image

*

0	1	0
1	0	-1
0	1	0

Filter or Kernel

=

2	1	-1
-1		

Output or Feature Map

1	0	1	0	1
1	0	0	1	1
0	1	1	0	0
1	0	0	1	0
0	0	1	1	0

Input Image

*

0	1	0
1	0	-1
0	1	0

Filter or Kernel

=

2	1	-1
-1	1	

Output or Feature Map

1	0	1	0	1
1	0	0	1	1
0	1	1	0	0
1	0	0	1	0
0	0	1	1	0

Input Image

*

0	1	0
1	0	-1
0	1	0

Filter or Kernel

=

2	1	-1
-1	1	3

Output or Feature Map

$$
\begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}
*
\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}
=
\begin{bmatrix} 2 & 1 & -1 \\ -1 & 1 & 3 \\ 2 & & \end{bmatrix}
$$

Input Image

Filter or Kernel

Output or Feature Map

1	0	1	0	1
1	0	0	1	1
0	1	1	0	0
1	0	0	1	0
0	0	1	1	0

Input Image

*

0	1	0
1	0	-1
0	1	0

Filter or Kernel

=

2	1	-1
-1	1	3
2	1	

Output or Feature Map

$$\begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 2 & 1 & -1 \\ -1 & 1 & 3 \\ 2 & 1 & 1 \end{bmatrix}$$

Input Image | Filter or Kernel | Output or Feature Map

Input 5x5 with filter 3x3, then we get 3x3 output

Calculating Feature Map Size

Feature Map Size = $n - f + 1 = m$

Feature Map Size = $5 - 3 + 1 = 3$

$$\begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 2 & 1 & -1 \\ -1 & 1 & 3 \\ 2 & 1 & 1 \end{bmatrix}$$

5 x 5 — $n \times n$

3 x 3 — $f \times f$

3 x 3 — $m \times m$

Stride

- It can use to control feature map output
- Larger stride has less overlap

Stride of 2

1	0	1	0	1
1	0	0	1	1
0	1	1	0	0
1	0	0	1	0
0	0	1	1	0

Input Image

*

0	1	0
1	0	-1
0	1	0

Filter or Kernel

=

2	

Output or Feature Map

Shift by 2:

$$\begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 2 & -1 \\ & \end{bmatrix}$$

Input Image | Filter or Kernel | Output or Feature Map

1	0	1	0	1
1	0	0	1	1
0	1	1	0	0
1	0	0	1	0
0	0	1	1	0

Input Image

*

0	1	0
1	0	-1
0	1	0

Filter or Kernel

=

2	-1
2	

Output or Feature Map

1	0	1	0	1
1	0	0	1	1
0	1	1	0	0
1	0	0	1	0
0	0	1	1	0

Input Image

*

0	1	0
1	0	-1
0	1	0

Filter or Kernel

=

2	-1
2	1

Output or Feature Map

Stride = 1

Padding = 0

1	0	1	0	1
1	0	0	1	1
0	1	1	0	0
1	0	0	1	0
0	0	1	1	0

Input Image
5 x 5

*

0	1	0
1	0	-1
0	1	0

Filter or Kernel
3 x 3

=

2	-1	1
2	1	0
1	-1	1

$$(n \times n) * (f \times f) = (\frac{n + 2p - f}{s} + 1) \times (\frac{n + 2p - f}{s} + 1) = (\frac{5 + (2 \times 0) - 3}{1} + 1) \times (\frac{5 + (2 \times 0) - 3}{1} + 1) = 3 \times 3$$

Stride = 2

Padding = 0

$$\begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 2 & -1 \\ 2 & 1 \end{bmatrix}$$

Input Image
5 x 5

Filter or Kernel
3 x 3

$$(n \times n) * (f \times f) = \left(\frac{n + 2p - f}{s} + 1\right) \times \left(\frac{n + 2p - f}{s} + 1\right) = \left(\frac{5 + (2 \times 0) - 3}{2} + 1\right) \times \left(\frac{5 + (2 \times 0) - 3}{2} + 1\right) = 2 \times 2$$

Padding

Padding = 1

0	0	0	0	0	0	0
0	1	0	1	0	1	0
0	1	0	0	1	1	0
0	0	1	1	0	0	0
0	1	0	0	1	0	0
0	0	0	1	1	0	0
0	0	0	0	0	0	0

Input Image
7 x 7

*

0	1	0
1	0	-1
0	1	0

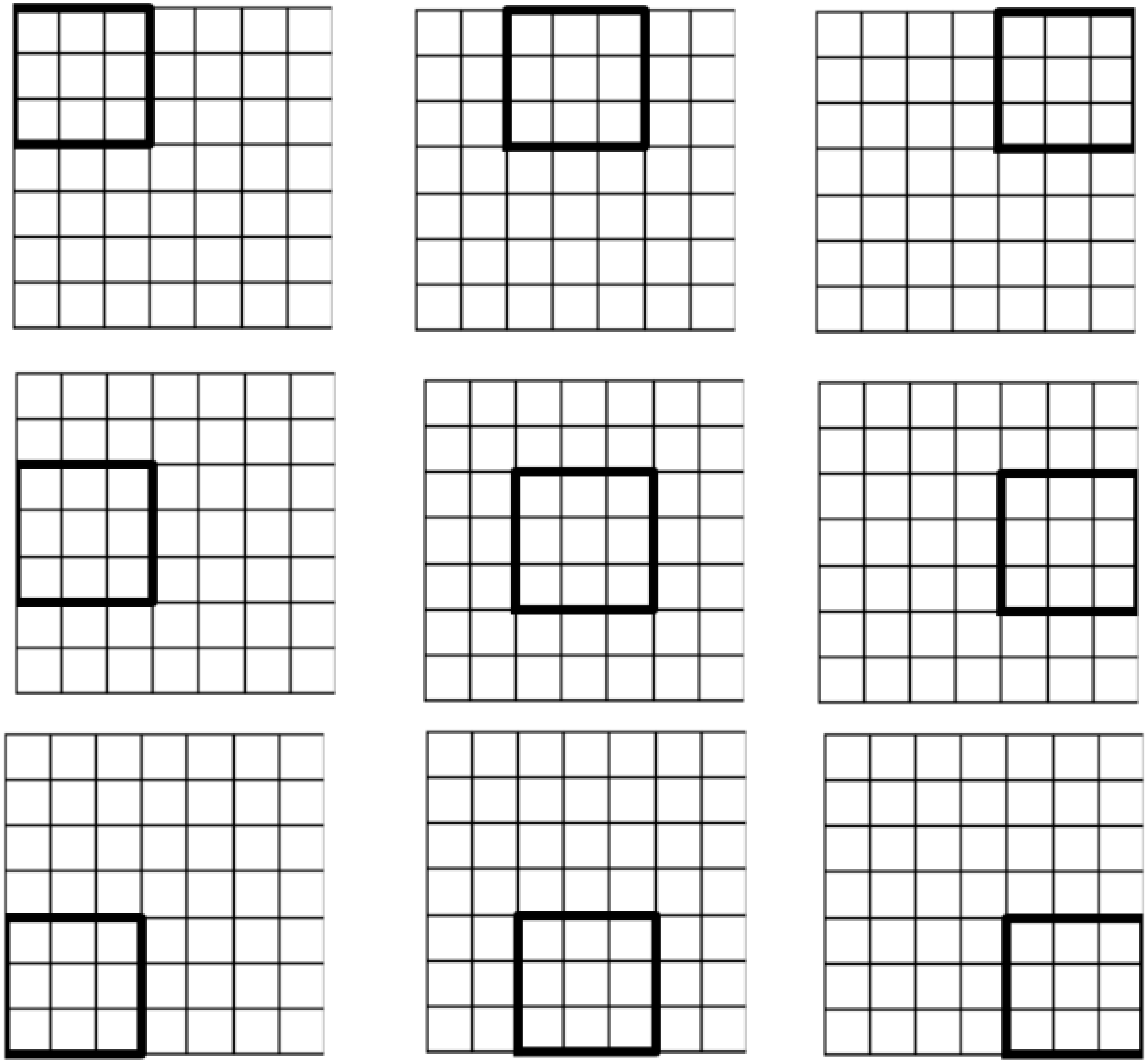
Filter or Kernel
3 x 3

=

2	-1	1	1	0
2	1	0	1	2
1	-1	1	0	1
0	1	2	1	1
2	0	1	0	2

$$(n\times n) * (f\times f) = (\frac{n+2p-f}{s}+1)\times(\frac{n+2p-f}{s}+1) = (\frac{5+(2\times 1)-3}{1}+1)\times(\frac{5+(2\times 1)-3}{1}+1) = 5\times 5$$

We have 3x3 filter and 7x7 image with a stride o 2. How many position which we apply the filter?



Output Size

- To guarantee, the integer output size, the following equation can be used to check:

$$((W - F + 2P)/S) + 1$$

When W is the Image size (square)

F is the kernel size (square)

S is stride

P is the padding

Example

- Alexnet model

The image size is 227x227.

The kernel size is 11x11.

No padding.

Stride is 4.

We obtain number below which is integer

$$((227 - 11 + 2(0))/4) + 1 = 55$$

Question?

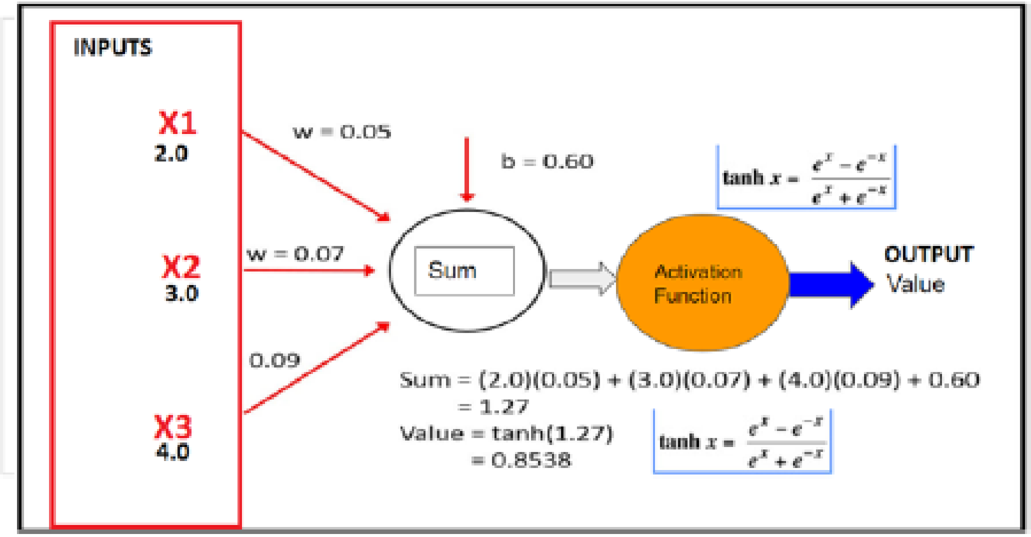
If we have a 21 x 21 filter and a 251 x 151 images with stride of 10. What is the output size?

Answer

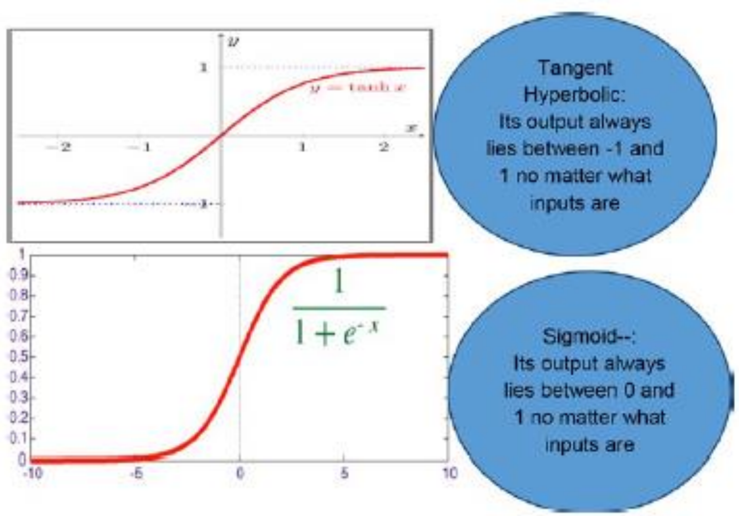
24x14 = [(251-21)/10 + 1][(151-21)/10+1]

Activation Layer

- Activation function is used to introduce non-linearity in the system



- Example of activation function

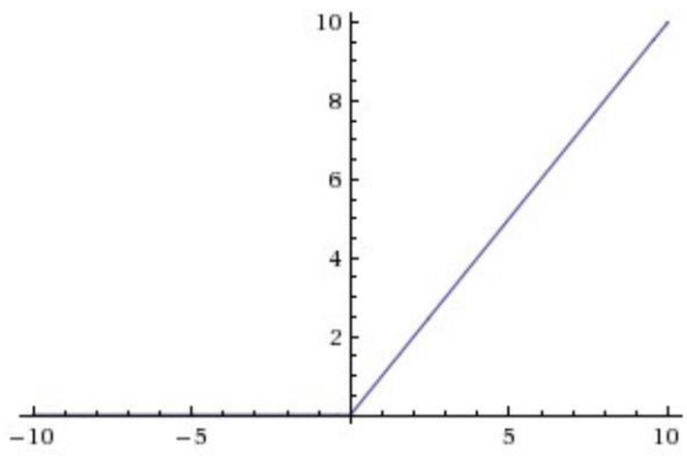


RELU Operation

- Change all negative values to zero

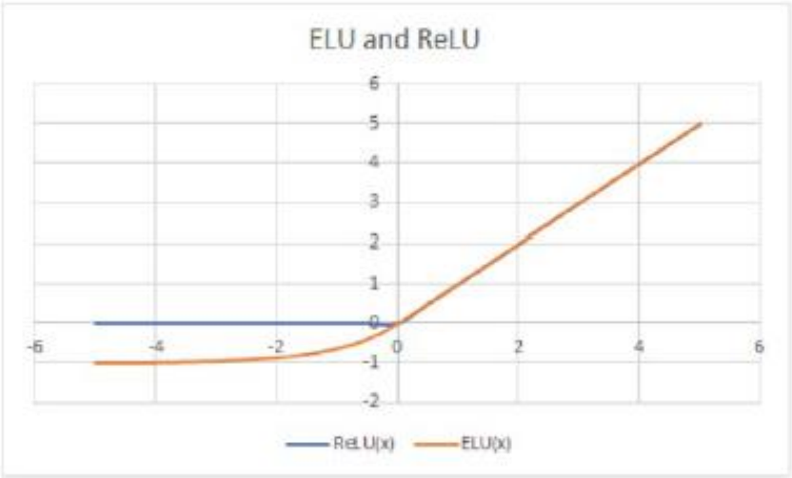
- Leave all positive value alone

$$f(x) = max(0,x)$$



Activation Function

- RELU (Rectifier Linear Unit) and Exponential Linear Unit (ELU)



- The Model:

INPUT => CONV => RELU => FC

Example

- RELU Activation Function

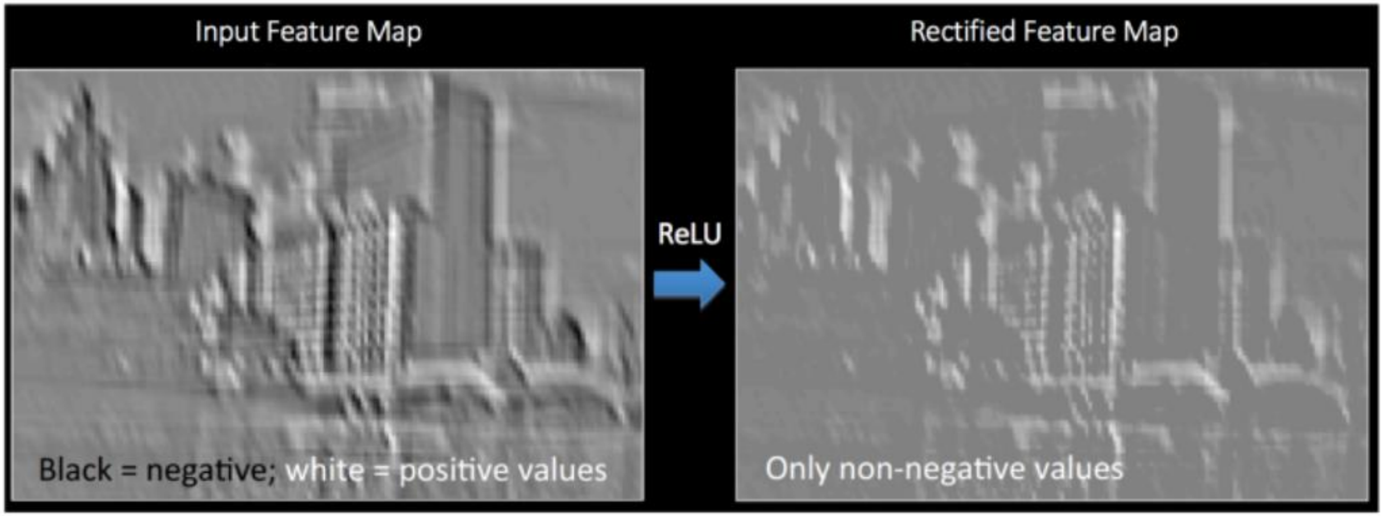
Input

-249	-91	-37
250	-134	101
27	61	-153

ReLU

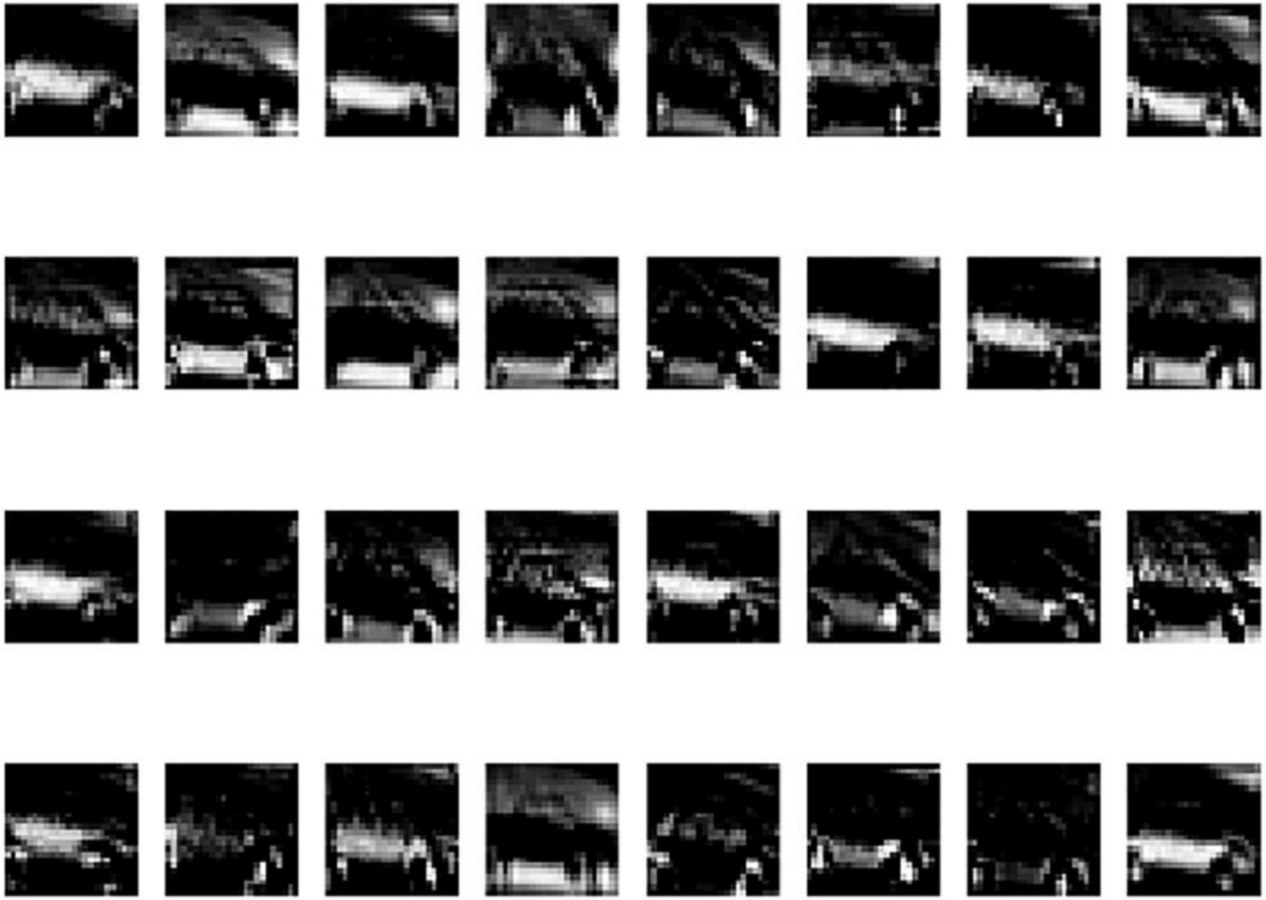
0	0	0
250	0	101
27	61	0

Output Example



Source - http://mlss.tuebingen.mpg.de/2015/slides/fergus/Fergus_1.pdf

Result after Activation Function



Reduce the Input Size

There are two ways to reduce the input size

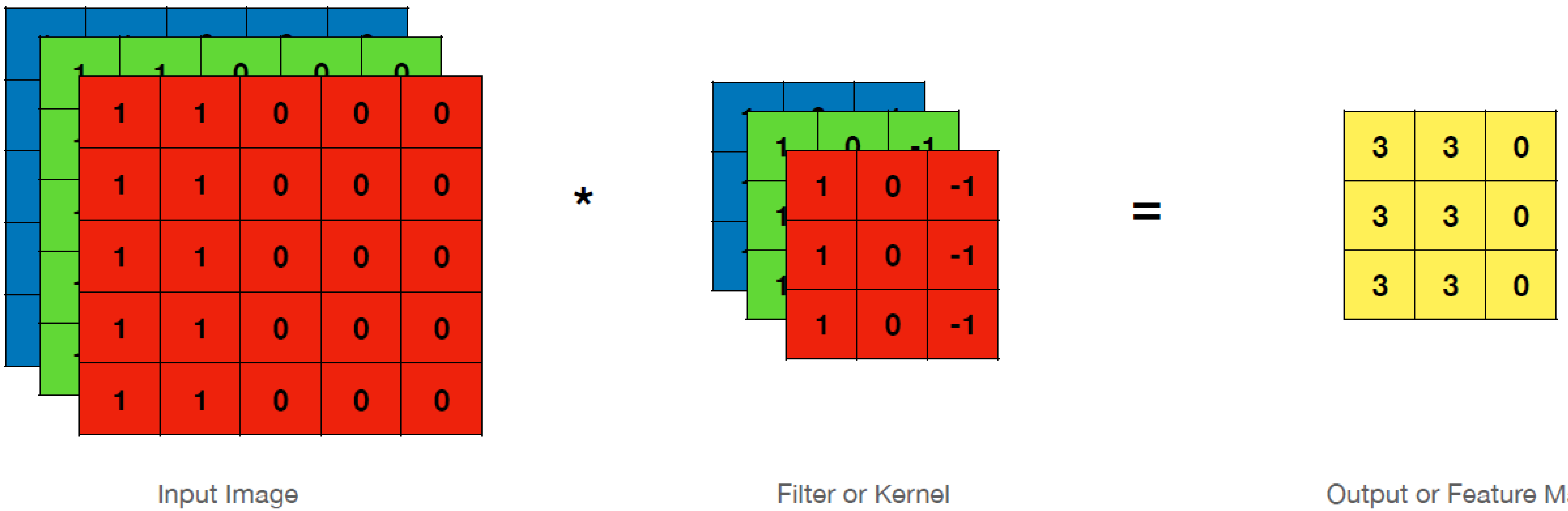
- Convolution with stride > 1

- Pooling layer

Convolution on Grey Scale

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 3 & 3 & 0 \\ 3 & 3 & 0 \\ 3 & 3 & 0 \end{bmatrix}$$

Convolution on Color Image



Output Size

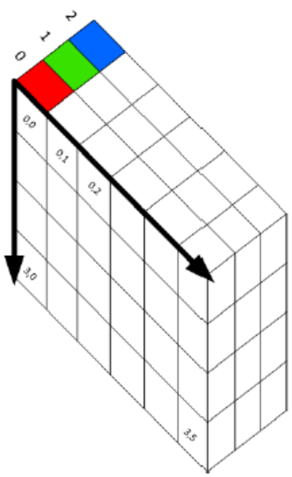
$(n x n x n_c)*(f x f x n_c) \Rightarrow (n-f+1) \times (n-f+1) \times n_f$

n = number of input pixel

n_c = number of channel

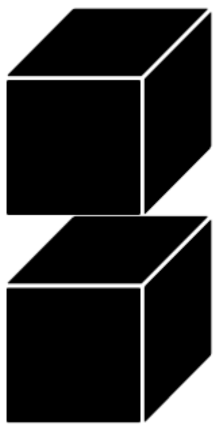
f = filter size

n_f = number of filter



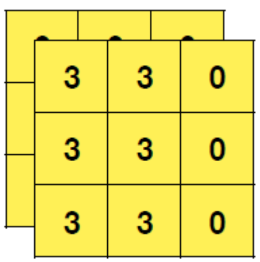
Input Image
5 x 5 x 3

*



2 Filters or Kernels
3 x 3 x 3 x 2

=

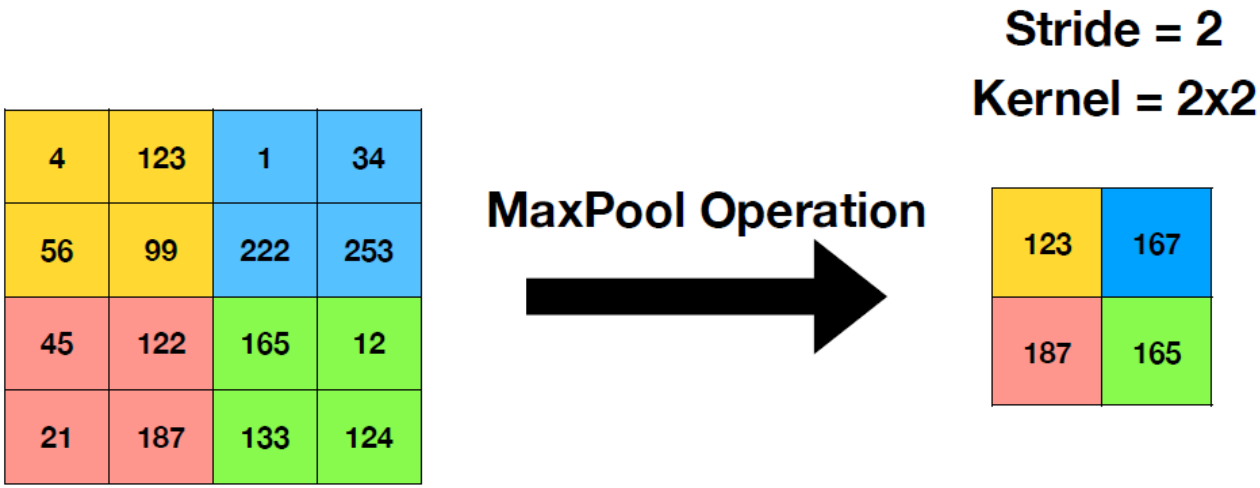


Output or Feature Map
3 x 3 x 2

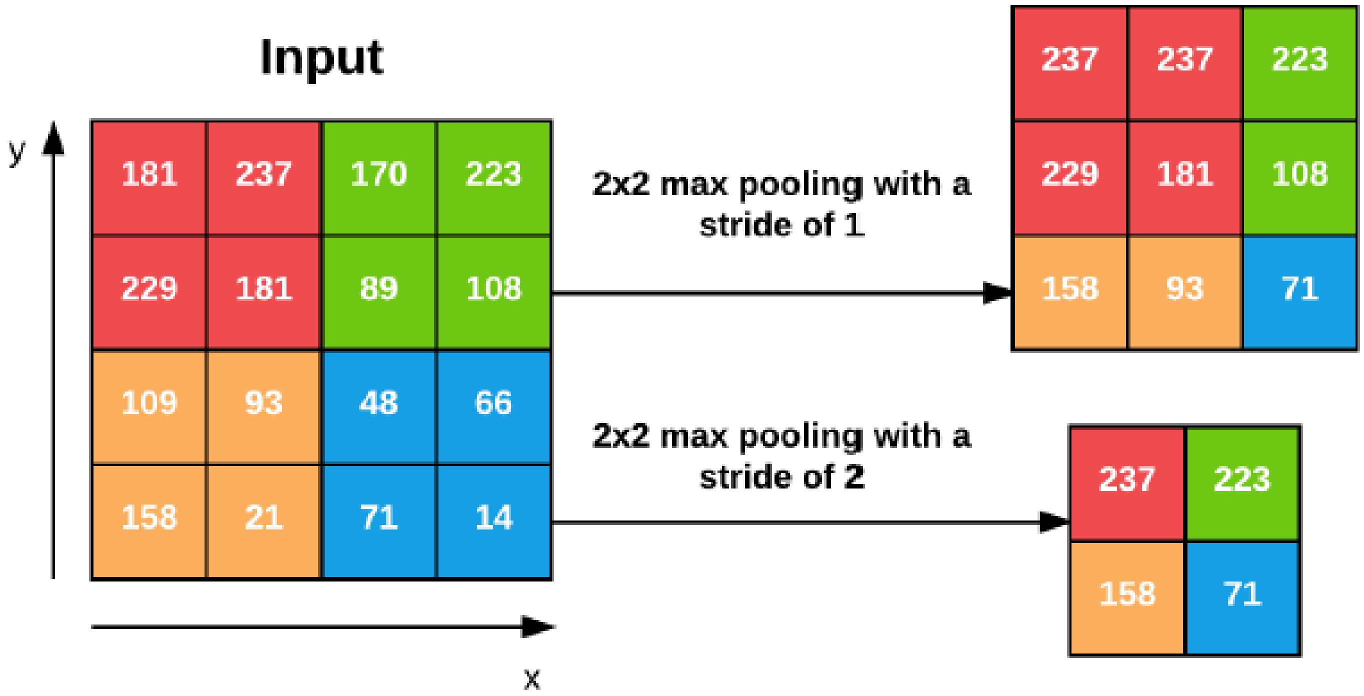
Pooling

- The pooling is a process of subsample (shrink the image) to reduce the computation
- There are many functions that can be applied such as max pooling, min pooling, mean pooling

Example


Stride = 2
Kernel = 2x2
4 123 1 34
56 99 222 253
45 122 165 12
21 187 133 124
MaxPool Operation
123 167
187 165

Example



Input
y
181
237
170
223
229
181
89
108
109
93
48
66
158
21
71
14
x
2x2 max pooling with a stride of 1
237
237
223
229
181
108
158
93
71
2x2 max pooling with a stride of 2
237
223
158
71

Pooling Example

- Example of pooling size using a 2x2 kernel with a stride of 2 and no padding

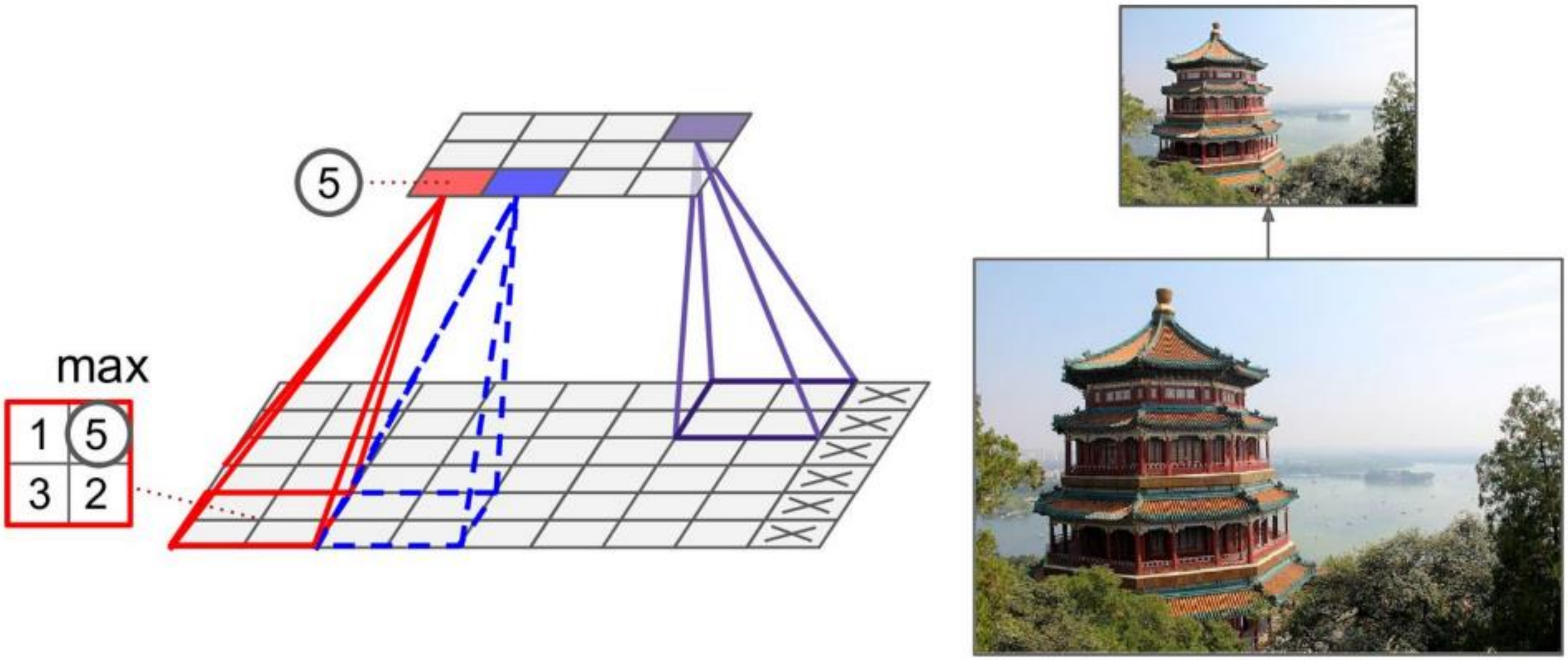


Image from Hands-on Machine Learning with SciKit-Learn and TensorFlow

Pooling Example

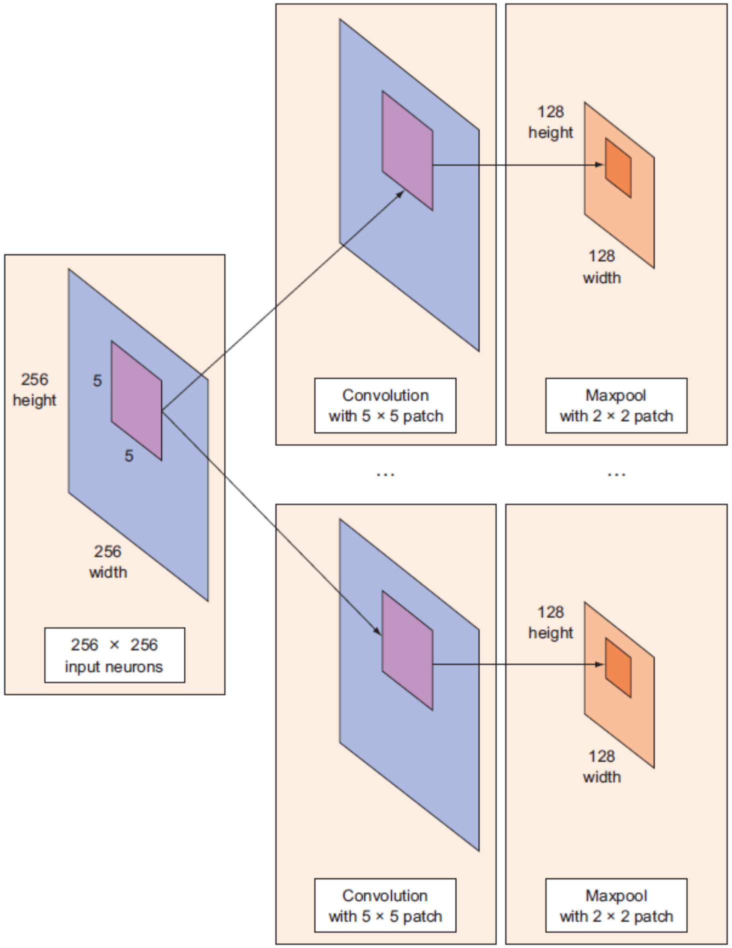
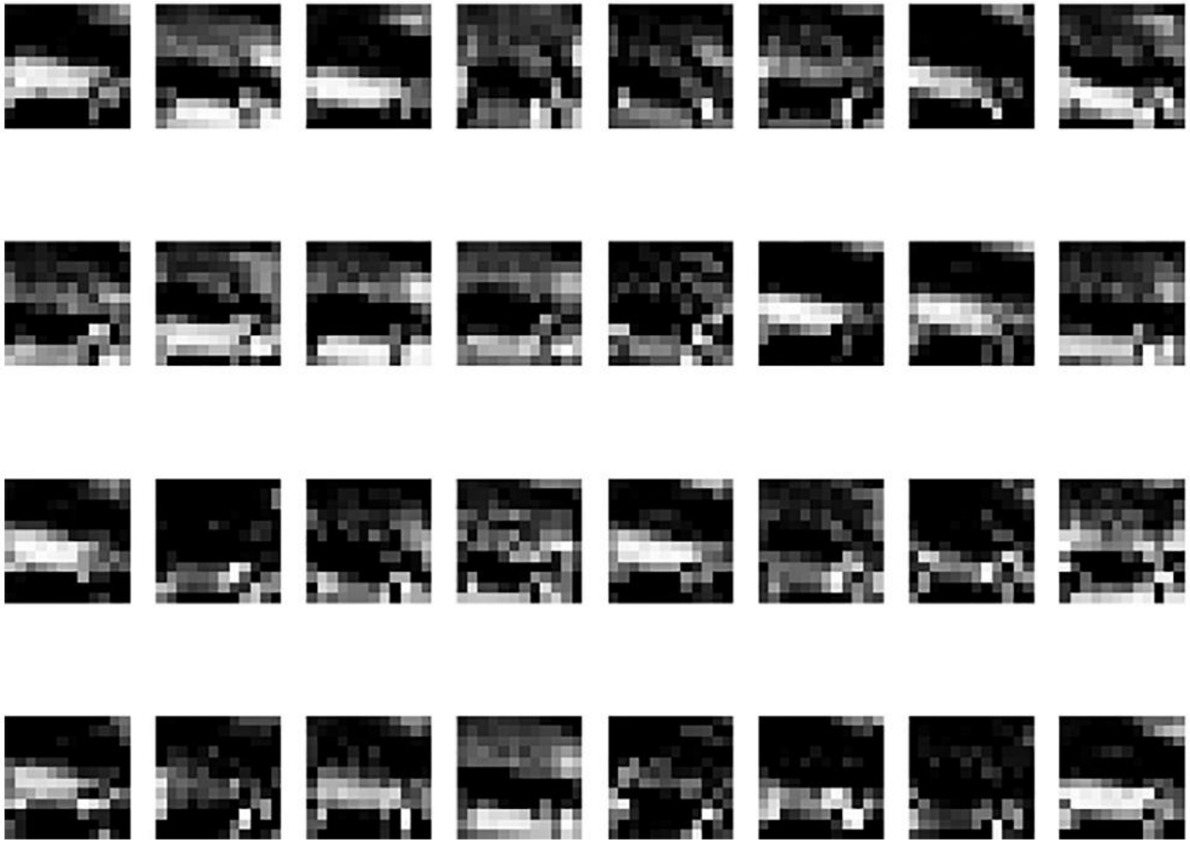


Image from Learning TensorFlow book

Results after Pooling



Model with Pooling

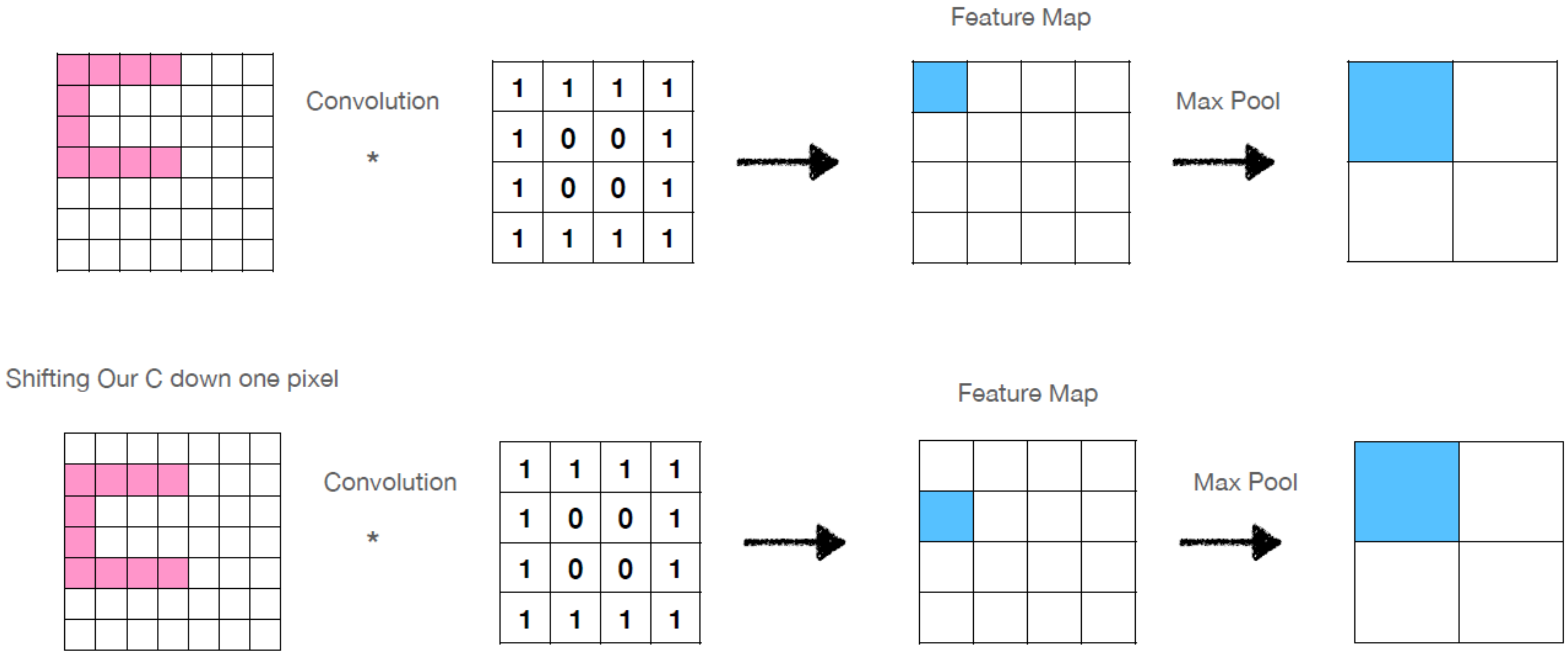
- Example of a model with Pooling:

INPUT => CONV => RELU => POOL => FC

More on Pooling

- Typically, we use 2x2 kernel and stride of 2
- With that, pooling reduces the dimension by a factor of 2 on width and height
- Pooling makes our model more invariant to minor transformations and distortions

How Max Pooling Achieves Translation Invariance



How Pooling Works?

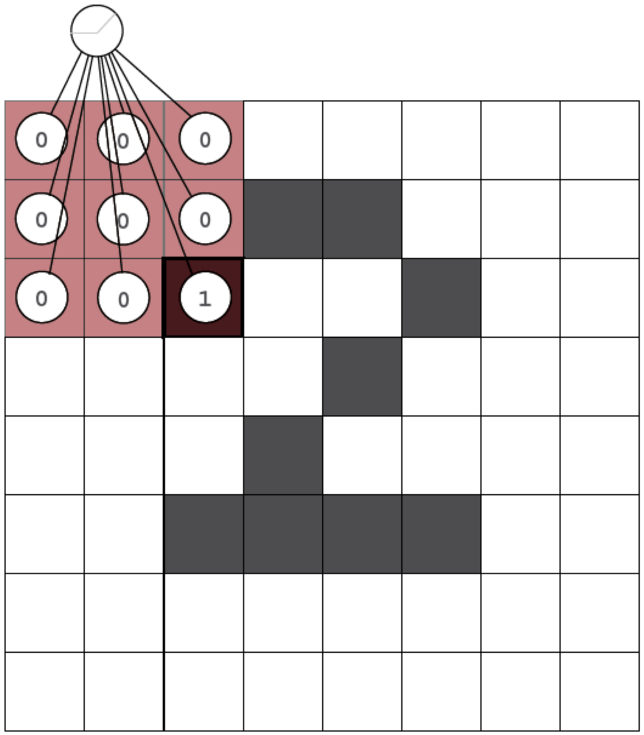
- Neighboring pixels are strongly correlated
- Hence, we can reduce the size of the output by subsampling the filter response without losing information
- A big stride in the pooling layer leads to high information loss
- In practice a stride of 2 was found to be effective

To POOL or CONV?

- In 2014 paper, striving for simplicity: The ALL Convolutional Net, Springenberg et al., propose to discard the POOL layer entirely and use CONV layers with a larger stride to handle downsmapling
- Now, it becomes increasingly common trend to not use POOL

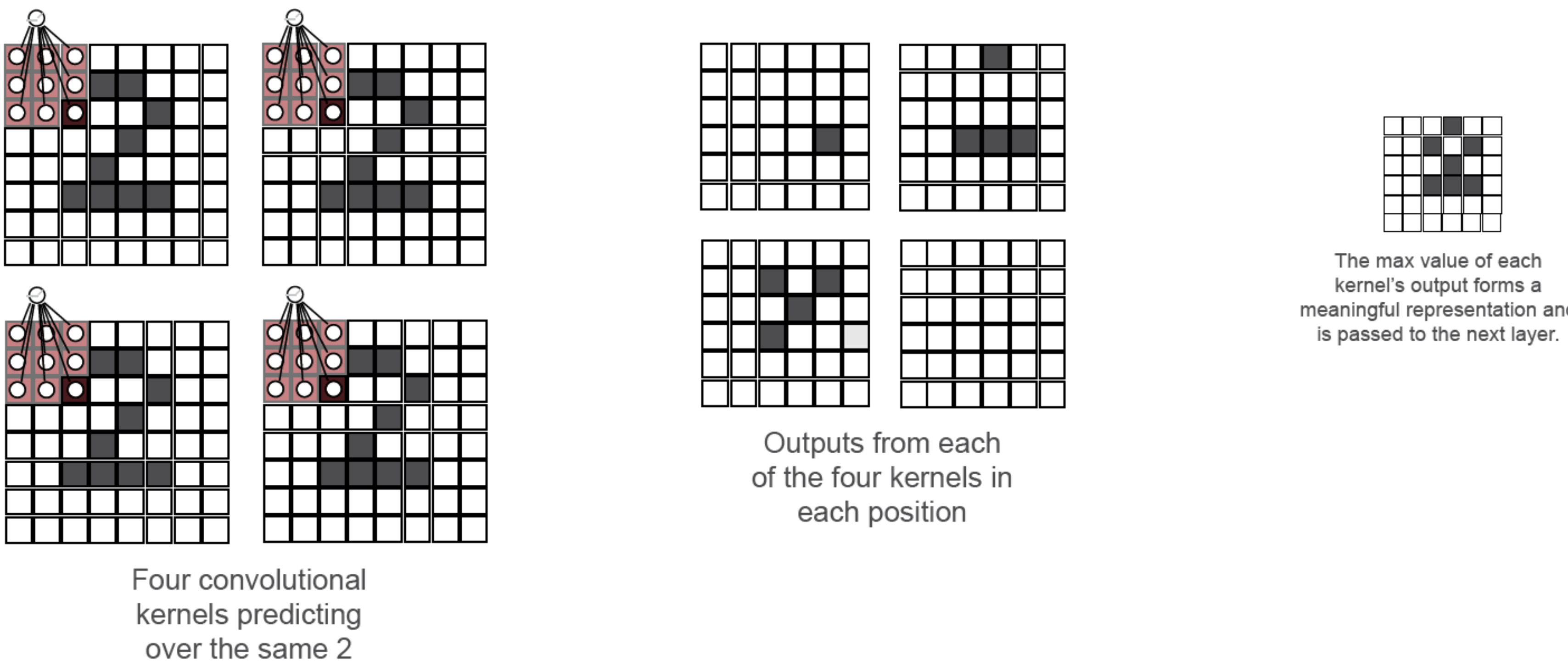
Another way of Pooling

- We can achieve good translation invariant with non-explode weight by using Convoluation Neural Network

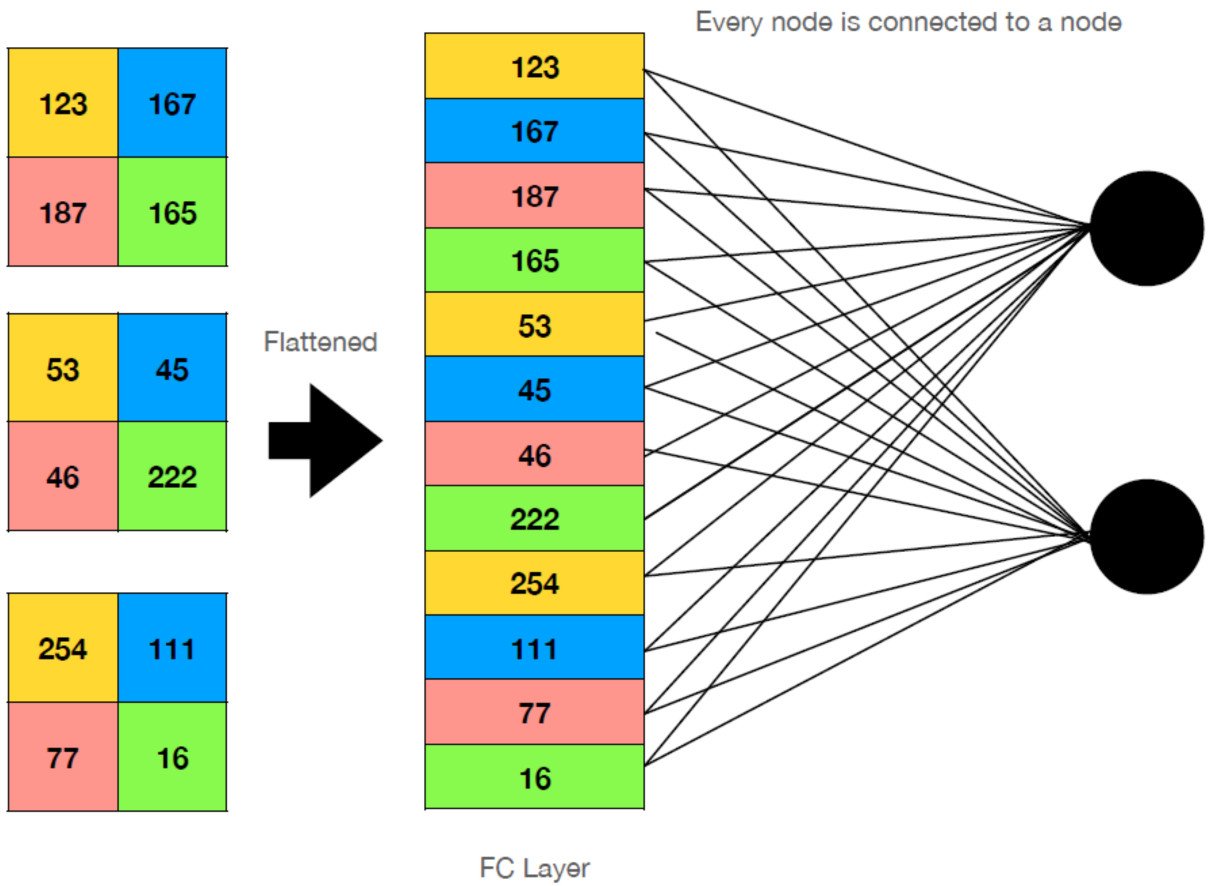


Another way of pooling

- Convolution is done on many filters automatically
- Pooling can also be used between filters



Flattening and Fully Connected Layer



Fully Connected Layer

- It is common to use one or two FC layers prior to applying the softmax classifier

- There is also a trend to not use FC layer as it is computing intensive

- Model:

INPUT => CONV => RELU => POOL => FC

Batch Normalization (BN)

- Recall- the output of CNN is

Batch size x Feature Map Height x Feature Map Width x Channels

- BN calculates the mean and standard deviation of each input variable, to a layer per mini-batch and uses this to perform the standardization

Batch Normalization

- It is introduced by Ioffe and Szegedy in 2015 paper, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift to add BN layer
- The idea is to normalize the data where x_i is mini-batch
- The equation is as follow:

$$\hat{x}_i = \frac{x_i - \mu_\beta}{\sqrt{\sigma_\beta^2 + \varepsilon}}$$

when

$$\mu_\beta = \frac{1}{M}\sum_{i=1}^{m} x_i \qquad \sigma_\beta^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_\beta)^2$$

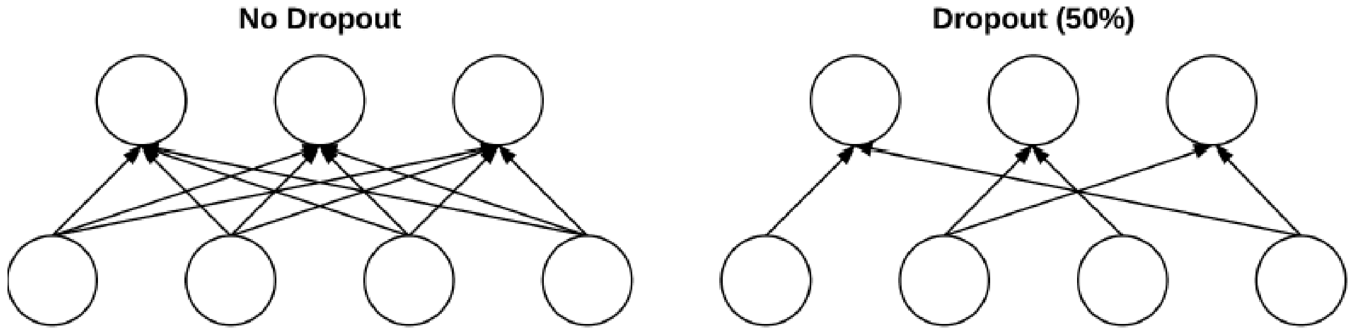
Batch Normalization

- Advantage:
  - Help reduce the number of epochs for training and help for regularization
  - Recommend to put wherever we can
- Drawback:
  - Slow down the system
- Model:

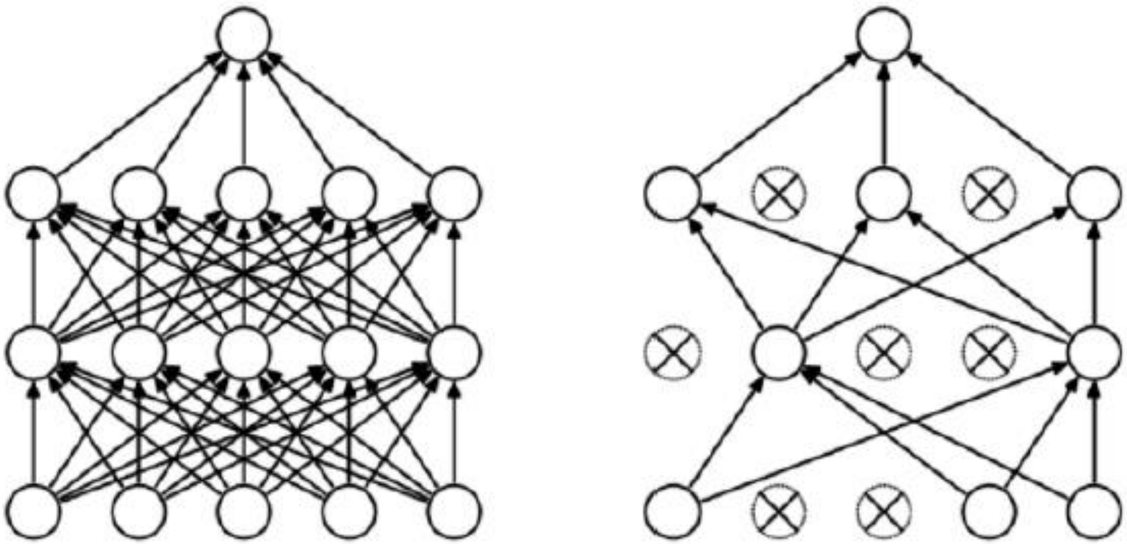
INPUT => CONV => RELU => BN => POOL => FC

Dropout

- The dropout is a regularization technique
- The idea is to keep turn off some neural so that we use many good neural nodes for prediction (not relying only on one)
- It provides multiple redundant nodes



Dropout Example



(a) Standard Neural Net (b) After applying dropout.

Dropout

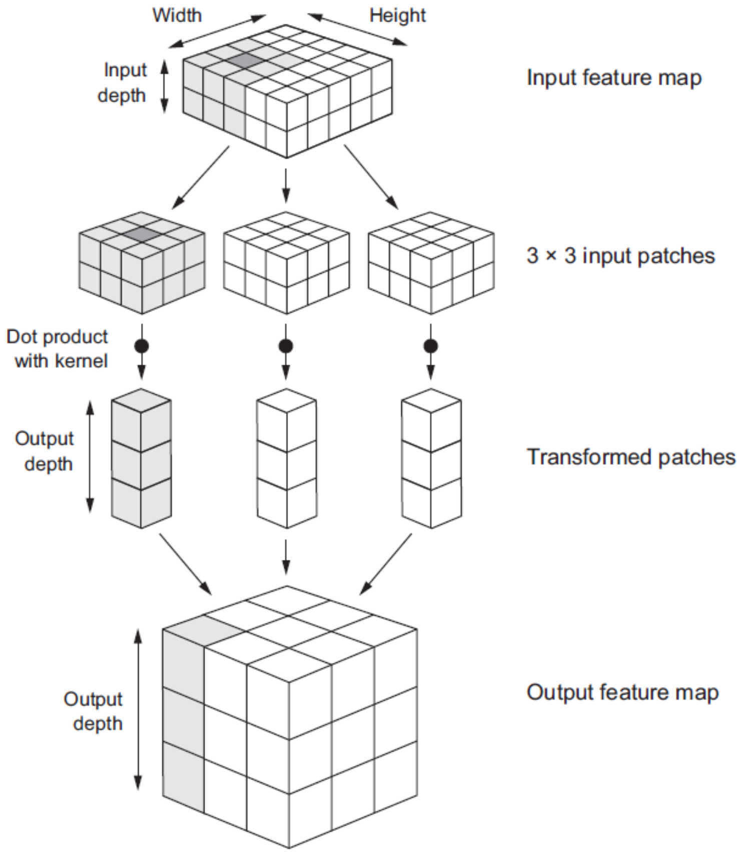
- Note that we randomly add dropout only during the training time,(testing time, we activate back all nodes)

- Model:

INPUT => CONV => RELU => BN => POOL => FC => DO => FC => DO

Convolution on an image

- How convolution work:



Common mistake is to use kernel that are too large

Image from Deep Learning with Python book

Complete CNN Architecture

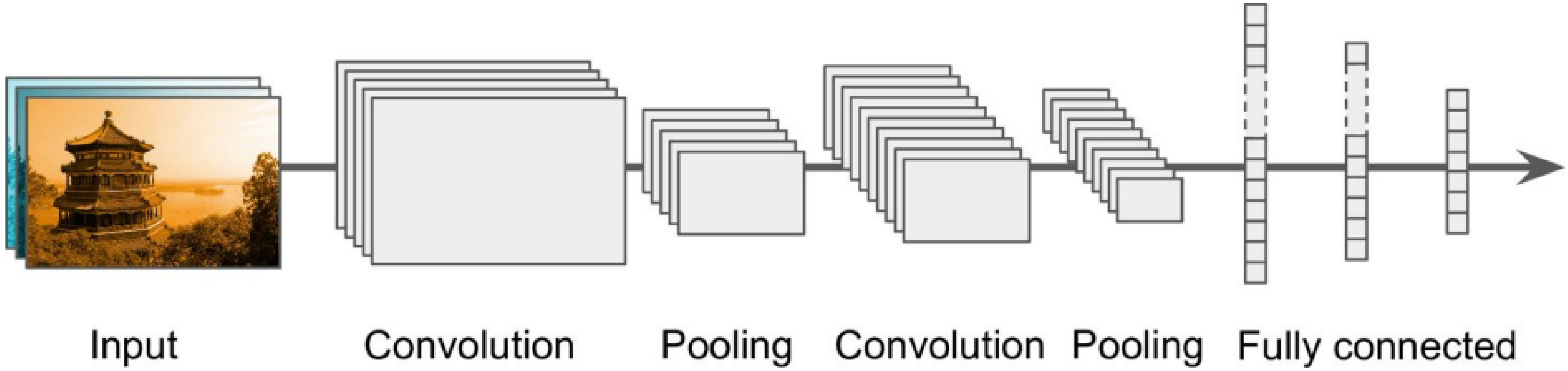
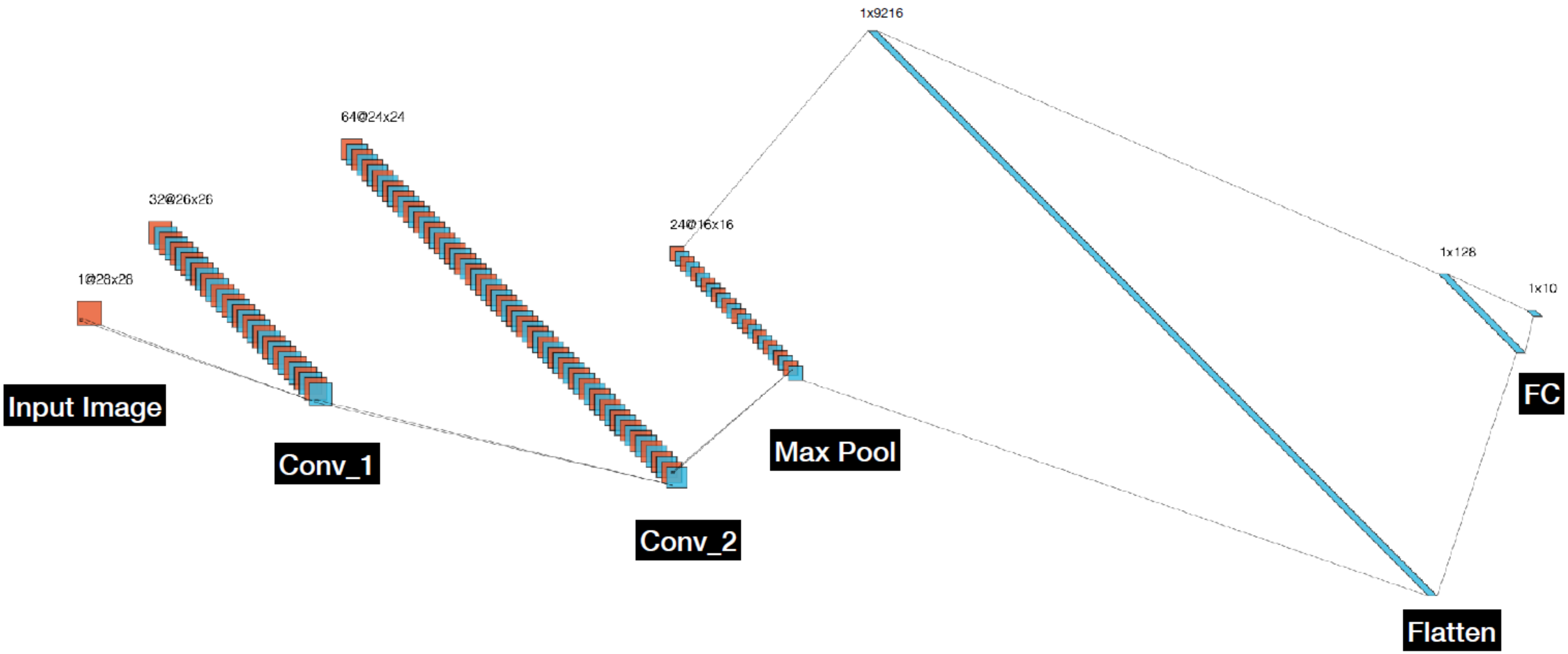


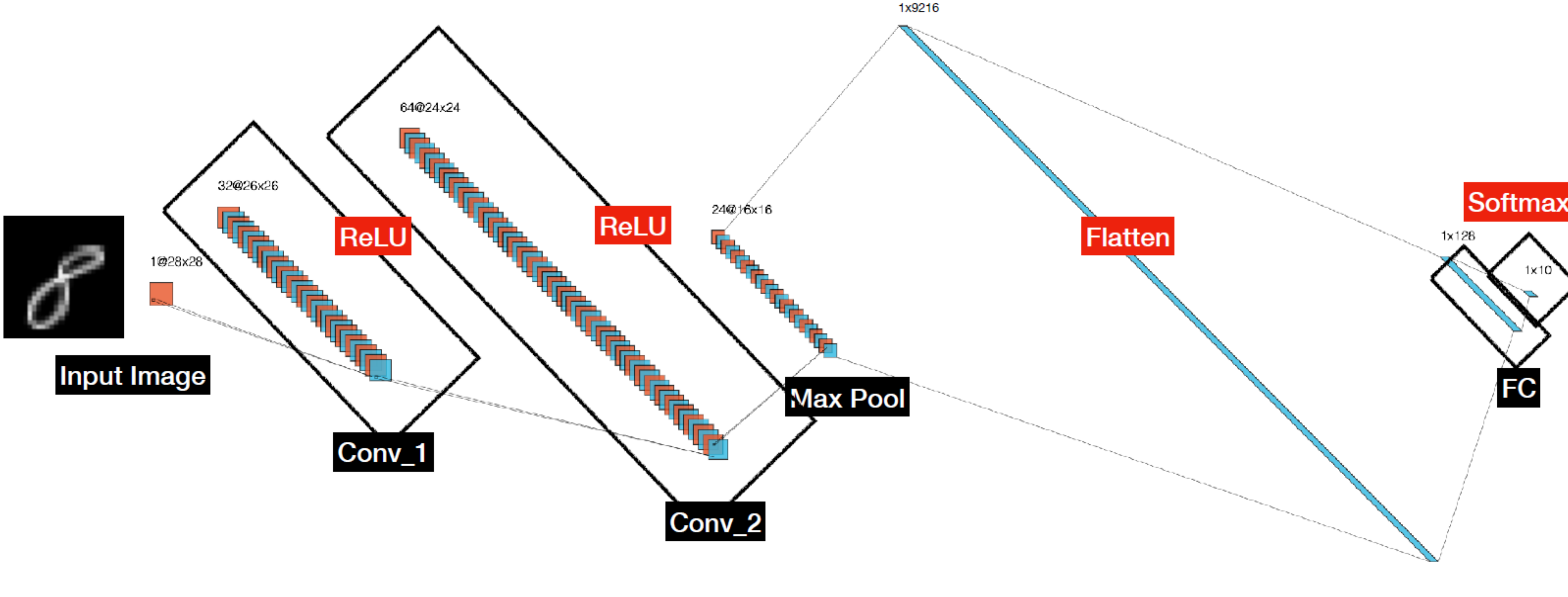
Image from Hands-on Machine Learning with SciKit-Learn and TensorFlow

A Simple CNN

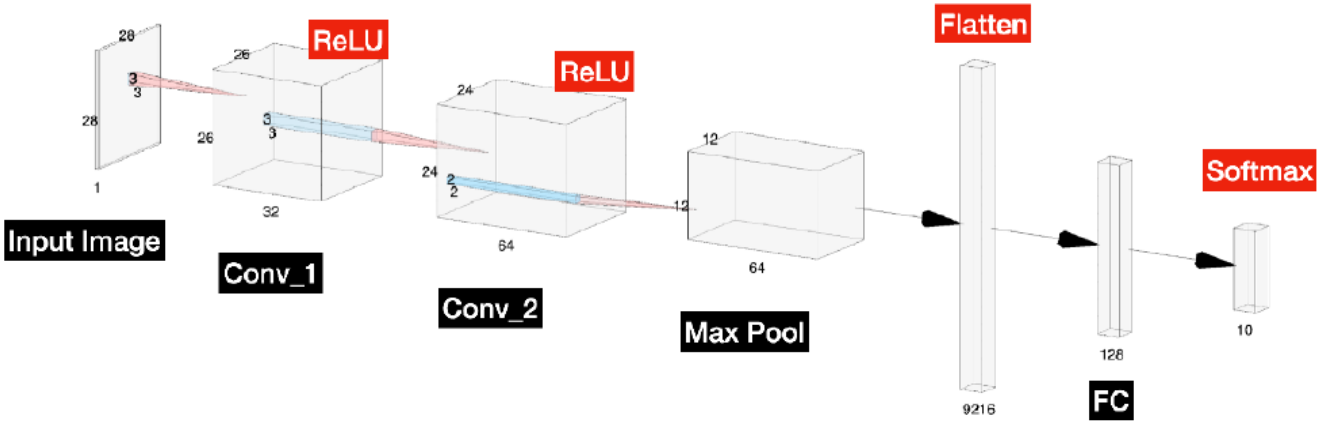


1x9216
64@24x24
32@26x26
24@16x16
1x128
1@28x28
1x10
FC
Input Image
Max Pool
Conv_1
Conv_2
Flatten

4 Layer Deep CNN for MNIST



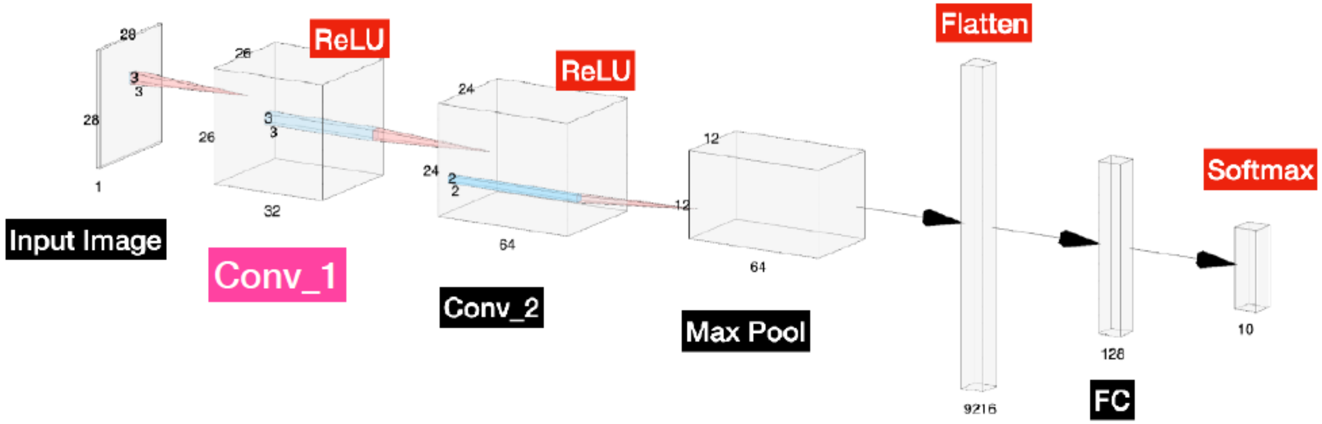
Calculate Output Size



Layer	Depth	Width	Height	Filter (w)	Filter (h)
Input	1	28	28		
Conv_1	32	26	26	3	3
Conv_2	64	24	24	3	3
Max Pool	64	12	12	2	2
Flatten	9216	1	1		
Fully Connected	128	1	1		
Output	10	1	1		

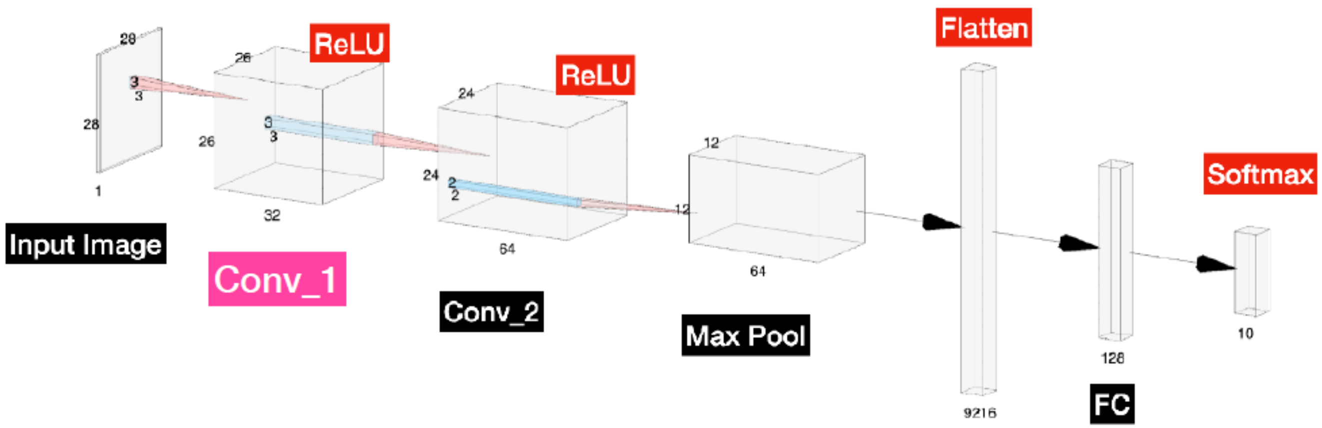
Conv_1 layer

What is the output size after Conv_1 layer?



Conv_1 layer

What is the output size after the layer?

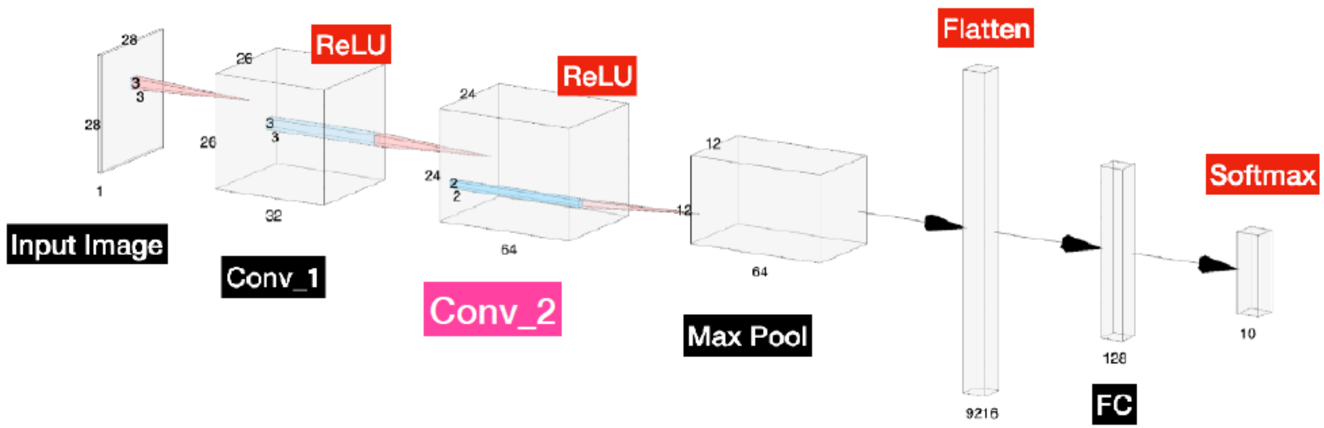


[(n+2p-f)/s + 1] x [(n+2p-f)/s + 1] = 26 x 26

n = 28, f =3 , s =1 , p = 0

Conv_2 layer

What is the output size after the layer?

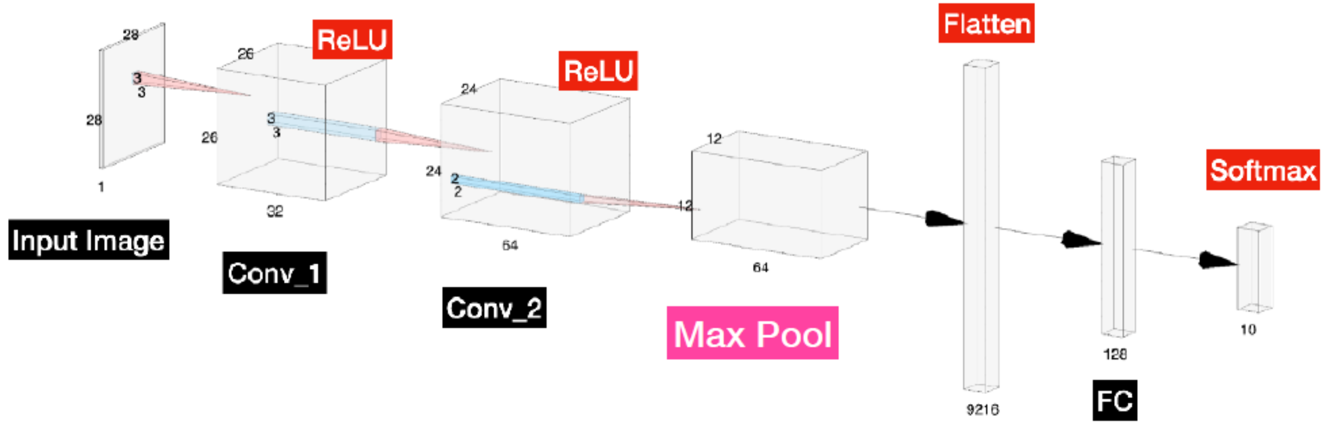


$[(n+2p-f)/s + 1] \times [(n+2p-f)/s + 1] = 24 \times 24$

$n = 26,\ \ f = 3\ ,\ s = 1\ ,\ p = 0$

Max Pool layer

What is the output size after the layer?

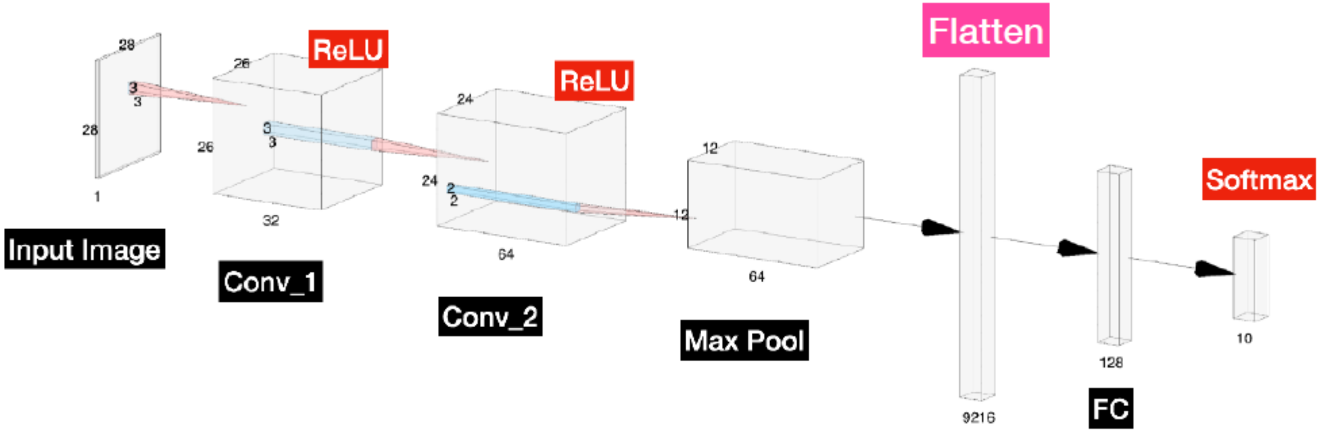


[(n+2p-f)/s + 1] x [(n+2p-f)/s + 1] = 12 x 12

n = 24, f =0 , s =2 , p = 0

Flatten layer

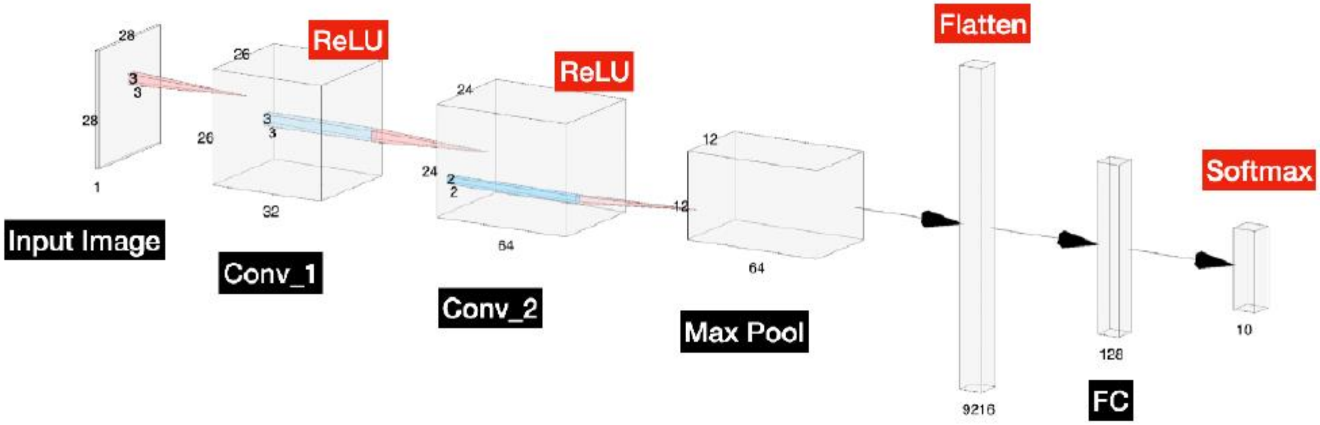
What is the output size after the layer?



12x12x64 = 9,216

FC1 layer

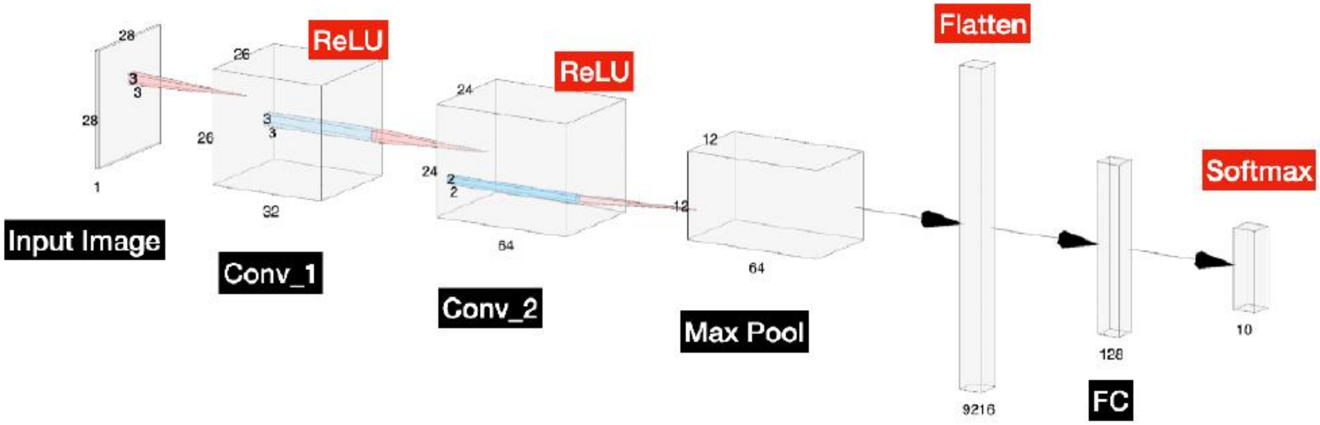
What is the output size after the layer?



128

FC2 layer

What is the output size after the layer?



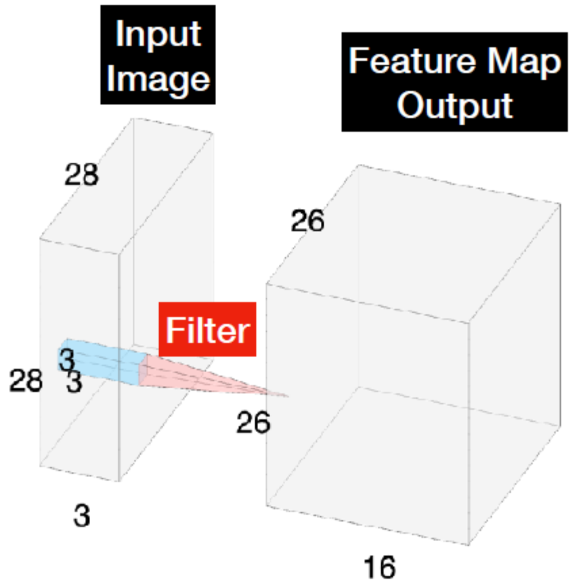
10

Calculate the number of parameters

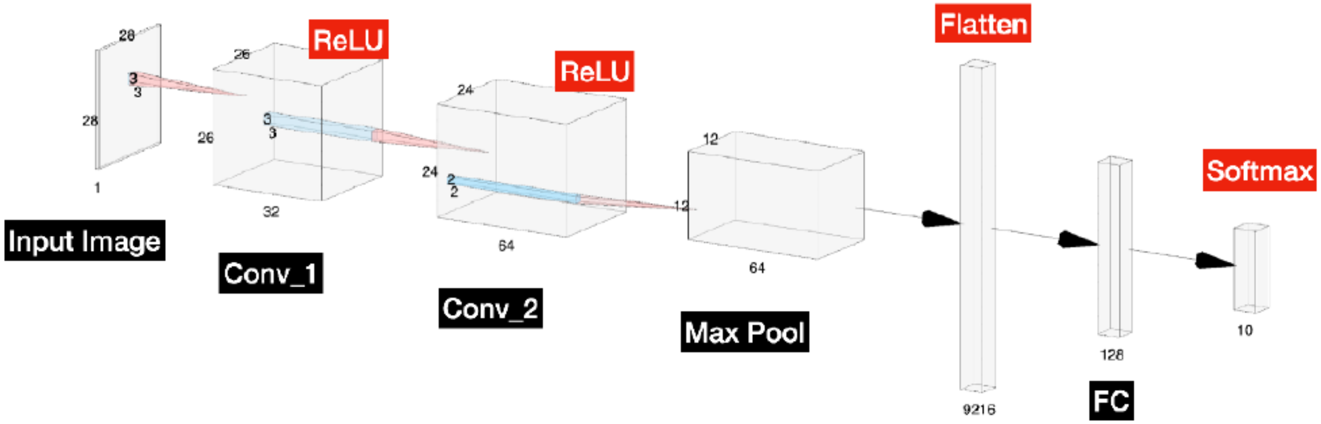
How many parameters are in 16 Conv. Filters with 3x3 kernels

((Height x Width x Dept) + bias) x #Filters =

((3x3x3)+1)x16 = 448

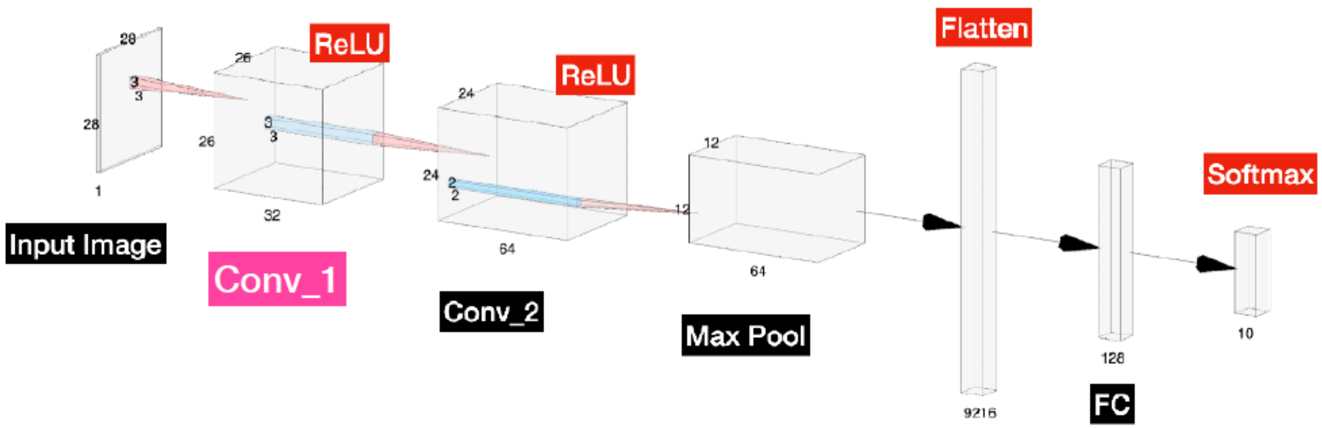


How many Parameter in this system?



Conv_1 layer

How many parameter?

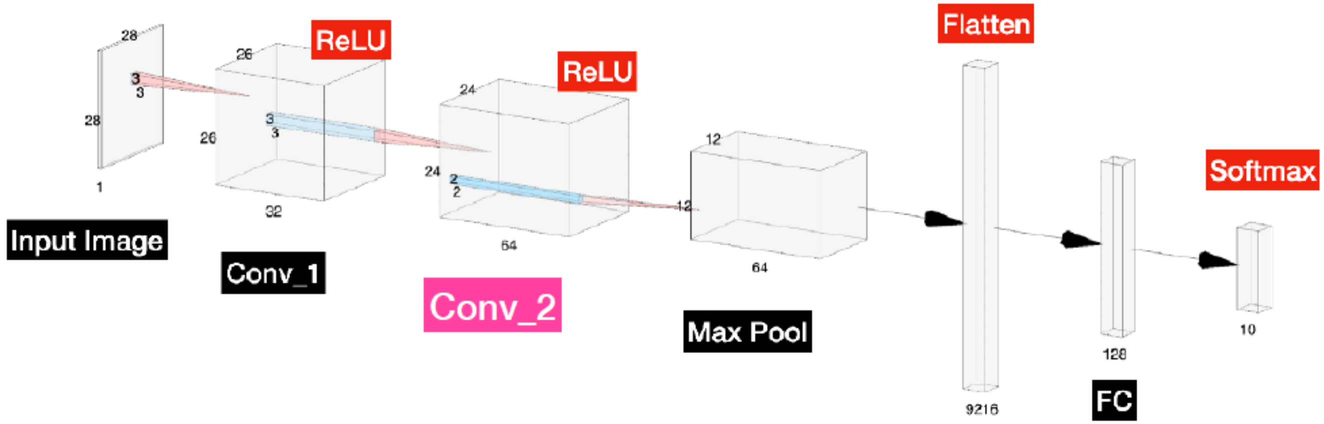


((Height x Width x Depth) + bias) x #Filters =

((3x3x1+1)x16 = 320

Conv_2 layer

How many parameter?

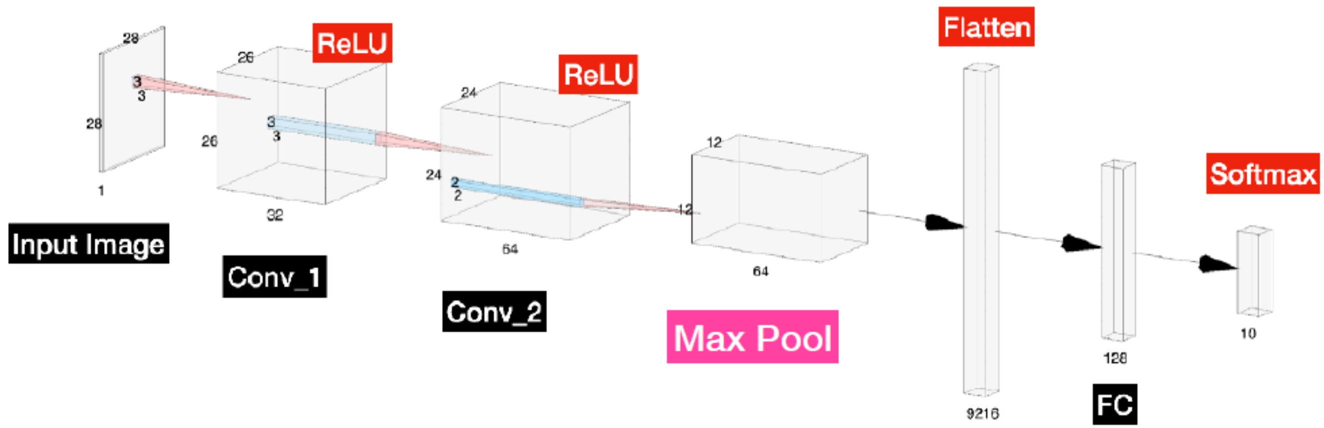


((Height x Width x Depth) + bias) x #Filters =

((3x3x32+1)x64 = 18,494

Max Pool layer

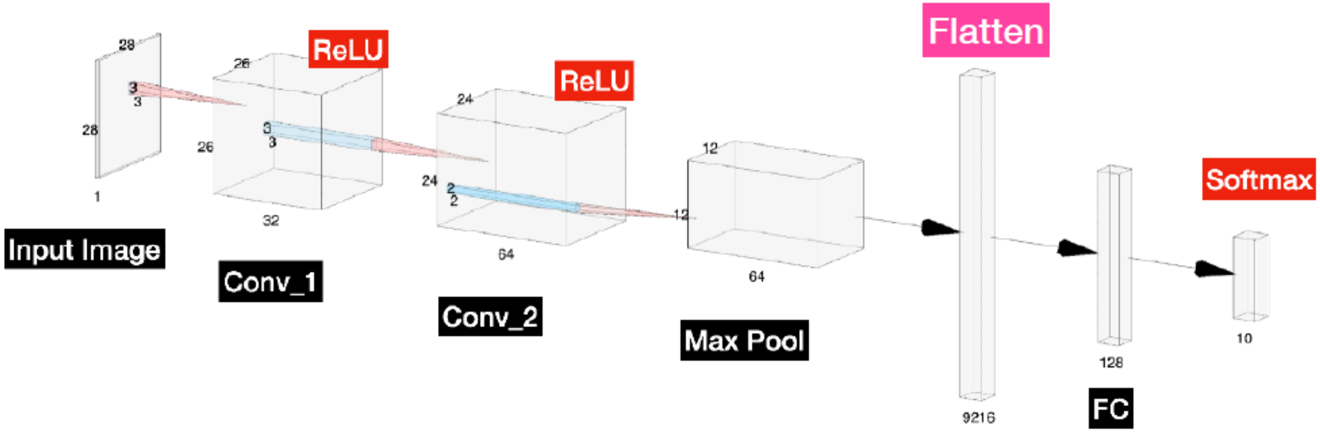
How many parameter?



No trainable parameters

Flatten layer

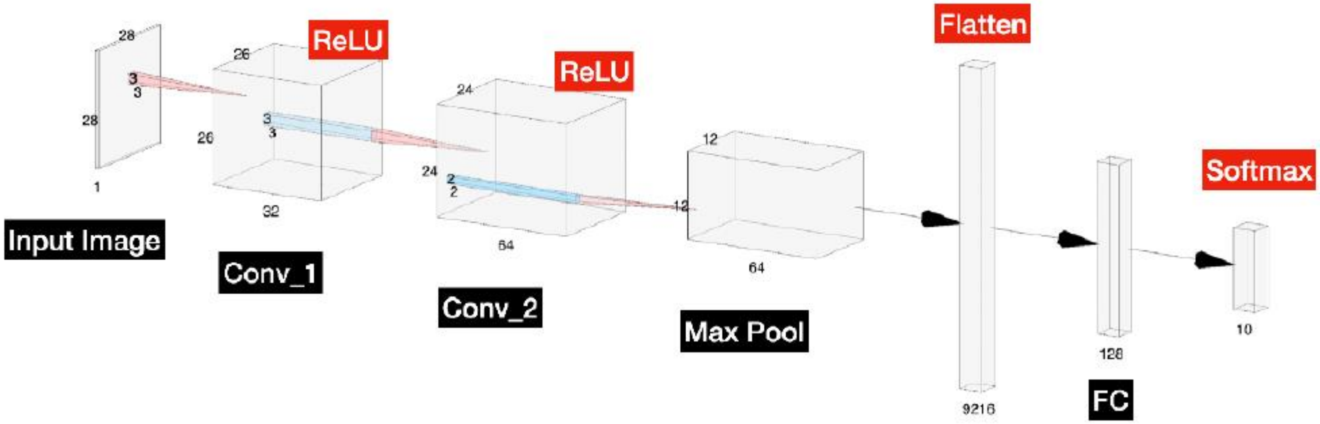
How many parameter?



No trainable parameters (9,216 nodes)

FC1 layer

What is the output size after the layer?

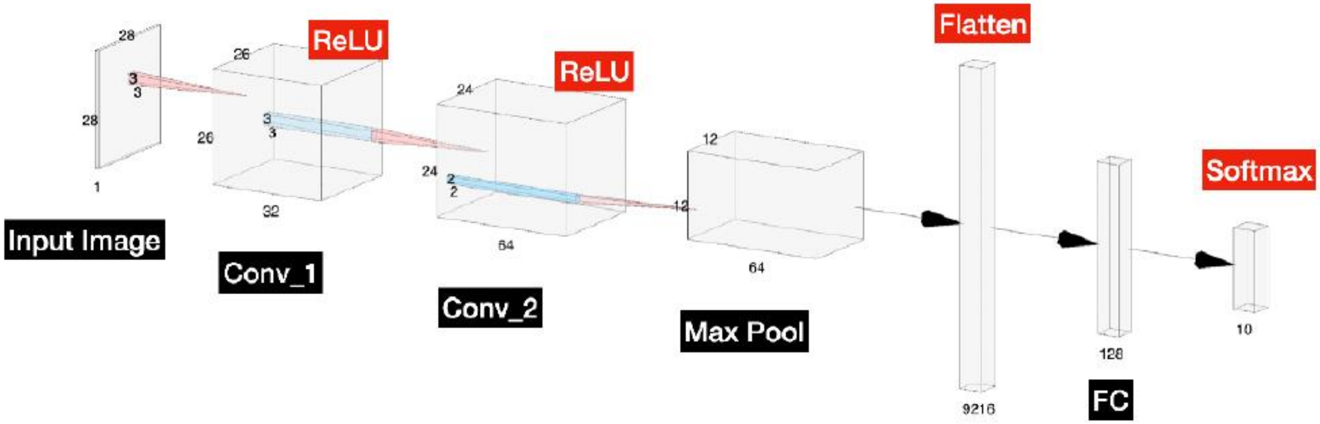


(Input node + bias) x Output node =

(9,216 + 1) x 128 = 1,179,776

FC2 layer

What is the output size after the layer?



(Input node + bias) x Output node =

(128 + 1) x 10 = 1,290

Total Parameters

Layer	Parameters
Conv_1 + ReLU	320
Conv_2 + ReLU	18494
Max Pool	0
Flatten	0
FC_1	1,179,776
FC_2 (Output)	1,290
Total	1,199,882

Four Important Feature for Deep Learning

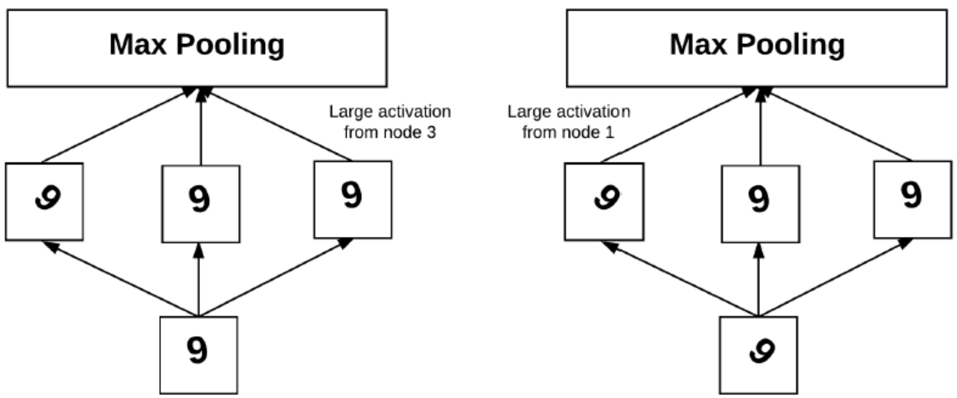
- Dataset
- A Loss Function
- A Neural Network Architecture
- An optimization method

Rules of Thumb

- Common input sizes include 32x32, 64x64, 96x96, 224x224, 227x227 and 229x229
- The input layer should be divisible by two multiple times (to use POOL)
- CONV layers should be small size such as 3x3, 5x5, or 1x1
- Large filter can be used in very first CONV such as 7x7 and 11x11

Is CNN Translation, rotation, and scaling invariant

- CNN is translation invariant with the help of convolutional layer
- It is not rotation and scaling invariant unless you let the network learn a lot of rotation and scaling samples



Why CNNs Work So Well For Images?

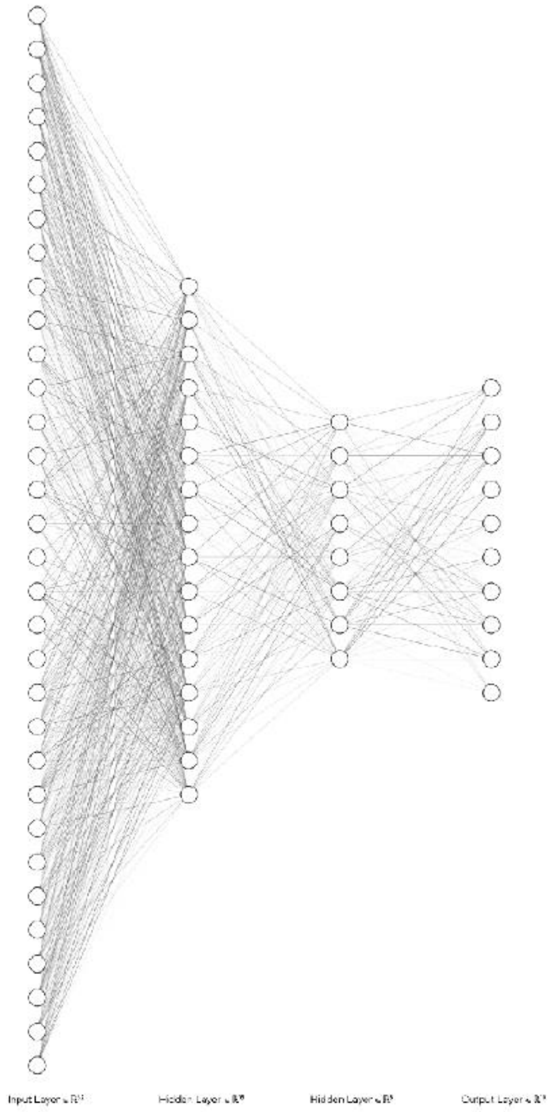
- ANN requires many parameters
- ANN is not translation-invariant

3 Layers of ANN on MNIST

- Input 28x28 = 784

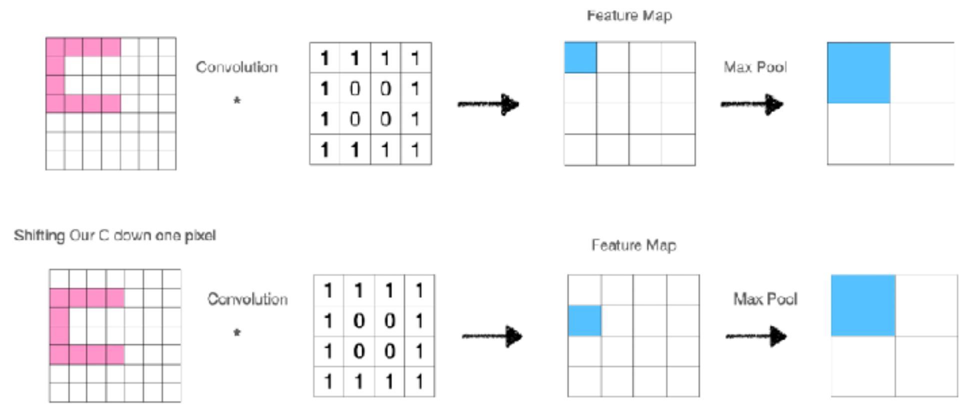
- For second layer, if we have same number of node as CNN Feature map second layer: 32x26x26 = 21,631 nodes

- #trainable parameter = 784x 21,631 = 16,958,704



Advantages of CNN

- Parameter sharing



- Less number of weight comparing with CNN

- Transition Invariance

CNN Assumptions

- Low-level features are local
- Features are translational invariant
- High-level features are made up of low-level features

Rules of Thumb

- We commonly use a stride of S=1, unless we want to do use CONV instead of POOL
- Zero padding should always be applied
- At a novice, POOL is easier to use. Once, you get enough experience, try to avoid it
- POOL should be used with max pooling with 2x2 size and stride =2
- BN and DO should be applied if possible

Questions?