

Table of Contents

- 1 Convolutional Neural Networks (CNN, ConvNet)
 - 1.1 Why CNNs?
 - 1.2 Convolution Operation
 - 1.3 Example with edges filters
 - 1.4 Padding
 - 1.5 Strided Convolutions
 - 1.6 Convolutions on volumes (RGB images)
 - 1.7 Layers in ConvNets
 - 1.8 Schematic Description of a ConvNet
 - 1.9 CNNs in Tensorflow
- 2 Hands-on
 - 2.1 steps
 - 2.2 Dropout
 - 2.3 Augmentation
 - 2.4 Transfer Learning

Convolutional Neural Networks basics

and an example with classification

(LENS ML School 2021; Emmanouela Rantsiou)

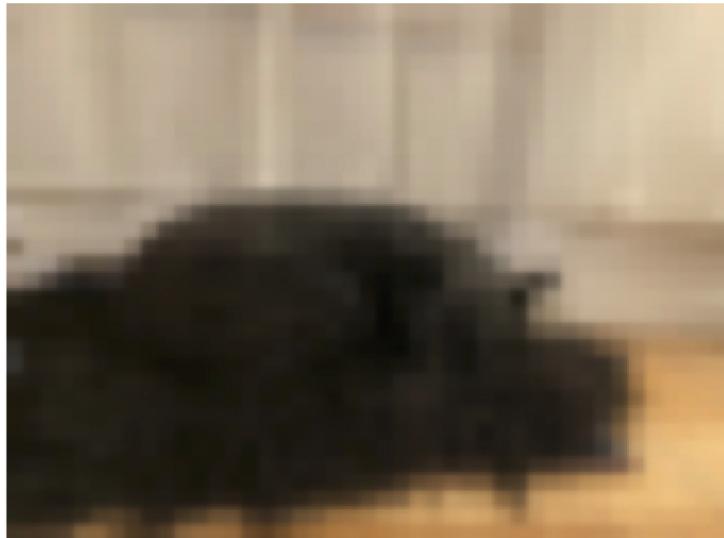
Convolutional Neural Networks (CNN, ConvNet)

- a class of artificial Neural Networks
- widely used in computer vision*
- learn hierarchy of features

*Computer vision is an essential, complex, wide-spread, and ever developing part of AI.
Computer vision tasks include:

- Image Classification
- Classification with localization
- Object detection
- Image captioning
- action classification
- Semantic segmentation
- Instance segmentation
- Neural Style Transfer

Why CNNs?



- Need not to be restricted to small input images.
- Image of 1300 X 1300px --> Input on a layer of a FC (Dense) network with e.g. 1000 hidden units → parameters (weights) to calculate : $(1000, 1300 \times 1300) = 1.69E9$ parameters (x3 for a color image) → Computationally expensive and data hungry.

Convolution Operation

IMAGE

1	4	12	11	0	2
9	15	3	0	4	17
1	4	8	6	6	10
16	10	0	11	13	6
0	2	9	3	0	8
2	14	12	5	14	1

Convolution Kernel (Filter)

3	0	8
1	5	10
0	2	9

1	4	12	11	0	2
9	15	3	0	4	17
1	4	8	6	6	10
16	10	0	11	13	6
0	2	9	3	0	8
2	14	12	5	14	1



3	0	8
1	5	10
0	2	9

=

293			

$$1 \times 3 + 4 \times 0 + 12 \times 8 + 9 \times 1 + 15 \times 5 + 3 \times 10 + 1 \times 0 + 4 \times 2 + 8 \times 9 = 293$$

1	4	12	11	0	2
9	15	3	0	4	17
1	4	8	6	6	10
16	10	0	11	13	6
0	2	9	3	0	8
2	14	12	5	14	1



3	0	8
1	5	10
0	2	9

=

293	216		

$$4 \times 3 + 12 \times 0 + 11 \times 8 + 15 \times 1 + 3 \times 5 + 0 \times 10 + 4 \times 0 + 8 \times 2 + 6 \times 9 = 216$$

1	4	12	11	0	2
9	15	3	0	4	17
1	4	8	6	6	10
16	10	0	11	13	6
0	2	9	3	0	8
2	14	12	5	14	1



3	0	8
1	5	10
0	2	9

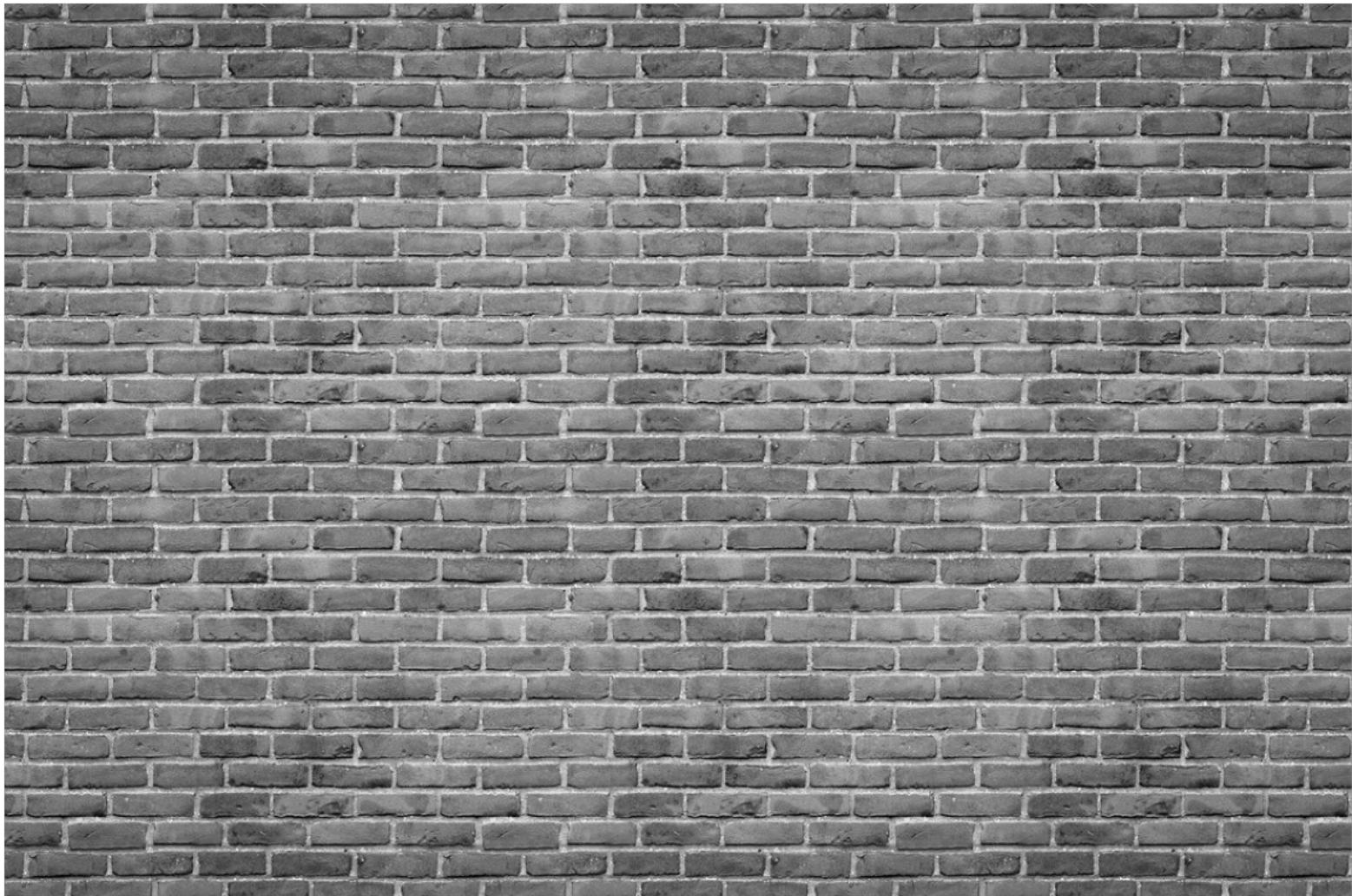
=

293			

Example with edges filters

(edge detectors)

```
In [17]: image = PIL.Image.open('brick2.jpeg').convert('L')
brick = asarray(image)
plt.figure(figsize=(20,20))
plt.imshow(brick,cmap='gray')
plt.axis('off')
plt.show()
```




```
In [18]: image = plt.imread('filters_conv.jpeg')
plt.figure(figsize = (20,20))
plt.imshow(image)
plt.axis('off')
plt.show()
```

'vertical' filter



1	0	-1
1	0	-1
1	0	-1

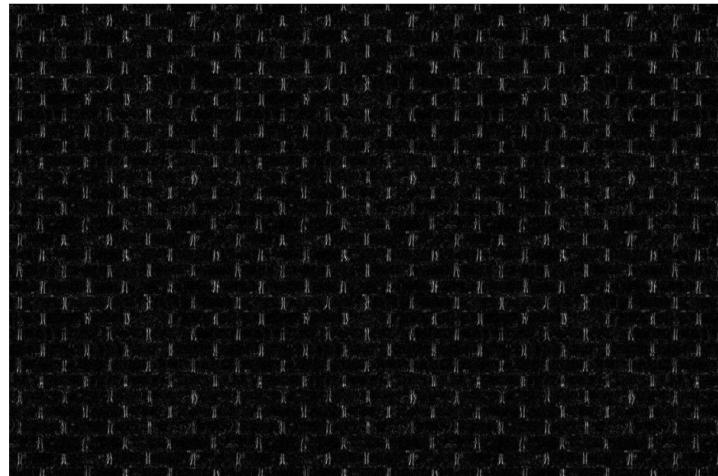
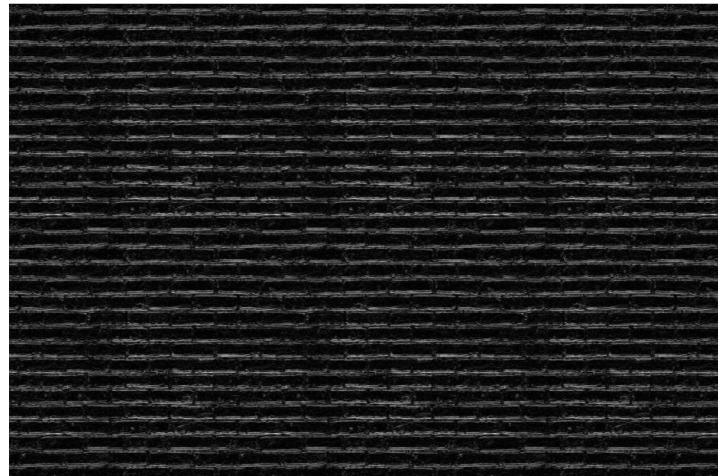
'horizontal' filter



1	1	1
0	0	0
-1	-1	-1

```
In [14]: out_conv_h=signal.convolve2d(brick,filter_h,mode='valid')
out_conv_v=signal.convolve2d(brick,filter_v,mode='valid')
f, axarr = plt.subplots(1,2,figsize = (50,50))
axarr[0].imshow(np.absolute(out_conv_h),cmap='gray')
axarr[0].set_axis_off()
axarr[1].imshow(np.absolute(out_conv_v),cmap='gray')
axarr[1].set_axis_off()

plt.show()
```

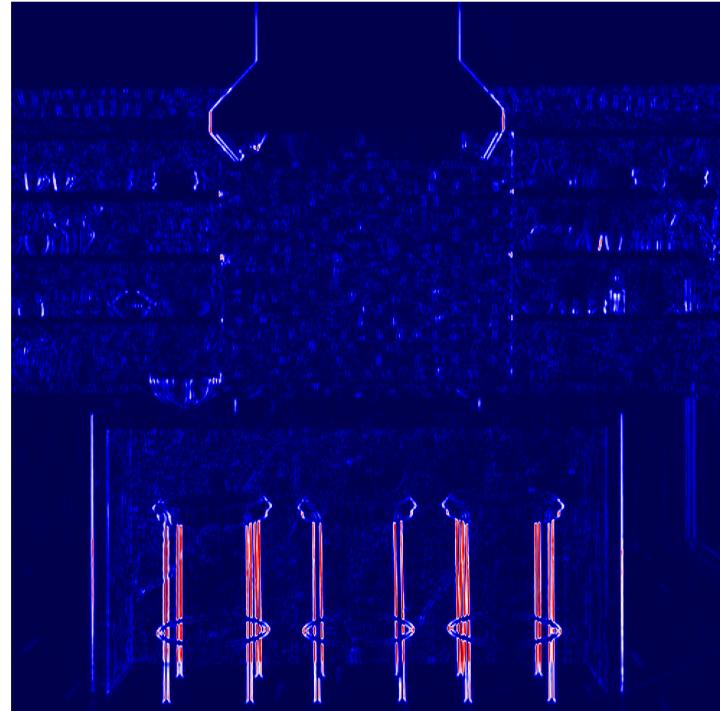
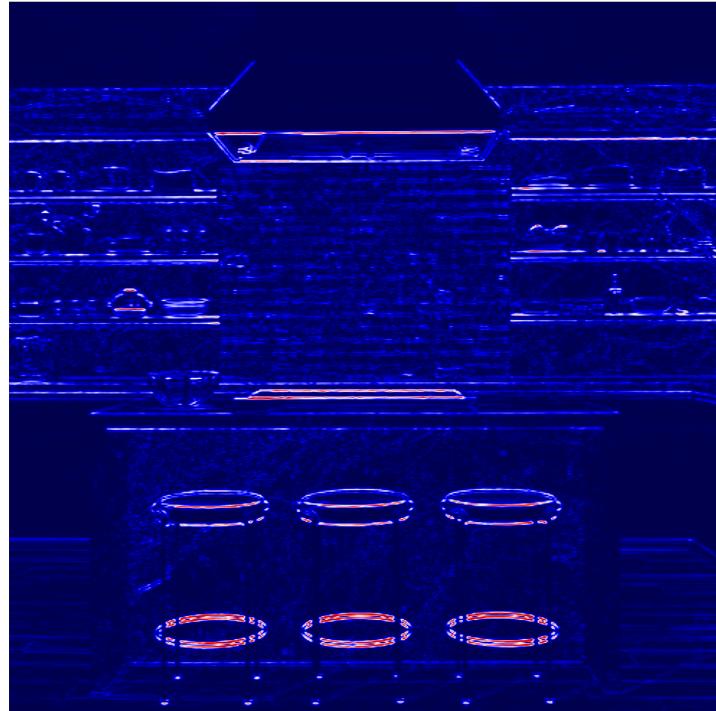


```
In [10]: image = PIL.Image.open('brick.jpeg')
brick = asarray(image)
plt.figure(figsize=(10,10))
plt.imshow(brick)
plt.axis('off')
plt.show()
```




```
In [11]: out_corr_v=signal.correlate(brick,filter_v_3d,mode='valid')
out_corr_h=signal.correlate(brick,filter_h_3d,mode='valid')

f, axarr = plt.subplots(1,2,figsize = (50,50))
axarr[0].imshow(np.absolute(out_corr_h),cmap='seismic')
axarr[0].set_axis_off()
axarr[1].imshow(np.absolute(out_corr_v),cmap='seismic')
axarr[1].set_axis_off()
plt.show()
```



6 X 6

1	4	12	11	0	2
9	15	3	0	4	17
1	4	8	6	6	10
16	10	0	11	13	6
0	2	9	3	0	8
2	14	12	5	14	1

3 X 3



3	0	8
1	5	10
0	2	9



4 X 4

293	216	158	342
181	258	278	352
234	230	273	308
293	264	280	206

$$\begin{matrix} n \times n & \text{Image} \\ f \times f & \text{filter} \end{matrix} \quad \} \quad (n - f + 1) \times (n - f + 1)$$

Padding

Padding

0	0	0	0	0	0	0	0	0
0	1	4	12	11	0	2	0	0
0	9	15	3	0	4	17	0	0
0	1	4	8	6	6	10	0	0
0	16	10	0	11	13	6	0	0
0	0	2	9	3	0	8	0	0
0	2	14	12	5	14	1	0	0
0	0	0	0	0	0	0	0	0



3	0	8
1	5	10
0	2	9

=

90					

Padding

0	0	0	0	0	0	0	0	0
0	1	4	12	11	0	2	0	0
0	9	15	3	0	4	17	0	0
0	1	4	8	6	6	10	0	0
0	16	10	0	11	13	6	0	0
0	0	2	9	3	0	8	0	0
0	2	14	12	5	14	1	0	0
0	0	0	0	0	0	0	0	0



3	0	8
1	5	10
0	2	9

=

90	198					

Padding

0	0	0	0	0	0	0	0	0
0	1	4	12	11	0	2	0	0
0	9	15	3	0	4	17	0	0
0	1	4	8	6	6	10	0	0
0	16	10	0	11	13	6	0	0
0	0	2	9	3	0	8	0	0
0	2	14	12	5	14	1	0	0
0	0	0	0	0	0	0	0	0

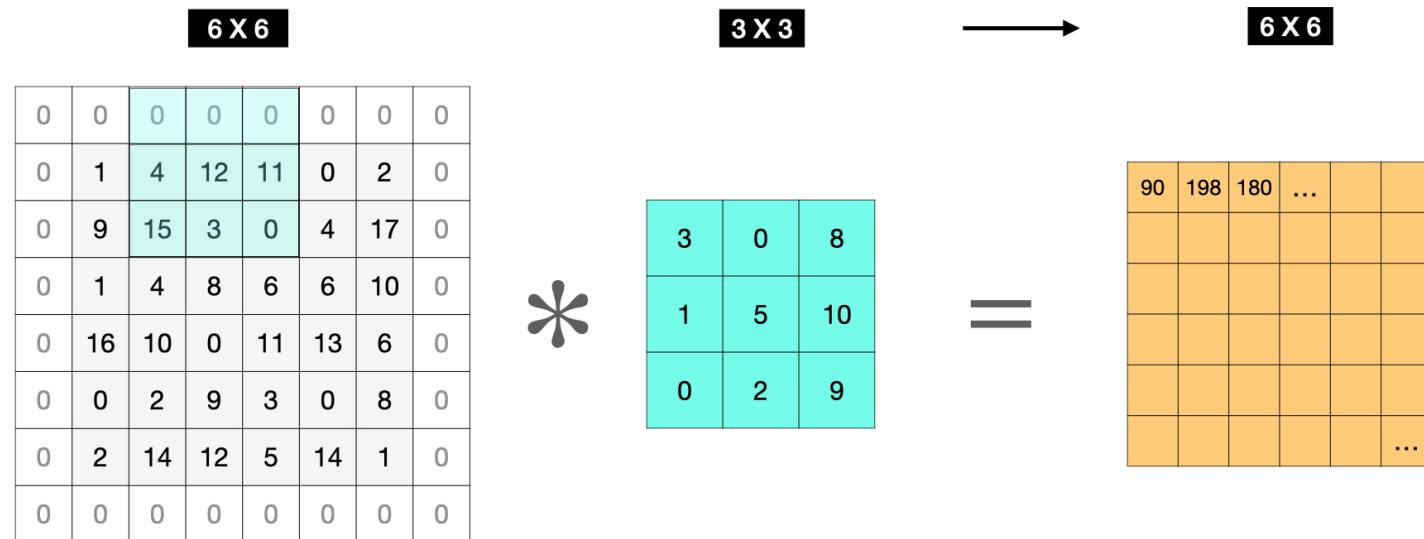


3	0	8
1	5	10
0	2	9

=

90	198	180	...		
					...

Padding



$$\begin{array}{l} n \times n \quad \text{Image} \\ f \times f \quad \text{filter} \\ p = 1 \quad \text{padding} \end{array} \} \quad (n + 2p) - f + 1 \times (n + 2p) - f + 1$$

Valid convolution: no padding / $p=0$
 Same convolution: output size = input size

Strided Convolutions

Strided Convolutions

1	4	12	11	0	2
9	15	3	0	4	17
1	4	8	6	6	10
16	10	0	11	13	6
0	2	9	3	0	8
2	14	12	5	14	1



3	0	8
1	5	10
0	2	9

=

293	

Strided Convolutions

Stride = 2

1	4	12	11	0	2
9	15	3	0	4	17
1	4	8	6	6	10
16	10	0	11	13	6
0	2	9	3	0	8
2	14	12	5	14	1



3	0	8
1	5	10
0	2	9

=

293	158

Strided Convolutions

Stride = 2

1	4	12	11	0	2
9	15	3	0	4	17
1	4	8	6	6	10
16	10	0	11	13	6
0	2	9	3	0	8
2	14	12	5	14	1

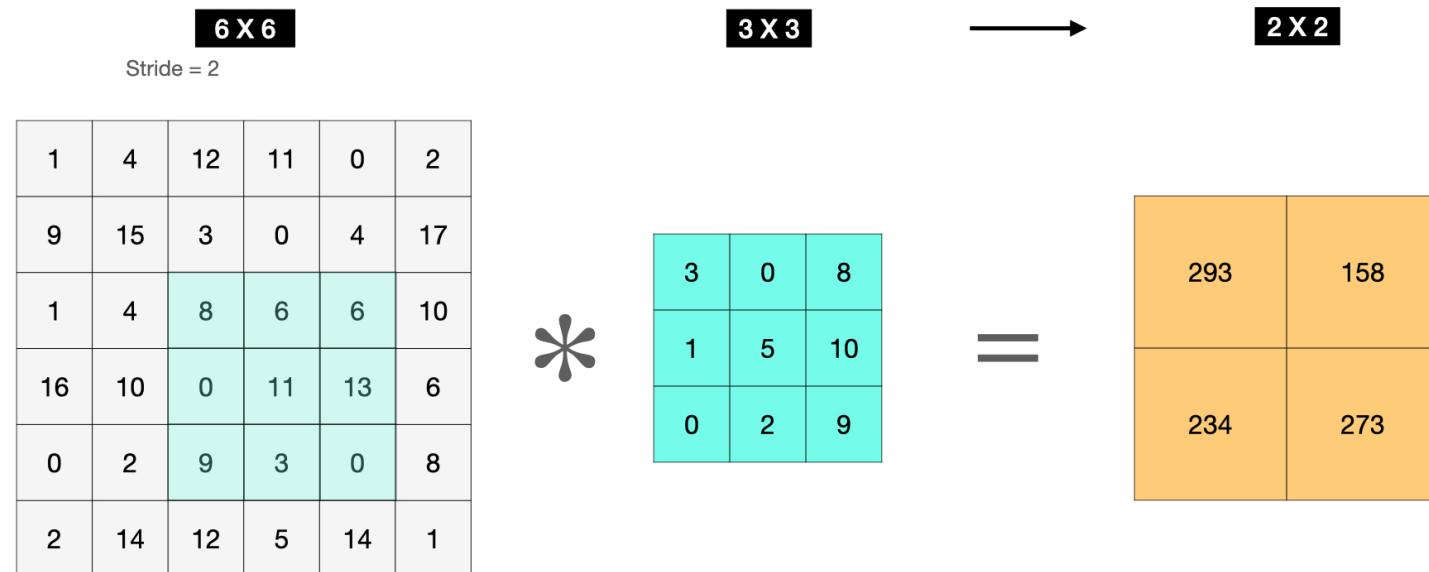


3	0	8
1	5	10
0	2	9

=

293	158
234	

Strided Convolutions



$$\left. \begin{array}{l} n \times n \\ f \times f \\ p \\ s \end{array} \right\} \text{Image filter padding stride} \quad \left| \frac{n+2p-f}{s} + 1 \times \frac{n+2p-f}{s} + 1 \right|$$

Convolutions on volumes (RGB images)

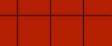
$$\begin{matrix} * & \begin{matrix} \text{3x3x3} \end{matrix} & = & \begin{matrix} \text{4x4} \end{matrix} \end{matrix}$$


The diagram illustrates the multiplication of a 3x3x3 tensor by a scalar. On the left, a 3x3x3 tensor is shown as a stack of three 3x3 matrices. A black asterisk (*) is positioned above it. To the right of the equals sign (=) is a 4x4 matrix, represented as a stack of four 4x4 red squares.

The diagram shows a convolution operation. On the left, a 6x6x3 input tensor is shown as a 3D grid with three green depth channels. To its right is a multiplication symbol (*). Next is the text "N filters" in red, followed by an equals sign (=). To the right of the equals sign are three ellipsis symbols (...). On the far right is a 4x4xN output tensor, represented as a 3D grid with N orange depth channels. The output tensor has a black border and a red circle around the N in the dimensions.

$$\begin{array}{c} * \\ \text{3x3x3} \\ = \\ \text{4x4} \end{array}$$

$$* = \text{relu}(\text{ } \boxed{4 \times 4} \text{ } + b)$$



A diagram showing a 3D tensor input. It consists of a grid of red squares representing the depth channel, with a green border around the top and left sides. Below the grid, the dimensions $6 \times 6 \times 3$ are written in a black box.

$*$ **N filters** = $\text{relu}(\dots + b)$

$$* \quad \quad \quad = \quad \text{relu}(\quad \quad \quad + b)$$

Layers in ConvNets

The building blocks (i.e. most commonly used types of layers in CNNs)

- Convolutional layers
- Pooling layers
 - Max Pooling
 - Average Pooling
 - Global Pooling
- Dense (Fully Connected) layers

Pooling

1	4	2	11
6	13	8	3
10	0	17	7
9	2	3	5

Max Pooling

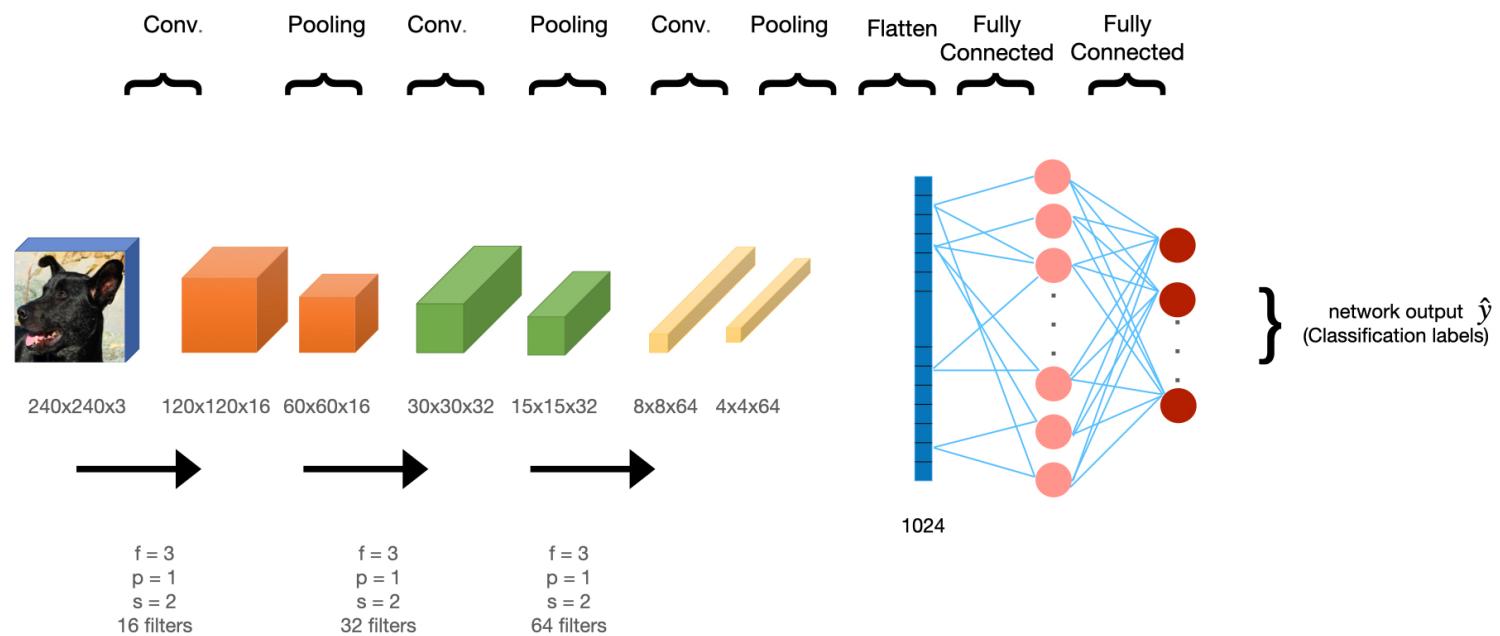
pool size 2x2
stride 2

13	11
10	17

Average Pooling

6	6
5	8

Schematic Description of a ConvNet



CNNs in Tensorflow

```
In [5]: import tensorflow
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import models

model = models.Sequential()
model.add(layers.Conv2D(8, (3, 3), activation='relu', input_shape=(80,80,3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(16, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(64,activation='relu'))
model.add(layers.Dense(4, activation='softmax'))
```

Hands-on

- Simple classification example using a ConvNet

Classification/Regression

- Regression:
 - output is a prediction on a quantity which takes continuous values
- Classification:
 - output is a prediction on a class, i.e. discrete labels
 - binary classification
 - multi-class classification
 - single-label classification
 - multi-label classification

steps

1. Data

- set of 8 different classes of images: airplane, car, cat, dog, flower, fruit, motorbike, person
- create the relevant datasets

2. CNN model

- build the CNN model, train it with the given data, check performance
- we will observe overfitting

3. Improve previous CNN model: i.e. avoid overfitting

- Dropout (interfere in the model architecture)
- Augmentation (interfere with the data)

4. Transfer learning

- Do not use own CNN model, but borrow a pretrained one. Use it on our given data.

Dropout

Dropout: A Simple Way to Prevent Neural Networks from Overfitting, JMLR (2014)
<http://jmlr.org/papers/v15/srivastava14a.html> (<http://jmlr.org/papers/v15/srivastava14a.html>)

SRIVASTAVA, HINTON, KRIZHEVSKY, SUTSKEVER AND SALAKHUTDINOV

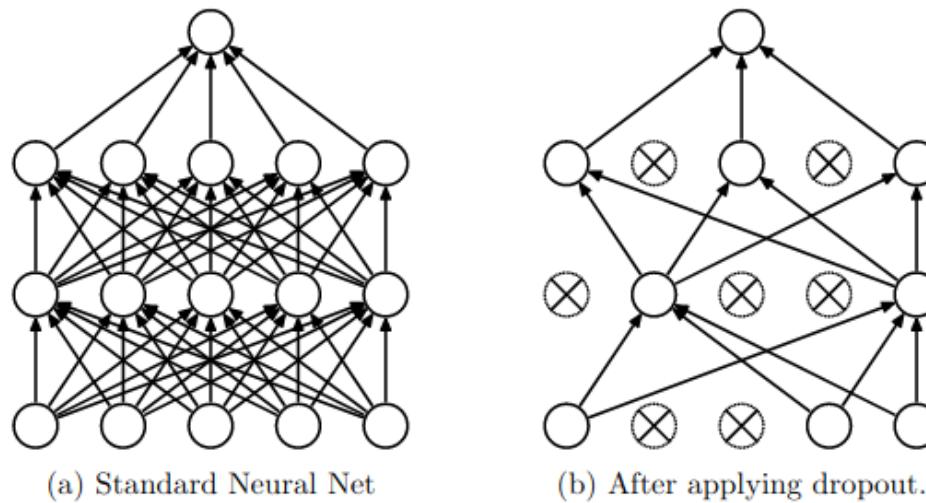


Figure 1: Dropout Neural Net Model. **Left:** A standard neural net with 2 hidden layers. **Right:** An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

Augmentation

Augmenting the training dataset, by applying transformations to existing data (images in this case): flip, rotate, shear, shift, zoom, color distortion, etc.



Transfer Learning

Using a pretrained network

