ECE 2195: Special Topics – Computers
Machine Learning

Deep Learning – ConvNet

Mai Abdelhakim, PhD

Assistant Professor of ECE

Swanson School of Engineering

University of Pittsburgh

maia@pitt.edu

Deep Learning

If you need computational resources, check with center for research computing: https://crc.pitt.edu/

- Used in commercial products (e.g. Siri, self-driving cars, recommenders system, fraud detection, translation, gaming, captioning, Grammar correction, … ,)
  - Typically tailored to specific application
- **Automatic feature extraction**
  - Reduces the effort of guessing which features would work well
- **Complex** models, **not easily interpretable**
- Requires **huge computing power** and large training data

Deep Learning Models

- Several popular deep leaning models, including:
  - **Convolutional Neural Network (CNN or ConvNet) --** application example: image recognition

    Input is 2D or 3D grid

  - **Recurrent Neural Network (RNN):** Sequential data, application examples -- text recognition, translation

Reading: https://www.nature.com/articles/nature14539

Python libraries for deep learnings

- There are other more powerful Python-based libraries for deep learning with neural networks (beyond Sklearn), such as:
  - **Keras:**
    - https://keras.io/
    - Model-level library works with several backends, including TensorFlow
  - **TensorFlow**
    - https://www.tensorflow.org/
    - Developed by Google
  - **CNTK**
    - **https://github.com/Microsoft/CNTK**
    - By Microsoft
  - **Theano:**
    - http://deeplearning.net/software/theano/
    - University of Montreal
  - **Lasagne**
    - https://lasagne.readthedocs.io/en/latest/

Deep Learning Concept

Neural networks with **two main phases**:

1. **Feature extraction**

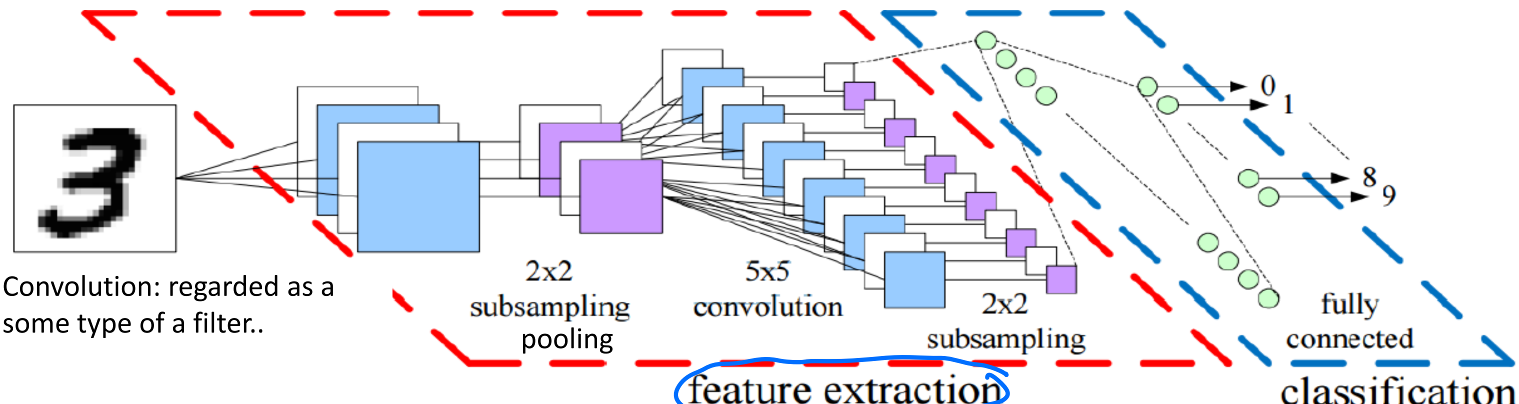
2. Output **prediction** phase

    - Fully connected networks (discussed earlier)

      Typical neural network that does prediction

Example: Pattern Recognition

- Example: Convolutional Neural Net (ConvNet) for pattern recognition
  - Automatic feature extraction made up of hierarchy of feature layers



Convolution: regarded as a some type of a filter..

Each neuron is connected to some units in previous layer (not fully connected)

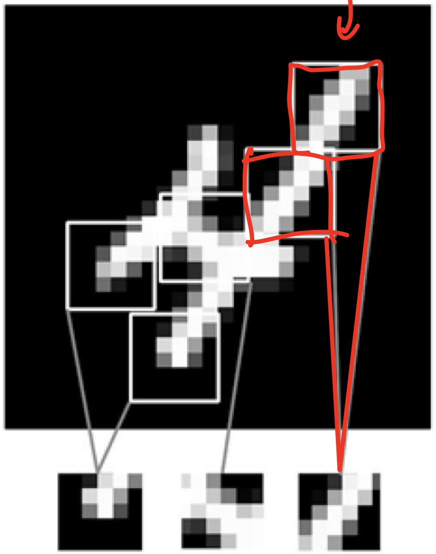
Figure: Peemen et al., "Efficiency Optimization of Trainable Feature Extractors for a Consumer Platform", International Conference on Advanced Concepts for Intelligent Vision Systems, 2011

Convolutional Neural Networks

- For feature extraction phase
  - Detect **local patterns** (instead of global patterns in the fully connected networks)
    - Neuron operates on small number of inputs
  - **Patterns in a small 2D grid** of the input
- Uses mathematical operation called "**convolution**" to detect patterns
  - Convolution between input and a Kernel

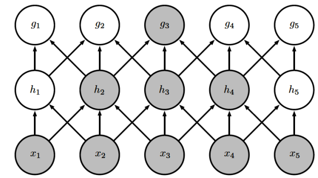
    (Dot product)

Only this part for a CNN



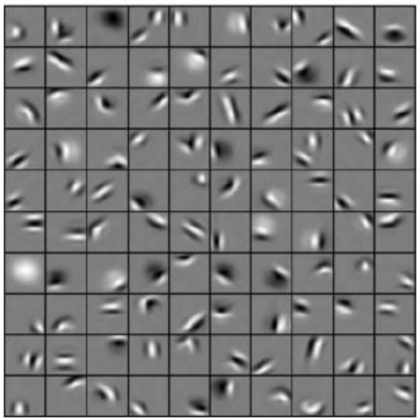
Chollet, Deep Learning with Python

Example of Features Extraction Phase in Layers for Face Recognition using ConvNet

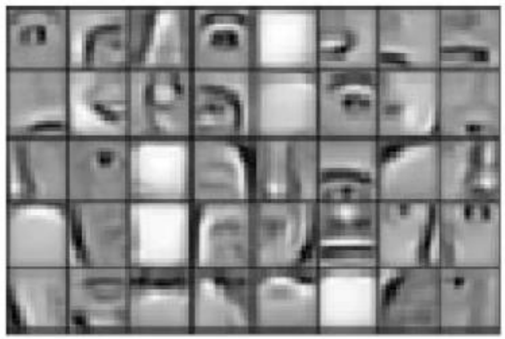


Low-level features

Patterns captured by neurons in first layer



Patterns captured by neurons in second layer



Patterns captured by neurons in third layer

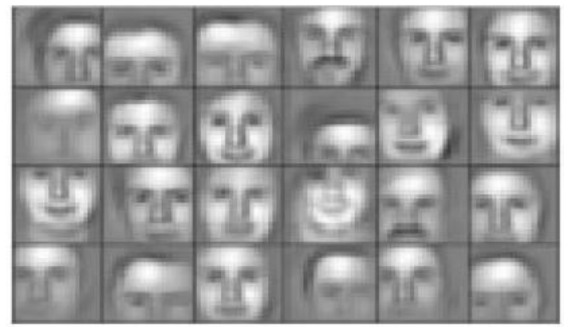


Fig. Ref: Honglak Lee et al. "Unsupervised Learning of Hierarchical Representations with Convolutional Deep Belief Networks". Communications of the ACM, 2011

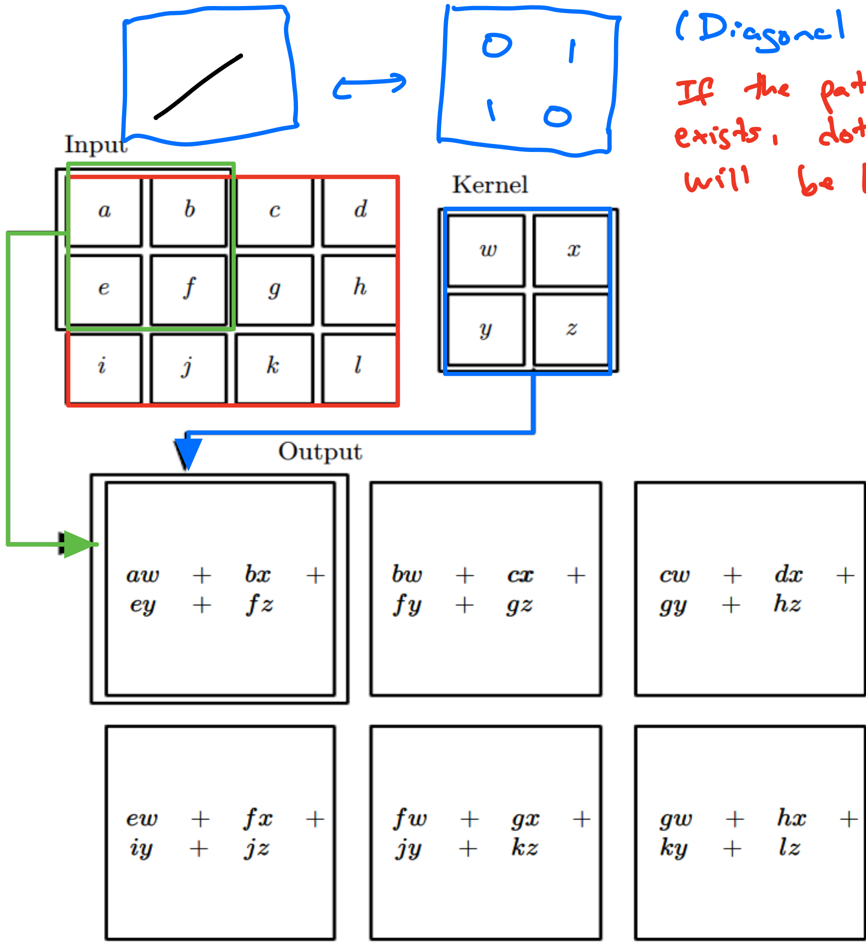
- In feature extraction phase, each neuron focuses on certain part of an image and detects a particular pattern
- First, level of feature extraction obtains some low-level features (like edges), then layers that follow capture higher-level features (like facial expression)
- Each layer uses a combination of features of previous layers

Convolution

- Dot product between the kernel and small window (grid) of the input (image)

- **Kernel** can be regarded as a filter **searching for a pattern**

- Slide the window over the input



(Diagonal edge)

If the pattern exists, dot product will be high

Stride = 1 in this example

Results in 2x3 matrix

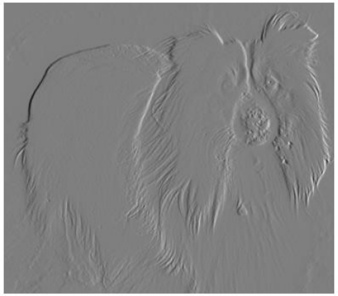
Example: Edge Detection

- Edge detection

Input

Kernel

1	-1



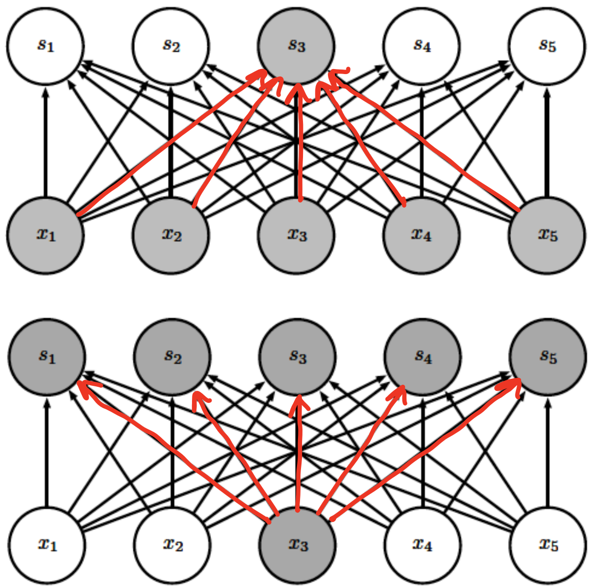
Output

Goodfellow 2016

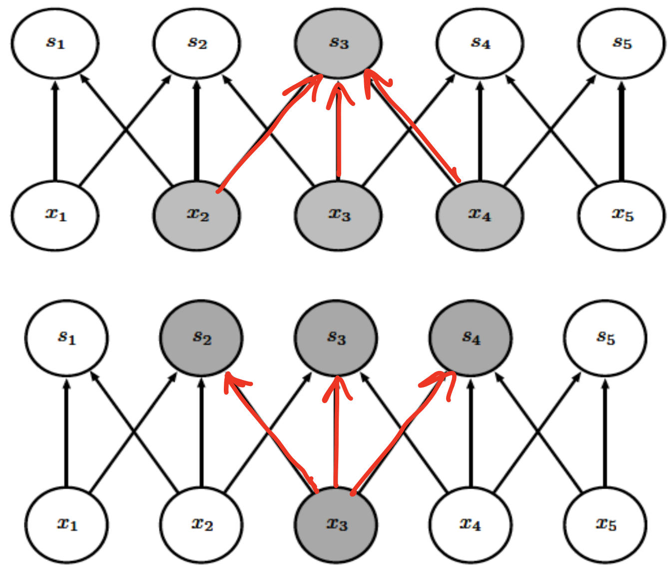
Sparse Connectivity

- Sparse connectivity: Kernel size is much smaller than the input size

Prediction phase, traditional neural net
Dense connectivity: output of each s_i depends on all input x_i

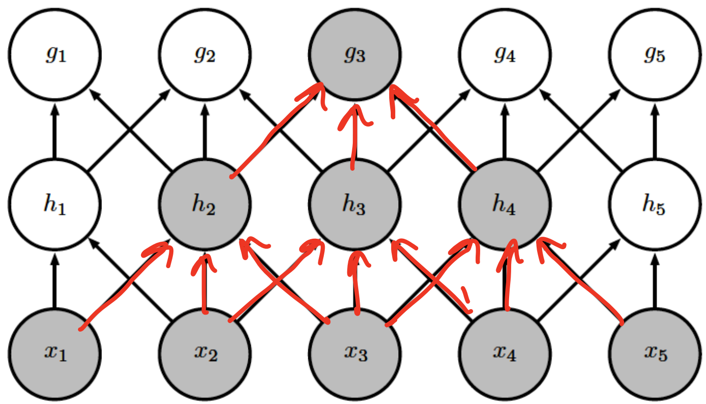


Feature extraction phase, CNN
Sparse connectivity: output of each s_i depends on few of the input x_i



Deep Convolution Networks: Receptive Field Grows with Layers

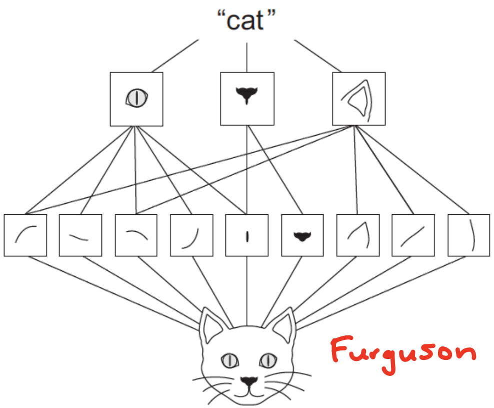
- Receptive field grows as you add more layers
  - Output of a hidden units will be function of more inputs as you add more layers (as the network gets deeper)



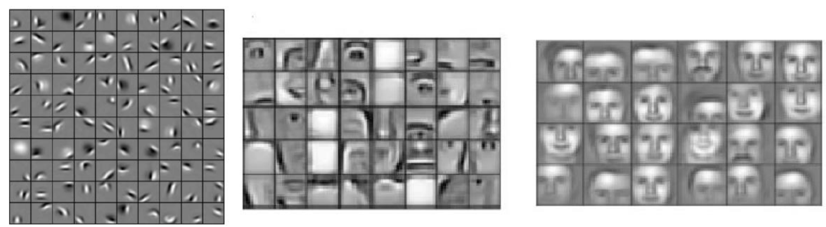
Goodfellow 2016

Learn Hierarchies of Patterns

- Learn spatial hierarchies of patterns
  - First layer: learn small local patterns such as edges
  - Next layer: learn larger patterns made of output features of the previous layer



Chollet, Deep Learning with Python



Face recognition

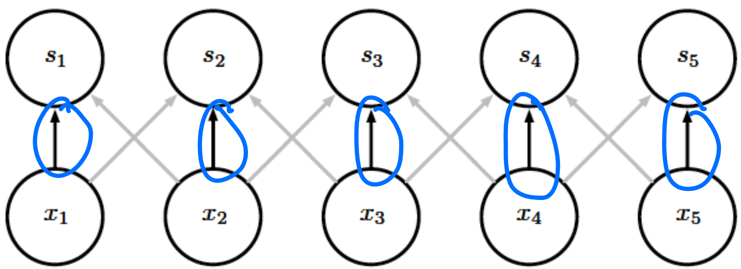
Parameter Sharing

- **Weights (Kernel)** need to be **learned** using training data and **backpropagation**

- Parameter sharing reduce the number of weights that need to be learned by sharing the same parameter (weights) across hidden units (spatial locations) at same level

Dark lines use same parameters

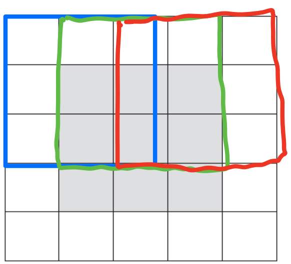
Detecting it anywhere in the image



- Learning **pattern** at one window of input, convnet can **recognize it anywherein the image**
  - Invariant to translation

Output of Convolution

- Output dimension can be different from input, that depends on Kernel size and the Stride (step)

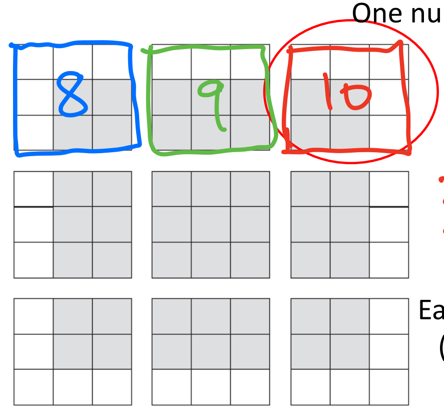


5x5 input feature map

Convolution with
3x3 Kernel

Stride =1

One number in output



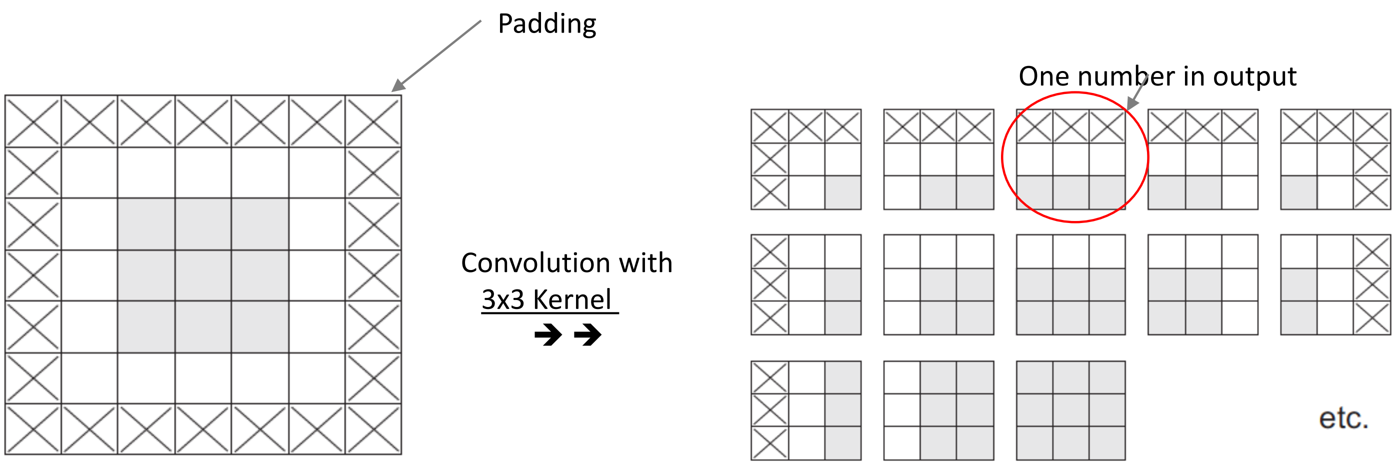
3x3 output

Each output dimension =
(input – kernel size)/stride + 1

$$\frac{(5-3)}{1}+1 \quad 3$$
$$= \quad \times$$
$$\frac{(5-3)}{1}+1 \quad 3$$

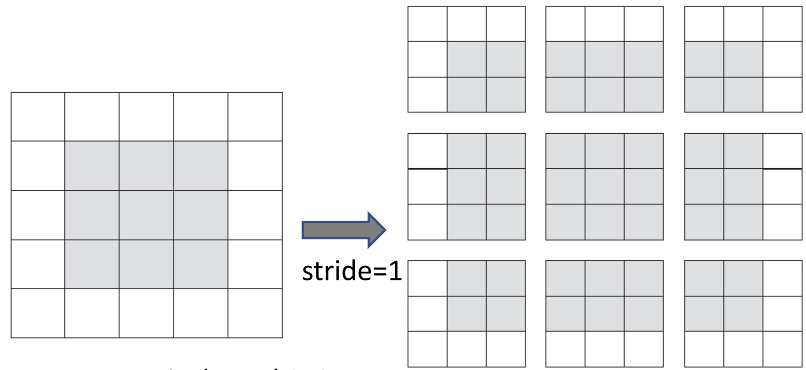
Padding

- We can add padding to make output size equals to input size



Stride will have effect of downsampling

- Stride: distance between two successive windows for convolution
  - Set to 1 by default

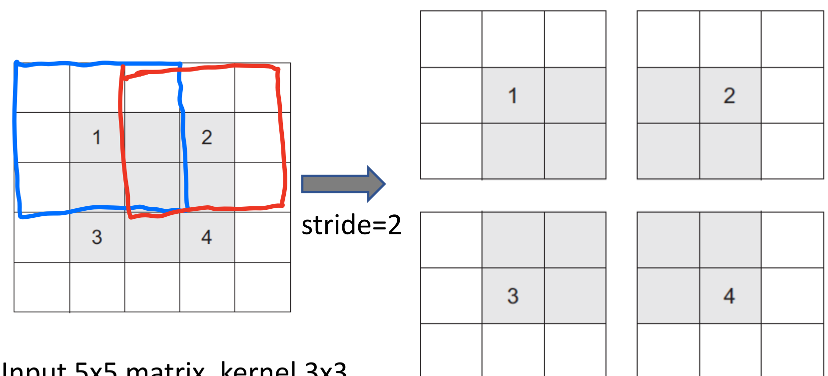


Input 5x5 matrix, kernel 3x3

Output 3x3 matrix

- Using stride **s** means the width and height of the feature map are downsampled by s
- Output dimension = (input – kernel size)/stride + 1

Stride is both horizontal and vertical

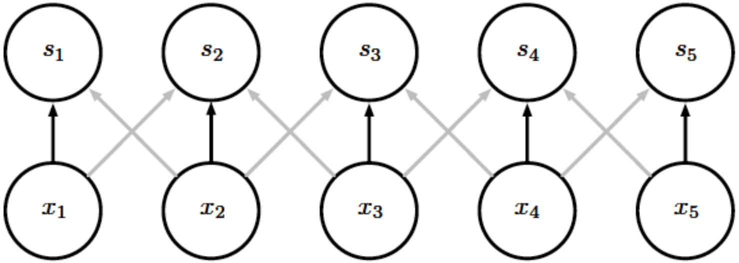


Input 5x5 matrix, kernel 3x3

Output 2x2 matrix

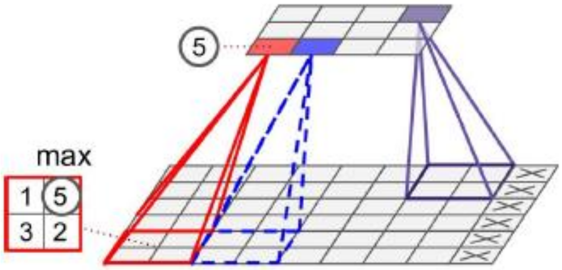
Activation function applied after Kernel at Conv. layer

- Convolutional layer: apply kernel, then pass result to activation function (e.g. Relu or tanh) to get the output



Pooling Layer

- A way to down-sample the output of a layer is through pooling
  - Also makes output invariant to small changes



Reference: Hands-On Machine Learning with Scikit-Learn and TensorFlow, Géron

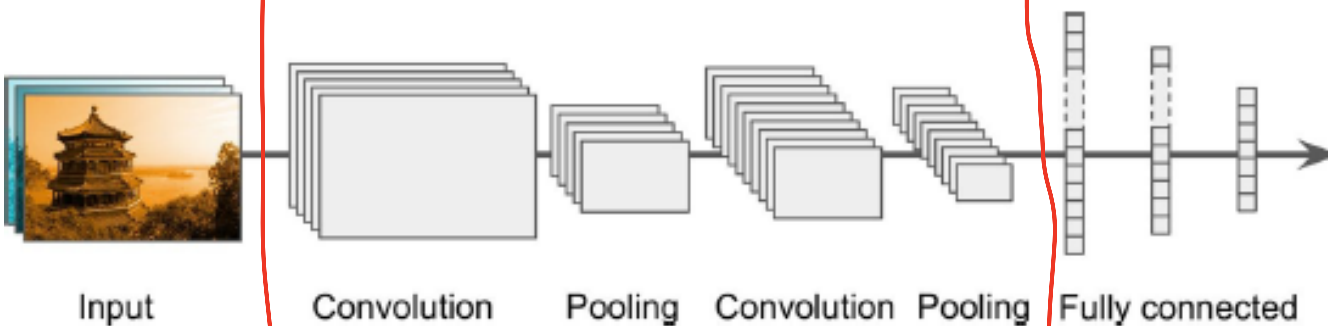
- Max pooling: output is the max value of the input

- Other techniques: e.g., average pooling

CNN Architecture

Feature extraction

Reference: Hands-On Machine Learning with Scikit-Learn and TensorFlow, Géron



Convolutional layer: apply kernel, then pass result to activation function (e.g. Relu or tanh) to get the output

3D Tensors of Convolution

- Operation of convolution is over 3D tensor

Tensor : multidimensional data/array

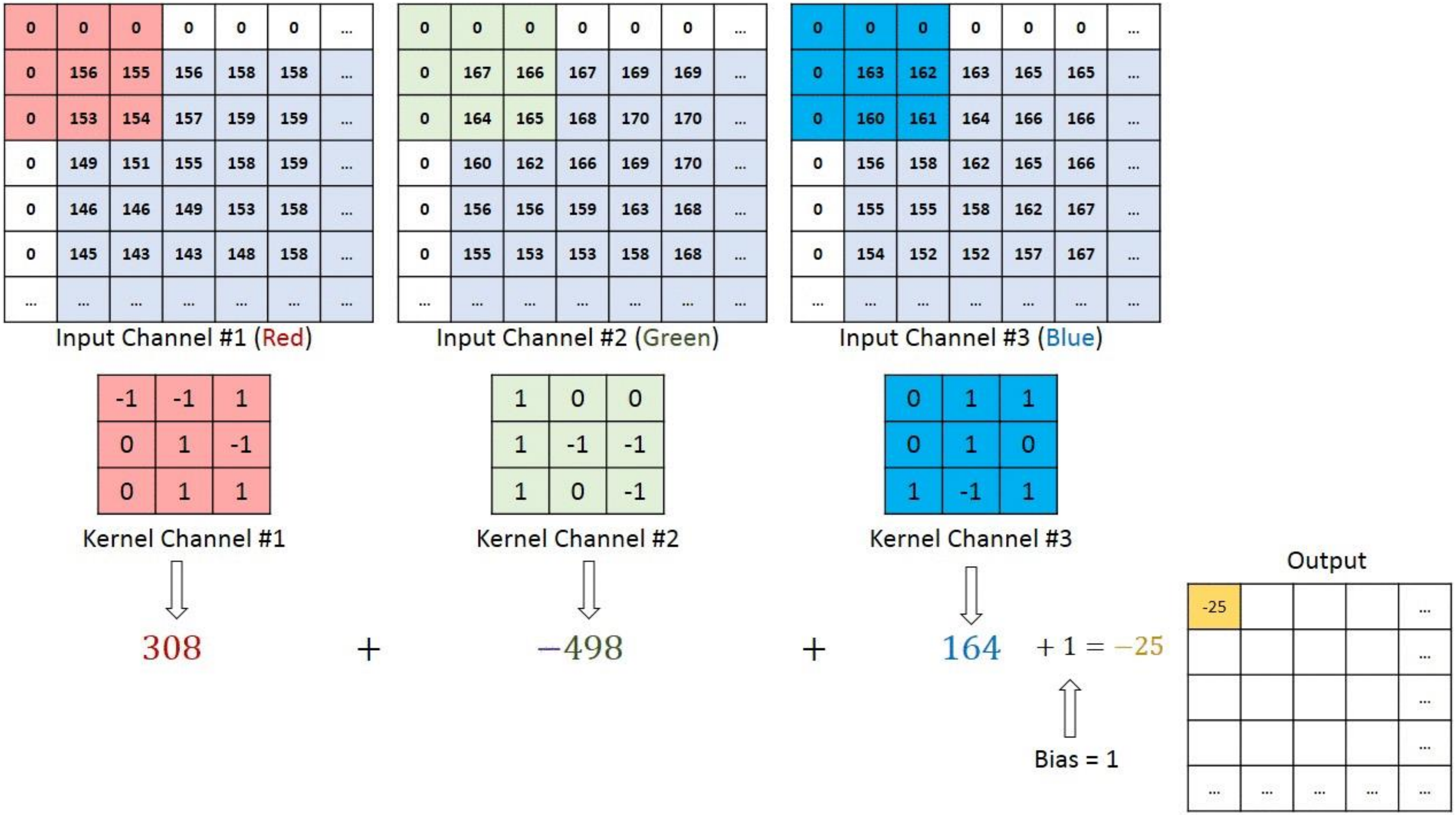
  - Two spatial axis: **height** and **width**

  - **Depth** axis, also called channels

    - For input:
      - RGB images have 3 channels corresponding to the three colors (Red, Green and Blue)
      - White and black images have depth of 1

    - For hidden layers
      - Depth corresponds to the number of filters/kernels (not color as in the input)
      - Depth is a parameter of the layer and can be set

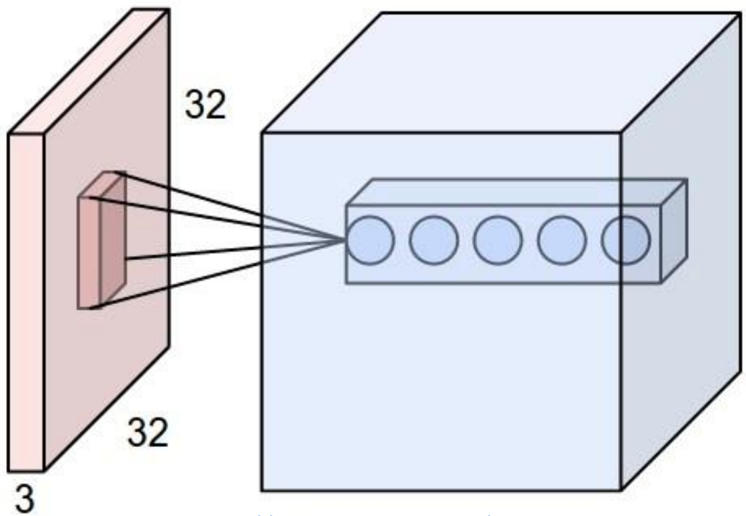
Depth at input



Depth at hidden layer:Many Kernels are Applied

- At each layer of feature extraction phase, many kernels are learned and applied to small grid of input

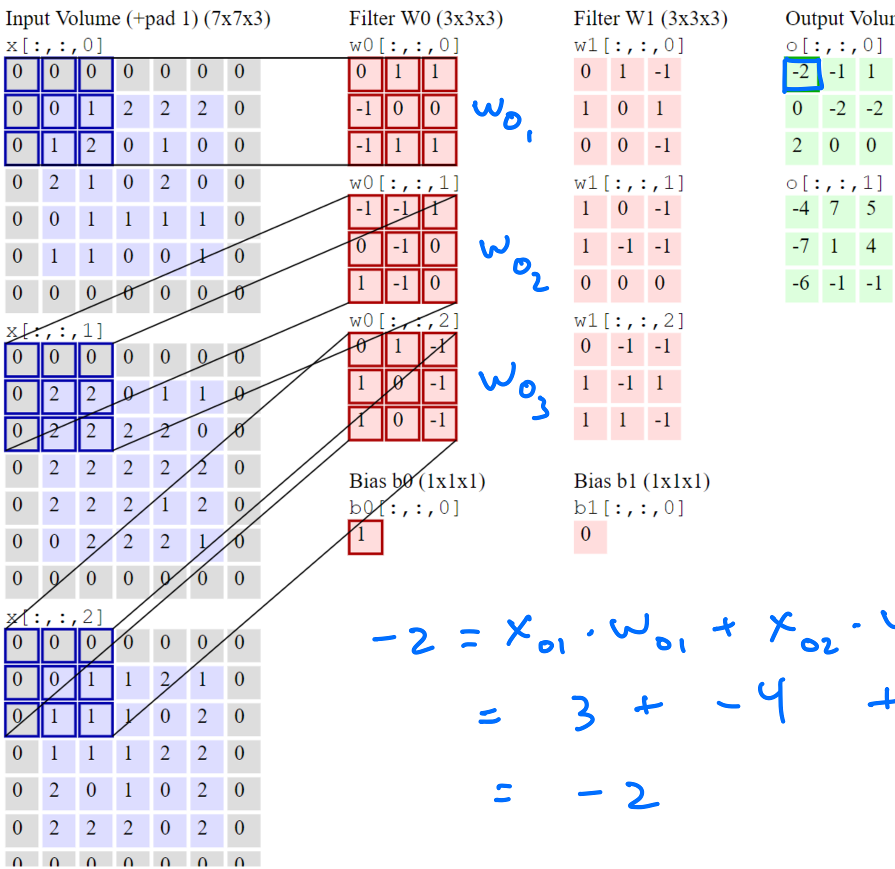
  - Number of kernels at each layer is a parameter that needs to be set
    - This is the depth



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

A conv layer with depth 2, filter 3x3, and stride 2

2 kernels (depth)



x_{01} x_{02} x_{03}

Input depth = 3

w_{01} w_{02} w_{03}

Comes from first kernel (w0)

Comes from second kernel (w1)

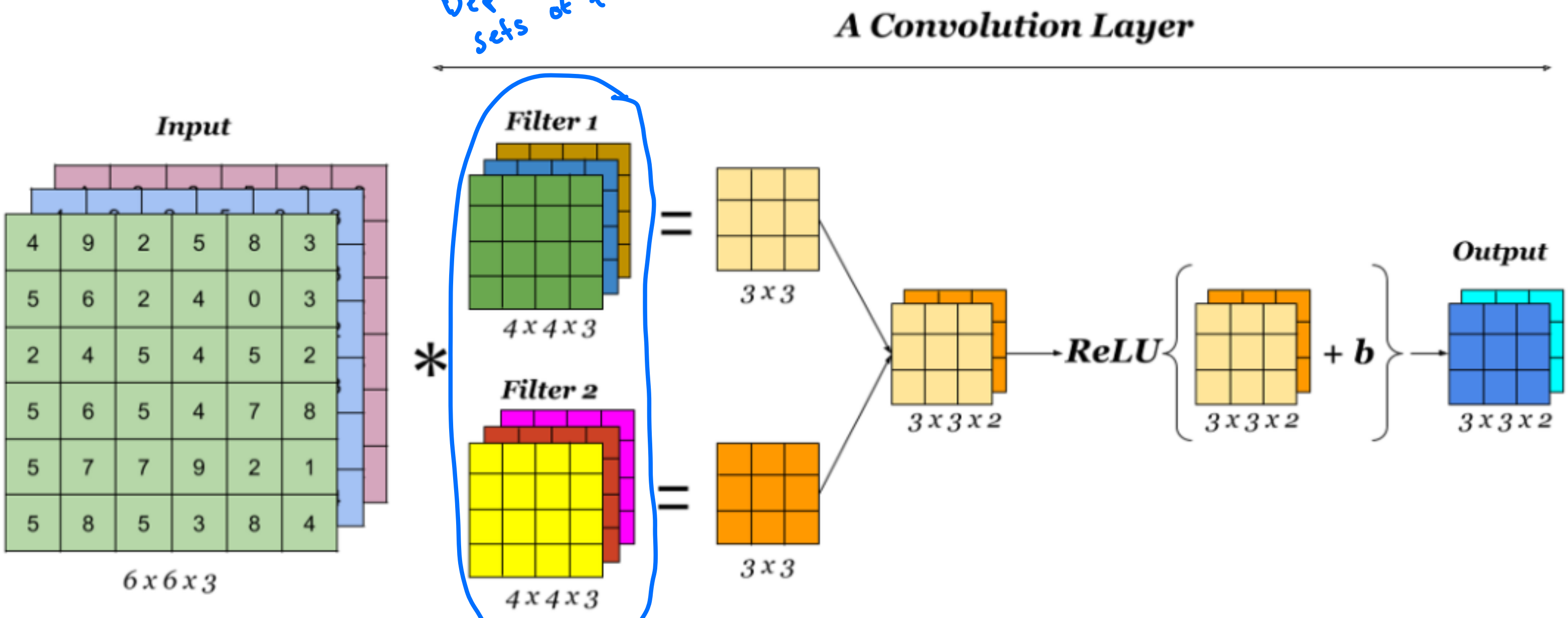
3x3x2 output

$$-2 = x_{01}\cdot w_{01} + x_{02}\cdot w_{02} + x_{03}\cdot w_{03} + b_0$$
$$= 3 + -4 + -2 + 1$$
$$= -2$$

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

One conv layer – after convolution apply activation

Depth = 2 : 2 sets of filters



Demo: https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/

$2 \cdot (4 \times 4 \times 3) + 2 = 98$ total parameters

Defining the model with Keras - Python

- It has a user-friendly API that makes it easy to quickly prototype deep-learning models

- Use TensorFlow backend

- Define **sequential** class model
  - Most common
  - **Sequential stack of layers**
  - Another class is the functional class for defining arbitrary connections

Reference: Deep Learning with Python, by Chollet

Install:
- conda install -c conda-forge keras
- pip install keras, pip install tensorflow, pip install keras.utils

Handwritten Digit Recognition Example -- Define the model

Conv – Pooling – Conv – Pooling – Conv

Feature extraction phase

Check Keras documentation for details: https://keras.io/layers/convolutional/

How many kernels in layer – How many different patterns do you want to detect

Depth

Kernel size

```
from keras import layers
from keras import models

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
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'))
```

Black & white image (depth = 1) of size 28 x 28

```
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

Prediction Phase

Reference: Deep Learning with Python, by Chollet

Model.summary

- Model.summary()

```
from keras import layers
from keras import models

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
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.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 3, 3, 64)          36928
_________________________________________________________________
flatten_1 (Flatten)          (None, 576)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                36928
_________________________________________________________________
dense_2 (Dense)              (None, 10)                650
=================================================================
Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0
```

$32(3\times3) + 32 = 320$

max pooling

input depth = 32

$[64(3\times3)]\cdot 32 + 64$

$3\times3\times64$

Prediction phase (Fully connected network)

$576\cdot64 + 64$

$64\cdot10 + 10$

Handwritten Digit Recognition Example -- Apply to data

Reference: Deep Learning with Python, by Chollet

```
from keras.datasets import mnist
from keras.utils import to_categorical

(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

train_images = train_images.reshape((60000, 28, 28, 1))
train_images = train_images.astype('float32') / 255

test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype('float32') / 255

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(train_images, train_labels, epochs=5, batch_size=64)

test_acc = model.evaluate(test_images, test_labels)
```

Dataset of 60,000 28x28 grayscale images of the 10 digits, test set of 10,000 images.

using model architecture from above slide

Compile: Configure the model, prepare for training, define loss function (objective function to minimize), optimizer, and can specify the metric to print while training .

https://keras.io/api/models/model_training_apis/

Can add a dropout layer

- https://keras.io/api/layers/regularization_layers/dropout/
  - tf.keras.layers.Dropout(rate, ...)
    - Rate < 1
      - Set input to 0 with that rate during training