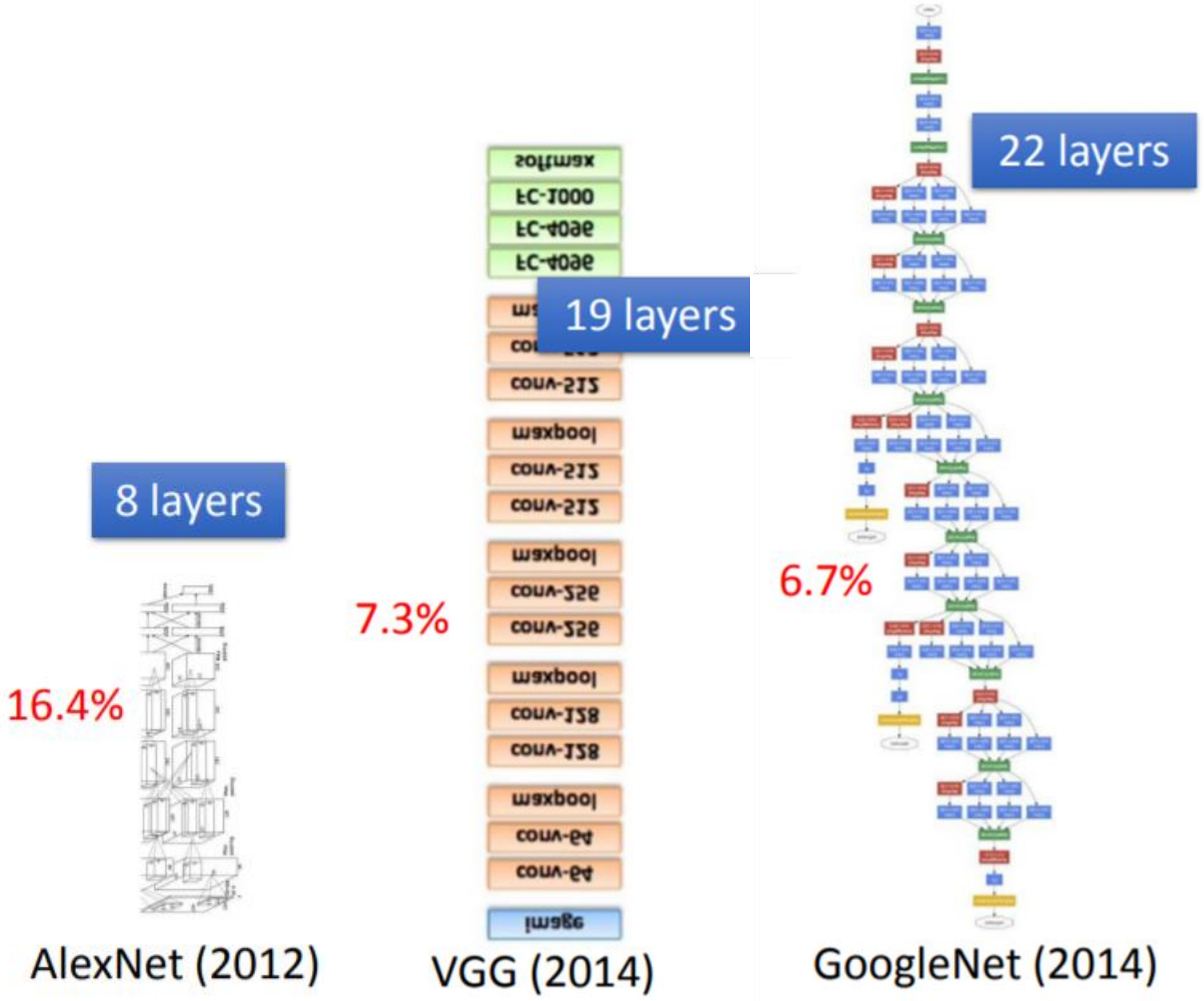
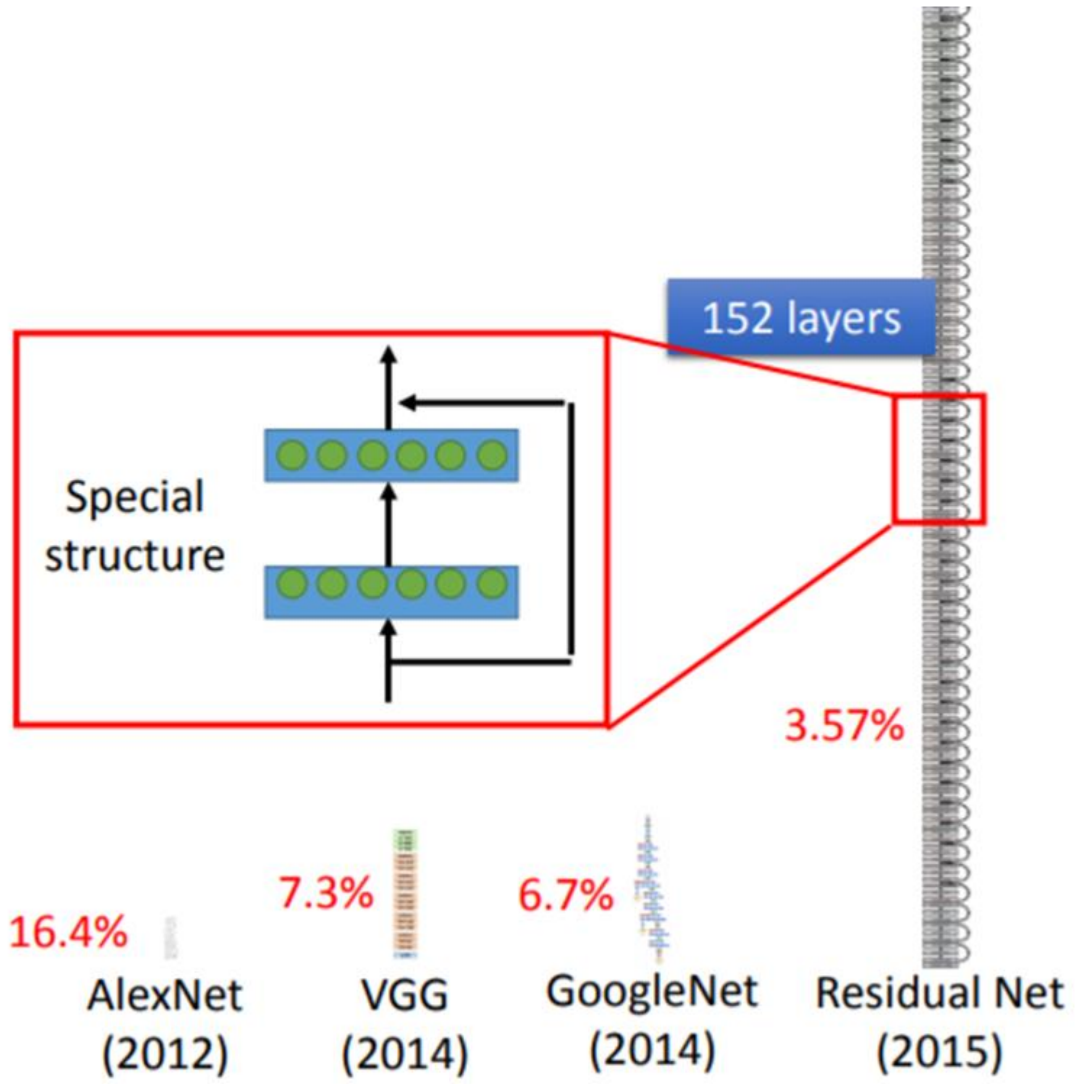
ResNet

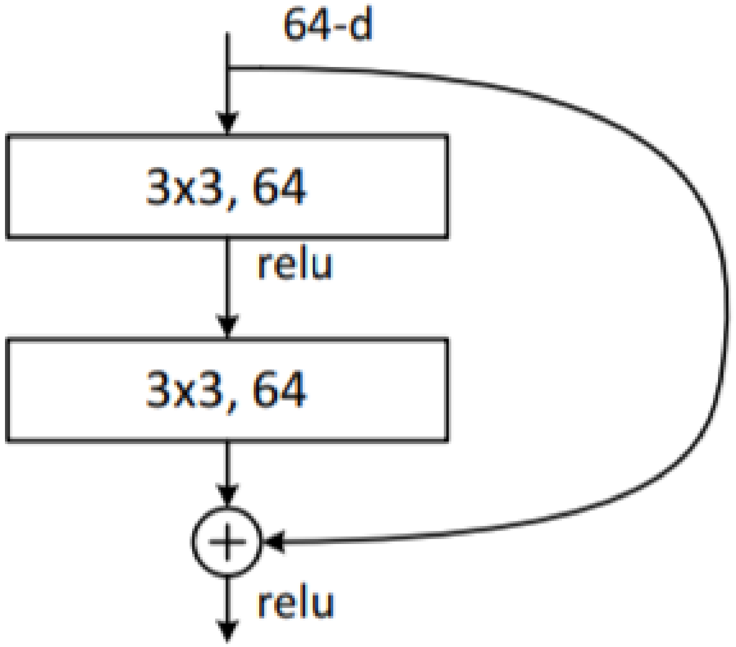
Going deeper and deeper…



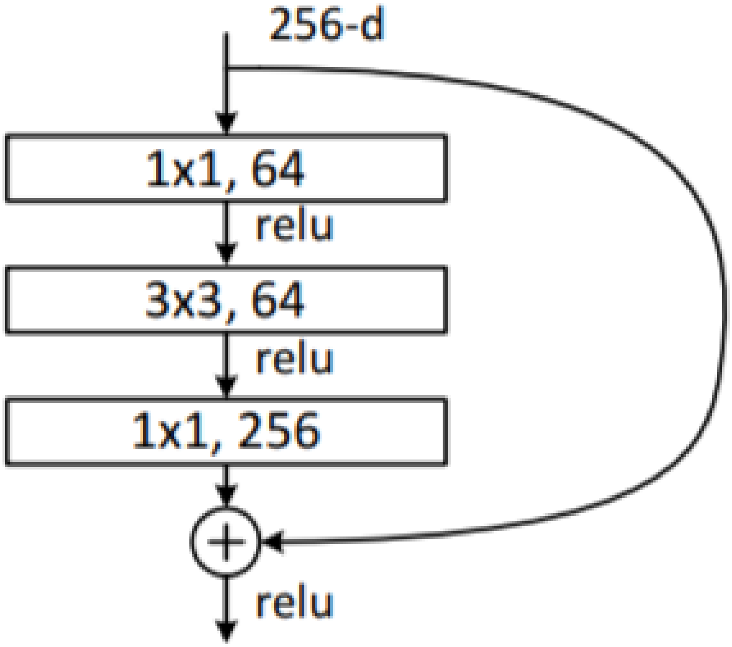


Reference: 李弘毅 ML Lecture 6 https://youtu.be/Dr-WRlEFefw

ResNet



Basic block



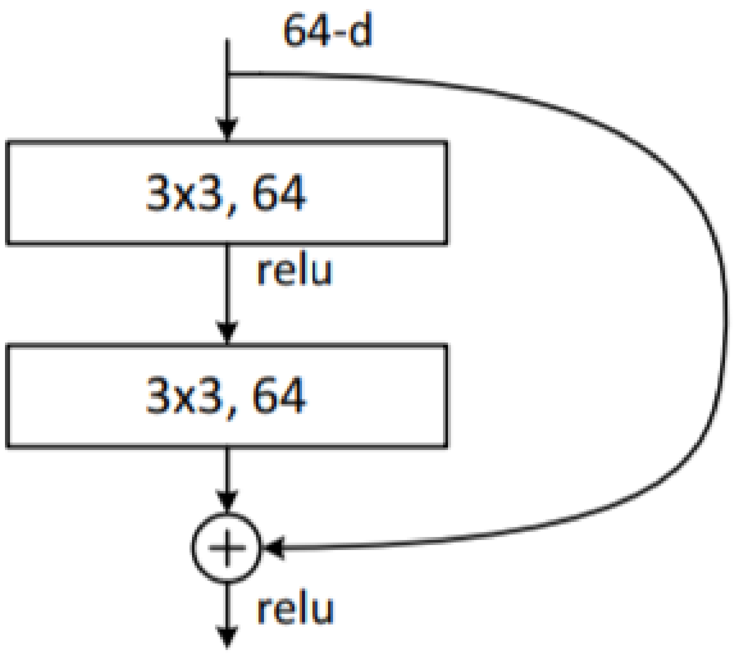
Bottleneck block

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

Practice

- Run “7.4. Build ResNet from scratch.ipynb"

Basic loop




```python
class BasicBlock(nn.Module):
  expansion = 1
  def __init__(self, inplanes, planes, stride=1, downsample=None,):
    super(BasicBlock, self).__init__()
    self.conv1=conv3x3(inplanes,planes,stride)
    self.bn1=nn.BatchNorm2d(planes)
    self.relu=nn.ReLU(inplace=True)
    self.conv2=conv3x3(planes,planes)
    self.bn2=nn.BatchNorm2d(planes)
    self.downsample=downsample
    self.stride=stride

    if(stride!=1 or inplanes!=planes*self.expansion):
      self.downsample=nn.Sequential(
        nn.Conv2d(inplanes,planes*self.expansion,kernel_size=1,stride
        nn.BatchNorm2d(planes*self.expansion),
      )

  def forward(self, x):
    residual = x
    out = self.conv1(x)
    out = self.bn1(out)
    out = self.relu(out)
    out = self.conv2(out)
    out = self.bn2(out)

    # Downsample:feature Map size/2 || Channel increase
    if (self.downsample is not None):
      residual = self.downsample(x)
    print("out= ", out.shape, "residual= ", residual.shape)
    out+=residual
    out=self.relu(out)
    return out
```

Batch Normalization

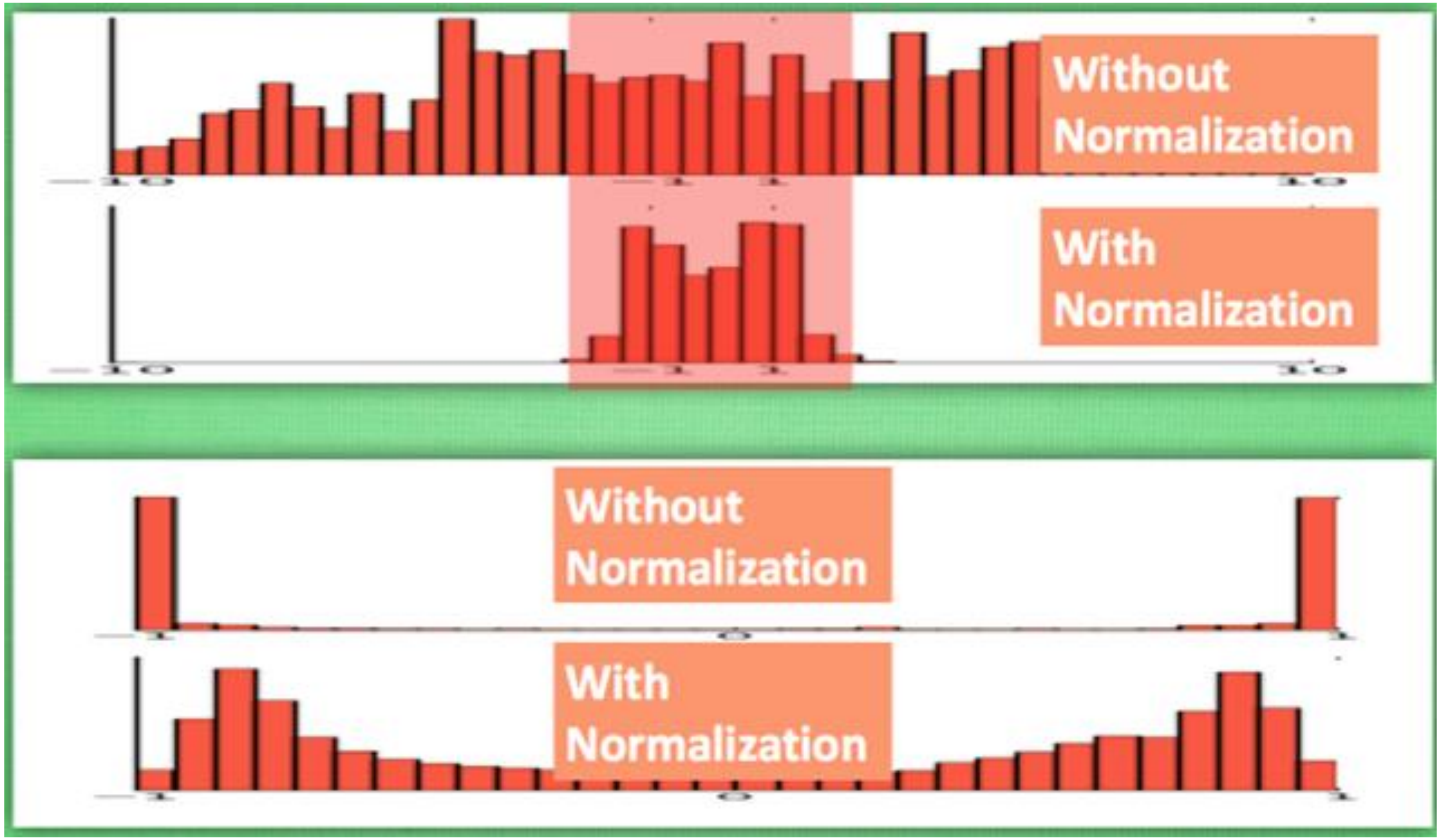
Applies Batch Normalization over a 4D input (a mini-batch of 2D inputs with additional channel dimension) as described in the paper Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift .

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$$

- The mean and standard-deviation are calculated per-dimension over the mini-batches.
- By default, the elements of γare set to 1 and the elements of β are set to 0.

https://pytorch.org/docs/stable/generated/torch.nn.BatchNorm2d.html

Batch Normalization



https://medium.com/ching-i/batch-normalization-%E4%BB%8B%E7%B4%B9-135a24928f12

```python
class MyResNet(nn.Module):
  def __init__(self, block, layers, num_classes=2):
    super(MyResNet, self).__init__()
    self.inplanes = 64
    self.dilation = 1
    self.conv1=nn.Conv2d(3,self.inplanes,kernel_size=7,stride=2,
    self.maxpool=nn.MaxPool2d(kernel_size=3,stride=2, padding=1)
    self.layer1=self._make_layer(block,64,layers[0])
    self.layer2=self._make_layer(block,128,layers[1],stride=2)
    self.avgpool=nn.AdaptiveAvgPool2d((1,1))
    self.fc=nn.Linear(128*block.expansion,num_classes)
    self.linear=nn.Linear(128*block.expansion,num_classes)

  def _make_layer(self, block, planes, blocks, stride=1):
    layers=[]
    layers.append(block(self.inplanes,planes,stride))
    self.inplanes=planes*block.expansion

    for i in range(1,blocks):
      layers.append(block(self.inplanes,planes))
    return nn.Sequential(*layers)

  def forward(self, x):
    x=self.conv1(x)
    x=self.maxpool(x)
    x=self.layer1(x)
    x=self.layer2(x)
    x=self.avgpool(x)
    x=torch.flatten(x, 1)
    x=self.fc(x)
    return x
```

```python
model=MyResNet(BasicBlock,[1,1]).to(device)
print(model)
```

```
MyResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=128, out_features=2, bias=True)
  (linear): Linear(in_features=128, out_features=2, bias=True)
)
```

Practice – Draw the structure of MyResNet

```
MyResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
```

```
[14]: out1=model.conv1(imageTensor.to(device))
      print(out1.shape)

torch.Size([1, 64, 112, 112])

[15]: out2=model.maxpool(out1)
      print(out2.shape)

torch.Size([1, 64, 56, 56])
```

Practice – Draw the structure of MyResNet

```
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
```

```
[16]: out3=model.layer1(out2)
```

```
out=  torch.Size([1, 64, 56, 56]) residual=  torch.Size([1, 64, 56, 56])
```

Practice – Draw the structure of MyResNet

```
(layer2): Sequential(
  (0): BasicBlock(
    (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (downsample): Sequential(
      (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
```

```
[17]: out4 = model.layer2(out3)
```

```
out=  torch.Size([1, 128, 28, 28]) residual=  torch.Size([1, 128, 28, 28])
```

Practice – Draw the structure of MyResNet

```
,
(avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
(fc): Linear(in_features=128, out_features=2, bias=True)
(linear): Linear(in_features=128, out_features=2, bias=True)
```

```
[18]: out5= model.avgpool(out4)
      print(out5.shape)

torch.Size([1, 128, 1, 1])
```

```
[19]: out6=torch.flatten(out5,1)
      print(out6.shape)

torch.Size([1, 128])
```

```
[20]: out7 = model.fc(out6)
      print(out7)

tensor([[-0.0661, -0.1440]], device
```

Practice – Load pre-trained ResNet

```
In [2]: import torchvision
        model = torchvision.models.resnet18(pretrained=True)

Downloading: "https://download.pytorch.org/models/resnet18-5c106cde.pth" t

HBox(children=(FloatProgress(value=0.0, max=46827520.0), HTML(value='')))
```

ResNet

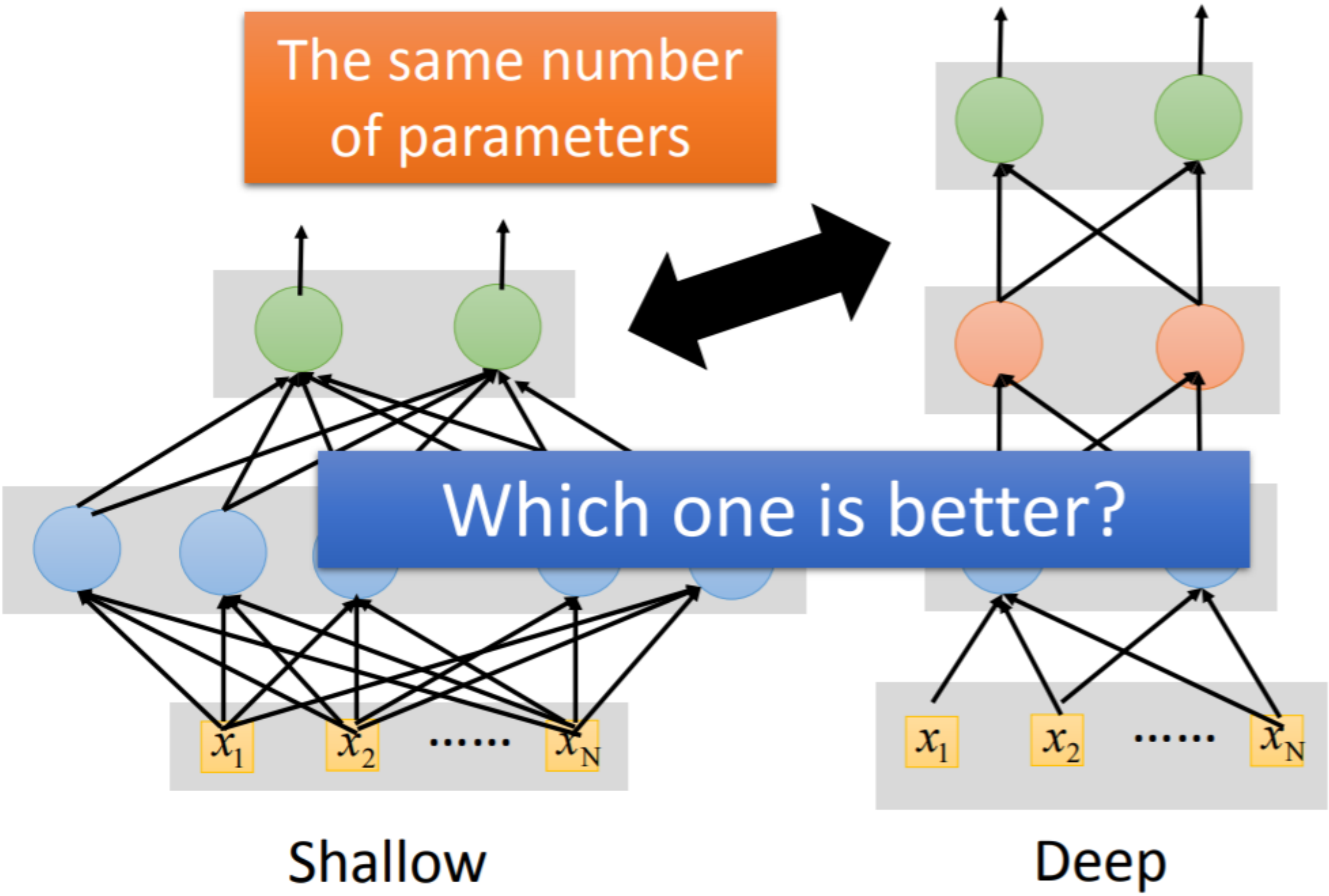
```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
```

ResNet

```
(layer2): Sequential(
  (0): BasicBlock(
    (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (downsample): Sequential(
      (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (1): BasicBlock(
    (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU(inplace=True)
    (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
)
```

Why deep ?

With same number of parameters, which NN is better?



Reference: 李弘毅 ML Lecture 11 https://youtu.be/XsC9byQkUH8

Deep is better

Layer X Size	Word Error Rate (%)	Layer X Size	Word Error Rate (%)
1 X 2k	24.2		
2 X 2k	20.4		
3 X 2k	18.4		
4 X 2k	17.8		
5 X 2k	17.2	1 X 3772	22.5
7 X 2k	17.1	1 X 4634	22.6
		1 X 16k	22.1

Why?

deep + thin

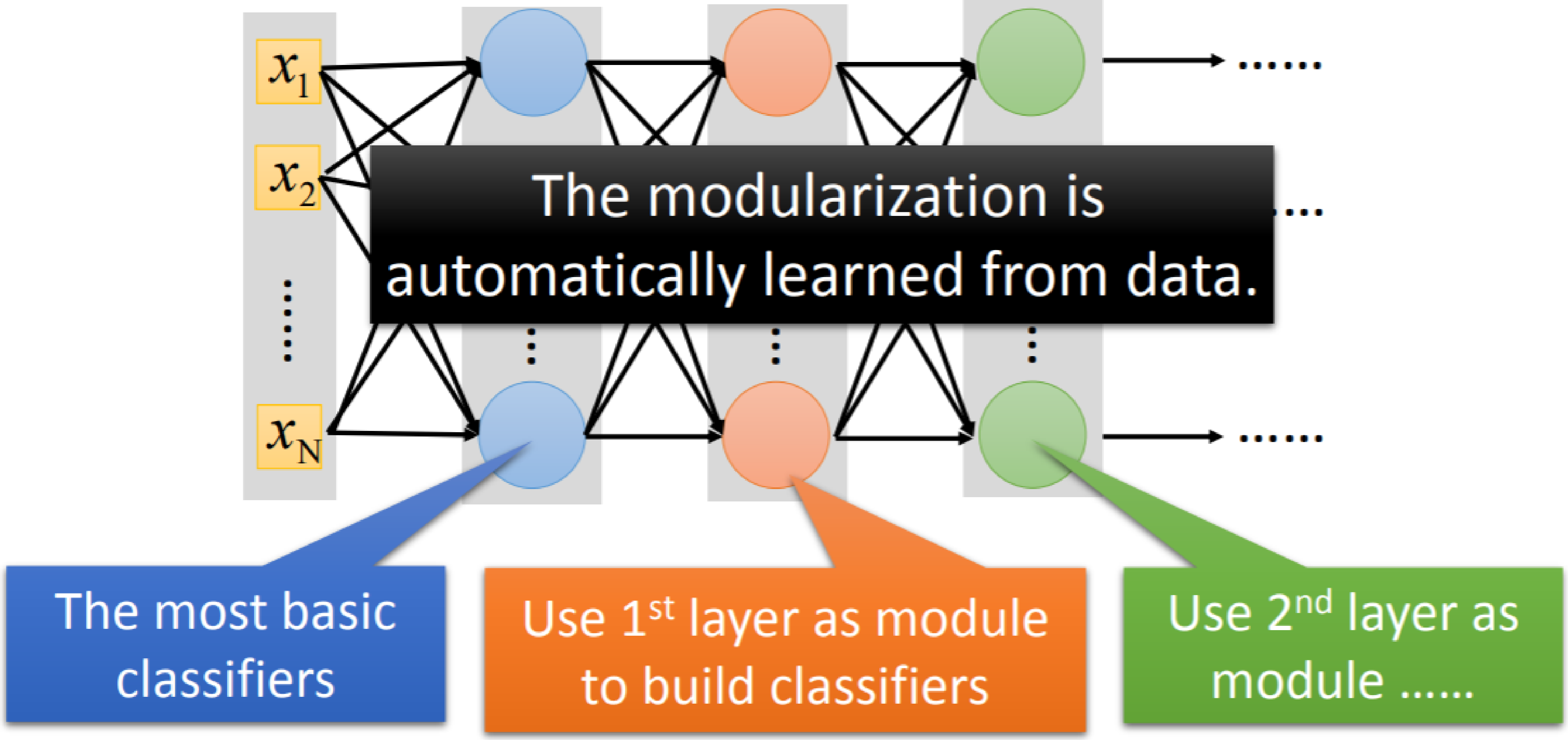
short + fat

Seide, Frank, Gang Li, and Dong Yu. "Conversational Speech Transcription Using Context-Dependent Deep Neural Networks." *Interspeech*. 2011.

Reference: 李弘毅 ML Lecture 11 https://youtu.be/XsC9byQkUH8

Reason 1 – Modularization

- Deep → Modularization → Less training data?



Reference: 李弘毅 ML Lecture 11 https://youtu.be/XsC9byQkUH8

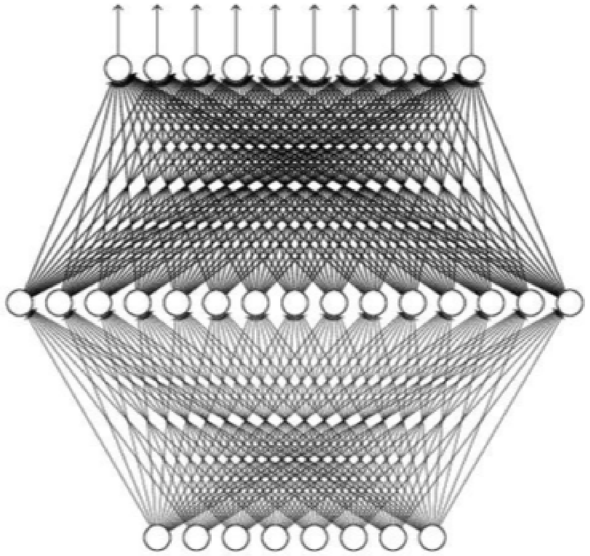
Universality theorem

Any continuous function f

$$f : R^N \rightarrow R^M$$

Can be realized by a network with one hidden layer

(given **enough** hidden neurons)



Reference for the reason: http://neuralnetworksanddeeplearning.com/chap4.html

Yes, shallow network can represent any function.

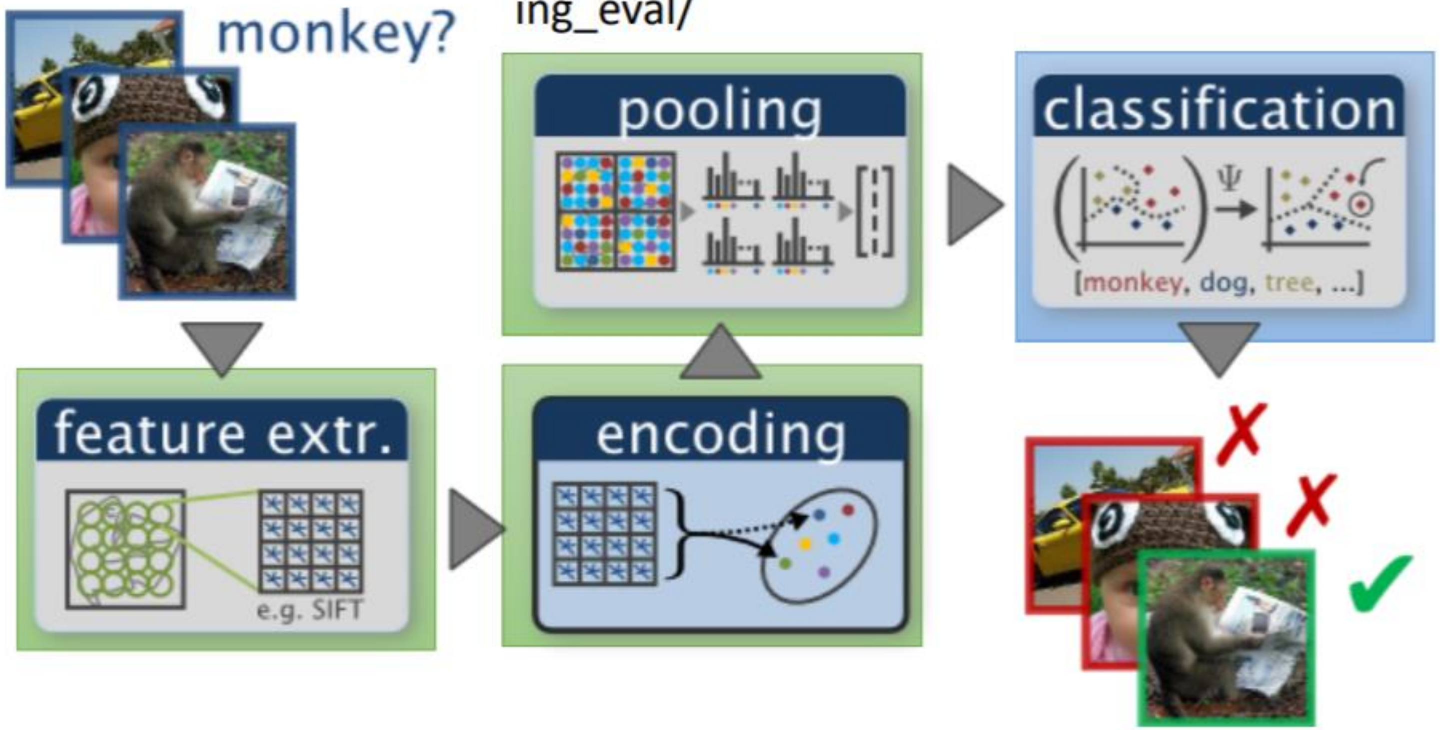
However, using deep structure is more effective.

Reference: 李弘毅 ML Lecture 11 https://youtu.be/XsC9byQkUH8

Reason 2: End-to-end learning

- Shallow Approach

http://www.robots.ox.ac.uk/~vgg/research/encoding_eval/

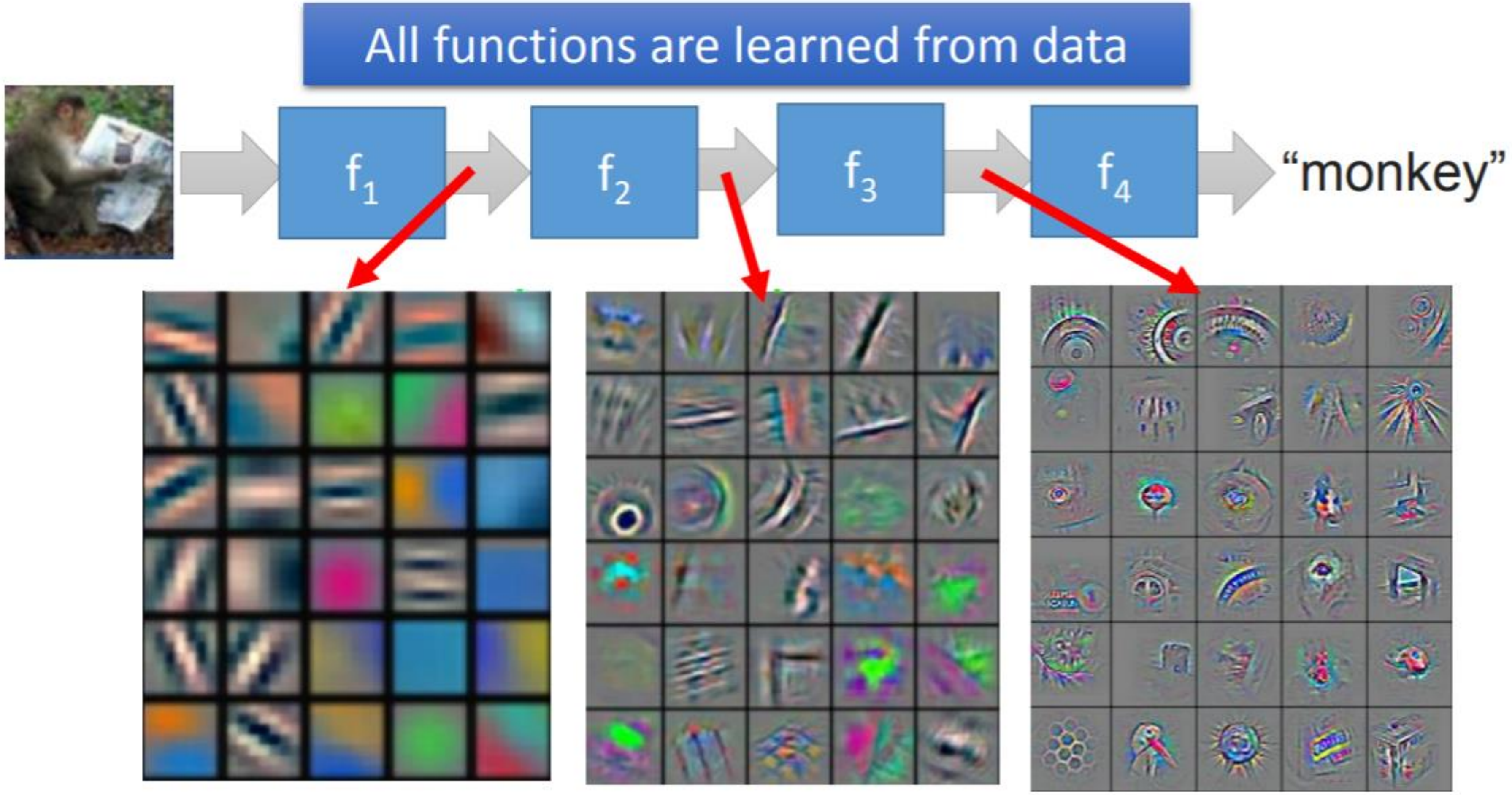


:hand-crafted :learned from data

Reference: 李弘毅 ML Lecture 11 https://youtu.be/XsC9byQkUH8

End-to-end learning

- **Deep Learning**

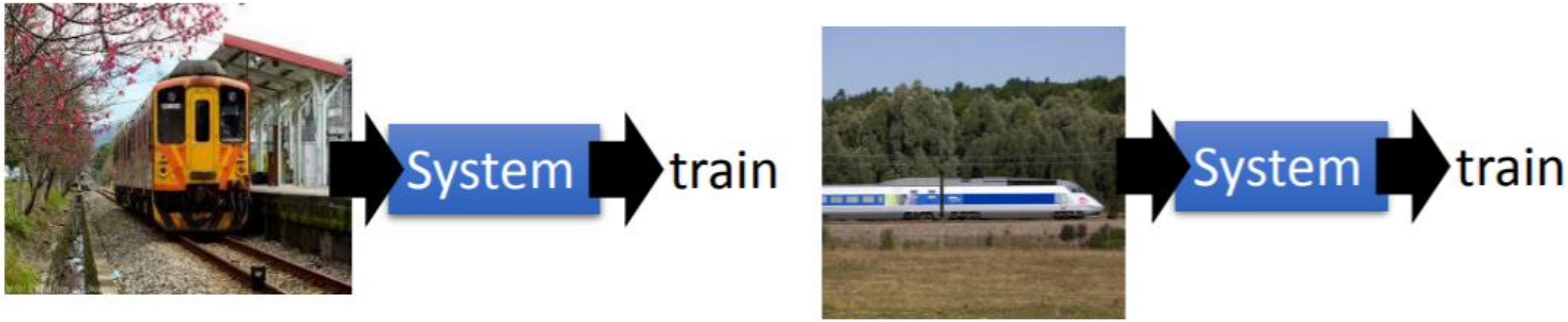


Reference: 李弘毅 ML Lecture 11 https://youtu.be/XsC9byQkUH8

Reason 3 - Easier to handle complex task

- Very similar input, different output

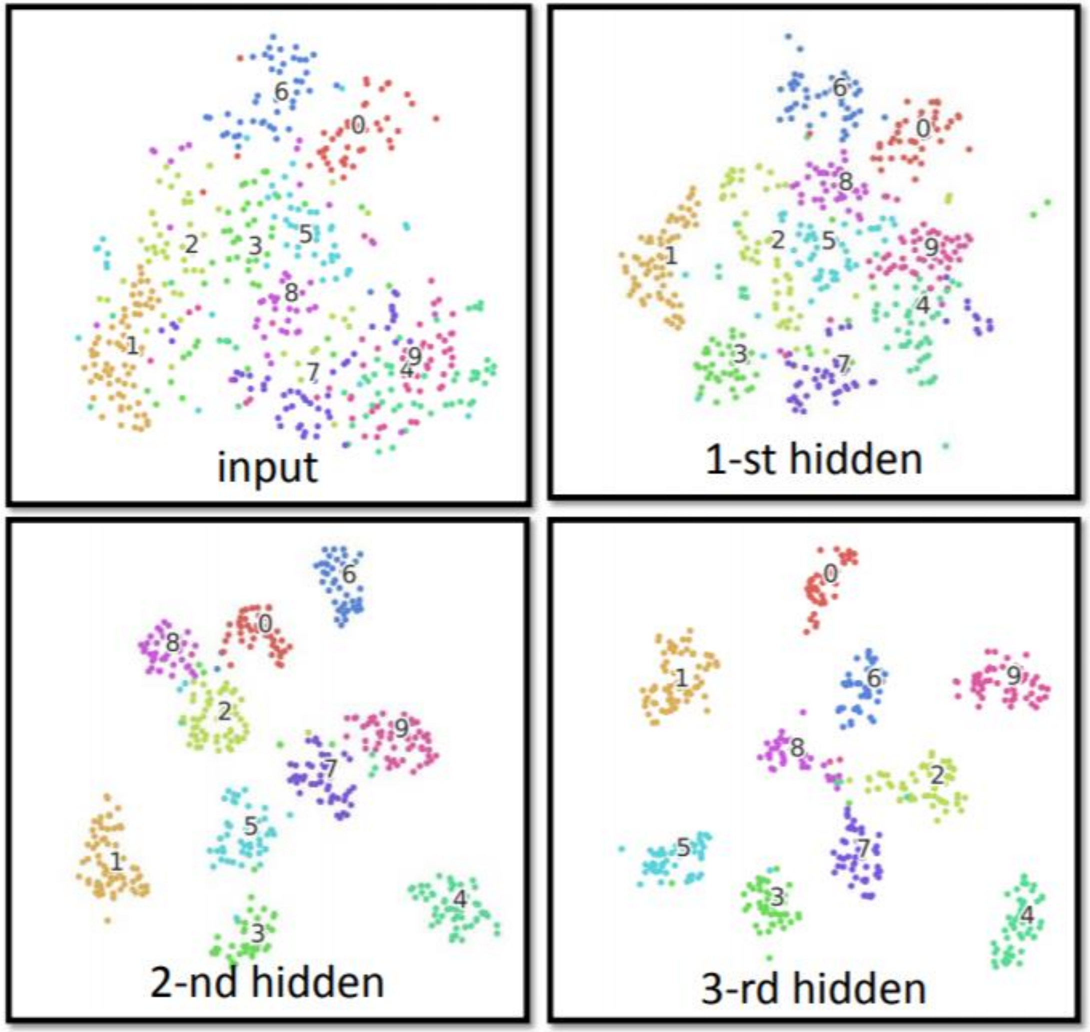
- Very different input, similar output



Reference: 李弘毅 ML Lecture 11 https://youtu.be/XsC9byQkUH8

Easier to handle complex task with DL

MNIST

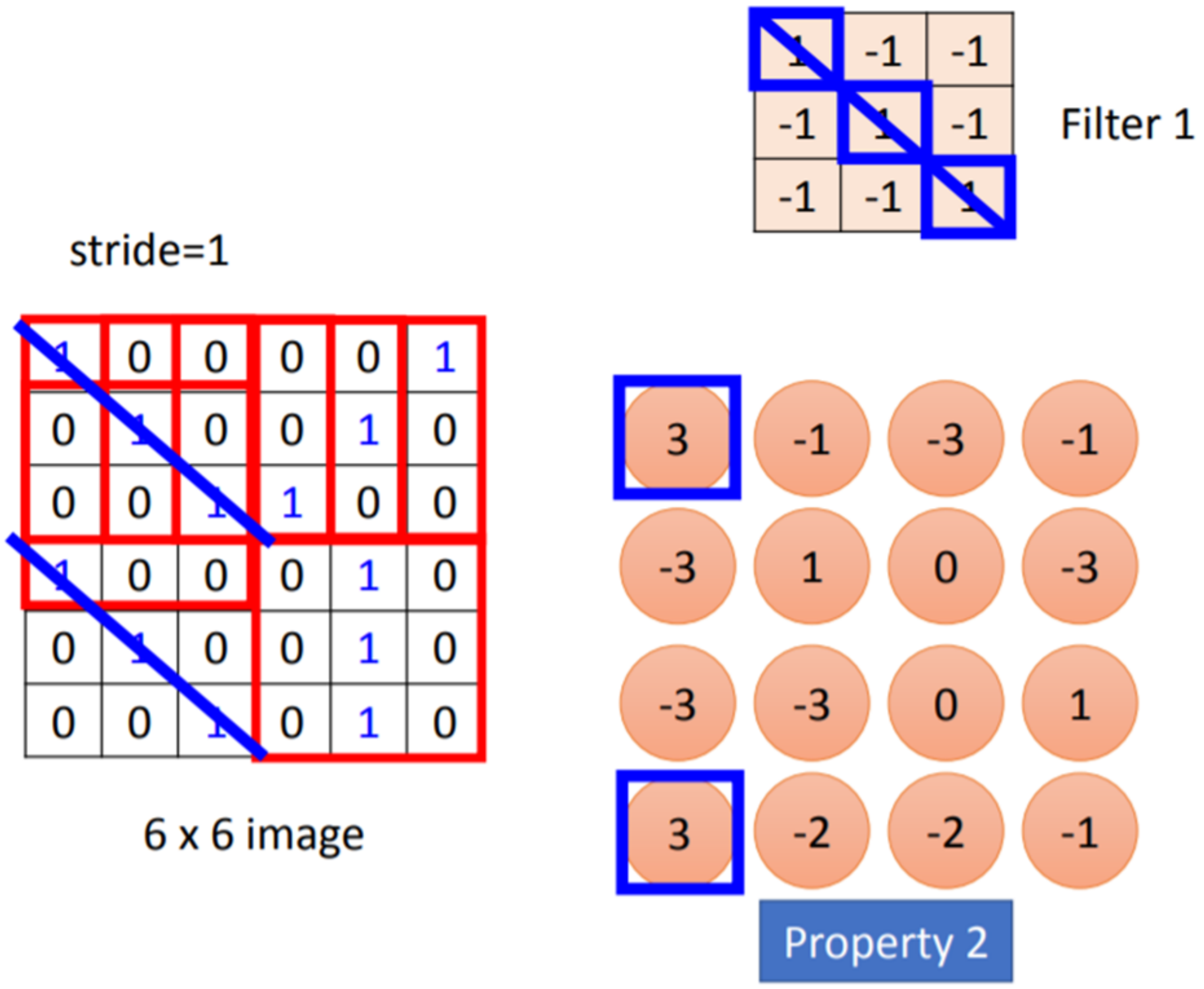


How to implement this in PyTorch?

Reference: 李弘毅 ML Lecture 11 https://youtu.be/XsC9byQkUH8

What does CNN learn?

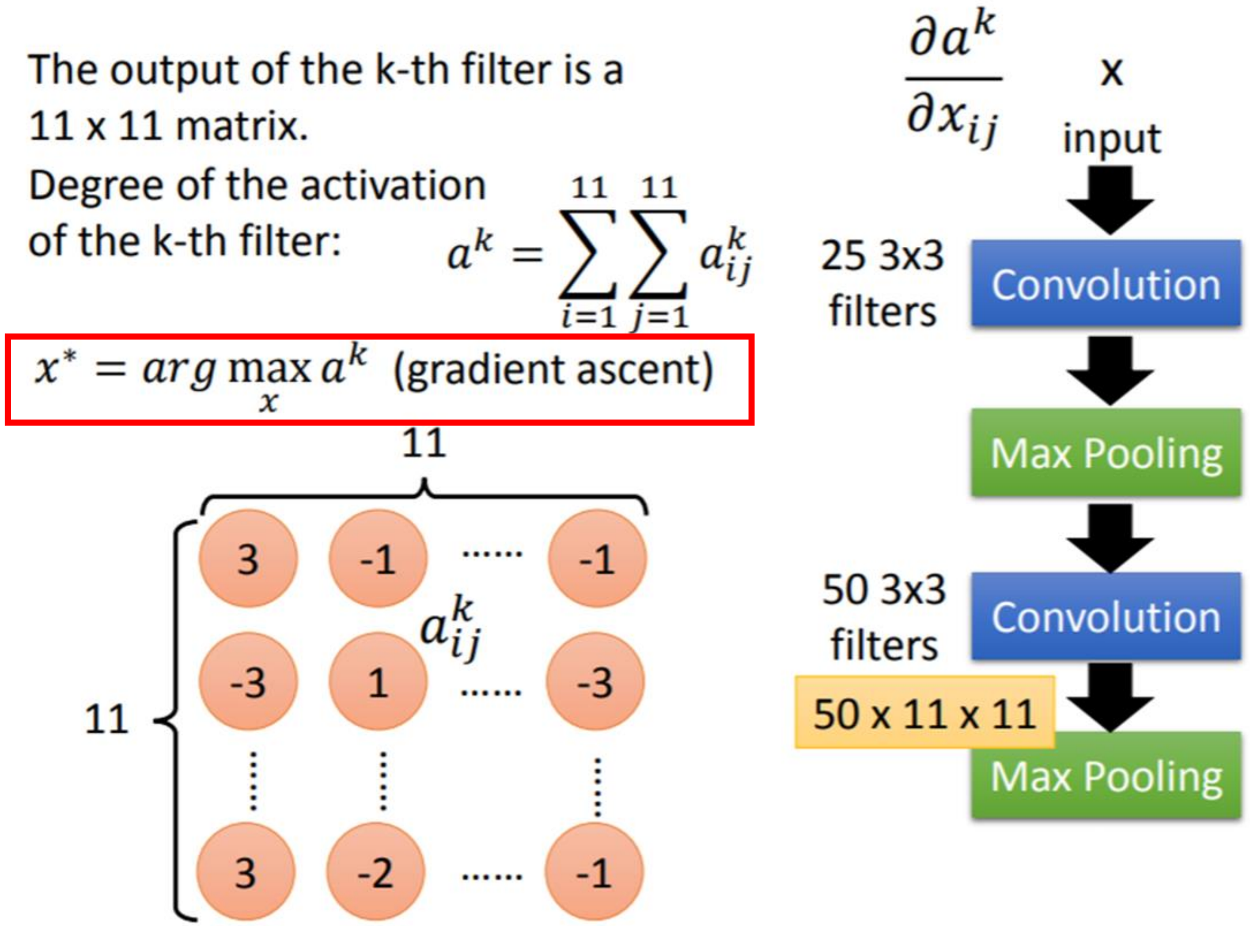
Recap – Filter searches a particular pattern in different regions and summarize the results in feature map



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

Only the weight of the 1st convolution filters can be directly visualized. How to interpret the filter weights of other convolution layers?

How to use gradient ascent to implement this in PyTorch?



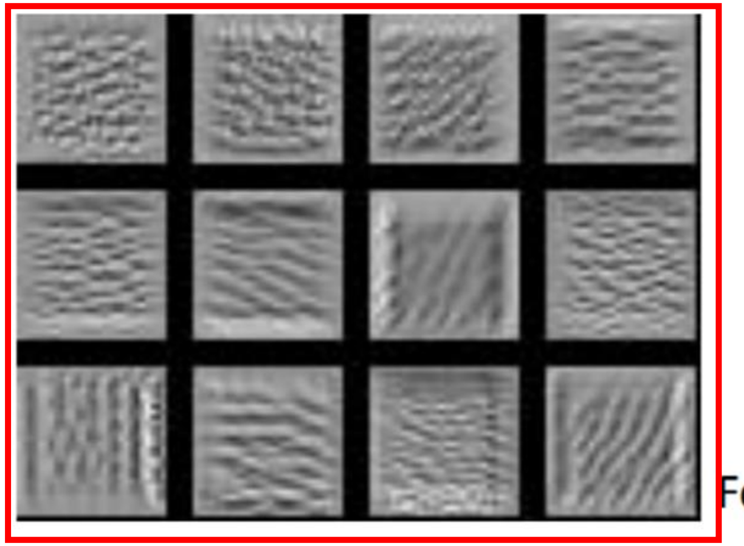
Reference: 李弘毅 ML Lecture 10 https://youtu.be/FrKWiRv254g

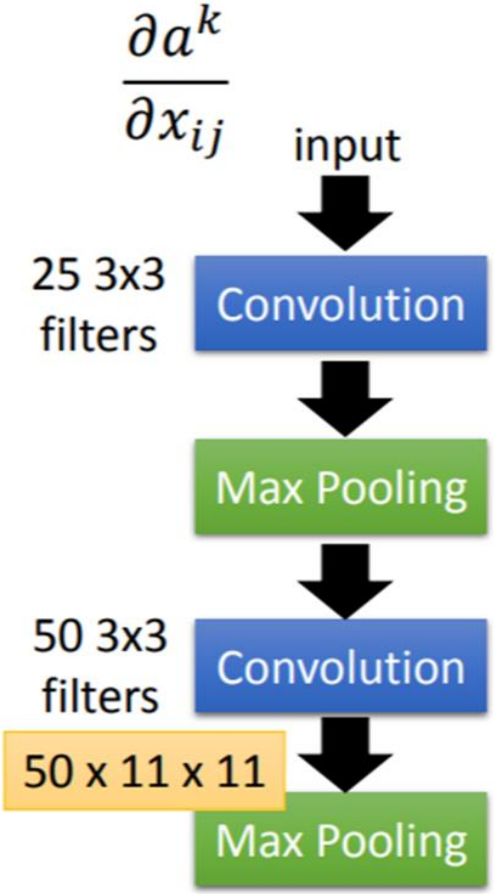
With MNIST data set, in the convolution layer, the filters detects a particular texture pattern.

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)





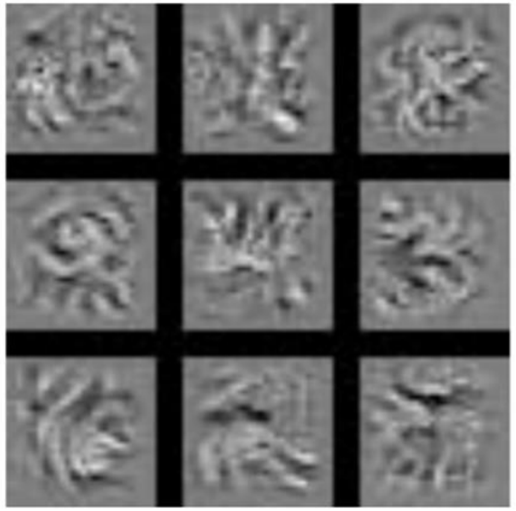
For each filter

How to implement this in PyTorch?

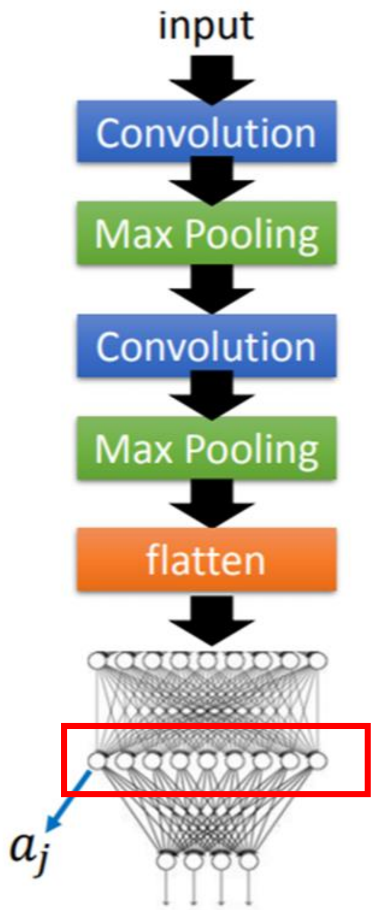
Reference: 李弘毅 ML Lecture 10 https://youtu.be/FrKWiRv254g

In the hidden layer of the fully-connected NN, each neuron detects an overall pattern in the picture rather than a particular texture pattern.

Find an image maximizing the output of neuron: $x^* = arg \max_x a^j$

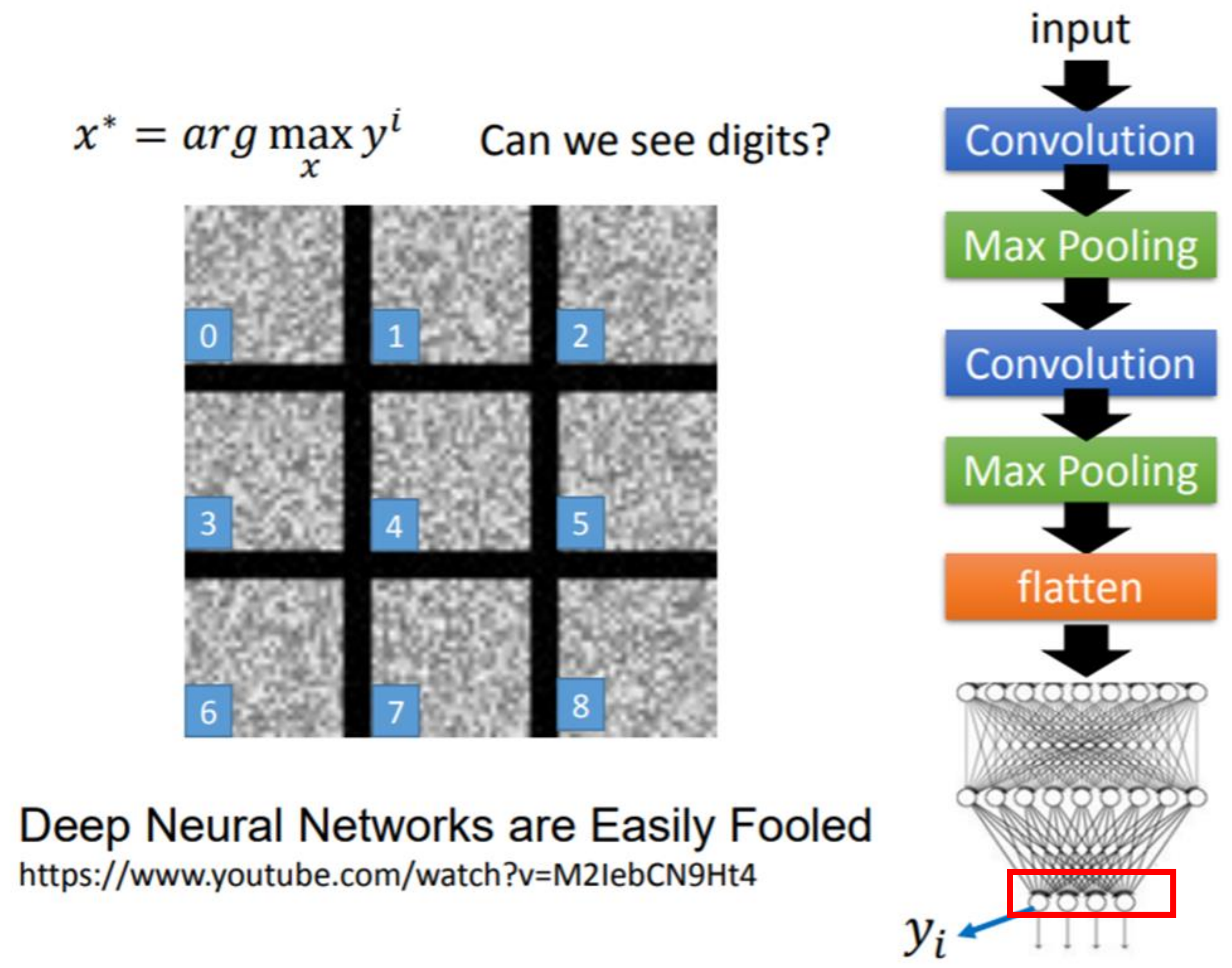


Each figure corresponds to a neuron

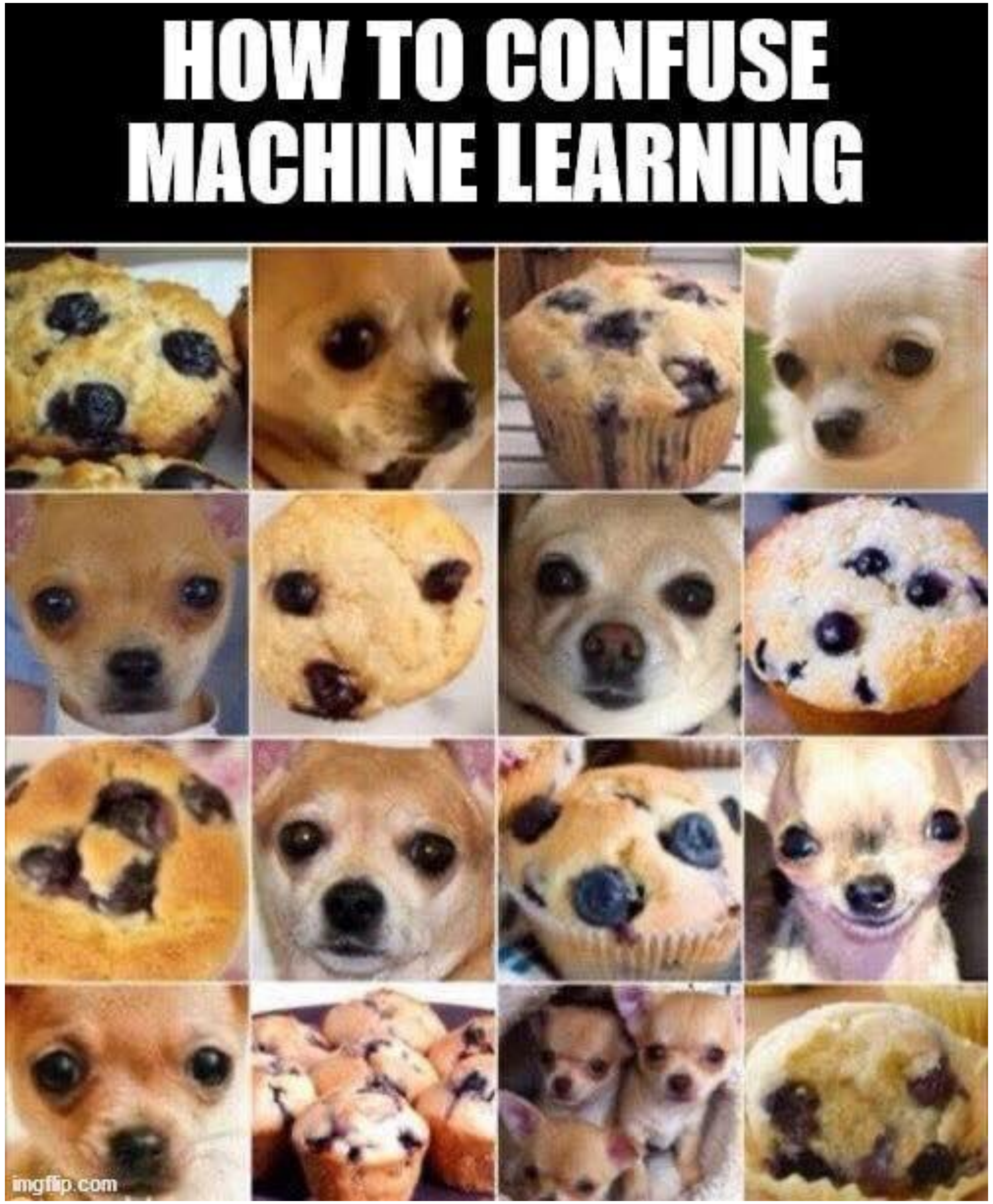


Reference: 李弘毅 ML Lecture 10 https://youtu.be/FrKWiRv254g

If we watch the output layer node, it is easy to see that CNN is easily fooled.



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

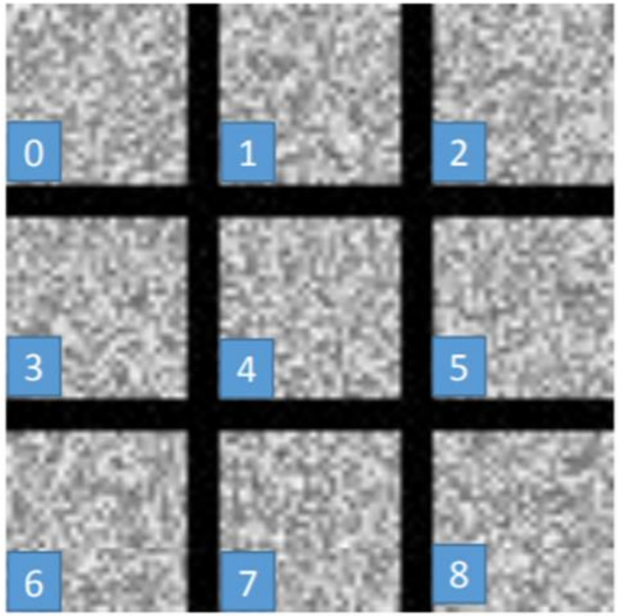


https://www.facebook.com/105366834575253/photos/a.106433371135266/217334733378462/

Adding regularization to the objective function to force most pixels be "NO INK"

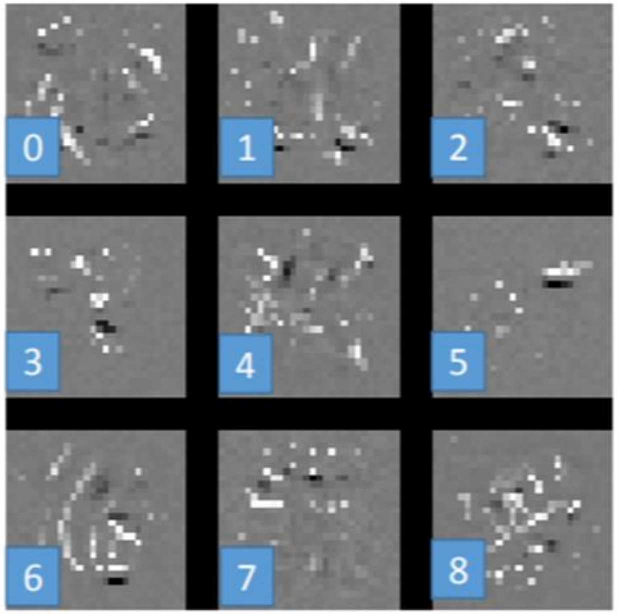
Here white pixels indicate ink, and black pixels indicate “NO INK”.

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



Over all pixel values

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

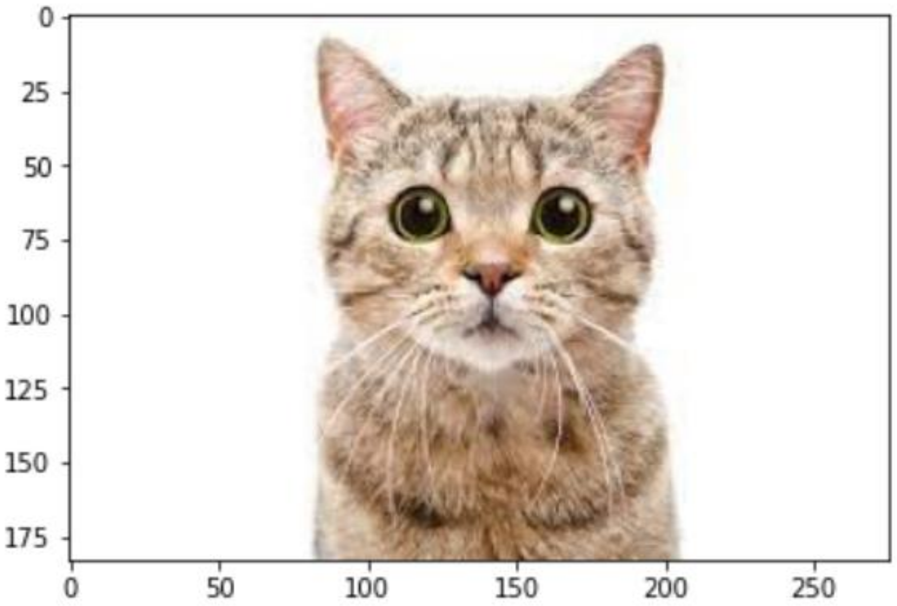


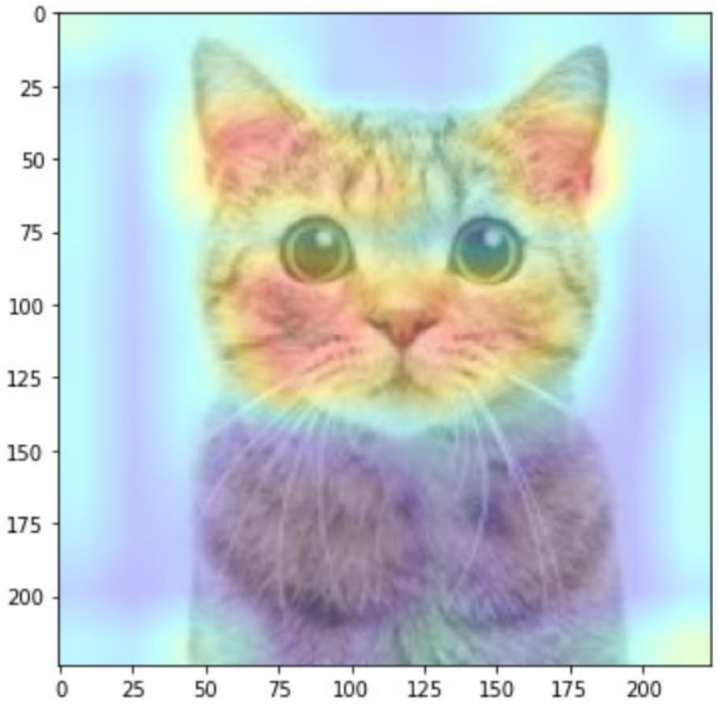
L1 regularization to force xij=0, i.e., force most pixels to be black, NO INK (as only small part of the image has ink)

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

Practice – What does CNN learn?

- Run “6.5 GradCAM.ipynb"





HW5 (3)

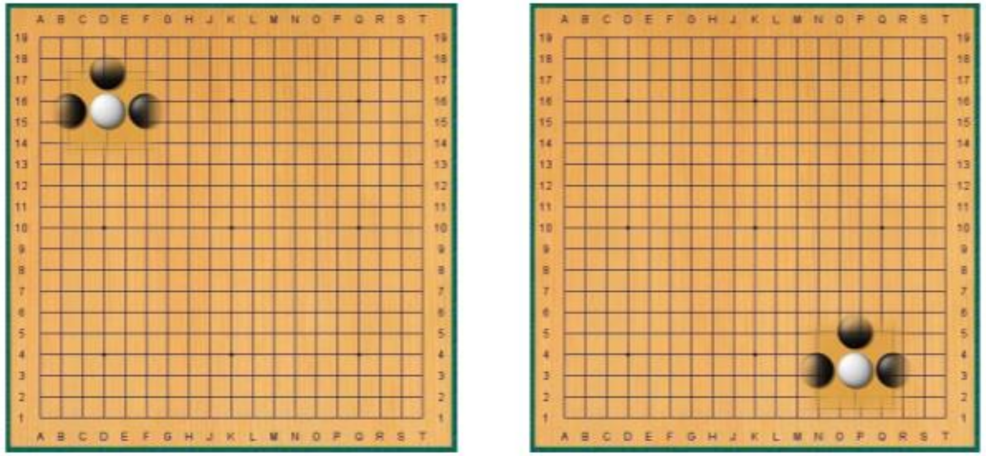
	Class index predicted by the model	Class index you assigned
AlexNet		
VGG		
ResNet18		

Use CNN in Alpha 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.



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

Use CNN in Alpha GO

Neural network architecture. The input to the policy network is a 19 × 19 × 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 strid [illegible] max func-
tion. The [illegible] Extended Data Table 3 additionally show the results of training with $k = 128$, 256 and 384 filters.

Alpha Go does not use Max Pooling ……

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