CAI 4104/6108 – Machine Learning Engineering:

Convolutional Neural Networks (2)

Prof. Vincent Bindschaedler

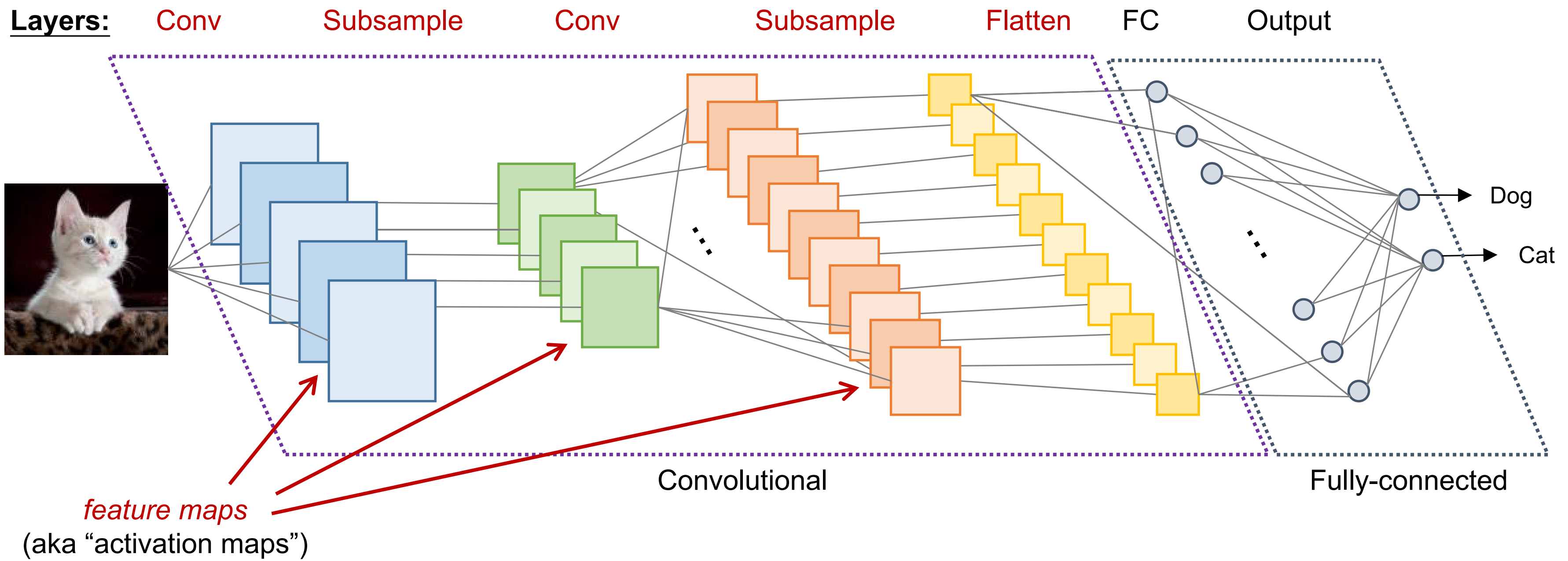
Spring 2024

Reminder: Convolutional Neural Networks

- History:
  - 1958: Hubel and Wiesel experiments on cats
    - Won the Nobel Prize in Physiology or Medicine (1981)
    - Insight: neurons in visual cortex have a small local receptive field
  - 1998: LeCun et al. propose the LeNet-5 architecture
- Convolutional Neural Networks:
  - Architecture for neural networks using convolutional layers
    - Convolutional layers: each neuron/unit is only connected to a small number of neurons/units in the previous layer
    - Fewer neurons/units than fully-connected layers
  - Well-suited to computer vision tasks or tasks on image data
    - Can also be applied to other tasks: for example some tasks in natural language processing
  - Preeminent neural network architectures for many state-of-the-art applications
    - E.g.: self-driving cars, video classification, image search systems, etc.
  - Remark: CNNs have high memory usage *during training*

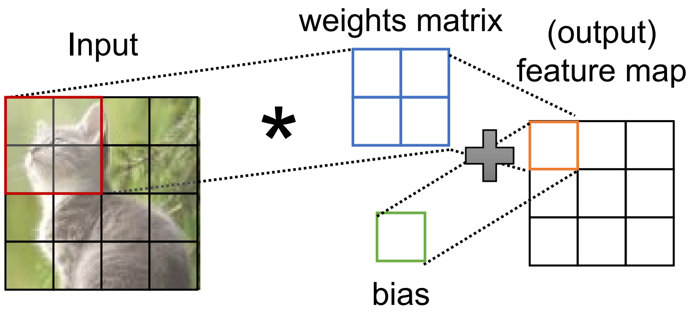
Reminder: CNN Architecture

- Example & Terminology:



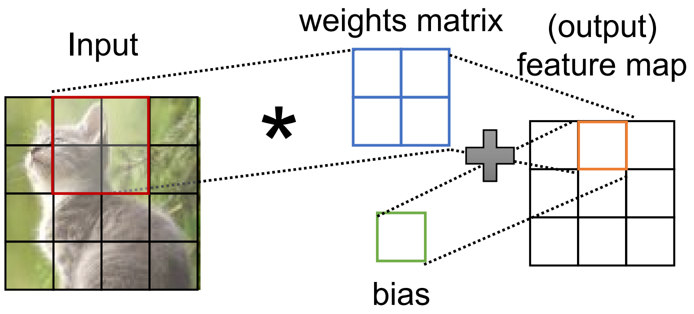
Convolutional Layer

- A convolutional layer has a set of filters (aka kernels)
  - Each filter slides (i.e., convolves) across the image (or previous layer's output) producing a feature map
  - The filter is represented by a $f_w \times f_h$ matrix of weights $\boldsymbol{F}$ and a bias b; there is also an activation function
    - Applying the filter produces a ***single*** *output value* (real number) for each sliding window
  - Parameters: weight matrix $\boldsymbol{F}$ and bias b
  - Hyperparameters: filter/kernel size (f_w, f_h), stride, padding strategy ('valid' or 'same'), and activation function



Convolutional Layer

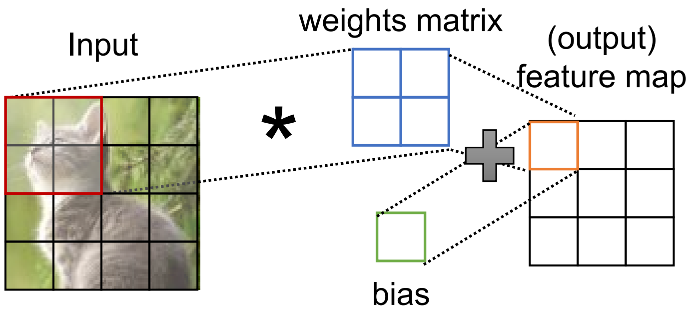
- A convolutional layer has a set of filters (aka kernels)
  - Each filter slides (i.e., convolves) across the image (or previous layer's output) producing a feature map
  - The filter is represented by a $f_w \times f_h$ matrix of weights $\boldsymbol{F}$ and a bias b; there is also an activation function
    - Applying the filter produces a ***single*** *output value* (real number) for each sliding window
  - Parameters: weight matrix $\boldsymbol{F}$ and bias b
  - Hyperparameters: filter/kernel size (f_w, f_h), stride, padding strategy ('valid' or 'same'), and activation function



- Remarks:
  - The weights matrix remains the same throughout the convolution
    - There are only $f_w\ f_h$ +1 parameters for the filter (and it does not depend on the size of the input)
  - Typically we have multiple filters per layer, so we get one feature map as output for each filter
  - Output size of feature map depends on the size of the filter, stride, and padding strategy

Convolutional Layer

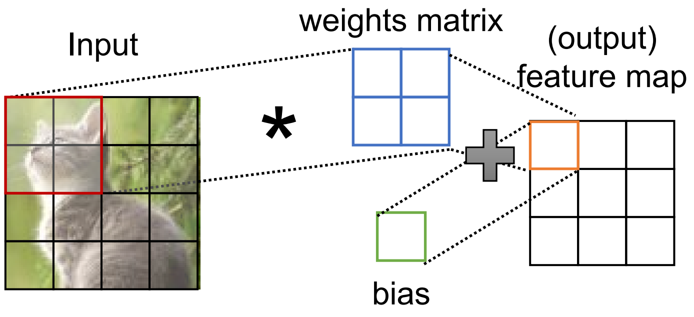
- A convolutional layer has a set of filters (aka kernels)
  - Each filter slides (i.e., convolves) across the image (or previous layer's output) producing a feature map
  - The filter is represented by a $f_w \times f_h$ matrix of weights $\boldsymbol{F}$ and a bias b; there is also an activation function
    - Applying the filter produces a ***single*** *output value* (real number) for each sliding window
  - Parameters: weight matrix $\boldsymbol{F}$ and bias b
  - Hyperparameters: filter/kernel size (f_w, f_h), stride, padding strategy ('valid' or 'same'), and activation function

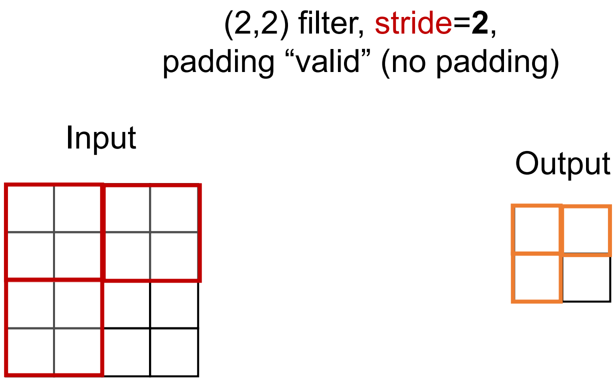


(2,2) filter, stride=1,
padding "valid" (no padding)

Convolutional Layer

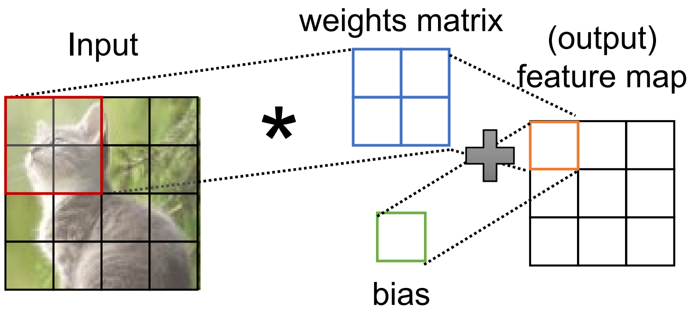
- A convolutional layer has a set of filters (aka kernels)
  - Each filter slides (i.e., convolves) across the image (or previous layer's output) producing a feature map
  - The filter is represented by a $f_w \times f_h$ matrix of weights $\boldsymbol{F}$ and a bias b; there is also an activation function
    - Applying the filter produces a ***single*** *output value* (real number) for each sliding window
  - Parameters: weight matrix $\boldsymbol{F}$ and bias b
  - Hyperparameters: filter/kernel size (f_w, f_h), stride, padding strategy ('valid' or 'same'), and activation function



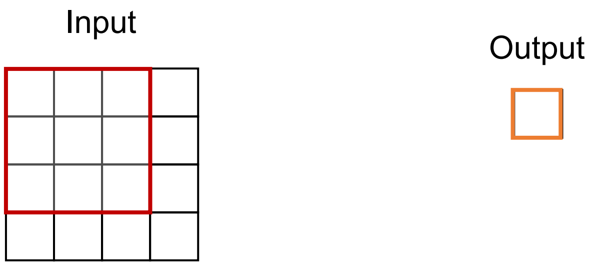


Convolutional Layer

- A convolutional layer has a set of filters (aka kernels)
  - Each filter slides (i.e., convolves) across the image (or previous layer's output) producing a feature map
  - The filter is represented by a $f_w \times f_h$ matrix of weights $\boldsymbol{F}$ and a bias b; there is also an activation function
    - Applying the filter produces a ***single*** *output value* (real number) for each sliding window
  - Parameters: weight matrix $\boldsymbol{F}$ and bias b
  - Hyperparameters: filter/kernel size (f_w, f_h), stride, padding strategy ('valid' or 'same'), and activation function

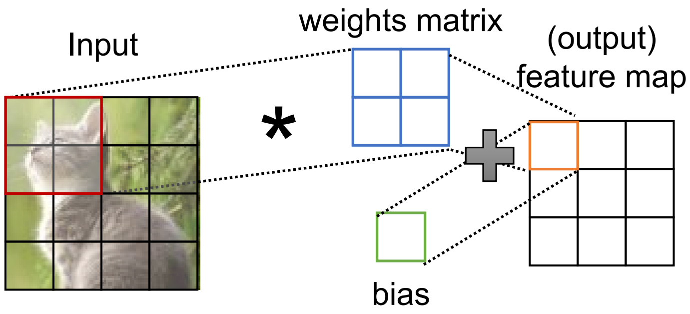


(3,3) filter, stride=2,
padding "valid" (no padding)

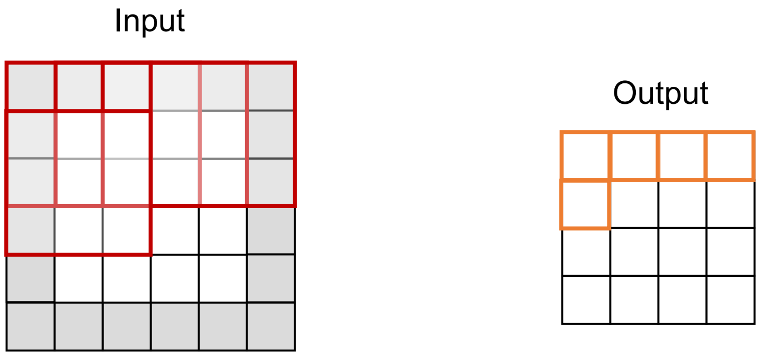


Convolutional Layer

- A convolutional layer has a set of filters (aka kernels)
  - Each filter slides (i.e., convolves) across the image (or previous layer's output) producing a feature map
  - The filter is represented by a $f_w \times f_h$ matrix of weights $\boldsymbol{F}$ and a bias b; there is also an activation function
    - Applying the filter produces a ***single*** *output value* (real number) for each sliding window
  - Parameters: weight matrix $\boldsymbol{F}$ and bias b
  - Hyperparameters: filter/kernel size (f_w, f_h), stride, padding strategy ('valid' or 'same'), and activation function

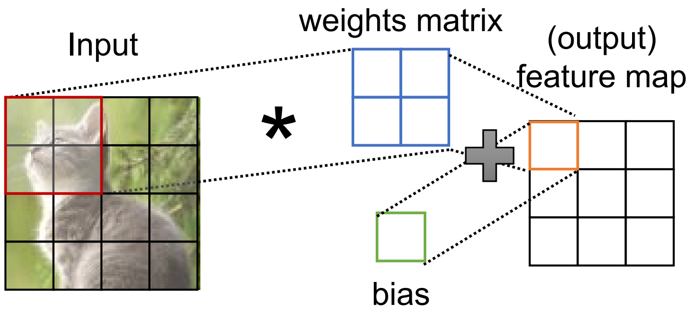


(3,3) filter, stride=1,
padding "**same**" (*zero* padding)



Convolutional Layer

- A convolutional layer has a set of filters (aka kernels)
  - Each filter slides (i.e., convolves) across the image (or previous layer's output) producing a feature map
  - The filter is represented by a $f_w \times f_h$ matrix of weights $\boldsymbol{F}$ and a bias b; there is also an activation function
    - Applying the filter produces a ***single*** *output value* (real number) for each sliding window
  - Parameters: weight matrix $\boldsymbol{F}$ and bias b
  - Hyperparameters: filter/kernel size (f_w, f_h), stride, padding strategy ('valid' or 'same'), and activation function



How to calculate output (i.e., feature map) size?

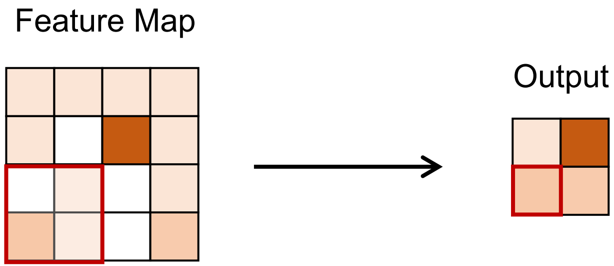
$$\text{output\_size} = \text{floor}[\,(V - K + 2P) / S\,] + 1$$

- V: input volume (e.g., width or height)
- K: kernel size
- P: padding (note: 0 for 'valid)
- S: stride

Subsampling / Pooling Layers

- Subsampling Layers
  - After one (or more) convolutional layers, we can have subsampling (aka "pooling") layers to reduce the dimensions of feature maps
    - Max pooling layer: take the *maximum* value of the sliding window
    - Average/mean pooling layer: take the *mean* value of the sliding window
  - Hyperparameter: pooling size (width, height) — for example: (2, 2) or (3, 3)
  - Note: pooling layers do **not** have **any** parameters

Example: Max pooling (2, 2)



Example Architecture

- Example & Terminology:

Layers:

Conv
Filters=5
Size=(5,5)
Stride=1
Padding: Same

Max Pool
(2,2)

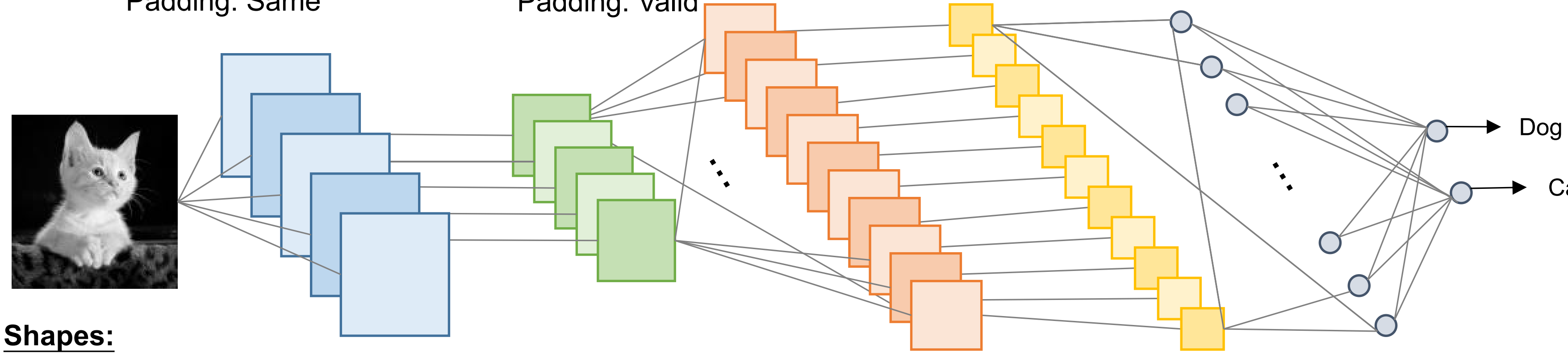
Conv
Filters=10
Size=(2,2)
Stride=2
Padding: Valid

Max Pool
(2,2)

Flatten

FC
128 units

Output



Shapes:

Input: (batch_size, width, height, num_channels)

(50, 64, 64, **5**)

(50, 32, 32, 5)

(50, 16, 16, 10)

(50, 8, 8, 10)

(50, 640)

(50, 128)

(50, 2)

Example: batch size=50, 64 x 64 images, 1 channel

number of feature maps

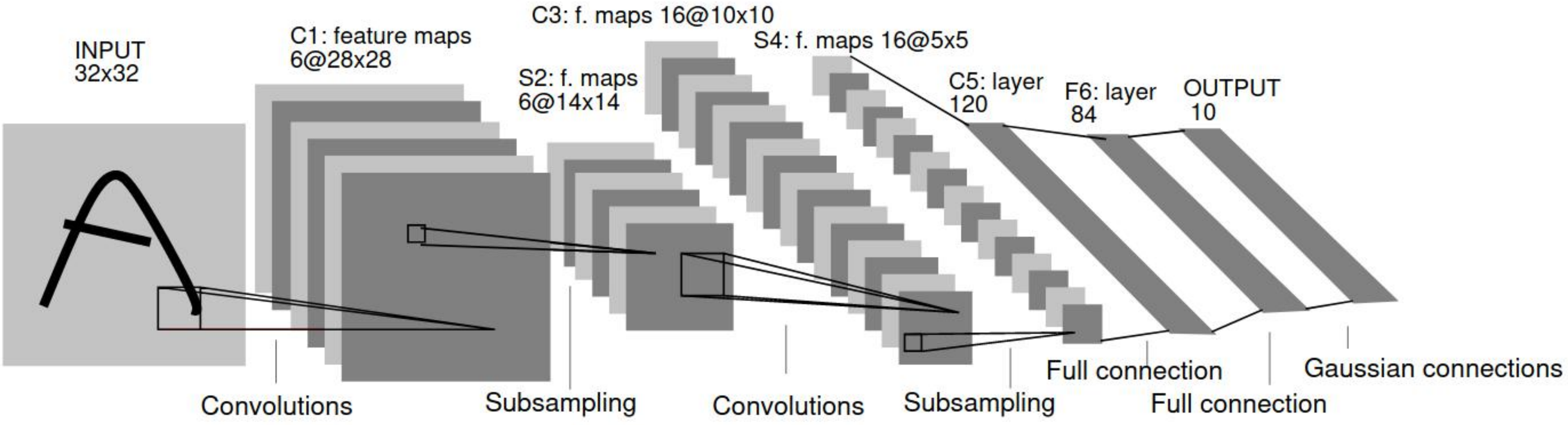
Flatten: 8*8*10 = 640

Architecture & Hyperparameters Tuning

- How do we know what is a good CNN architecture for a problem?
  - There is no one-size-fits-all solution
  - Look at successful CNN architectures (e.g., LeNet-5, AlexNet, ResNet, etc) and adapt them to your problem

- Rules of thumb:
  - Avoid large filter sizes; stick to (2,2), (3,3) etc. But: the first layer can be larger (e.g., (5,5))
  - Use repetition / variants of the following patterns:
    - Pattern1: Conv, MaxPool
    - Pattern2: Conv, Conv, MaxPool
  - Use ReLU as the activation function (for convolutional layers)
  - The deeper you are in the network the more filters you want
    - For example: you could use 32 filters for the first conv layer, then 64 for the second, 128 for the third, etc.
  - Use dropout on the FC (dense) layer after you flatten

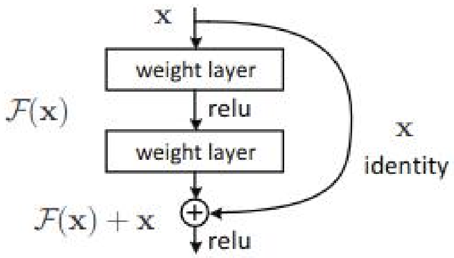
Example: LeNet-5

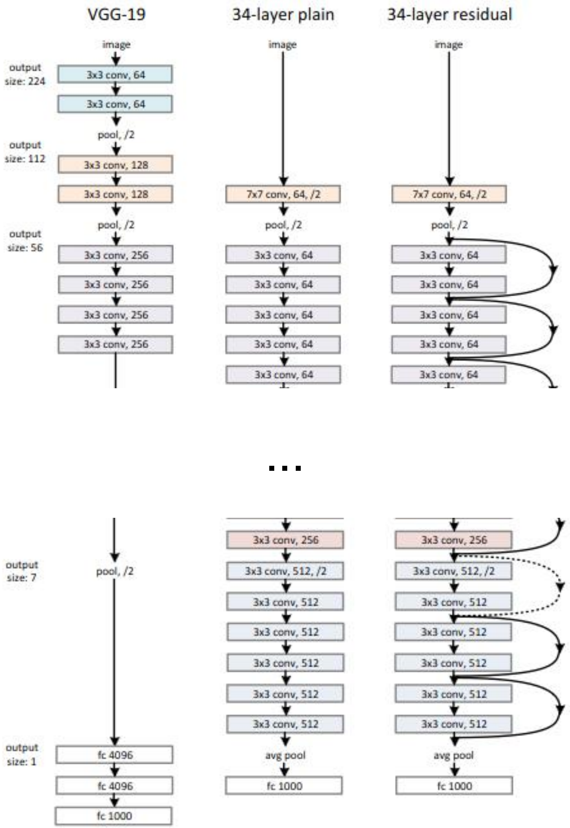


- Source:
  - LeCun et al. "*Gradient-based learning applied to document recognition.*" Proceedings of the IEEE 86.11 (1998): 2278-2324.
- Notes:
  - All layers but the last use **tanh** as activation; nowadays we would use ReLU
  - The subsampling layers are doing **average pooling**; nowadays we would use max pooling
  - The output uses **RBF** activation; nowadays we would use softmax with crossentropy loss

Example: ResNets

Residual Learning building block





Source: He et al. "Deep residual learning for image recognition." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770-778. 2016.

Next Time

- Friday (3/22): Exercise

- Upcoming:
  - Homework 3 is due 3/20 (today)
  - Project