Image classification

6.1. CNN (AlexNet).ipynb

Recap: Regression and Classification

$$L = \frac{1}{N}\sum_{i=1}^{N}(y^i - \hat{y}^i)^2$$

- Regression
- Classification

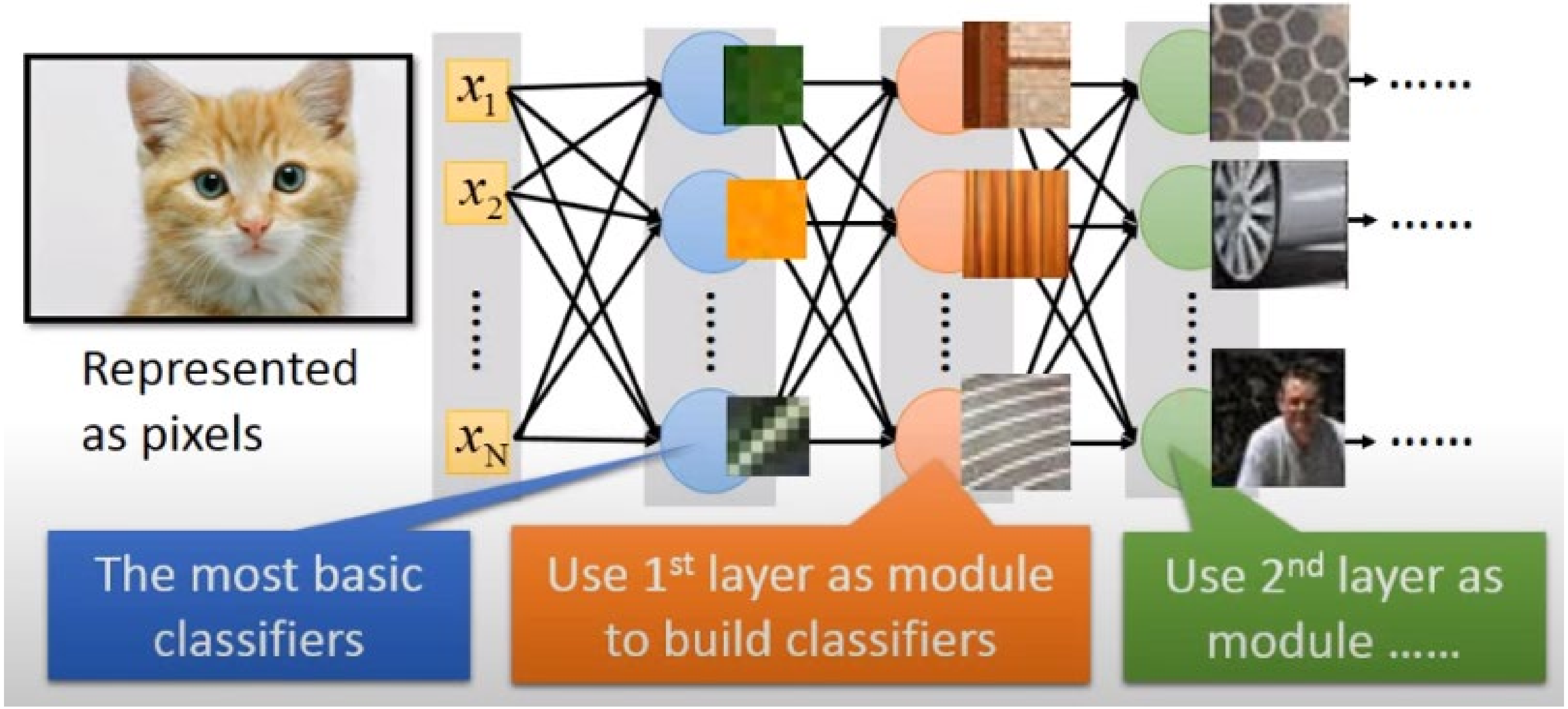
$$y = f(x)$$

$$L = \frac{1}{N}\sum_{i=1}^{N} CE(p, \hat{p})$$

$$CE(p, \hat{p}) = -\sum_{k=1}^{C} p_k ln(\hat{p}_k)$$

Why not using MLP for image classification?

If we feed an image to MLP, then each neuron "sees" the whole image's pixels.



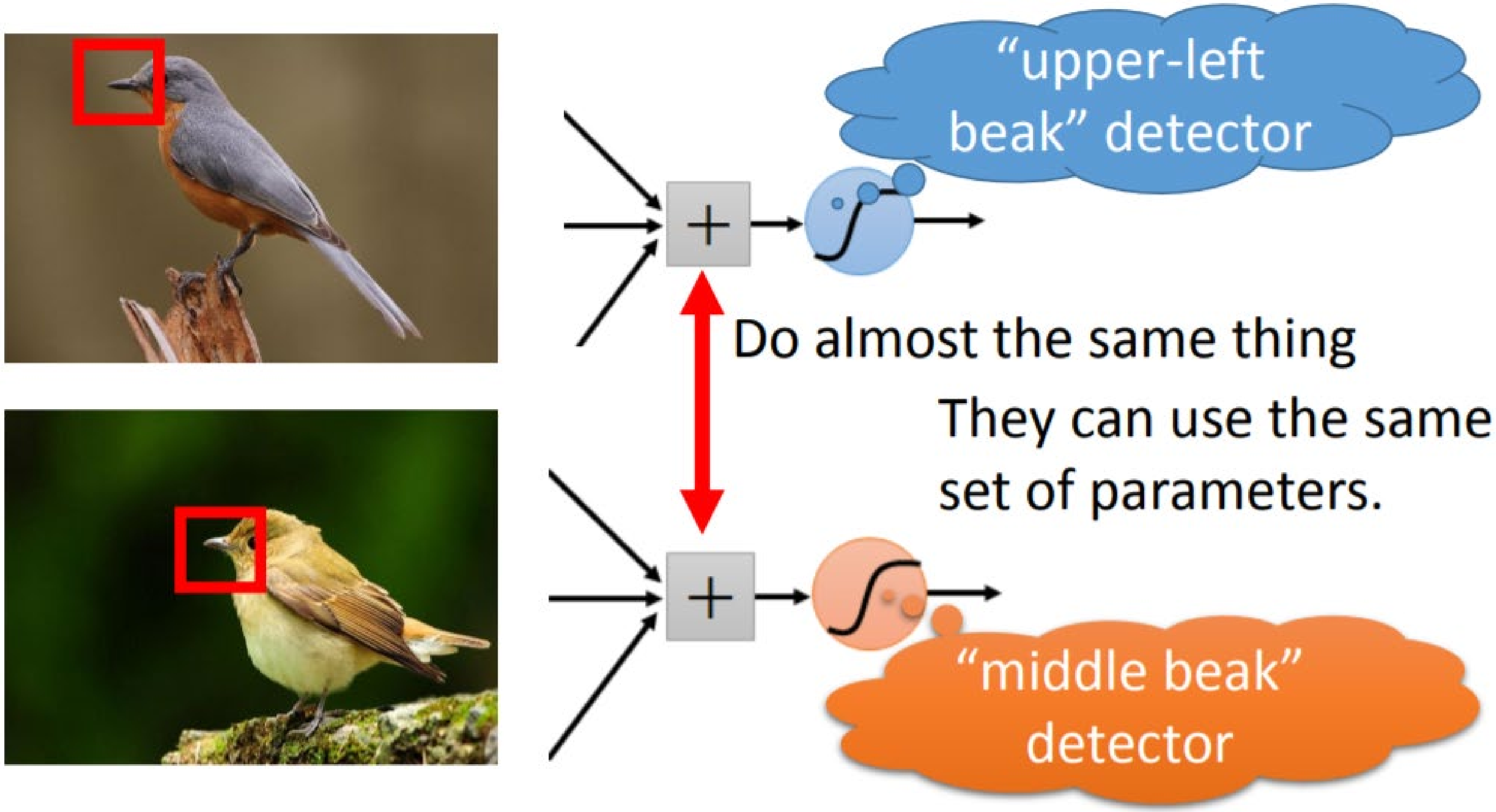
Reference: 李弘毅 ML Lecture 10 https://youtu.be/FrKWiRv254g

Some patterns are much smaller than the whole image

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

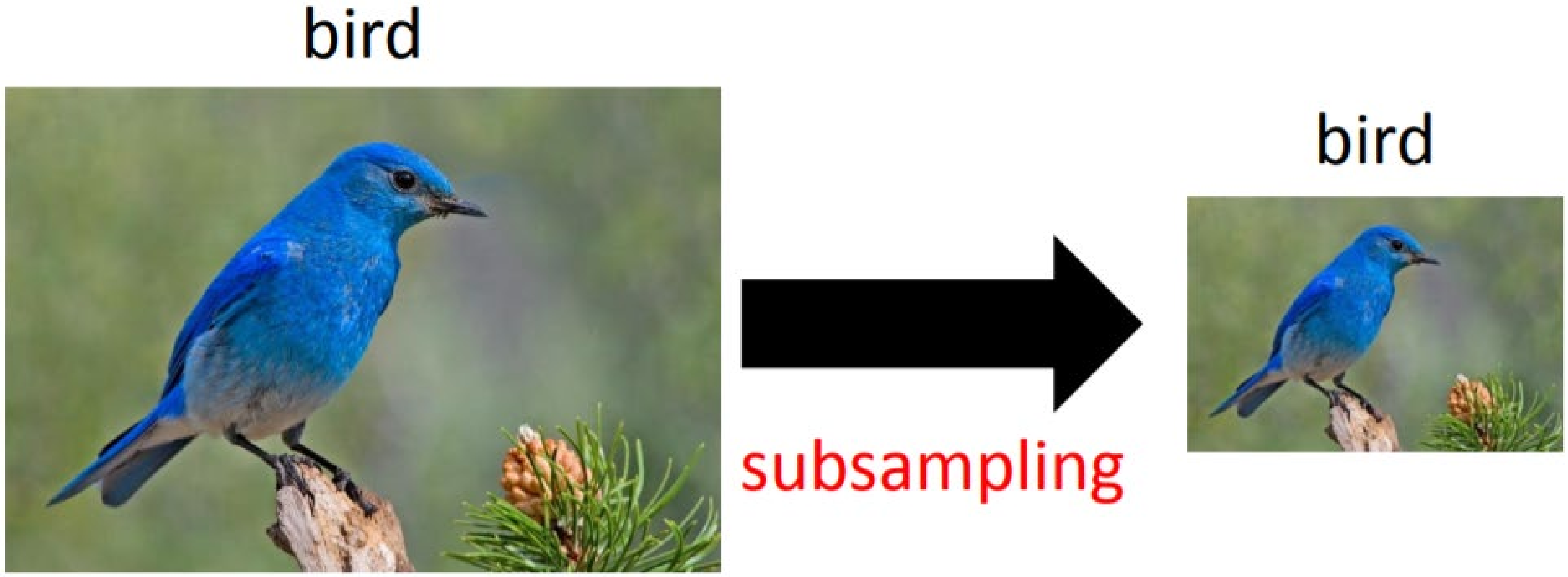
Connecting to small region with less parameters

The same patterns appear in different regions



Reference: 李弘毅 ML Lecture 10 https://youtu.be/FrKWiRv254g

Subsampling the pixels will not change the object

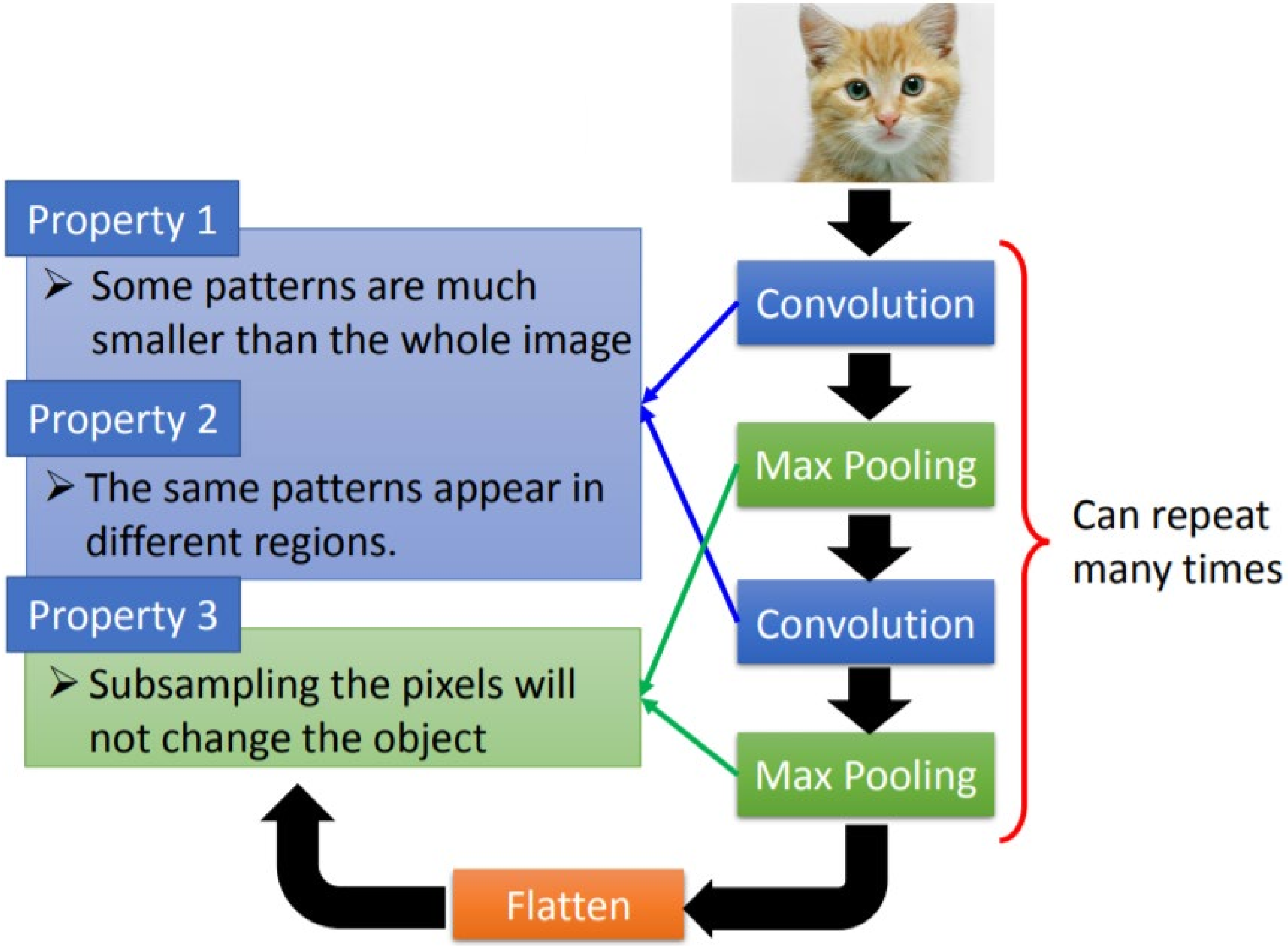


We can subsample the pixels to make image smaller

→ Less parameters for the network to process the image

Reference: 李弘毅 ML Lecture 10 https://youtu.be/FrKWiRv254g

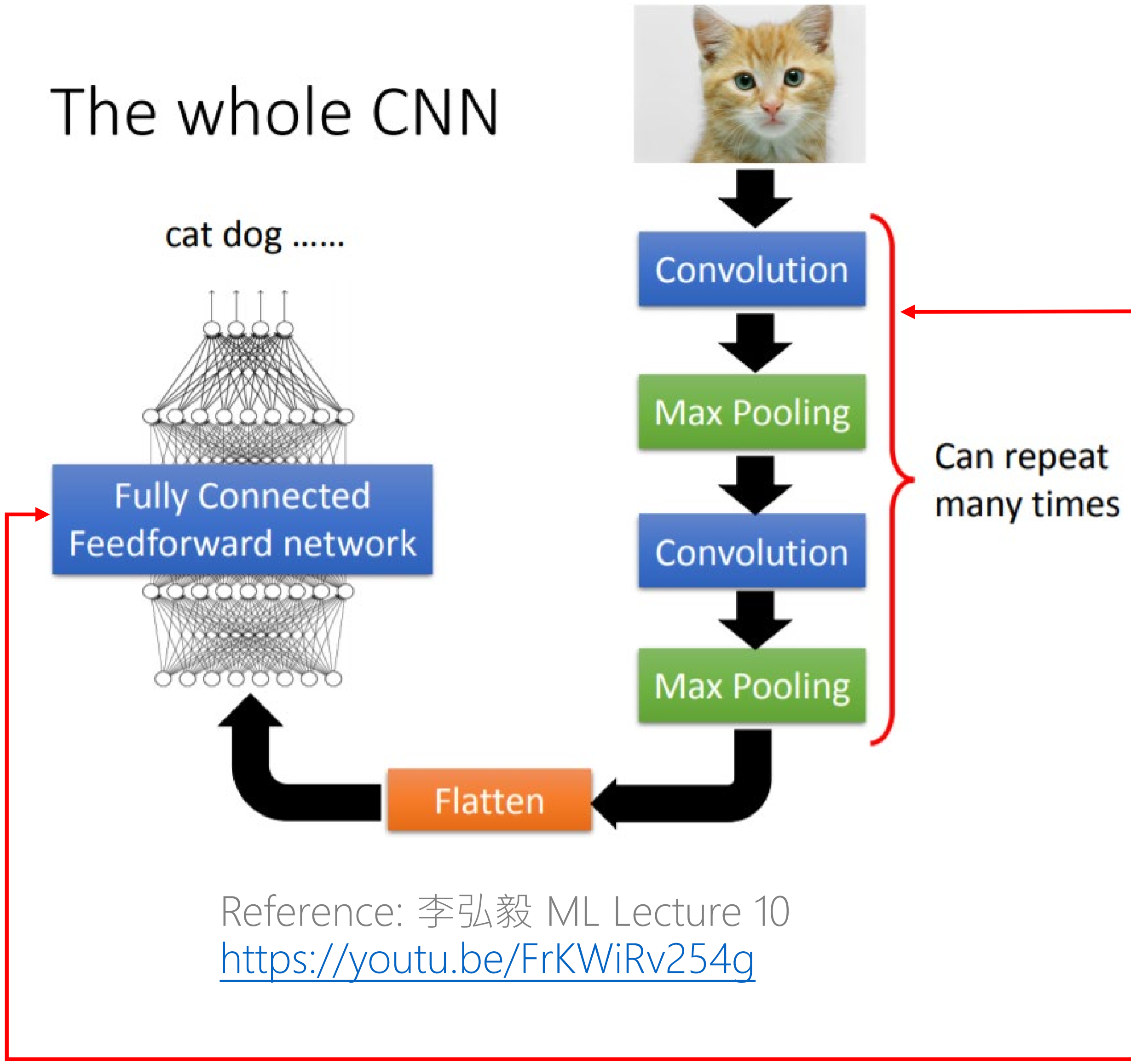
Use convolution and pooling operations to extract important features from input image



Reference: 李弘毅 ML Lecture 10 https://youtu.be/FrKWiRv254g

CNN contains two sections: "features" and "classifier"

The whole CNN



Reference: 李弘毅 ML Lecture 10
https://youtu.be/FrKWiRv254g

```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), paddi
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), paddi
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padd
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padd
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), pad
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=9216, out_features=4096, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=4096, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
```

Load pre-trained CNN for image classification

```
[2] import torchvision
    model = torchvision.models.alexnet(pretrained=True)

    Downloading: "https://download.pytorch.org/models
    100%  233M/233M
```

Torchvision - https://pytorch.org/vision/stable/index.html

ImageNet - http://www.image-net.org/

Image Classification - https://machinelearningmastery.com/applications-of-deep-learning-for-computer-vision/

Deep neural networks for image understanding

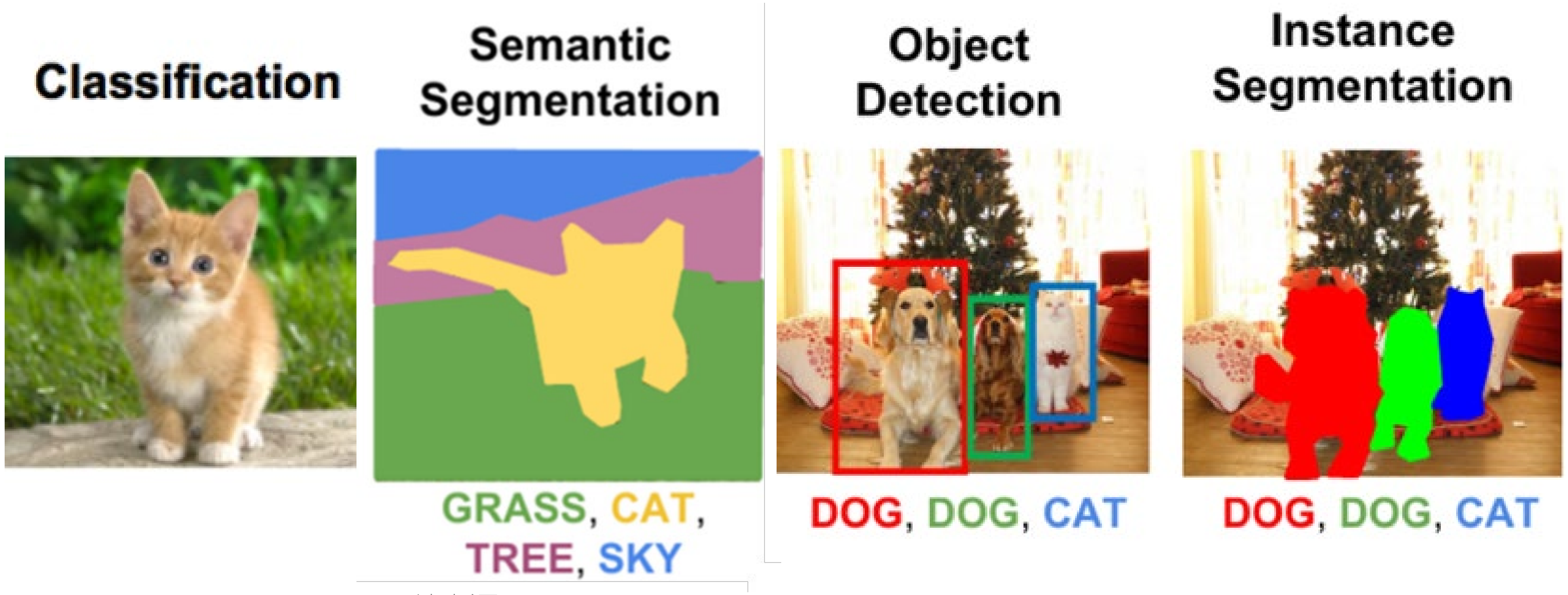
Alex Net
VGG16
Res Net

U Net

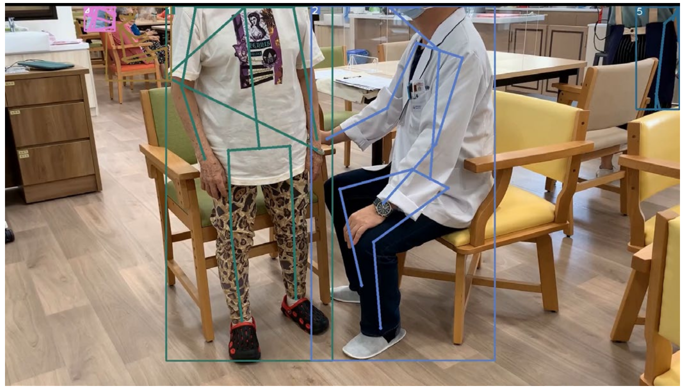
Yolo
Faster RCNN

Mask RCNN

OpenPose
Keypoints RCNN



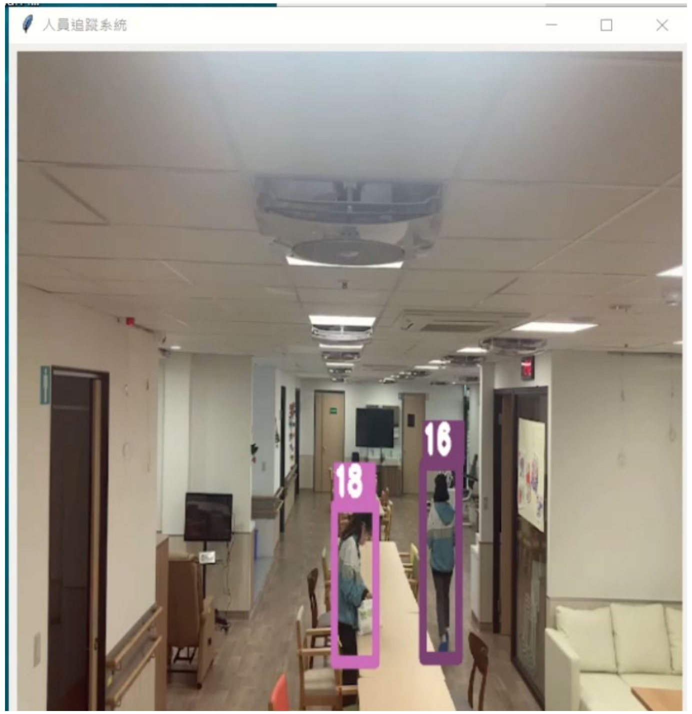
Joint detection



圖片來源: https://kharshit.github.io/blog/2019/08/23/quick-intro-to-instance-segmentation

Image sequence understanding

SORT, ByteTrack DeepSORT, JDE	SlowFast
Object tracking	**Action classification**



Pre-trained NN for CV

Alex Net
VGG16
Res Net

Faster RCNN
Keypoints
RCNN

torchvision

Detectron2

SlowFast

PyTorch
Hub

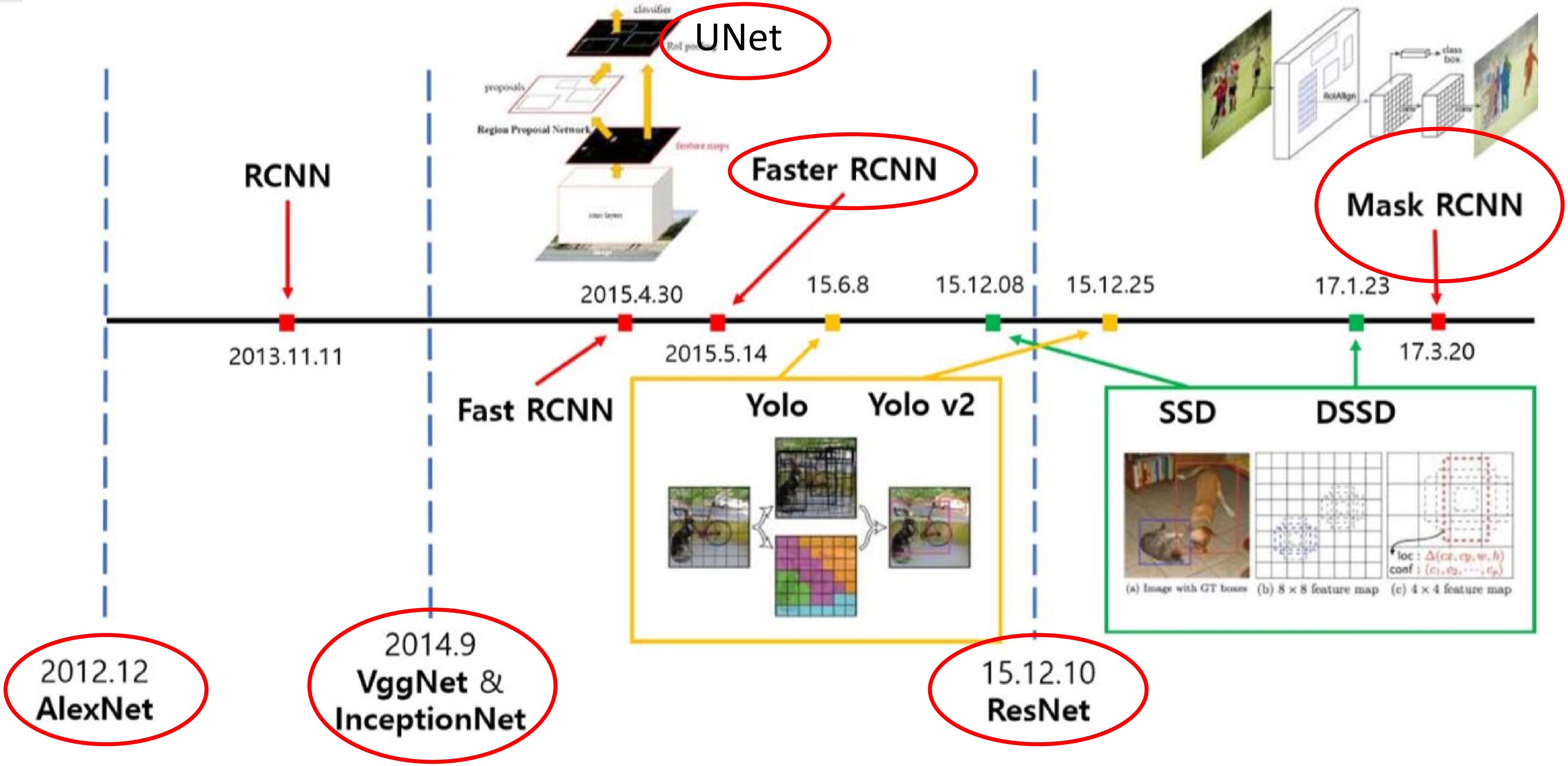
SORT, ByteTrack
DeepSORT, JDE

GitHub

CV
COMPUTER VISION
BEST PRACTICES FOR COMPUTER VISION

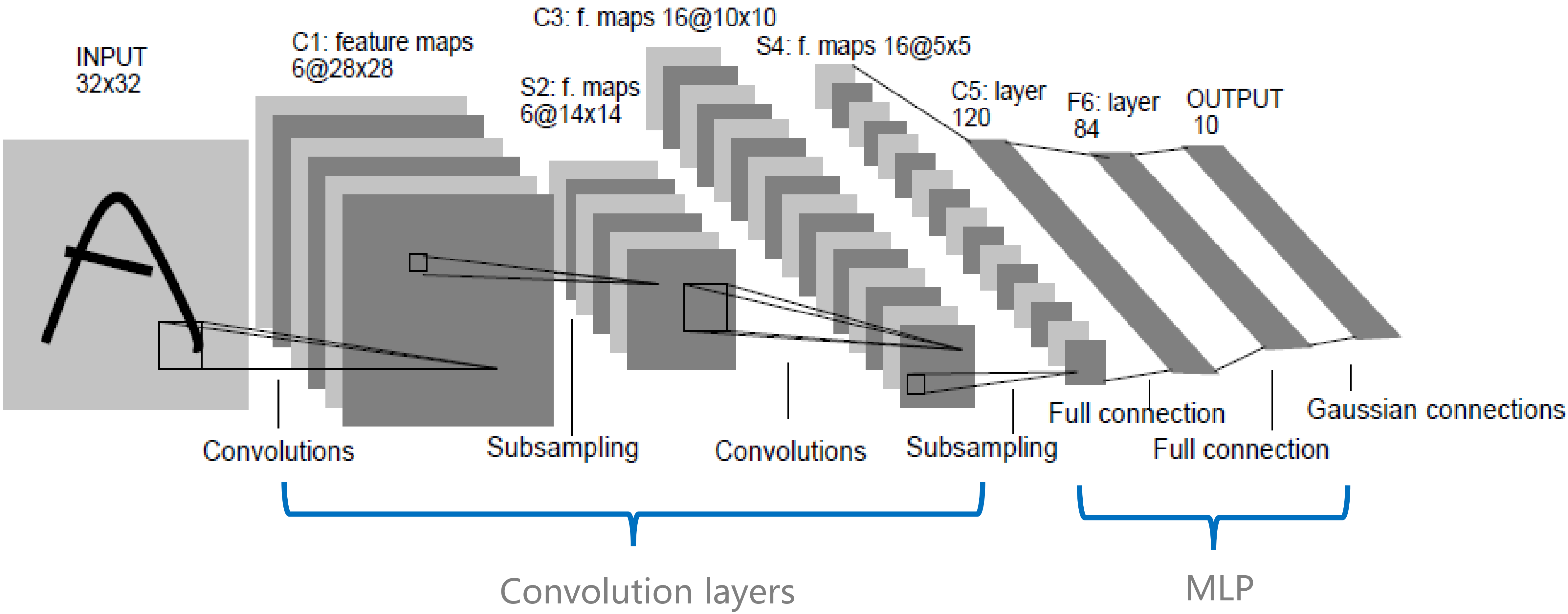
https://github.com/microsoft/computervision-recipes

History of CNN families



圖來源: 李春煌 FasterRCNN講義 https://youtu.be/2i9CcmJp2yI

LeNet



LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

Read image file

In [6]:
```
import cv2
import matplotlib.pyplot as plt
image = cv2.imread(fname)
image = cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
plt.imshow(image)
plt.show()
```

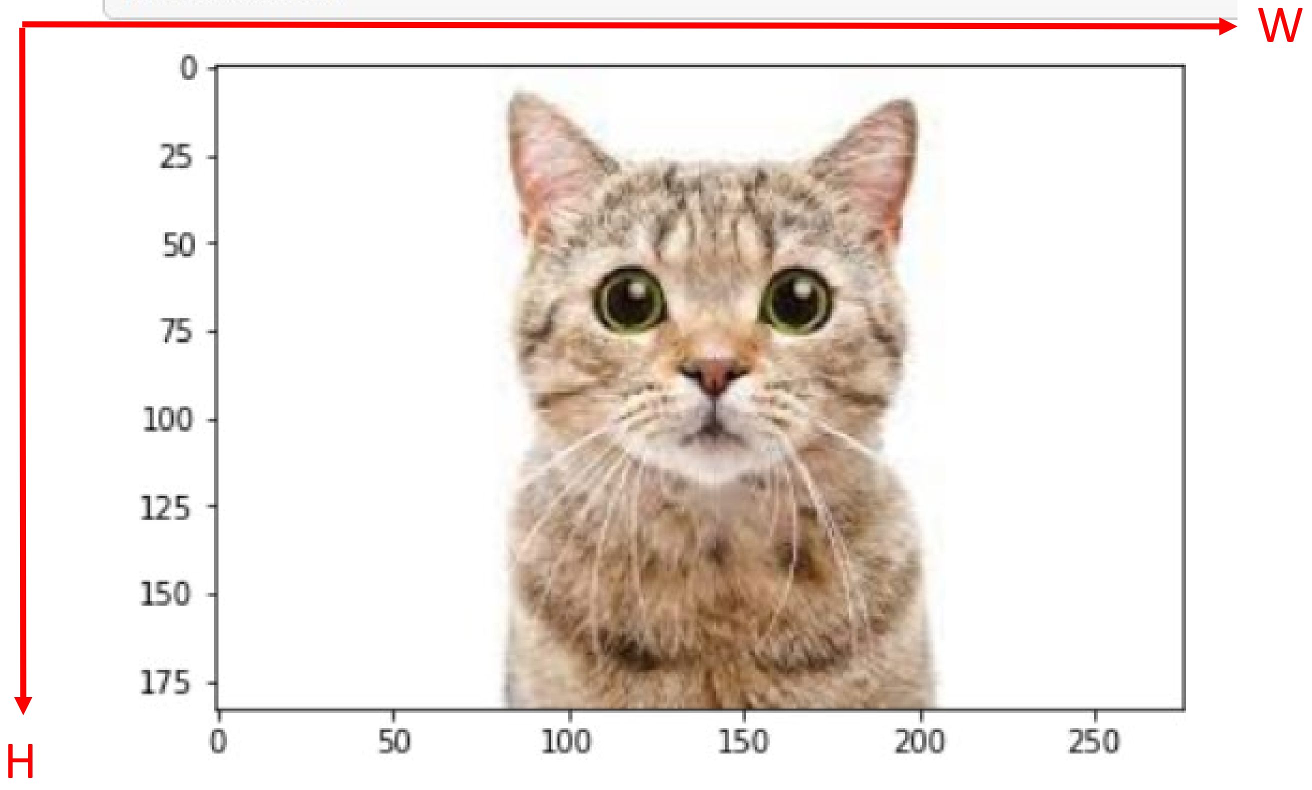


Image pre-processing

- Image width and height – resize, center crop
- Pixel values – Standardized to [0, 1], normalized to N(0, 1)

```
In [7]: from torchvision import transforms
        transformer = transforms.Compose([
             transforms.Resize(224),
             transforms.CenterCrop(224),
             transforms.ToTensor(),
             transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5] )])
```

Feature scaling

```
In [8]: from PIL import Image
        PILImg= Image.fromarray(image.astype('uint8')).convert('RGB')
        PILImg = transformer(PILImg)
        PILImg.shape

Out[8]: torch.Size([3, 224, 224])
```

Input shape = [batch size, input data size]

Input to CNN

```
In [9]: imageTensor = torch.unsqueeze(PILImg, 0)
        imageTensor.shape

Out[9]: torch.Size([1, 3, 224, 224])
```

Input to MLP

2.1. Regression HW.ipynb

```
# test mini-batch
for (batchX, batchY) in loader:
    break
print(batchX.shape, batchY.shape)
```

```
torch.Size([500, 7]) torch.Size([500, 1])
```

4. Classifier.ipynb

torch.size([batch size, input data size])
torch.size([batch size, 1])

1st convolution

```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), paddi
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), paddi
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padd
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padd
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), pad
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=
```

```
In [10]: conv1 = model.features[0]
         print(conv1)
         #InChannel=3(RGB),OutChannel=64, filter size=11, stride=4, padding=2
```

```
Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
```

Filter and convolution operation

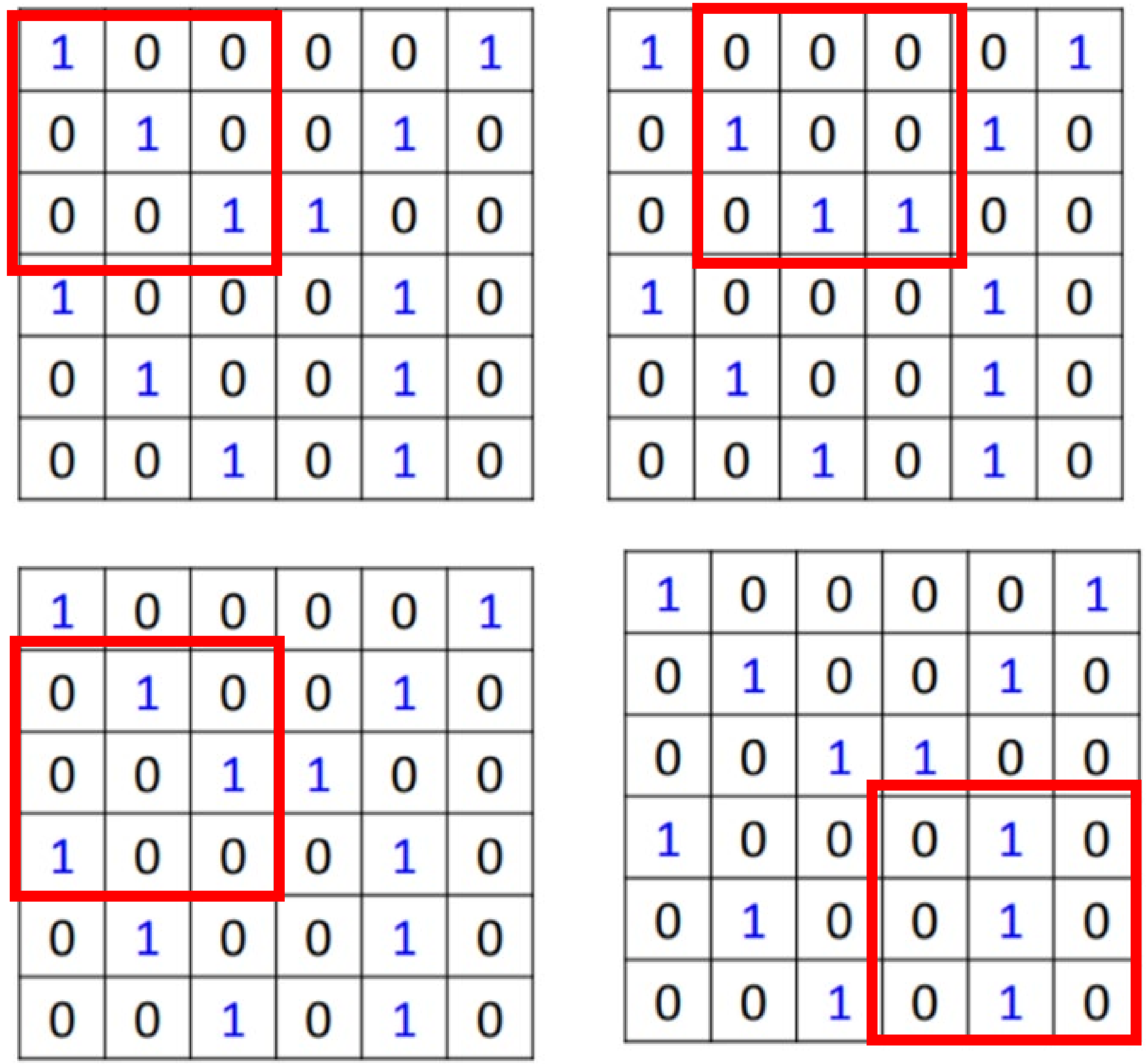
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

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

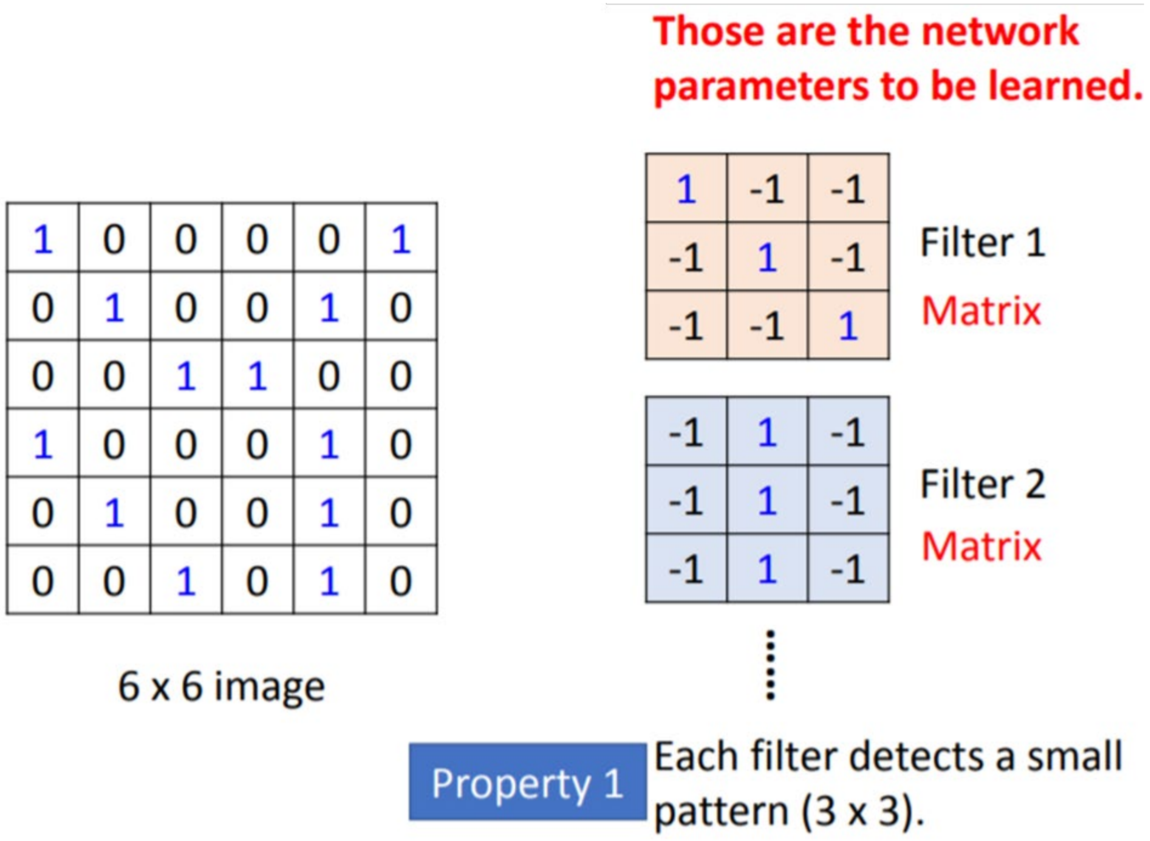
Filter 1

Matrix

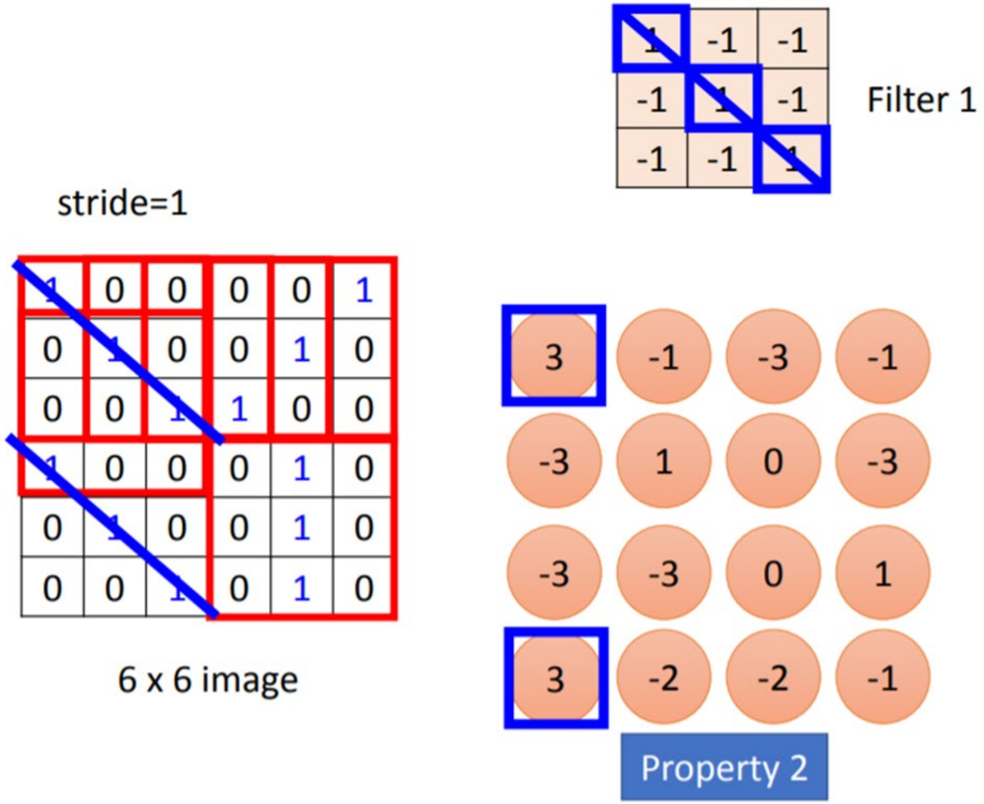


Filter + Convolution to extract features

Filter searches patterns in a small region



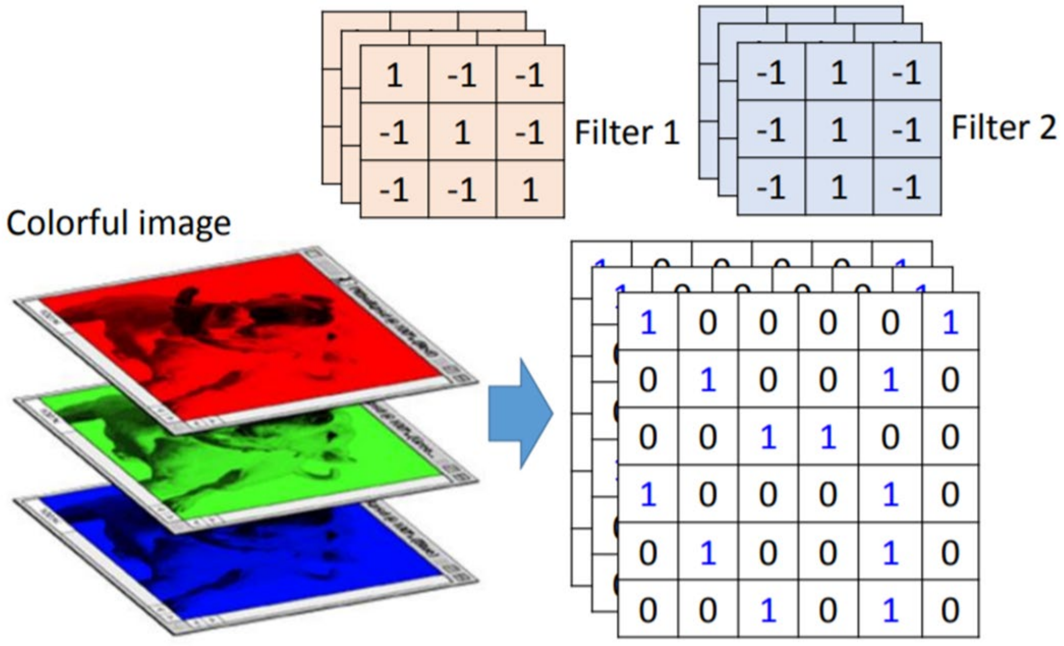
Filter searches a particular pattern in different regions



Reference: 李弘毅 ML Lecture 10 https://youtu.be/FrKWiRv254g

Filter has depth

If input image has 3 channels, then each convolution filter also has 3 channels

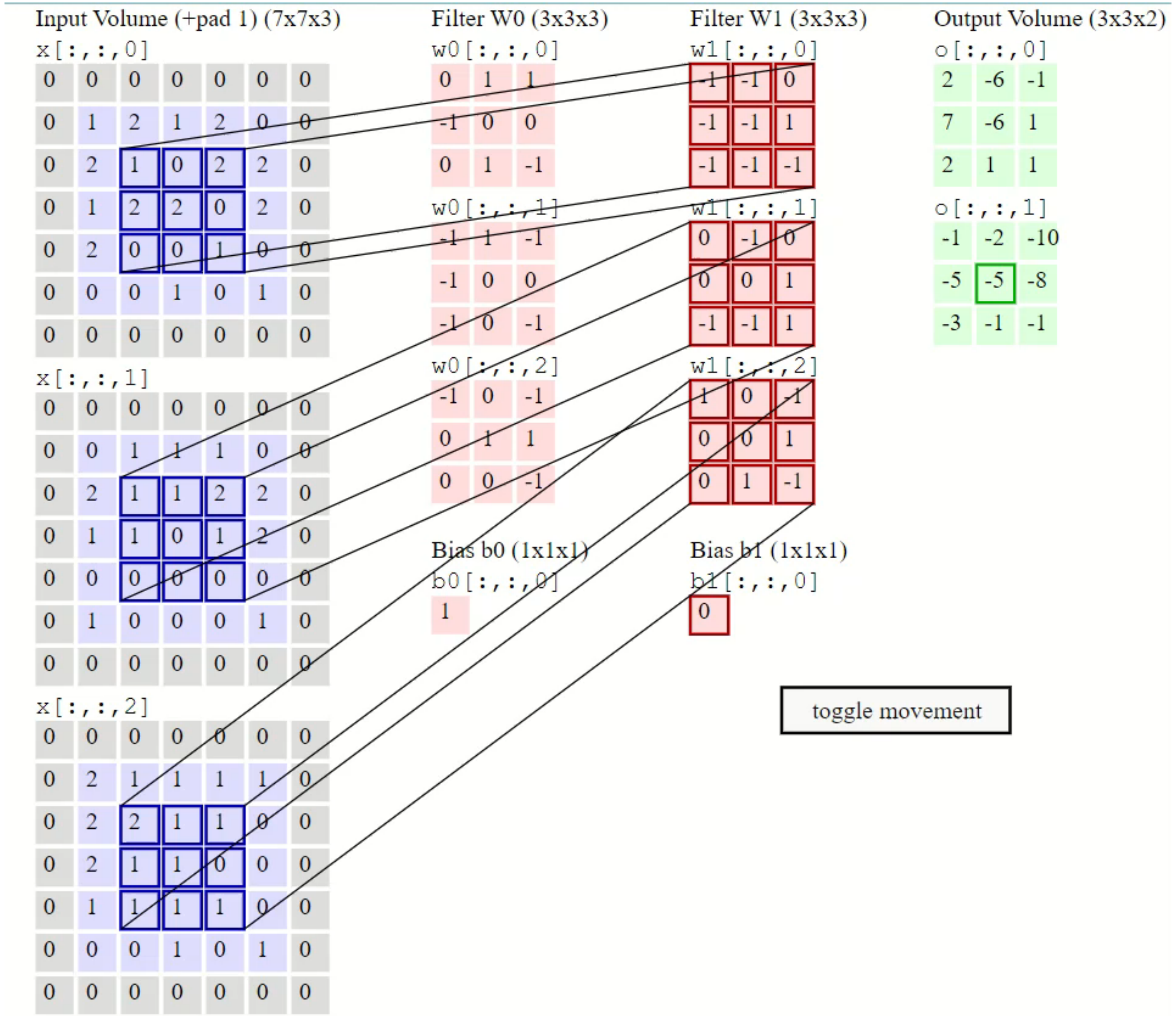



```
[10] conv1 = model.features[0]
     print(conv1)
     #InChannel=3(RGB),OutChannel=64, filter size=11,

     Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4),
```

```
[11] weight1 = conv1.weight.data.cpu().numpy()
     print(weight1.shape)
     #64 filters, depth=3, size =11 by 11

     (64, 3, 11, 11)
```

Reference: 李弘毅 ML Lecture 10 https://youtu.be/FrKWiRv254g

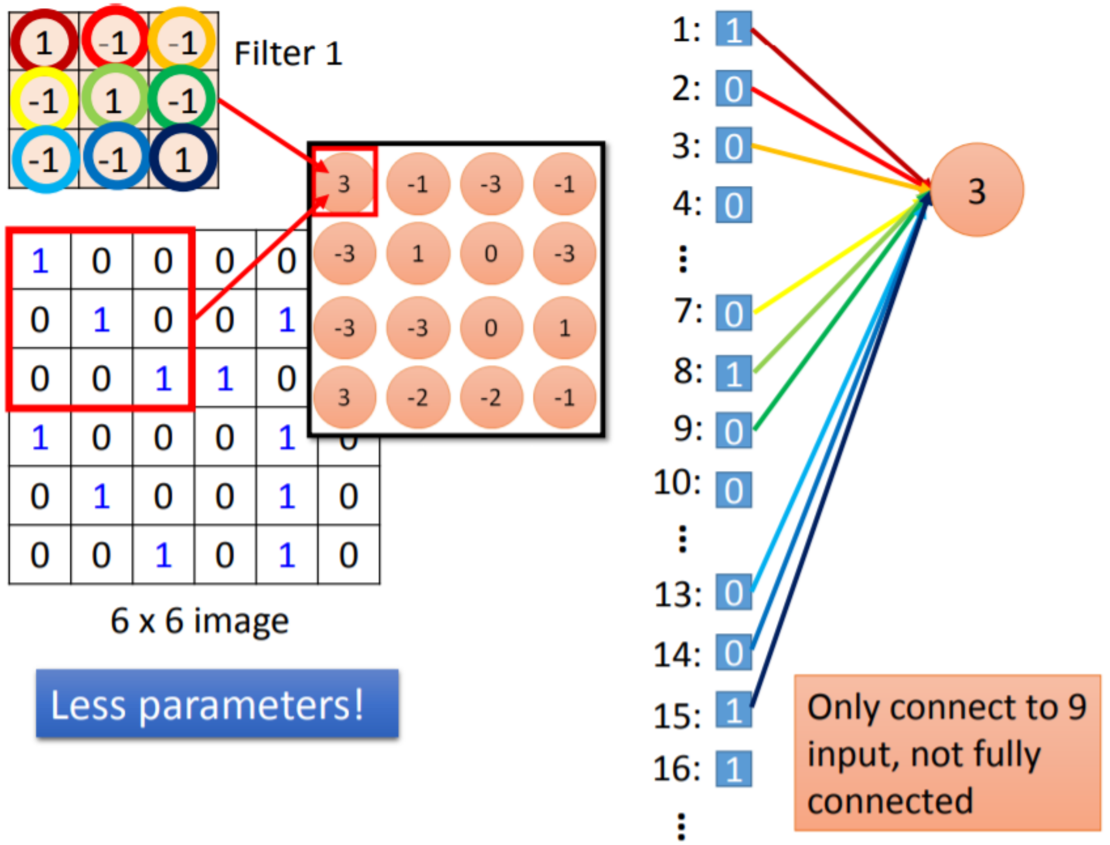
How filter and convolution work



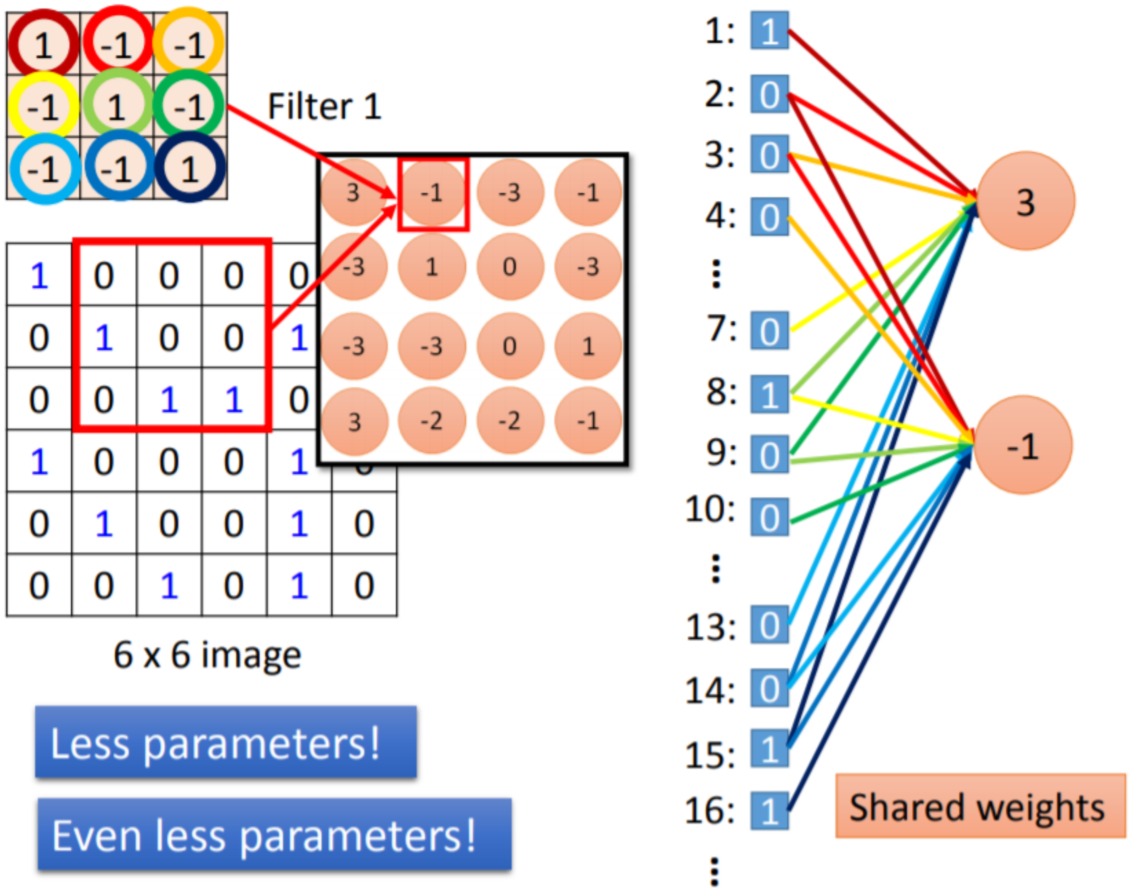
https://cs231n.github.io/convolutional-networks/

Why convolution helps?

Convolution can be represented as partially connected NN, which has less parameters and is less complicated than the fully connected NN.



Partially connected NN with shared weights and hence with even less parameters.

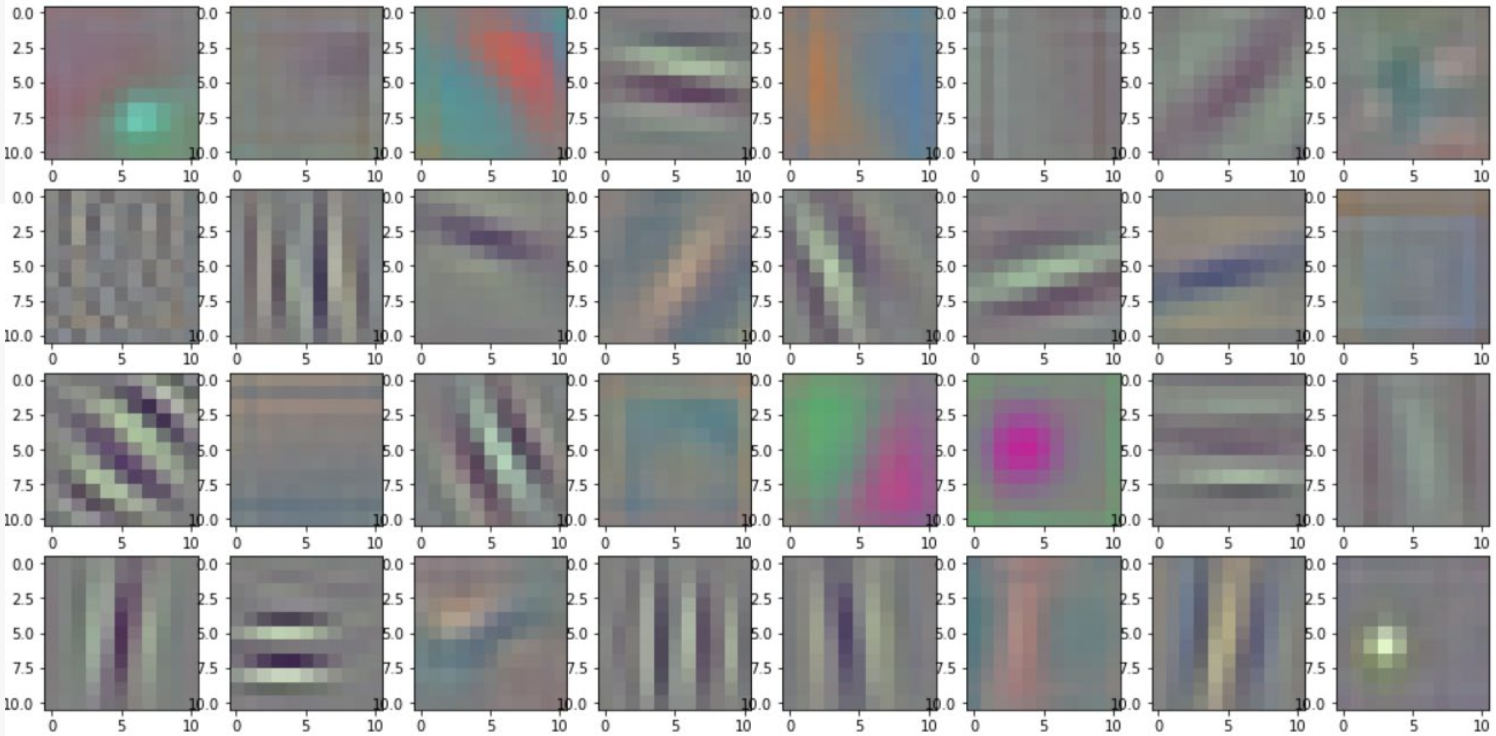


Reference: 李弘毅 ML Lecture 10 https://youtu.be/FrKWiRv254g

Filter weights of 1st convolution layer in AlexNet

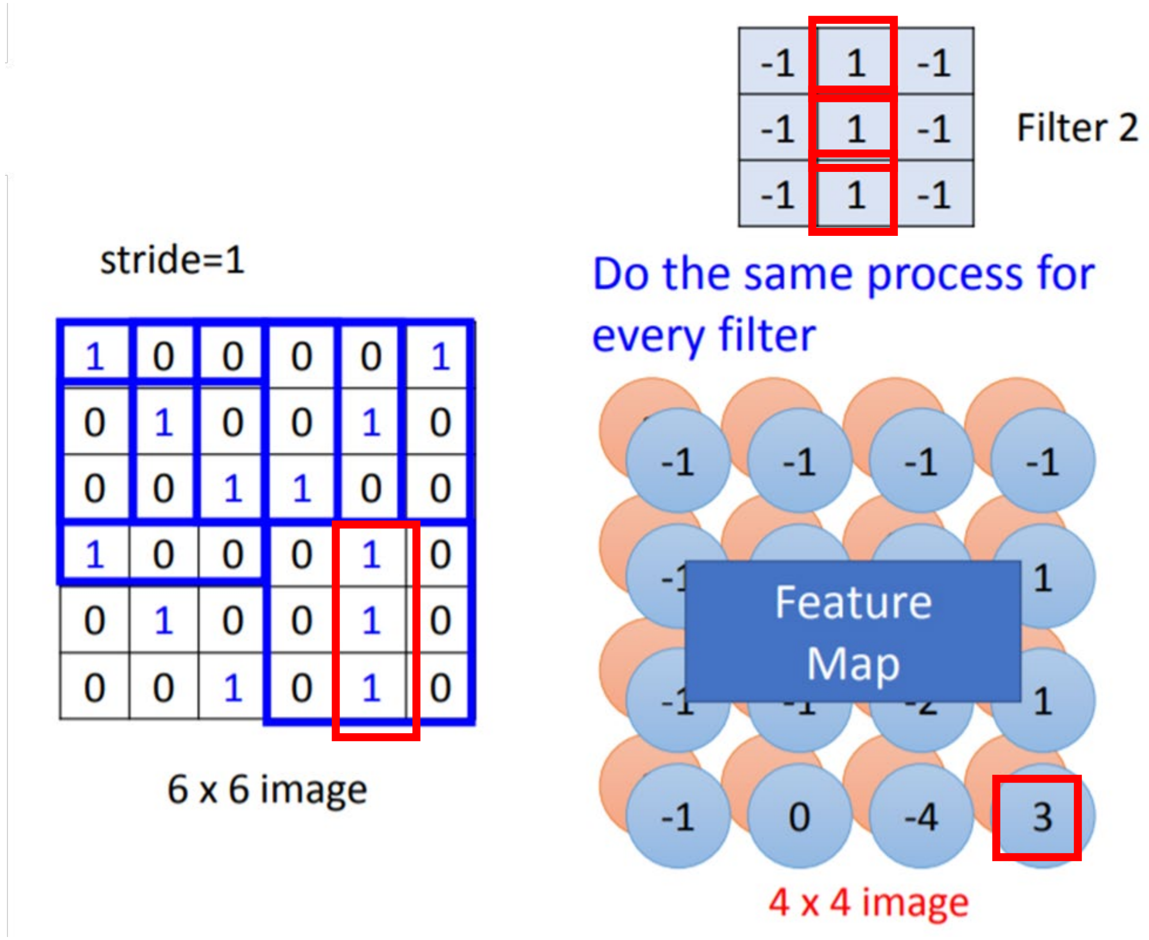
```
[11] weight1 = conv1.weight.data.cpu().numpy()
     print(weight1.shape)
     #64 filters, depth=3, size =11 by 11

     (64, 3, 11, 11)
```

```
[13] # Visualize the first 32 of the filter weights
     import numpy as np
     fig=plt.figure(figsize=(18, 9))
     for i in range(32):
         fig.add_subplot(4, 8, i+1)
         w = weight1[i]
         ImgArray = np.zeros((w.shape[1], w.shape[2], 3))
         ImgArray[:,:,0] = w[0, :, :]
         ImgArray[:,:,1] = w[1, :, :]
         ImgArray[:,:,2] = w[2, :, :]
         ImgArray = ImgArray*0.5+0.5   # convert[-1, 1] to [0, 1]
         plt.imshow(ImgArray)
     plt.show()
```



Feature maps

Each filter searches a small region and summarizes how the specified pattern appears in different regions in a feature map




```
In [10]: conv1 = model.features[0]
         print(conv1)
         #InChannel=3(RGB),OutChannel=64, filter size

         Conv2d(3, 64, kernel_size=(11, 11), stride=(

In [11]: weight1 = conv1.weight.data.cpu().numpy()
         print(weight1.shape)
         #64 filters, depth=3, size =11 by 11

         (64, 3, 11, 11)

In [12]: conv1_out = conv1(imageTensor.to(device))
         conv1_out.shape
         #output image (feature map) has 64 channels

Out[12]: torch.Size([1, 64, 55, 55])
```

Width and height of feature maps

```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
```

```
imageTensor = torch.unsqueeze(PILImg, 0)
imageTensor.shape
```

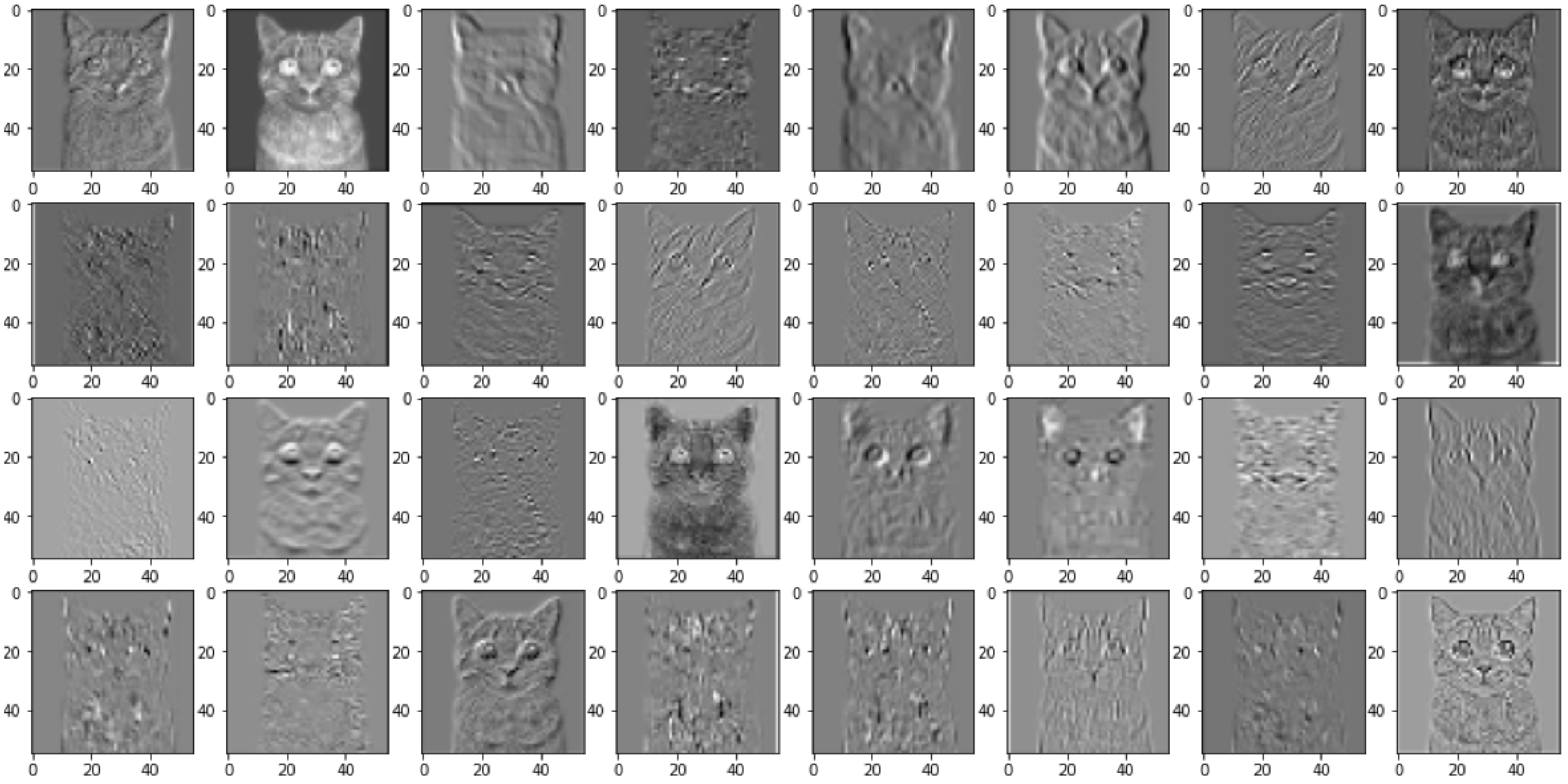
```
torch.Size([1, 3, 224, 224])
```

$$H_{out} = \frac{H_{in} + 2 \times padding - kernel\ size}{Stride} + 1$$

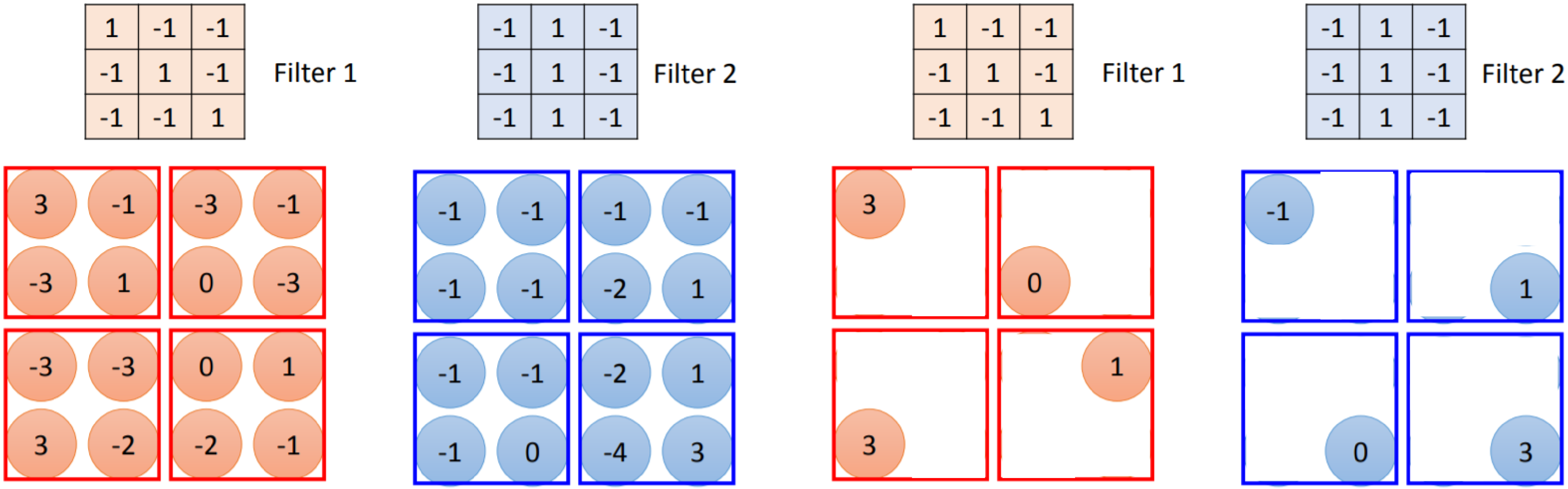
$$\frac{224 + 2 \times 2 - 11}{4} + 1 = 55.25$$

```
In [12]: conv1_out = conv1(imageTensor.to(device))
         conv1_out.shape
         #output image (feature map) has 64 channels

Out[12]: torch.Size([1, 64, 55, 55])
```

First 32 channels of the output feature map (shape = 55x55x64)



Max pooling



Reference: 李弘毅 ML Lecture 10 https://youtu.be/FrKWiRv254g

Apply max pooling to the feature map from 1st convolution

features[1, 2]

[14]:
```python
conv1_pooling = model.features[1:3]
conv1_out1 = conv1_pooling(conv1_out)
print(conv1_out1.shape)
imgArray=conv1_out1[0].data.cpu().numpy()
fig=plt.figure(figsize=(18, 9))
for i in range(32): #visualize the first 32 channe
  fig.add_subplot(4, 8, i+1)
  plt.imshow(imgArray[i], cmap='gray')
plt.show()
```

```
torch.Size([1, 64, 27, 27])
```

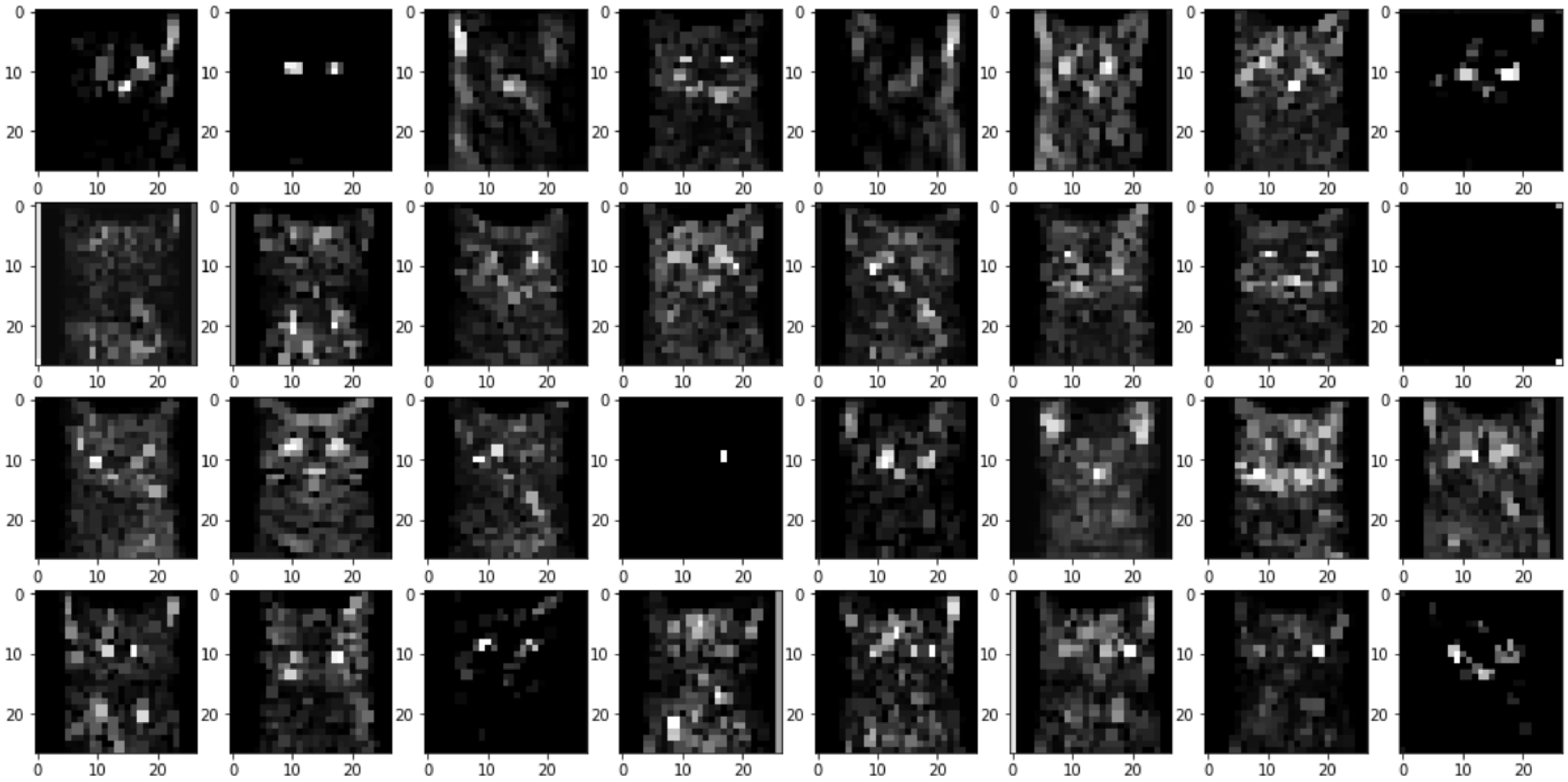
$$\frac{55 + 2 \times 2 - 3}{2} + 1 = 27$$

$$H_{out} = \frac{H_{in} + 2 \times padding - kernel\ size}{Stride} + 1$$

```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padd
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padd
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), pad
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), pad
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), pa
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation
  )
```

PYTORCH

First 32 channels of the output feature map from max pooling, shape = 27x27x64



2^{nd} convolution

192 filters, each has 64 channels, are applied to the input feature map (with 64 channels)

```
[15]: conv2 = model.features[3]
      conv2_out = conv2(conv1_out1)
      print(conv2_out.shape)
      imgArray=conv2_out[0].data.cpu().numpy()
      fig=plt.figure(figsize=(18, 9))
      for i in range(32): #visualize the first 32 channels
        fig.add_subplot(4, 8, i+1)
        plt.imshow(imgArray[i], cmap='gray')
      plt.show()

      torch.Size([1, 192, 27, 27])
```

```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padd
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padd
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), pad
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), pad
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), pa
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation
  )
```

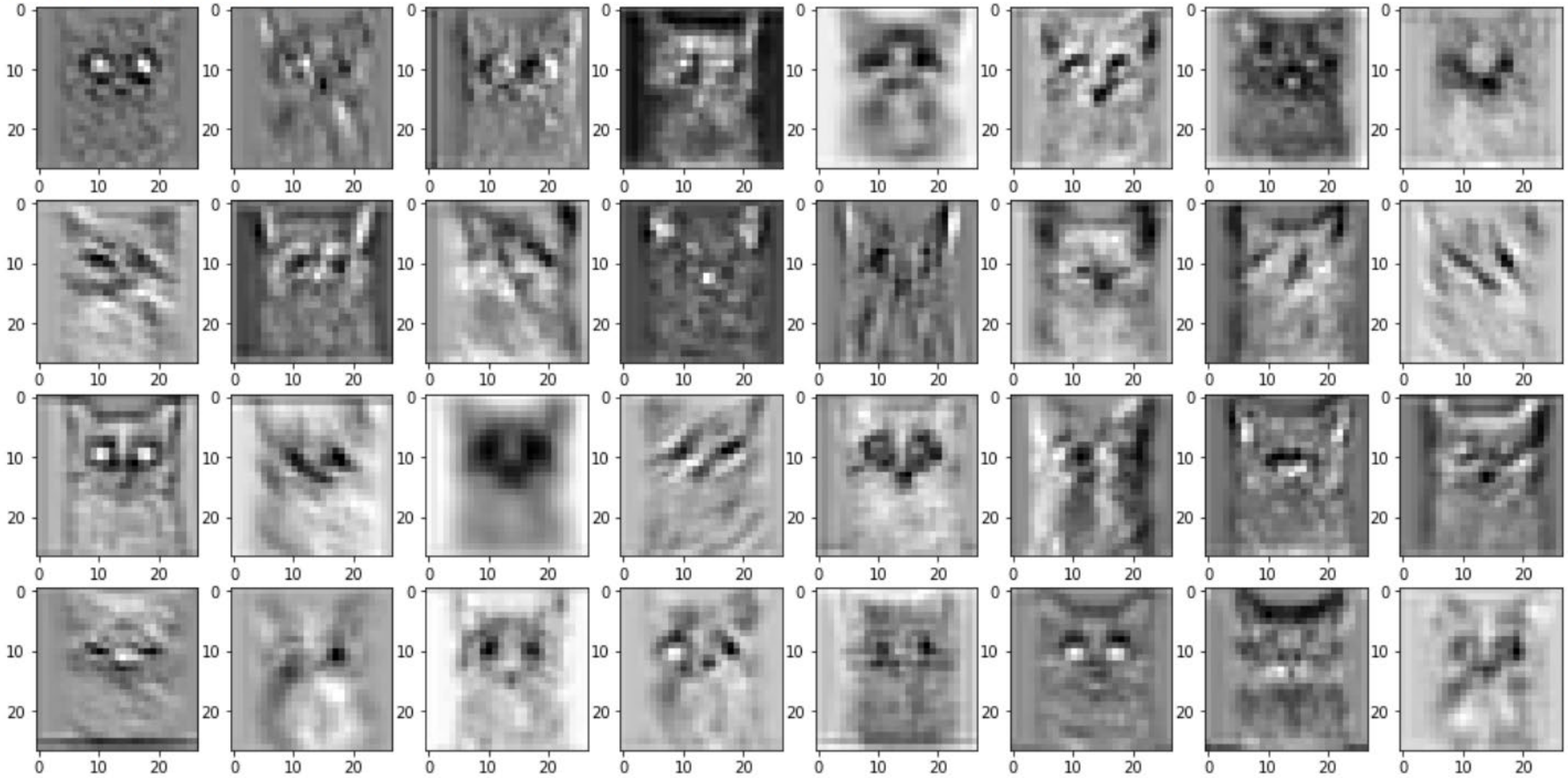
After convolution, the output feature map has 192 channels

$$\frac{27 + 2 \times 2 - 5}{1} + 1 = 27$$

$$H_{out} = \frac{H_{in} + 2 \times padding - kernel\ size}{Stride} + 1$$

Feature map after 2nd convolution

```
torch.Size([1, 192, 27, 27])
```



Apply max pooling to the feature map from 2^{nd} convolution

features[4, 5]

[16]:
```
conv2_pooling = model.features[4:6]
conv2_out1 = conv2_pooling(conv2_out)
print(conv2_out1.shape)
imgArray=conv2_out1[0].data.cpu().numpy()
fig=plt.figure(figsize=(18, 9))
for i in range(32): #visualize the first 32 channels
  fig.add_subplot(4, 8, i+1)
  plt.imshow(imgArray[i], cmap='gray')
plt.show()
```

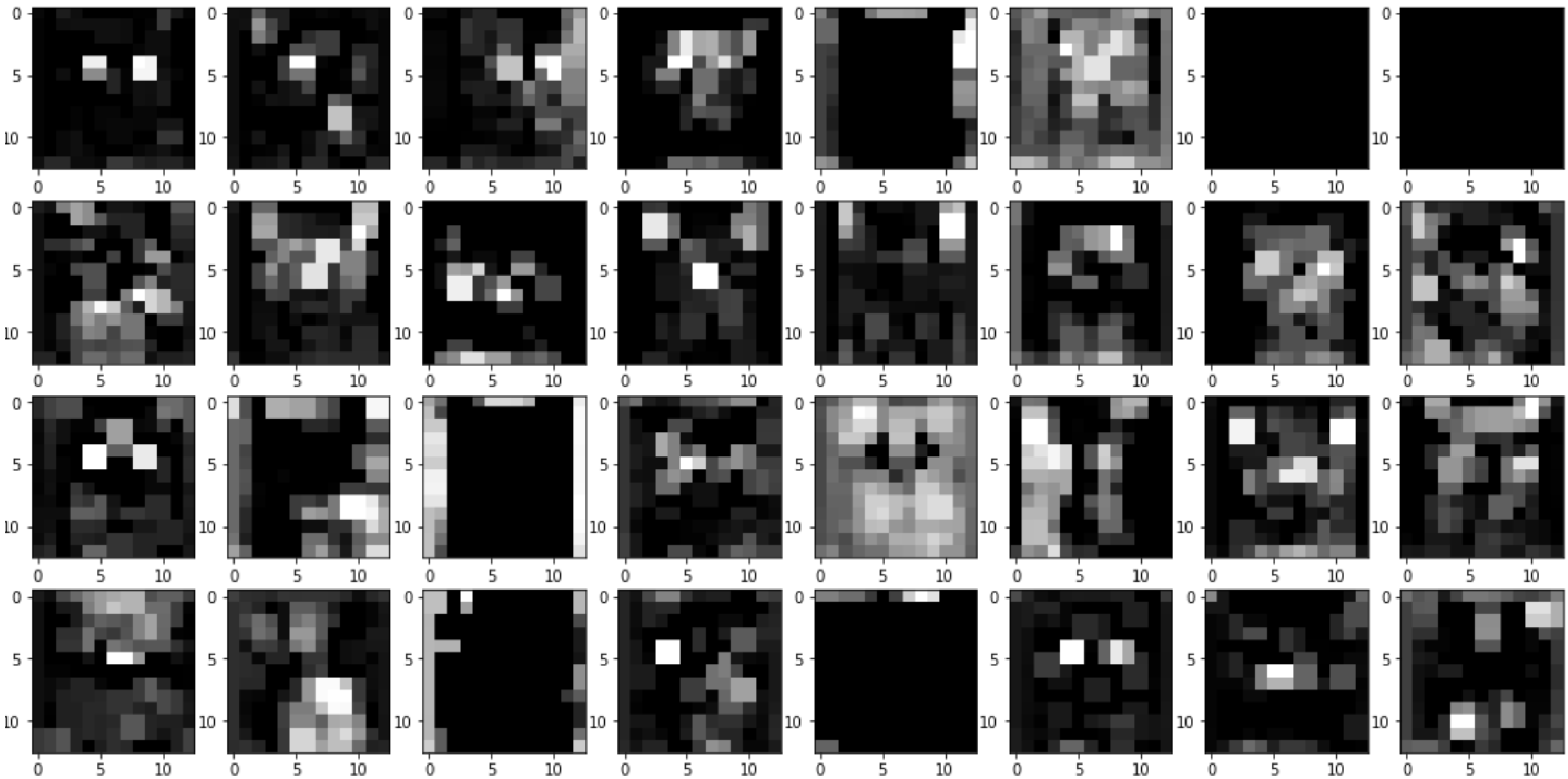
```
torch.Size([1, 192, 13, 13])
```

$$\frac{27 + 2 \times 0 - 3}{2} + 1 = 13$$

$$H_{out} = \frac{H_{in} + 2 \times padding - kernel\ size}{Stride} + 1$$

```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padd
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padd
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), pad
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), pad
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), pa
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation
  )
```

Feature map after 2nd convolution and max pooling



3rd convolution

394 filters, each has 192 channels, are applied to the input feature map (with 192 channels)

```
[17]: conv3 = model.features[6]
      conv3_out = conv3(conv2_out1)
      print(conv3_out.shape)
      imgArray=conv3_out[0].data.cpu().numpy()
      fig=plt.figure(figsize=(18, 9))
      for i in range(32): #visualize the first 32 channels
        fig.add_subplot(4, 8, i+1)
        plt.imshow(imgArray[i], cmap='gray')
      plt.show()

      torch.Size([1, 384, 13, 13])
```

```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padd
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padd
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), pad
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), pad
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), pa
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation
  )
```

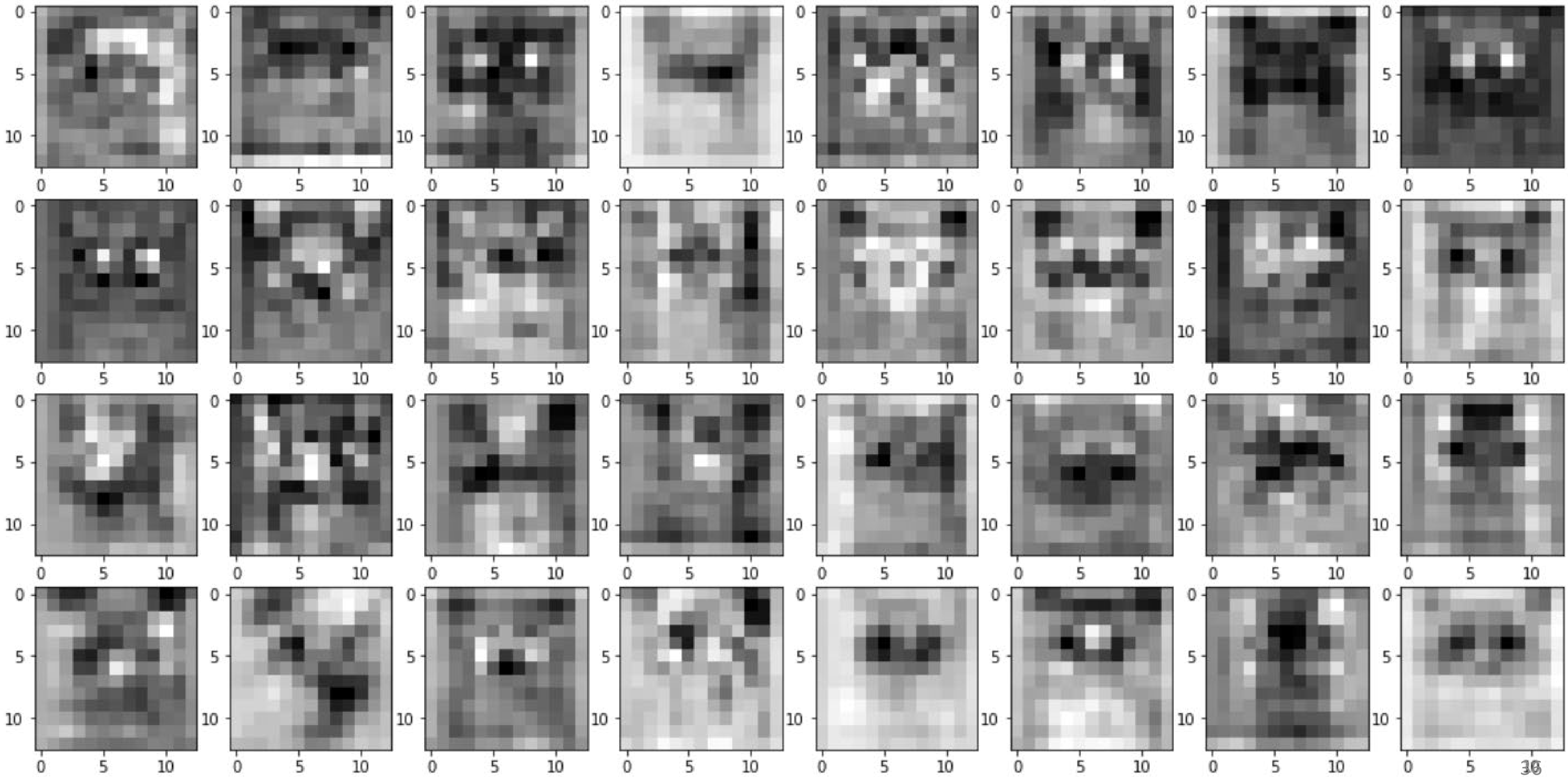
After convolution, the output feature map has 394 channels

$$\frac{13 + 2 \times 1 - 3}{1} + 1 = 13$$

$$H_{out} = \frac{H_{in} + 2 \times padding - kernel\ size}{Stride} + 1$$

Feature map after 3rd convolution

```
torch.Size([1, 384, 13, 13])
```



Apply max pooling to feature map from 3rd convolution

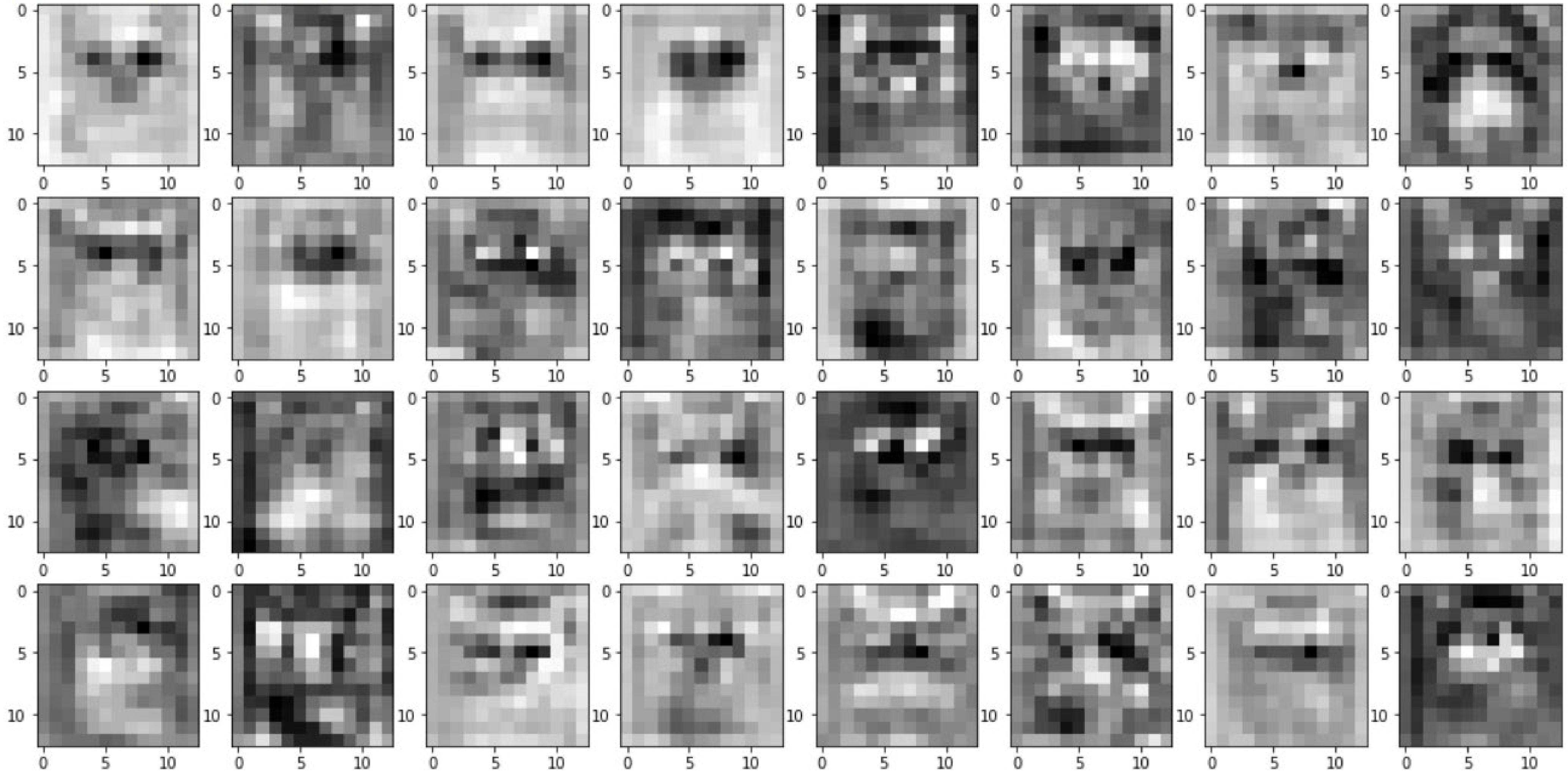
features[7, 8]

```
[18]: conv3_pooling = model.features[7:9]
      conv3_out1 = conv3_pooling(conv3_out)
      print(conv3_out1.shape)
      imgArray=conv3_out1[0].data.cpu().numpy()
      fig=plt.figure(figsize=(18, 9))
      for i in range(32): #visualize the first 32 channels
        fig.add_subplot(4, 8, i+1)
        plt.imshow(imgArray[i], cmap='gray')
      plt.show()

      torch.Size([1, 256, 13, 13])
```

```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padd
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padd
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), pad
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), pad
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), pa
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation
  )
```

Feature map after 3rd convolution and max pooling

```
torch.Size([1, 256, 13, 13])
```



Flatten

```
[19]: WholeConvLayers = model.features
      out1 = WholeConvLayers(imageTensor.to(device))
      print(out1.shape)

      AvgPoolLayer = model.avgpool
      out2 = AvgPoolLayer(out1)
      print(out2.shape)
```

```
torch.Size([1, 256, 6, 6])
torch.Size([1, 256, 6, 6])
```

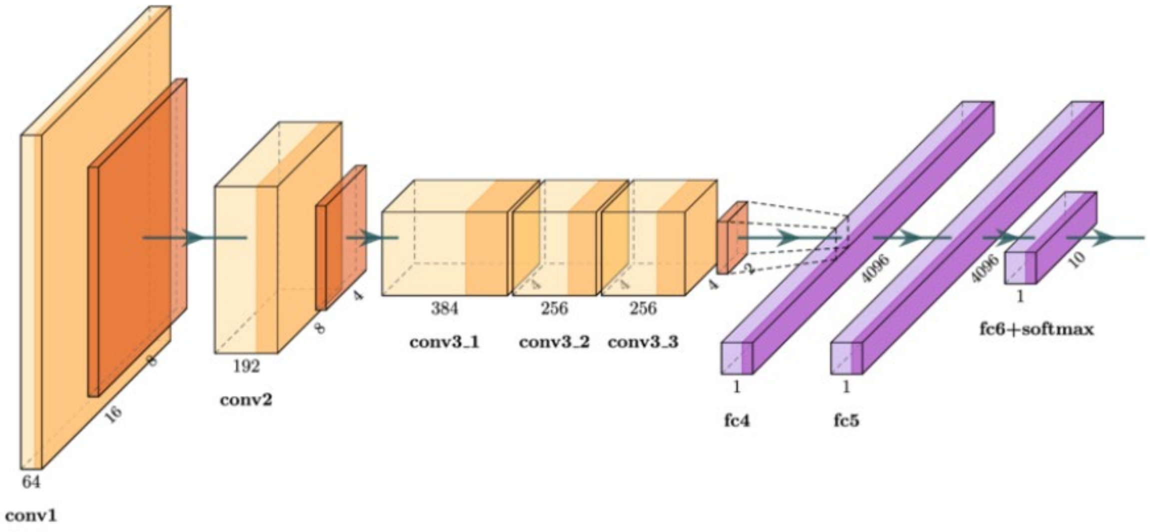
After last convolution and max pooling, the output feature map has 256 channels

$$256 \times 6 \times 6 = 9216$$

```
(features): Sequential(
  (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
  (1): ReLU(inplace=True)
  (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (4): ReLU(inplace=True)
  (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (7): ReLU(inplace=True)
  (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (9): ReLU(inplace=True)
  (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (11): ReLU(inplace=True)
  (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
)
(avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
(classifier): Sequential(
  (0): Dropout(p=0.5, inplace=False)
  (1): Linear(in_features=9216, out_features=4096, bias=True)
  (2): ReLU(inplace=True)
  (3): Dropout(p=0.5, inplace=False)
  (4): Linear(in_features=4096, out_features=4096, bias=True)
  (5): ReLU(inplace=True)
  (6): Linear(in_features=4096, out_features=1000, bias=True)
)
```

Practice – Draw AlexNet

```
(features): Sequential(
  (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
  (1): ReLU(inplace=True)
  (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (4): ReLU(inplace=True)
  (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (7): ReLU(inplace=True)
  (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (9): ReLU(inplace=True)
  (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (11): ReLU(inplace=True)
  (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
)
(avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
(classifier): Sequential(
  (0): Dropout(p=0.5, inplace=False)
  (1): Linear(in_features=9216, out_features=4096, bias=True)
  (2): ReLU(inplace=True)
  (3): Dropout(p=0.5, inplace=False)
  (4): Linear(in_features=4096, out_features=4096, bias=True)
  (5): ReLU(inplace=True)
  (6): Linear(in_features=4096, out_features=1000, bias=True)
)
```



https://www.researchgate.net/figure/AlexNet-architecture-used-as-the-baseline-model-for-the-analysis-of-results-on-the_fig5_339756908