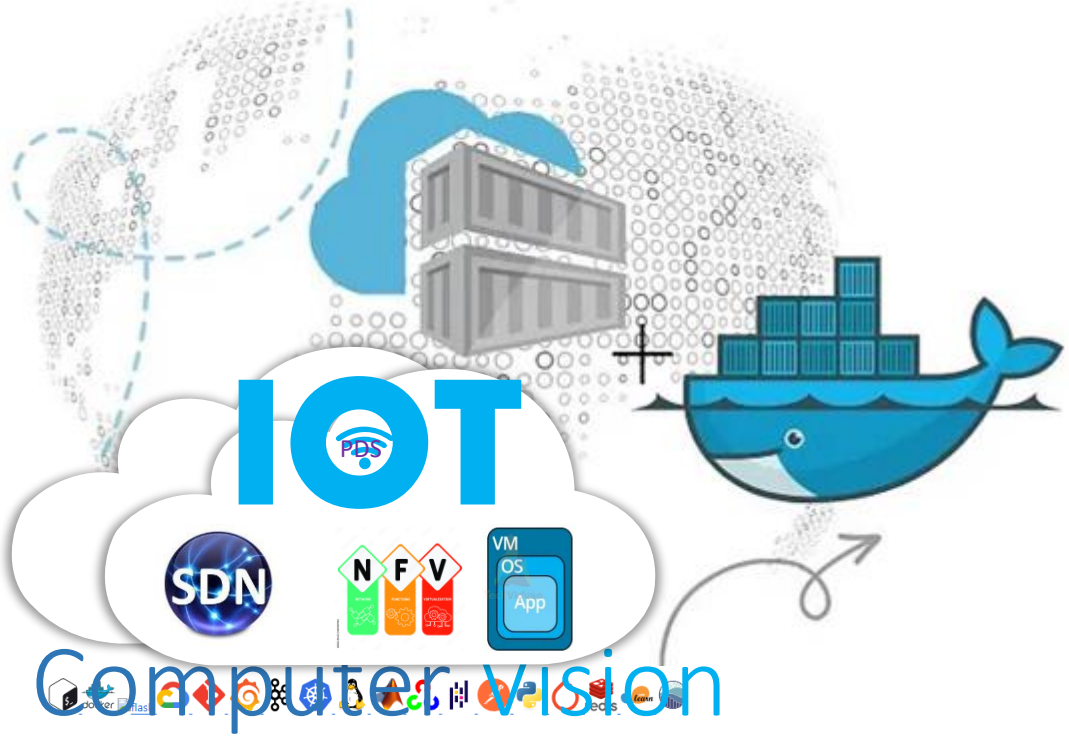


Computer Vision

Chapter 5:

Convolusion Neural Network

https://github.com/amelcharolinesgn2/IoT_simulator-mqtt-NodeRed

https://github.com/siagianp
https://www.youtube.com/@AmelOline/videos

Why CNN for Image

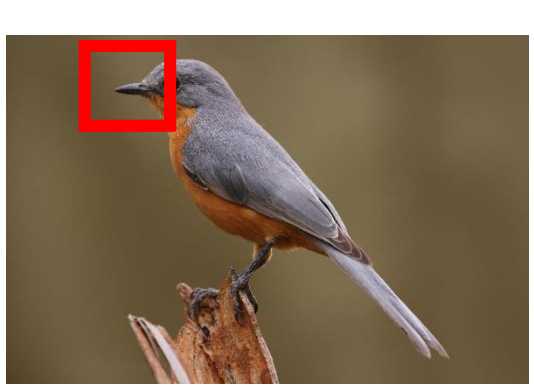
- Some patterns are much smaller than the whole image

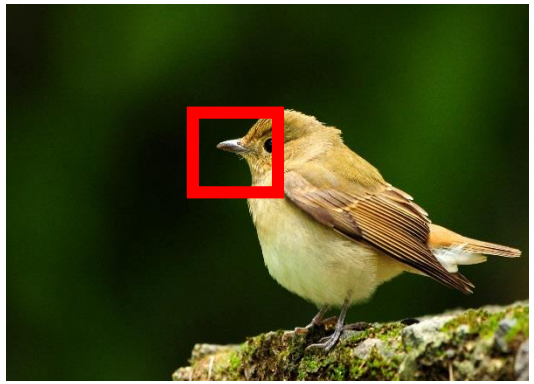
A neuron does not have to see the whole image to discover the pattern.

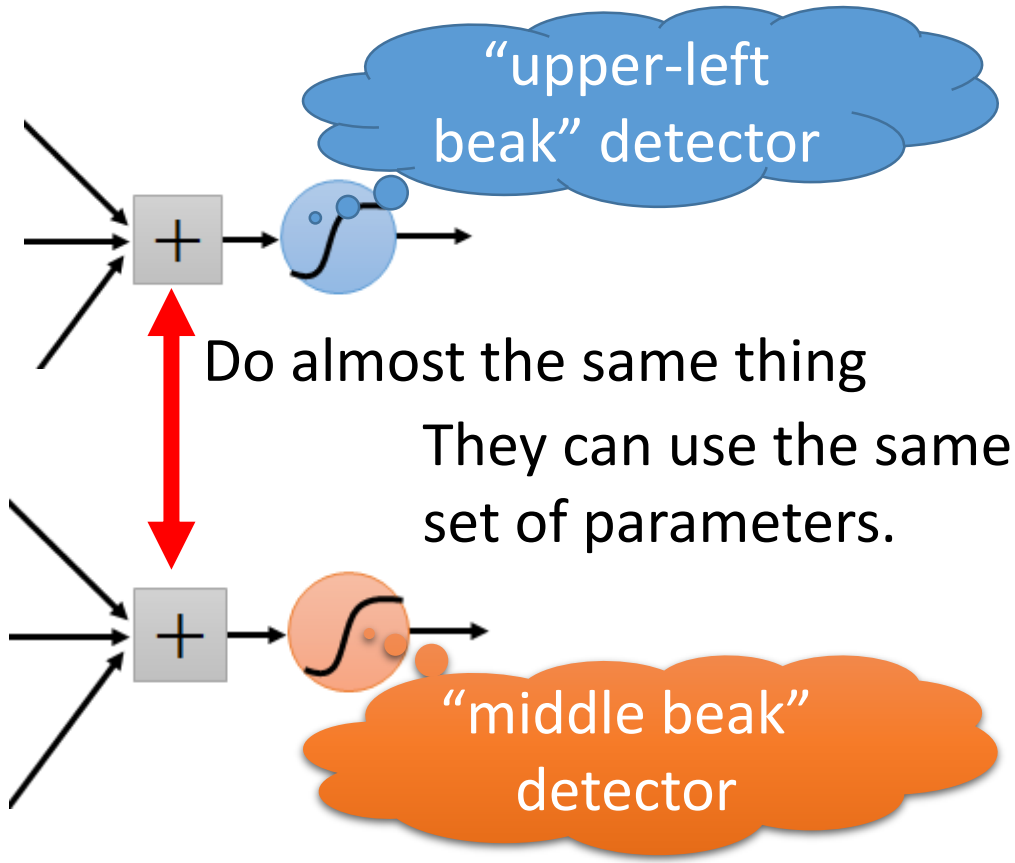
Connecting to small region with less parameters

Why CNN for Image

- The same patterns appear in different regions.







Why CNN for Image

- Subsampling the pixels will not change the object

bird

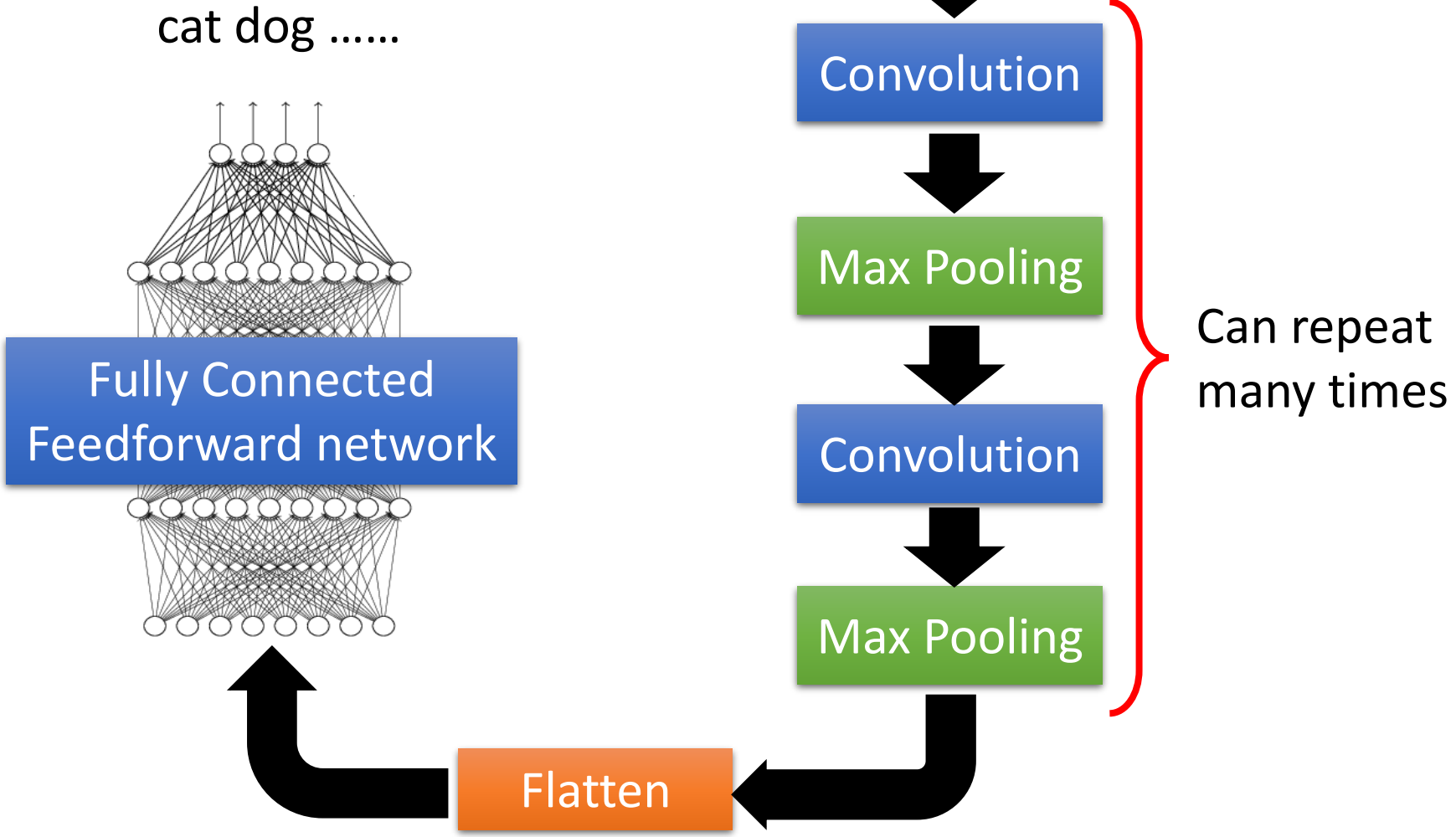
subsampling

bird

We can subsample the pixels to make image smaller

→ Less parameters for the network to process the image

The whole CNN



The whole CNN

Property 1

- Some patterns are much smaller than the whole image

Property 2

- The same patterns appear in different regions.

Property 3

- Subsampling the pixels will not change the object

Convolution

Max Pooling

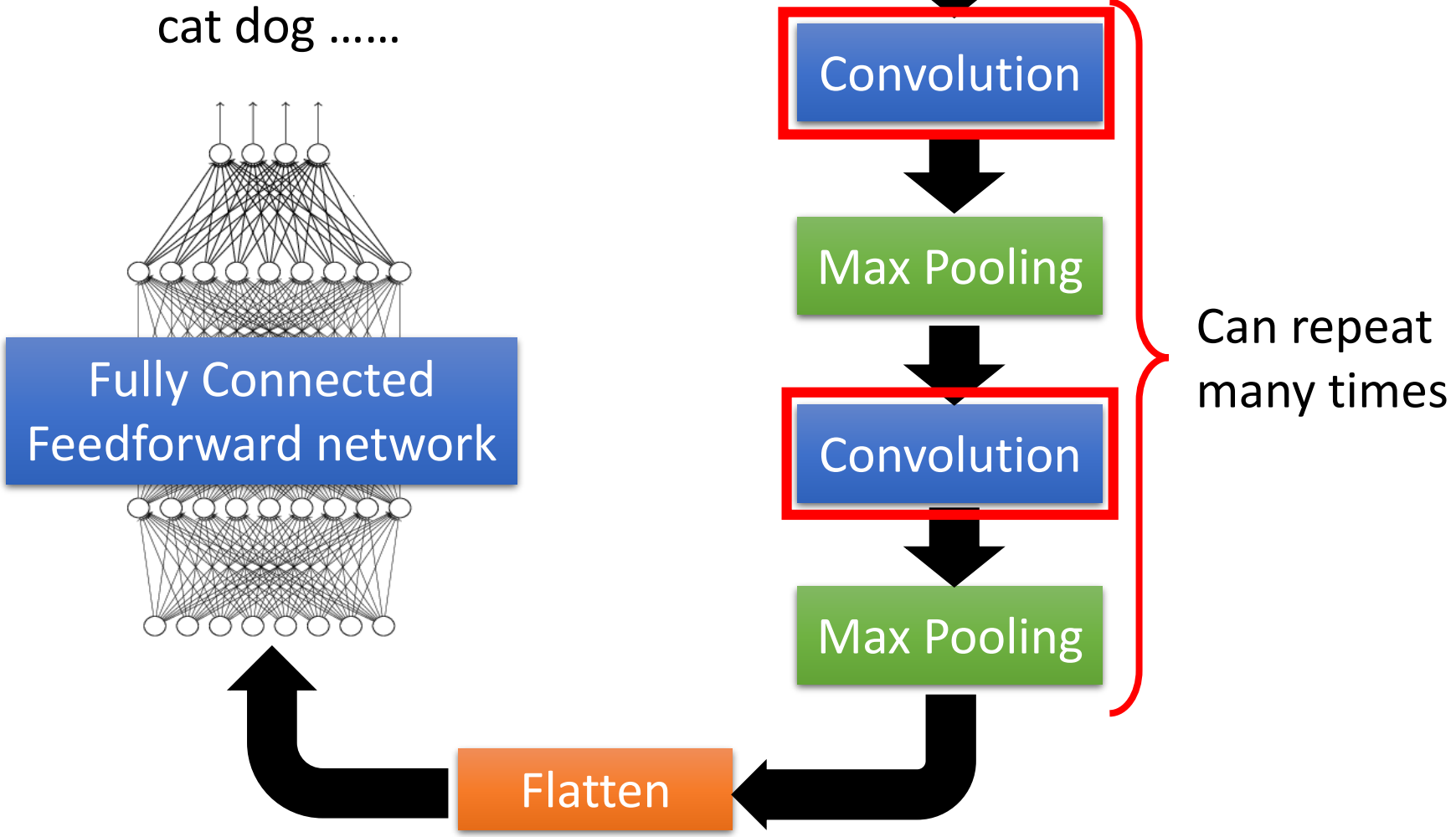
Convolution

Max Pooling

Can repeat many times

Flatten

The whole CNN



CNN – Convolution

Those are the network parameters to be learned.

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

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

Matrix

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

Matrix

⋮

Property 1

Each filter detects a small pattern (3 x 3).

CNN – Convolution

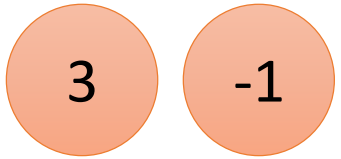
1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

stride=1

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

6 x 6 image



CNN – Convolution

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

If stride=2

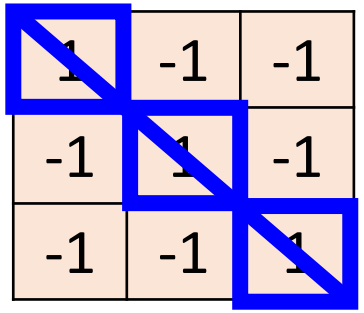
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

3 -3

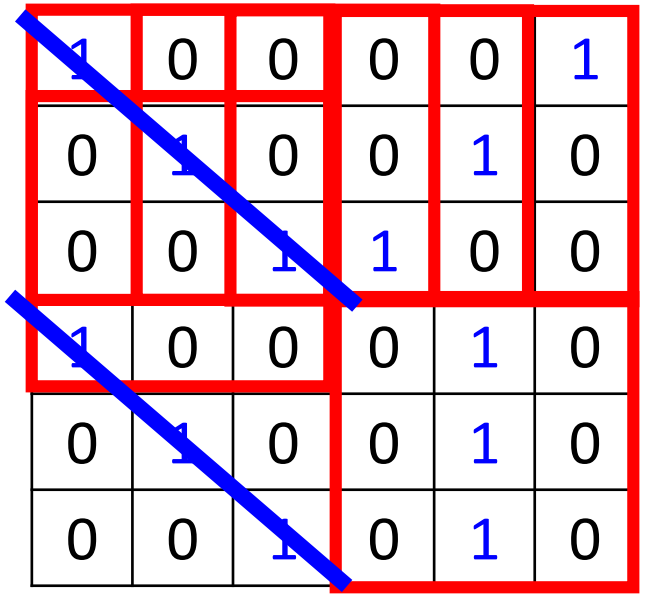
We set stride=1 below

CNN – Convolution

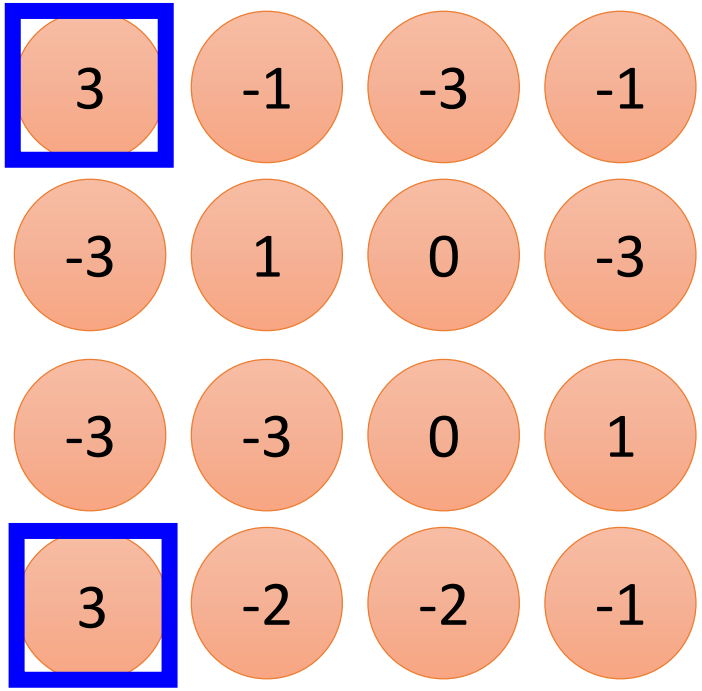


Filter 1

stride=1



6 x 6 image



Property 2

CNN – Convolution

-1	1	-1
-1	1	-1
-1	1	-1

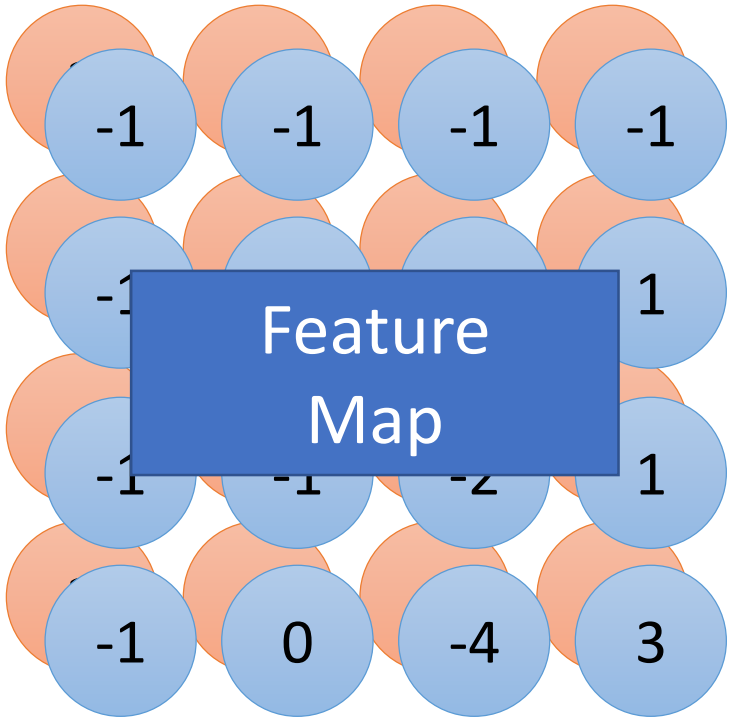
Filter 2

stride=1

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

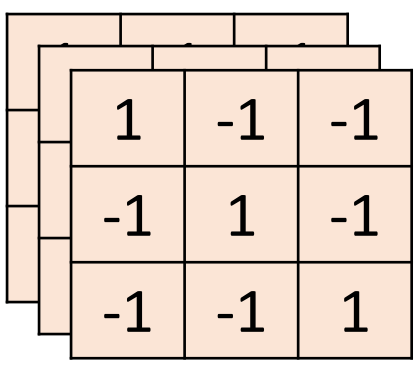
6 x 6 image

Do the same process for every filter

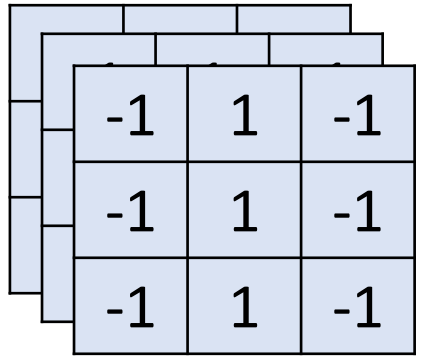


4 x 4 image

CNN – Colorful image

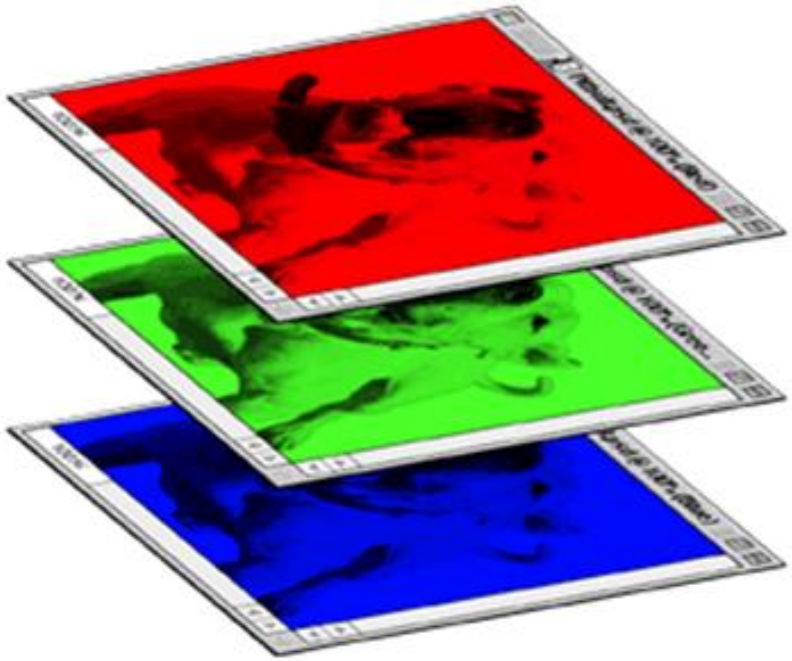


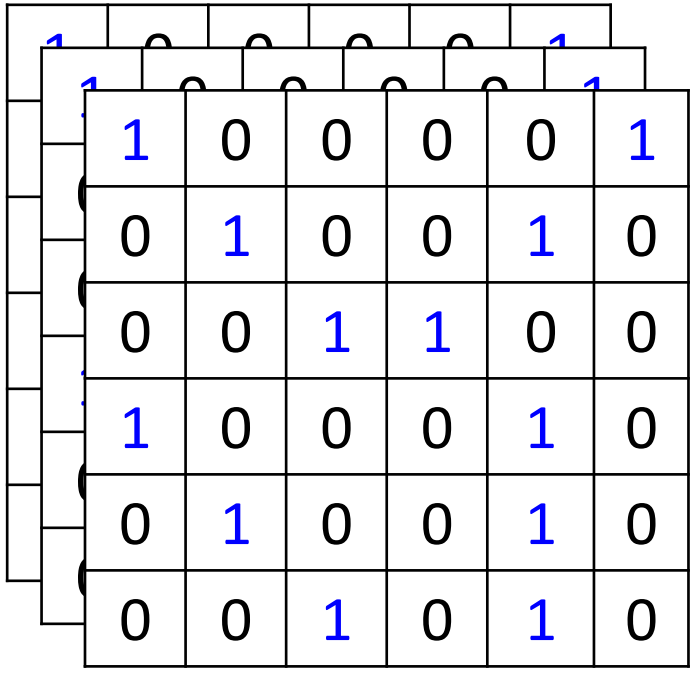
Filter 1



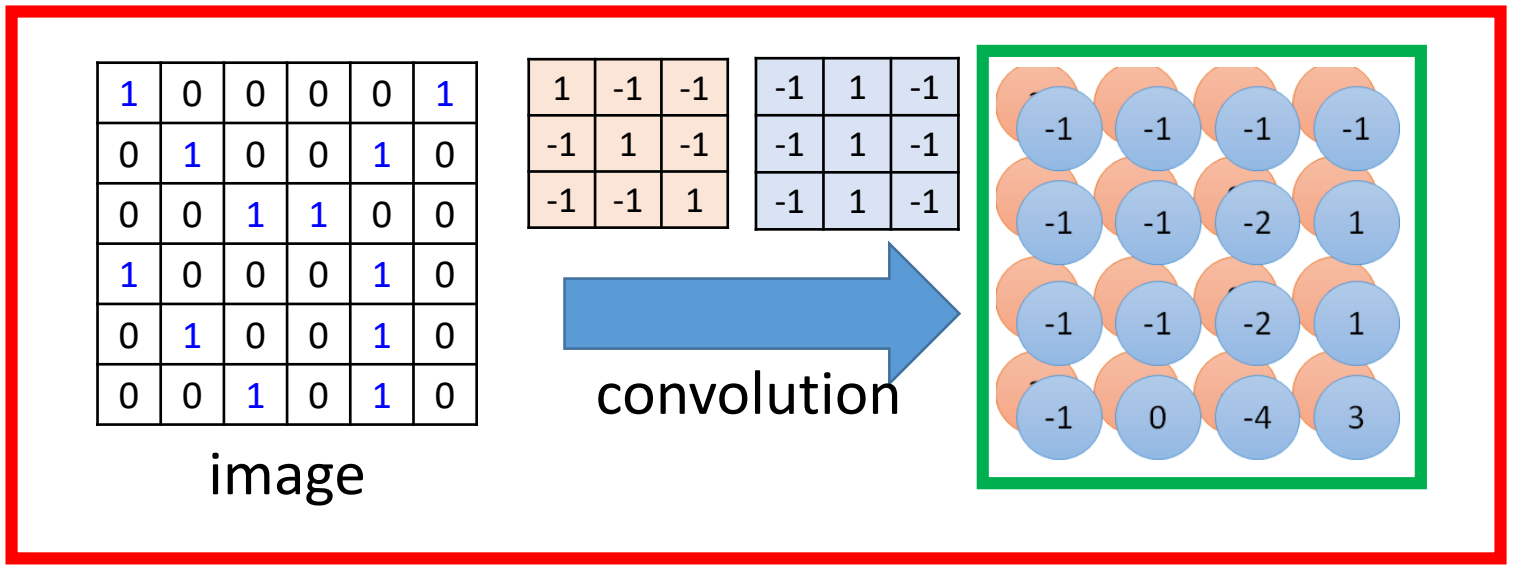
Filter 2

Colorful image

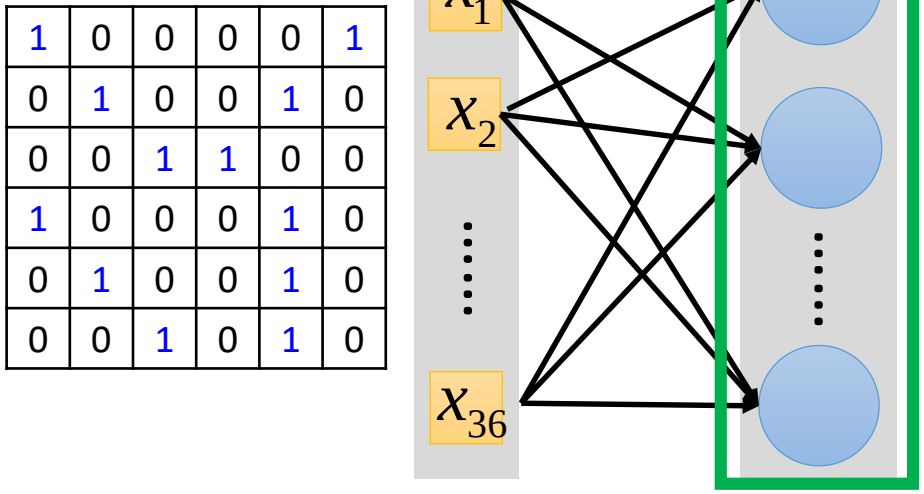


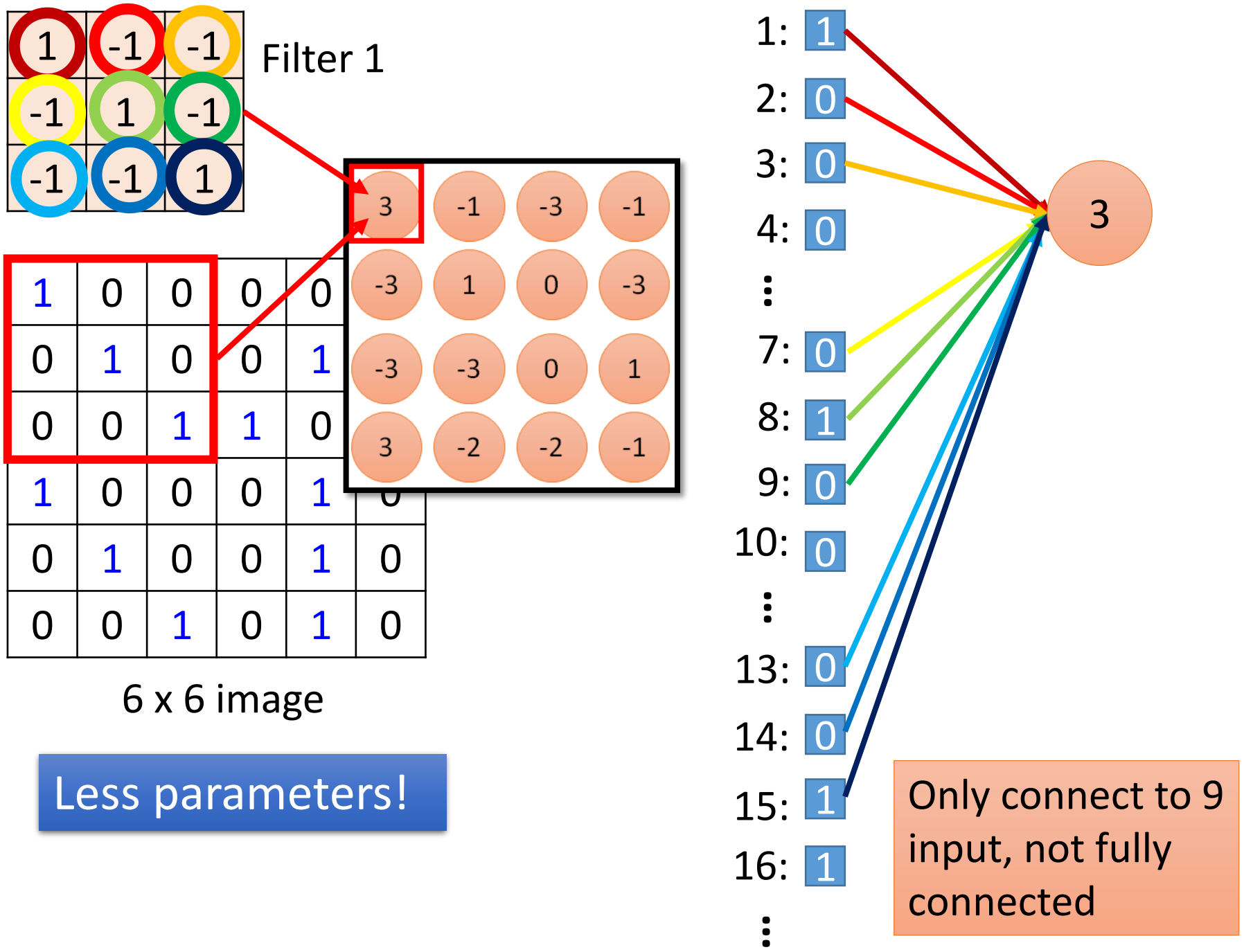


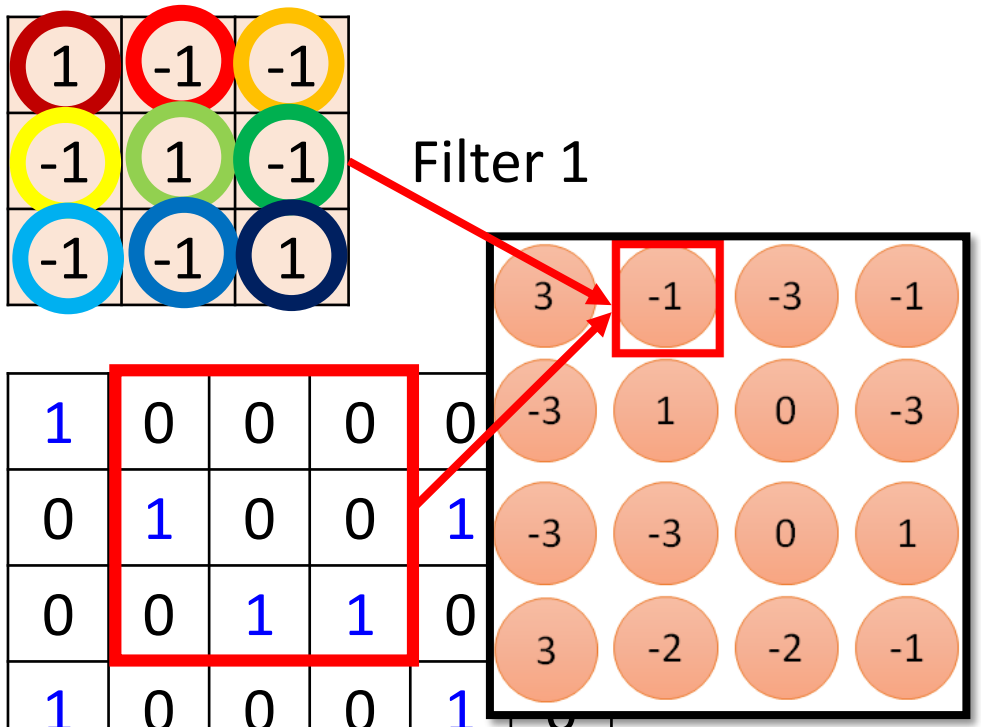
Convolution v.s. Fully Connected



Fully-connected

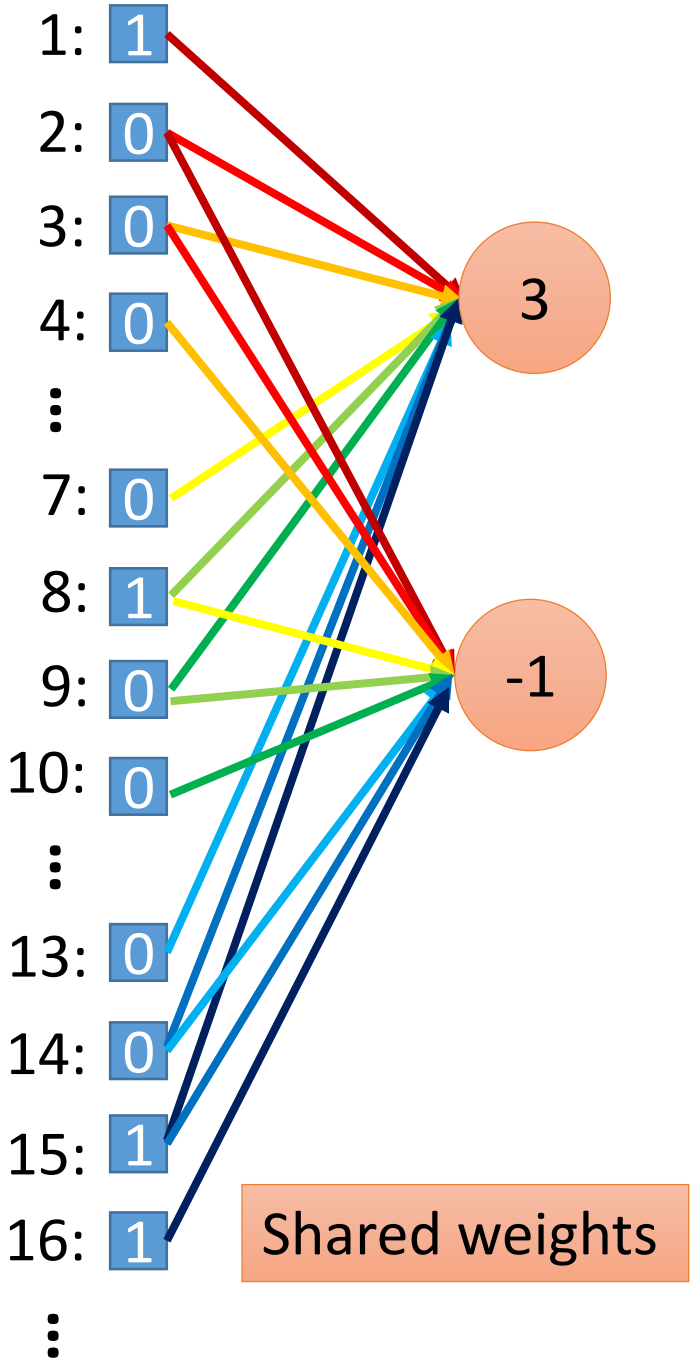



1 -1 -1
-1 1 -1
-1 -1 1
Filter 1
3 -1 -3 -1
-3 1 0 -3
-3 -3 0 1
3 -2 -2 -1
1 0 0 0 0
0 1 0 0 1
0 0 1 1 0
1 0 0 0 1
0 1 0 0 1 0
0 0 1 0 1 0
6 x 6 image
Less parameters!
1: 1
2: 0
3: 0
4: 0
7: 0
8: 1
9: 0
10: 0
13: 0
14: 0
15: 1
16: 1
3
Only connect to 9 input, not fully connected

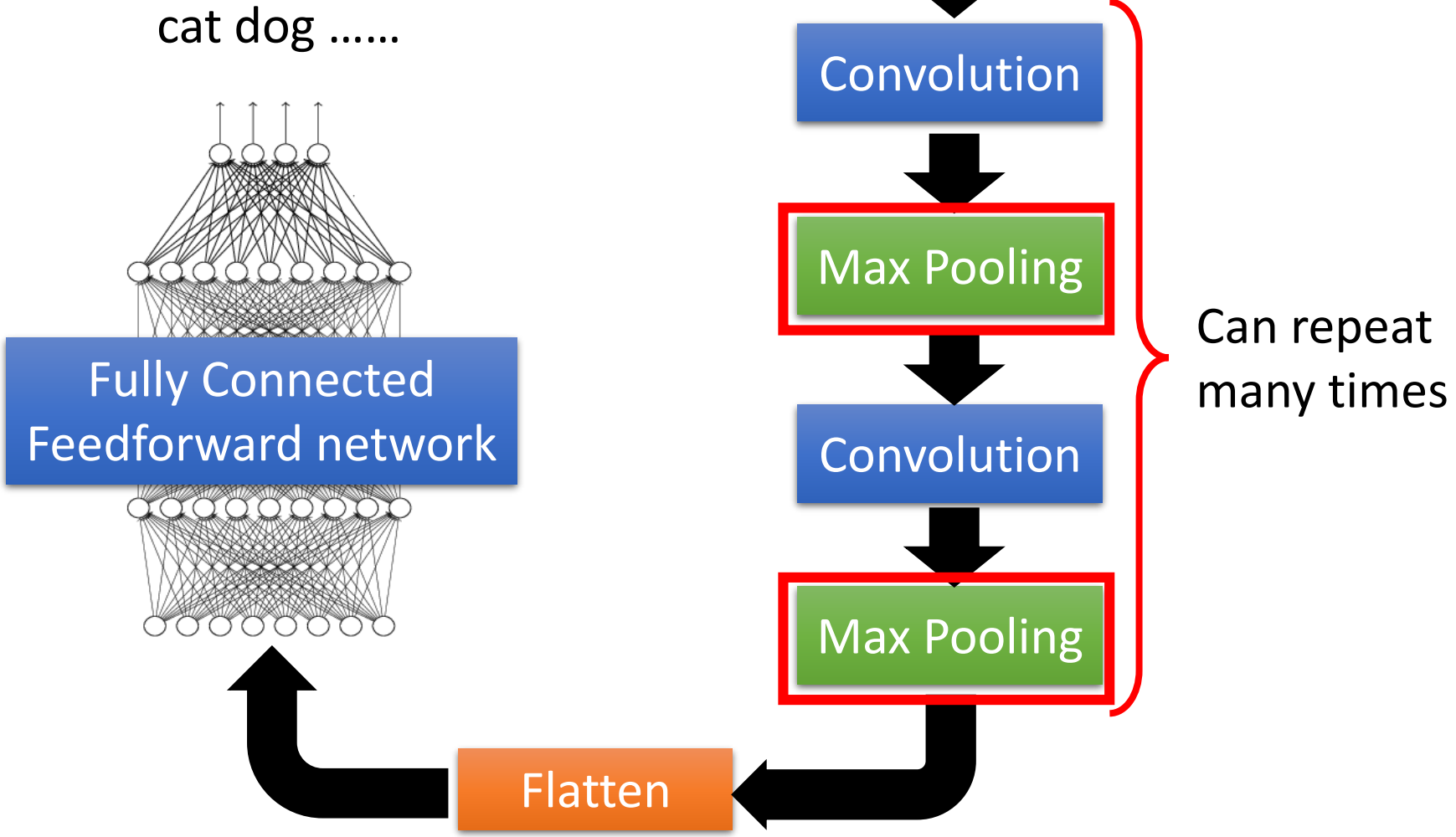


Less parameters!

Even less parameters!



The whole CNN



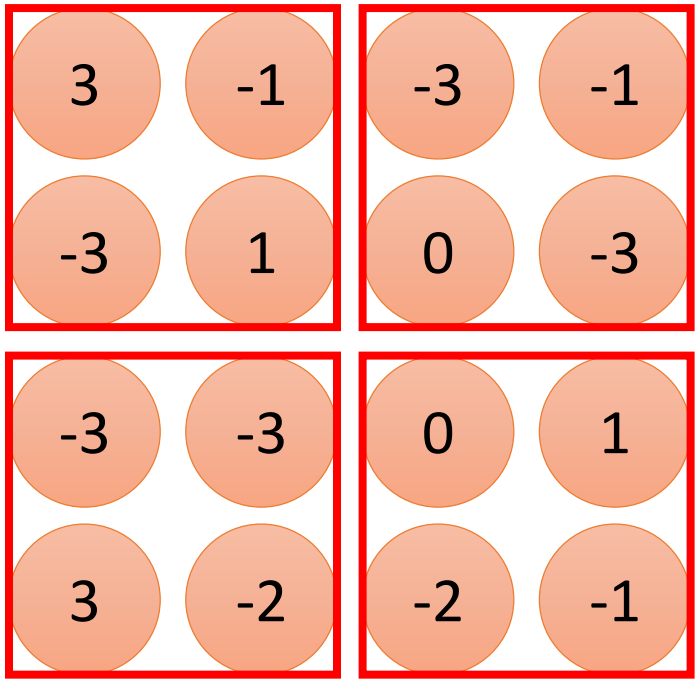
CNN – Max Pooling

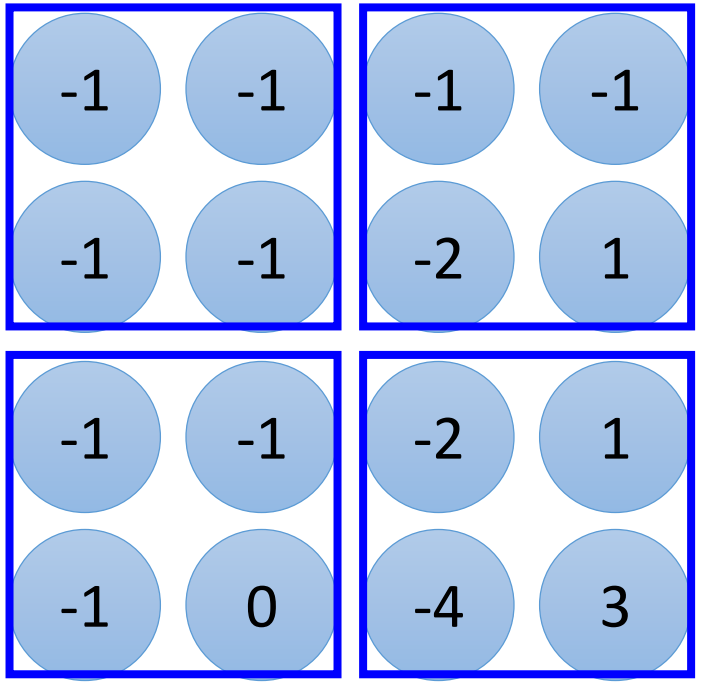
1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

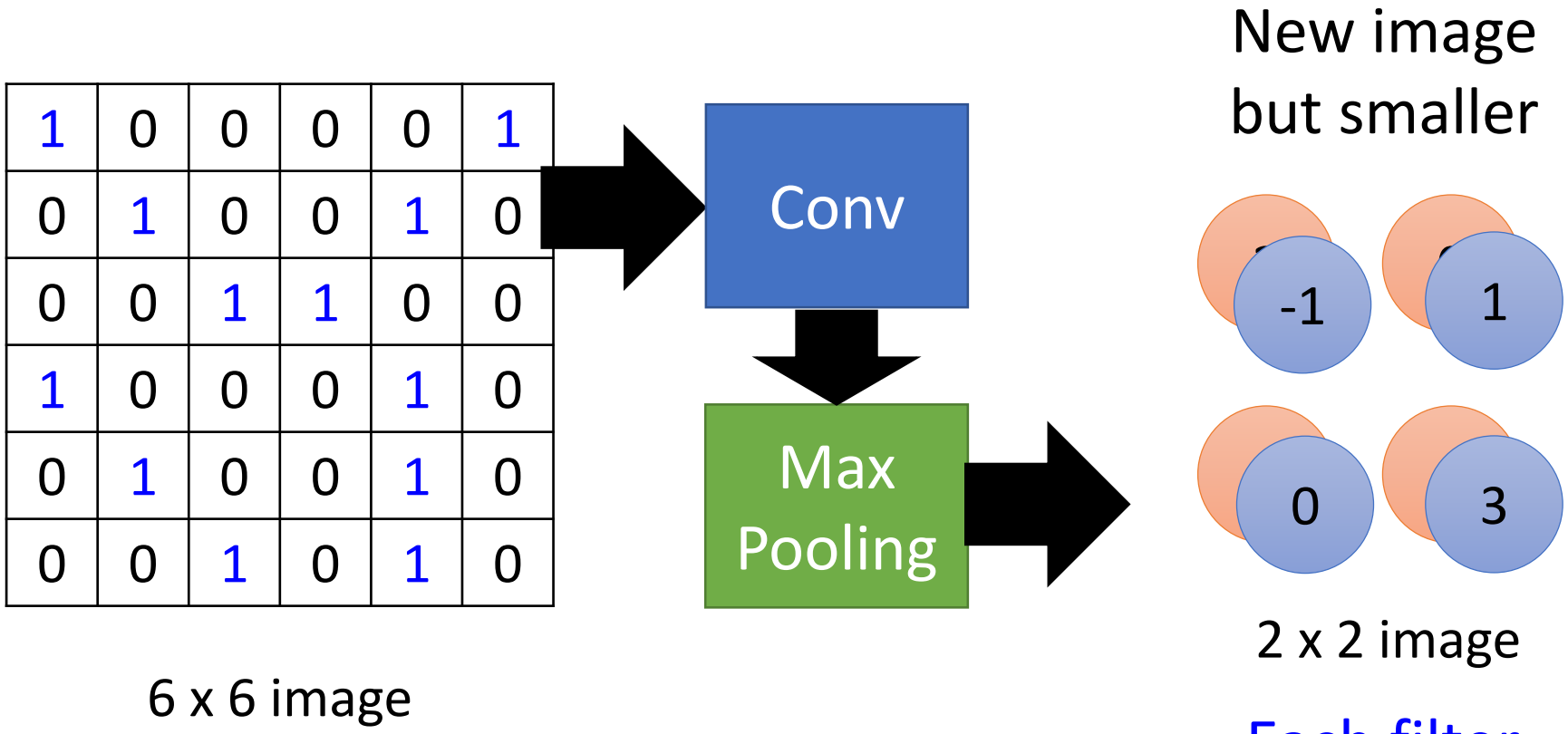
-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

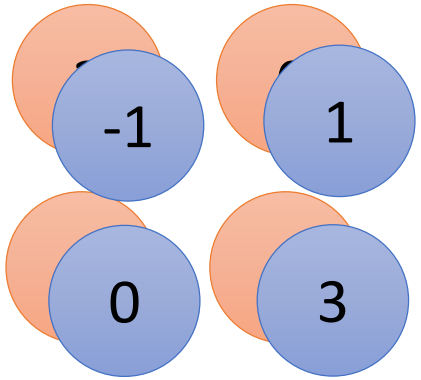




CNN – Max Pooling



The whole CNN



A new image

Convolution

Max Pooling

Convolution

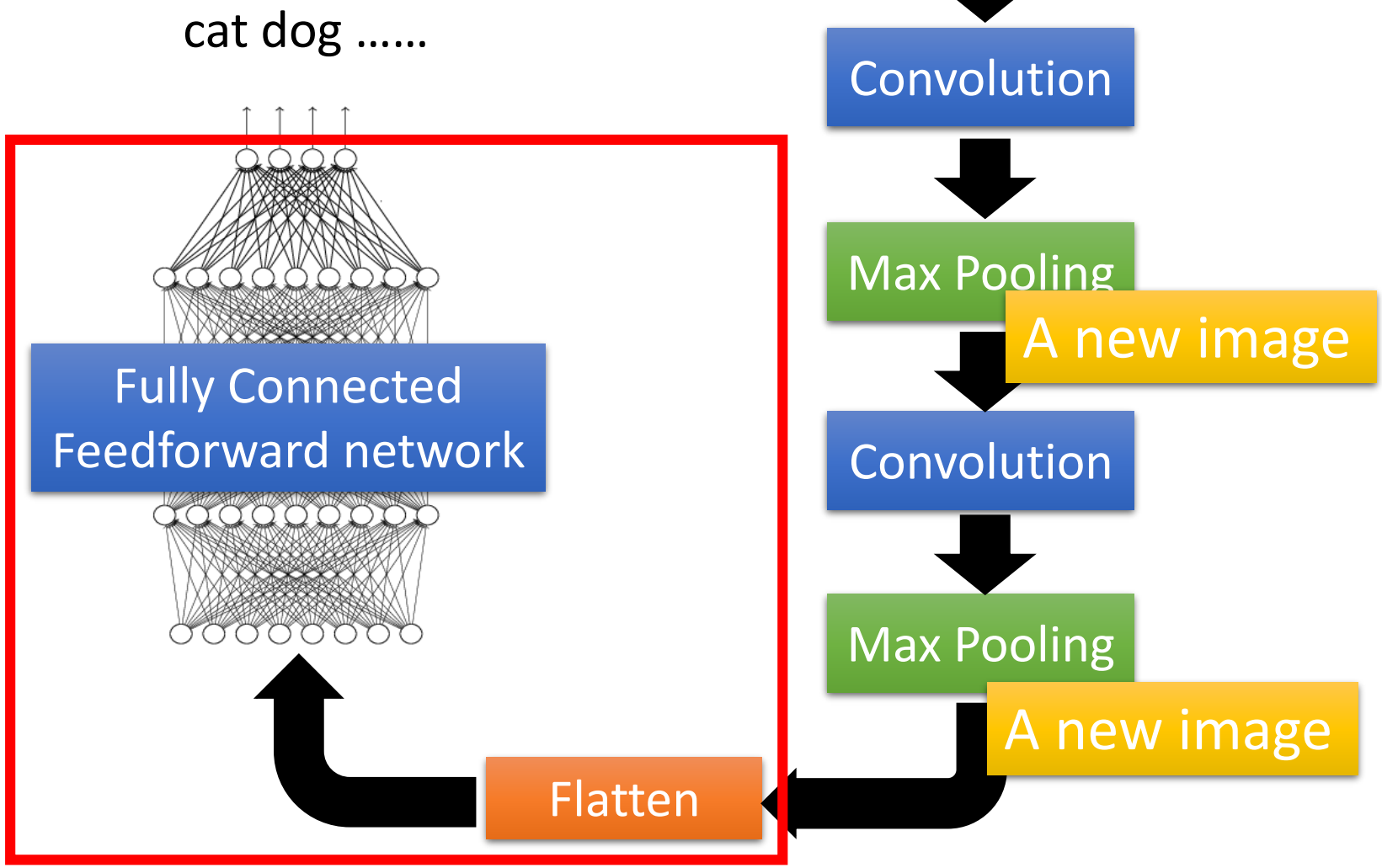
Max Pooling

Can repeat many times

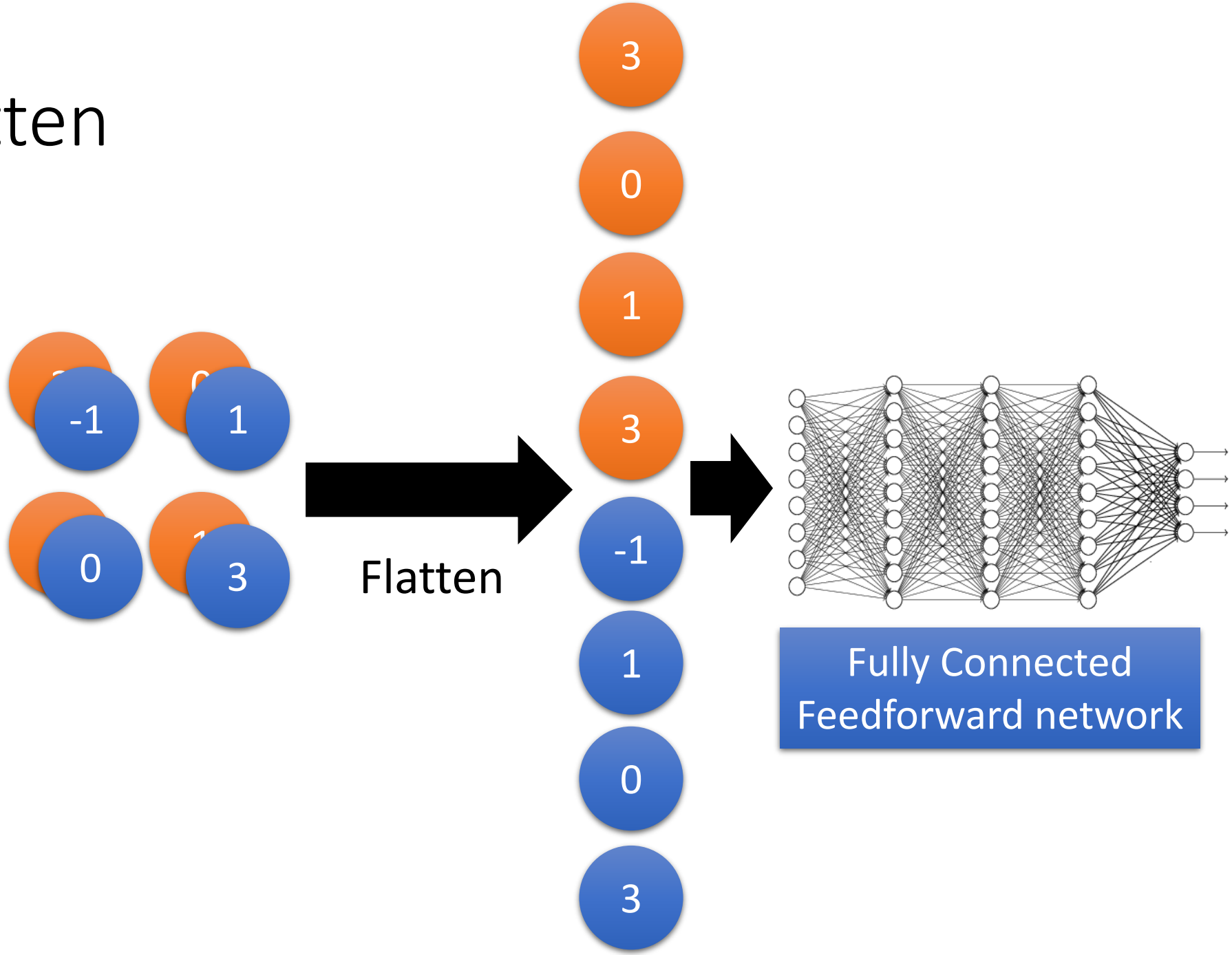
Smaller than the original image

The number of the channel is the number of filters

The whole CNN



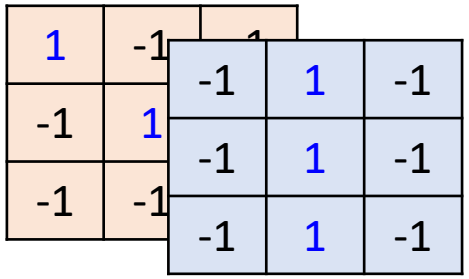
Flatten


-1
1
0
3
Flatten
3
0
1
3
-1
1
0
3
Fully Connected
Feedforward network

CNN in Keras

Only modified the ***network structure*** and ***input format (vector -> 3-D tensor)***

```
model2.add( Convolution2D( 25,3,3,
                input_shape=(28,28,1)) )
```



.....

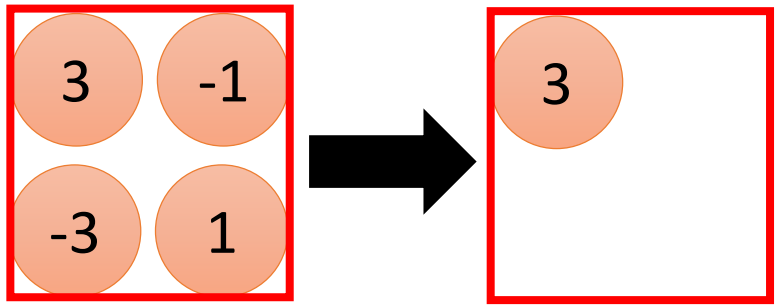
There are 25 3x3 filters.

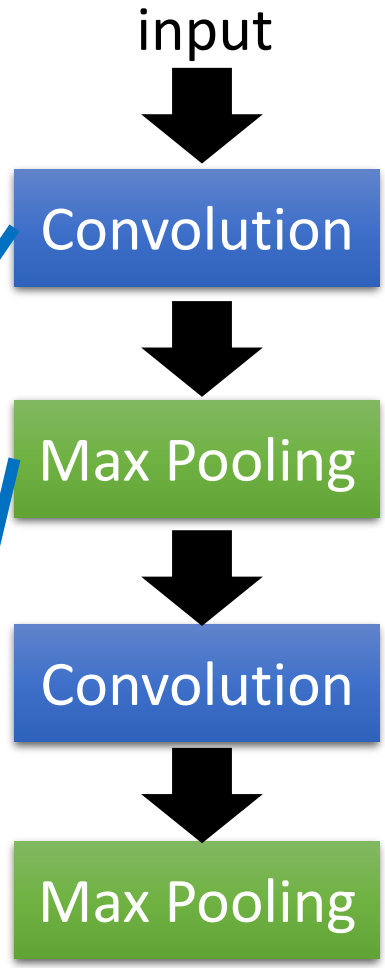
Input_shape = (28 , 28 , 1)

28 x 28 pixels

1: black/white, 3: RGB

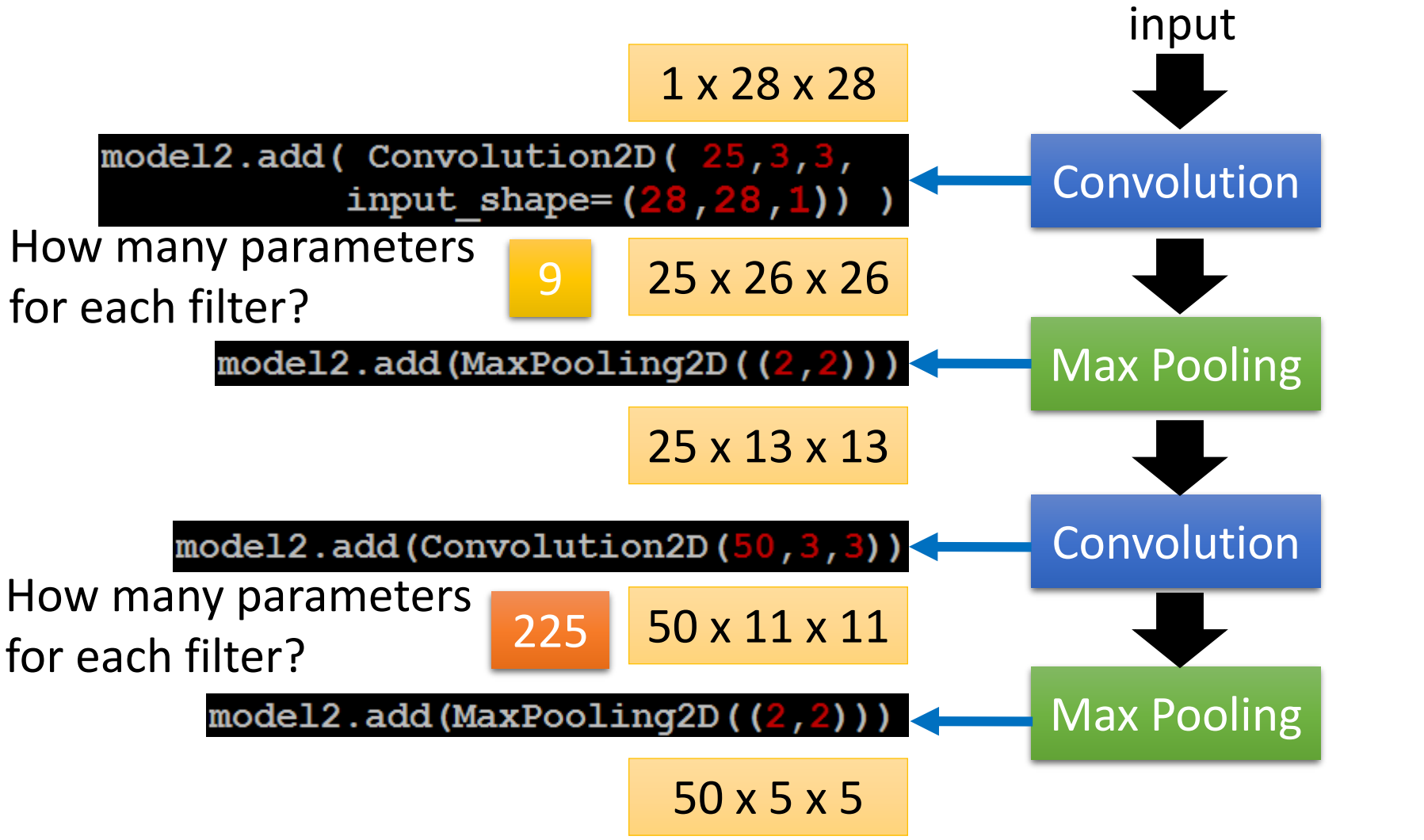
```
model2.add(MaxPooling2D((2,2)))
```





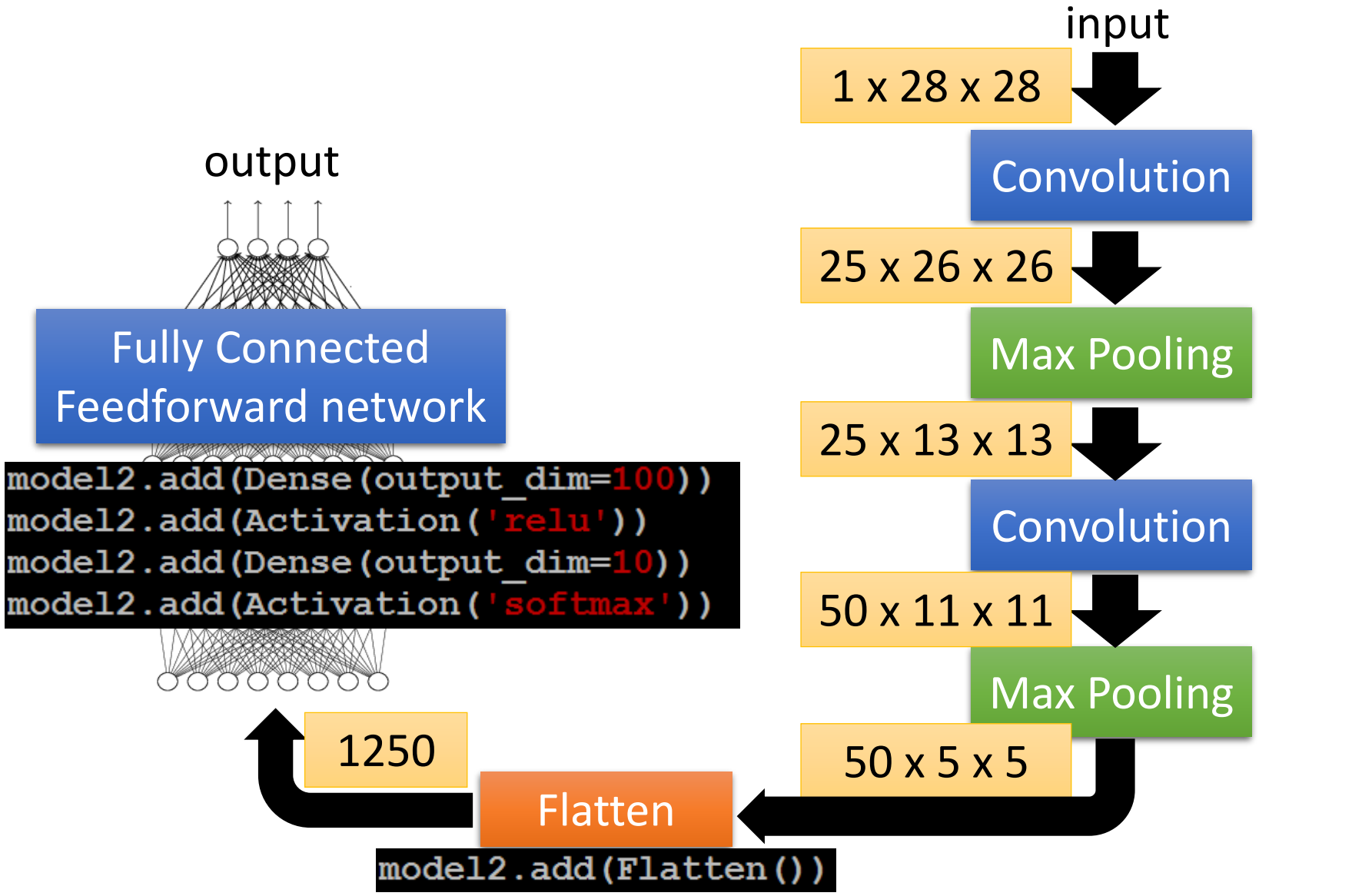
CNN in Keras

Only modified the ***network structure*** and ***input format (vector -> 3-D tensor)***



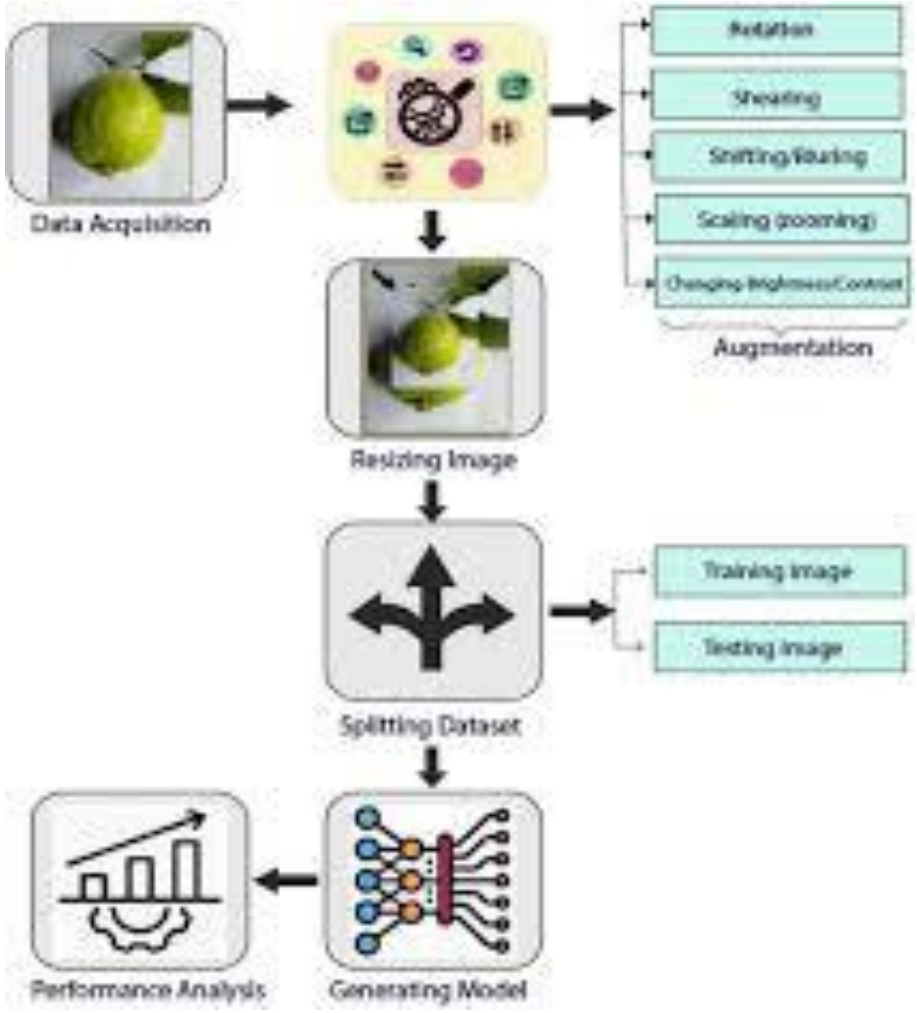
CNN in Keras

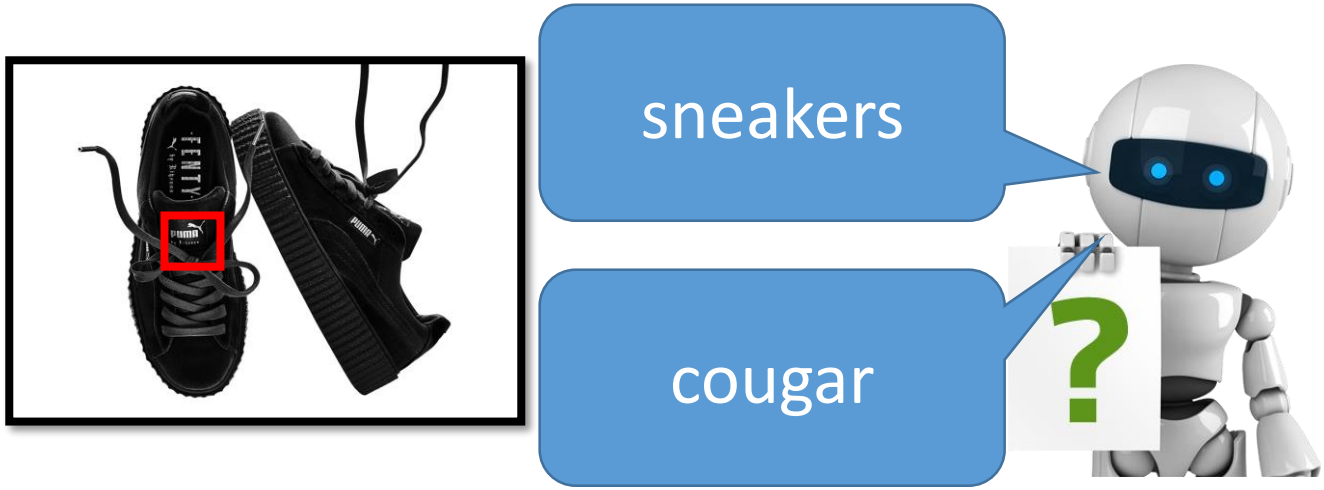
Only modified the ***network structure*** and ***input format (vector -> 3-D tensor)***



Next..

What does machine learn?

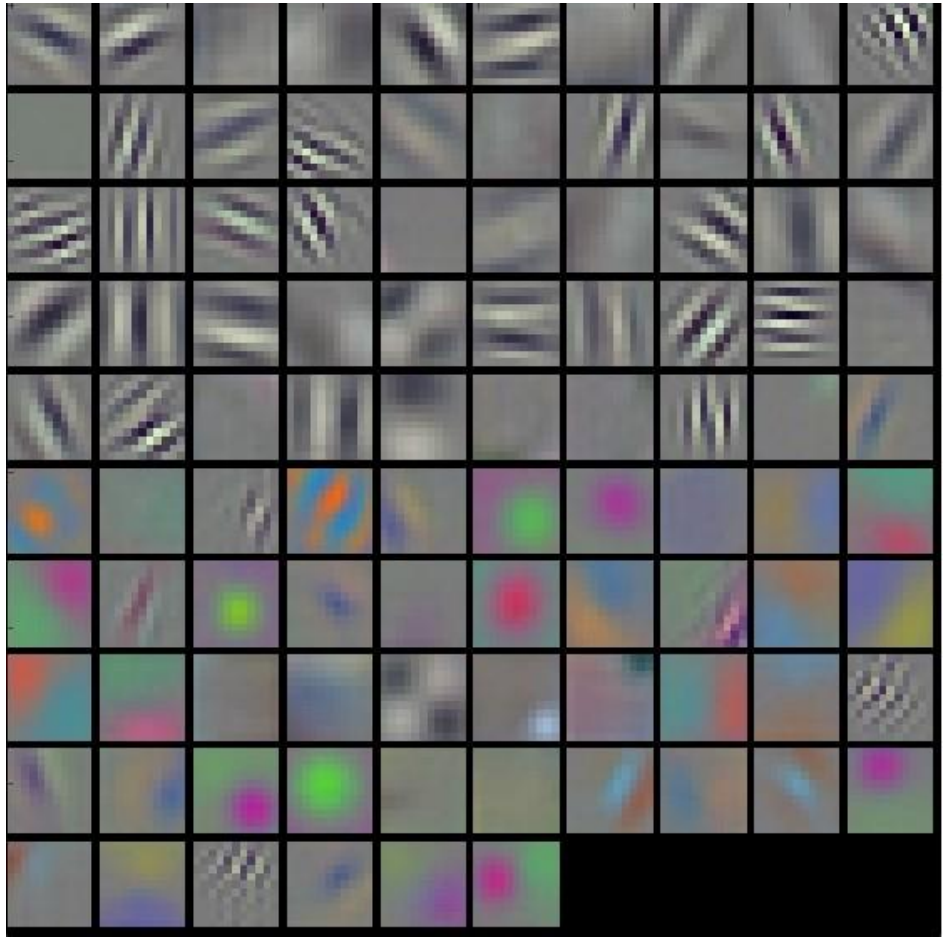




http://newsneakernews.wpengine.netdna-cdn.com/wp-content/uploads/2016/11/rihanna-puma-creeper-velvet-release-date-02.jpg

First Convolution Layer

- Typical-looking filters on the trained first layer

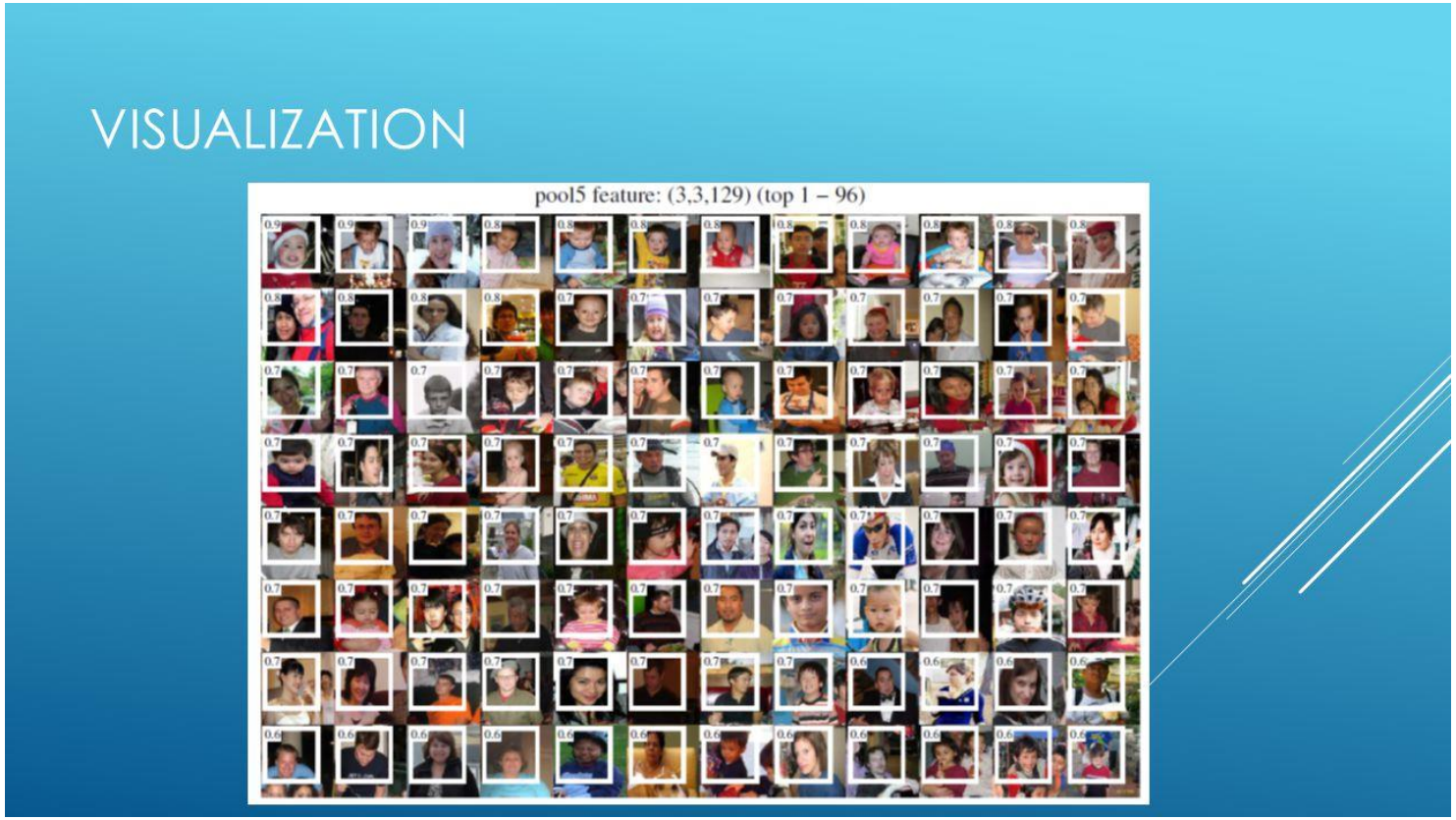


11 x 11
(AlexNet)

http://cs231n.github.io/understanding-cnn/

How about higher layers?

- Which images make a specific neuron activate



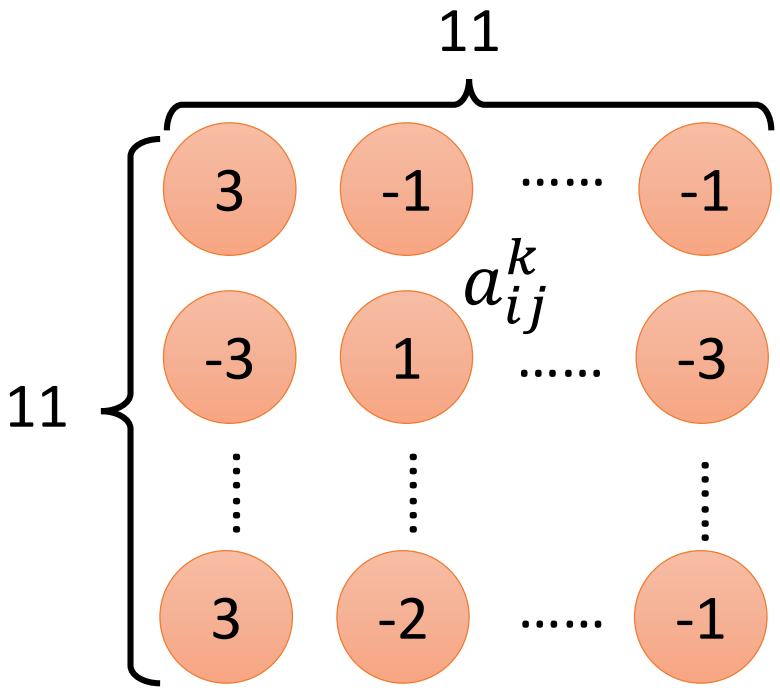
Rich feature Hierarchies for Accurate object detection and semantic segmentation Ross Girshick, Jeff Donahue, Trevor Darrell, Jitandra Malik (UC Berkeley)

What does CNN learn?

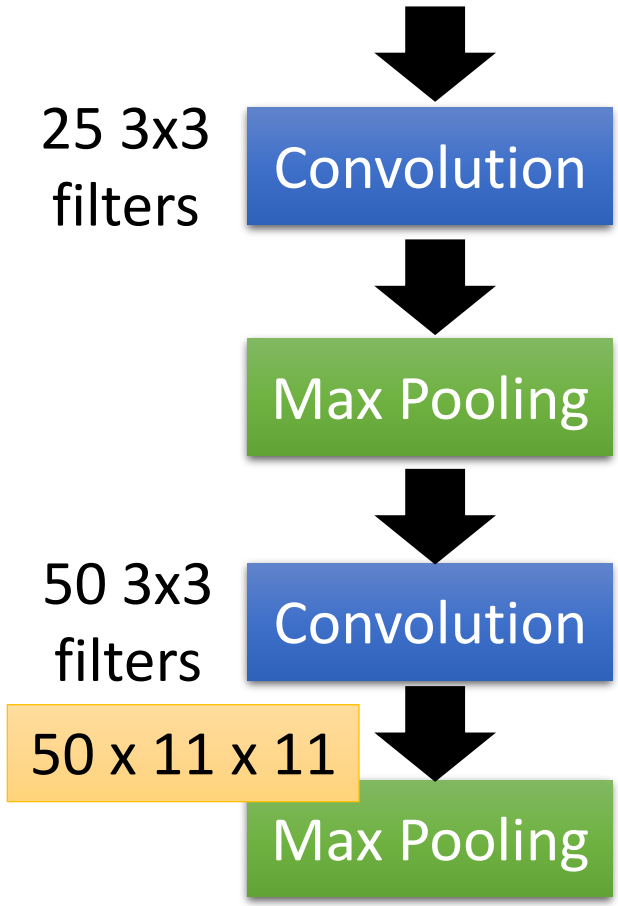
The output of the k-th filter is a 11 x 11 matrix.

Degree of the activation of the k-th filter: $a^k = \sum_{i=1}^{11} \sum_{j=1}^{11} a_{ij}^k$

$x^* = arg \max_x a^k$ (gradient ascent)



$\frac{\partial a^k}{\partial x_{ij}}$ x input



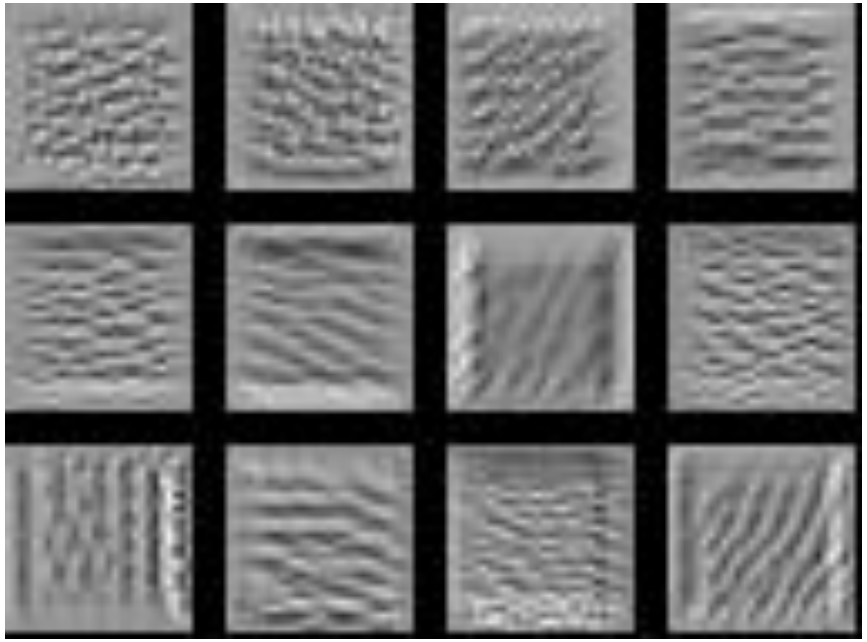
What does CNN learn?

The output of the k-th filter is a 11 x 11 matrix.

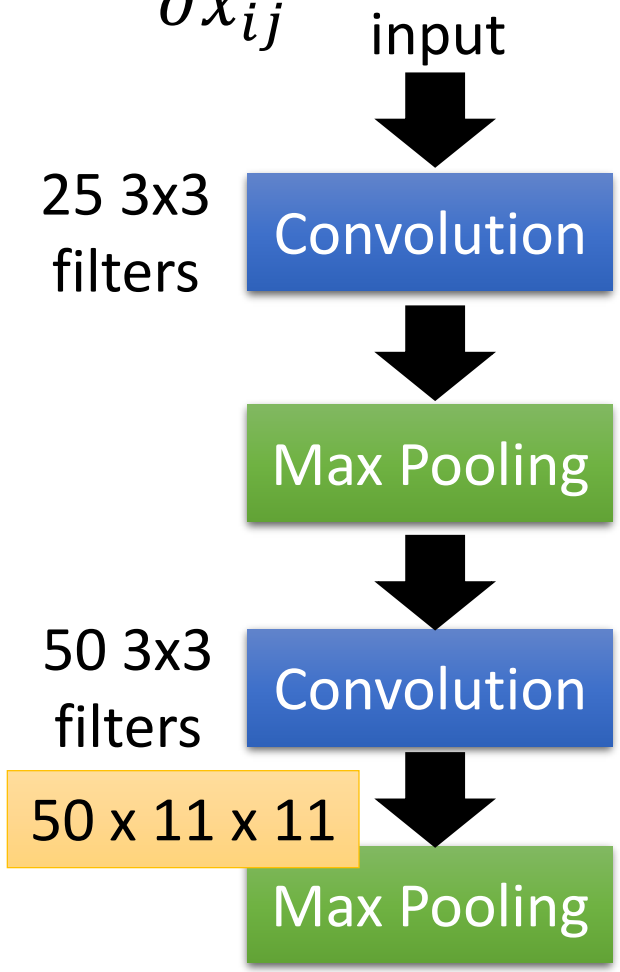
Degree of the activation of the k-th filter: $a^k = \sum_{i=1}^{11} \sum_{j=1}^{11} a_{ij}^k$

$x^* = arg \max_x a^k$ (gradient ascent)

$$\frac{\partial a^k}{\partial x_{ij}}$$



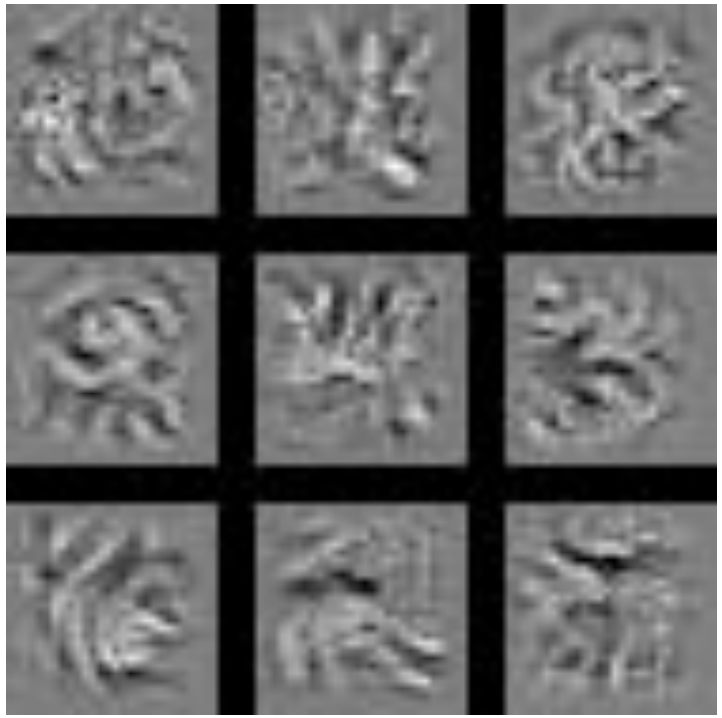
For each filter



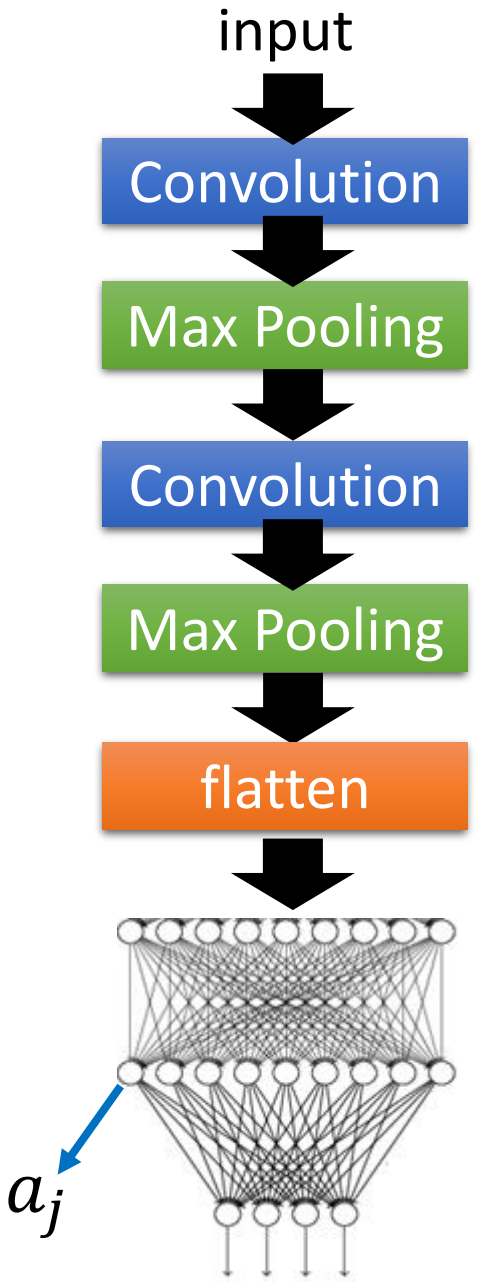
What does CNN learn?

Find an image maximizing the output of neuron:

$$x^* = arg\max_x a^j$$



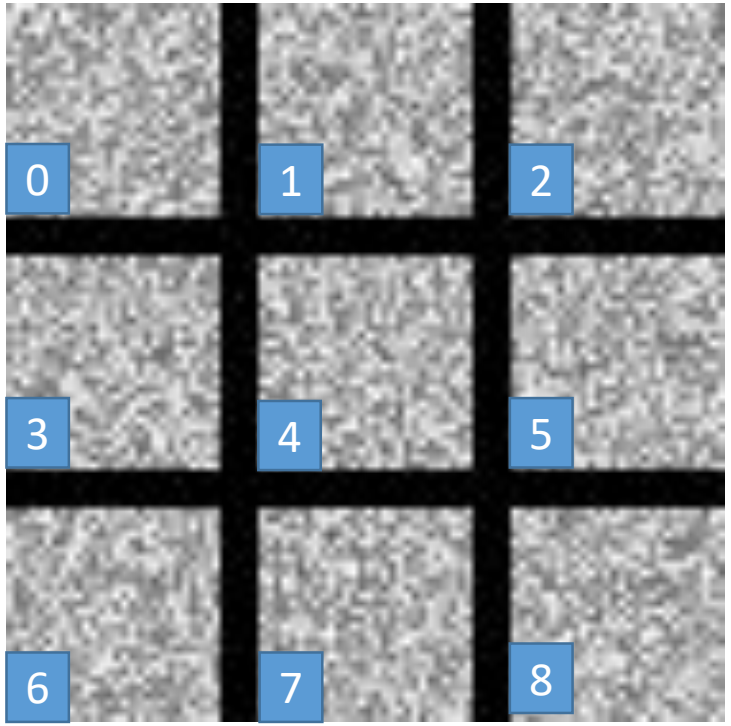
Each figure corresponds to a neuron



What does CNN learn?

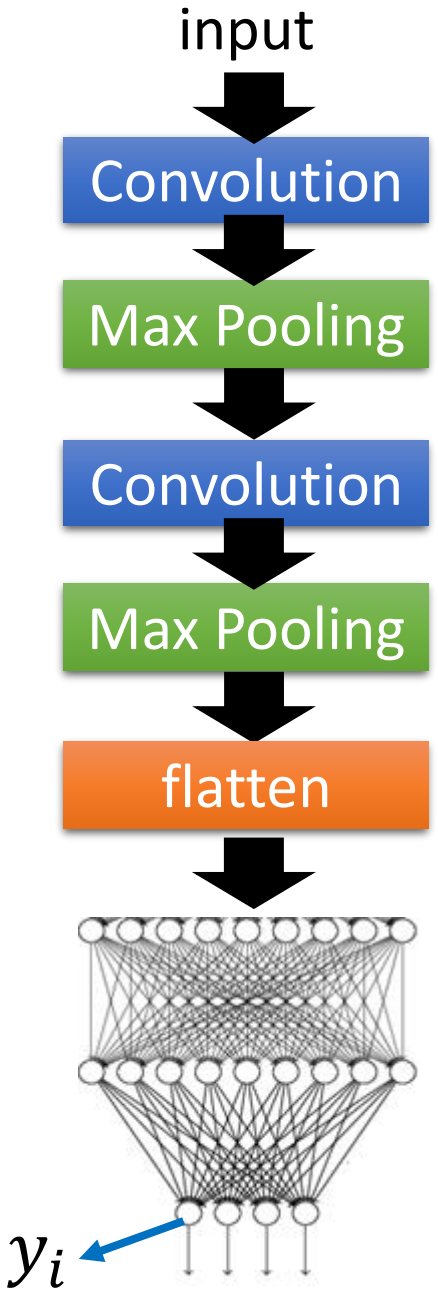
$$x^* = arg\max_x y^i$$

Can we see digits?



Deep Neural Networks are Easily Fooled

https://www.youtube.com/watch?v=M2IebCN9Ht4

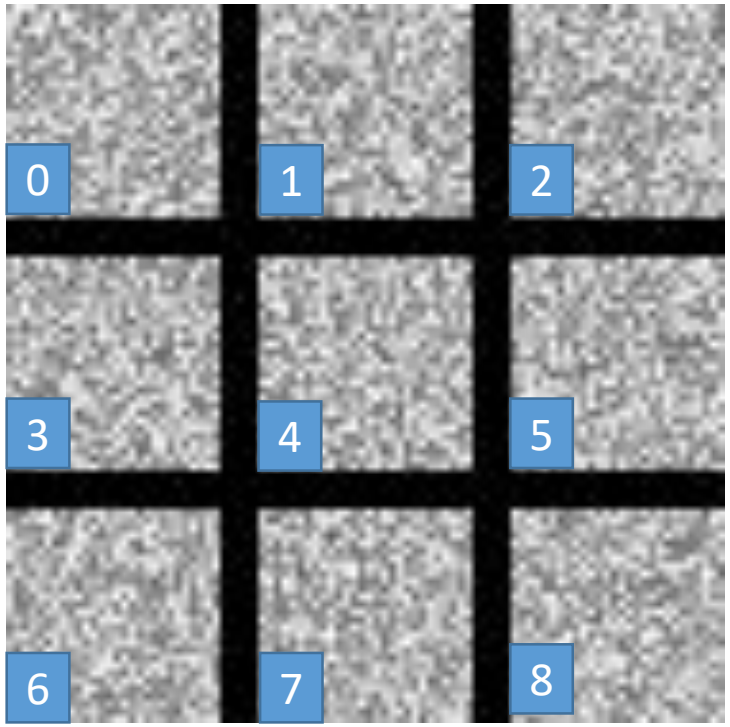


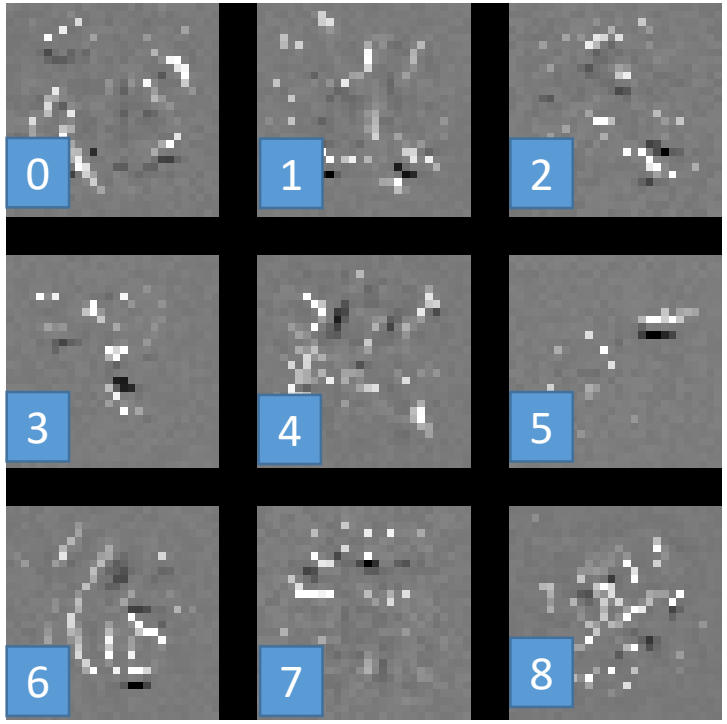
What does CNN learn?

$$x^* = arg\max_x y^i$$

Over all pixel values

$$x^* = arg\max_x \left(y^i - \sum_{i,j} |x_{ij}| \right)$$





Karen Simonyan, Andrea Vedaldi, Andrew Zisserman, “Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps”, ICLR, 2014

$$\left|\frac{\partial y_k}{\partial x_{ij}}\right|$$

y_k: the predicted class of the model

Karen Simonyan, Andrea Vedaldi, Andrew Zisserman, “Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps”, ICLR, 2014

Review……

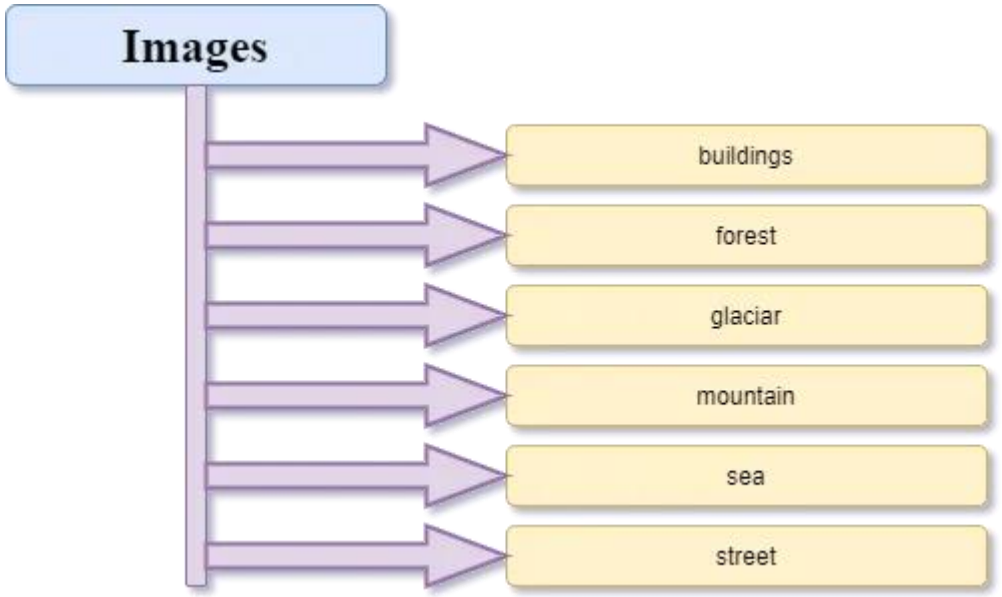
- Examples

Example 1:

CNN Model With PyTorch For Image Classification

- Preparing The Dataset
- Splitting Data and Prepare Batches
- Base Model For Image Classification
- Convolution, Padding, Stride, Pooling
- CNN Model For Classification
- Hyperparameters, Model Training, And Evaluation

Preparing the Dataset :




```
import torch
import torchvision
from torchvision import transforms
from torchvision.datasets import ImageFolder
```

```
#train and test data directory
data_dir = "../input/intel-image-
classification/seg_train/seg_train/"
test_data_dir = "../input/intel-image-
classification/seg_test/seg_test"
```

```
#load the train and test data
dataset = ImageFolder(data_dir,transform =
transforms.Compose([
    transforms.Resize((150,150)),transforms.ToTensor()
]))
test_dataset =
ImageFolder(test_data_dir,transforms.Compose([
    transforms.Resize((150,150)),transforms.ToTensor()
]))
```

CNN Model With PyTorch

The torchvision.transforms module provides various functionality to preprocess the images, here first we resize the image for (150*150) shape and then transforms them into tensors.

```
img, label = dataset[0]
print(img.shape,label)

#output :
#torch.Size([3, 150, 150]) 0
```

first image in the dataset has a shape (3,150,150) which means the image has 3 channels (RGB), height 150, and width 150. The image has a label 0, which represents the “buildings” class.

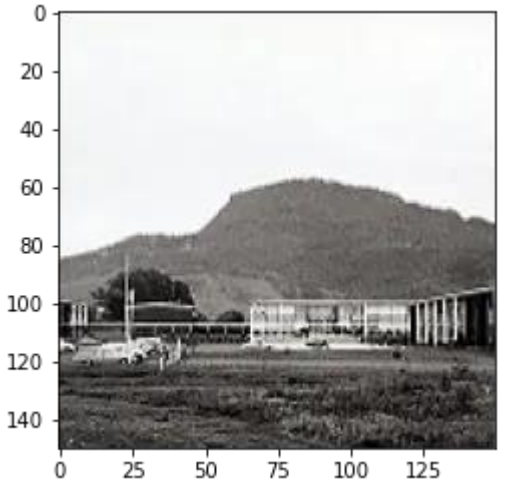
The image label set according to the class index in data.classes.

```
print("Follwing classes are there : \n",dataset.classes)
#output:
#Follwing classes are there :
# ['buildings', 'forest', 'glacier', 'mountain', 'sea', 'street']
```

dataset has 6 types of images in the dataset. dataset consists of images in form of *Tensors,* imshow*()* method of matplotlib python library can be used to visualize images.

```
def display_img(img,label):
    print(f"Label : {dataset.classes[label]}")
    plt.imshow(img.permute(1,2,0))

#display the first image in the dataset
display_img(*dataset[0])
```

permute method reshapes the image from (3,150,150) to (150,150,3)



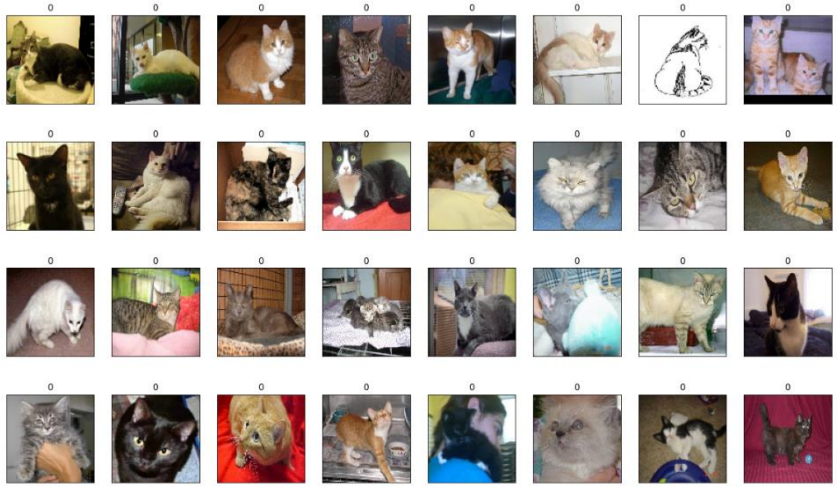
CNN Model With PyTorch

Splitting Data and Prepare Batches

```
from torch.utils.data.dataloader import DataLoader
from torch.utils.data import random_split

batch_size = 128
val_size = 2000
train_size = len(dataset) - val_size

train_data,val_data =
random_split(dataset,[train_size,val_size])
print(f"Length of Train Data : {len(train_data)}")
print(f"Length of Validation Data : {len(val_data)}")

#output
#Length of Train Data : 12034
#Length of Validation Data : 2000

#load the train and validation into batches.
train_dl = DataLoader(train_data, batch_size, shuffle = True,
num_workers = 4, pin_memory = True)
val_dl = DataLoader(val_data, batch_size*2, num_workers = 4,
pin_memory = True)
```

CNN Model With PyTorch

Visualizing the images



```
from torchvision.utils import make_grid
import matplotlib.pyplot as plt

def show_batch(dl):
    """Plot images grid of single batch"""
    for images, labels in dl:
        fig,ax = plt.subplots(figsize = (16,12))
        ax.set_xticks([])
        ax.set_yticks([])
        ax.imshow(make_grid(images,nrow=16).permute(1,2,0))
        break

show_batch(train_dl)
```

https://python.plainenglish.io/cat-dog-classification-with-cnn-84af3ae98c44

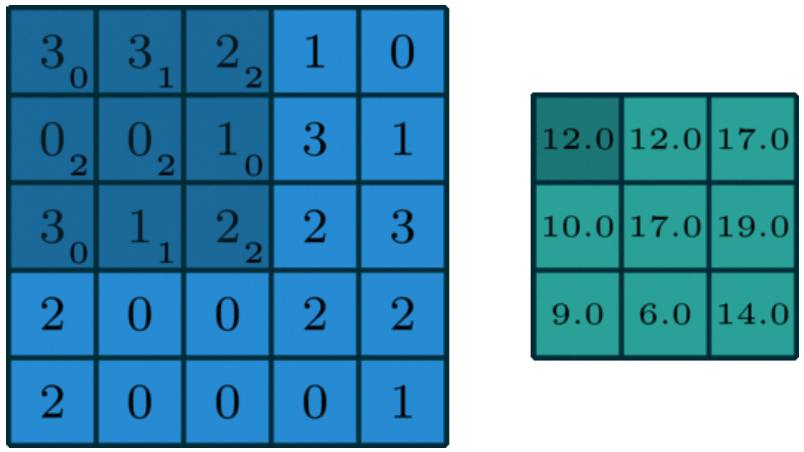
CNN Model With PyTorch

Base Model For Image Classification

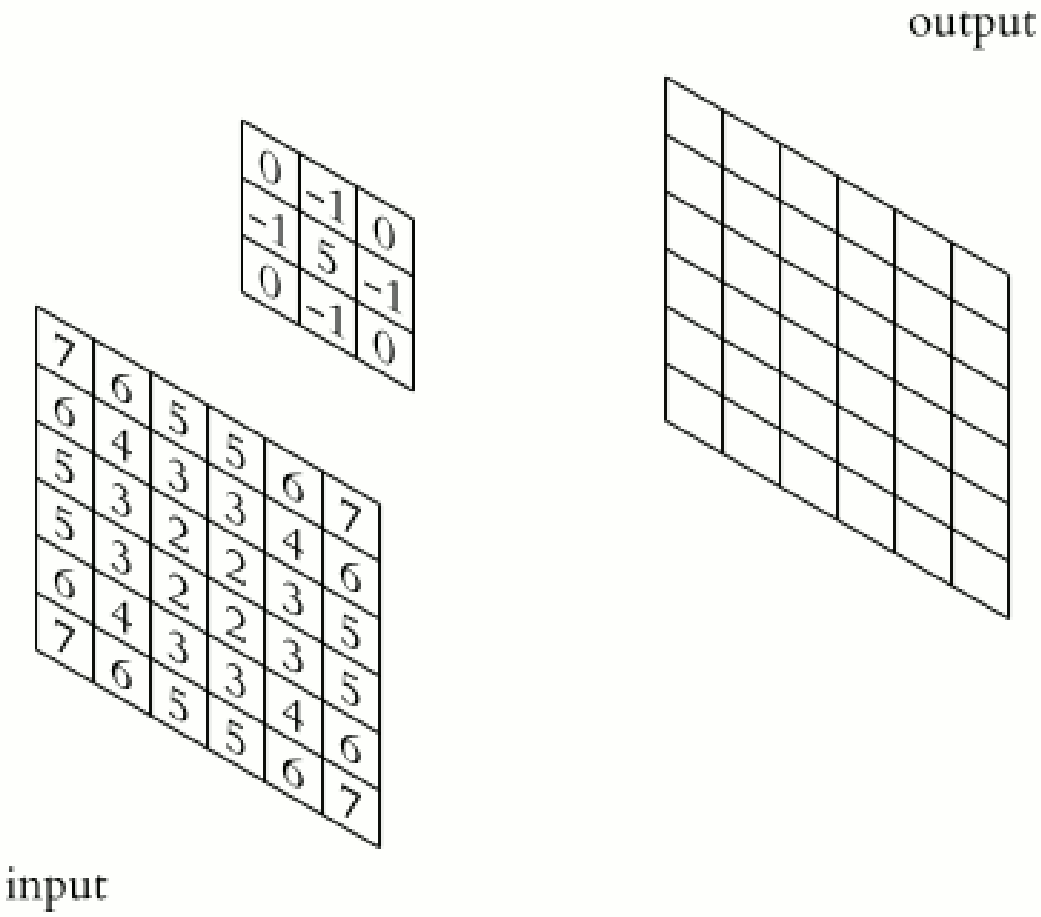
```
import torch.nn as nn
import torch.nn.functional as F

class ImageClassificationBase(nn.Module):

    def training_step(self, batch):
        images, labels = batch
        out = self(images)                  # Generate predictions
        loss = F.cross_entropy(out, labels) # Calculate loss
        return loss

    def validation_step(self, batch):
        images, labels = batch
        out = self(images)                    # Generate predictions
        loss = F.cross_entropy(out, labels)   # Calculate loss
        acc = accuracy(out, labels)           # Calculate accuracy
        return {'val_loss': loss.detach(), 'val_acc': acc}

    def validation_epoch_end(self, outputs):
        batch_losses = [x['val_loss'] for x in outputs]
        epoch_loss = torch.stack(batch_losses).mean()   # Combine
losses
        batch_accs = [x['val_acc'] for x in outputs]
        epoch_acc = torch.stack(batch_accs).mean()      # Combine
accuracies
        return {'val_loss': epoch_loss.item(), 'val_acc':
epoch_acc.item()}

    def epoch_end(self, epoch, result):
        print("Epoch [{}], train_loss: {:.4f}, val_loss: {:.4f}, val_acc:
{:.4f}".format(
            epoch, result['train_loss'], result['val_loss'],
result['val_acc']))
```

CNN Model With PyTorch

Convolution, Padding, Stride, Maxpooling

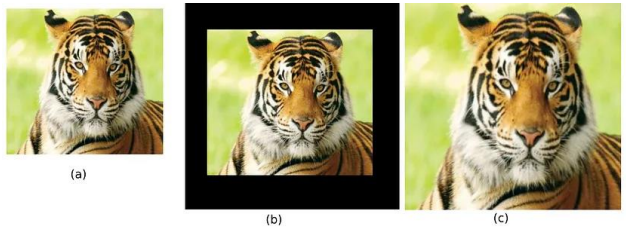


Convolution



CNN Model With PyTorch

Convolution, Padding, Stride, Maxpooling



A small image (a) resized to 180 × 180 pixels using zero-padding (b) and interpolation (c)

Zerro Padding

0	0	0	0	0	0	0
0	60	113	56	139	85	0
0	73	121	54	84	128	0
0	131	99	70	129	127	0
0	80	57	115	69	134	0
0	104	126	123	95	130	0
0	0	0	0	0	0	0

Kernel

0	-1	0
-1	5	-1
0	-1	0

114				

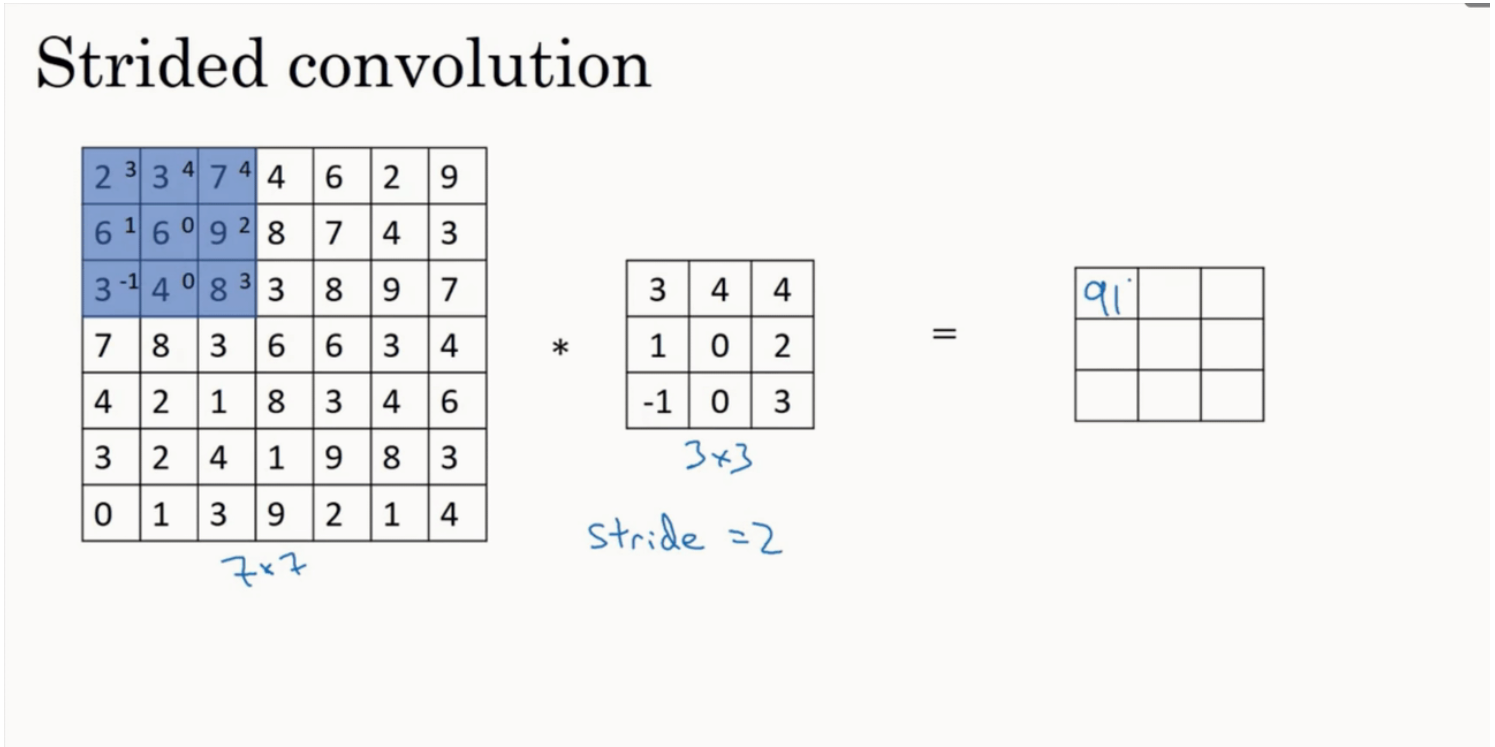
((W - F * 2P)/S) + 1 and ((H - F * 2P)/S) + 1

n = number of pixels of an input image
p= padding number
f = filter size
from these (n*n) + (p*p) convolution (f*f) gives → (n+2p-f+1) * (n+2p-f+1)

CNN Model With PyTorch

Convolution, Padding, Stride, Maxpooling

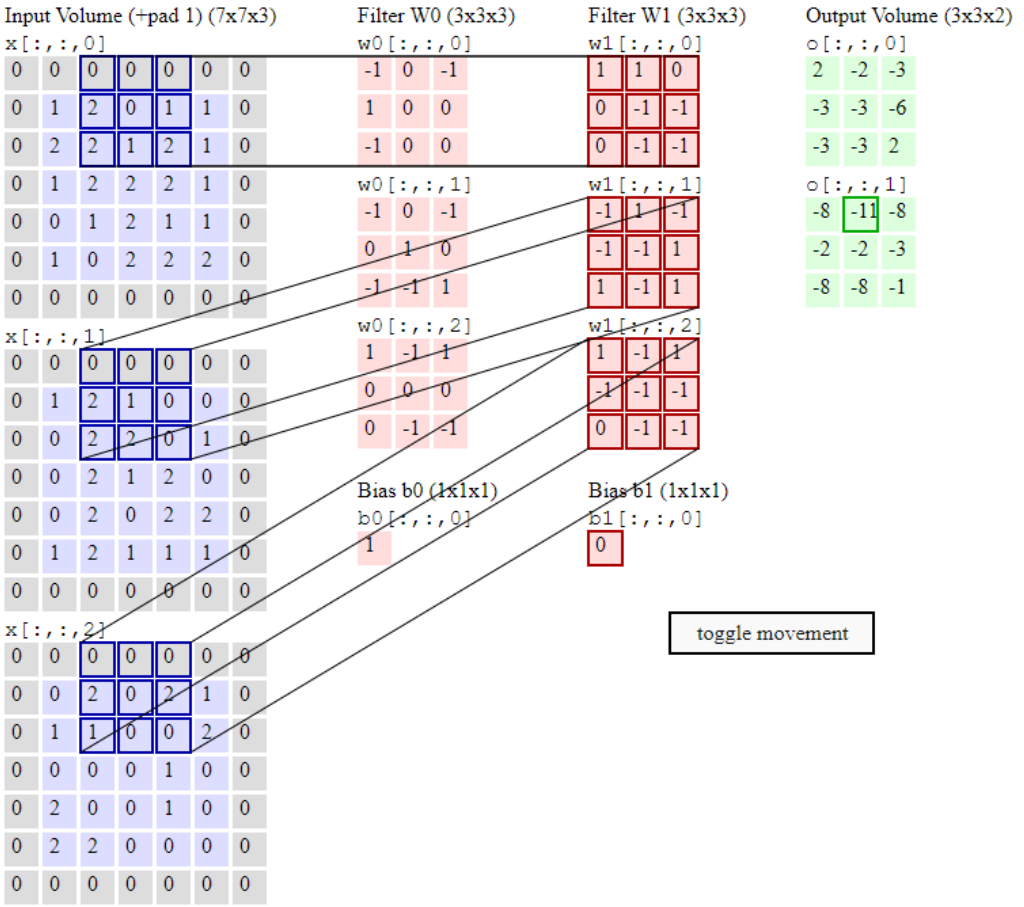
Stride



CNN Model With PyTorch

Convolution, Padding, Stride, Maxpooling

MaxPooling



CNN Model With PyTorch

```
Model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))

model.summary()
```

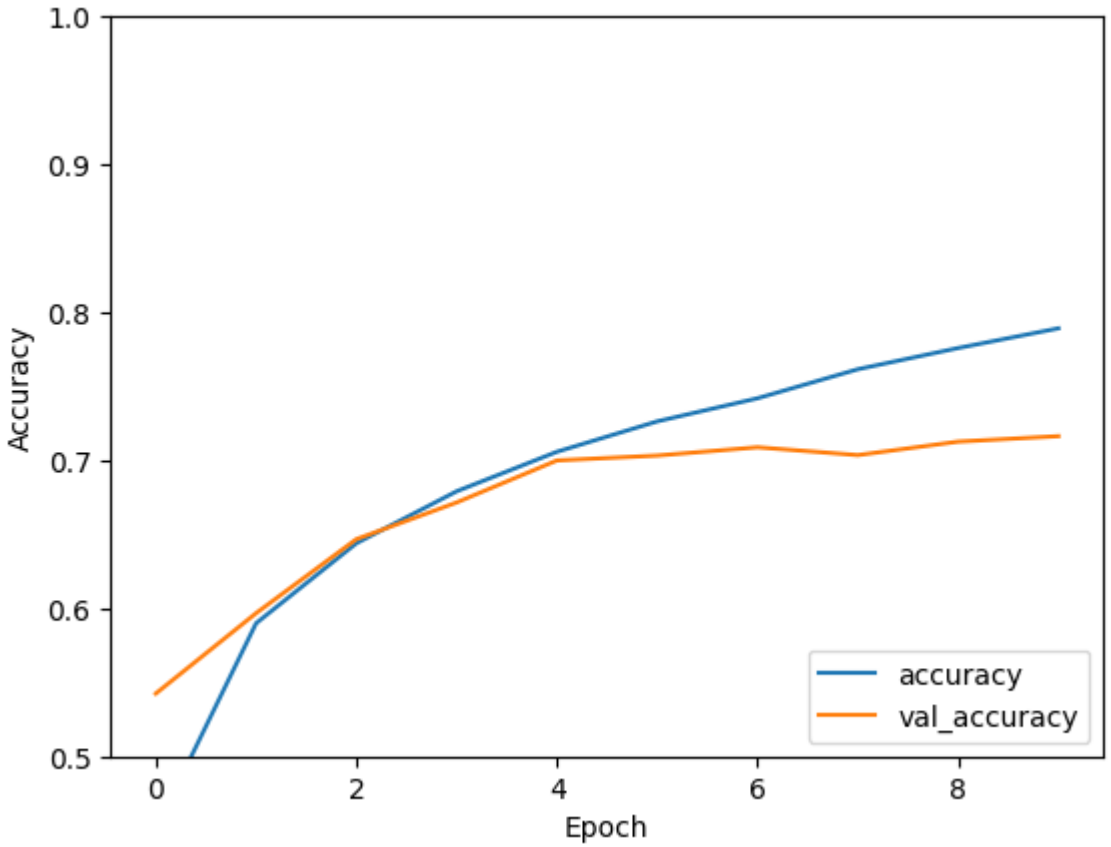
```
model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 32)        896

 max_pooling2d (MaxPooling2D  (None, 15, 15, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 13, 13, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 6, 6, 64)         0
 2D)

 conv2d_2 (Conv2D)           (None, 4, 4, 64)          36928

=================================================================
Total params: 56,320
Trainable params: 56,320
Non-trainable params: 0
```

```
model.compile(optimizer='adam',

loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits
=True),
              metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))
```

```
Epoch 1/10
1563/1563 [==============================] - 8s 4ms/step - loss: 1.4971 - accuracy: 0.4553 - val_loss: 1.2659 -
val_accuracy: 0.5492
Epoch 2/10
1563/1563 [==============================] - 6s 4ms/step - loss: 1.1424 - accuracy: 0.5966 - val_loss: 1.1025 -
val_accuracy: 0.6098
Epoch 3/10
1563/1563 [==============================] - 6s 4ms/step - loss: 0.9885 - accuracy: 0.6539 - val_loss: 0.9557 -
val_accuracy: 0.6629
Epoch 4/10
1563/1563 [==============================] - 6s 4ms/step - loss: 0.8932 - accuracy: 0.6878 - val_loss: 0.8924 -
val_accuracy: 0.6935
Epoch 5/10
1563/1563 [==============================] - 6s 4ms/step - loss: 0.8232 - accuracy: 0.7130 - val_loss: 0.8679 -
```

Evaluasi model

```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')

test_loss, test_acc = model.evaluate(test_images,  test_labels,
verbose=2)
```



Example 2:

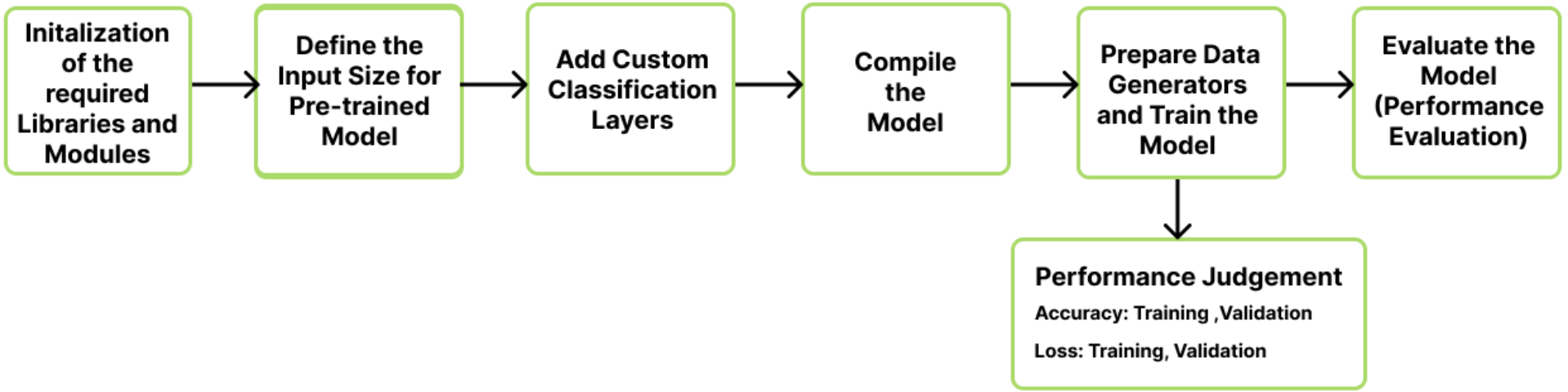
Flower Classification with CNN

https://github.com/siagianp/KerasTF-Flower-Classification-Convolutional-Neural-Network

CNN Model With KTF

1. Importing Libraries
2. Data Collection and Preparation
3. Utilize Pre-trained Architectures (Xception)
4. Feature Extraction and Model Architecture
5. Data Augmentation and Preprocessing
6. Model Training
7. Model Evaluation
8. User Interaction and Engagement
9. Deployment and Integration

CNN Model With KTF



CNN Model With KTF

flower_images (5 directories)

About this directory

The directory consists of five flower sub-directories; **Lilly, Lotus, Orchid, Sunflower and Tulip**. Each sub-directory has **1000 images** of the respective flowers.

Lilly	Lotus	Orchid	Sunflower	Tulip
1000 files	1000 files	1000 files	1000 files	1000 files

Data Explorer

261.51 MB

- flower_images
  - Lilly
  - Lotus
  - Orchid
  - Sunflower
  - Tulip

Summary

- 5000 files

CNN Model With KTF

```
# Setting the data
"""

import os
import numpy as np
import matplotlib.pyplot as plt
import cv2
import pickle

data_dir = "../data/train/"

categories = ["daisy", "dandelion", "rose", "sunflower", "tulip"]

data = []

def make_data():
    for category in categories:
        path = os.path.join(data_dir, category)
        label = categories.index(category)

        for img_name in os.listdir(path):
```

Example 3:

Pytorch : Cat Dog Classification with CNN

checkpoint	commit2
sample/sample	Add files via upload
.gitattributes	Initial commit
README.md	Update README.md
preprocessing.py	commit2
testSample.py	Add files via upload
train.py	commit2

Checkpoint: epoch60_93.pt
sample/sample : flower1.jpg

CNN Model With PyTorch

Preprocessing.py

```
import torchvision.datasets as dset
import torchvision.transforms as transforms
import torch

batch_size = 64

def trainDataset():
    dataset = dset.ImageFolder(root="dataset/train",
                               transform=transforms.Compose([
                                   transforms.RandomRotation(30),
                                   transforms.RandomHorizontalFlip(),
                                   transforms.RandomResizedCrop(224,
scale=(0.96, 1.0), ratio=(0.95, 1.05)),
                                   transforms.ToTensor(),
                                   transforms.Normalize([0.485, 0.456, 0.406],
[0.229, 0.224, 0.225])
                               ]))
    dataloader = torch.utils.data.DataLoader(dataset
```

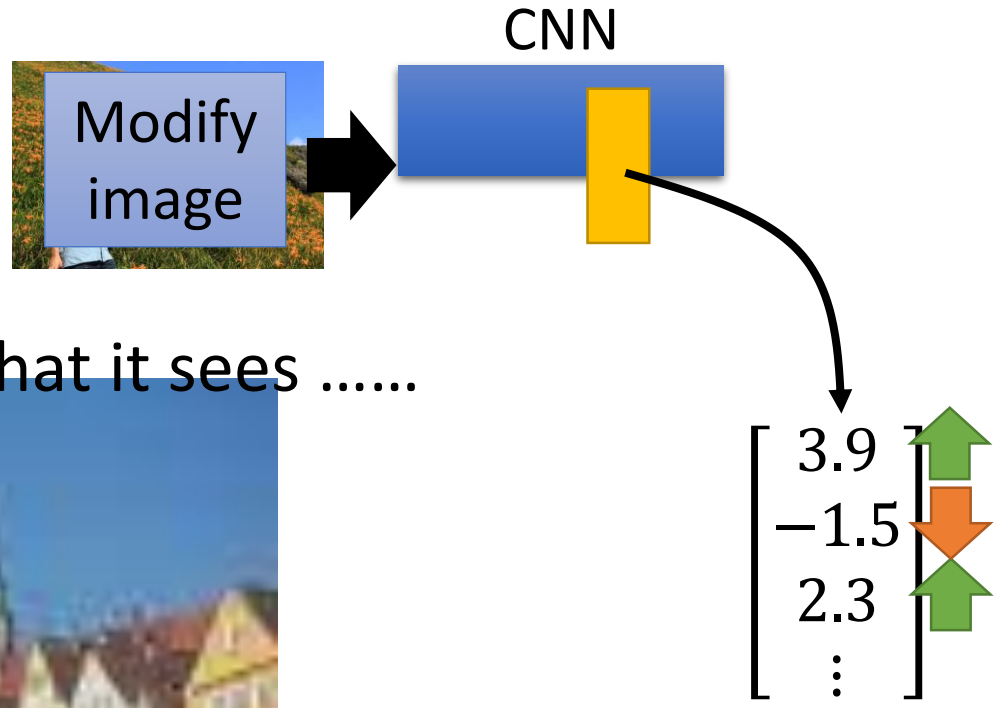
##Test Sample.py

```
import torchvision.datasets as dset
import torchvision.transforms as transforms
import torch
import numpy
from torchvision import models

dataset = dset.ImageFolder(root="sample",
                           transform=transforms.Compose([
                               transforms.Resize([224, 224]),
                               transforms.ToTensor(),
                               transforms.Normalize([0.485, 0.456, 0.406],
[0.229, 0.224, 0.225]),
                           ]))
dataloader = torch.utils.data.DataLoader(dataset,
                                         batch_size=1,
                                         shuffle=False,
                                         num_workers=0)
```

##train dir

```
from prepr
import tim

import tor
import tor
import tor
from torch
import tor
from torch

lr = 0.003
device = to

train_load
test_loade
```

Deep Dream



- Given a photo, machine adds what it sees ……

https://deepdreamgenerator.com/generate

http://deepdreamgenerator.com/

Deep Dream

- Given a photo, machine adds what it sees ……

http://deepdreamgenerator.com/

Deep Style

- Given a photo, make its style like famous paintings

https://dreamscopeapp.com/

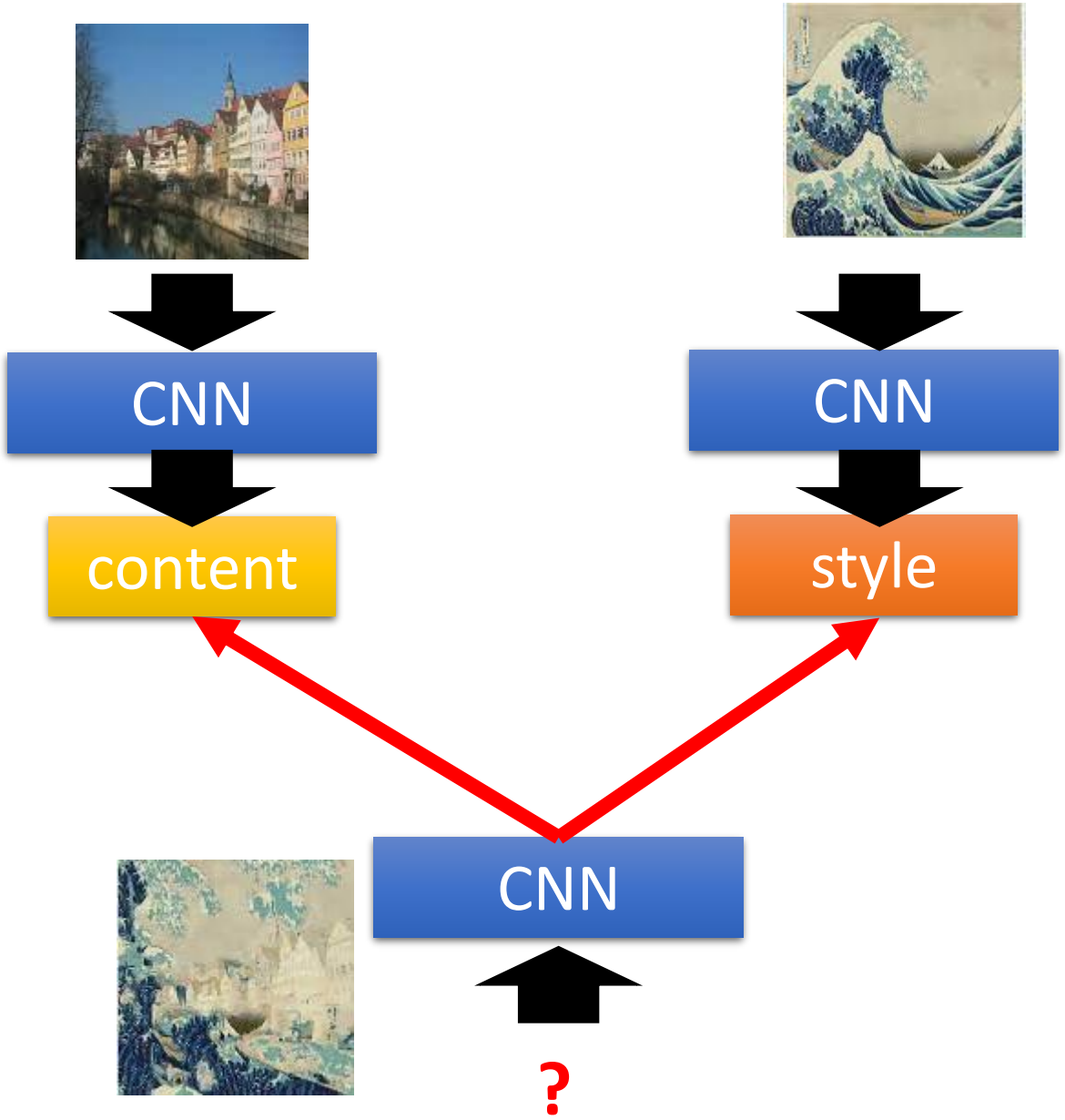
Deep Style

- Given a photo, make its style like famous paintings

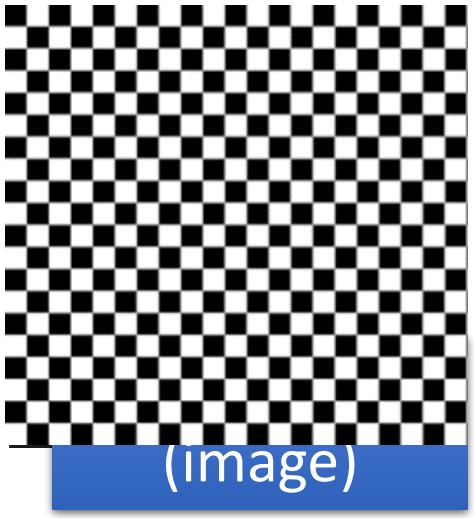
https://dreamscopeapp.com/

Deep Style

A Neural Algorithm of Artistic Style



More Application: Chess



Network

Next move (19 x 19 positions)

19 x 19 vector

Black: 1

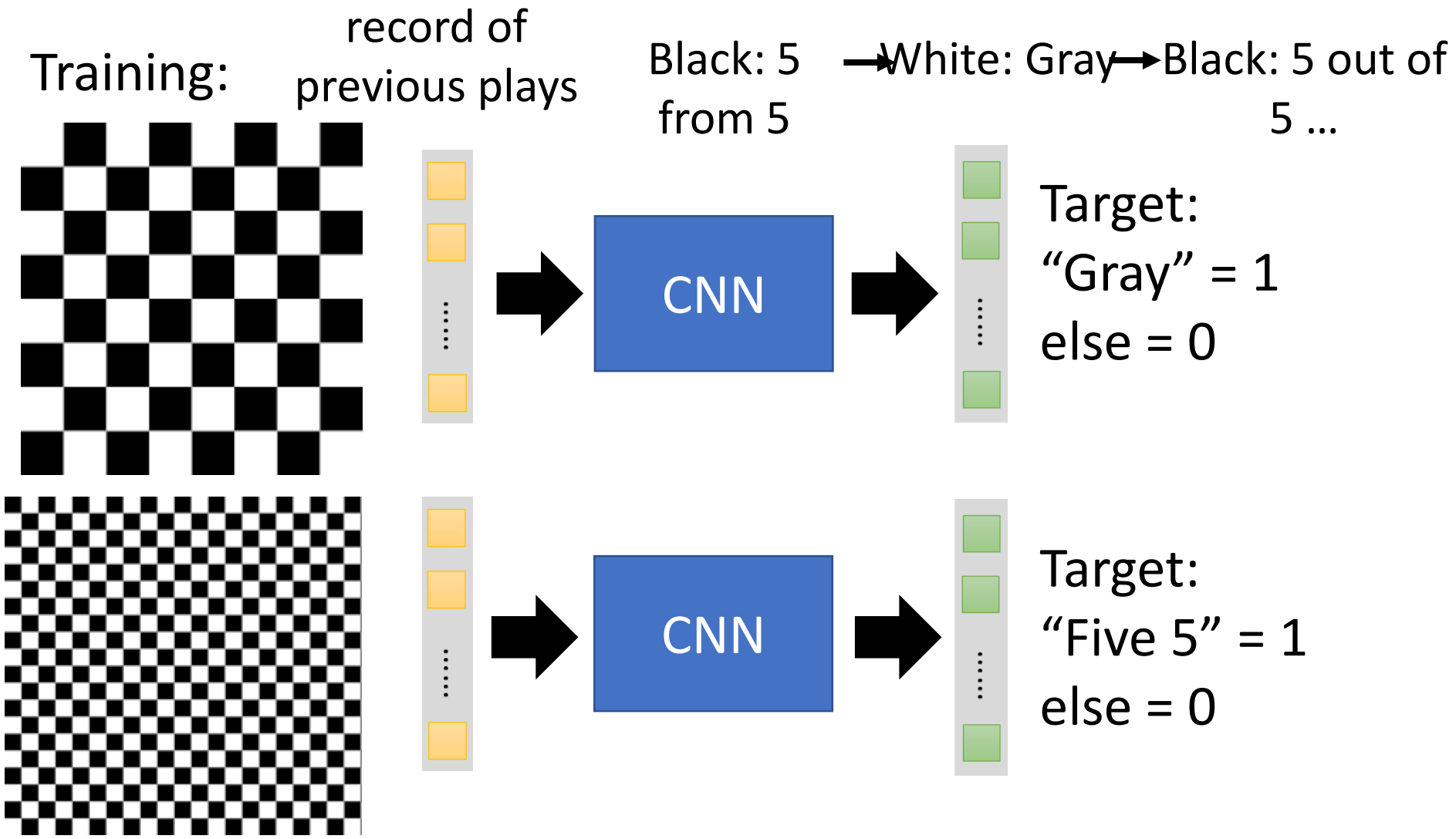
white: -1

none: 0

Fully-connected feedforward network can be used

But CNN performs much better.

More Application: Catur



Why CNN for playing Chess Go?

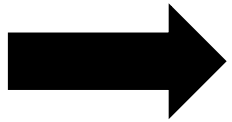
- Some patterns are much smaller than the whole image

Alpha Go uses 5 x 5 for first layer

- The same patterns appear in different regions.

Why CNN for playing Chess Go?

- Subsampling the pixels will not change the object



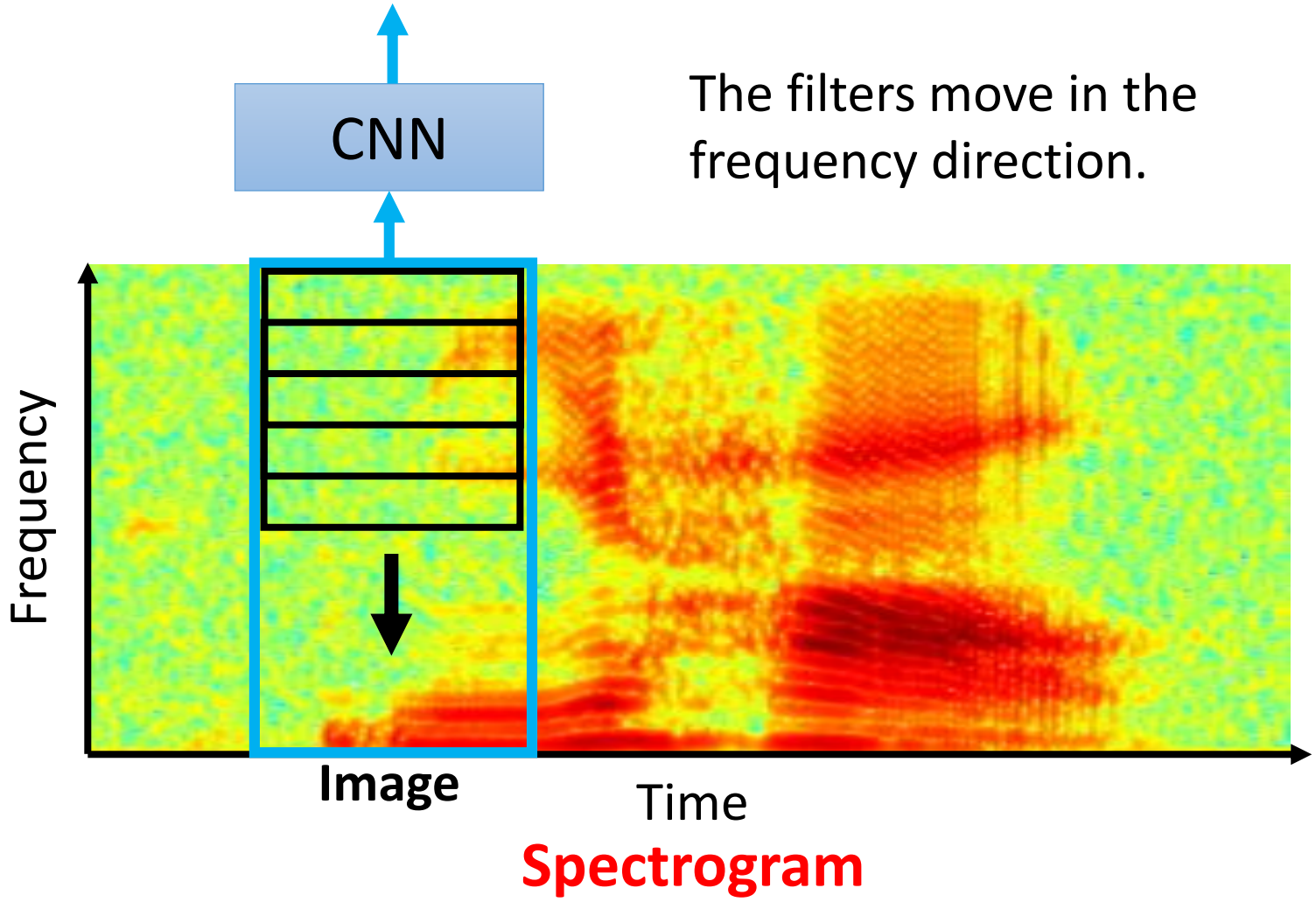
Max Pooling

How to explain this???

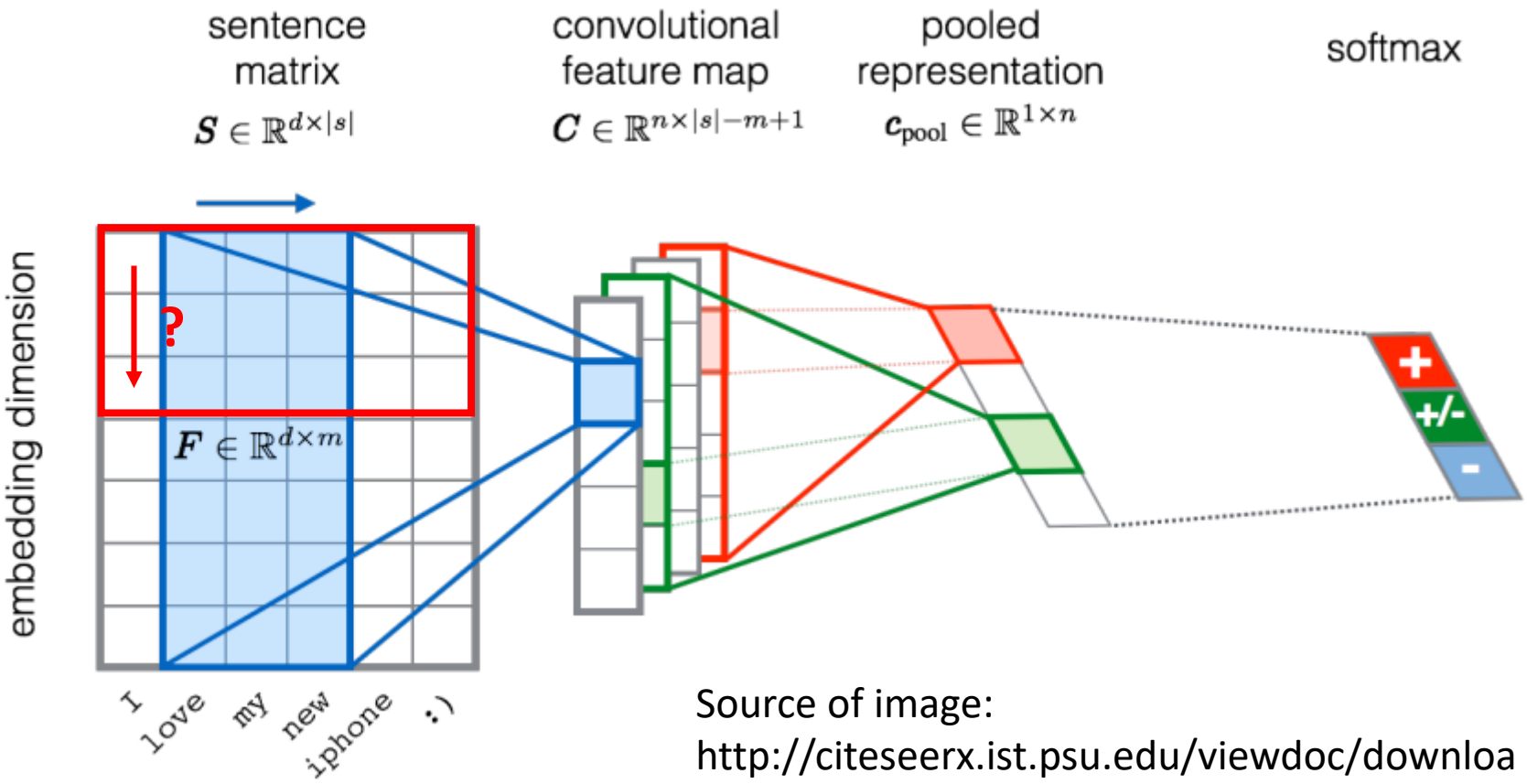
Neural network architecture. The input to the policy network is a $19 \times 19 \times 48$ image stack consisting of 48 feature planes. The first hidden layer zero pads the input into a 23×23 image, then convolves k filters of kernel size 5×5 with stride 1 with the input image and applies a rectifier nonlinearity. Each of the subsequent hidden layers 2 to 12 zero pads the respective previous hidden layer into a 21×21 image, then convolves k filters of kernel size 3×3 with stride 1, again followed by a rectifier nonlinearity. The final layer convolves 1 filter of kernel size 1×1 with stride 1, with a different bias for each position, and applies a softmax function. The Extended Data Table 3 additionally show the results of training with $k = 128$, 256 and 384 filters.

Alpha Go does not use Max Pooling ……

More Application: Speech

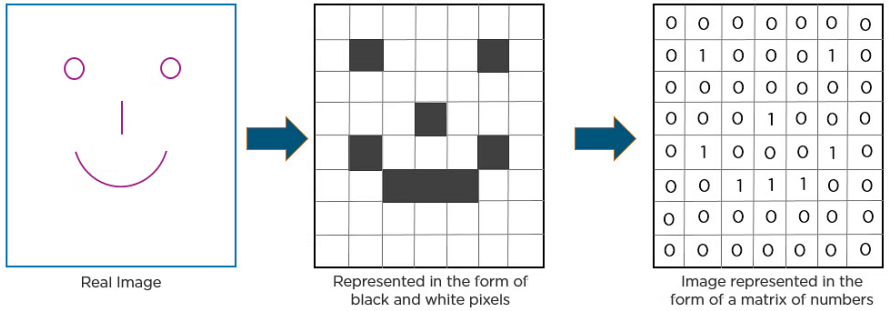


More Application: Text



Source of image: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.703.6858&rep=rep1&type=pdf

CNN

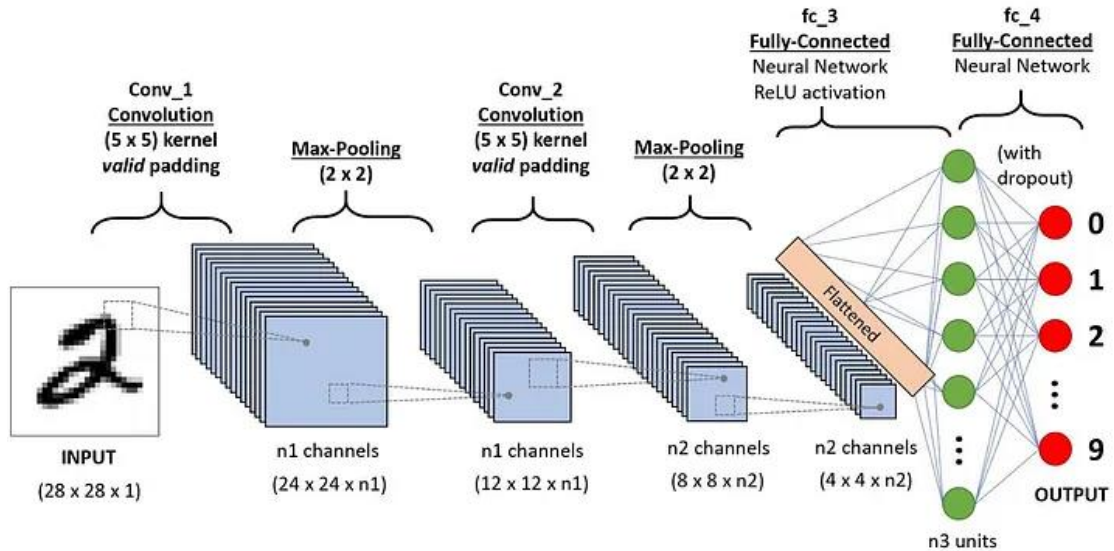


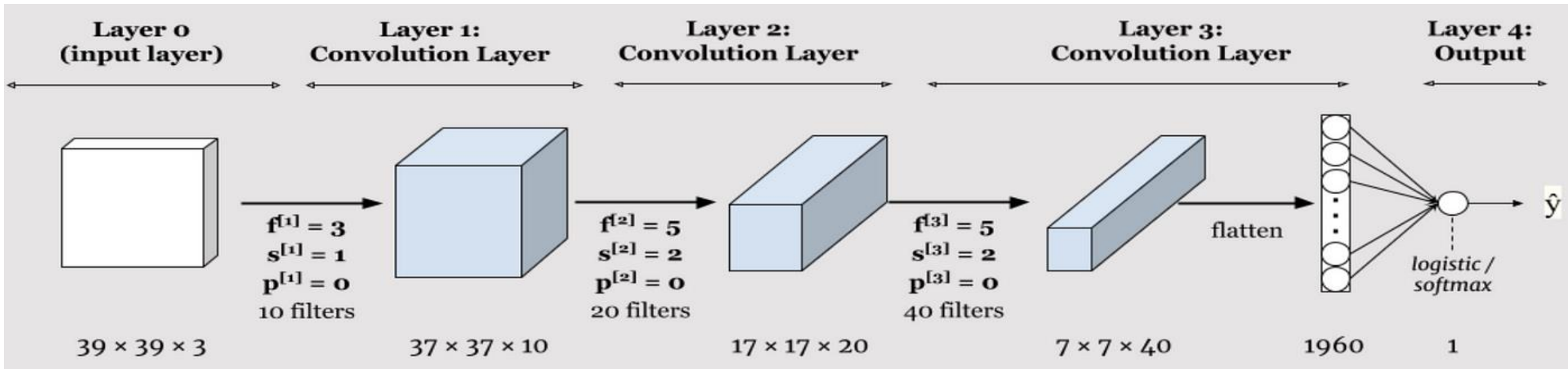
- Understanding Convolutions

http://colah.github.io/posts/2014-07-Understanding-Convolutions/
https://python.plainenglish.io/cat-dog-classification-with-cnn-84af3ae98c44

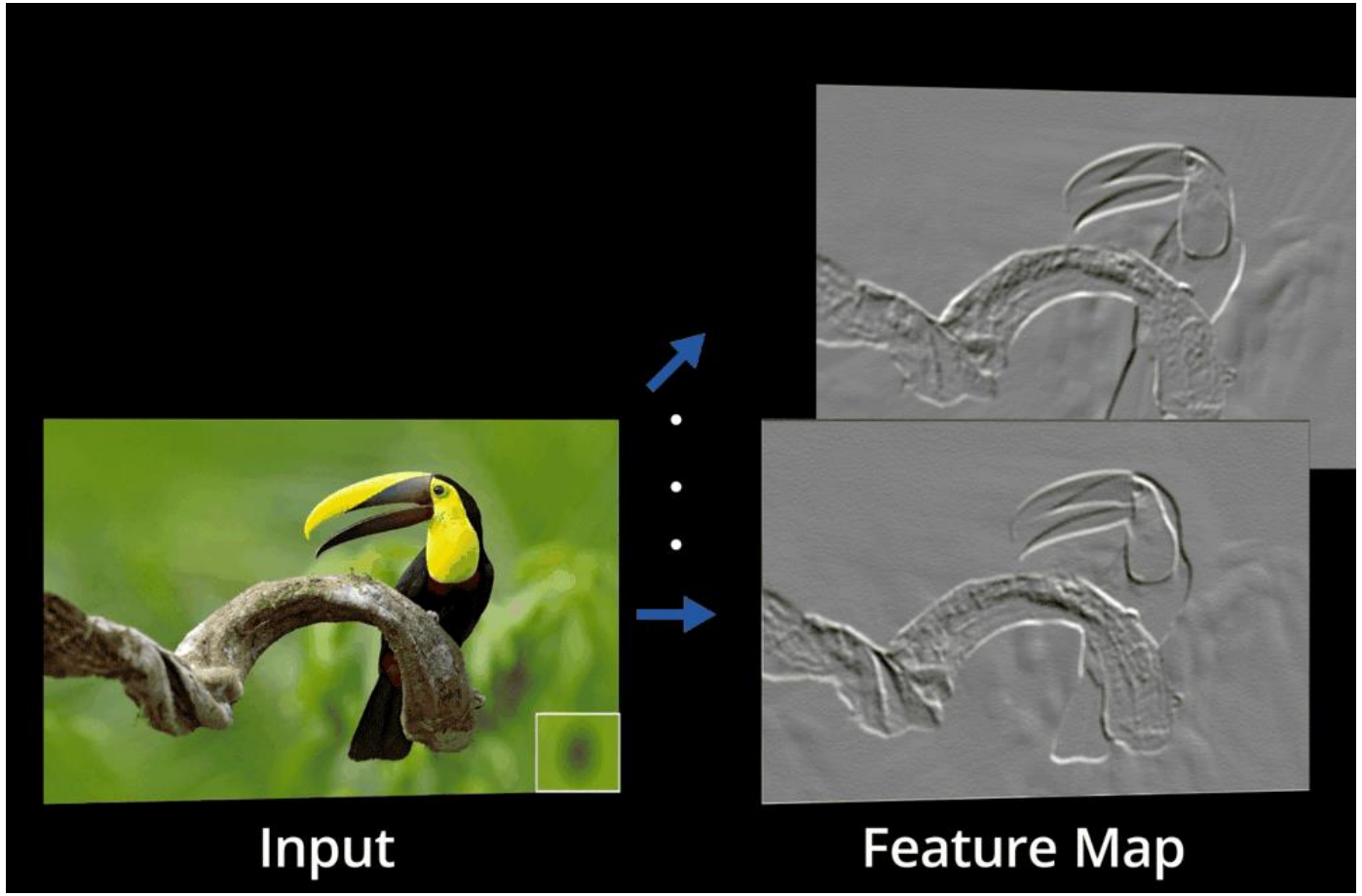
https://betterexplained.com/articles/intuitive-convolution/

https://www.tomasbeuzen.com/deep-learning-with-pytorch/chapters/chapter5_cnns-pt1.html
https://poloclub.github.io/cnn-explainer/
https://medium.com/thedeephub/convolutional-neural-networks-a-comprehensive-guide-5cc0b5eae175


Conv_1
Convolution
(5 x 5) kernel
valid padding
Max-Pooling
(2 x 2)
Conv_2
Convolution
(5 x 5) kernel
valid padding
Max-Pooling
(2 x 2)
fc_3
Fully-Connected
Neural Network
ReLU activation
fc_4
Fully-Connected
Neural Network
(with
dropout)
Flattened
INPUT
(28 x 28 x 1)
n1 channels
(24 x 24 x n1)
n1 channels
(12 x 12 x n1)
n2 channels
(8 x 8 x n2)
n2 channels
(4 x 4 x n2)
n3 units
0
1
2
9
OUTPUT


Layer 0
(input layer)
Layer 1:
Convolution Layer
Layer 2:
Convolution Layer
Layer 3:
Convolution Layer
Layer 4:
Output
f[1] = 3
s[1] = 1
p[1] = 0
10 filters
f[2] = 5
s[2] = 2
p[2] = 0
20 filters
f[3] = 5
s[3] = 2
p[3] = 0
40 filters
flatten
logistic /
softmax
ŷ
39 × 39 × 3
37 × 37 × 10
17 × 17 × 20
7 × 7 × 40
1960
1

Feature ?



Next…….

The First Step:

- **Understanding “convolution” operations in CNN**
- **Understanding “Image RGB**
- **Understanding “Image Greyscale**
- **Mathematical understanding**

I…….

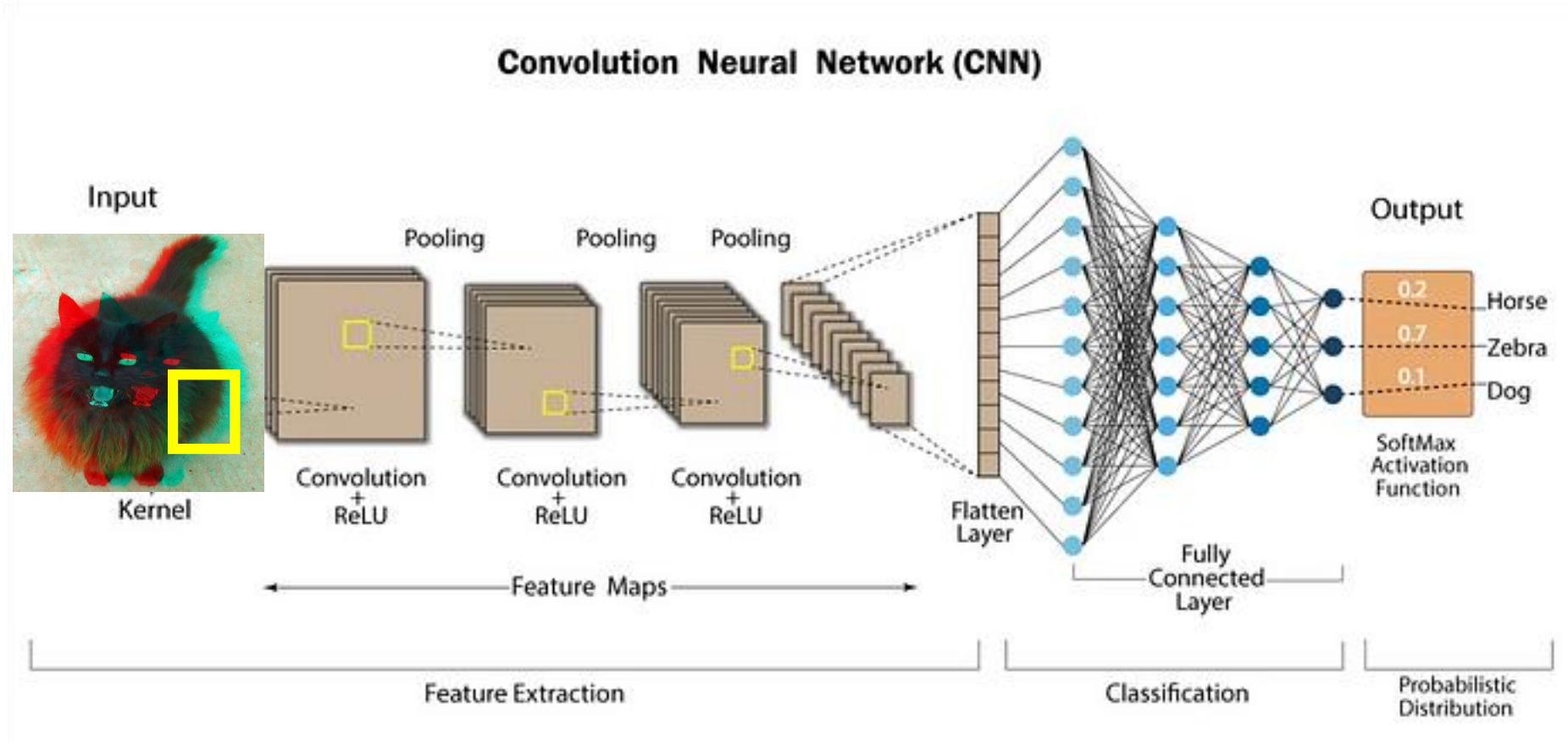
The Second Step:

- What is Convolutional Neural Network?
- Convolutional Layer
- Padding and Stride
- Pooling
- ReLU
- Basic Python Implementation

II…….

The First Step:

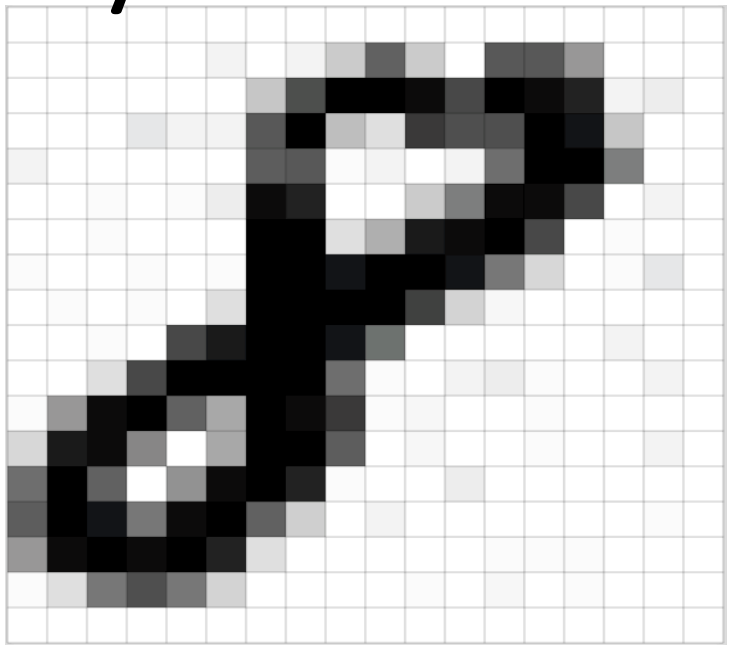
Understanding “convolution” operations in CNN



Understanding “Image RGB

24x24x3

Understanding “Image Greyscale



24x24x1

Ix ➔0 - 255

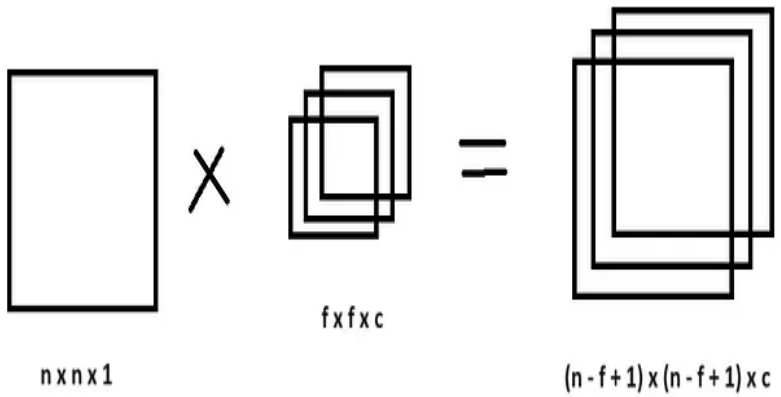
Understanding “Image Binary

24x24x1

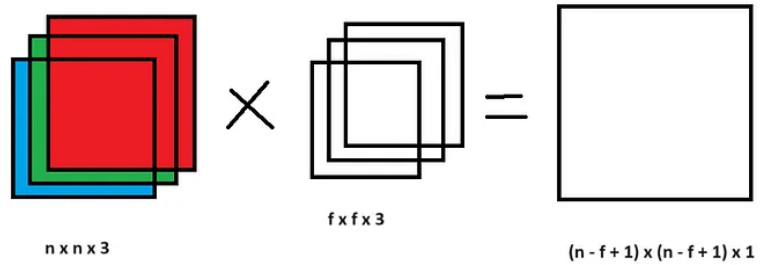
Ix ➔0 - 1

Mathematical understanding

Grayscale image, lets say the image size is 24x24x1 (24 is the width, 24 is the height, 1 is the number of color channels, since its a grayscale image, it will have 1 channel; if it would have been a coloured image, number of channels would have been 3) and the kernel size is 3x3x64 (3 is the width, 3 is the height, 64 is the number of such kernels); then after the convolution operation the size of the output image will be 22x22x64 ((24–3+1)x(24–3+1)x64)



The image is a coloured one, lets consider the size of the image as 24 x 24 x 3, (for performing convolution operation on a coloured image, the kernel should also have 3 channels), kernel size as 4 x 4 x 3 (where 3 is the number of channels). The number of channels of the image and the kernel should match incase of coloured image for the convolution operation to happen. The convolution operation will happen by super imposing the kernel on the image. The size of the image after the convolution operation will be (21 x 21 x 1). Here 1 denotes the number of colour channels, which means after the convolution operation, grayscale image is obtained. This is a common scenario in convolutional neural networks, where intermediate layers may reduce the number of channels while preserving spatial information. **If there would have been multiple kernels lets say 64, then the size of the image would have been 21 x 21 x 1 x 64.**



Feature Map = Input Image x Feature Detector

- Size of Image: N*N
- Size of Filter : F*F
- (N*N)(F*F)= (N-F+1)*(N-F+1)= Convolution Operation
- Let us consider dimensions of feature map be H*W*C
- H — Height of feature map
- W — Width of feature map
- C — Number of channels in feature map
- F — Filter / Kernel
- S — Stride
- The output image will be {(H-F)/S+1} * {(W-F)/S+1} * C

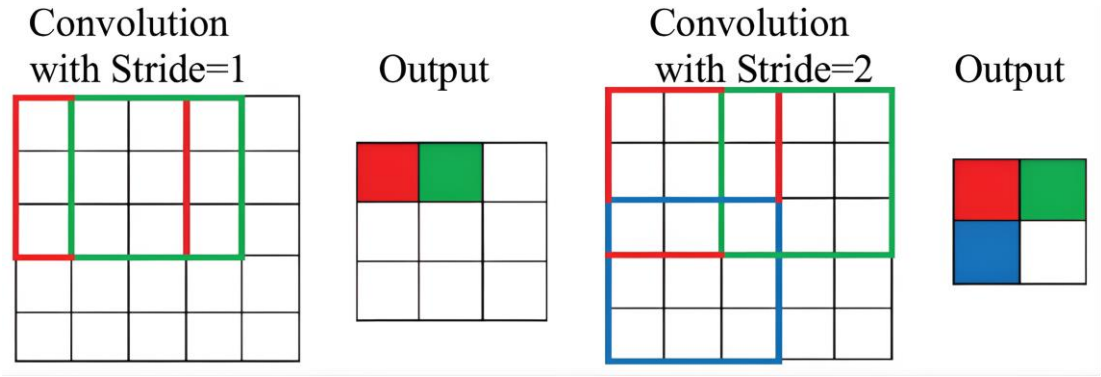
Convolution process

$$I_{y,x} \star h = \sum_{i=-n_2}^{n_2} \sum_{j=-m_2}^{m_2} I(y-i,\, x-j) \cdot h(i,\, j)$$

n2 is half the height of the filter
m2 is half the length of the filter
x is the column position of a certain pixel in the image
y is the row position of a certain pixel in the image.
n2 = [n/2], m2 = [m/2] above, *n* is the height of the filter, and *m* is the length of the filter *h*.

$$I = \begin{bmatrix} I_{00} & I_{01} & I_{02} & I_{03} \\ I_{10} & I_{11} & I_{12} & I_{13} \\ I_{20} & I_{21} & I_{22} & I_{23} \\ I_{30} & I_{31} & I_{32} & I_{33} \end{bmatrix} = \begin{bmatrix} 0 & 3 & 6 & 9 \\ 1 & 4 & 7 & 10 \\ 2 & 5 & 8 & 11 \\ 3 & 6 & 9 & 12 \end{bmatrix} h = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Stride



Padding

0	0	0	0	0	0	0
0	60	113	56	139	85	0
0	73	121	54	84	128	0
0	131	99	70	129	127	0
0	80	57	115	69	134	0
0	104	126	123	95	130	0
0	0	0	0	0	0	0

Kernel

0	-1	0
-1	5	-1
0	-1	0

114				

i.e. ***padding****=1, and* ***stride****=1,*

$$n_{out} = \lfloor \frac{n_{in} + 2padding - n}{stride} \rfloor + 1$$

$$m_{out} = \lfloor \frac{m_{in} + 2padding - m}{stride} \rfloor + 1$$

Where n_in, m_in – height and length of the input image,
n_out, m_out – are the height and length of the output image.

$$I = \begin{bmatrix} I_{00} & I_{01} & I_{02} & I_{03} \\ I_{10} & I_{11} & I_{12} & I_{13} \\ I_{20} & I_{21} & I_{22} & I_{23} \\ I_{30} & I_{31} & I_{32} & I_{33} \end{bmatrix} \quad I = \begin{bmatrix} I_{00} & I_{01} & I_{02} & I_{03} \\ I_{10} & I_{11} & I_{12} & I_{13} \\ I_{20} & I_{21} & I_{22} & I_{23} \\ I_{30} & I_{31} & I_{32} & I_{33} \end{bmatrix}$$

$$I = \begin{bmatrix} I_{00} & I_{01} & I_{02} & I_{03} \\ I_{10} & I_{11} & I_{12} & I_{13} \\ I_{20} & I_{21} & I_{22} & I_{23} \\ I_{30} & I_{31} & I_{32} & I_{33} \end{bmatrix} \quad I = \begin{bmatrix} I_{00} & I_{01} & I_{02} & I_{03} \\ I_{10} & I_{11} & I_{12} & I_{13} \\ I_{20} & I_{21} & I_{22} & I_{23} \\ I_{30} & I_{31} & I_{32} & I_{33} \end{bmatrix}$$

$$I = \begin{bmatrix} I_{00} & I_{01} & I_{02} & I_{03} \\ I_{10} & I_{11} & I_{12} & I_{13} \\ I_{20} & I_{21} & I_{22} & I_{23} \\ I_{30} & I_{31} & I_{32} & I_{33} \end{bmatrix} = \begin{bmatrix} 0 & 3 & 6 & 9 \\ 1 & 4 & 7 & 10 \\ 2 & 5 & 8 & 11 \\ 3 & 6 & 9 & 12 \end{bmatrix}$$

$$\begin{aligned} I_{y=1,x=1} \star h &= \sum_{i=-n_2}^{n_2} \sum_{j=-m_2}^{m_2} I(x-i,\, y-j) \cdot h(i,\, j) \\ &= \sum_{i=-1}^{1} \sum_{j=-1}^{1} I(1-i,\, 1-j) \cdot h(i,\, j) \\ &= I_{00} \cdot h_{00} + I_{01} \cdot h_{01} + I_{02} \cdot h_{02} + \\ &\quad I_{10} \cdot h_{10} + I_{11} \cdot h_{11} + I_{12} \cdot h_{12} + \\ &\quad I_{20} \cdot h_{20} + I_{21} \cdot h_{21} + I_{22} \cdot h_{22} \\ &= 0 \cdot 0 + 3 \cdot 1 + 6 \cdot 0 + \\ &\quad 1 \cdot 0 + 4 \cdot 1 + 7 \cdot 0 + \\ &\quad 2 \cdot 0 + 5 \cdot 1 + 8 \cdot 0 \\ &= 3 + 4 + 5 = 12 \end{aligned}$$

$$\begin{aligned} I_{y=1,x=2} \star h &= \sum_{i=-1}^{1} \sum_{j=-1}^{1} I(2-i,\, 1-j) \cdot h(i,\, j) \\ &= 3 \cdot 0 + 6 \cdot 1 + 9 \cdot 0 + \\ &\quad 4 \cdot 0 + 7 \cdot 1 + 10 \cdot 0 + \\ &\quad 5 \cdot 0 + 8 \cdot 1 + 11 \cdot 0 \\ &= 6 + 7 + 8 = 21 \end{aligned}$$

$$I \star h = \begin{bmatrix} I_{y=1,x=1} & I_{y=1,x=2} \\ I_{y=2,x=1} & I_{y=2,x=2} \end{bmatrix} = \begin{bmatrix} 12 & 21 \\ 15 & 24 \end{bmatrix}$$

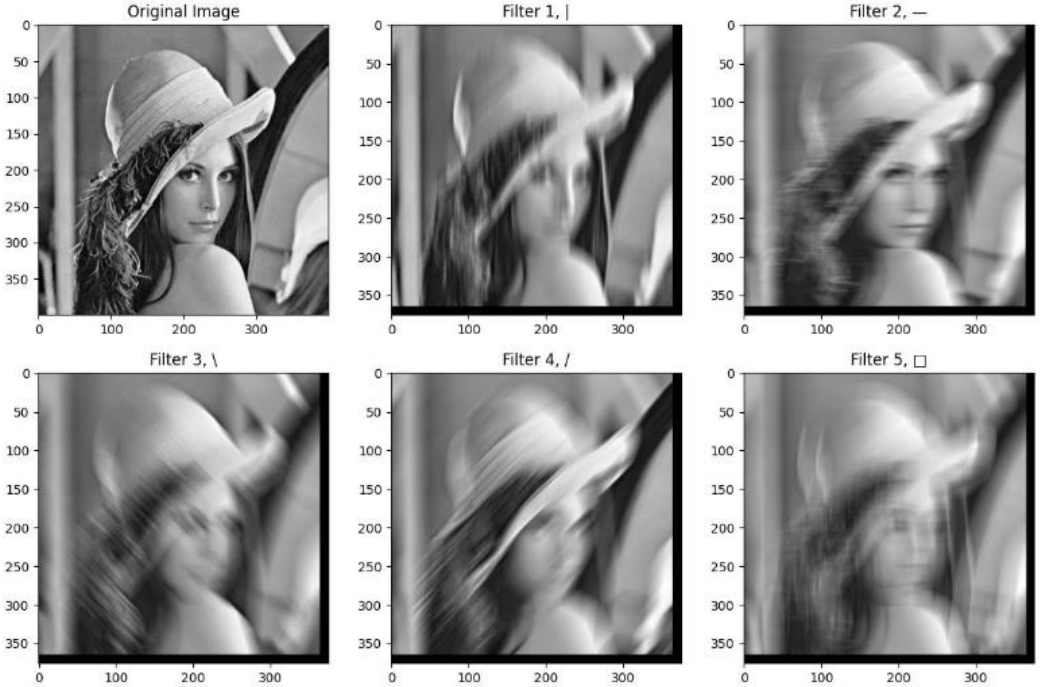
```
    for i in range(k):
        for j in range(k):
            a = img[y+i+k_start][x+j+k_start] * filter_[i][j]
            print(f"i[{y+i+k_start},{x+j+k_start}]*f[{i},{j}]={a}",
end='\t")
            sum += a
        print("")
    # Return the result of convolution for a given point
    return sum

# Kernel size
k = 3
# Filter for convolution
filter_ = [
    [0, 1, 0],
    [0, 1, 0],
    [0, 1, 0],
]
# Input image
img = [
    [0, 3, 6, 9 ],
    [1, 4, 7, 10],
    [2, 5, 8, 11],
    [3, 6, 9, 12]
]
```

```
i[0,0]*f[0,0]=0         i[0,1]*f[0,1]=3         i[0,2]*f[0,2]=0
i[1,0]*f[1,0]=0         i[1,1]*f[1,1]=4         i[1,2]*f[1,2]=0
i[2,0]*f[2,0]=0         i[2,1]*f[2,1]=5         i[2,2]*f[2,2]=0
process (x=1, y=1) image point

i[0,1]*f[0,0]=0         i[0,2]*f[0,1]=6         i[0,3]*f[0,2]=0
i[1,1]*f[1,0]=0         i[1,2]*f[1,1]=7         i[1,3]*f[1,2]=0
i[2,1]*f[2,0]=0         i[2,2]*f[2,1]=8         i[2,3]*f[2,2]=0
process (x=2, y=1) image point

i[1,0]*f[0,0]=0         i[1,1]*f[0,1]=4         i[1,2]*f[0,2]=0
i[2,0]*f[1,0]=0         i[2,1]*f[1,1]=5         i[2,2]*f[1,2]=0
i[3,0]*f[2,0]=0         i[3,1]*f[2,1]=6         i[3,2]*f[2,2]=0
process (x=1, y=2) image point

i[1,1]*f[0,0]=0         i[1,2]*f[0,1]=7         i[1,3]*f[0,2]=0
i[2,1]*f[1,0]=0         i[2,2]*f[1,1]=8         i[2,3]*f[1,2]=0
i[3,1]*f[2,0]=0         i[3,2]*f[2,1]=9         i[3,3]*f[2,2]=0
process (x=2, y=2) image point

array([[12., 21.],
       [15., 24.]])
```

Convolution for Black and White Images

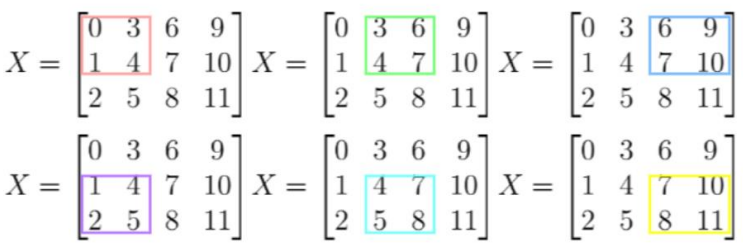
```
    # Determine the size of the kernel
    k = filter_.shape[0]

    # Define the boundaries within which convolution will be
performed
    row_start = int(np.floor(k/2))
    row_end = height-k+2
    col_start = int(np.floor(k/2))
    col_end = width-k+2

    # Create a new image filled with zeros, smaller than the
original by the size of the kernel
    new_image = np.zeros((height-k+1, width-k+1))
    for y in range(row_start, row_end):
        # for x in range(col_start, col_end):
            # Convolve for each point in the area defined by the
borders
            new_image[y-row_start][x-col_start] = convolve(img,
filter_, k, x, y)
    return new_image


# Load the image
img = cv2.imread('Lenna_(test_image).png', 0)
height, width = img.shape


# Create a figure with 2 graphs
fig, axs = plt.subplots(ncols=2, nrows=1, figsize=(14, 6))
```

```
    filter2 = np.zeros((k, k), dtype=int)
    filter2[k//2, :] = 1
    """
    # Filter 3
    """
    [[1, 0, 0],
     [0, 1, 0],
     [0, 0, 1]],
    """
    filter3 = np.zeros((k, k), dtype=int)
    np.fill_diagonal(filter3, 1)
    """
    # Filter 4
    """
    [[0, 0, 1],
     [0, 1, 0],
     [1, 0, 0]],
    """
    filter4 = np.zeros((k, k), dtype=int)
    np.fill_diagonal(np.fliplr(filter4), 1)

    """
    # Filter 5
    """
    [[1, 1, 1],
     [1, 0, 1],
     [1, 1, 1]]
```



$$Y = \max_{i \in I, j \in J} X[i, j]$$

$$X = \begin{bmatrix} 0 & 3 & 6 & 9 \\ 1 & 4 & 7 & 10 \\ 2 & 5 & 8 & 11 \end{bmatrix}$$



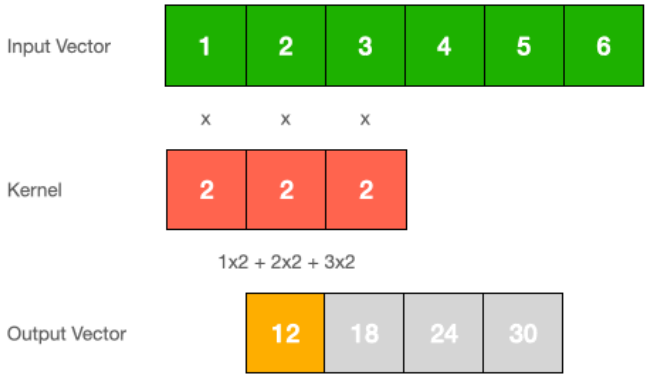
$$Y = \begin{bmatrix} 4 & 7 & 10 \\ 5 & 8 & 11 \end{bmatrix}$$

The Second Step:

The Second Step:

- What is Convolutional Neural Network?
- Convolutional Layer
- Padding and Stride
- Pooling
- ReLU
- Basic Python Implementation

Kernel/filter



Kernel 3

```
nn.Conv1d(in_channels=1, out_channels=1,
   kernel_size=3,
   bias=False
)
```

Size

$$L_{out} = \left\lfloor \frac{L_{in} + 2 \times \text{padding} - \text{dilation} \times (\text{kernel\_size} - 1) - 1}{\text{stride}} + 1 \right\rfloor$$

$$4 = \frac{6 + (2 * 0) - 1 * (3 - 1) - 1}{1} + 1$$

jinglescode.github.io

1	0	1
0	1	0
1	0	1

Filter 1

Matrix

0	1	0
0	1	0
0	1	0

Filter 2

Matrix

Original	Gaussian Blur	Sharpen	Edge Detection
$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$
			

Stride

Input

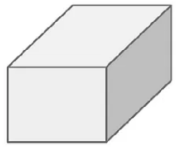
a	b	c	d	e	f
g	h	i	j	k	l
m	n	o	p	q	r
s	t	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Kernel

U’	V’	W’
X’	Y’	Z’
L’	M’	N’

Sum

=	**a.U’**	+	**b.V’**	+	**c.W’**	+
	g.X’	+	h.Y’	+	i.Z’	+
	m.L’	+	n.M’	+	o.N’	+



N x N x Ch

Filter : f x f Ch
Stride : S
Padding : P

$$= \left(\frac{n+2p-f}{S}+1\right) \text{ x } \left(\frac{n+2p-f}{S}+1\right) \text{ x NCh}$$

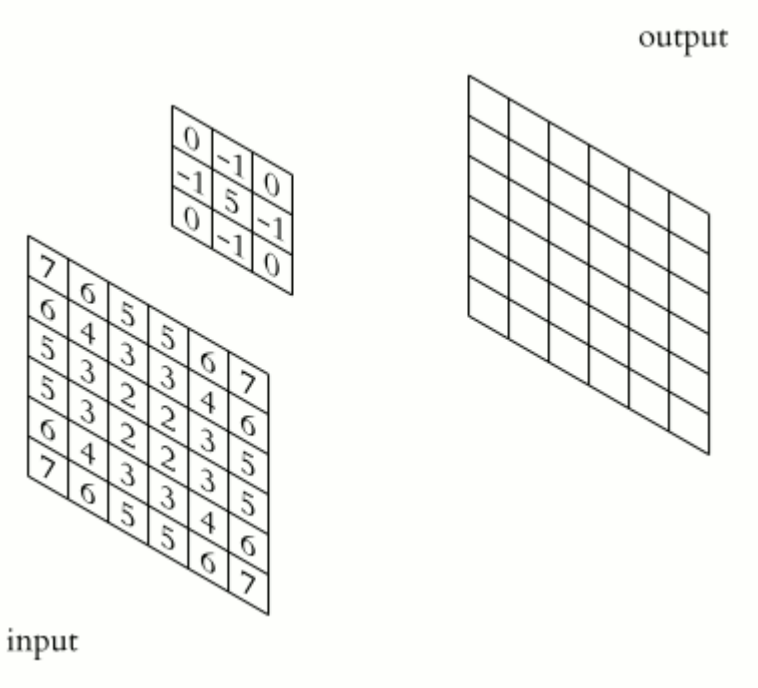
$$h' = \left\lfloor \frac{h-f+p}{s} + 1 \right\rfloor$$

$$w' = \left\lfloor \frac{w-f+p}{s} + 1 \right\rfloor$$

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

padding = p stride= s resultant dimension

Stride



Those are the network parameters to be learned.

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

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

Matrix

-1	1	-1
-1	1	-1
-1	1	-1

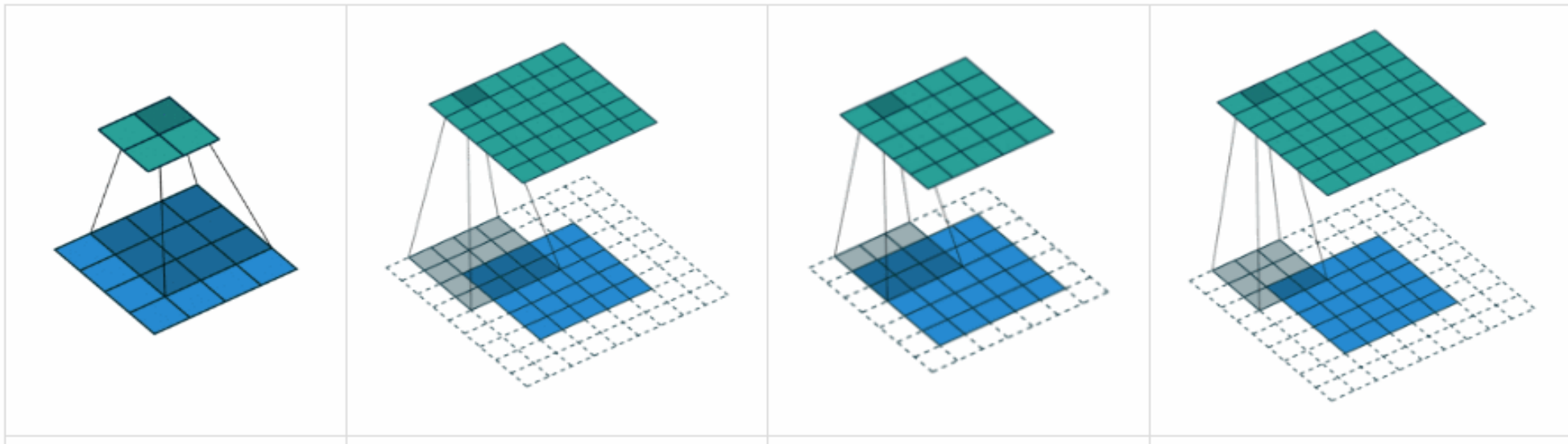
Filter 2

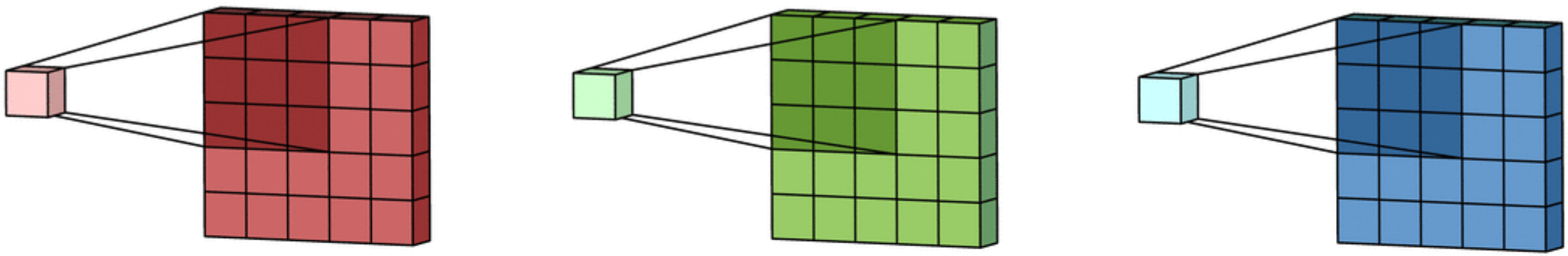
Matrix

⋮

Property 1

Each filter detects a small pattern (3 x 3).





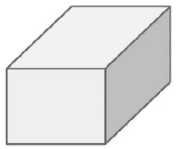
Padding , Stride

0	0	0	0	0	0	0
0	60	113	56	139	85	0
0	73	121	54	84	128	0
0	131	99	70	129	127	0
0	80	57	115	69	134	0
0	104	126	123	95	130	0
0	0	0	0	0	0	0

Kernel

0	-1	0
-1	5	-1
0	-1	0

114				



N x N x Ch

Filter : f x f Ch
Stride : S
Padding : P

$$= \left(\frac{n+2p-f}{S}+1\right) \text{ x } \left(\frac{n+2p-f}{S}+1\right) \text{ x NCh}$$

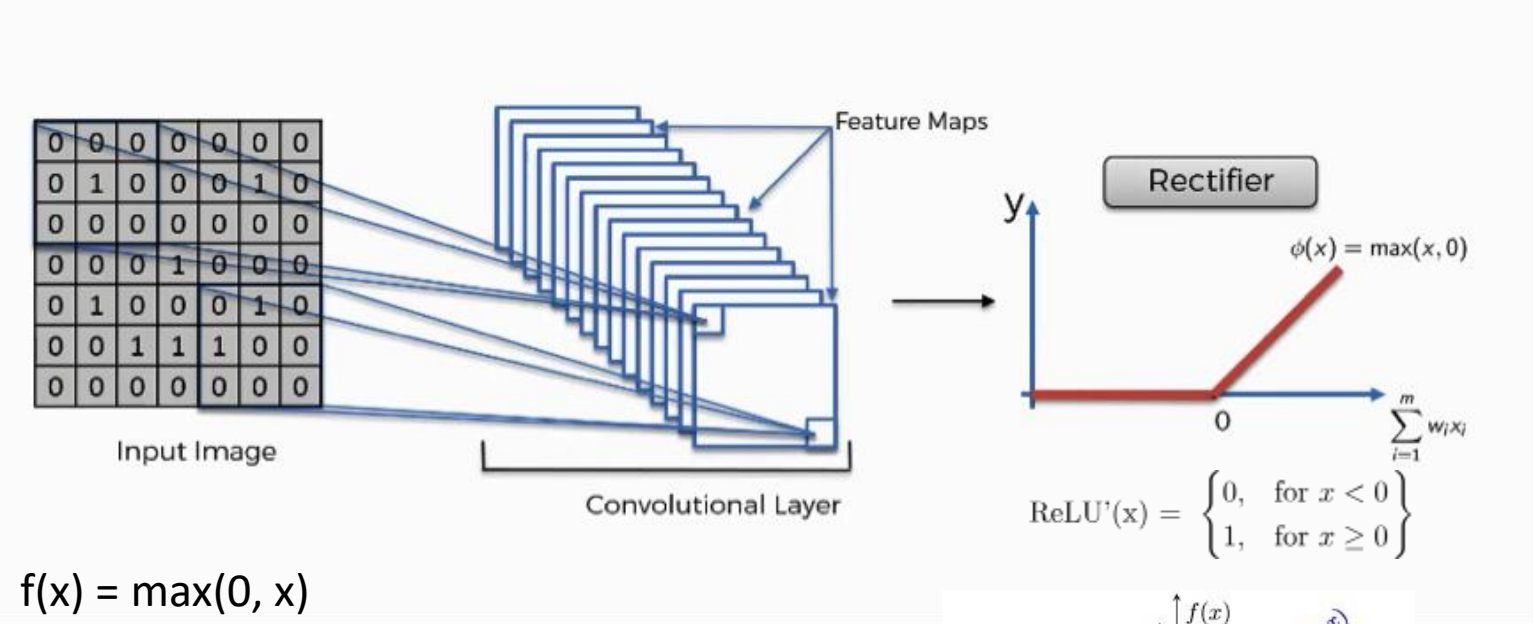
$$output = \frac{W - N + 2P}{S} + 1$$

W = Panjang/Tinggi Input
N = Panjang/Tinggi Filter
P = Zero Padding
S = Stride

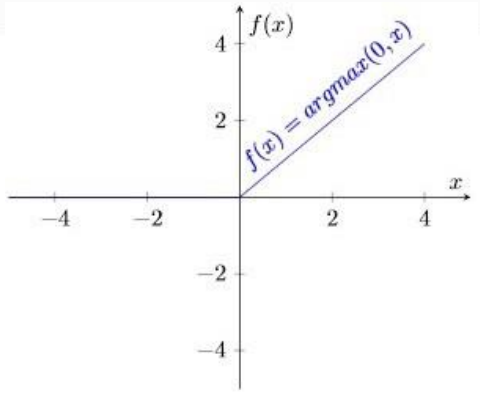
[(W−K+2P)/S]+1

ReLU
The rectified linear activation function

ReLU merupakan fungsi non-linear yang sederhana dan efektif dalam mempercepat proses pelatihan jaringan saraf tiruan, Goodfellow et al. (2016)

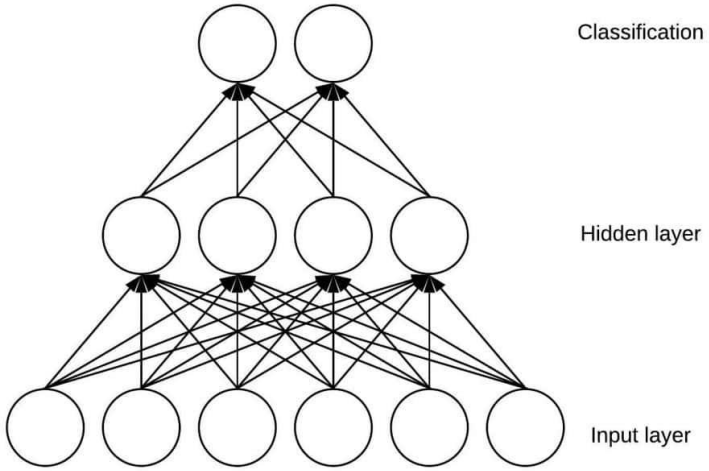


f(x) = max(0, x)
f(x): fungsi aktivasi ReLU
x: input ke neuron dalam jaringan saraf

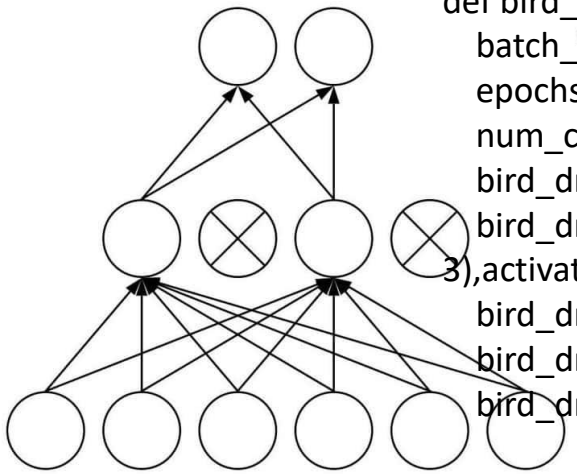



```
model = Sequential()
model.add(Dense(64, input_dim=100))
model.add(Activation('relu'))
model.add(Dense(10))
model.add(Activation('softmax'))
```

Dropout

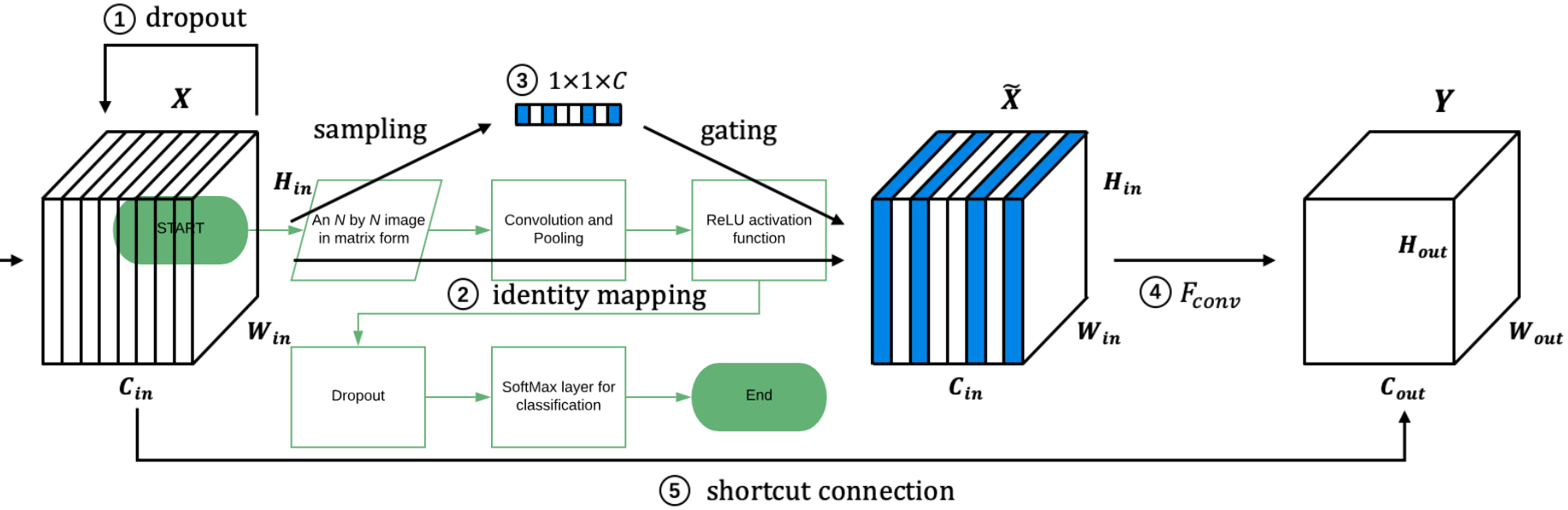


Without Dropout

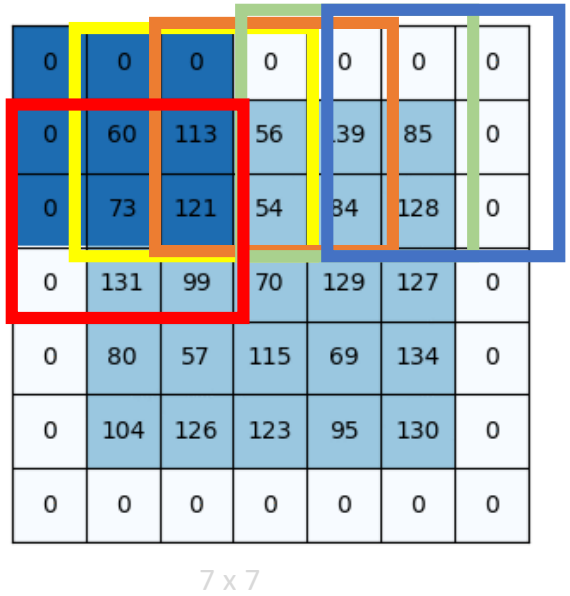


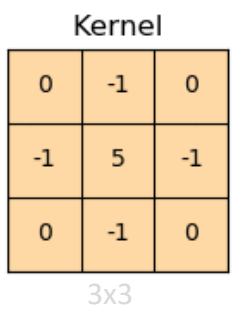
With Dropout

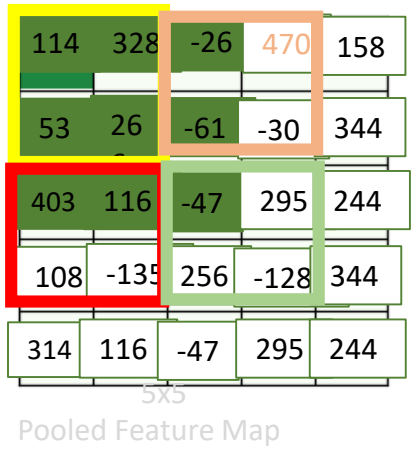
```
def bird_model()
    batch_size = 64
    epochs = 20
    num_classes = 10
    bird_drop_model = Sequential()
    bird_drop_model.add(Conv2D(32, kernel_size=(3,
3),activation='linear',padding='same',input_shape=(28,28,1)))
    bird_drop_model.add(LeakyReLU(alpha=0.1))
    bird_drop_model.add(MaxPooling2D((2, 2),padding='same'))
    bird_drop_model.add(Dropout(0.25))
```

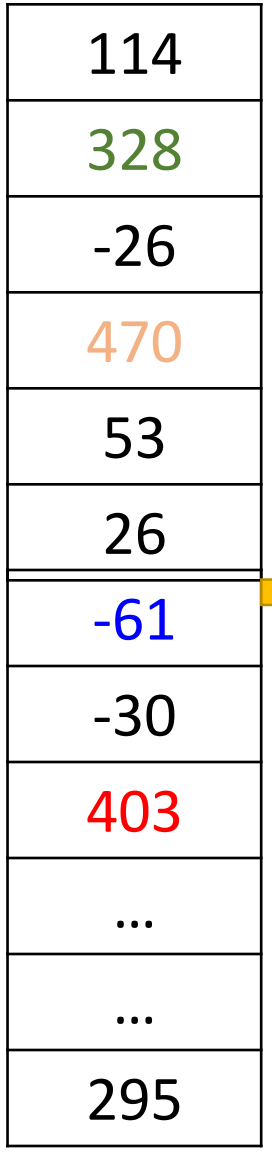


Pooling, Flatten



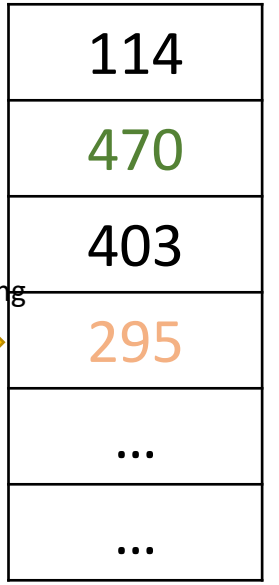




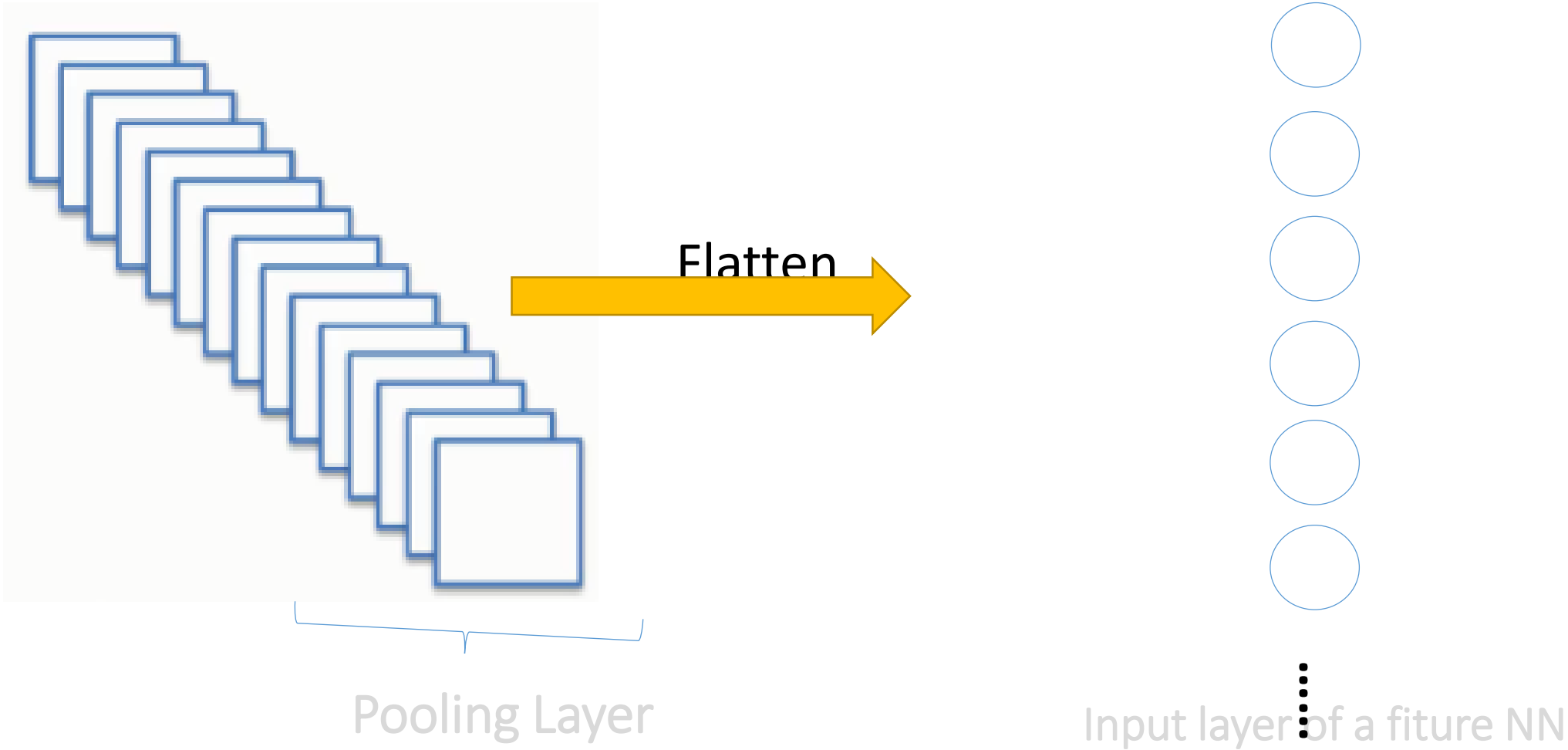


Flatten

Max-pooling

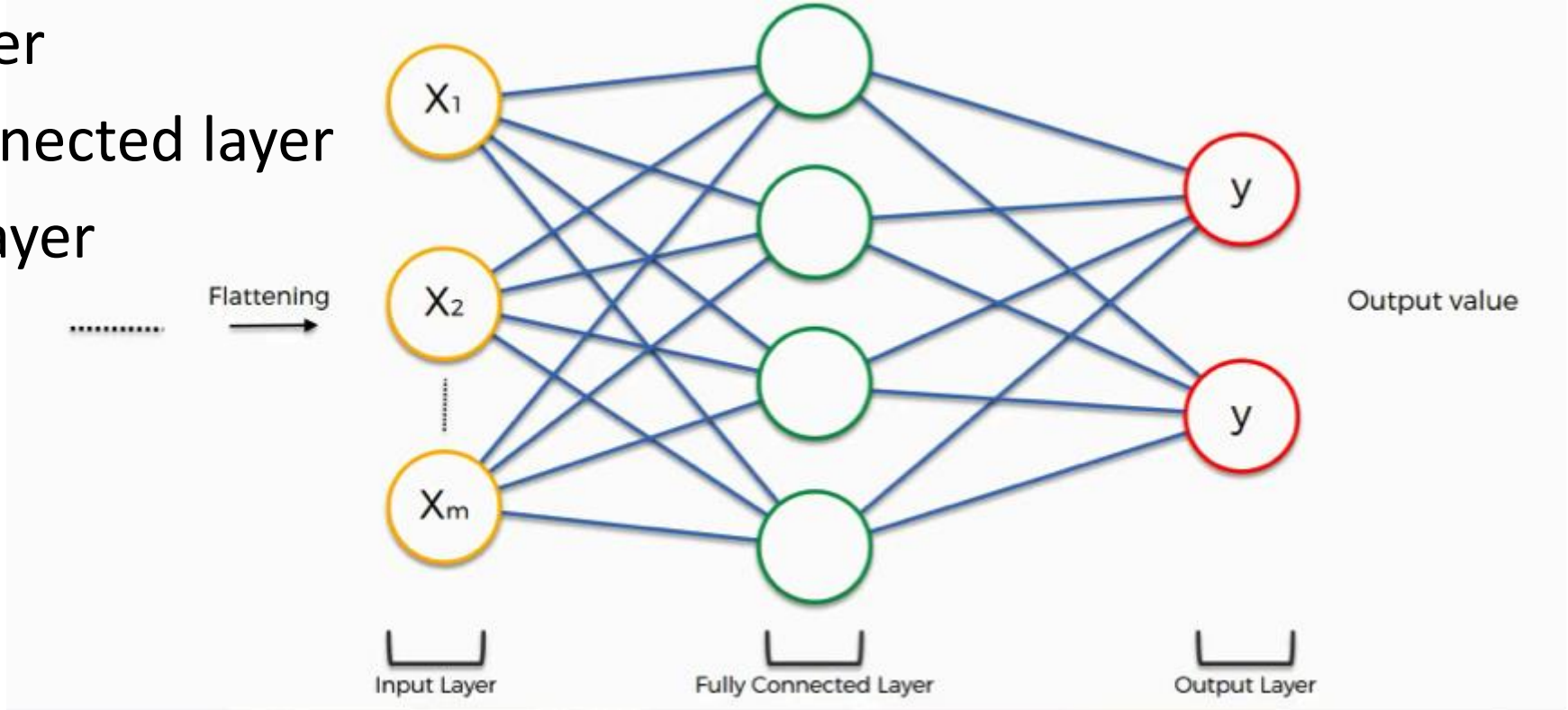


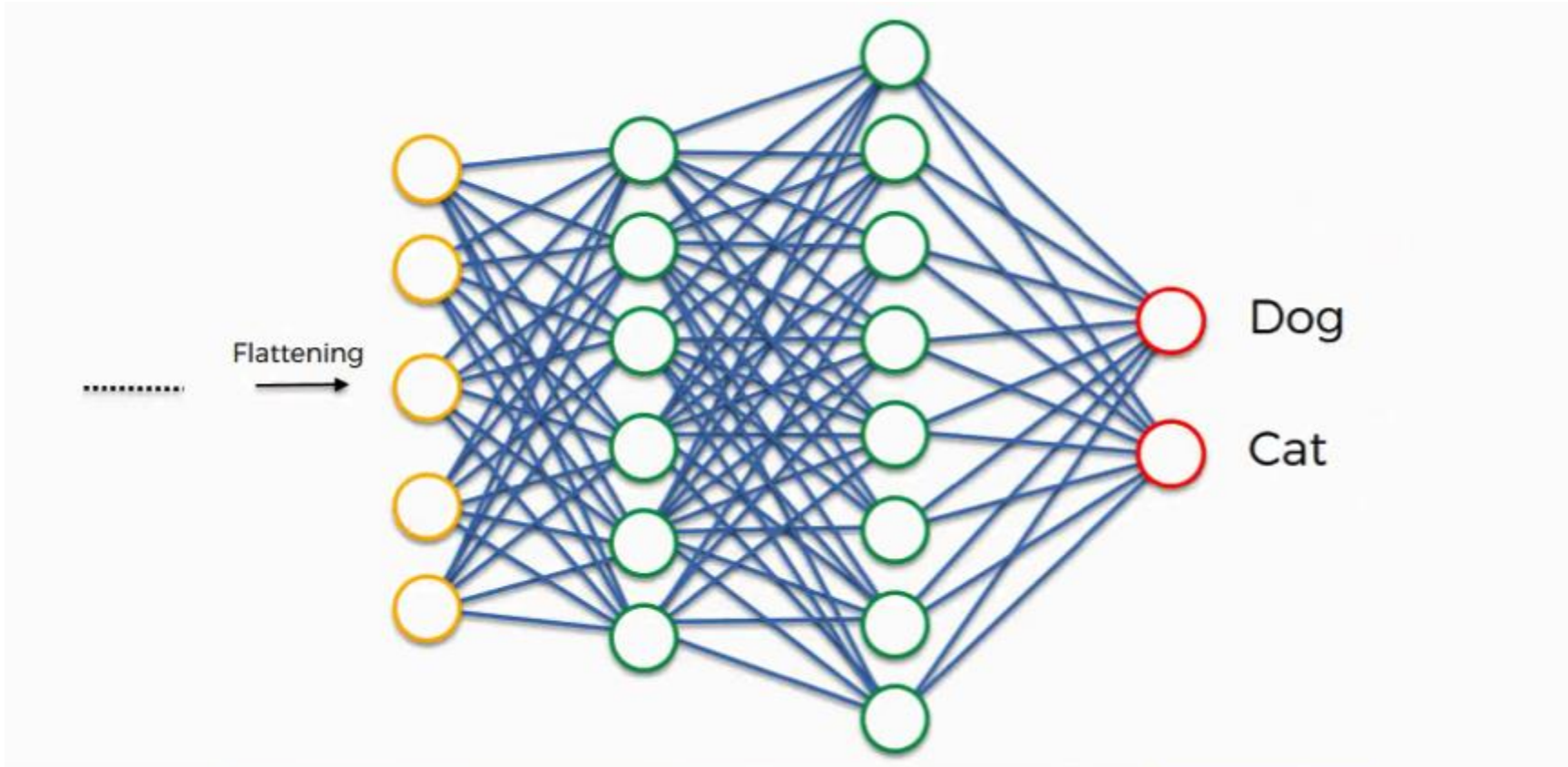
Pooling, Flatten



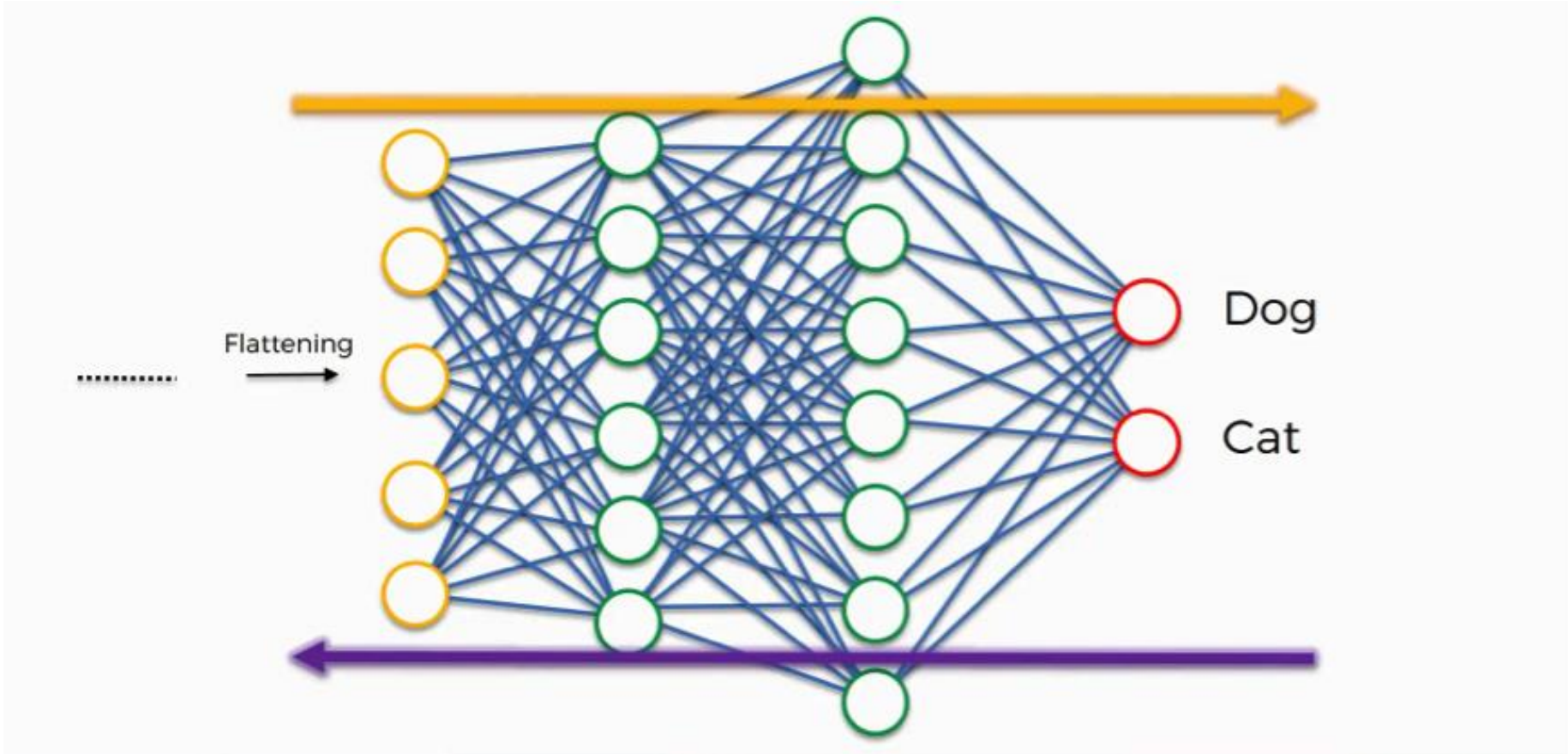
Full connection step:

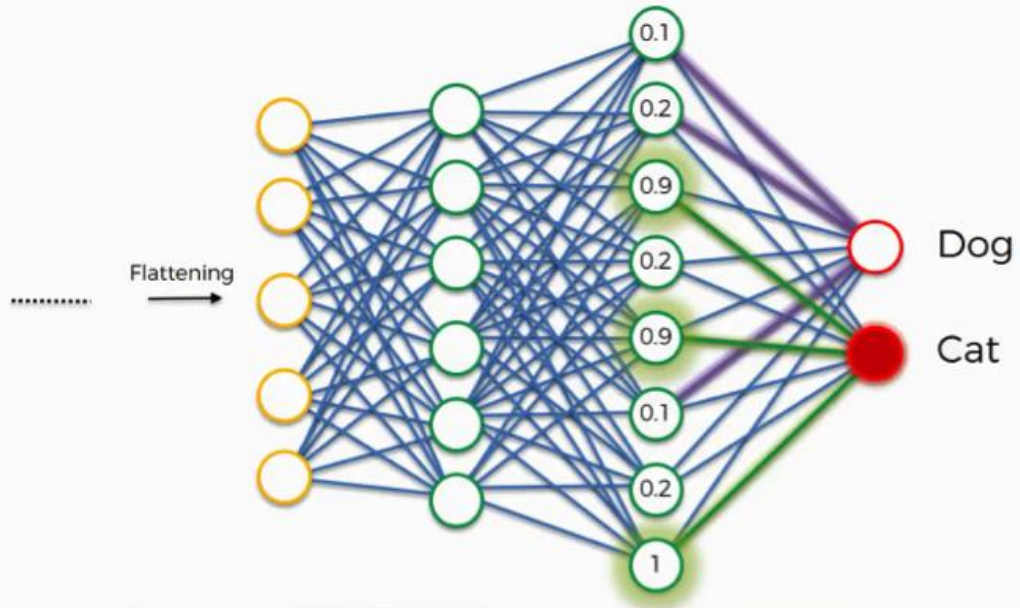
- Input layer
- Fully-connected layer
- Output layer

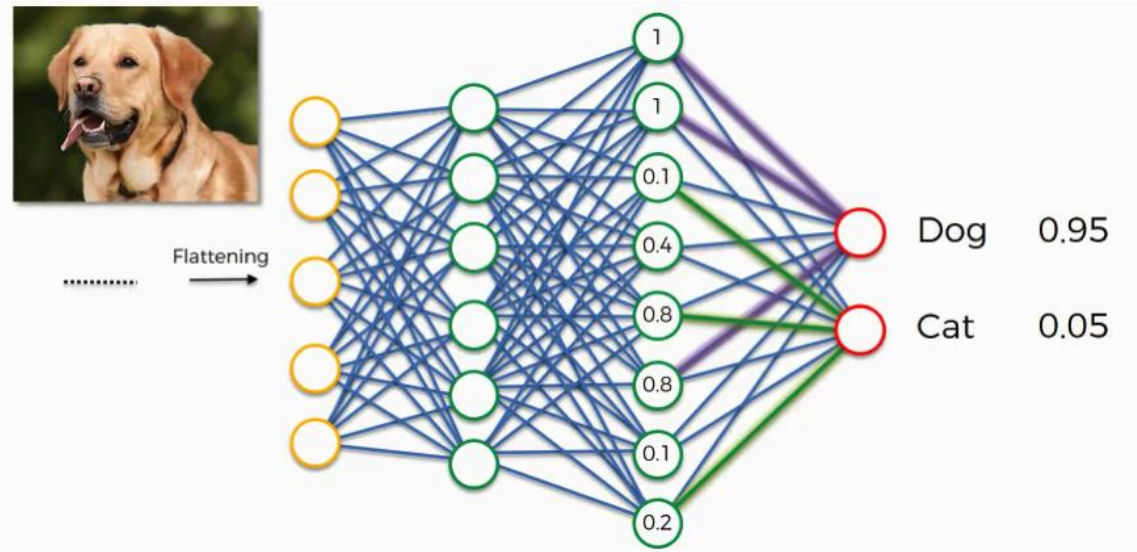


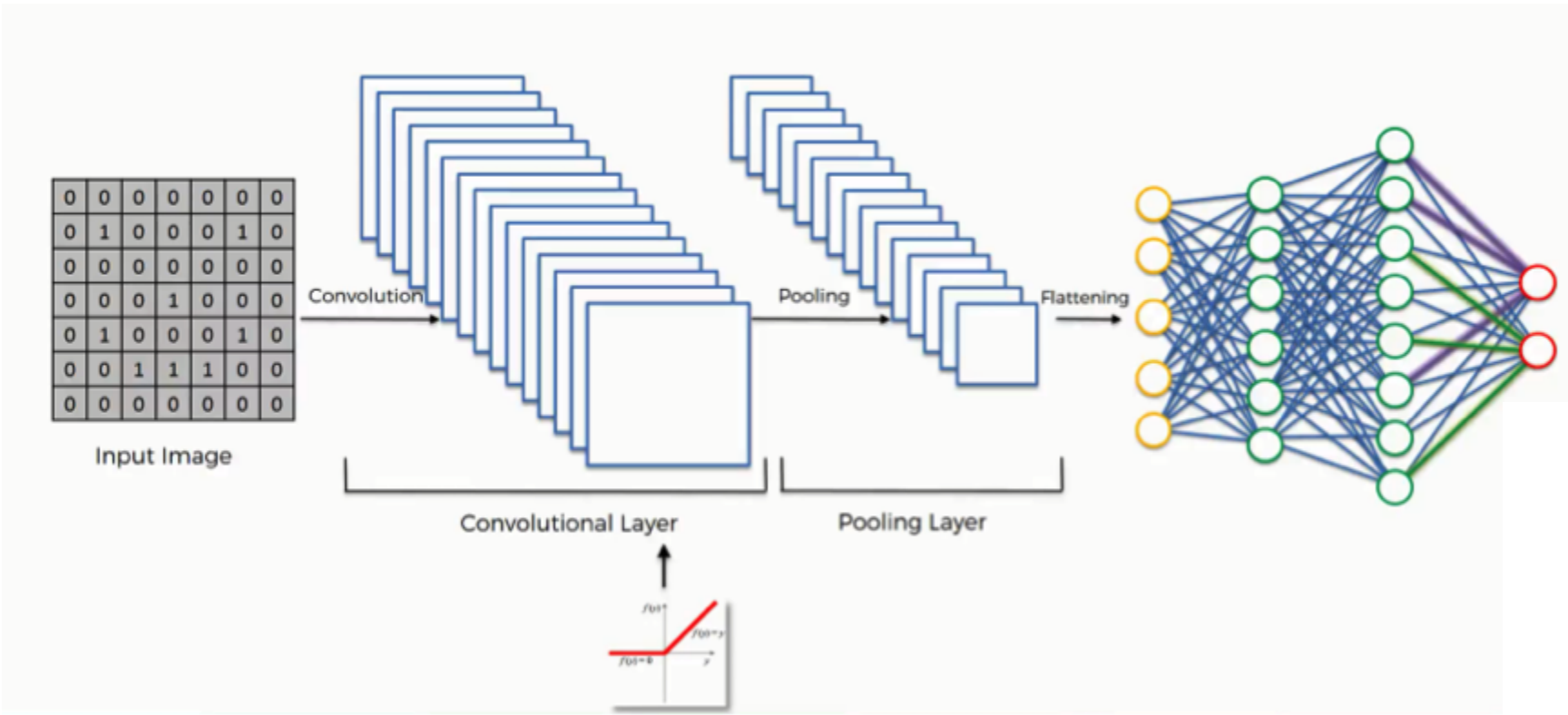

Flattening
Dog
Cat

Class Recognition




Flattening
0.1
0.2
0.9
0.2
0.9
0.1
0.2
1
Dog
Cat

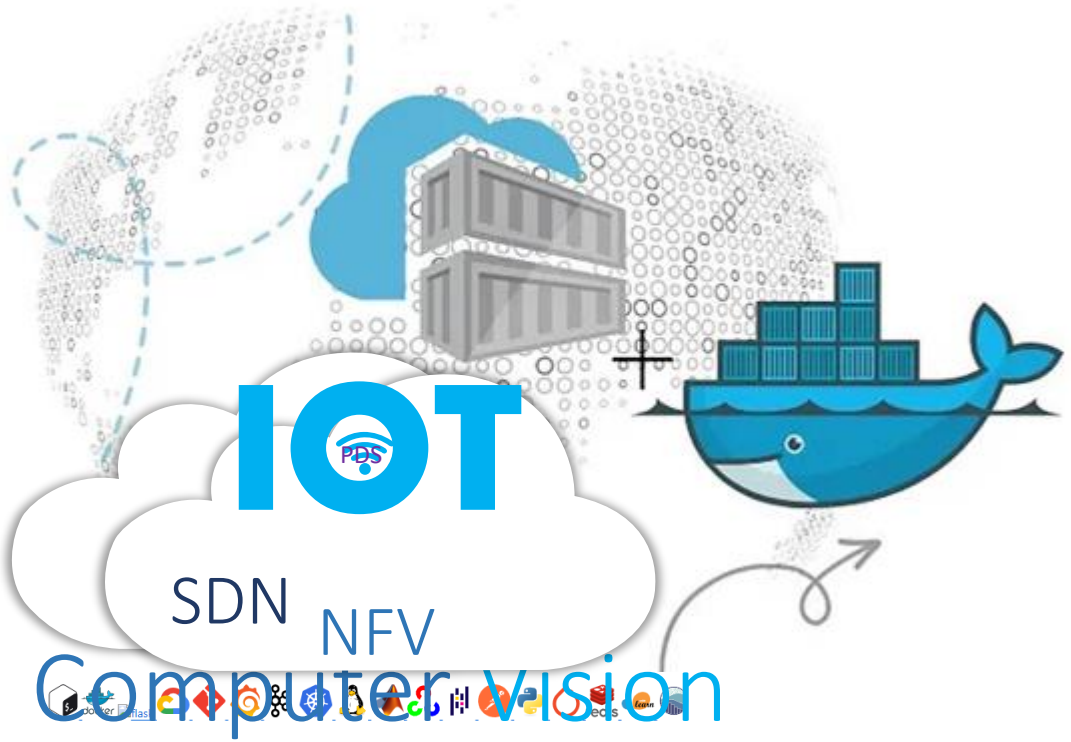

Flattening
1
1
0.1
0.4
0.8
0.8
0.1
0.2
Dog 0.95
Cat 0.05



$$z =: \sum_{i=1}^{n} w_i x_i$$

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

$$p_i = \text{softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$



Thank you!

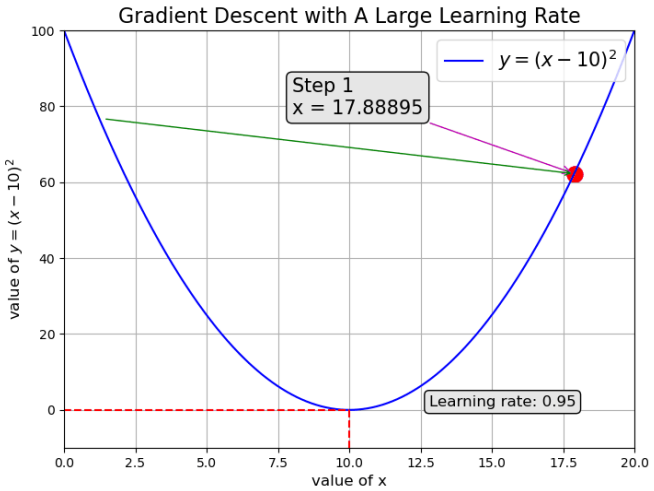
Thank you For being a great class!

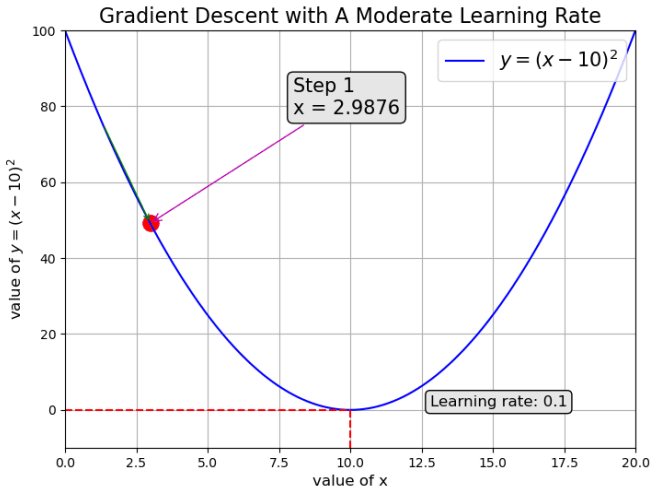
https://github.com/siagianp

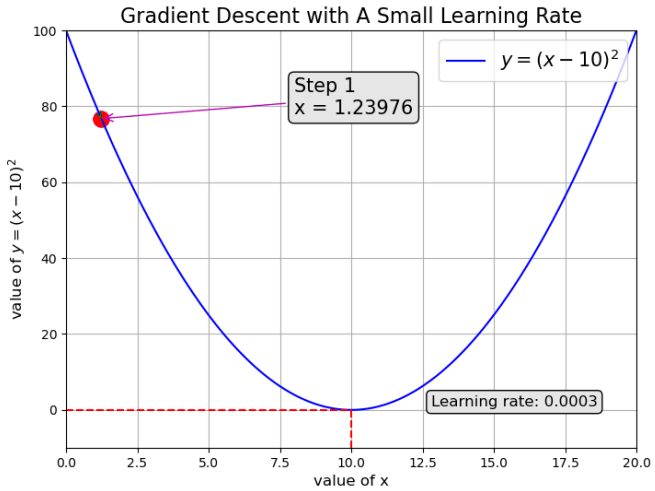
https://github.com/amelcharolinesgn2/IoT_simulator-mqtt-NodeRed

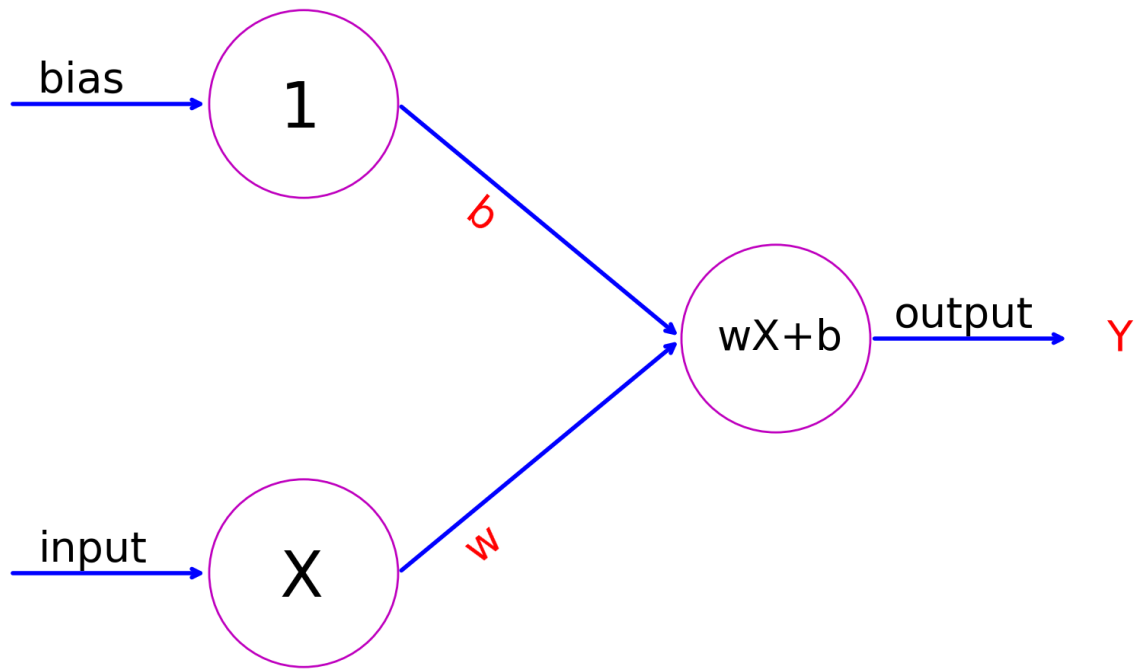
https://www.youtube.com/@AmelOline/videos

Gradient Descent -- Where Magic Happens



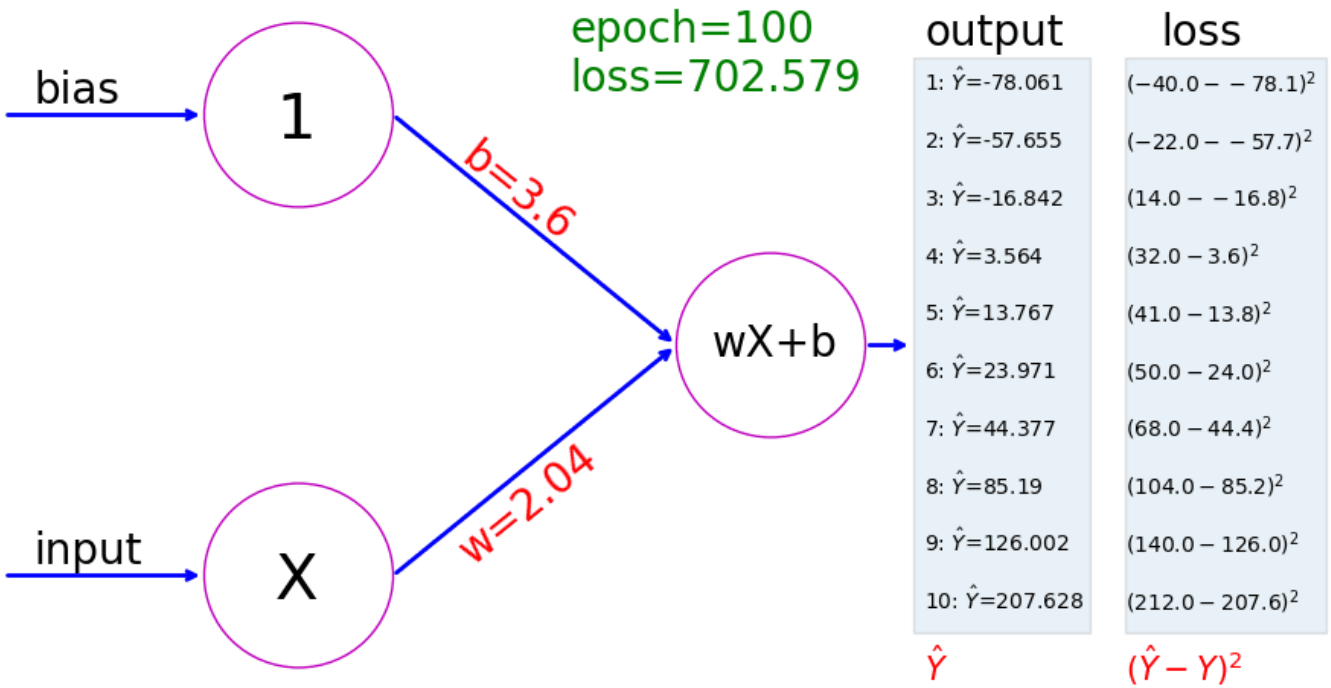




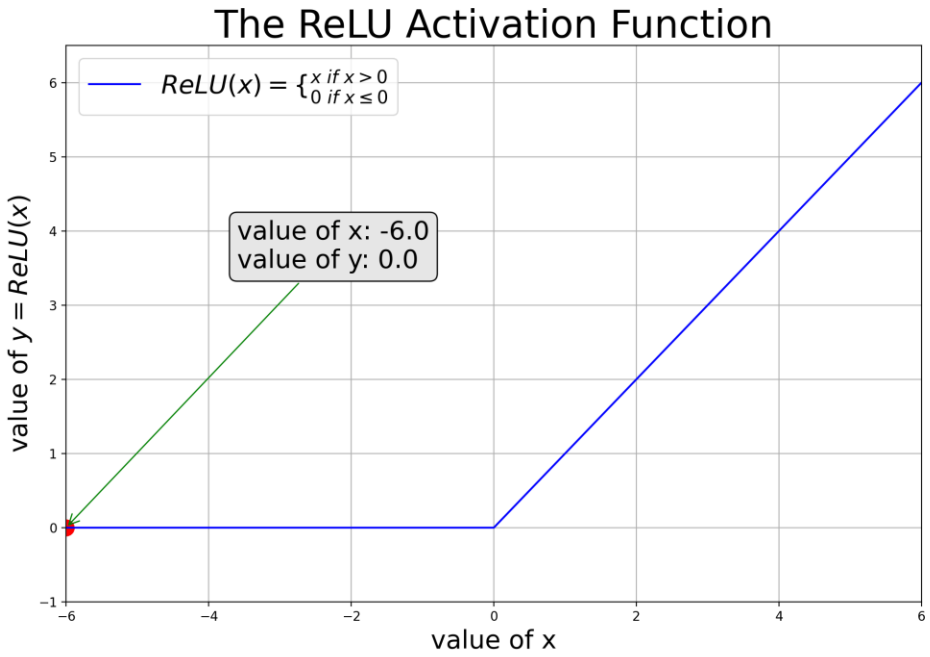

bias
1
b
wX+b
output
Y
input
X
w

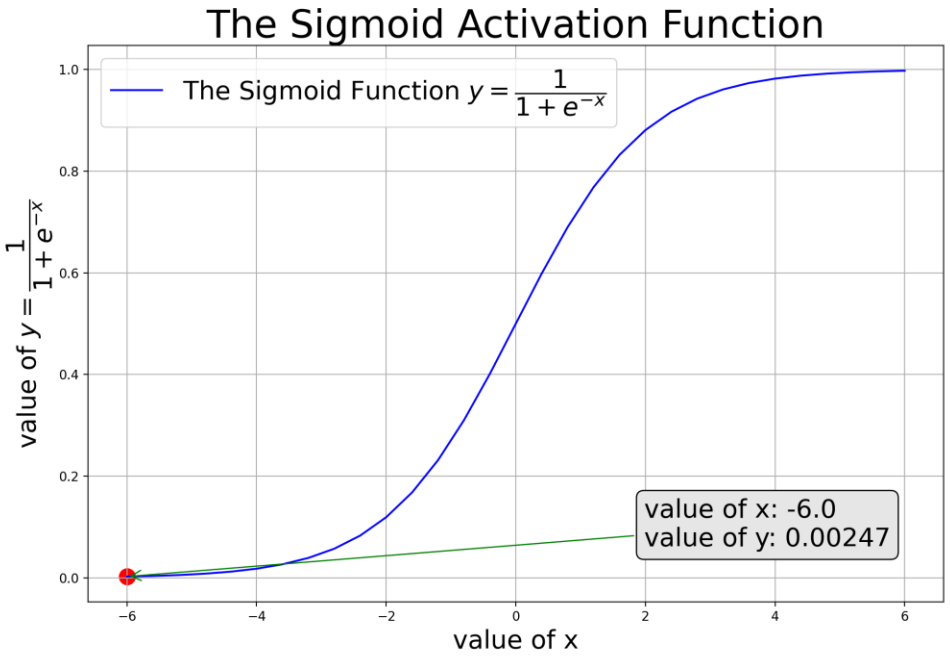
Introduction to Neural Networks

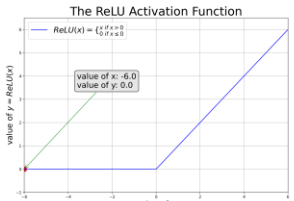
A neural network learns from ten pairs of values: X=−40, Y=−40;X=0, Y=32;...;X=100, Y=212. The model first assigns values to w and b in Y=wX+b; it then uses gradient descent to adjust the values of w and b; finally it figures out a linear relation between X and Y that corresponds to the relation between Celsius and Fahrenheit Y=1.8∗X+32. This animation shows the value of w and b in each step:

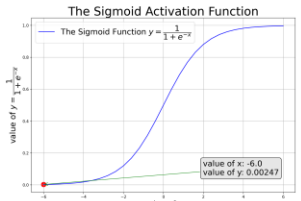


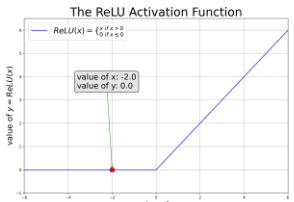
Activation Functions

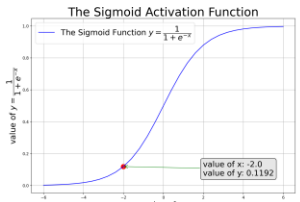


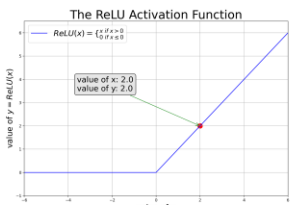


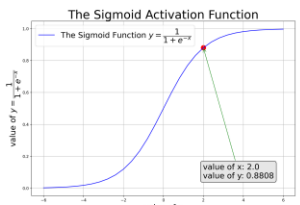

The ReLU Activation Function
$ReLU(x) = \begin{cases} x & if\ x > 0 \\ 0 & if\ x \leq 0 \end{cases}$
value of x: -6.0
value of y: 0.0
value of y = ReLU(x)
value of x

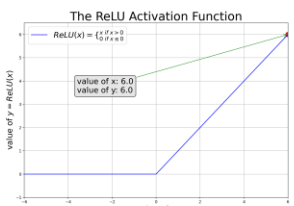

The Sigmoid Activation Function
The Sigmoid Function $y = \frac{1}{1+e^{-x}}$
value of x: -6.0
value of y: 0.00247
value of $y = \frac{1}{1+e^{-x}}$
value of x

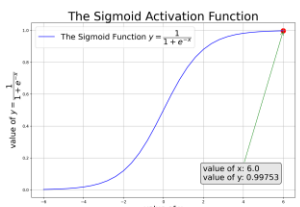

The ReLU Activation Function
$ReLU(x) = \begin{cases} x & if\ x > 0 \\ 0 & if\ x \leq 0 \end{cases}$
value of x: -2.0
value of y: 0.0
value of y = ReLU(x)
value of x

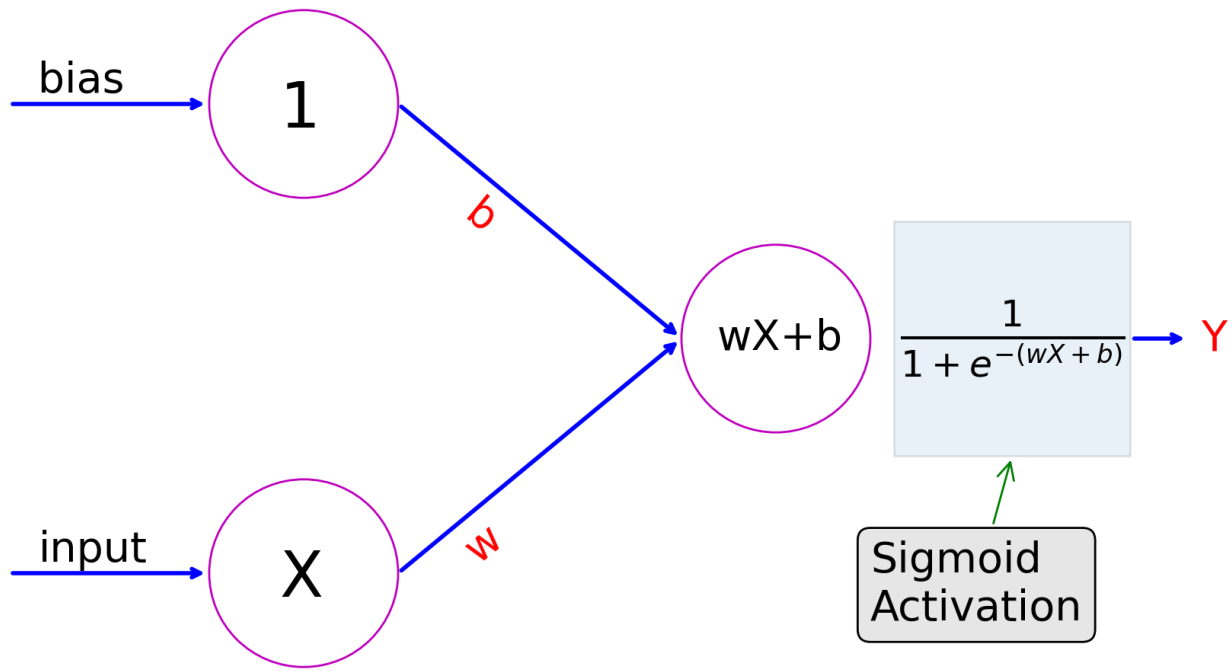

The Sigmoid Activation Function
The Sigmoid Function $y = \frac{1}{1+e^{-x}}$
value of x: -2.0
value of y: 0.1192
value of $y = \frac{1}{1+e^{-x}}$
value of x


The ReLU Activation Function
$ReLU(x) = \begin{cases} x & if\ x > 0 \\ 0 & if\ x \leq 0 \end{cases}$
value of x: 2.0
value of y: 2.0
value of y = ReLU(x)
value of x

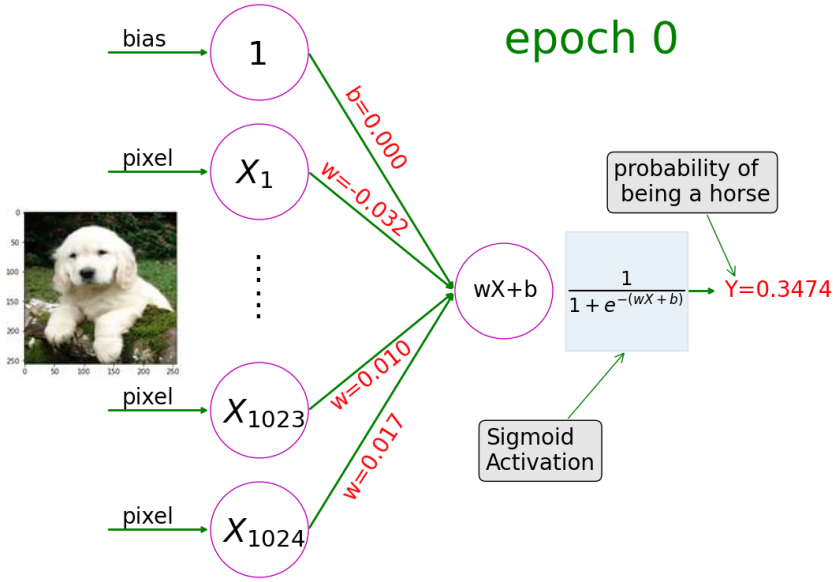

The Sigmoid Activation Function
The Sigmoid Function $y = \frac{1}{1+e^{-x}}$
value of x: 2.0
value of y: 0.8808
value of $y = \frac{1}{1+e^{-x}}$
value of x

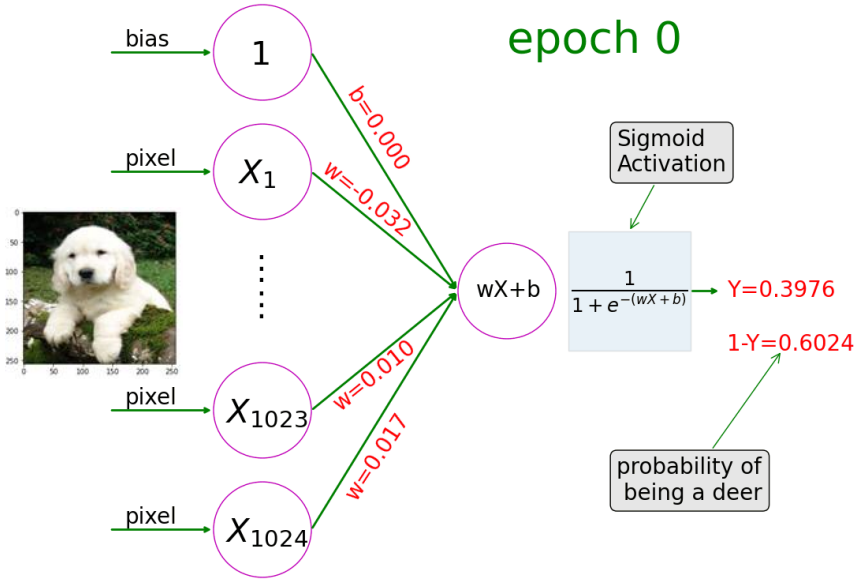

The ReLU Activation Function
$ReLU(x) = \begin{cases} x & if\ x > 0 \\ 0 & if\ x \leq 0 \end{cases}$
value of x: 6.0
value of y: 6.0
value of y = ReLU(x)
value of x


The Sigmoid Activation Function
The Sigmoid Function $y = \frac{1}{1+e^{-x}}$
value of x: 6.0
value of y: 0.99753
value of $y = \frac{1}{1+e^{-x}}$
value of x

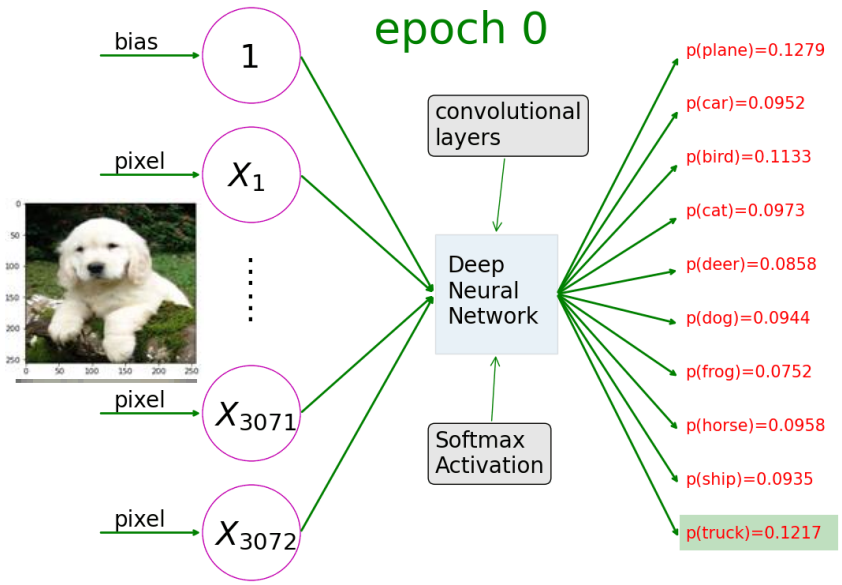

bias
1
b
wX+b
$\frac{1}{1+e^{-(wX+b)}}$
Y
input
X
w
Sigmoid
Activation

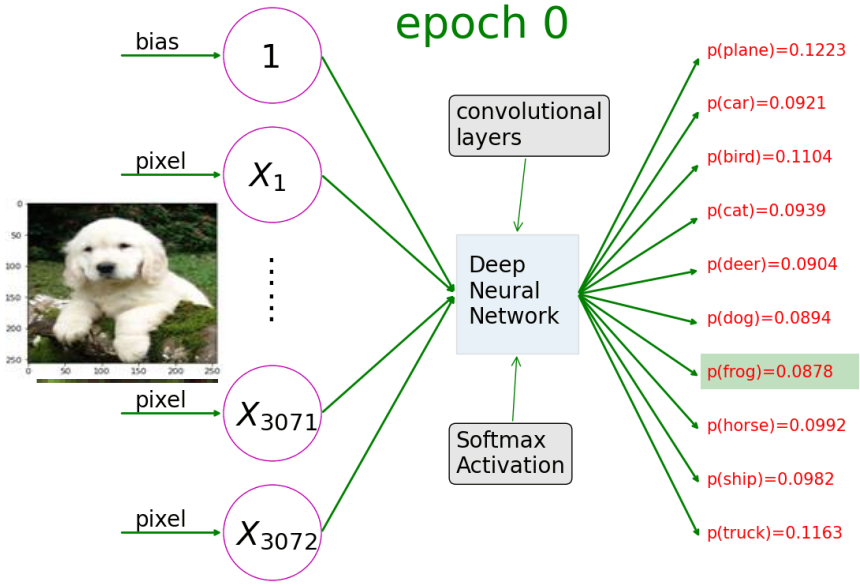
Binary Classifications





Multi-Class Image Classifications





Convolutional Neural Networks