Convolutional Neural Networks (CNN)

Sangseung Lee
Inha Univ.

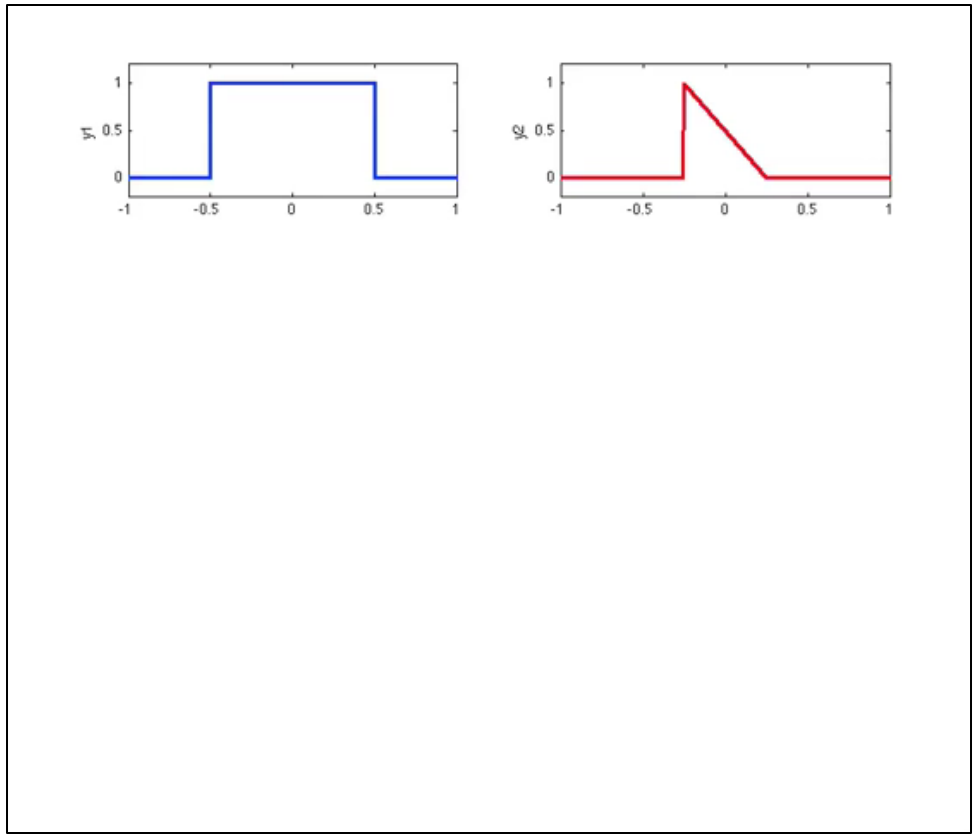
*Based on Prof. Seungchul Lee's (KAIST) lecture note

Convolution

Convolution

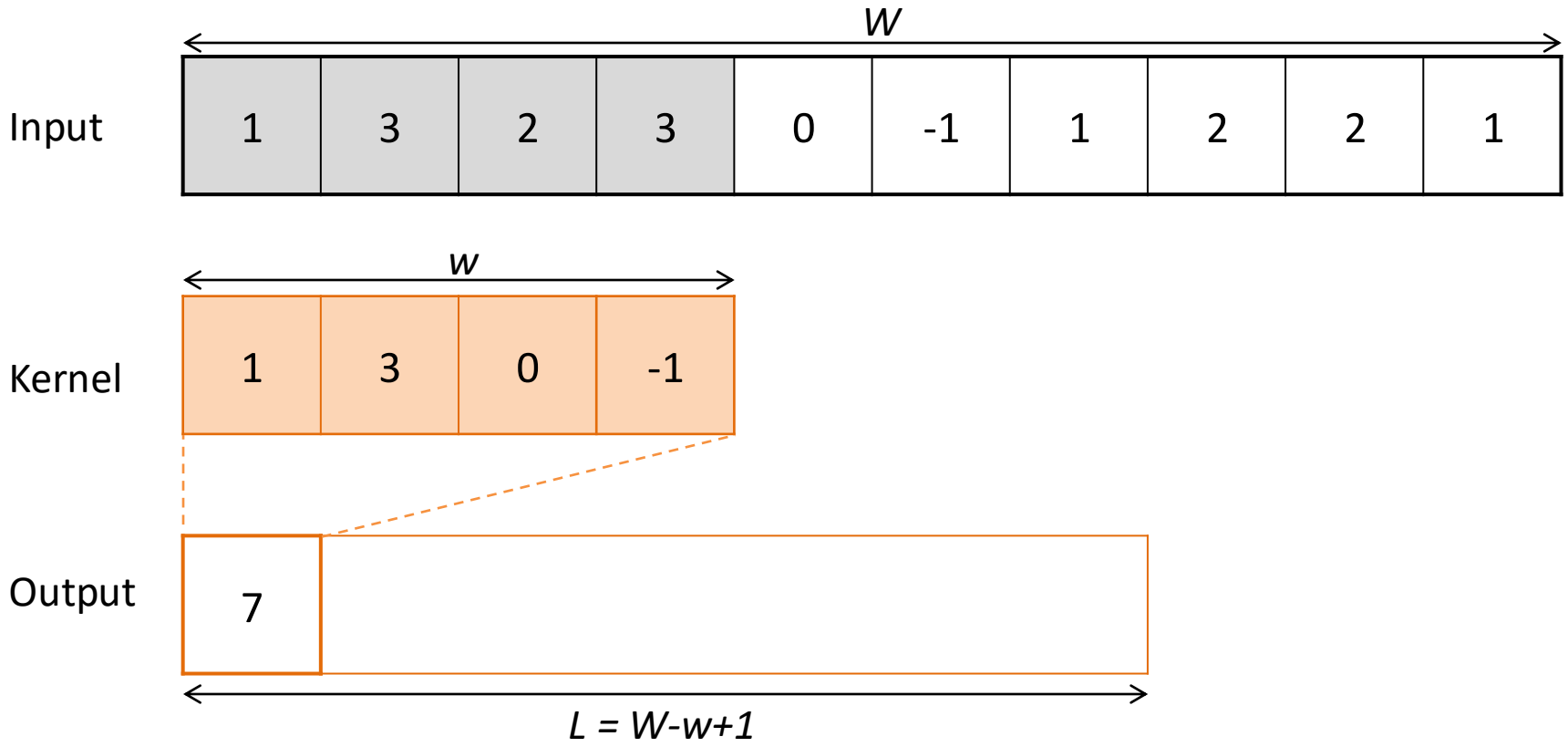
- Integral (or sum) of the product of the two signals after one is reversed and shifted
- Cross correlation and convolution

$$y[n] = \sum_{m=-\infty}^{\infty} h[n-m]\,x[m] = x[n] * h[n]$$



1D Convolution

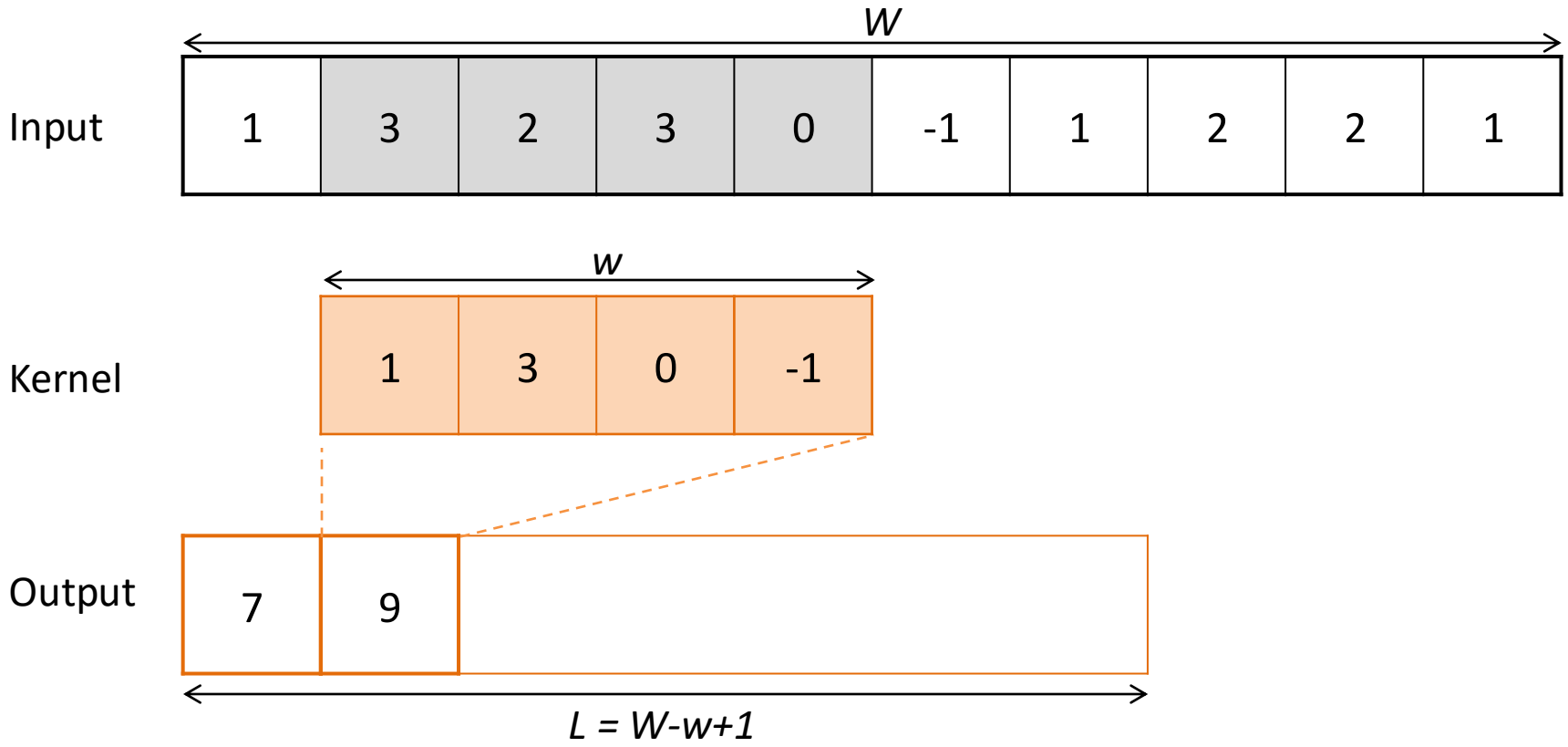
- (actually cross-correlation)



Source: Dr. Francois Fleuret at EPFL

1D Convolution

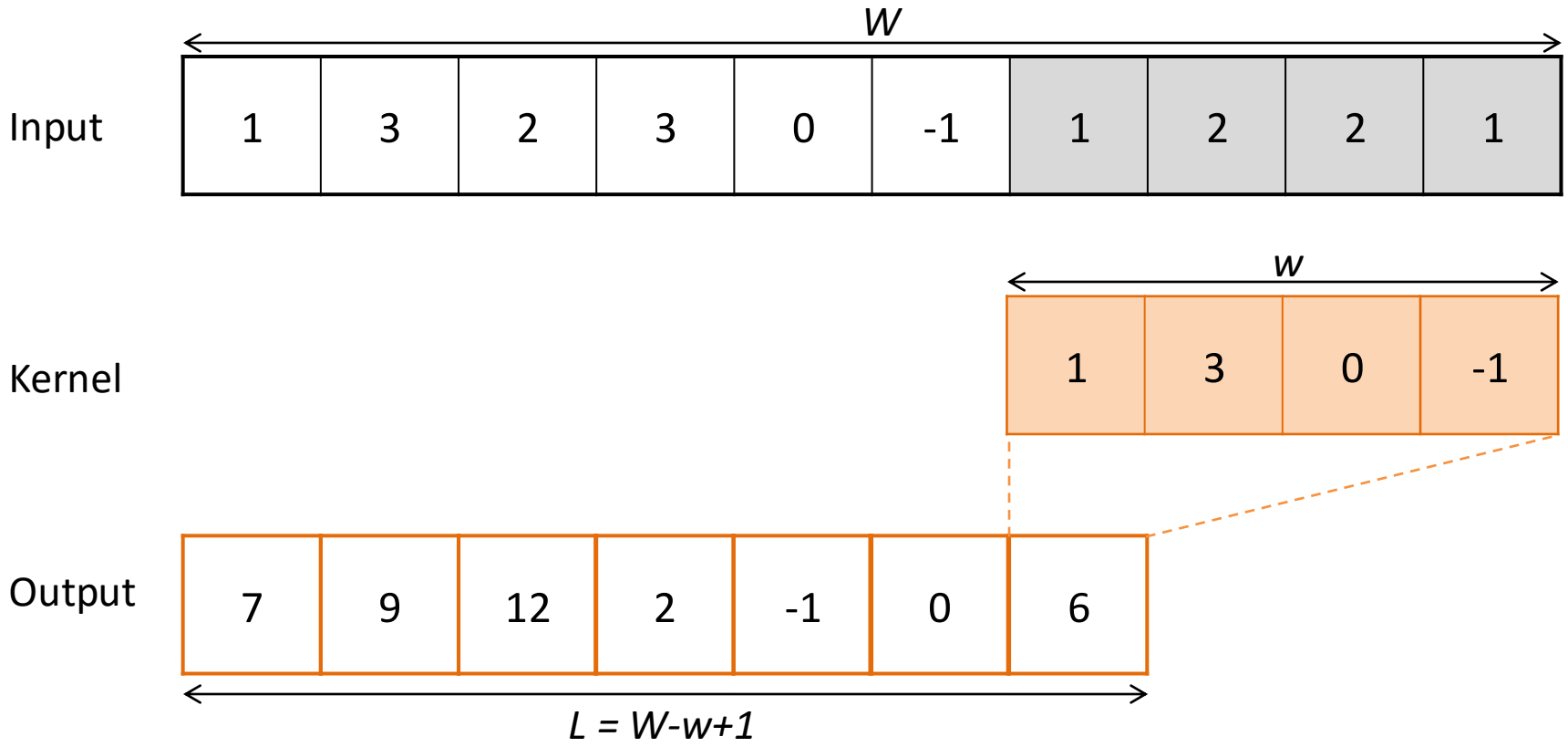
- (actually cross-correlation)



Source: Dr. Francois Fleuret at EPFL

1D Convolution

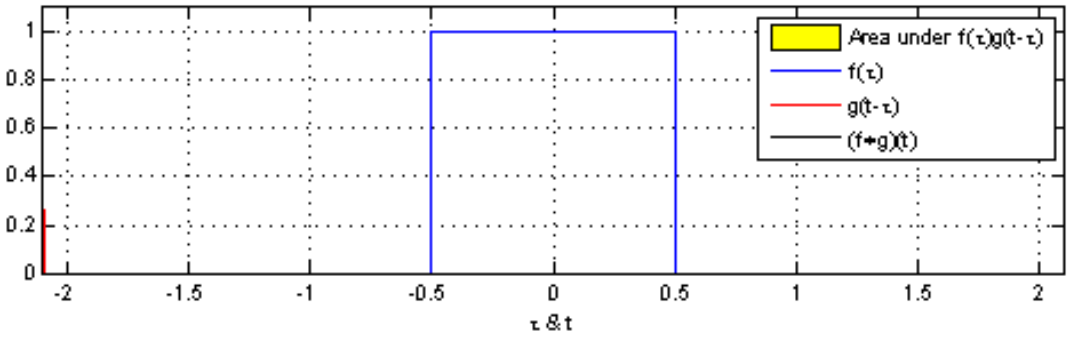
- (actually cross-correlation)

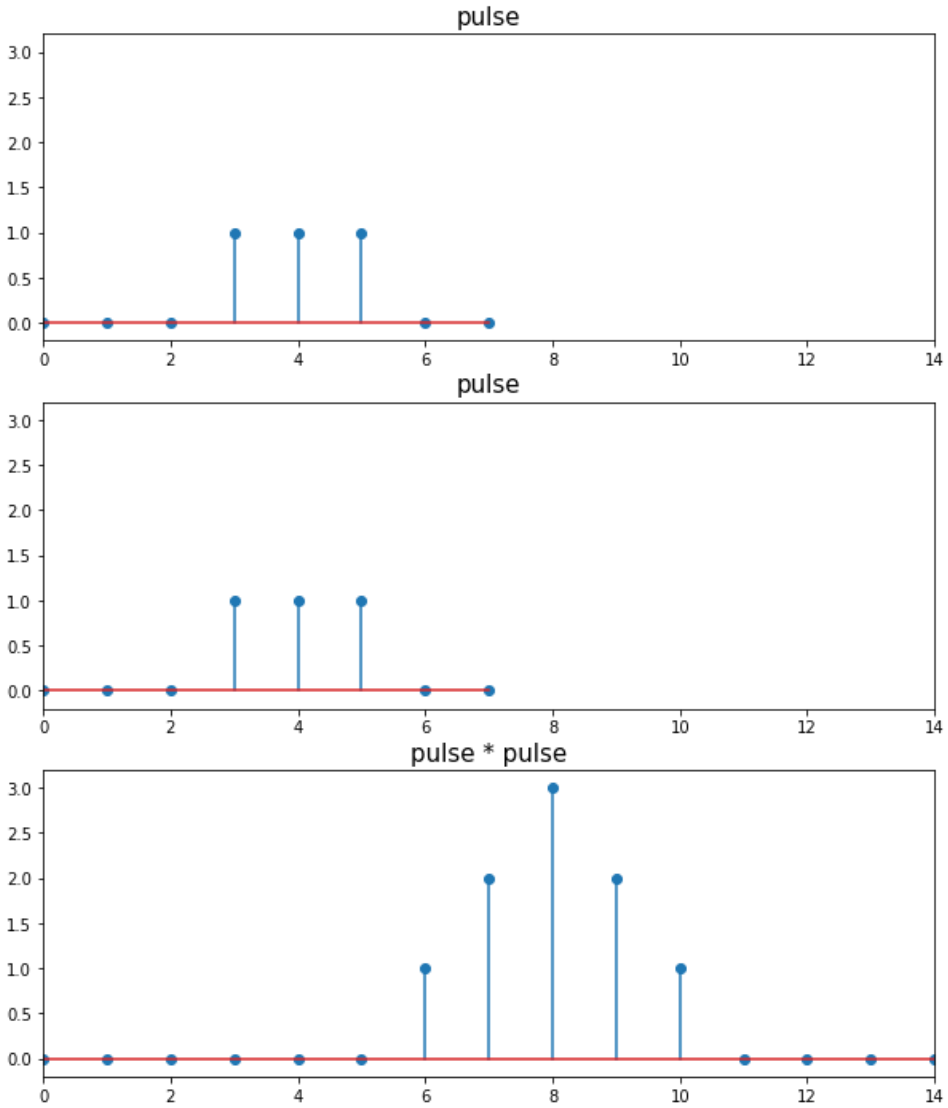


Source: Dr. Francois Fleuret at EPFL

Example: 1D Convolution

```
# pulse
N = 8
n = np.arange(N)
x = [0, 0, 0, 1, 1, 1, 0, 0]

# Convolve pulse with itself
y = np.convolve(x, x)
```





De-noising a Piecewise Smooth Signal

- Moving average (MA) filter
  - A moving average is the unweighted mean of the previous m data

$$\bar{x}[n] = \frac{x[n] + x[n-1] + \cdots + x[n-m+1]}{m}$$

De-noising a Piecewise Smooth Signal

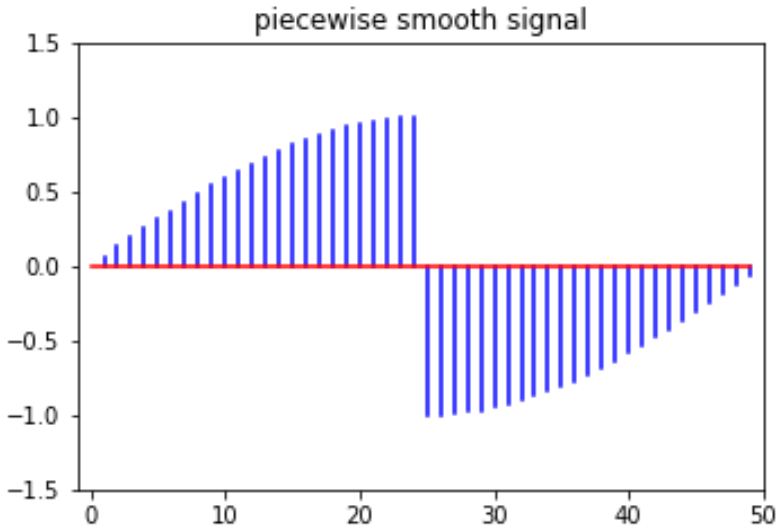
- Moving average (MA) filter
  - A moving average is the unweighted mean of the previous m data

$$\begin{aligned}\bar{x}[n] &= \frac{x[n] + x[n-1] + \cdots + x[n-m+1]}{m} \\ &= (x[n], x[n-1], \cdots, x[n-m+1]) * \left(\frac{1}{m}, \frac{1}{m}, \cdots, \frac{1}{m}\right)\end{aligned}$$

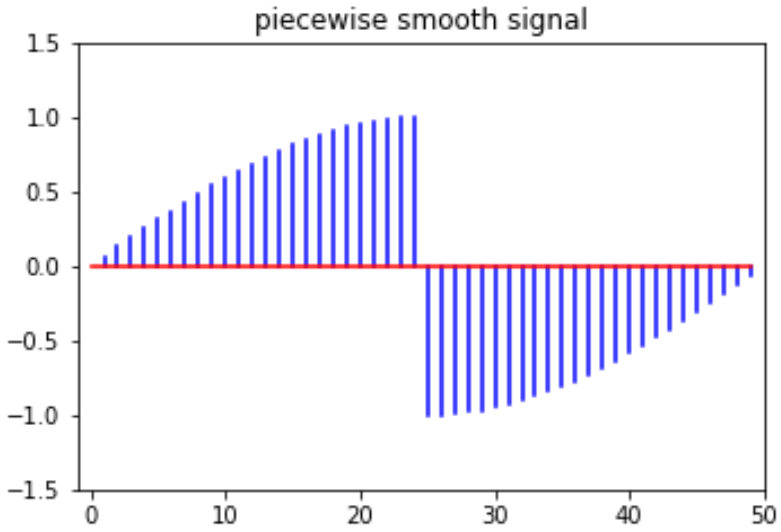
- Convolution with $\left(\frac{1}{m}, \frac{1}{m}, \cdots, \frac{1}{m}\right)$

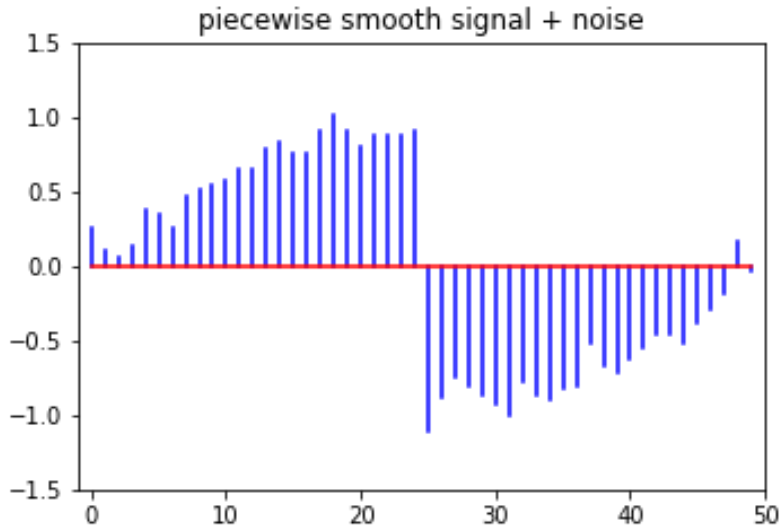
- Low-pass filter in time domain

De-noising a Piecewise Smooth Signal

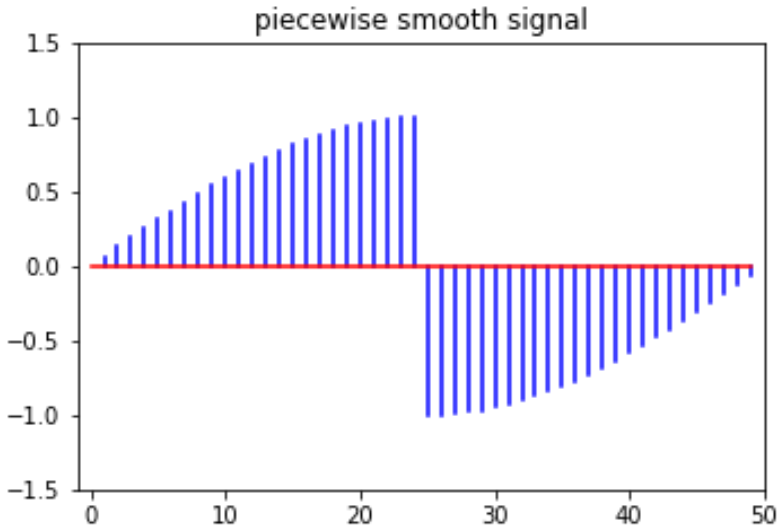


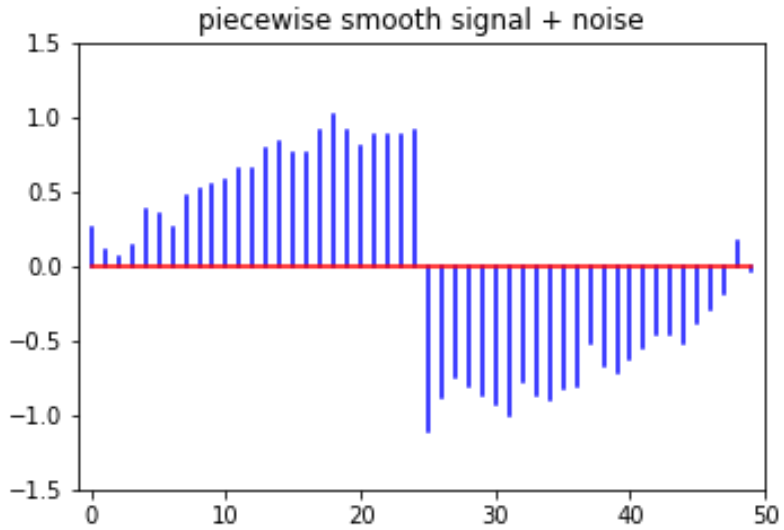
De-noising a Piecewise Smooth Signal

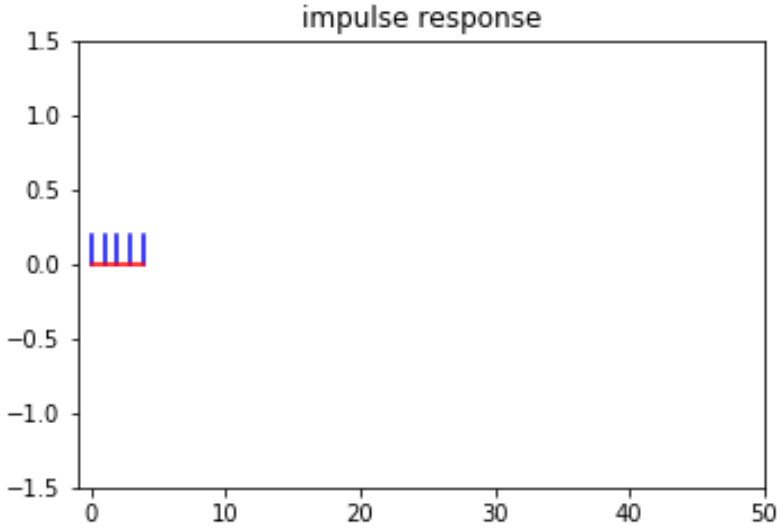




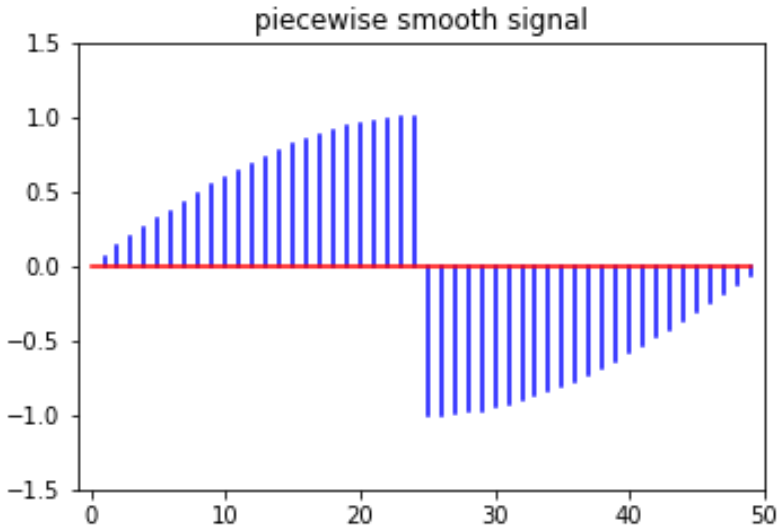
De-noising a Piecewise Smooth Signal

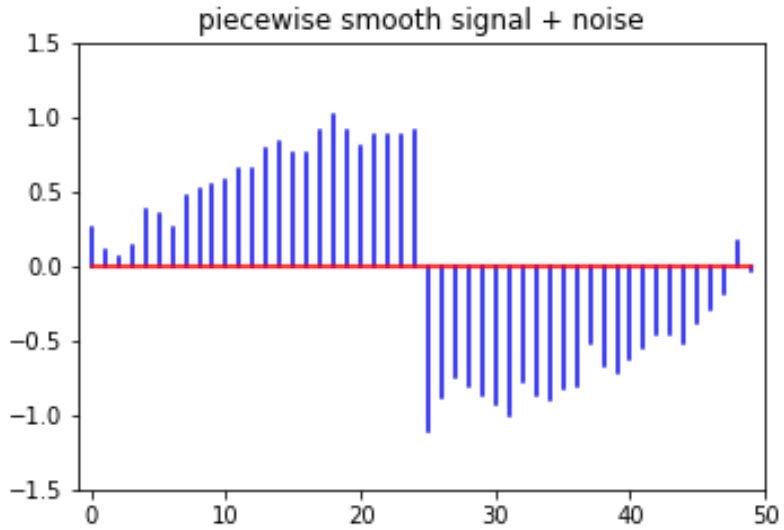


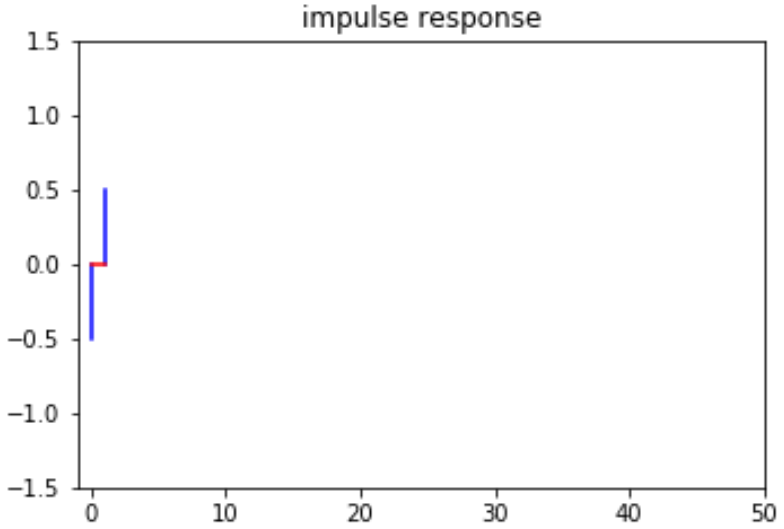




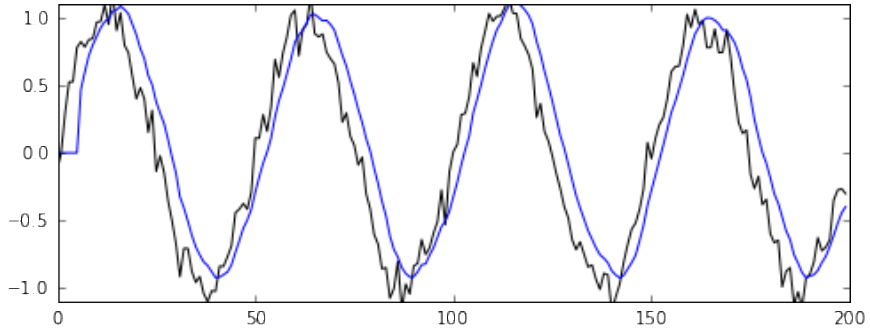
Edge Detection

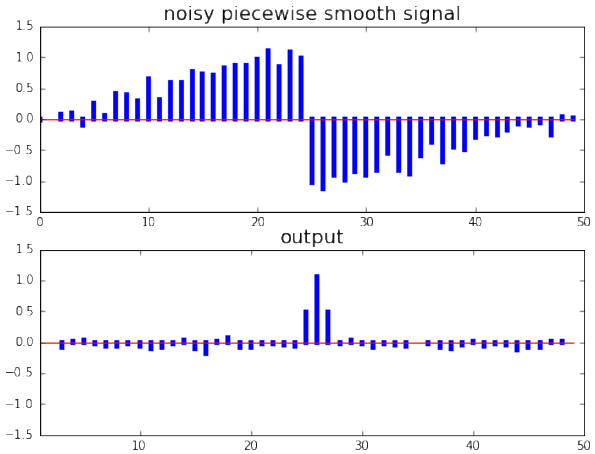


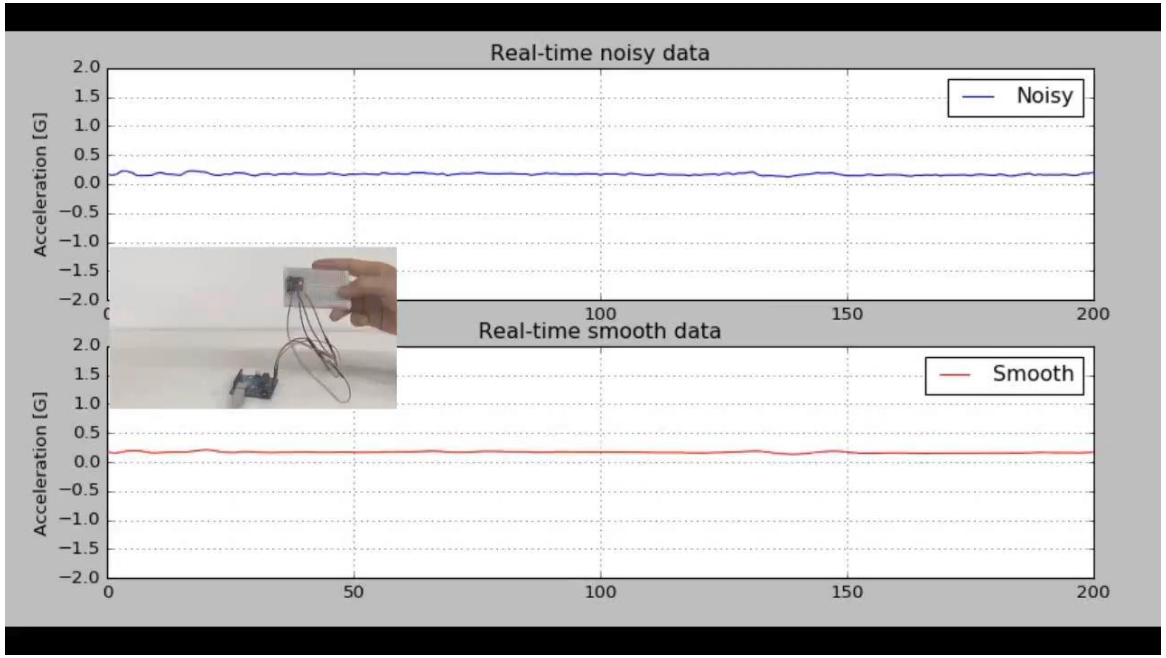


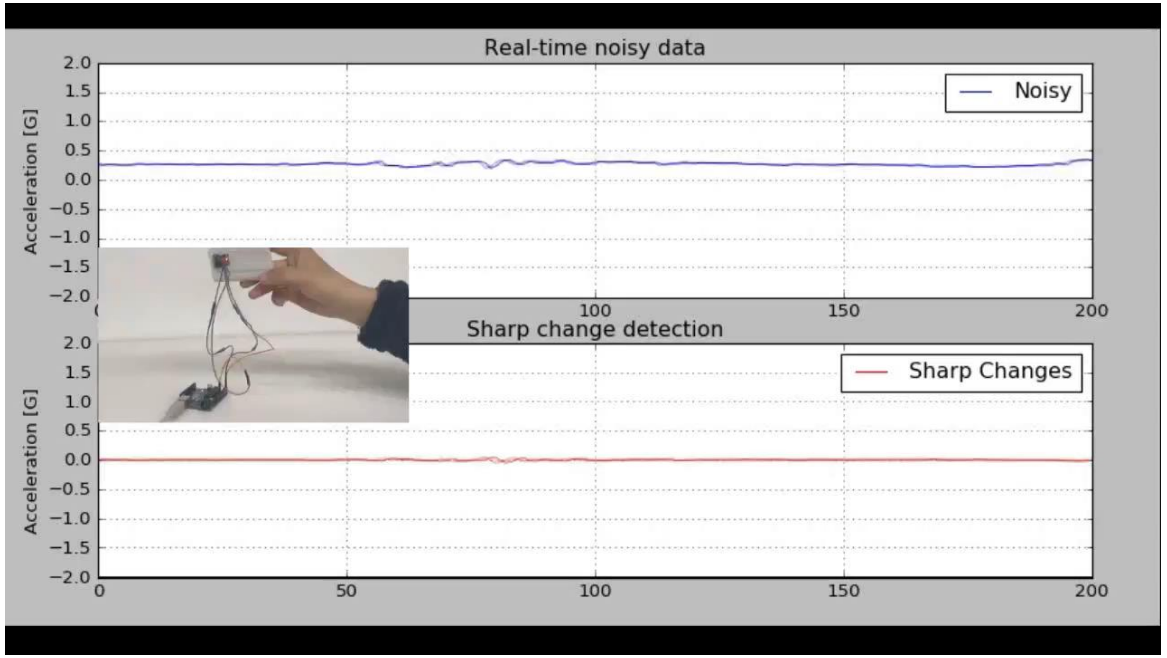


Smoothing and Detection of Abrupt Changes



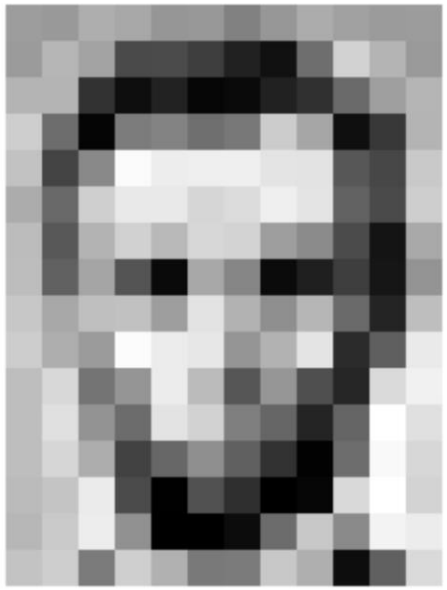






Images

Images Are Numbers



157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

What the computer sees

157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

An image is just a matrix of numbers [0,255]!
i.e., 1080x1080x3 for an RGB image

Source: 6.S191 Intro. to Deep Learning at MIT

Images

Original image

R

G

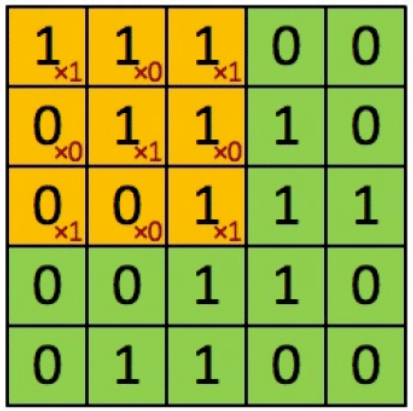
B

Gray image

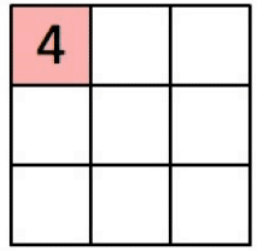
2D Convolution

Convolution on Image (= Convolution in 2D)

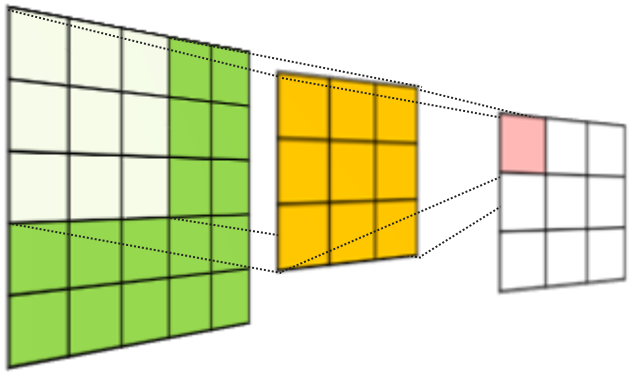
- Filter (or Kernel)
  - Discrete convolution can be viewed as **element-wise multiplication** by a matrix
  - Modify or enhance an image by filtering
  - Filter images to emphasize certain features or remove other features
  - Filtering includes smoothing, sharpening and edge enhancement



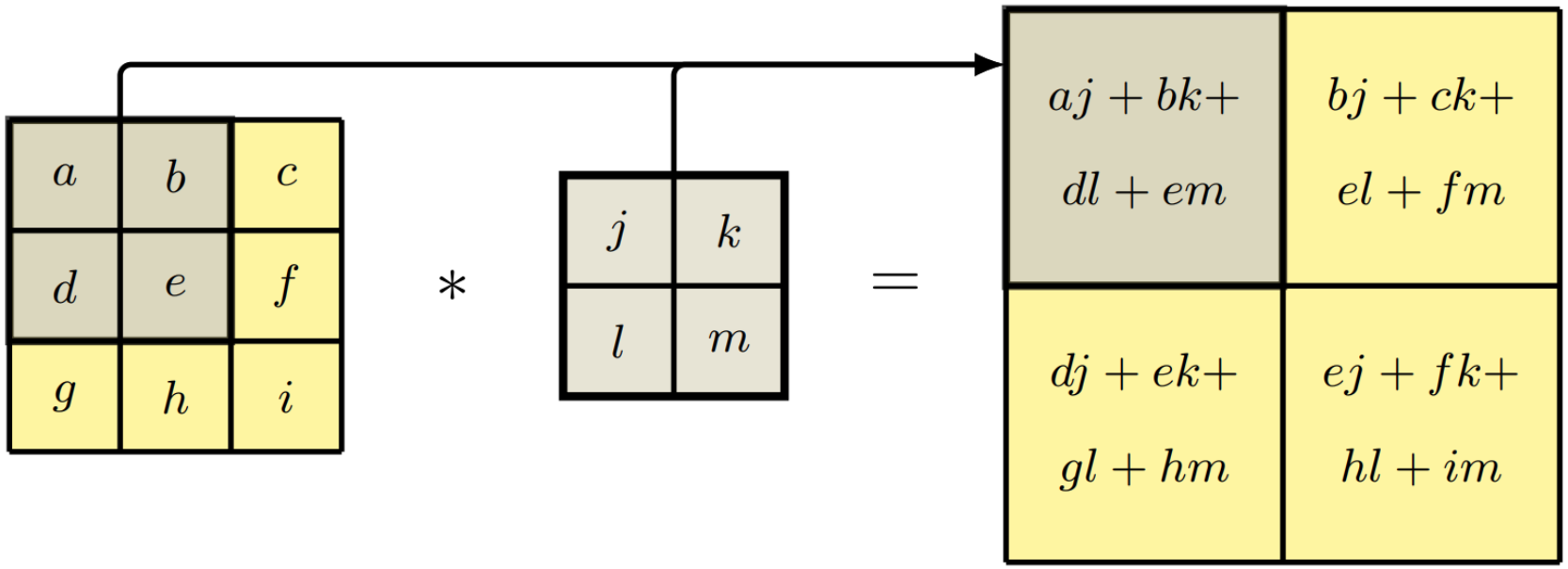
Image



Convolved Feature

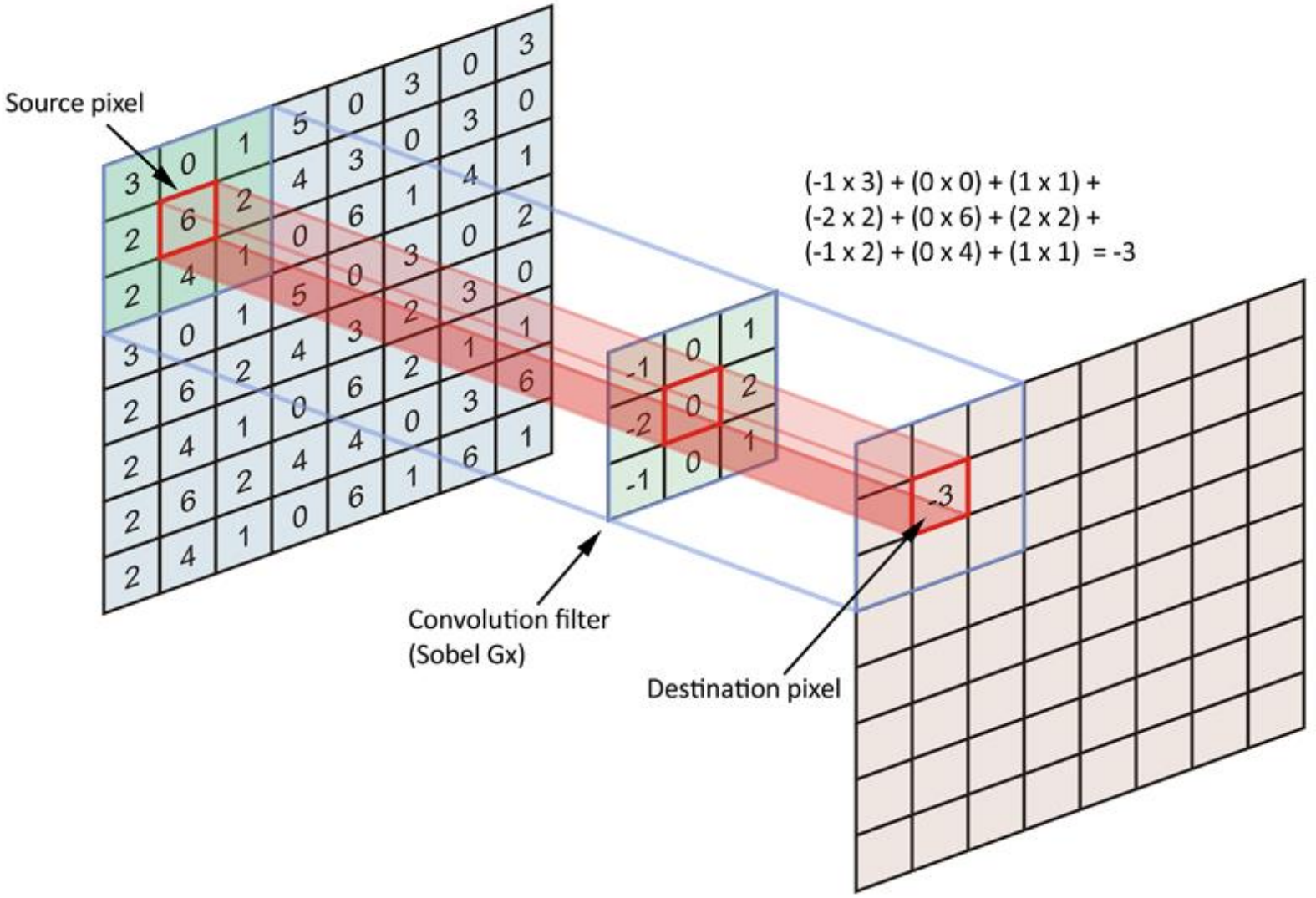


Convolution on Image (= Convolution in 2D)



Source: Deep Learning Foundations and Concepts

Convolution on Image (= Convolution in 2D)



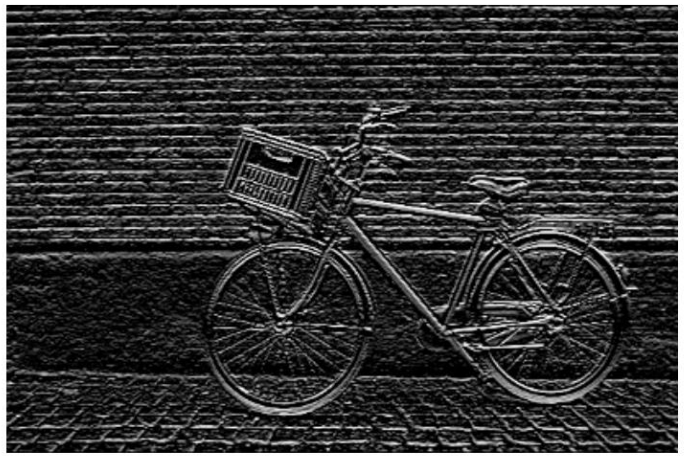
Convolution on Image

$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

Image

Convolution on Image



Source: Deep Learning Foundations and Concepts

Convolution on Image

```
M = np.ones([3,3])/9

img_conv = signal.convolve2d(img, M, 'same')
```

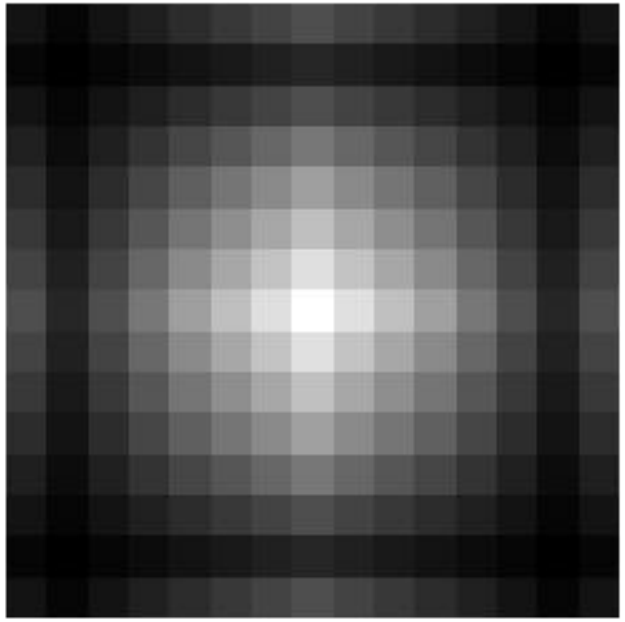
Noisy Image

Smoothed Image

Gaussian Filter: Blurring

Input image

Image filter (15 x 15)



How to Find the Right Kernels

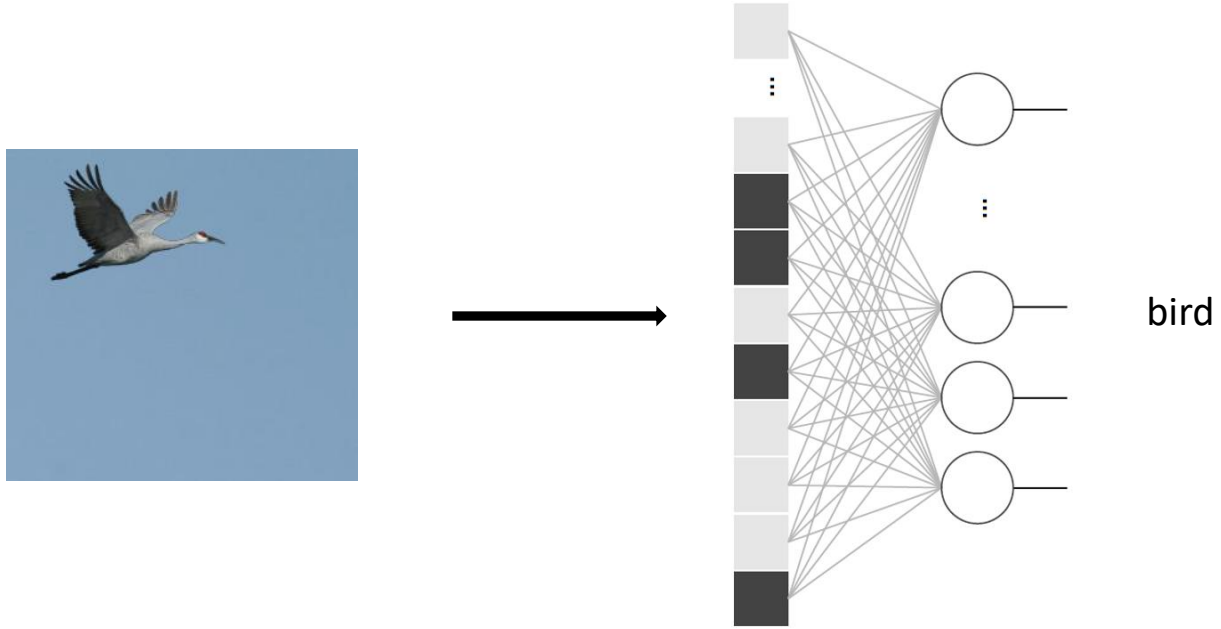
- We learn many different kernels that make specific effect on images
- Let’s apply an opposite approach
- We are not designing the kernel, but are learning the kernel from data
- Can learn feature extractor from data using a deep learning framework

Learning Visual Features

Convolutional Neural Networks (CNN)

- Motivation
  - The bird occupies a local area and looks the same in different parts of an image. We should construct neural networks which exploit these properties.

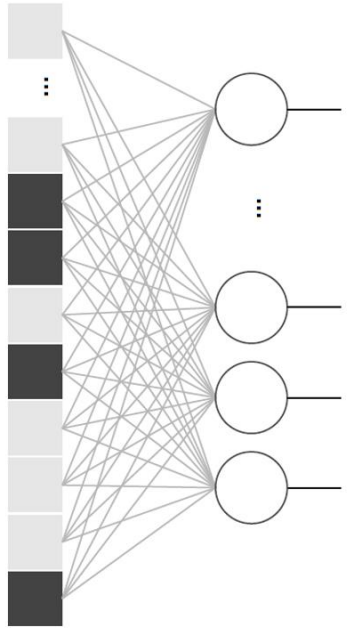
ANN Structure for Object Detection in Image



- Does not seem the best
- Did not make use of the fact that we are dealing with images

Fully Connected Neural Network

- Input
  - 2D image
  - Vector of pixel values

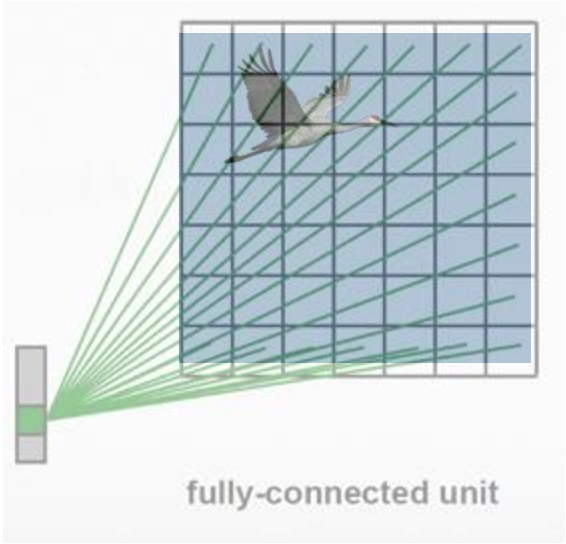


- Fully connected
  - Connect neuron in hidden layer to all neurons in input layer
  - No spatial information
  - Spatial organization of the input is destroyed by flatten
  - And many, many parameters !

- How can we use spatial structure in the input to inform the architecture of the network?

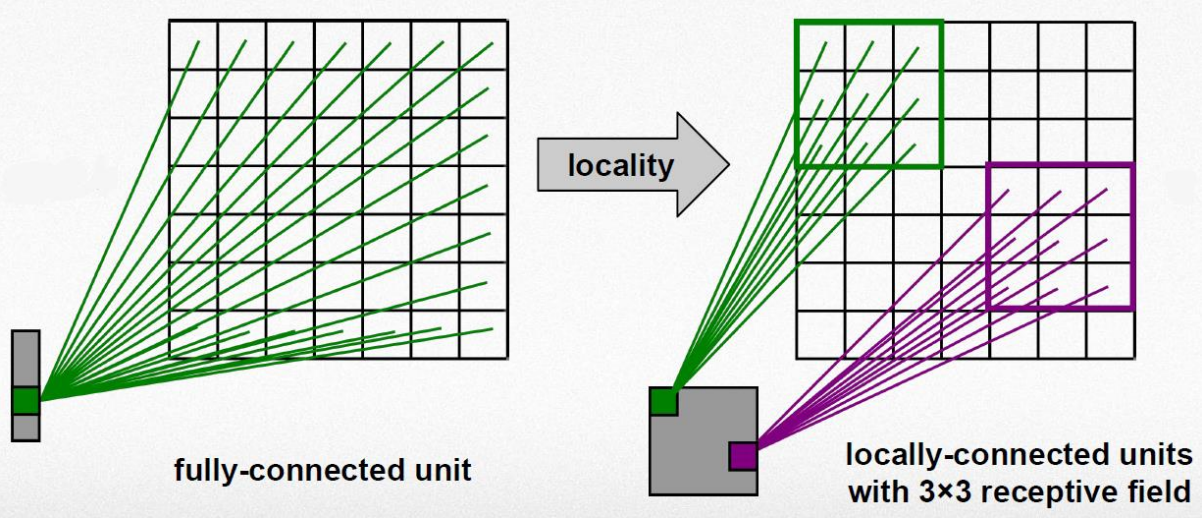
Source: 6.S191 Intro. to Deep Learning at MIT

Convolution Mask + Neural Network



Locality

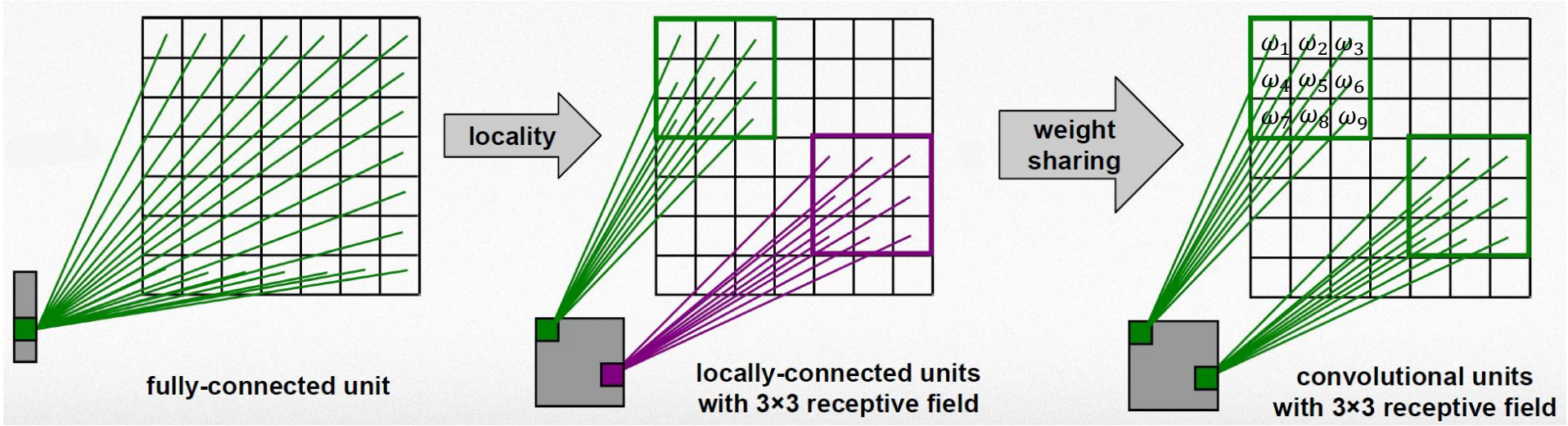
- Locality: objects tend to have a local spatial support
  - fully-connected layer → locally-connected layer



Locality

- Locality: objects tend to have a local spatial support
  - fully-connected layer → locally-connected layer

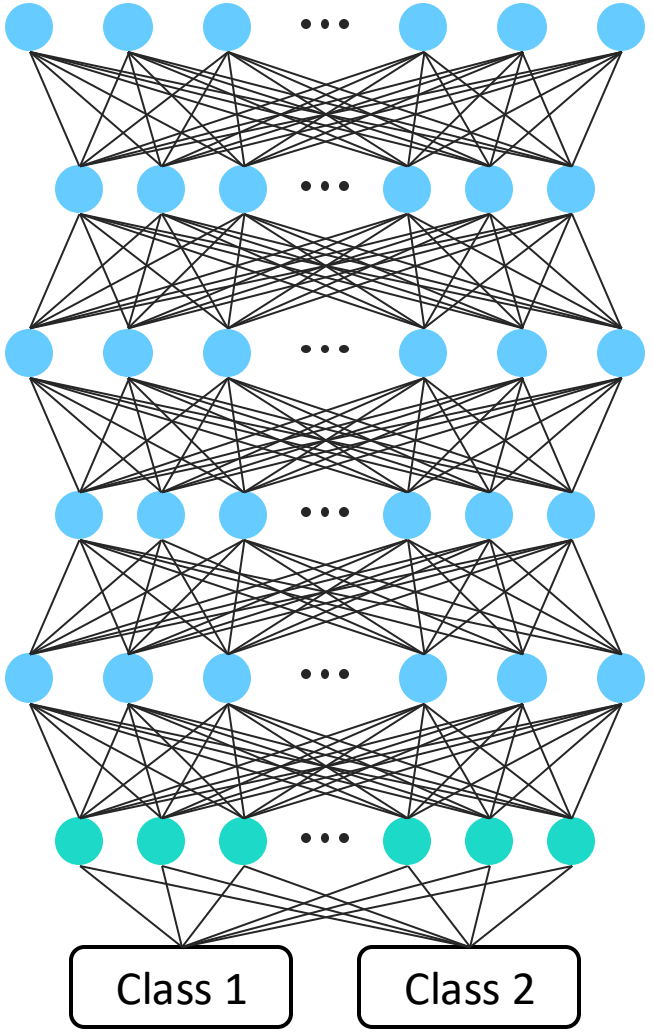
We are not designing the kernel, but are learning the kernel from data
→ Learning feature extractor from data



Deep Artificial Neural Networks

- Universal function approximator
  - Simple nonlinear neurons
  - Linear connected networks

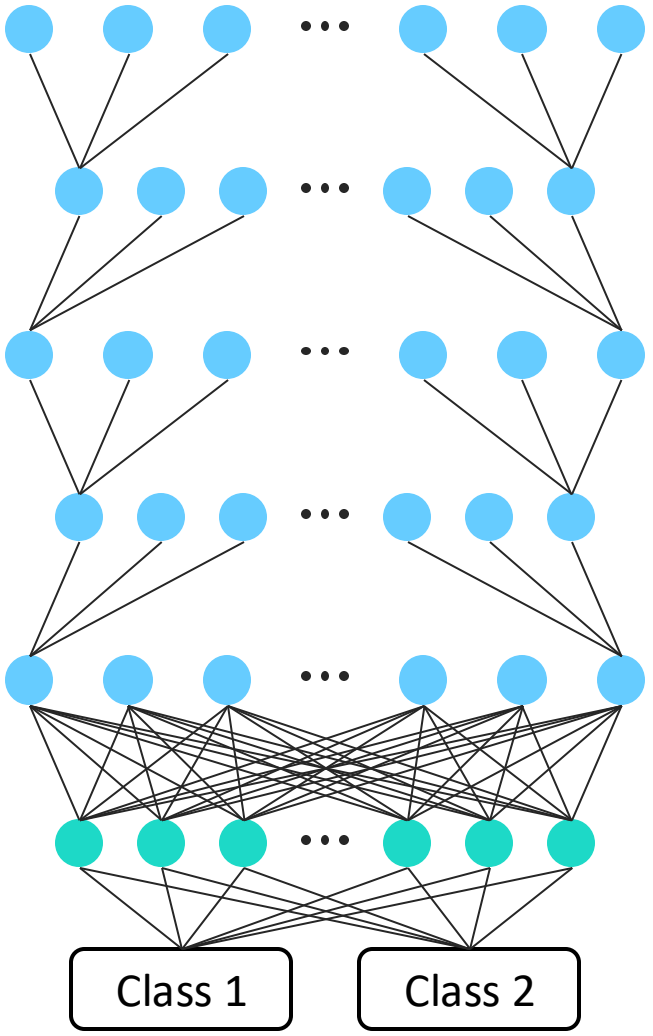
- Hidden layers
  - Autonomous feature learning



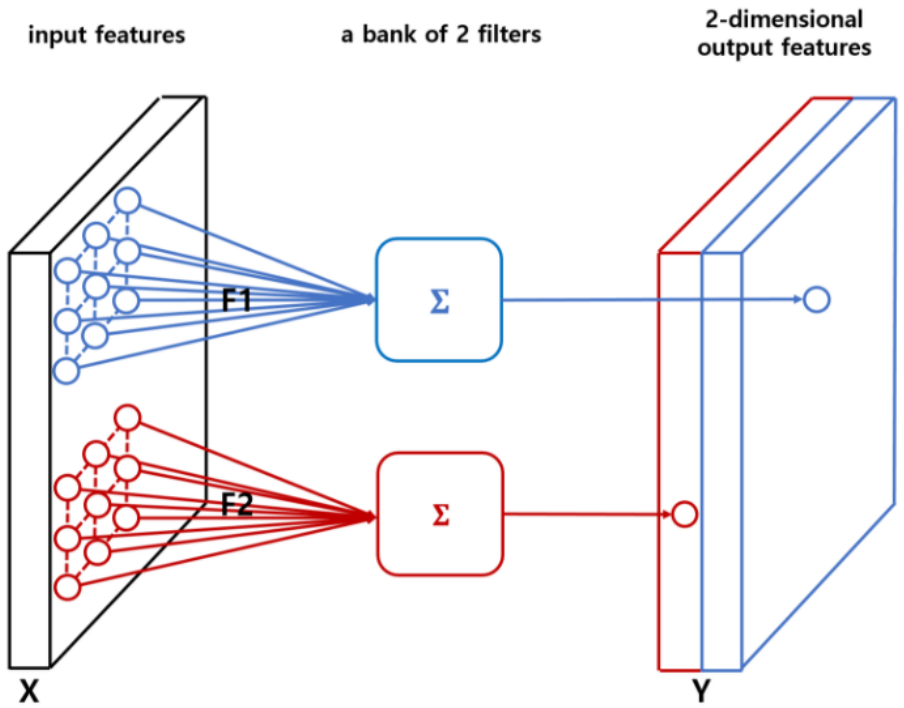
Convolutional Neural Networks

- Structure
  - Weight sharing
  - Local connectivity
  - Typically have sparse interactions

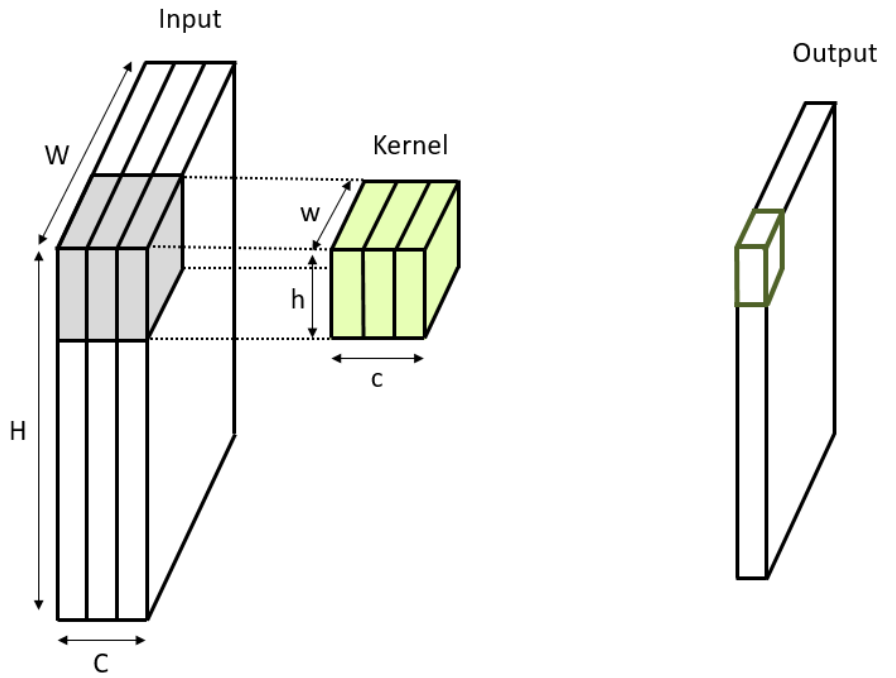
- Optimization
  - Smaller searching space



Multiple Filters (or Kernels)



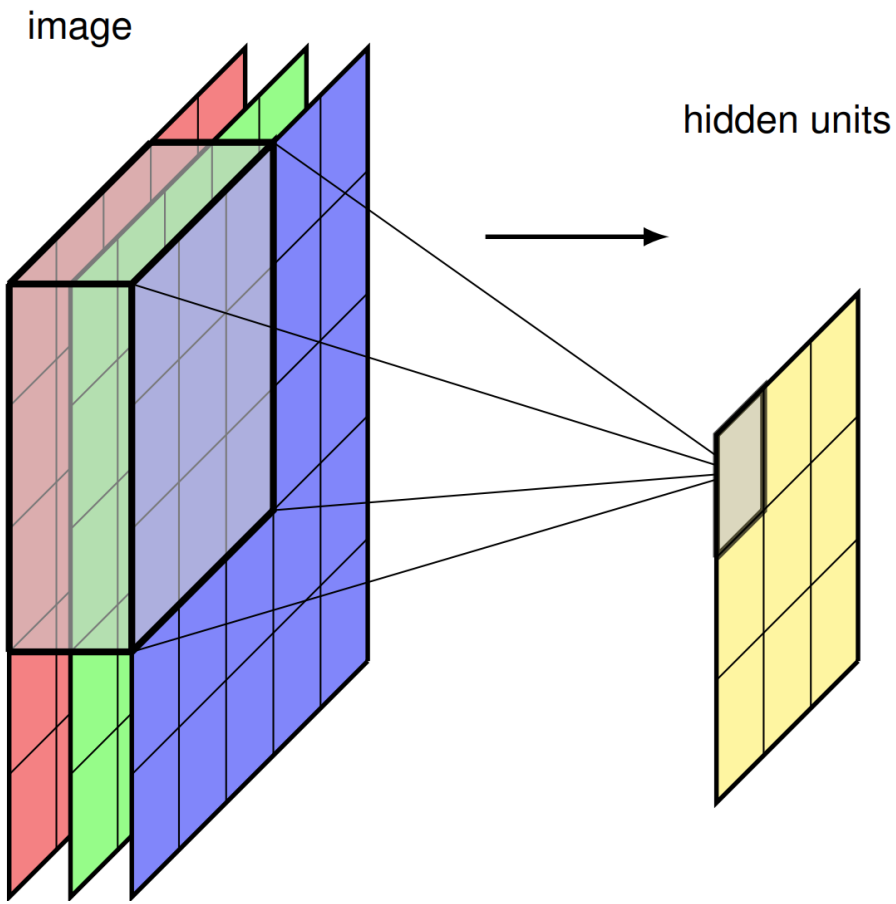
Channels

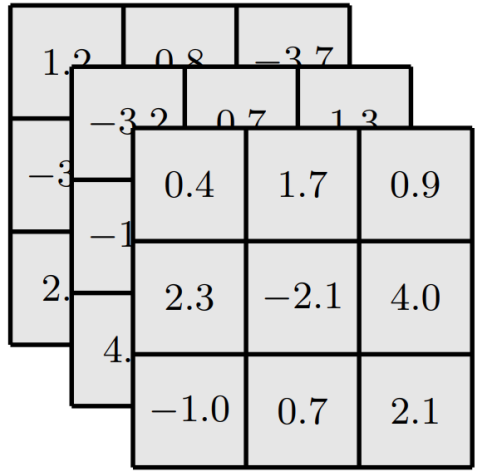


- Colored image = tensor of shape (height, width, channels)
- Convolutions are usually computed for each channel and summed:
- Kernel size aka receptive field (usually 1, 3, 5, 7, 11)

Source: Dr. Francois Fleuret at EPFL

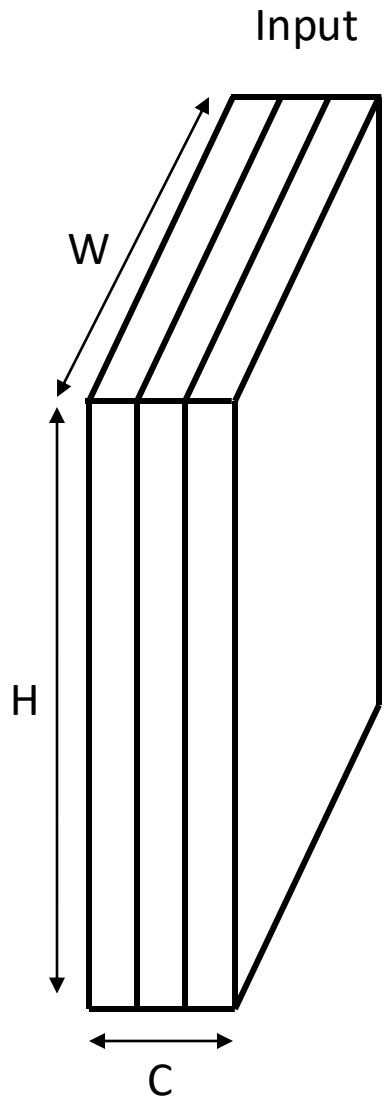
Multi-channels





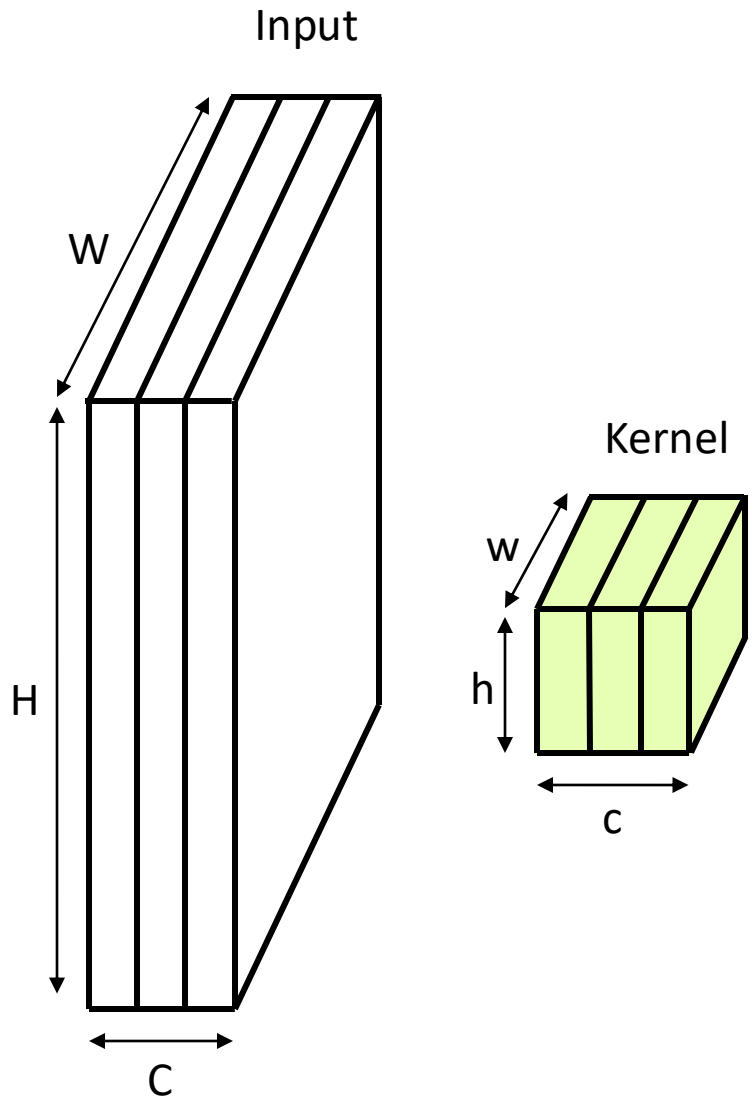
Source: Deep Learning Foundations and Concepts

Multi-channel 2D Convolution



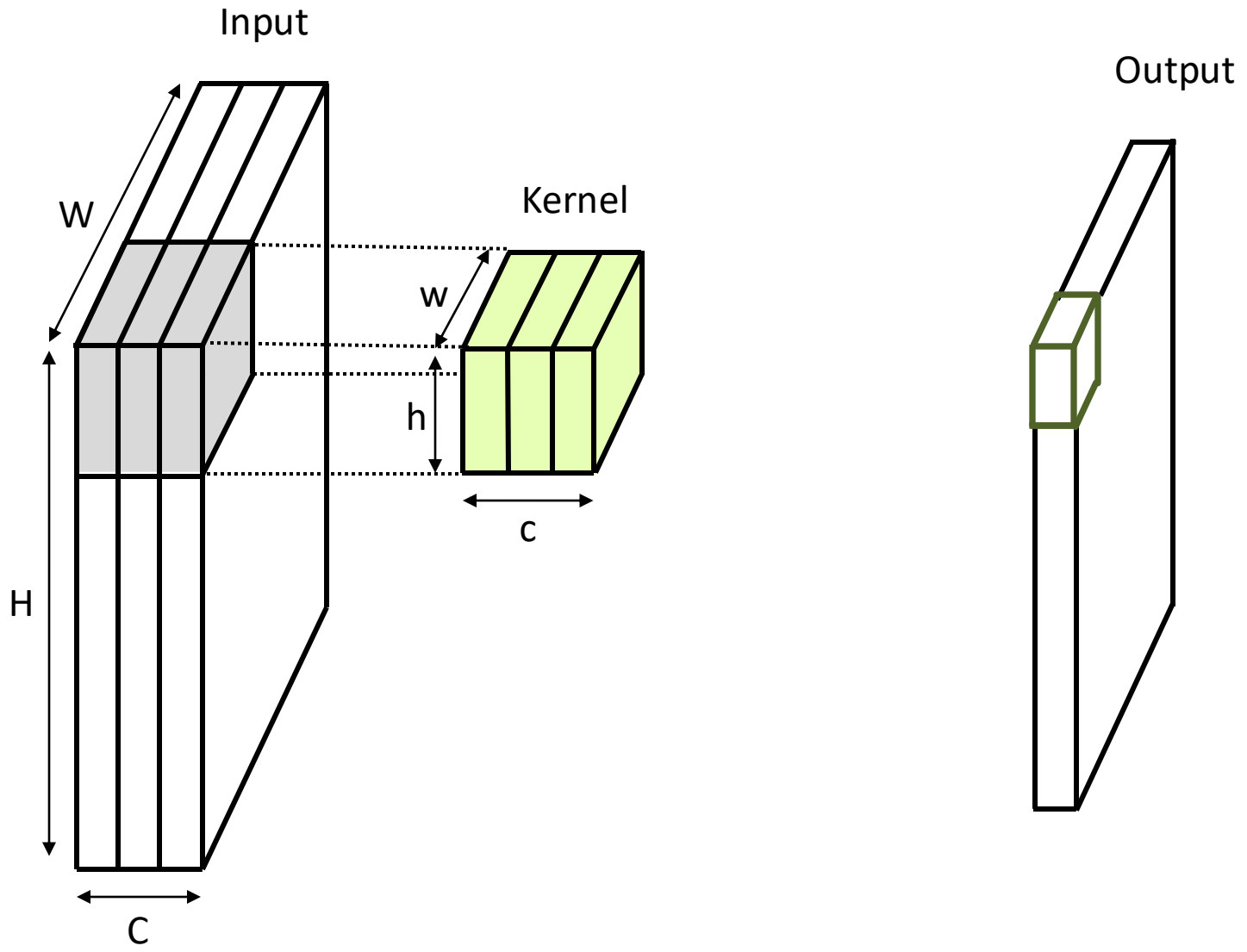
Source: Dr. Francois Fleuret at EPFL

Multi-channel 2D Convolution

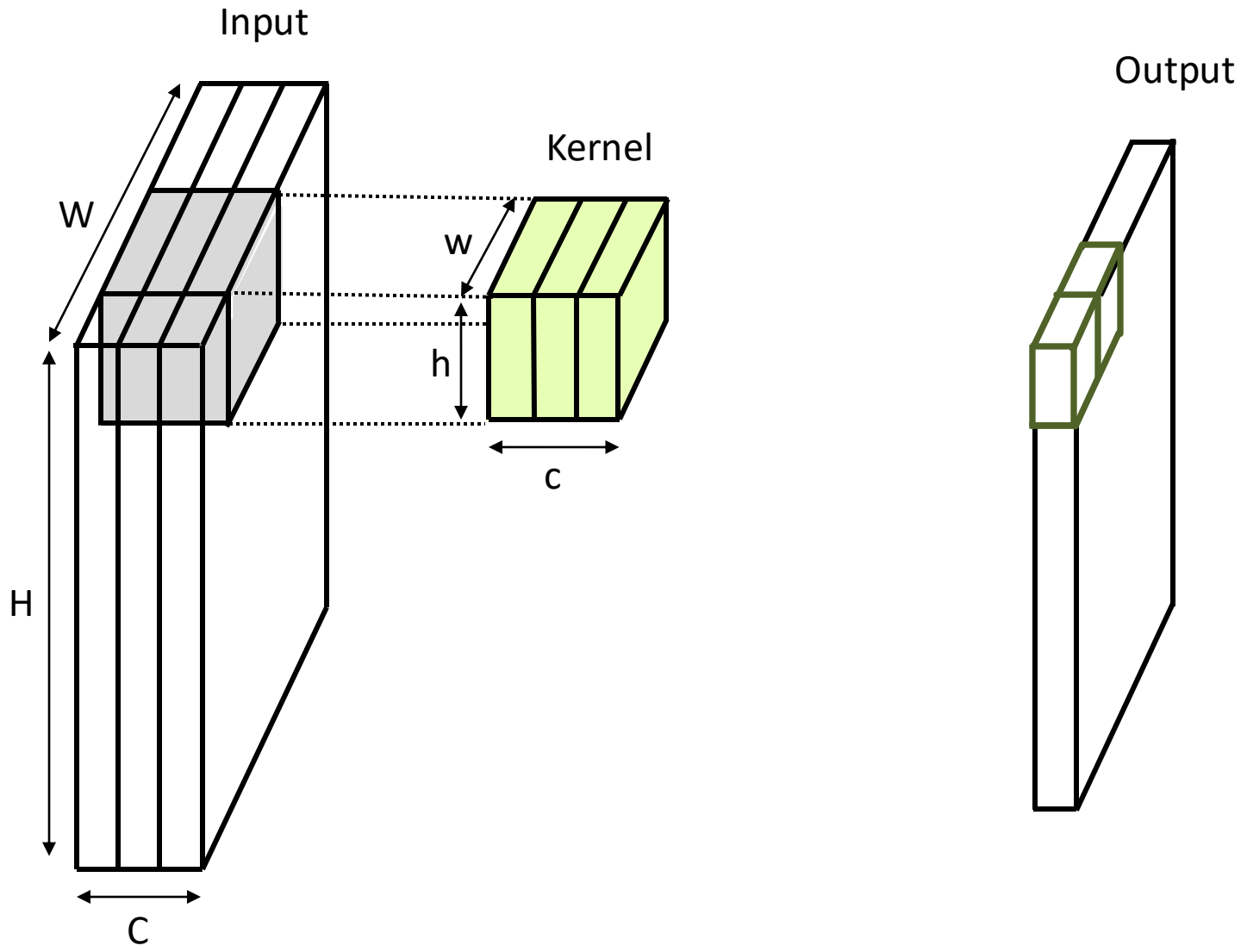


Source: Dr. Francois Fleuret at EPFL

Multi-channel 2D Convolution

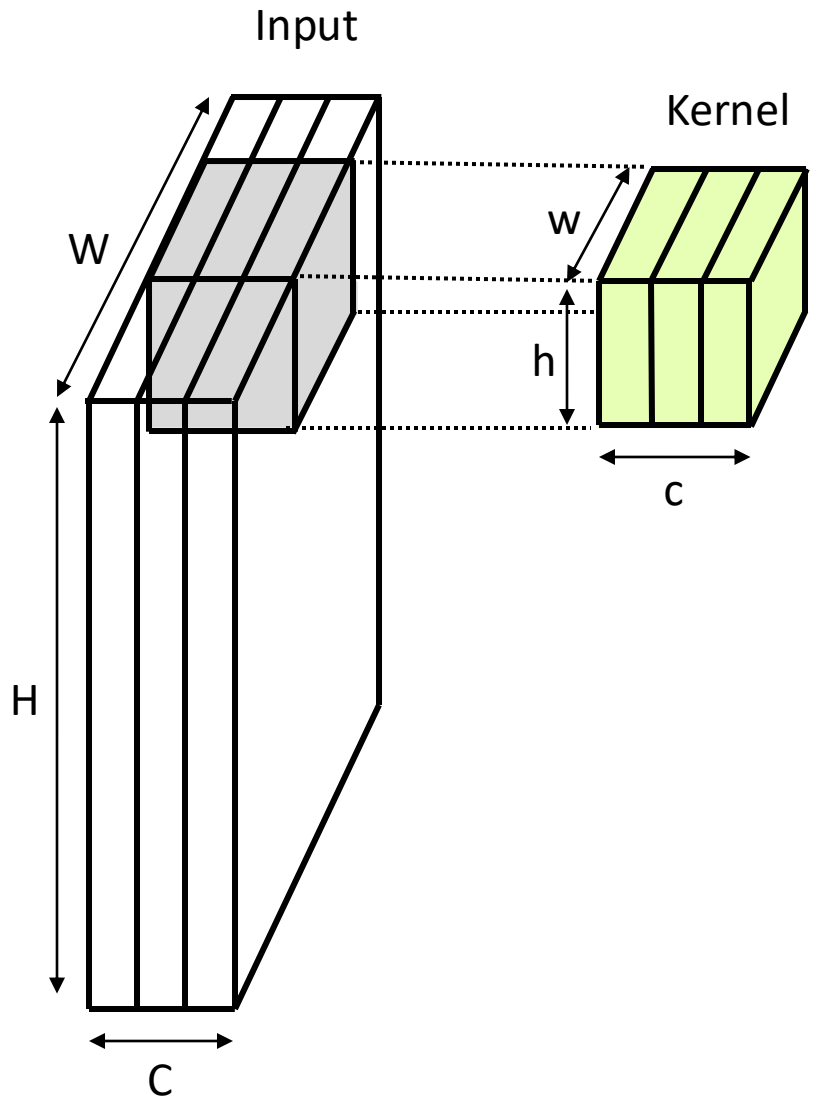


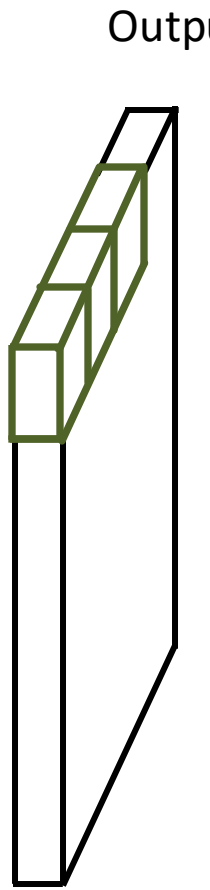
Multi-channel 2D Convolution



Source: Dr. Francois Fleuret at EPFL

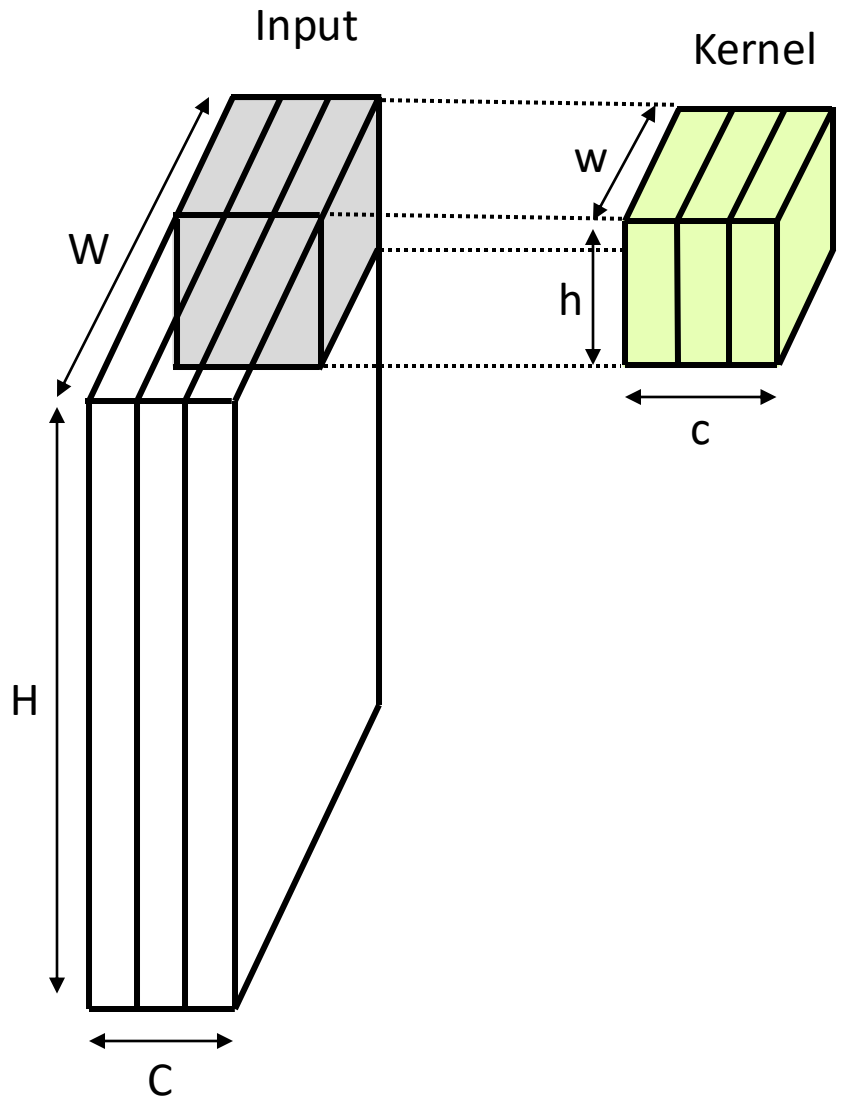
Multi-channel 2D Convolution

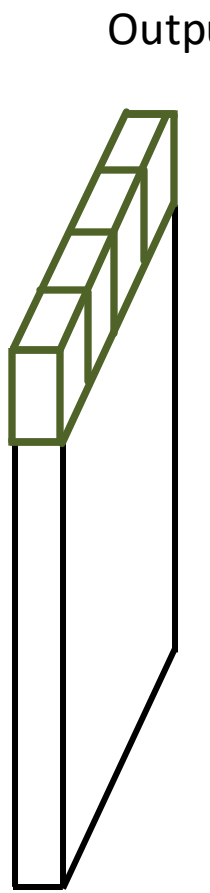




Source: Dr. Francois Fleuret at EPFL

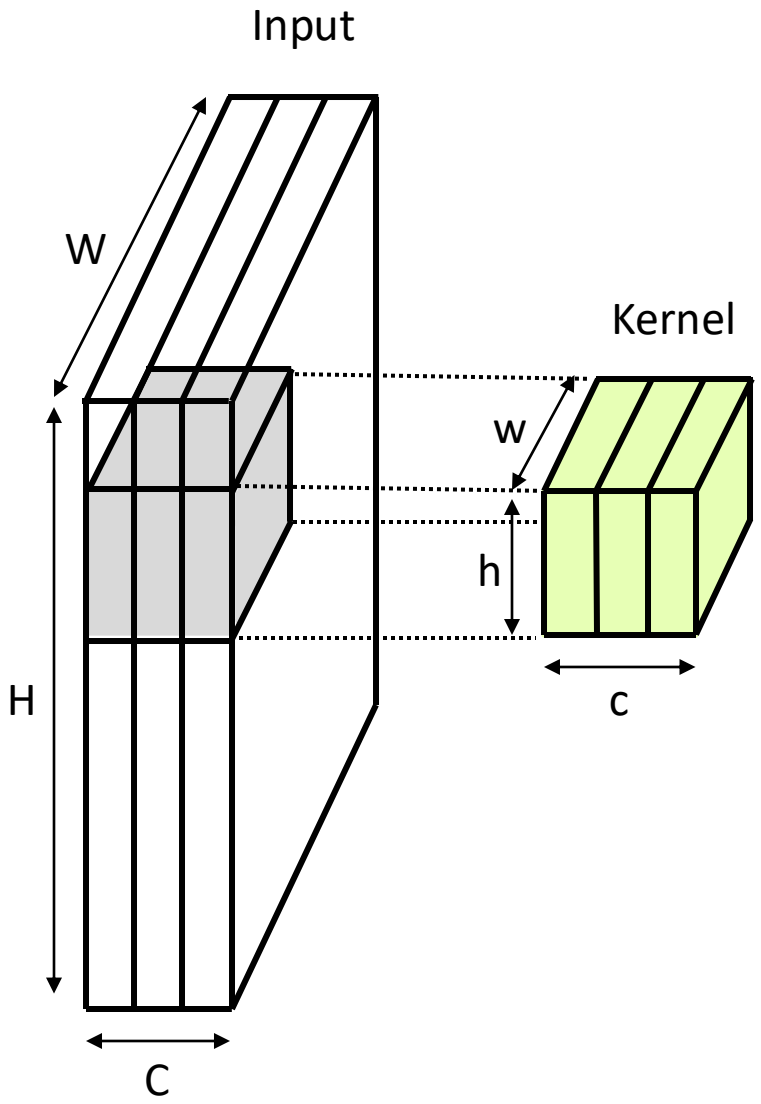
Multi-channel 2D Convolution

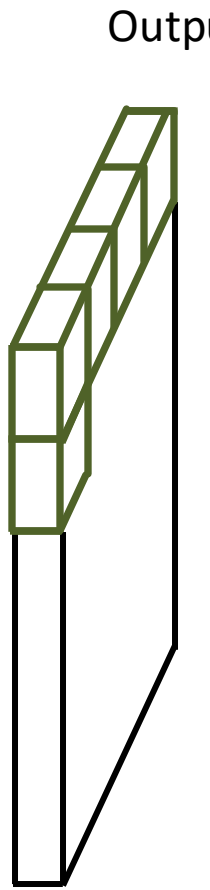




Source: Dr. Francois Fleuret at EPFL

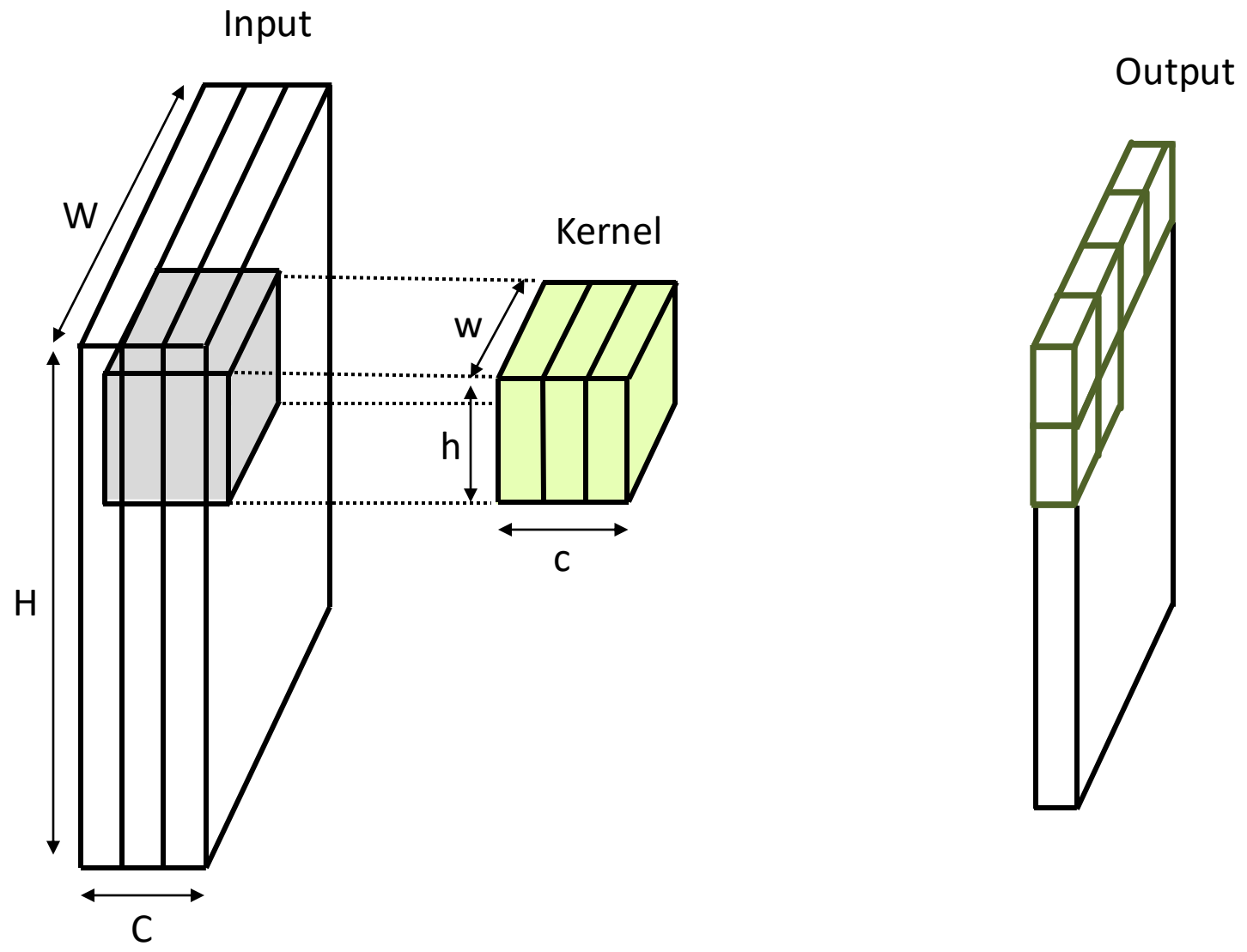
Multi-channel 2D Convolution





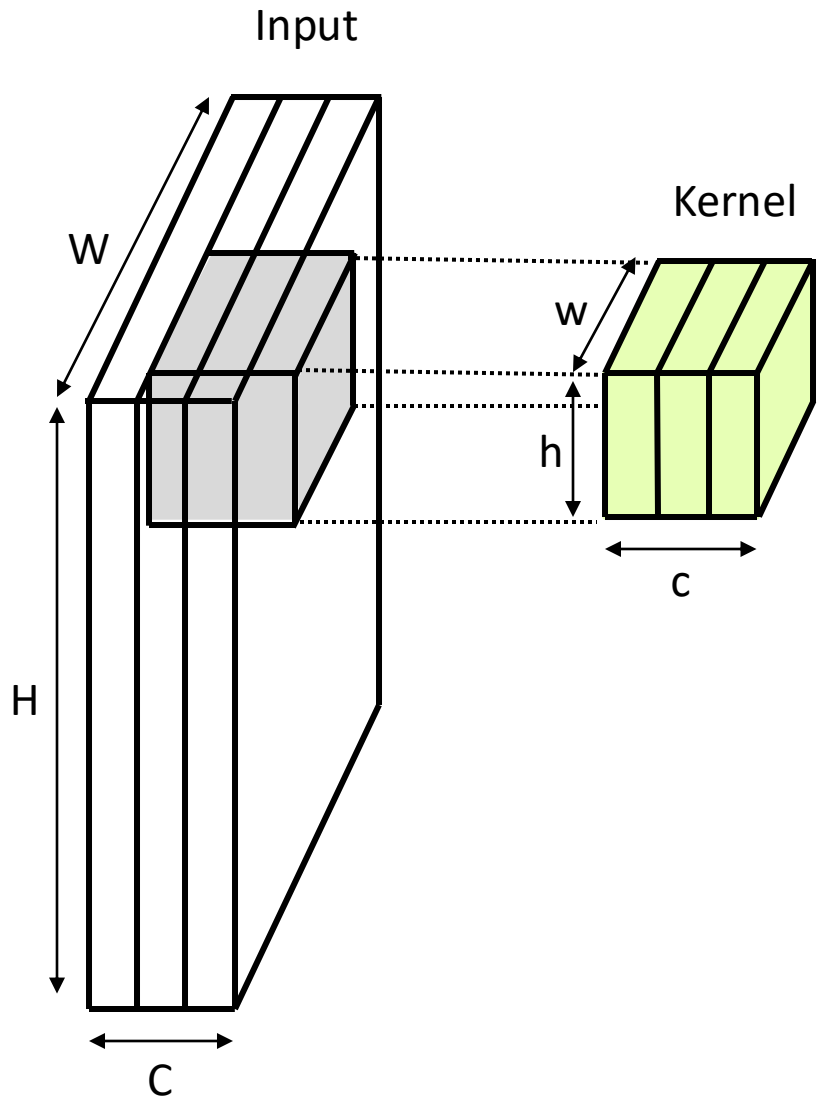
Source: Dr. Francois Fleuret at EPFL

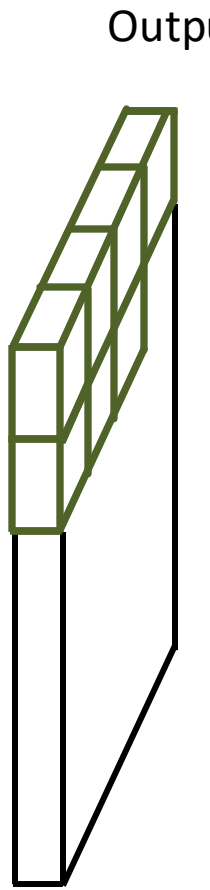
Multi-channel 2D Convolution



Source: Dr. Francois Fleuret at EPFL

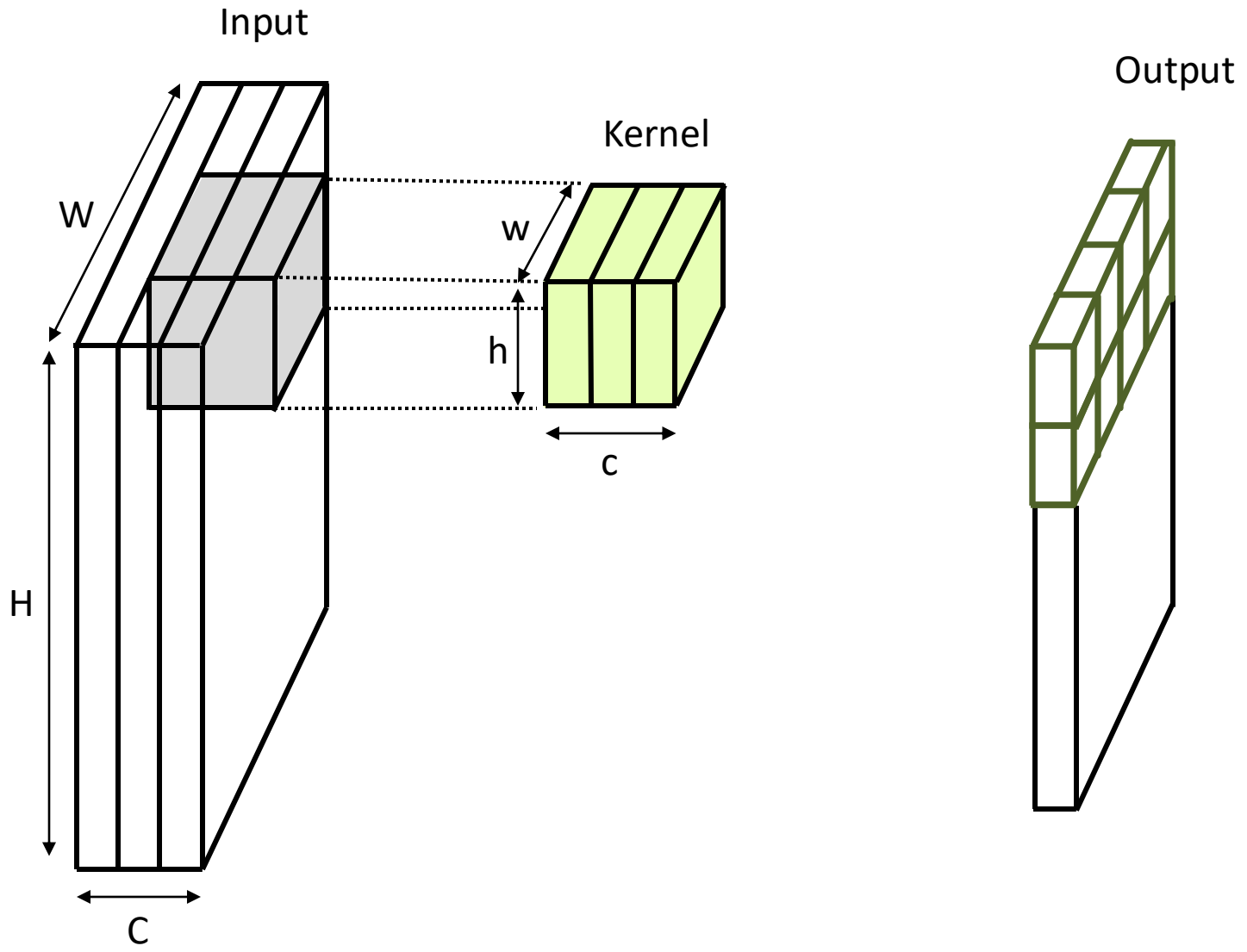
Multi-channel 2D Convolution





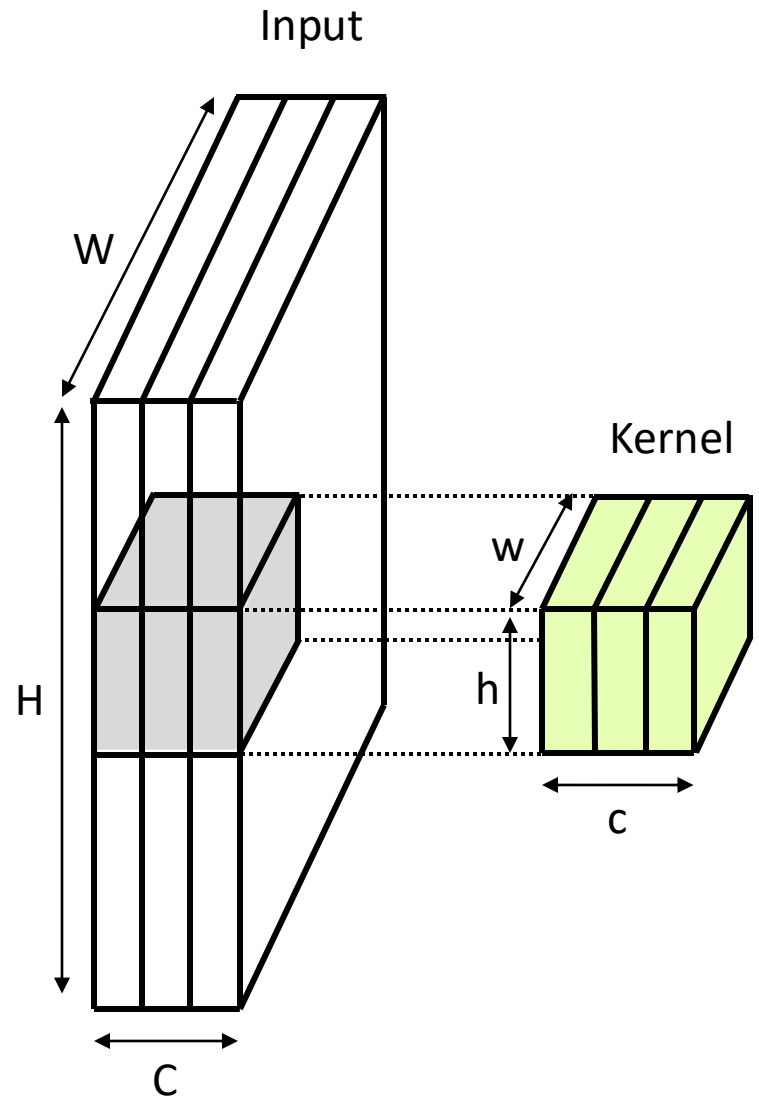
Source: Dr. Francois Fleuret at EPFL

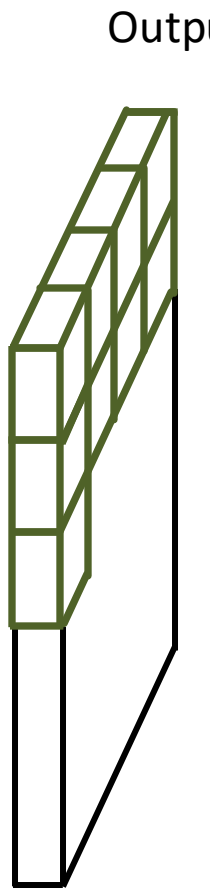
Multi-channel 2D Convolution



Source: Dr. Francois Fleuret at EPFL

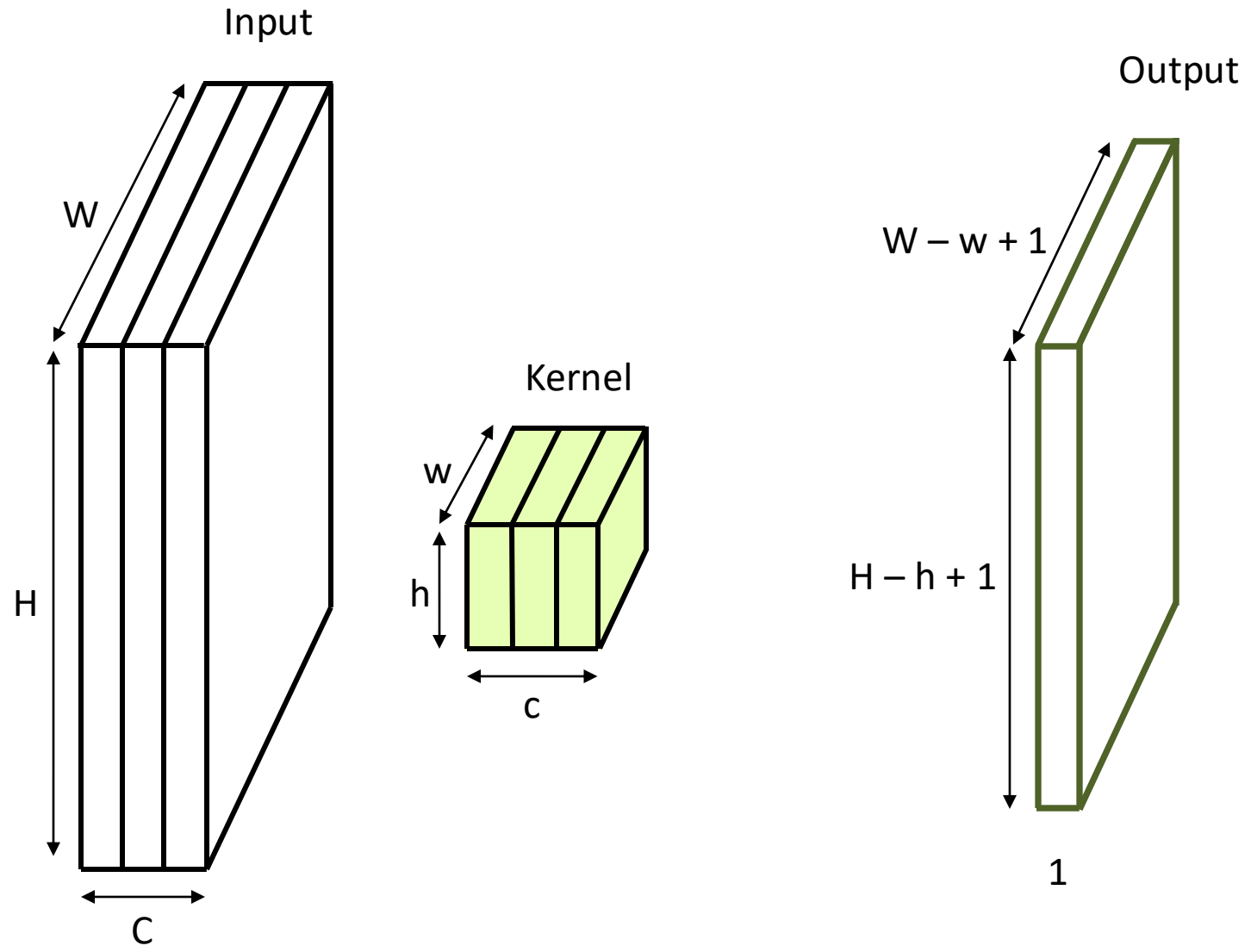
Multi-channel 2D Convolution





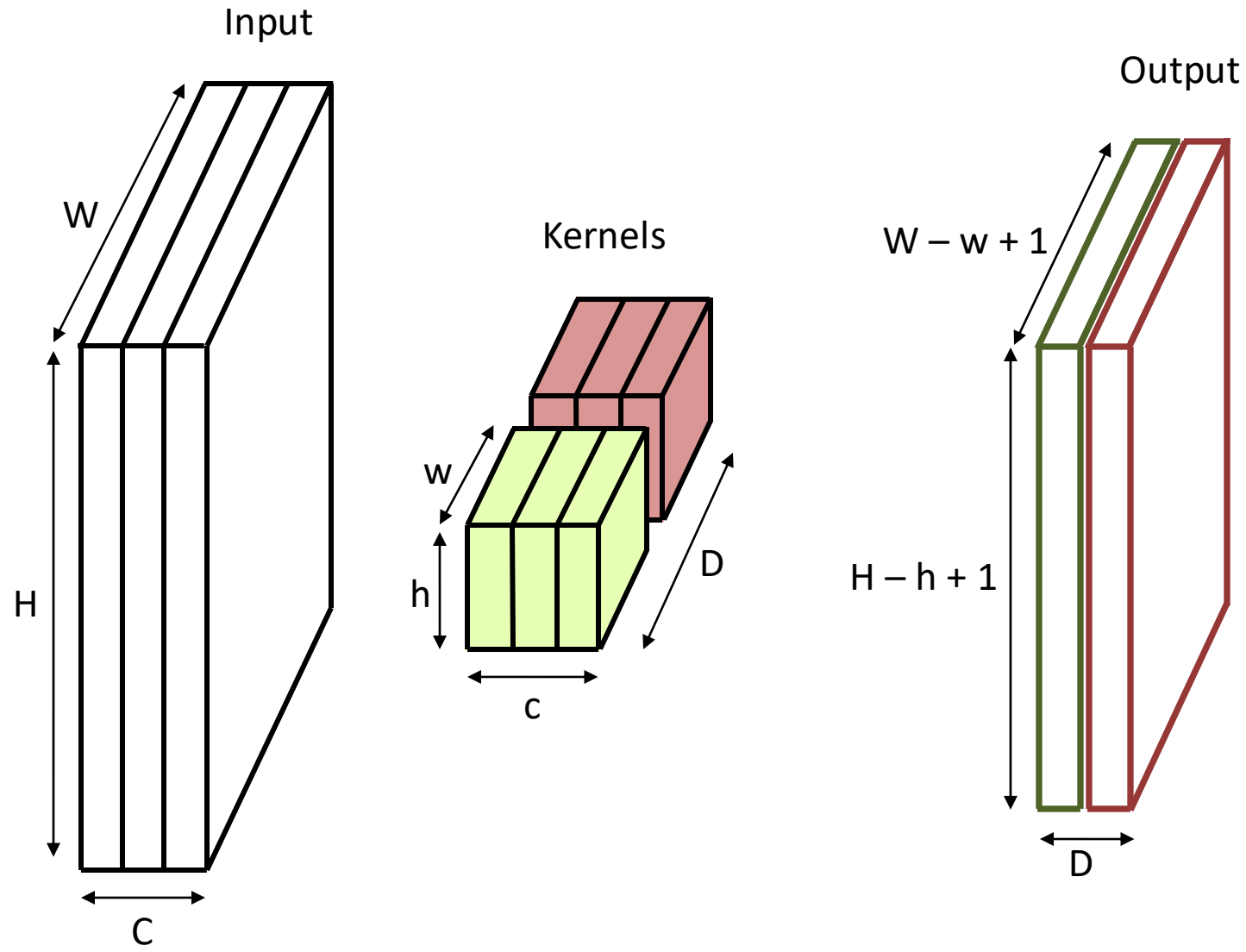
Source: Dr. Francois Fleuret at EPFL

Multi-channel 2D Convolution



Source: Dr. Francois Fleuret at EPFL

Multi-channel and Multi-kernel 2D Convolution



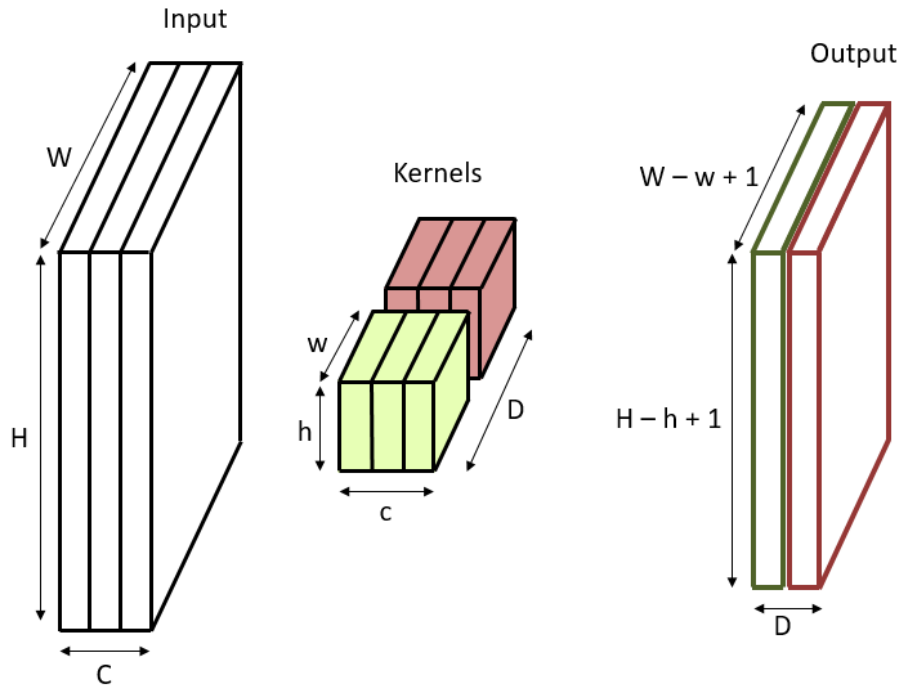
Source: Dr. Francois Fleuret at EPFL

Dealing with Shapes

- Activations or feature maps shape
  - Input (W^i, H^i, C)
  - Output (W^o, H^o, D)

- Kernel of Filter shape (w, h, C, D)
  - $w \times h$ Kernel size
  - C Input channels
  - D Output channels

- Numbers of parameters: $(w \times h \times C + 1) \times D$
  - bias

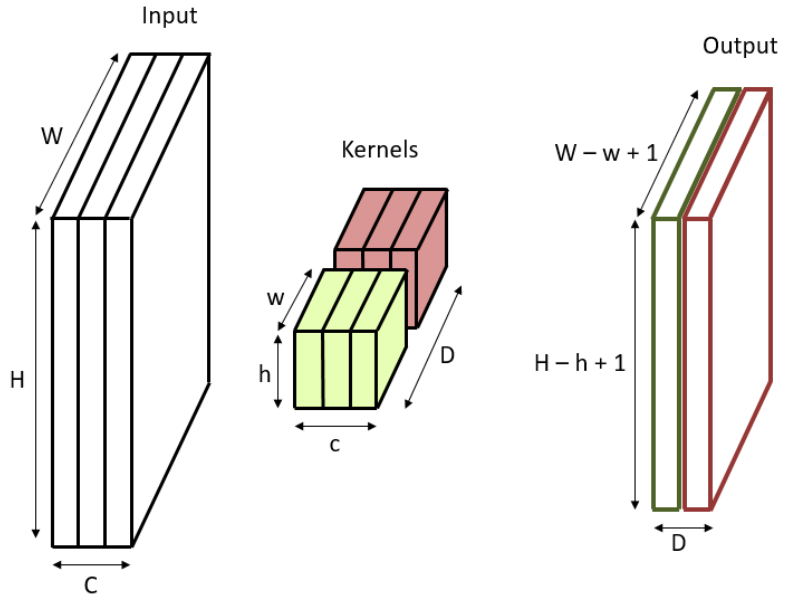


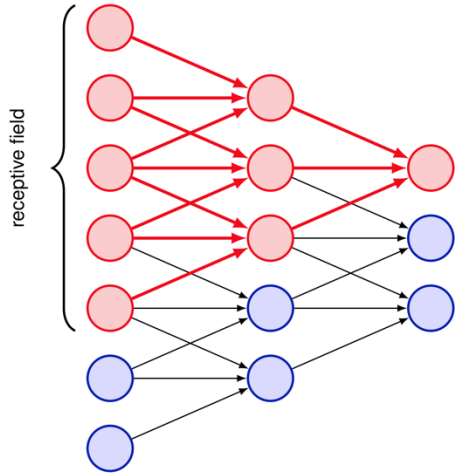
Source: Dr. Francois Fleuret at EPFL

Multi-channel 2D Convolution

- The kernel is not swiped across channels, just across rows and columns.
- Note that a convolution preserves the signal support structure.
  - A 1D signal is converted into a 1D signal, a 2D signal into a 2D, and neighboring parts of the input signal influence neighboring parts of the output signal.

- We usually refer to one of the channels generated by a convolution layer as an activation map.
- The sub-area of an input map that influences a component of the output as the receptive field of the latter.



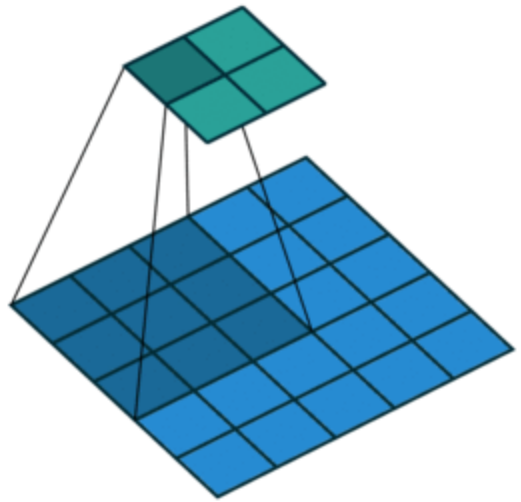


Source: Deep Learning Foundations and Concepts

Padding and Stride

Strides

- Strides: increment step size for the convolution operator
- Reduces the size of the output map

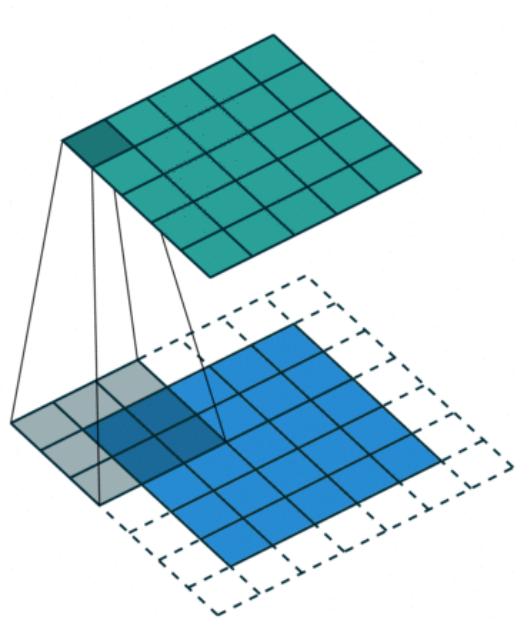


Example with kernel size 3×3 and a stride of 2 (image in blue)

Source: https://github.com/vdumoulin/conv_arithmetic

Padding

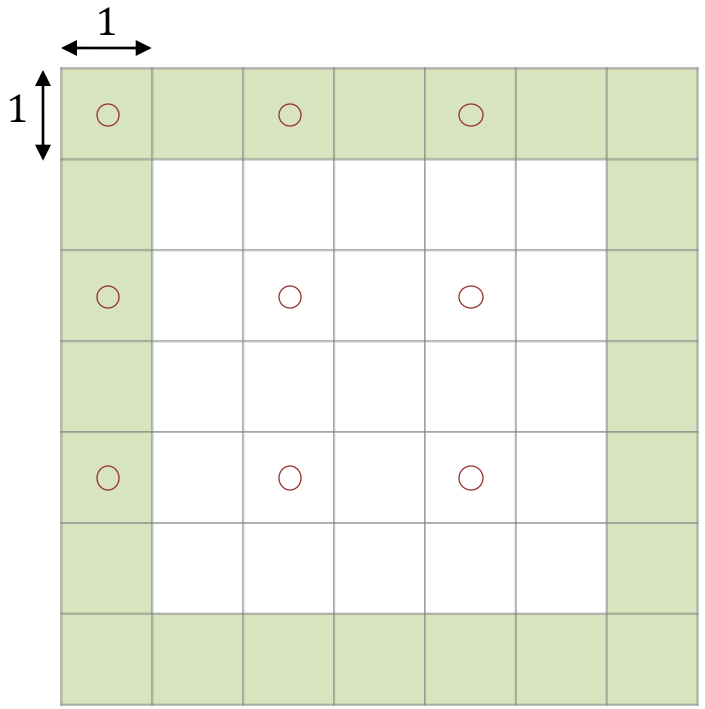
- Padding: artificially fill borders of image
- Useful to keep spatial dimension constant across filters
- Useful with strides and large receptive fields
- Usually fill with 0s



Source: https://github.com/vdumoulin/conv_arithmetic

Padding and Stride

- Here with $5 \times 5 \times C$ as input, a padding of $(1\ ,1)$, a stride of $(2\ ,2)$

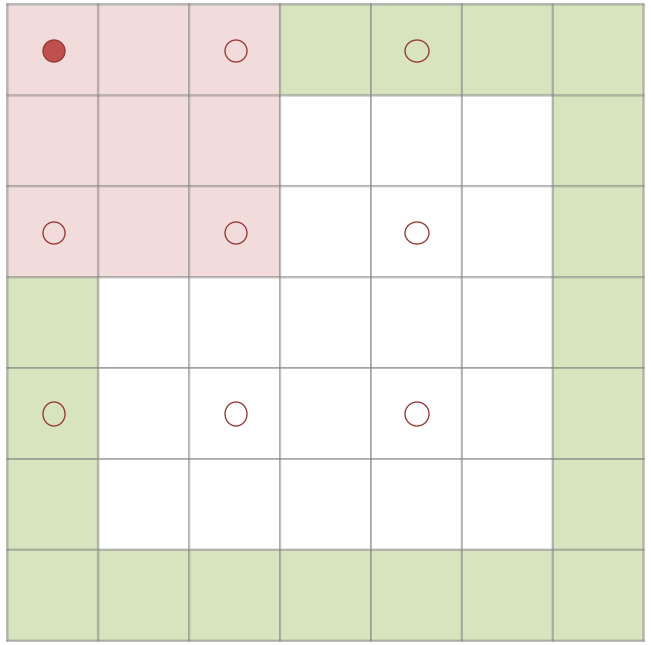


Input

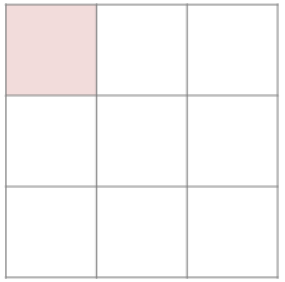
Source: Dr. Francois Fleuret at EPFL

Padding and Stride

- Here with $5 \times 5 \times C$ as input, a padding of $(1,1)$, a stride of $(2,2)$, and a kernel of size $3 \times 3 \times C$



Input

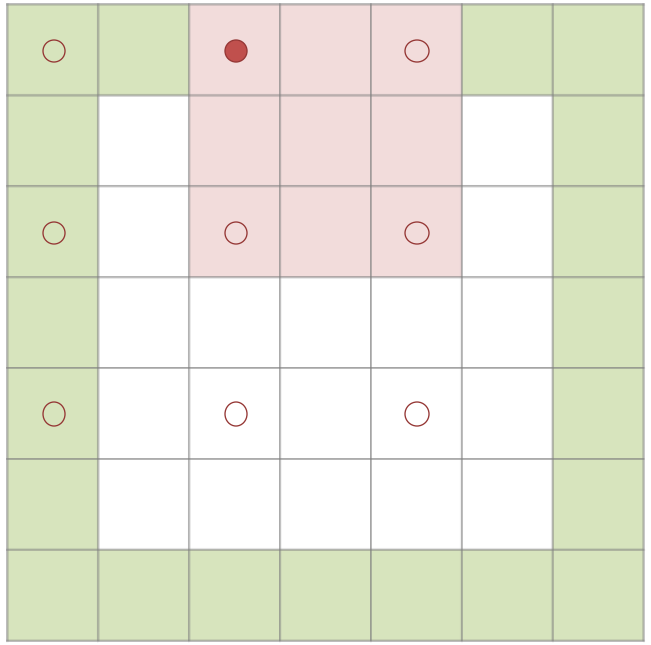


Output

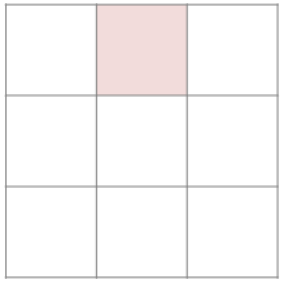
Source: Dr. Francois Fleuret at EPFL

Padding and Stride

- Here with $5 \times 5 \times C$ as input, a padding of $(1,1)$, a stride of $(2,2)$, and a kernel of size $3 \times 3 \times C$



Input

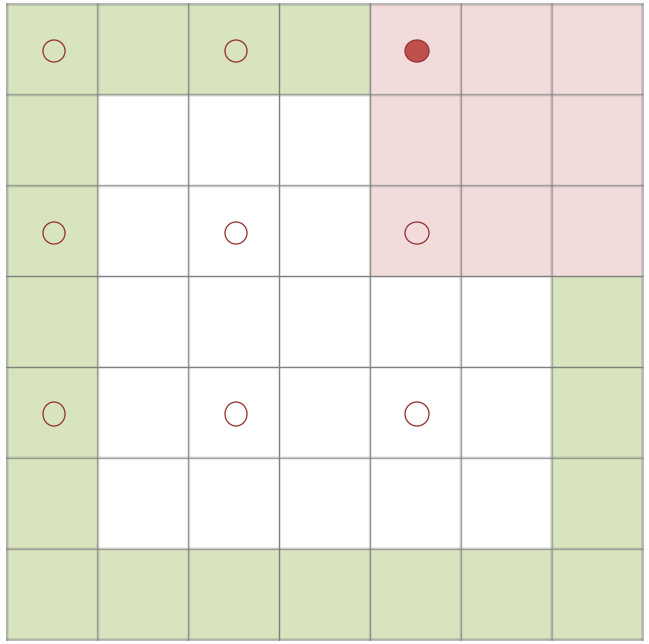


Output

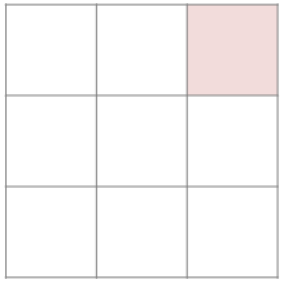
Source: Dr. Francois Fleuret at EPFL

Padding and Stride

- Here with $5 \times 5 \times C$ as input, a padding of $(1,1)$, a stride of $(2,2)$, and a kernel of size $3 \times 3 \times C$



Input

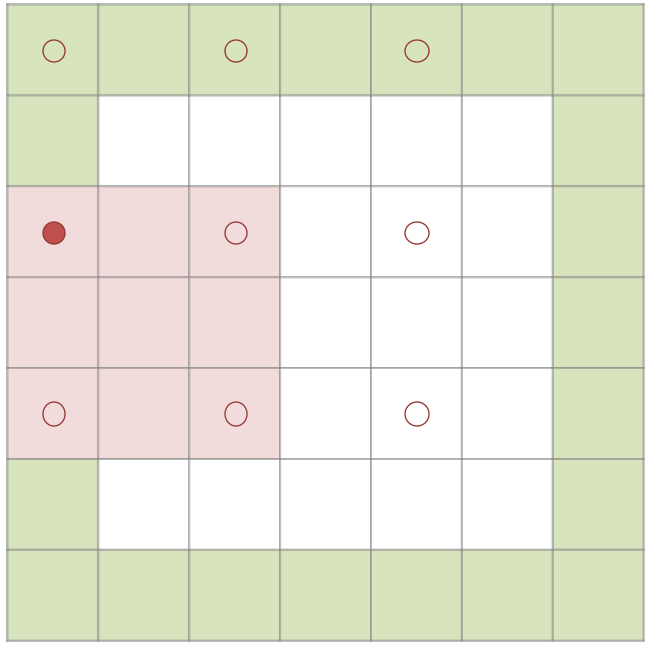


Output

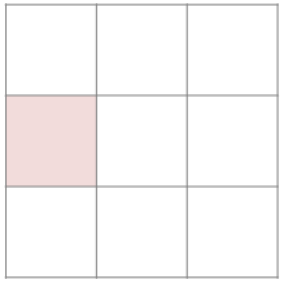
Source: Dr. Francois Fleuret at EPFL

Padding and Stride

- Here with $5 \times 5 \times C$ as input, a padding of $(1,1)$, a stride of $(2,2)$, and a kernel of size $3 \times 3 \times C$



Input

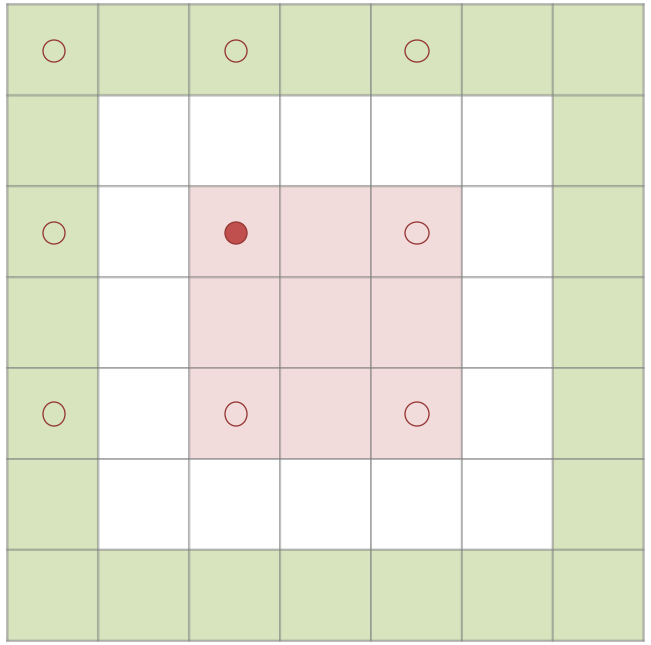


Output

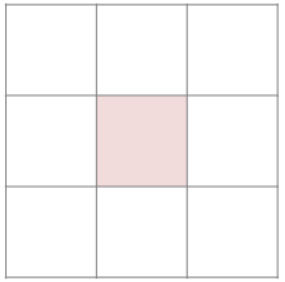
Source: Dr. Francois Fleuret at EPFL

Padding and Stride

- Here with $5 \times 5 \times C$ as input, a padding of $(1,1)$, a stride of $(2,2)$, and a kernel of size $3 \times 3 \times C$



Input

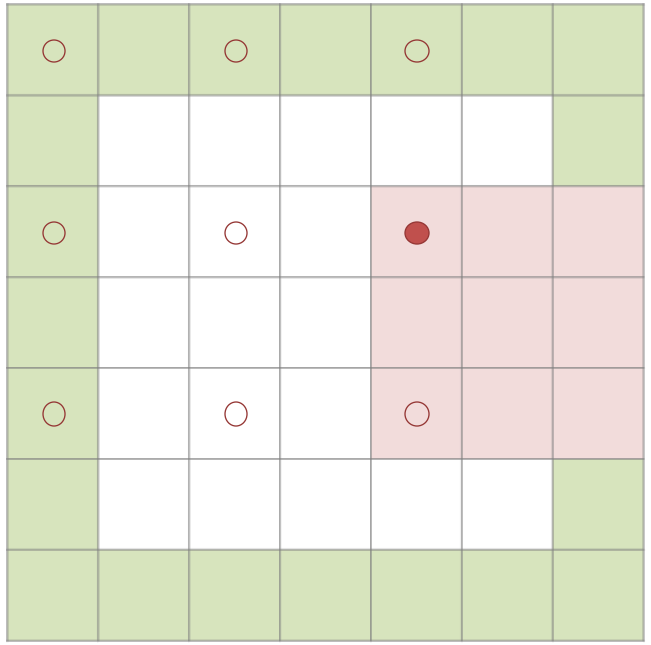


Output

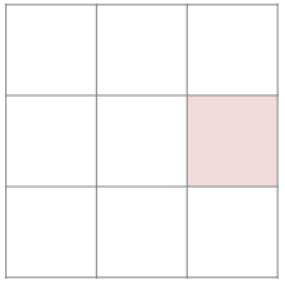
Source: Dr. Francois Fleuret at EPFL

Padding and Stride

- Here with $5 \times 5 \times C$ as input, a padding of $(1,1)$, a stride of $(2,2)$, and a kernel of size $3 \times 3 \times C$



Input

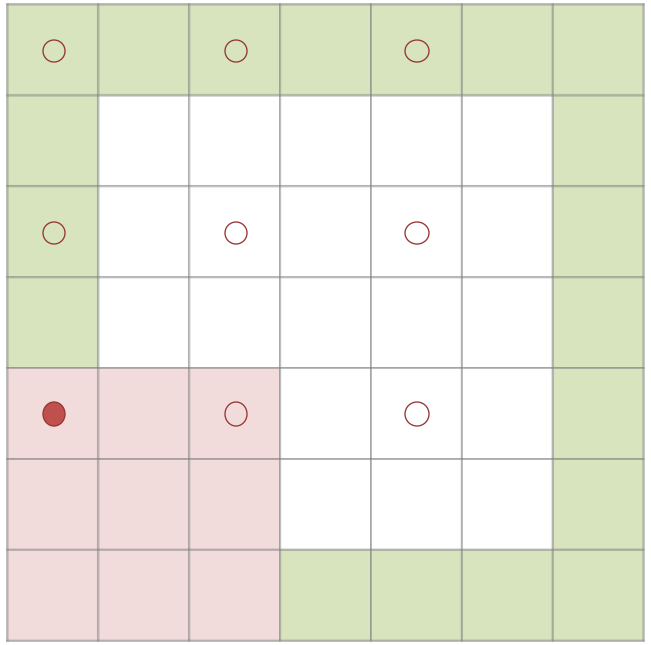


Output

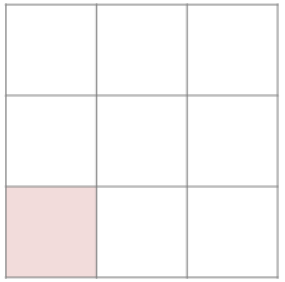
Source: Dr. Francois Fleuret at EPFL

Padding and Stride

- Here with $5 \times 5 \times C$ as input, a padding of $(1,1)$, a stride of $(2,2)$, and a kernel of size $3 \times 3 \times C$



Input

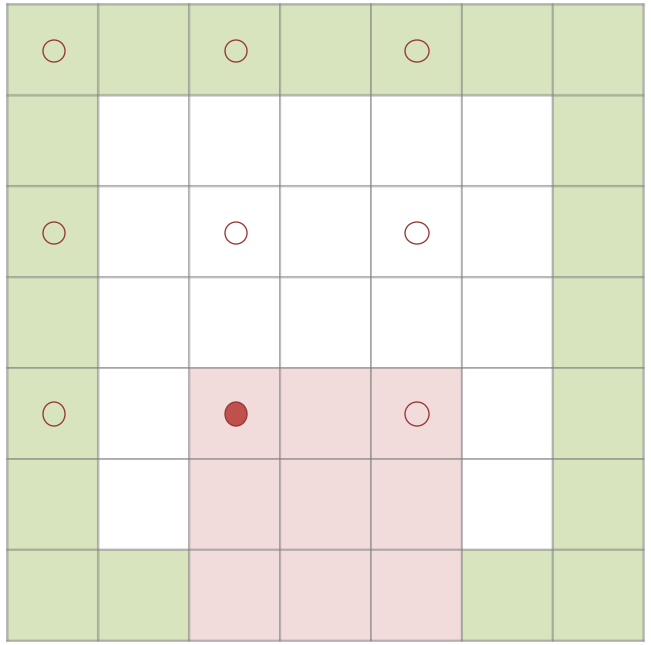


Output

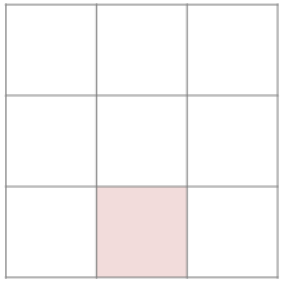
Source: Dr. Francois Fleuret at EPFL

Padding and Stride

- Here with $5 \times 5 \times C$ as input, a padding of $(1,1)$, a stride of $(2,2)$, and a kernel of size $3 \times 3 \times C$



Input

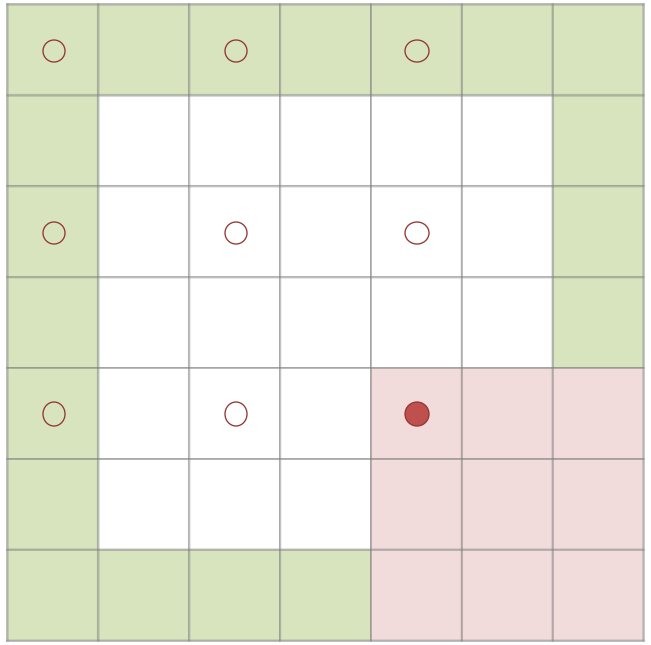


Output

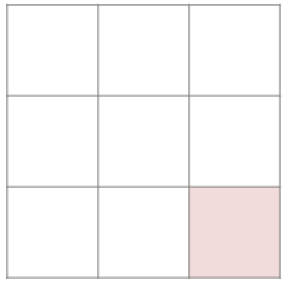
Source: Dr. Francois Fleuret at EPFL

Padding and Stride

- Here with $5 \times 5 \times C$ as input, a padding of $(1,1)$, a stride of $(2,2)$, and a kernel of size $3 \times 3 \times C$



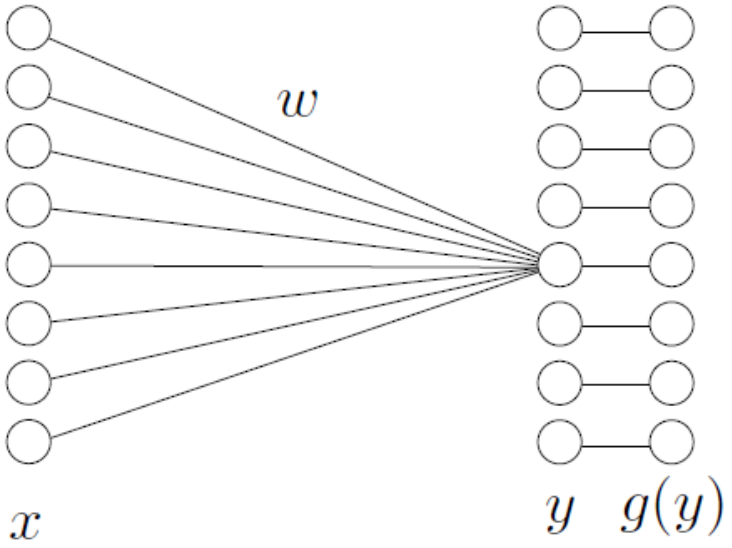
Input

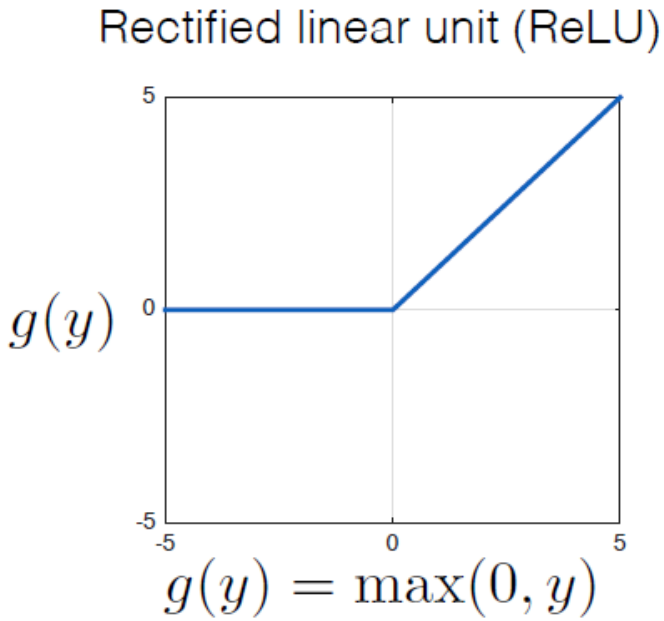


Output

Source: Dr. Francois Fleuret at EPFL

Nonlinear Activation Function

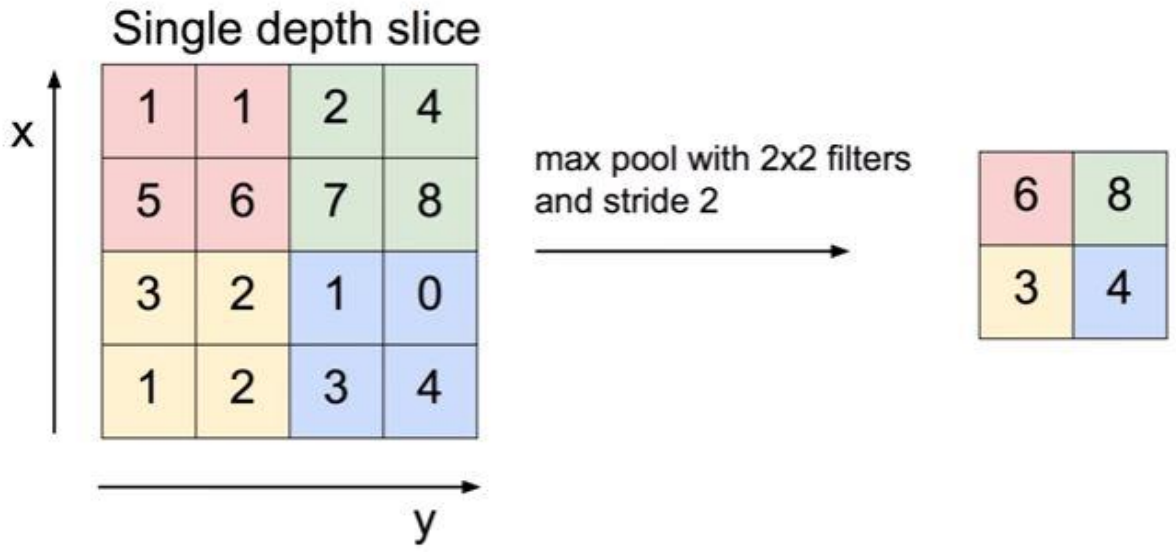




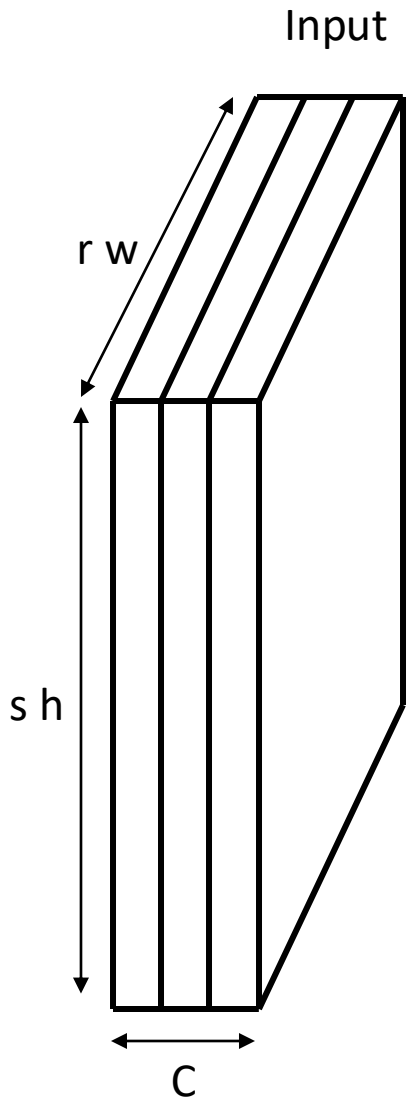
Pooling

Pooling

- Compute a maximum value in a sliding window (max pooling)
- Reduce spatial resolution for faster computation
- Achieve invariance to local translation
- Max pooling introduces invariances
  - Pooling size : 2×2
  - No parameters: max or average of 2x2 units

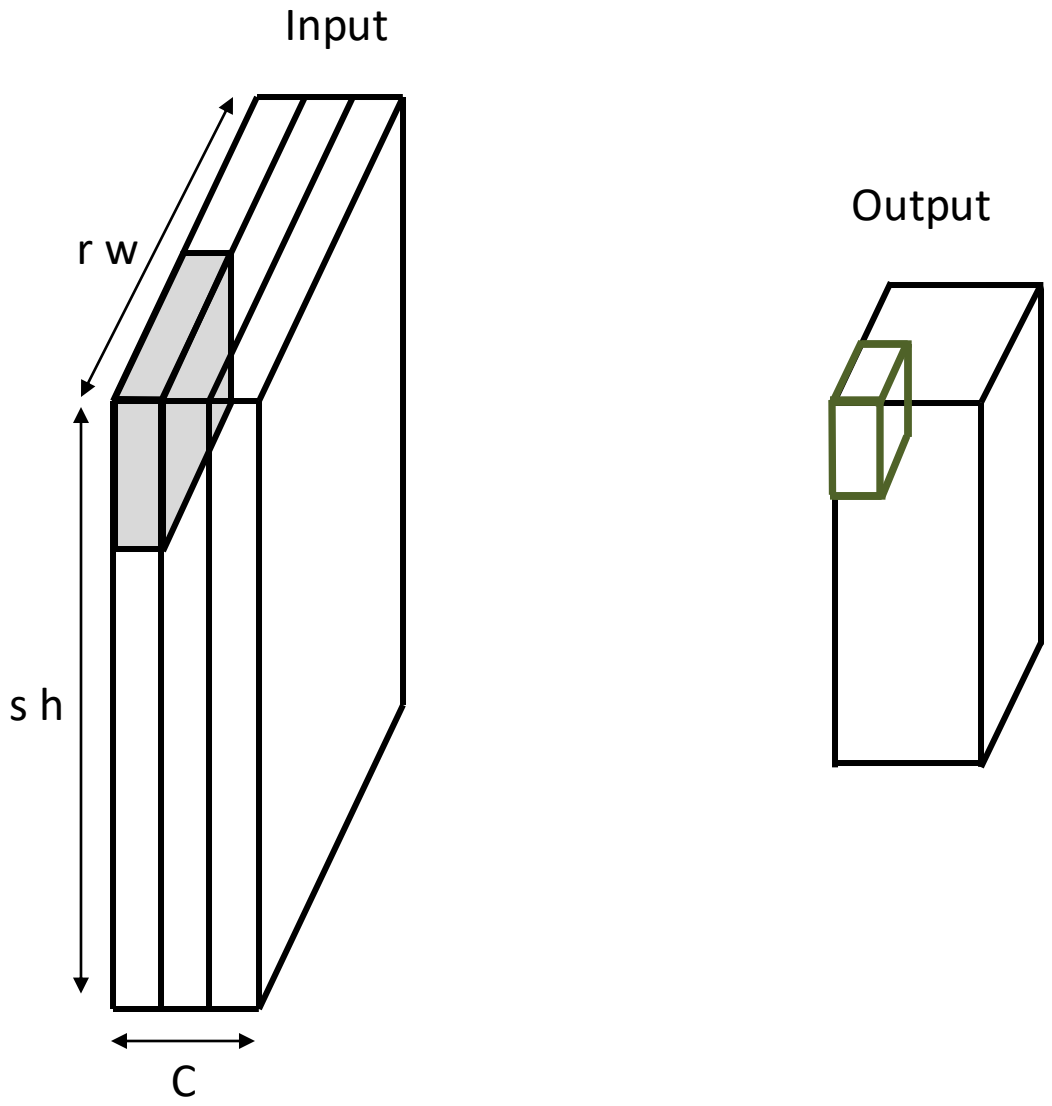


Multi-channel Pooling

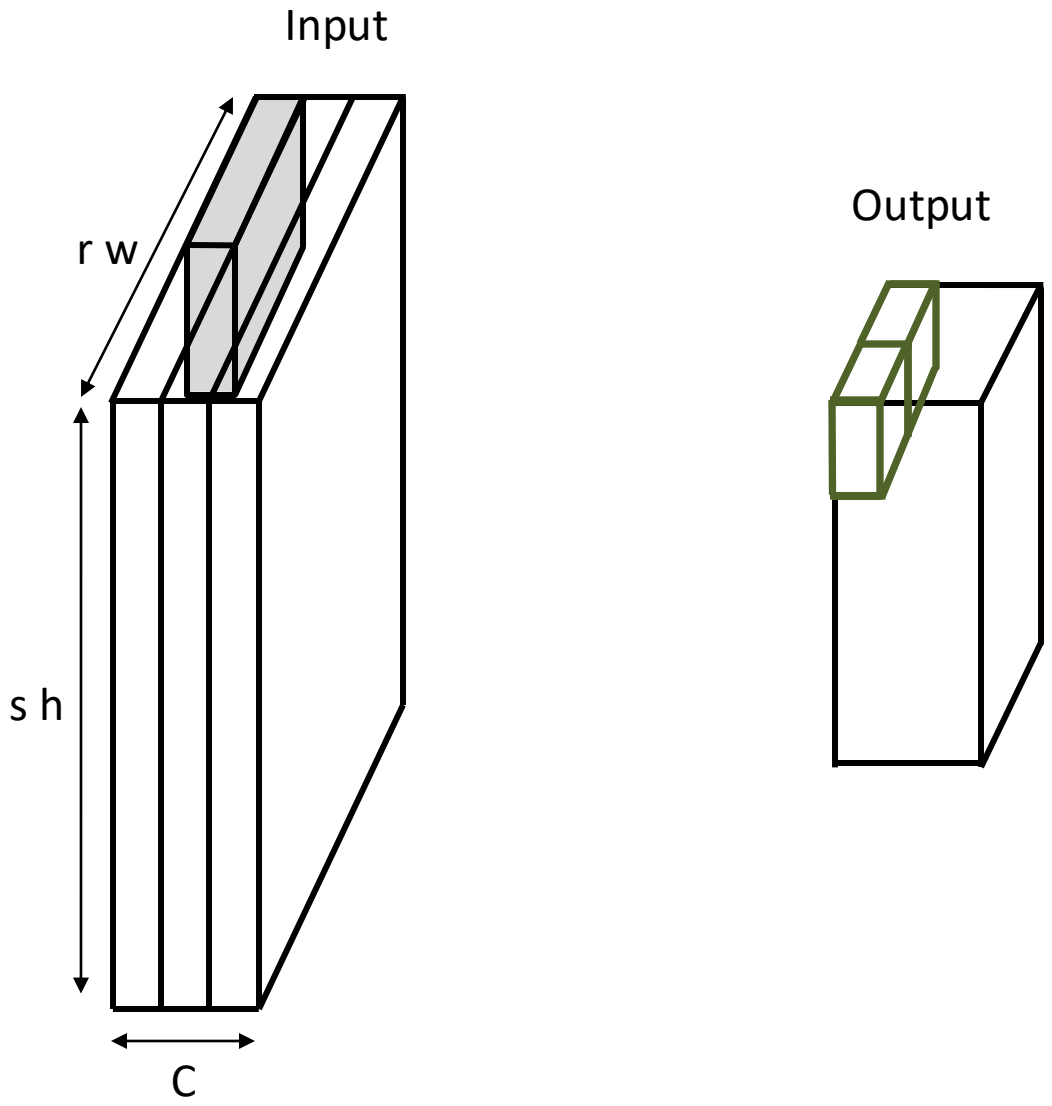


Source: Dr. Francois Fleuret at EPFL

Multi-channel Pooling

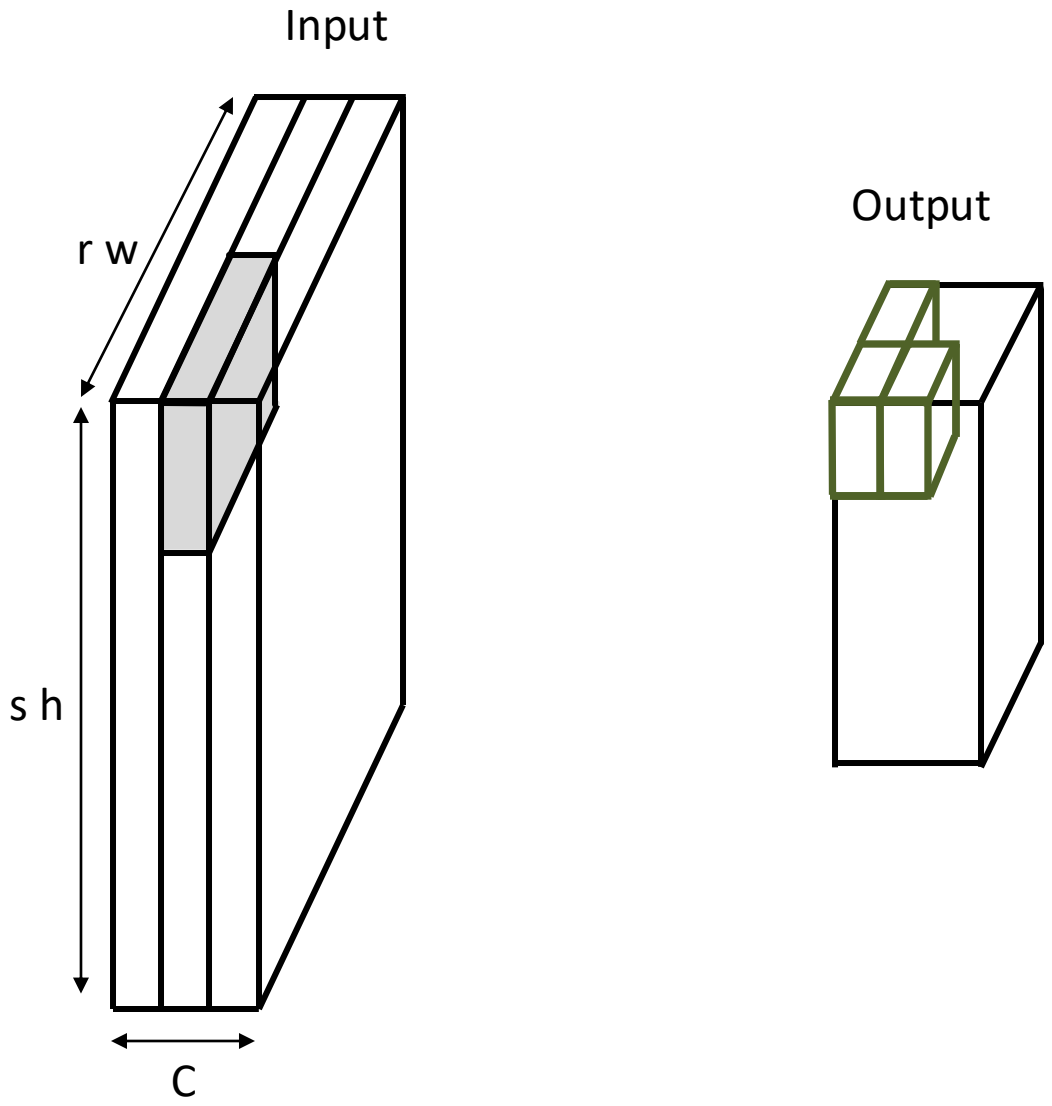


Multi-channel Pooling



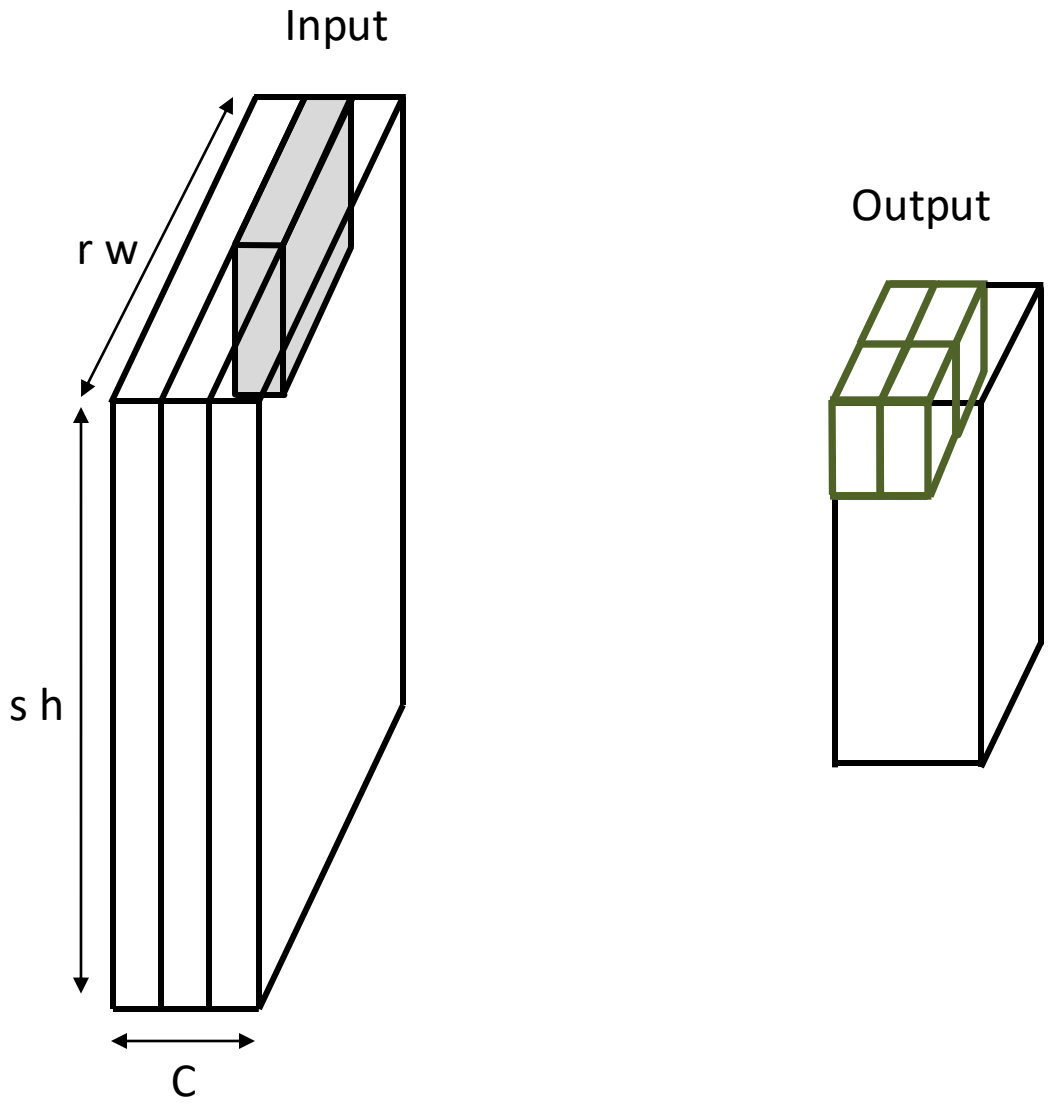
Source: Dr. Francois Fleuret at EPFL

Multi-channel Pooling



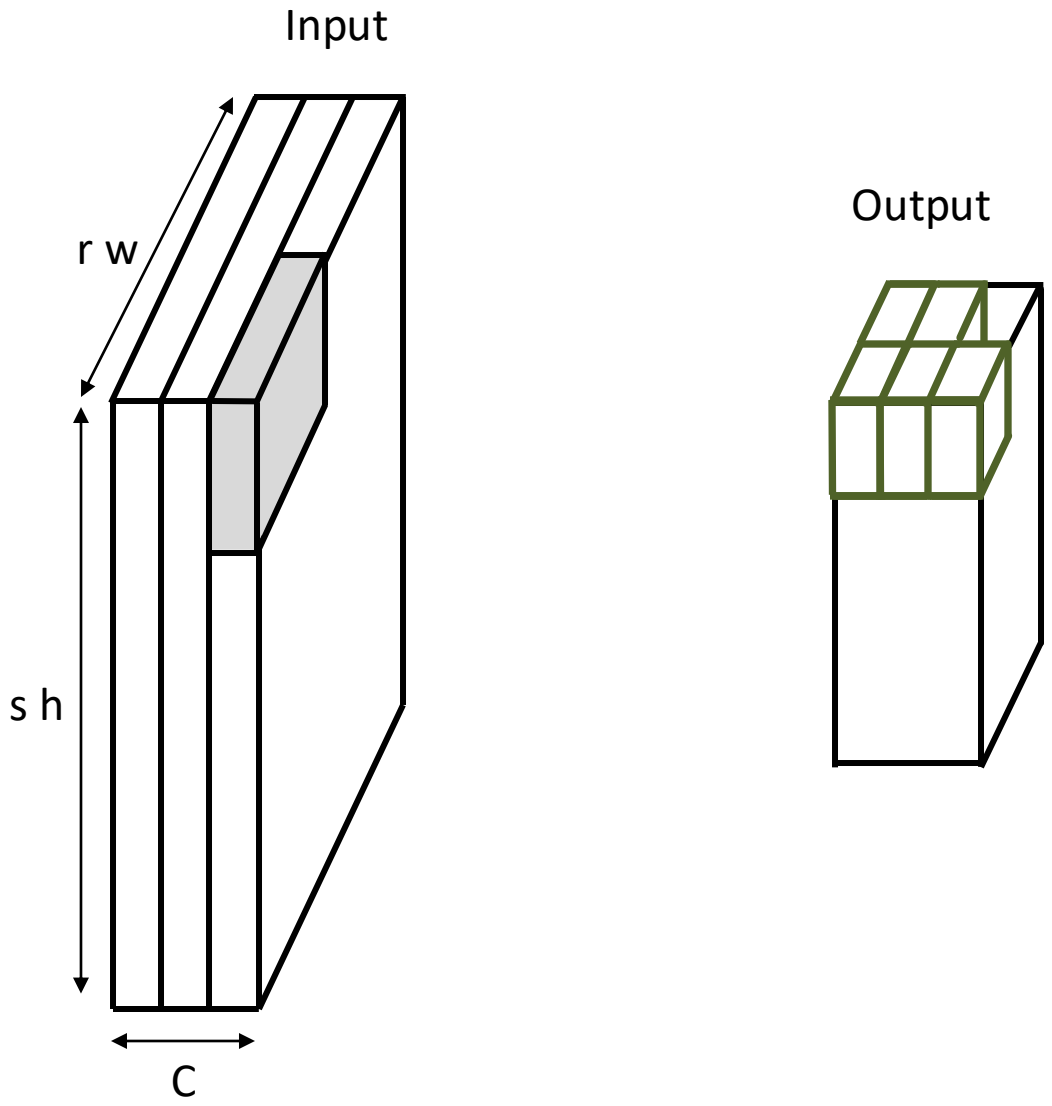
Source: Dr. Francois Fleuret at EPFL

Multi-channel Pooling



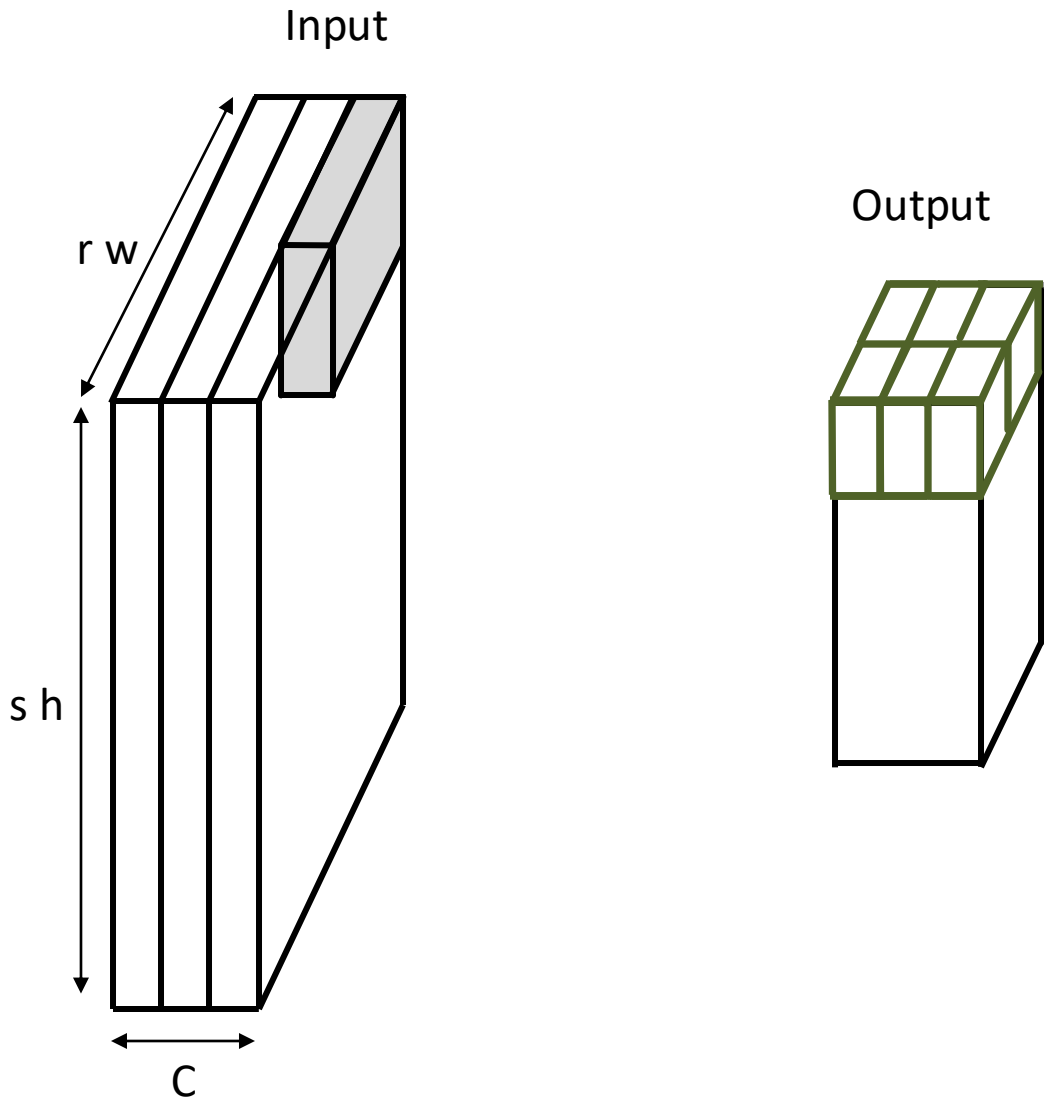
Source: Dr. Francois Fleuret at EPFL

Multi-channel Pooling



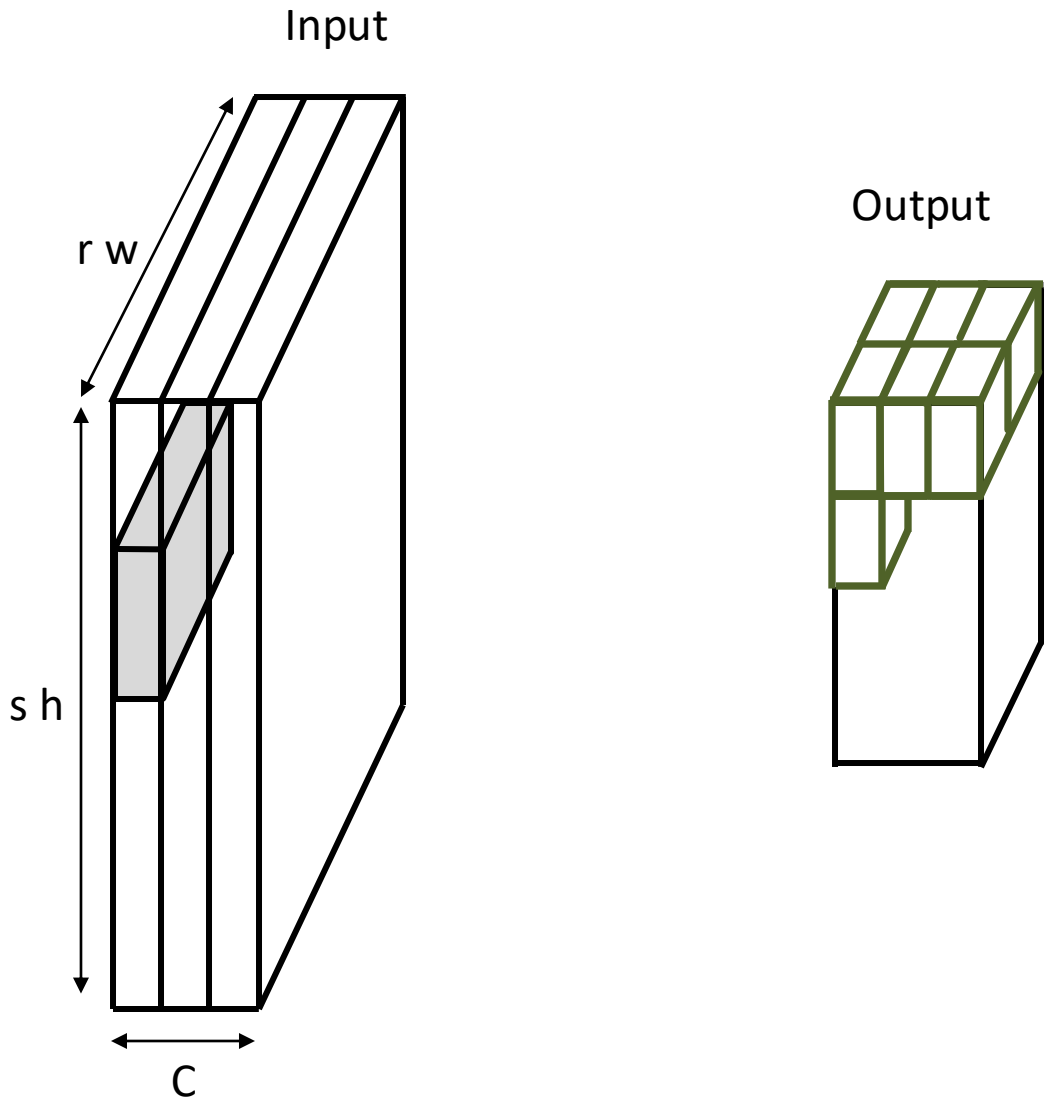
Source: Dr. Francois Fleuret at EPFL

Multi-channel Pooling



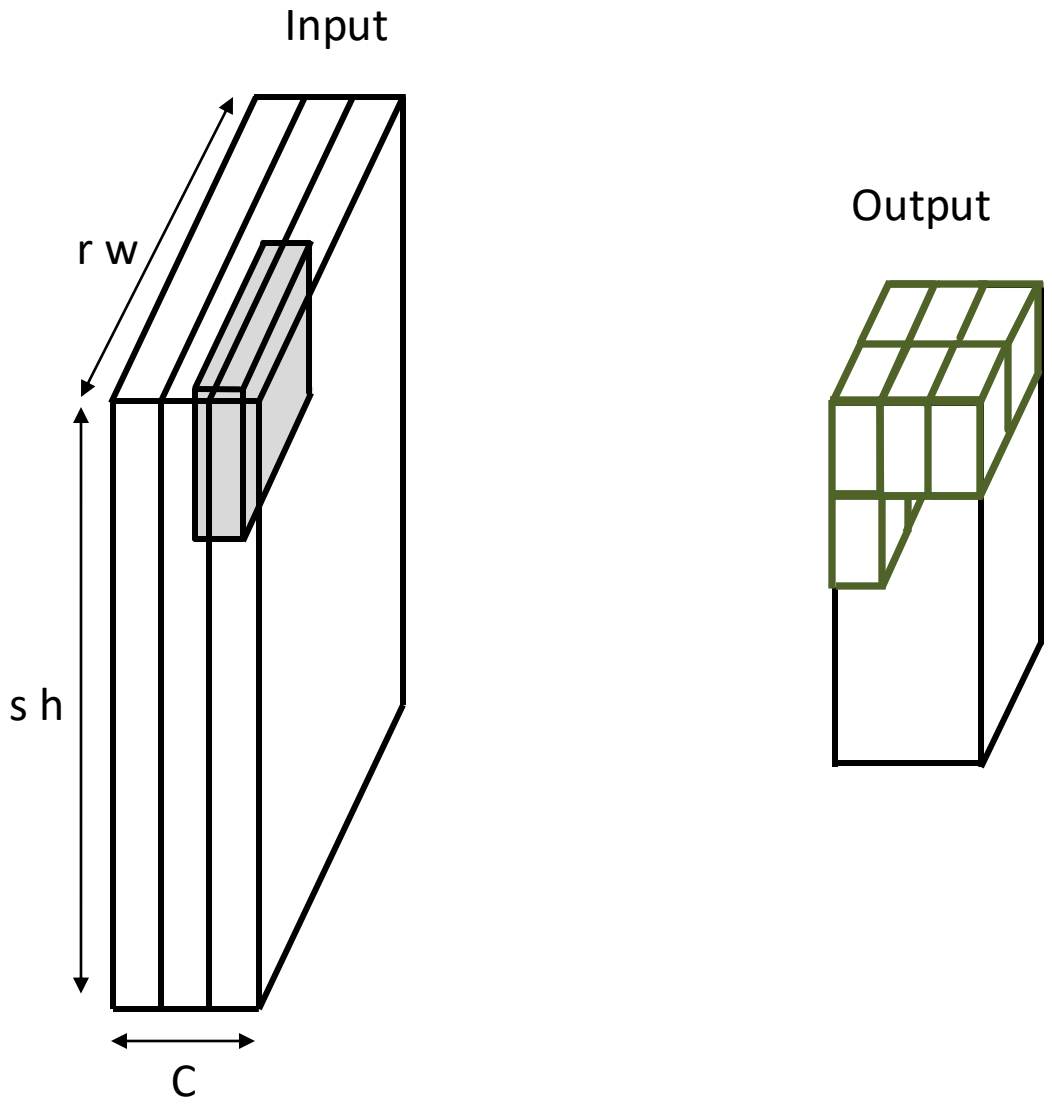
Source: Dr. Francois Fleuret at EPFL

Multi-channel Pooling



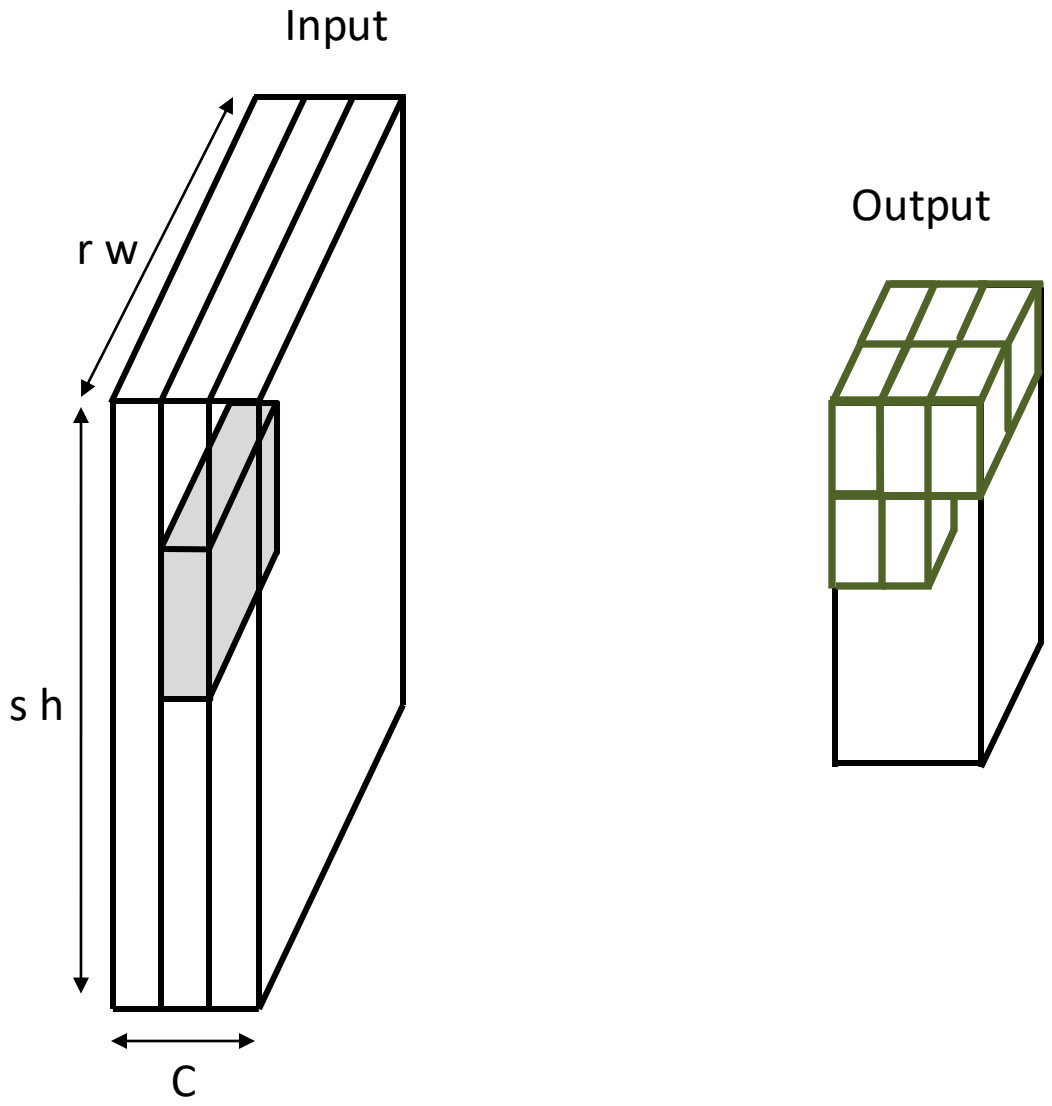
Source: Dr. Francois Fleuret at EPFL

Multi-channel Pooling



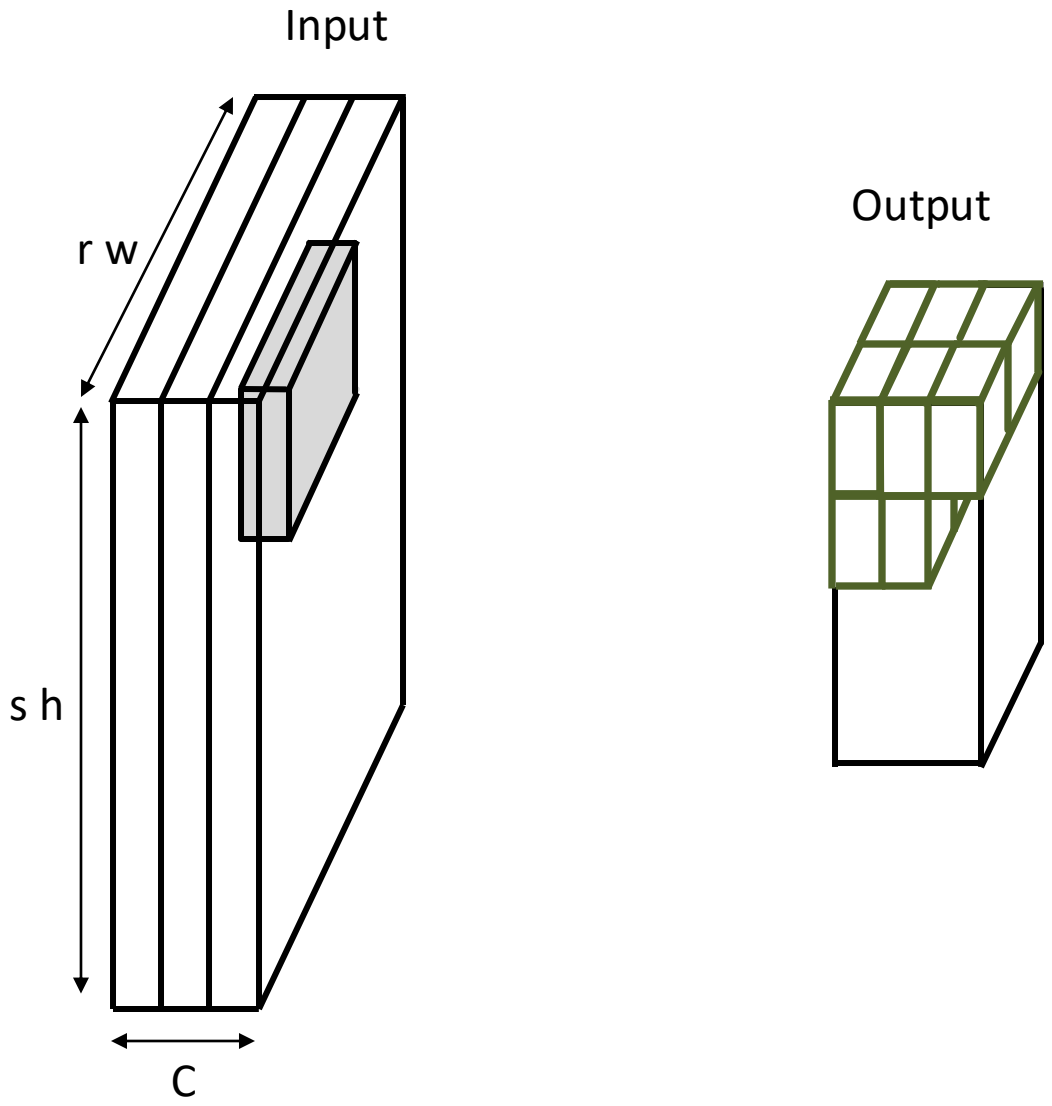
Source: Dr. Francois Fleuret at EPFL

Multi-channel Pooling



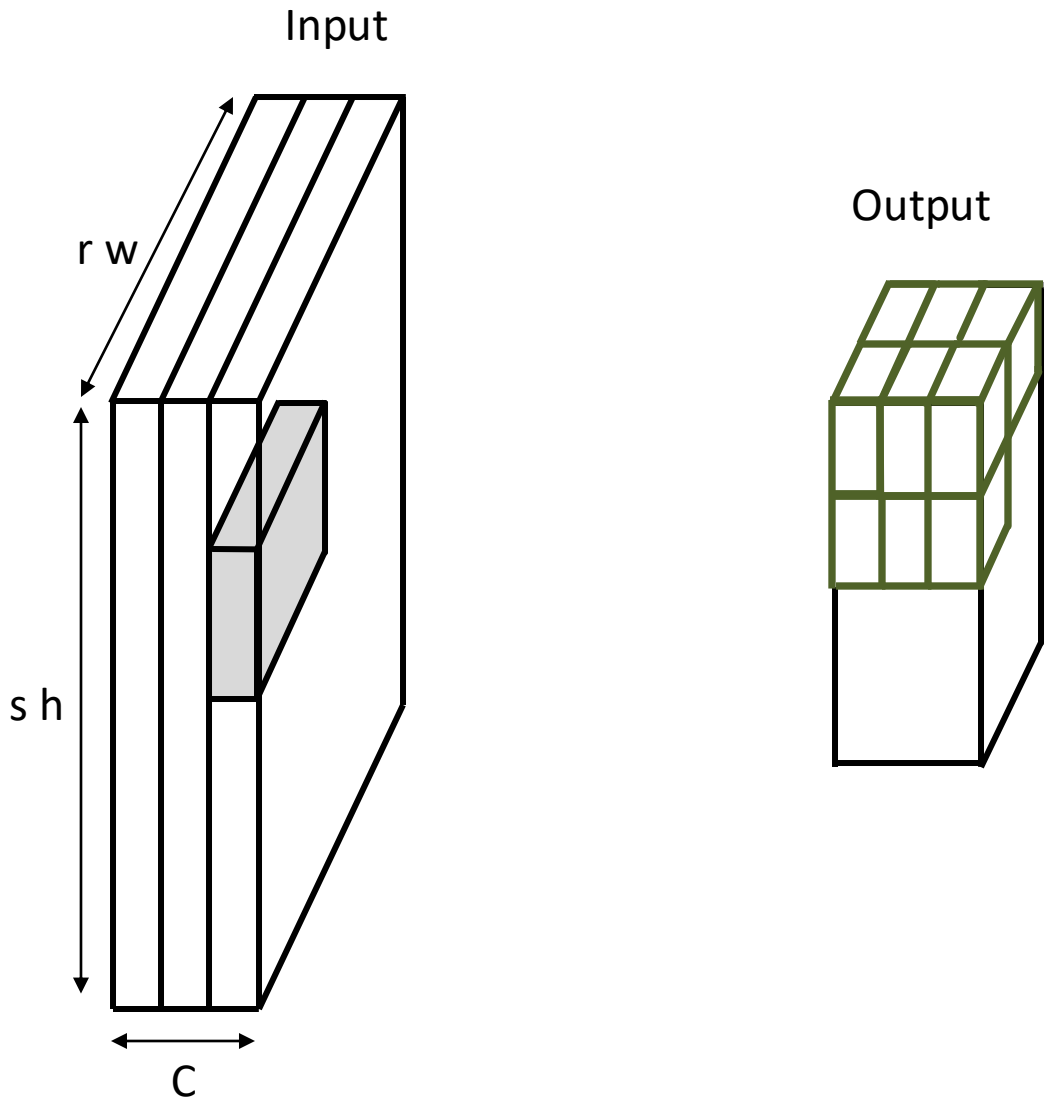
Source: Dr. Francois Fleuret at EPFL

Multi-channel Pooling



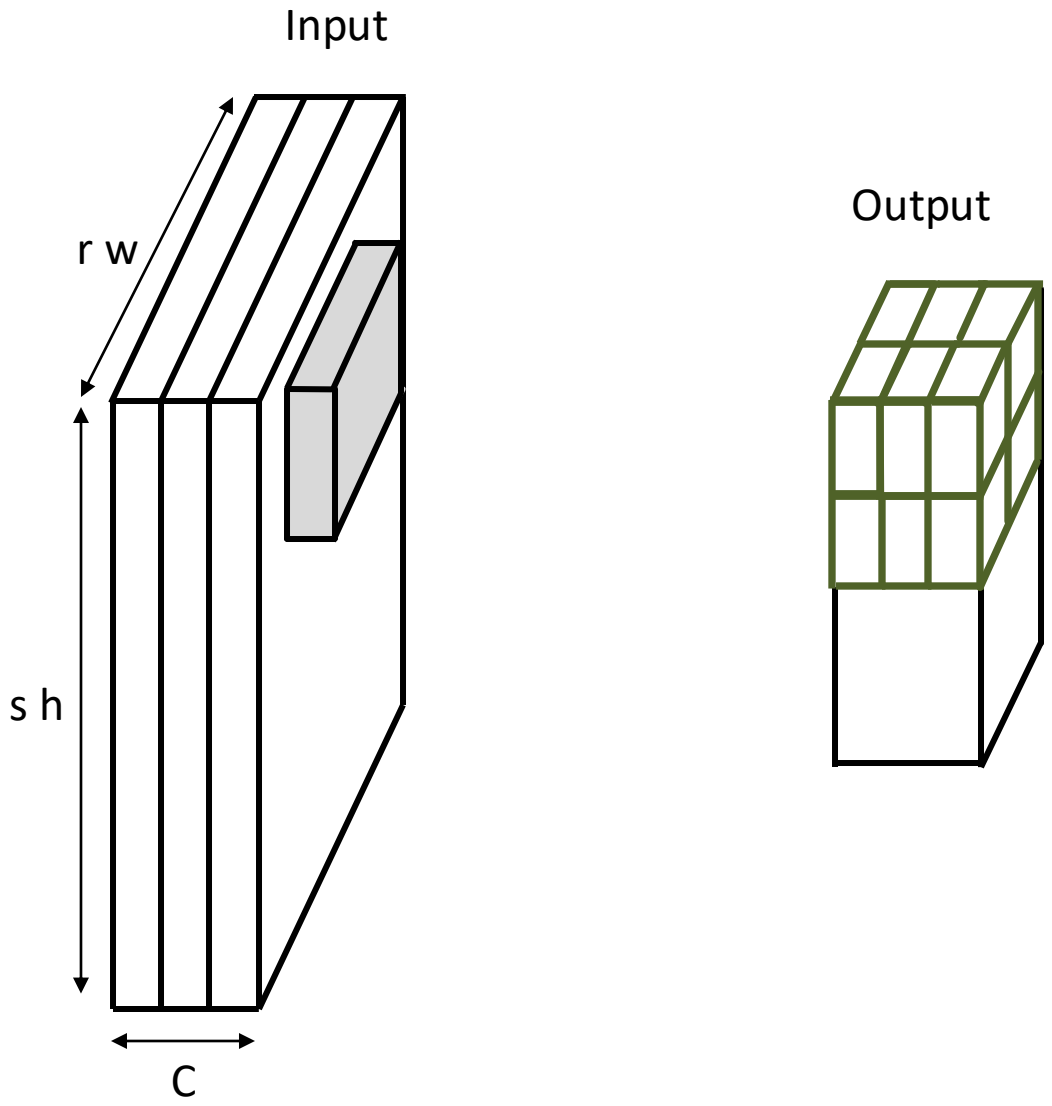
Source: Dr. Francois Fleuret at EPFL

Multi-channel Pooling



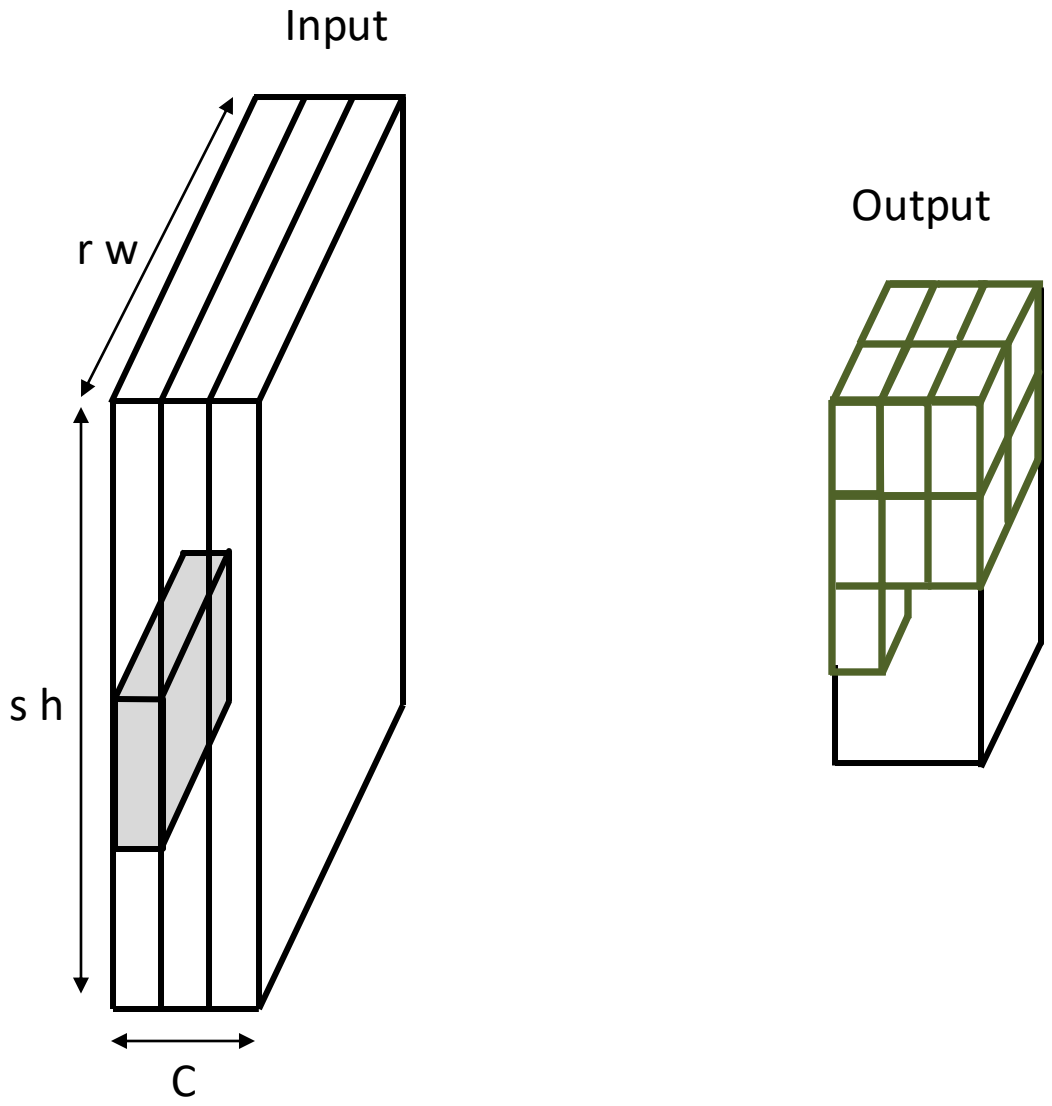
Source: Dr. Francois Fleuret at EPFL

Multi-channel Pooling



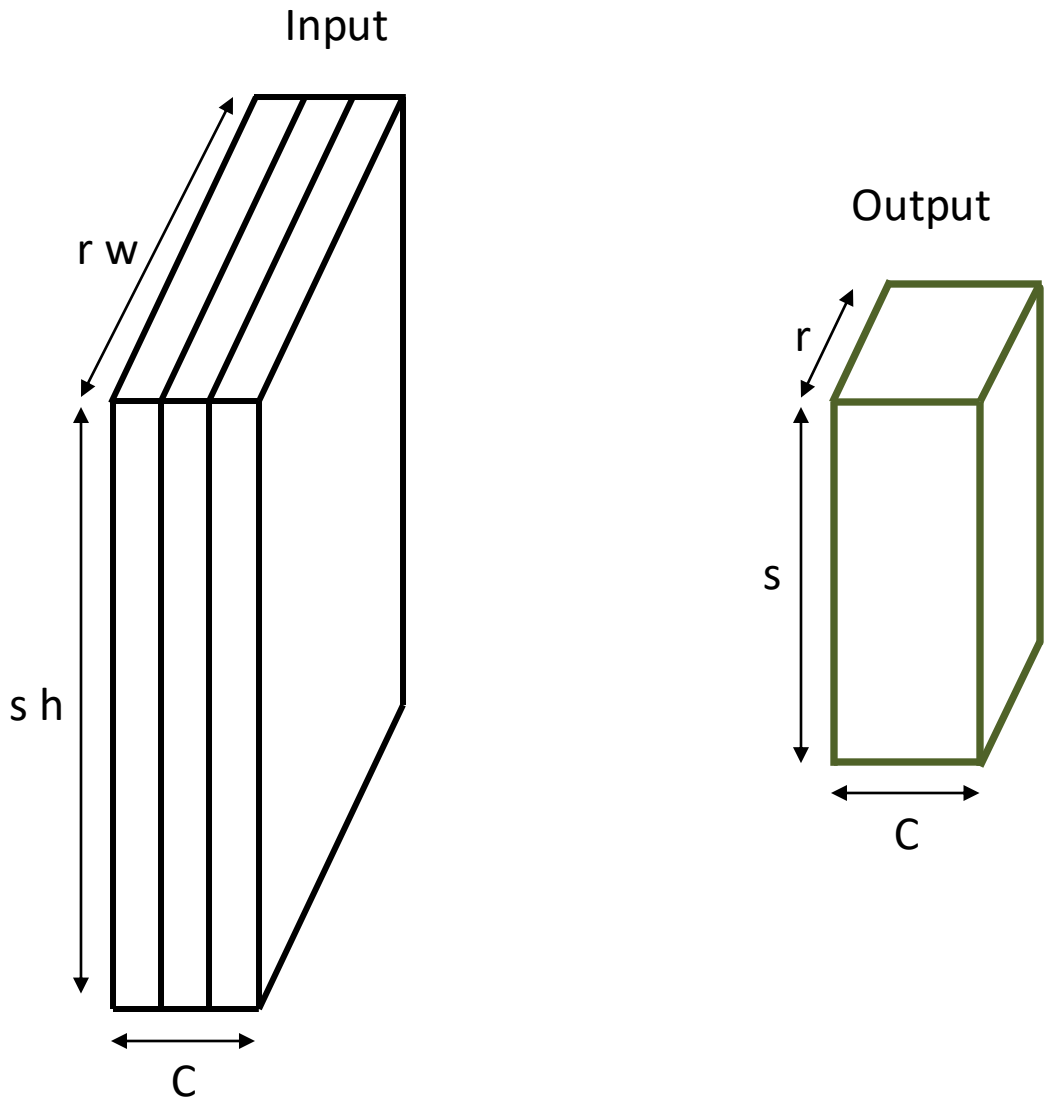
Source: Dr. Francois Fleuret at EPFL

Multi-channel Pooling



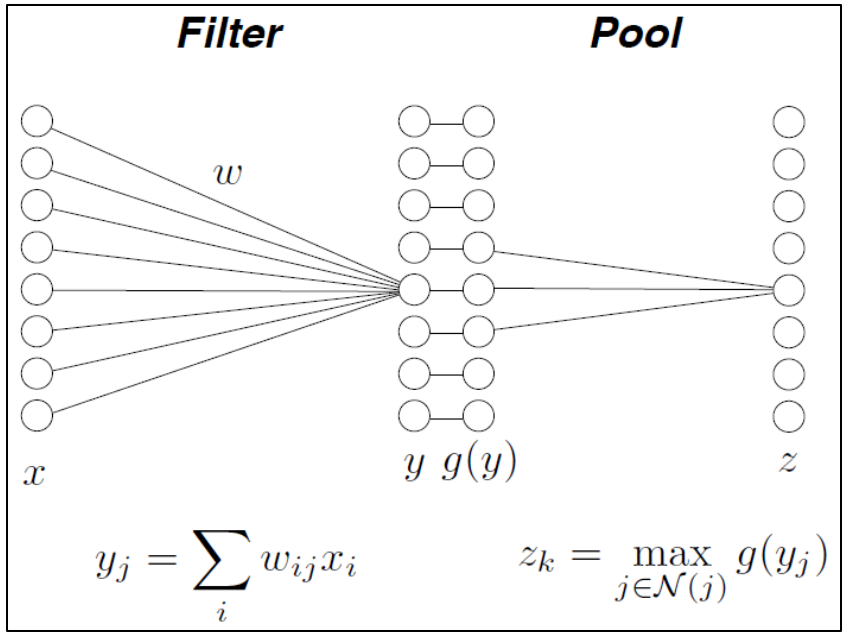
Source: Dr. Francois Fleuret at EPFL

Multi-channel Pooling

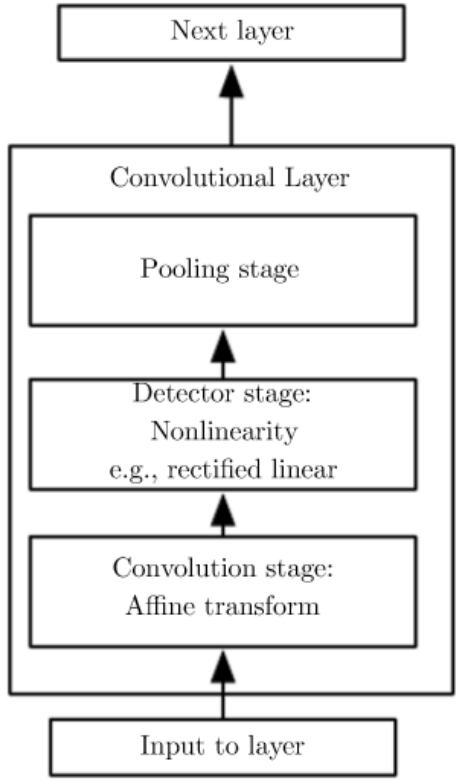


Source: Dr. Francois Fleuret at EPFL

Inside the Convolutional Layer Block



Conv blocks



Classic ConvNet Architecture

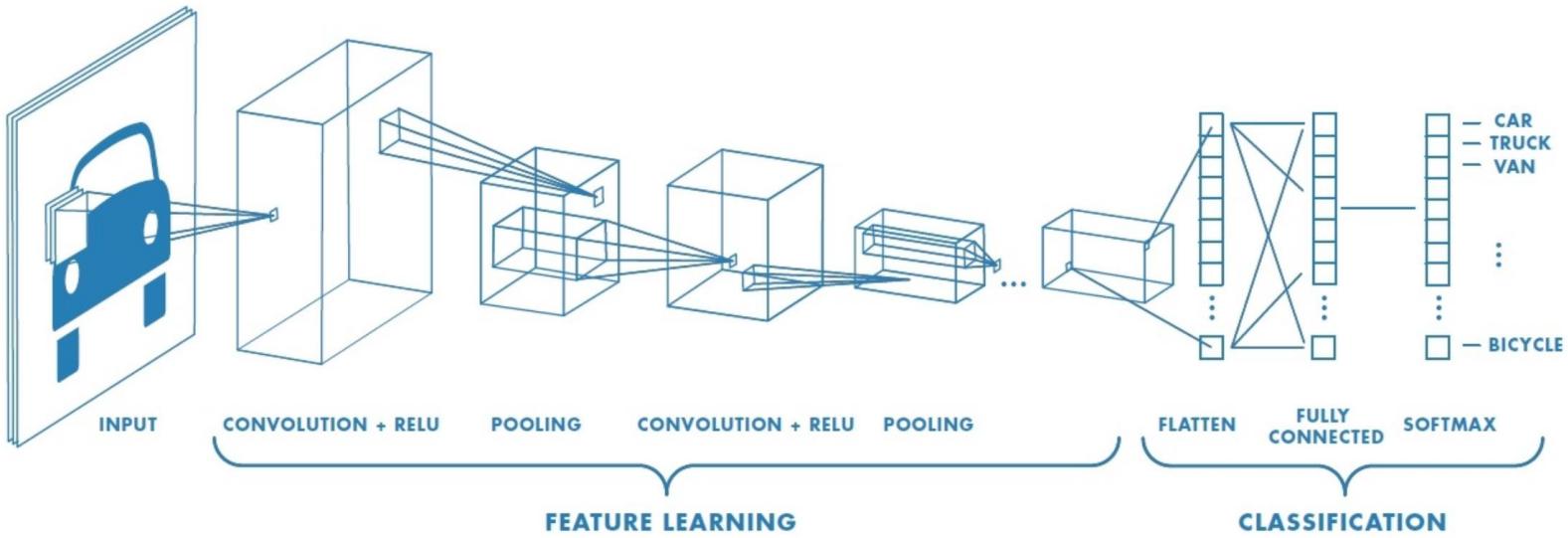
- Input

- Conv blocks
  - Convolution + activation (relu)
  - Convolution + activation (relu)
  - ...
  - Maxpooling

- Output
  - Fully connected layers
  - Softmax

CNNs: Training with Backpropagation

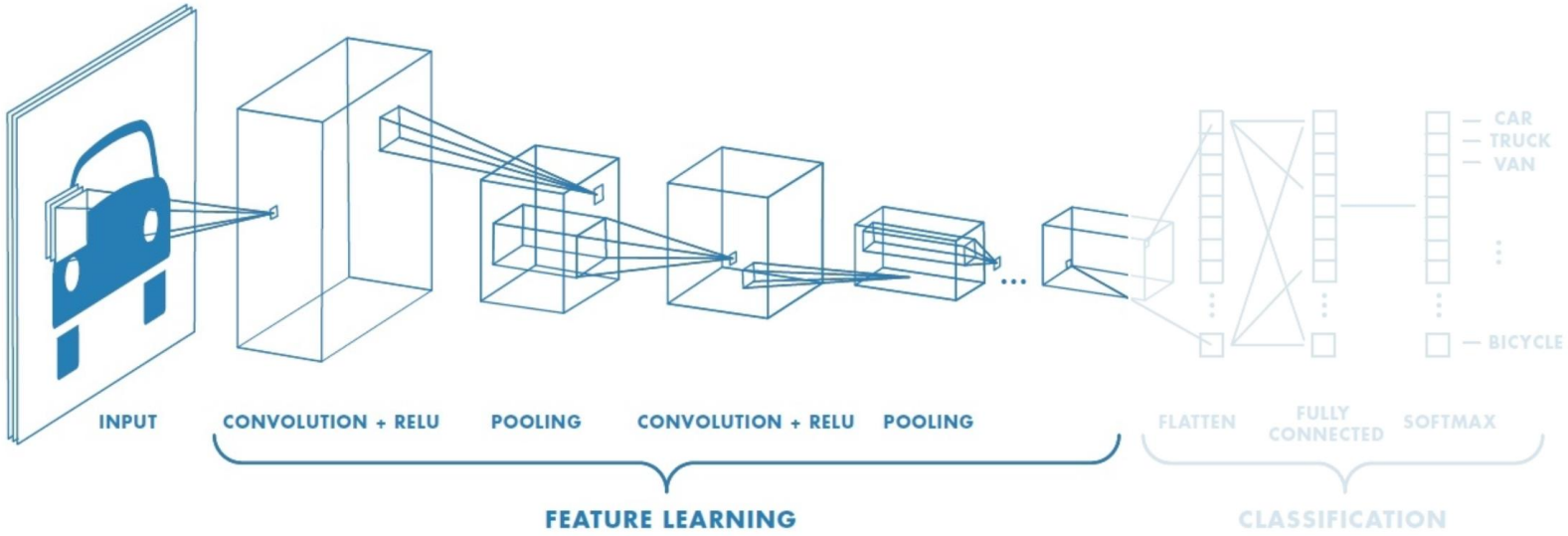
- Learn weights for convolutional filters and fully connected layers
- Backpropagation: cross-entropy loss



Source: 6.S191 Intro. to Deep Learning at MIT

CNNs: Feature Learning + Classifier / Regressor

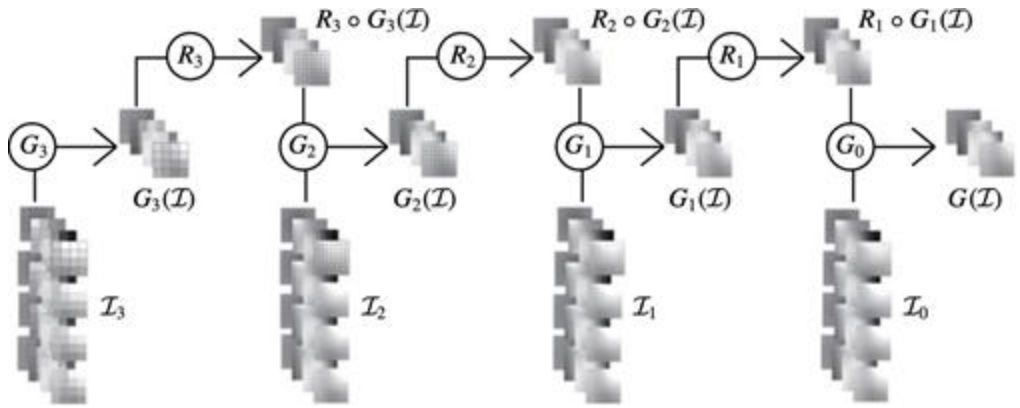
- Learn features in input image through convolution

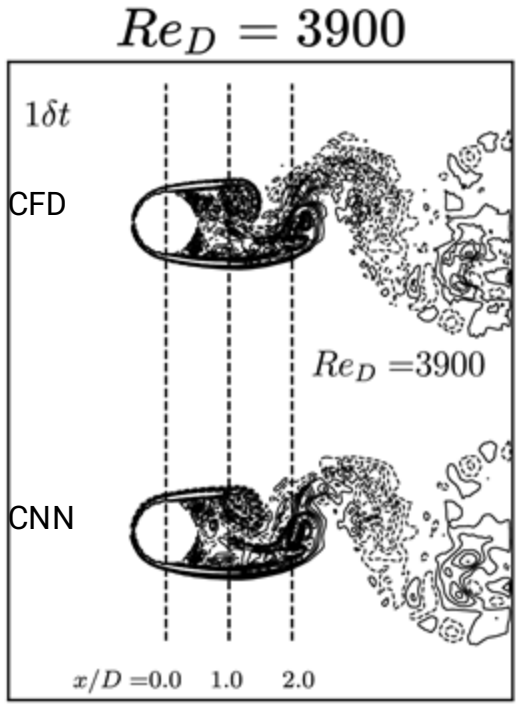


Source: 6.S191 Intro. to Deep Learning at MIT

CNNs for Engineering: (1) Predicting flow at different Reynolds Numbers

Lee & You
(2017, 2019 JFM)

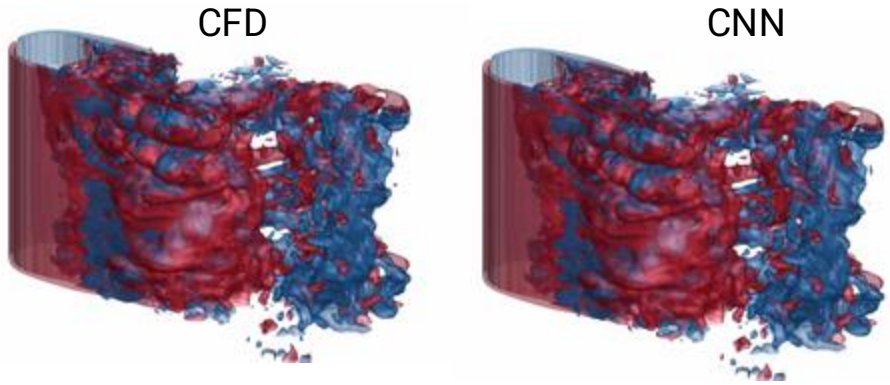


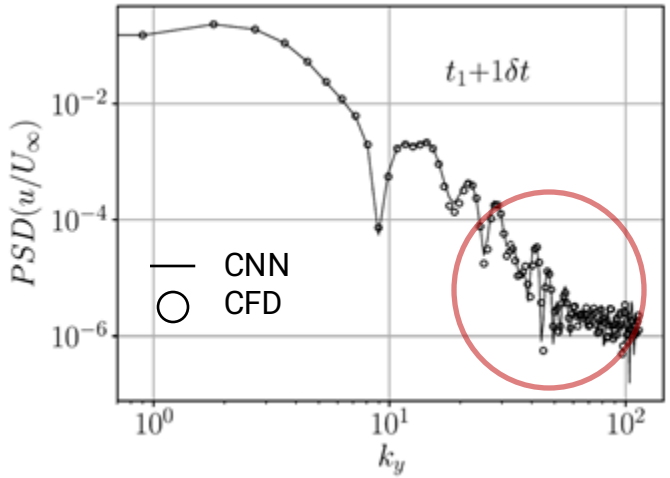


Prediction of flow at a higher Reynolds number (Re_D=3900) than trained (Re_D=300, 500)

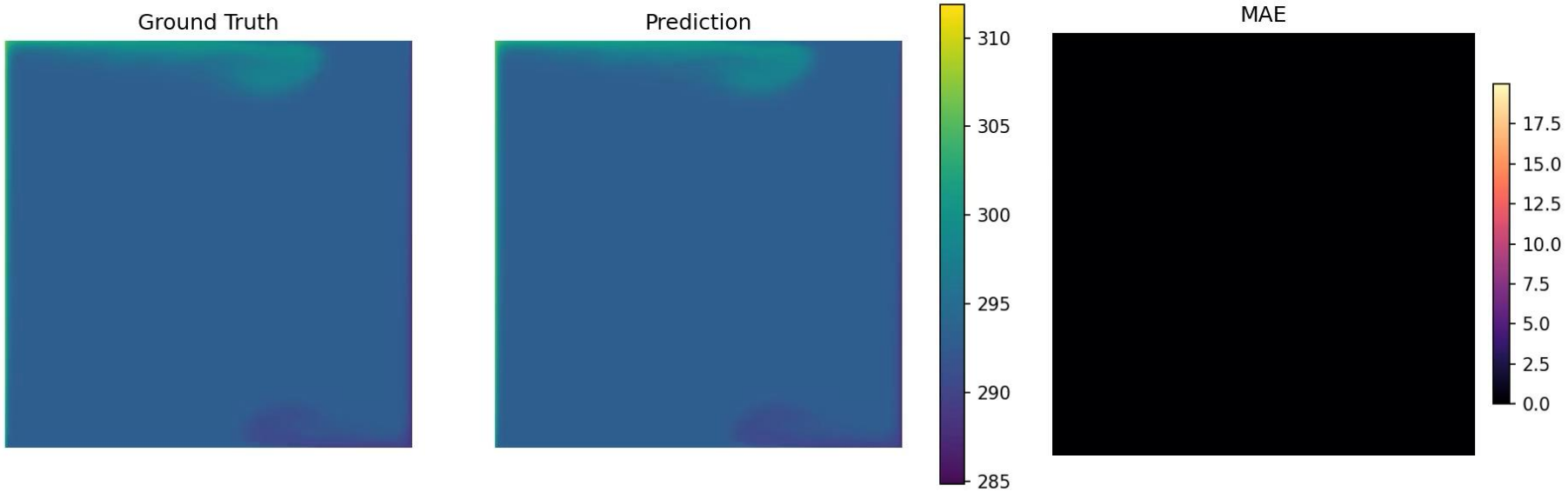
- Large scale convection captured
- Small-scale motions not captured

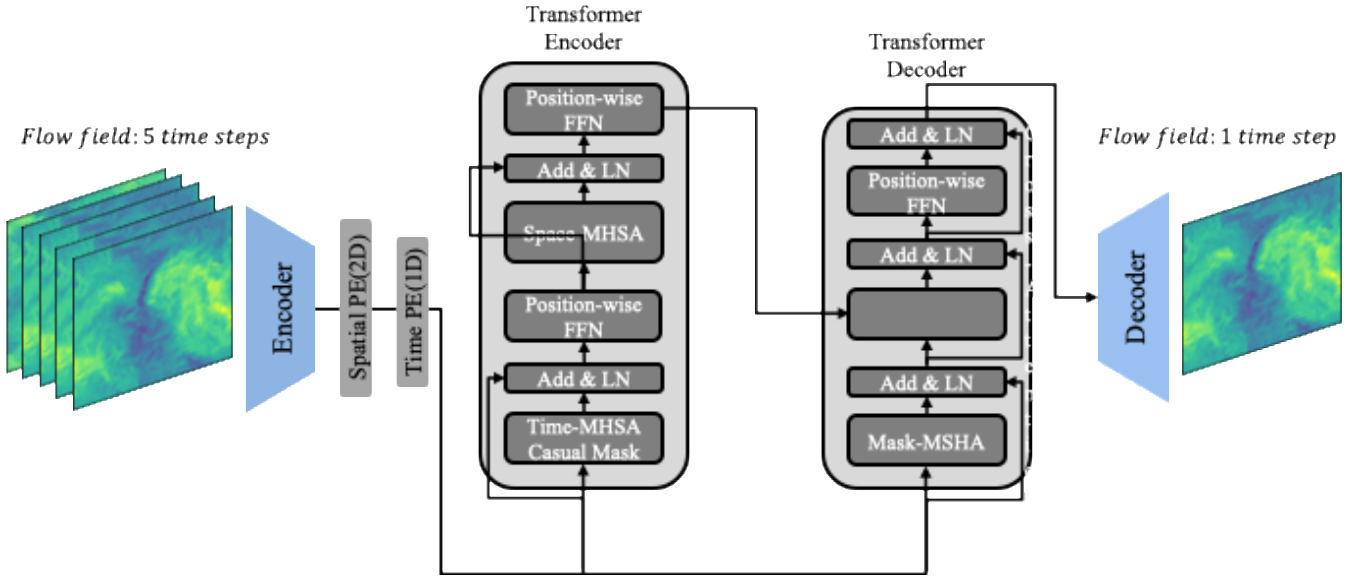
Lee & You
(2021 POF)



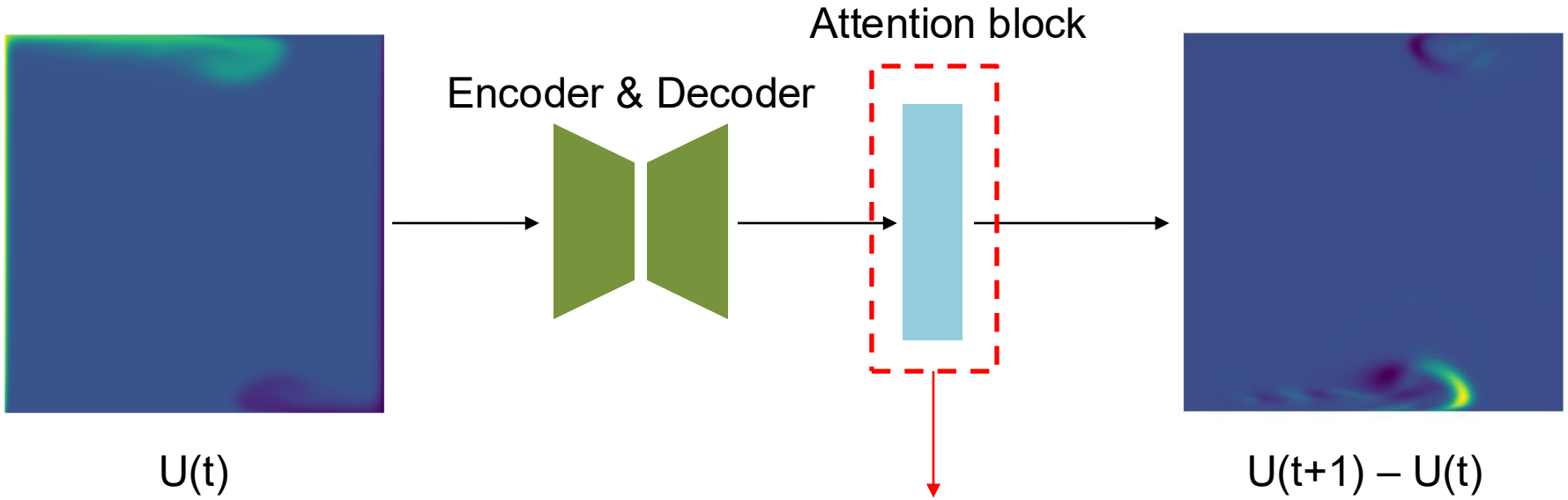


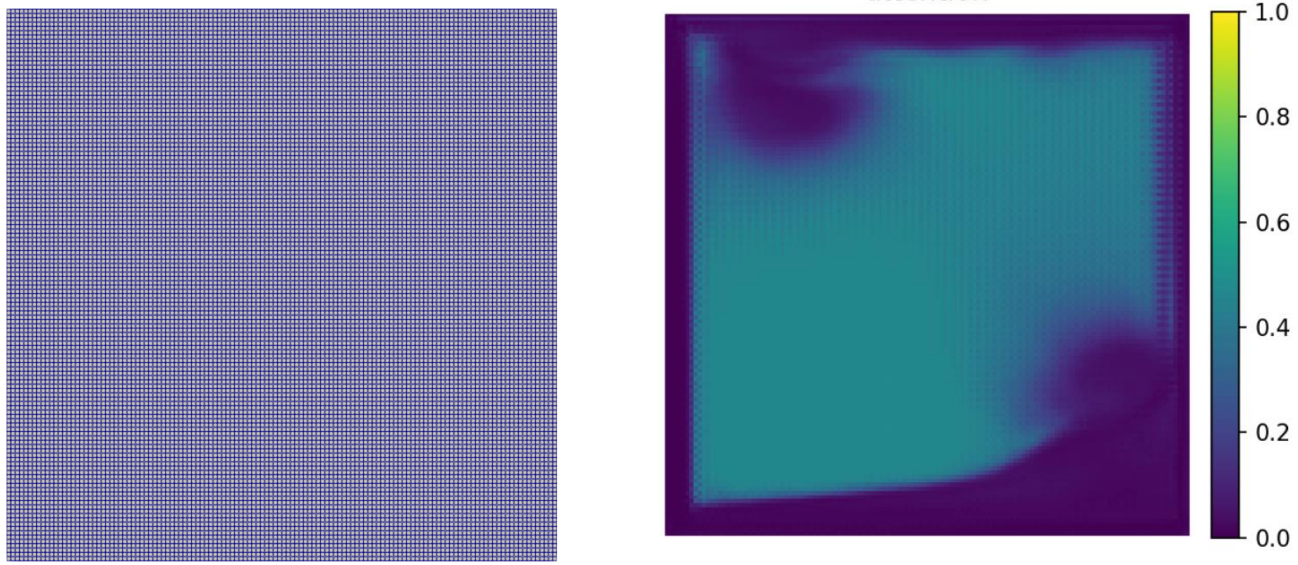
CNNs for Engineering: (2) Predicting Natural Circulation





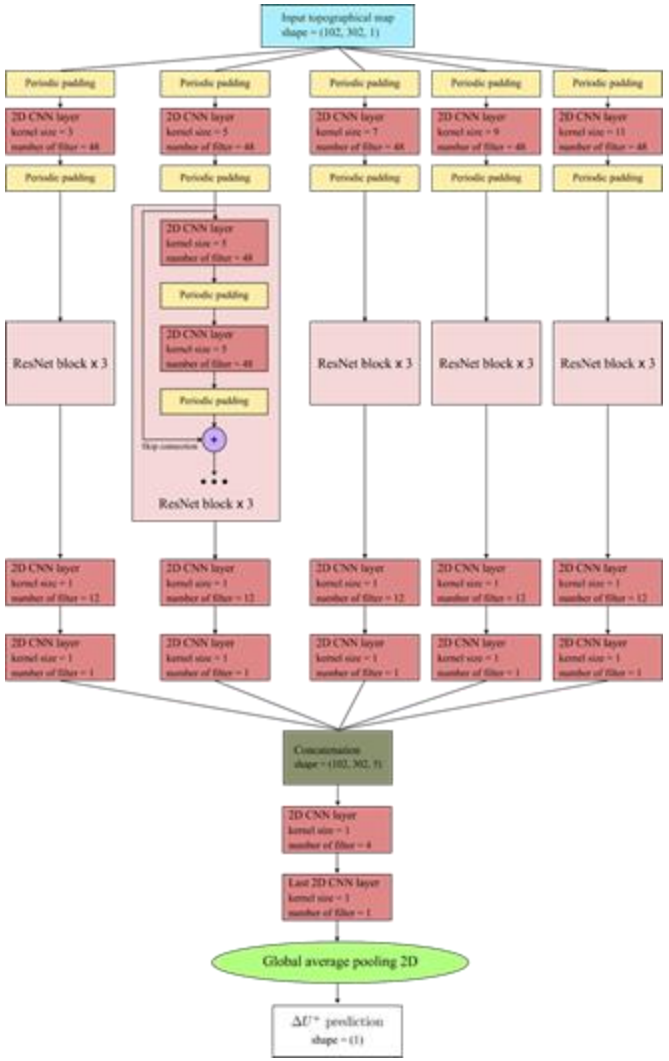
CNNs for Engineering: (3) Adaptive-Mesh-Refinement

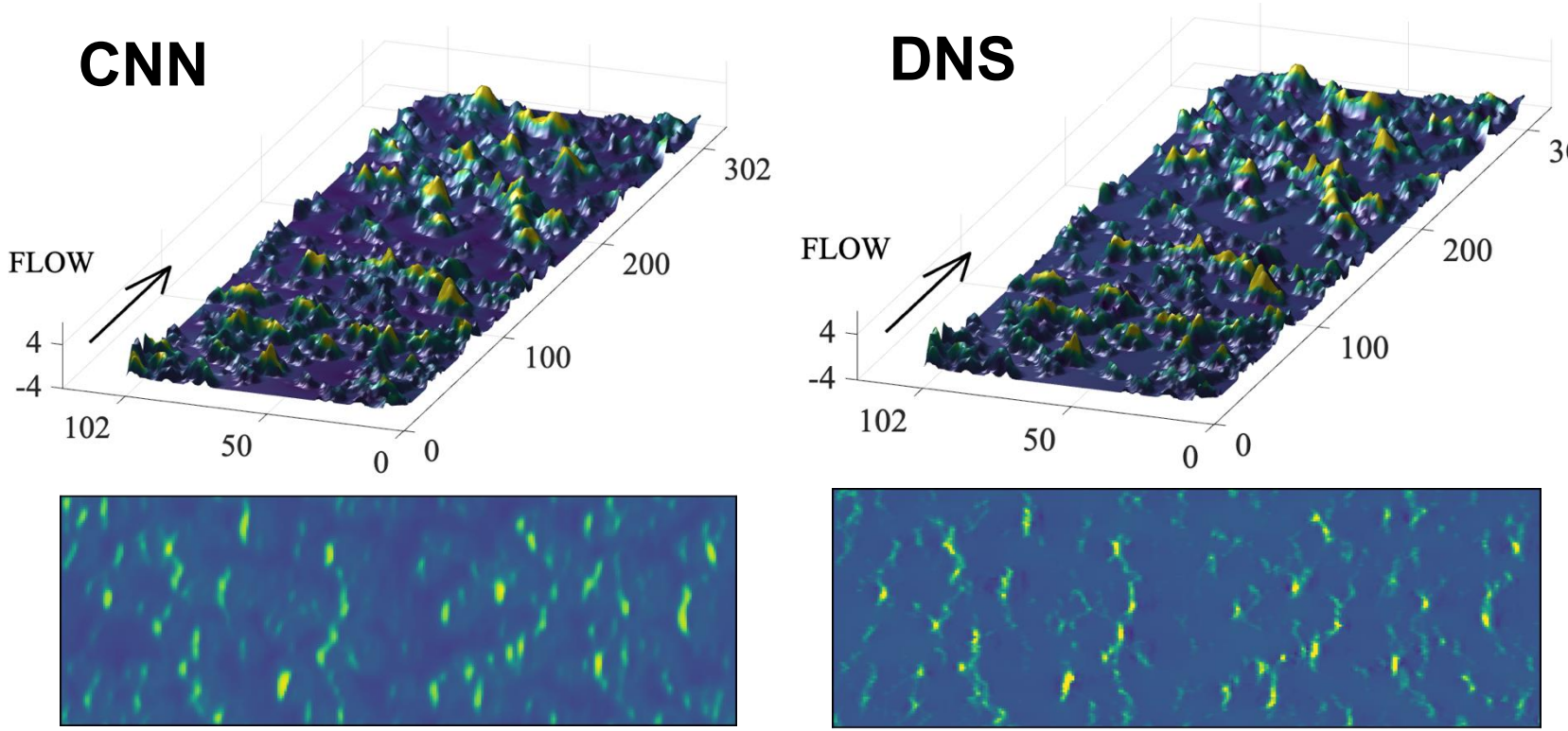




- **Generate a refineFlag** to refine areas above or below a certain value on **the attention map**

CNNs for Engineering: (4) Data-Driven-Discovery





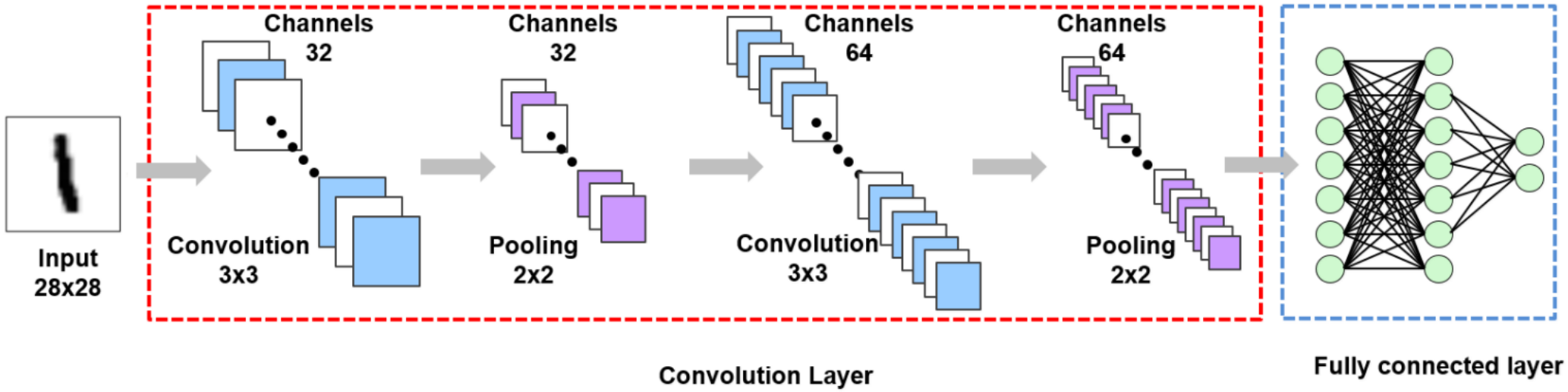
Discovering drag-inducing roughness structures

*Note: CNN haven't seen the DNS Drag Map during training
(Only trained with scalar value (ΔU^+))

CNN in TensorFlow

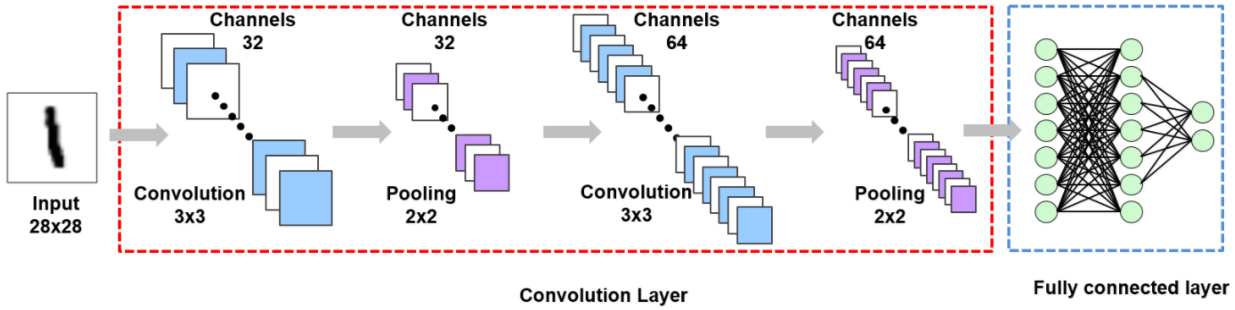
Lab: CNN with TensorFlow

- MNIST example
- To classify handwritten digits
- https://drive.google.com/file/d/18jBjc0GRCHFnam6joFVynR3j2KKDvkKl/view?usp=sharing



CNN Structure

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(filters = 32,
                           kernel_size = (3,3),
                           activation = 'relu',
                           padding = 'SAME',
                           input_shape = (28, 28, 1)),

    tf.keras.layers.MaxPool2D((2,2)),

    tf.keras.layers.Conv2D(filters = 64,
                           kernel_size = (3,3),
                           activation = 'relu',
                           padding = 'SAME',
                           input_shape = (14, 14, 32)),

    tf.keras.layers.MaxPool2D((2,2)),

    tf.keras.layers.Flatten(),

    tf.keras.layers.Dense(units = 128, activation = 'relu'),

    tf.keras.layers.Dense(units = 10, activation = 'softmax')
])
```



Loss and Optimizer

- Loss
  - Classification: Cross entropy
  - Equivalent to applying logistic regression
- Optimizer
  - GradientDescentOptimizer
  - AdamOptimizer: the most popular optimizer

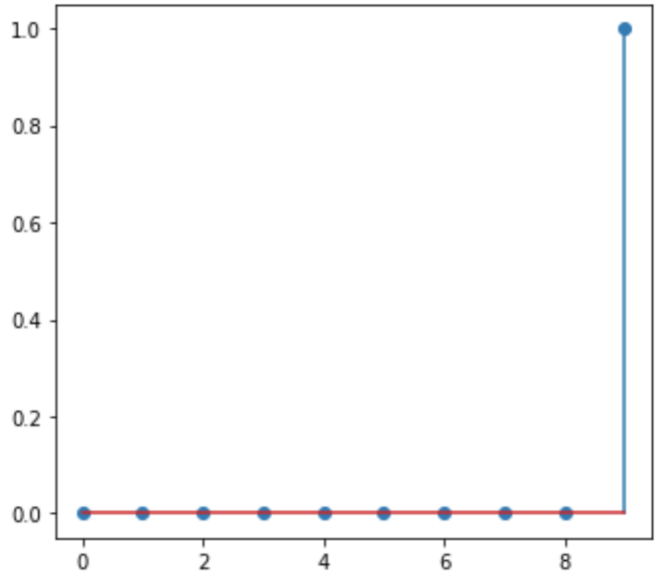
```
model.compile(optimizer = 'adam',
              loss = 'sparse_categorical_crossentropy',
              metrics = ['accuracy'])
```

```
model.fit(train_x, train_y)
```

Test or Evaluation

```
test_loss, test_acc = model.evaluate(test_x, test_y)
```

```
313/313 [==============================] - 1s 4ms/step - accuracy: 0.9838 - loss: 0.0466
loss = 0.05, Accuracy = 98 %
```

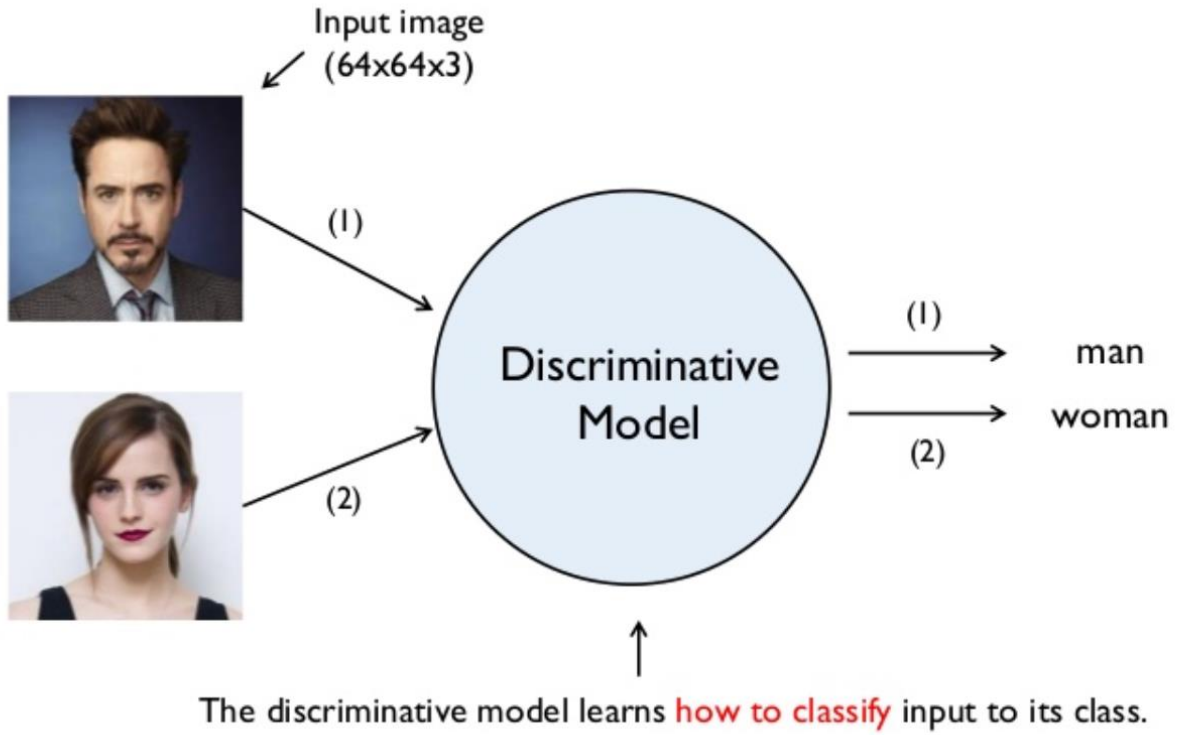



```
Prediction : 9
```

Generative Adversarial Networks (GAN)

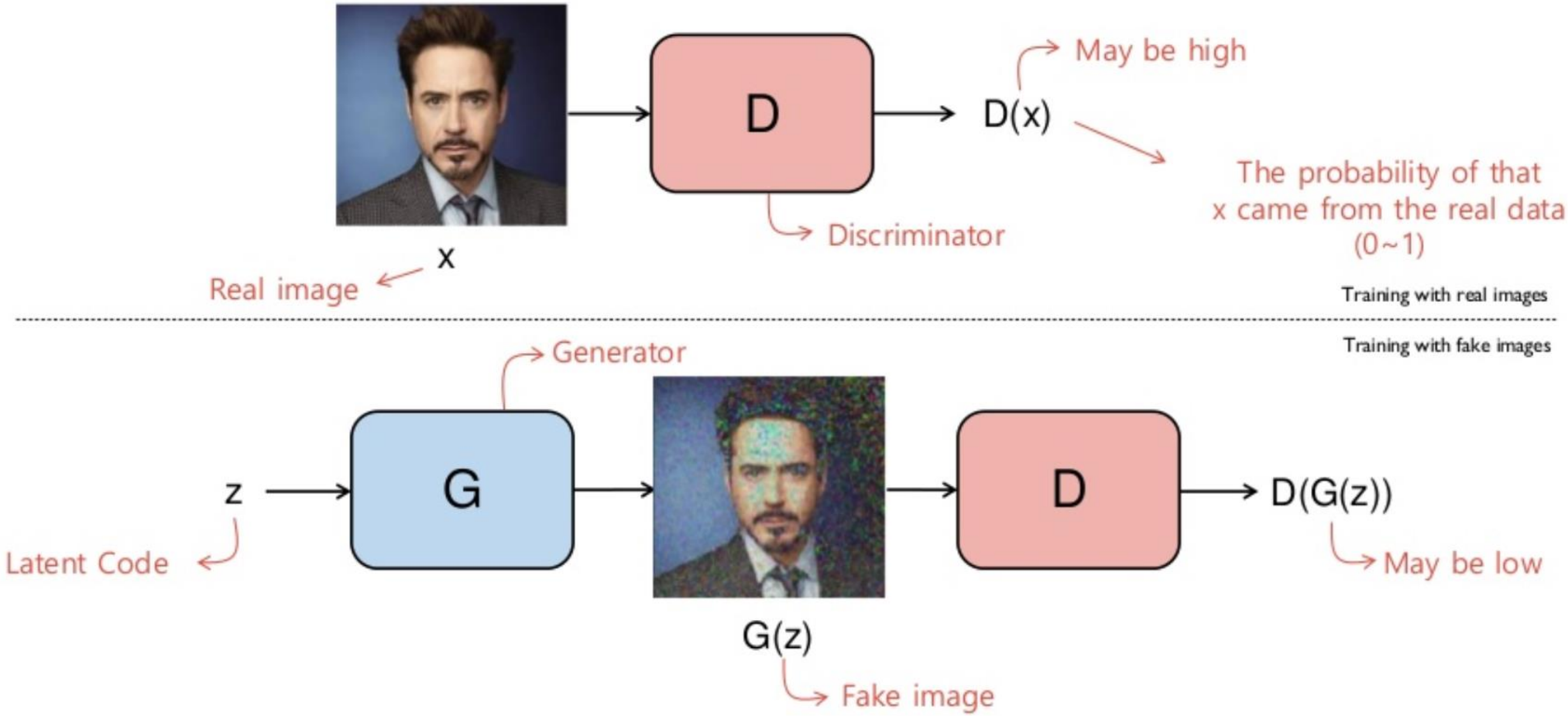
Supervised Learning

- Discriminative model



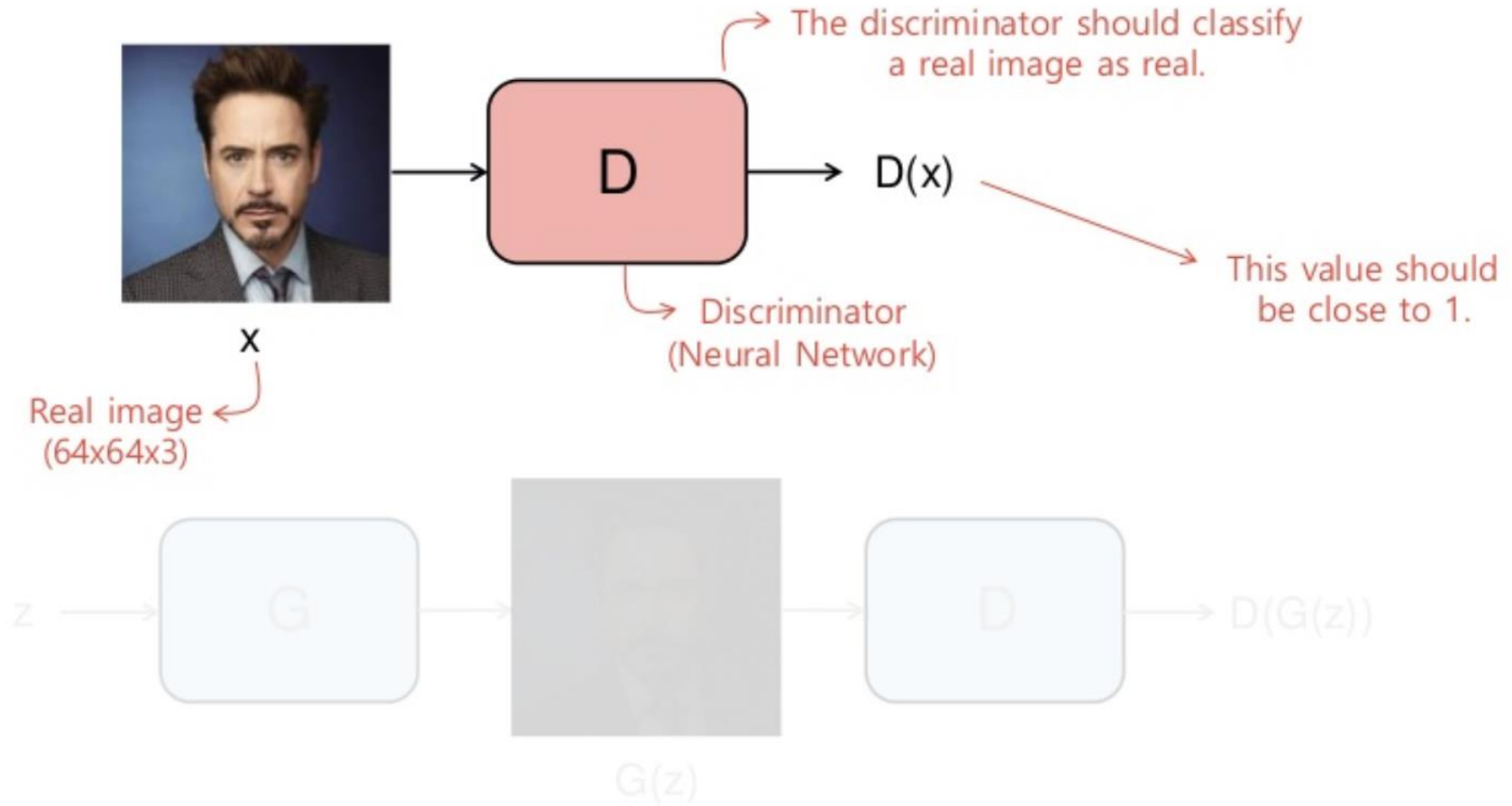
https://www.slideshare.net/NaverEngineering/1-gangenerative-adversarial-network

Intuition for GAN



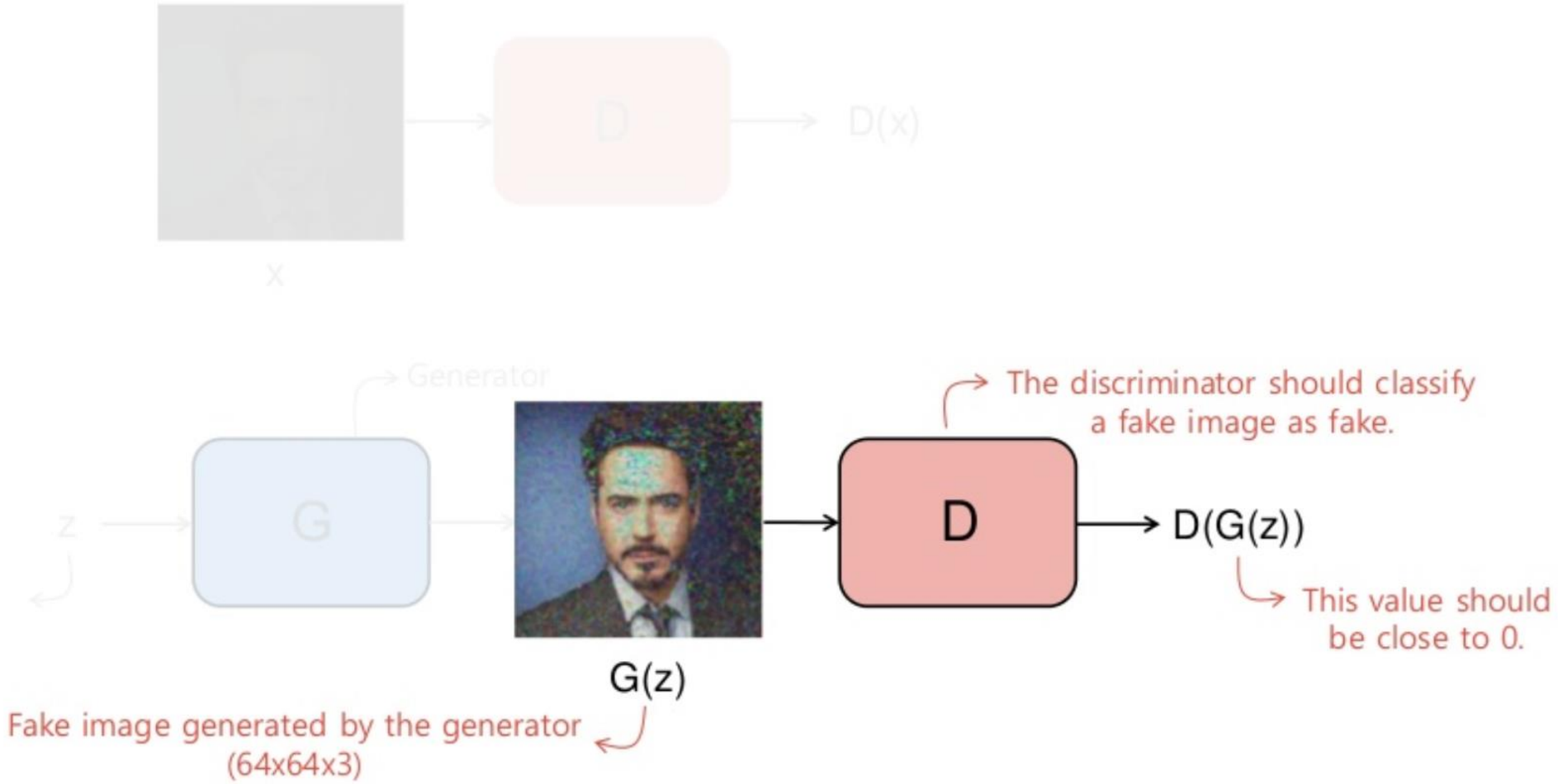
https://www.slideshare.net/NaverEngineering/1-gangenerative-adversarial-network

Discriminator Perspective (1/2)



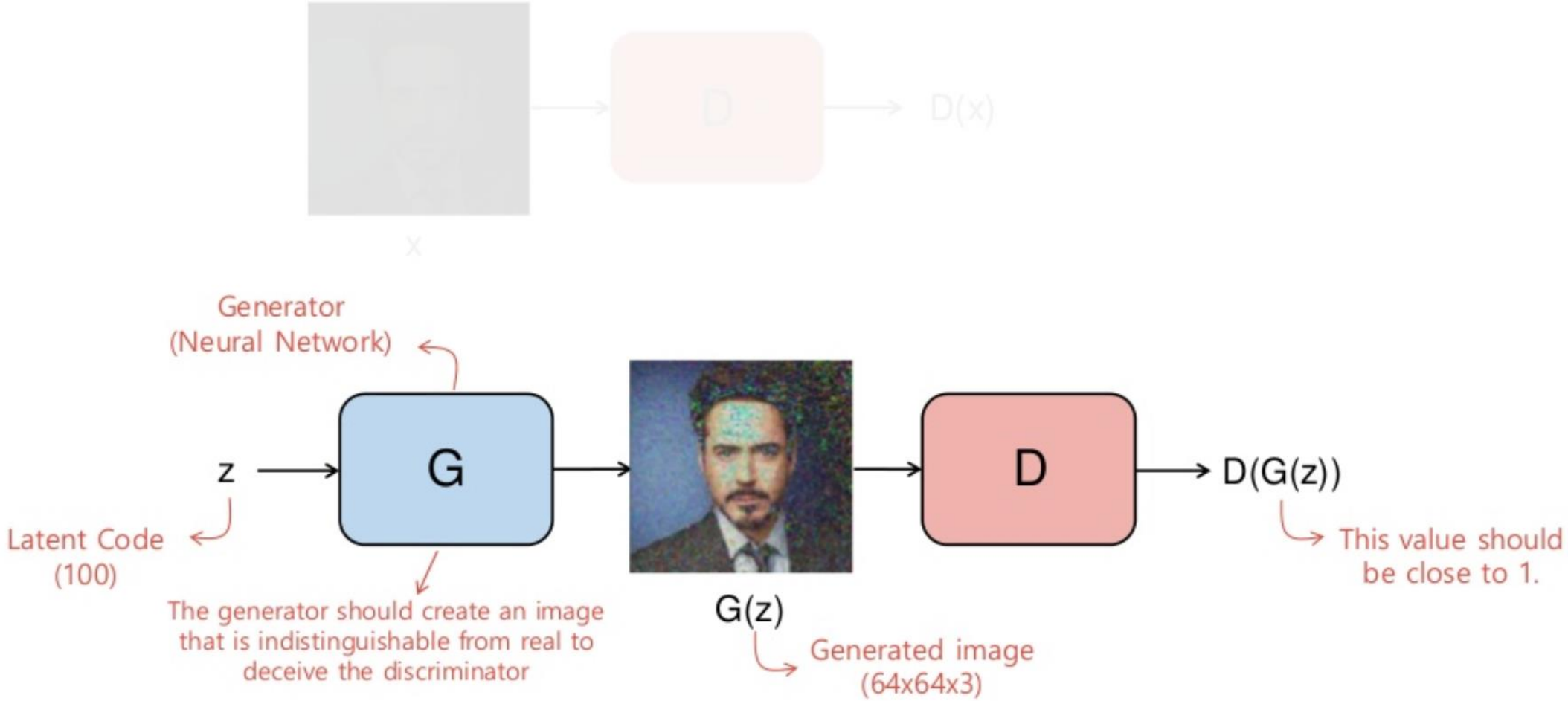
https://www.slideshare.net/NaverEngineering/1-gangenerative-adversarial-network

Discriminator Perspective (2/2)



https://www.slideshare.net/NaverEngineering/1-gangenerative-adversarial-network

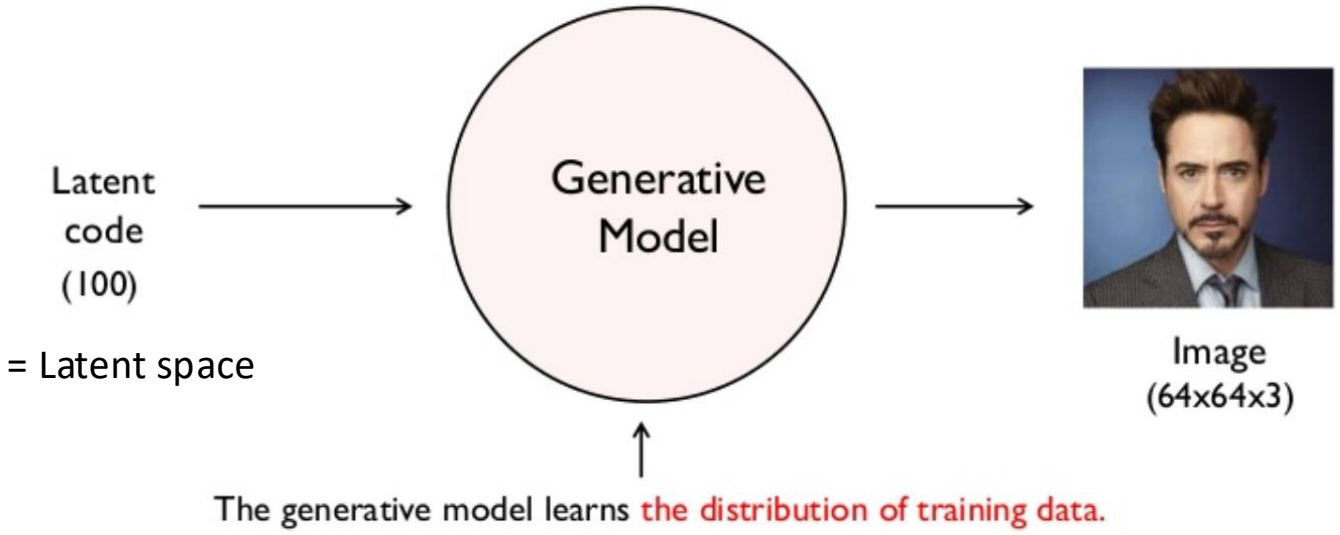
Generator Perspective



https://www.slideshare.net/NaverEngineering/1-gangenerative-adversarial-network

Unsupervised Learning

- Generative model



https://www.slideshare.net/NaverEngineering/1-gangenerative-adversarial-network

Loss Function

Total Loss function

$$\min_G \max_D V(D,G) = E_{x\sim p_{\text{data}}(x)}[\log D(x)] + E_{z\sim p_z(z)}[\log(1 - D(G(z)))]$$

Fix Generator and train Discriminator

$$\min_D E_{x\sim p_{\text{data}}(x)}[-\log D(x)] + E_{z\sim p_z(z)}[-\log(1 - D(G(z)))]$$

Real Data: D(x) = 1
Fake Data (G(z)): D(G(z)) = 0
Discriminator = **binary classifier**

Fix Discriminator and train Generator

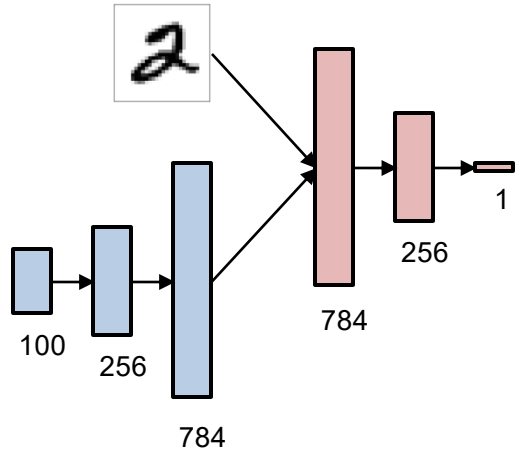
$$\min_G E_{z\sim p_z(z)}[-\log D(G(z))]$$

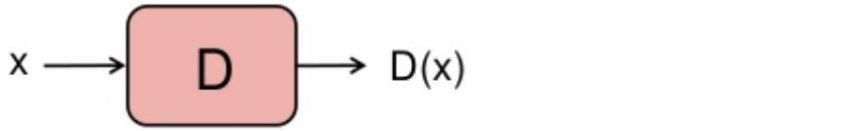
Train G to make D(G(z)) close to **1 (fool Discriminator)**

GAN Implementation in TensorFlow

TensorFlow Implementation

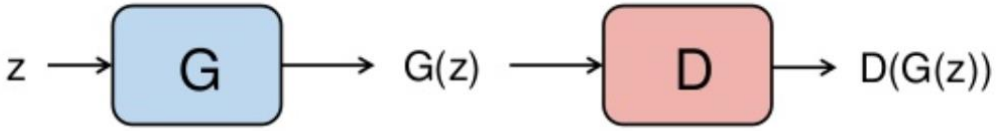
https://drive.google.com/file/d/13sH-PUQavPFJGZR_jC87Pf0zowAQGnzN/view?usp=sharing





Training with real images

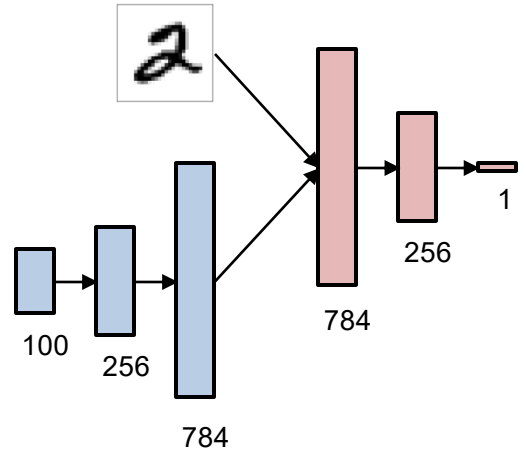
Training with fake images

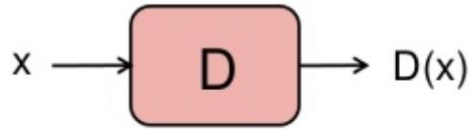


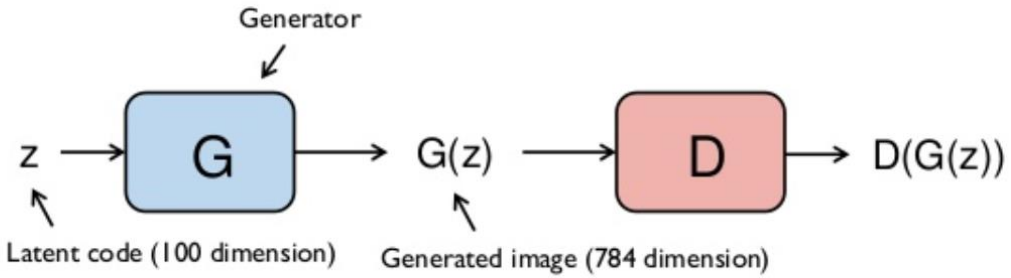
https://www.slideshare.net/NaverEngineering/1-gangenerative-adversarial-network

Generator

```
generator = tf.keras.models.Sequential([
    tf.keras.layers.Dense(units = 256, input_dim = 100, activation = 'relu'),
    tf.keras.layers.Dense(units = 784, activation = 'sigmoid')
])
```



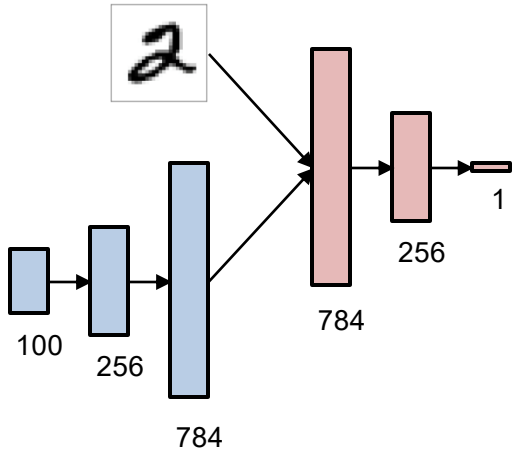


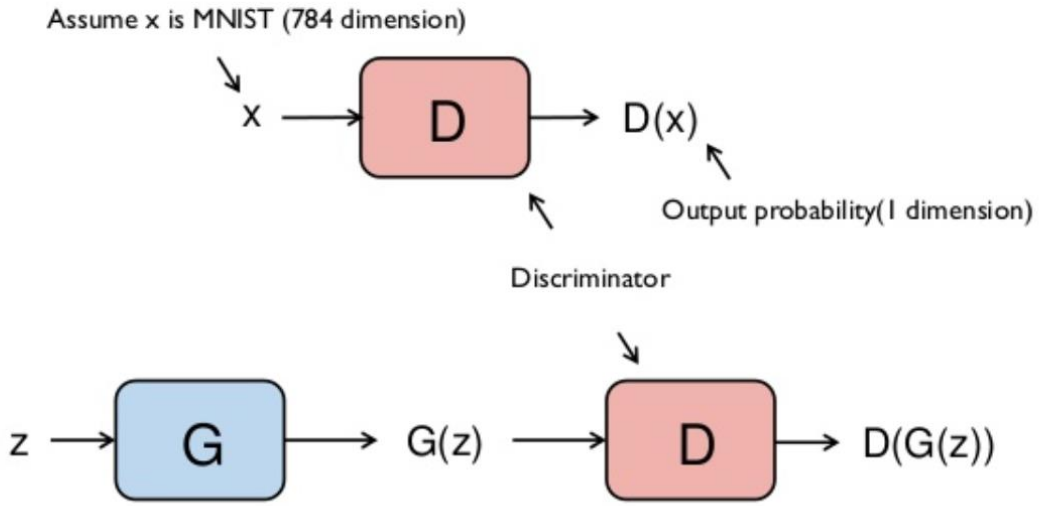


https://www.slideshare.net/NaverEngineering/1-gangenerative-adversarial-network

Discriminator

```
discriminator = tf.keras.models.Sequential([
    tf.keras.layers.Dense(units = 256, input_dim = 784, activation = 'relu'),
    tf.keras.layers.Dense(units = 1, activation = 'sigmoid'),
])
```

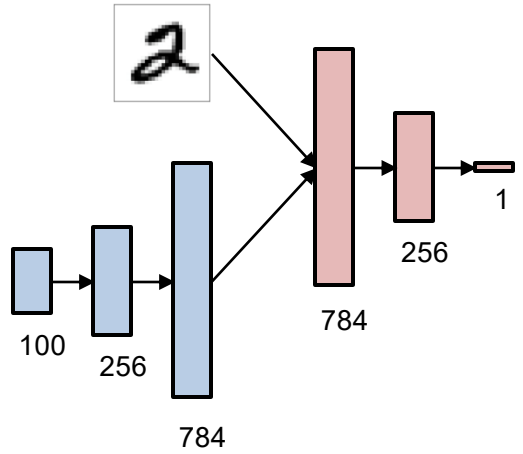


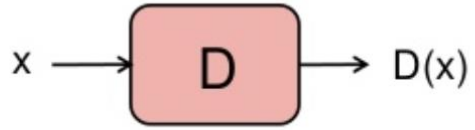


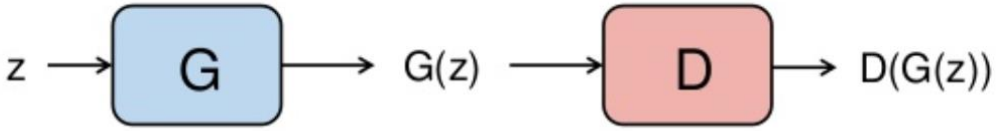
https://www.slideshare.net/NaverEngineering/1-gangenerative-adversarial-network

Combined

```
combined_input = tf.keras.layers.Input(shape = (100,))
generated = generator(combined_input)
discriminator.trainable = False
combined_output = discriminator(generated)

combined = tf.keras.models.Model(inputs = combined_input, outputs = combined_output)
```

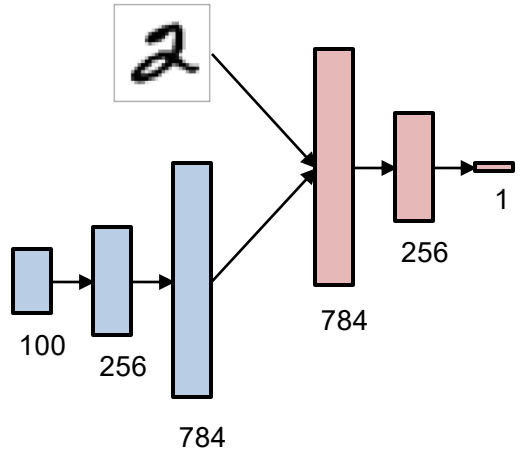


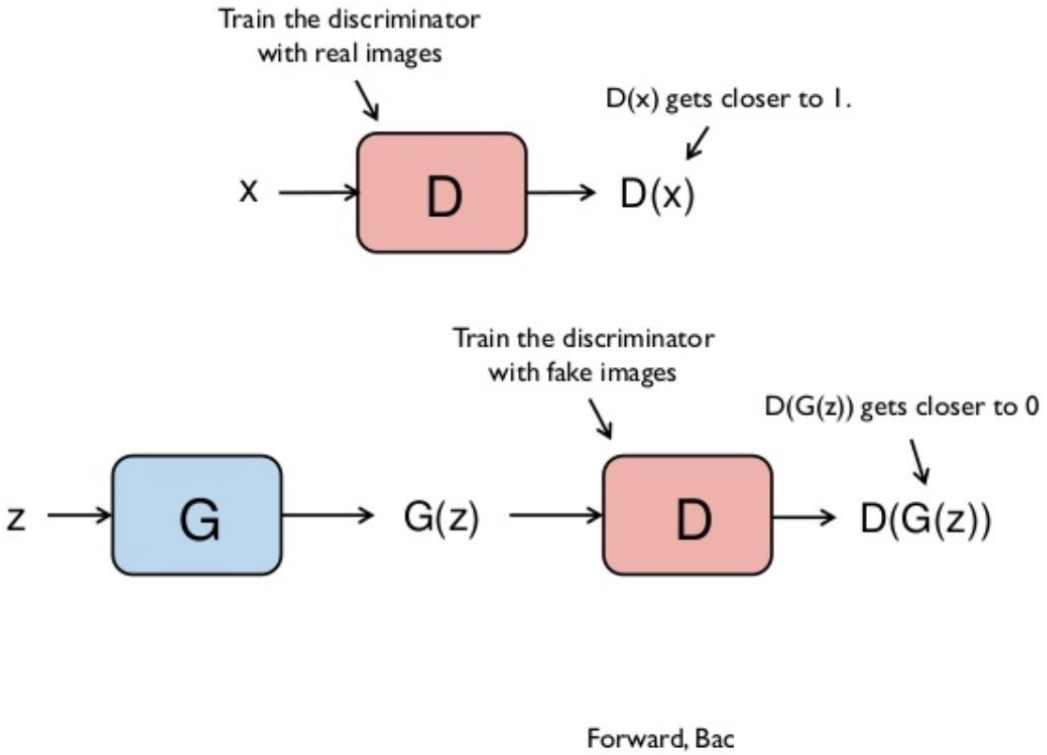




https://www.slideshare.net/NaverEngineering/1-gangenerative-adversarial-network

Training: Discriminator

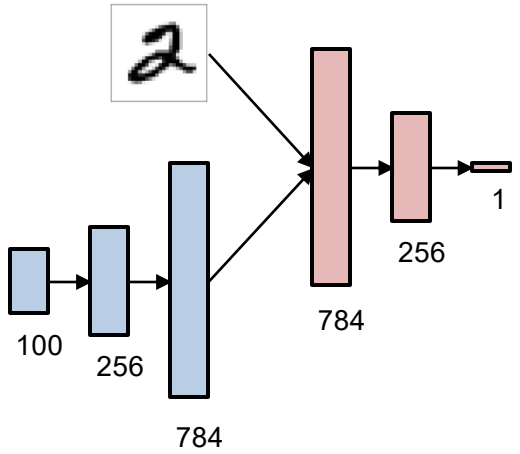


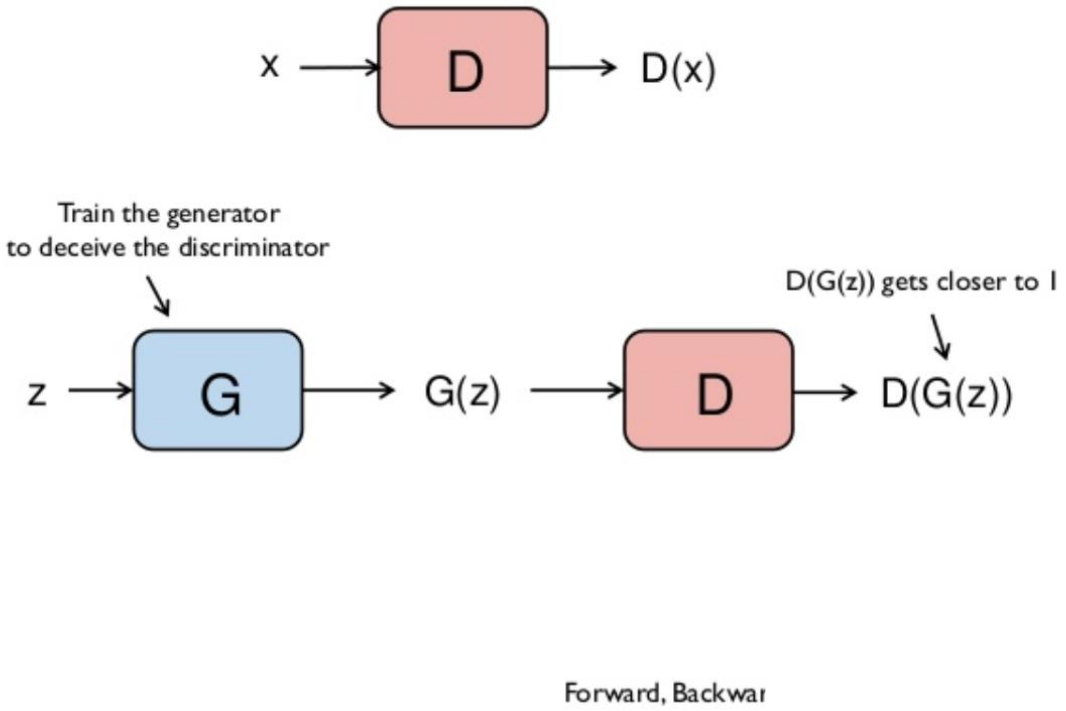



```
n_iter = 5000
batch_size = 50

fake = np.zeros(batch_size)
real = np.ones(batch_size)

for i in range(n_iter):

    # Train Discriminator
    noise = make_noise(batch_size)
    generated_images = generator.predict(noise)

    idx = np.random.randint(0, train_x.shape[0], batch_size)
    real_images = train_x[idx]

    D_loss_real = discriminator.train_on_batch(real_images, real)
    D_loss_fake = discriminator.train_on_batch(generated_images, fake)
    D_loss = D_loss_real + D_loss_fake

    # Train Generator
    noise = make_noise(batch_size)
    G_loss = combined.train_on_batch(noise, real)
```

https://www.slideshare.net/NaverEngineering/1-gangenerative-adversarial-network

Training: Generator



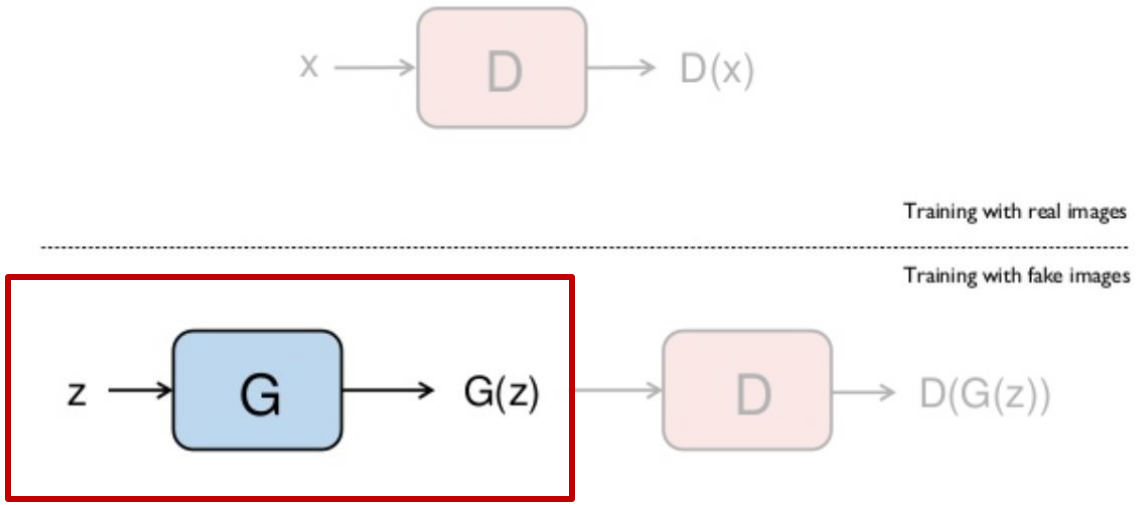



```
n_iter = 5000
batch_size = 50

fake = np.zeros(batch_size)
real = np.ones(batch_size)

for i in range(n_iter):

    # Train Discriminator
    noise = make_noise(batch_size)
    generated_images = generator.predict(noise)

    idx = np.random.randint(0, train_x.shape[0], batch_size)
    real_images = train_x[idx]

    D_loss_real = discriminator.train_on_batch(real_images, real)
    D_loss_fake = discriminator.train_on_batch(generated_images, fake)
    D_loss = D_loss_real + D_loss_fake

    # Train Generator
    noise = make_noise(batch_size)
    G_loss = combined.train_on_batch(noise, real)
```

https://www.slideshare.net/NaverEngineering/1-gangenerative-adversarial-network

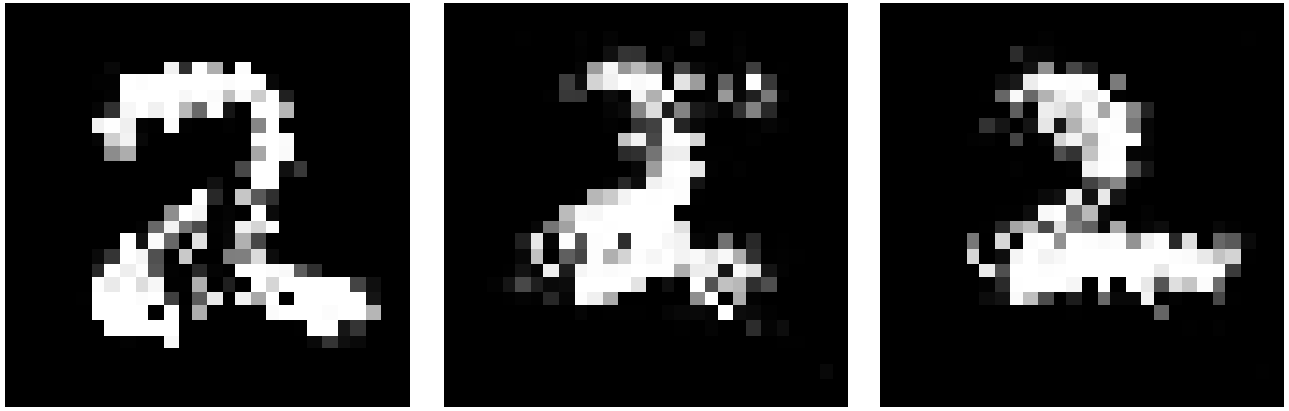
After Training

- After training, use generator network to generate new data



https://www.slideshare.net/NaverEngineering/1-gangenerative-adversarial-network

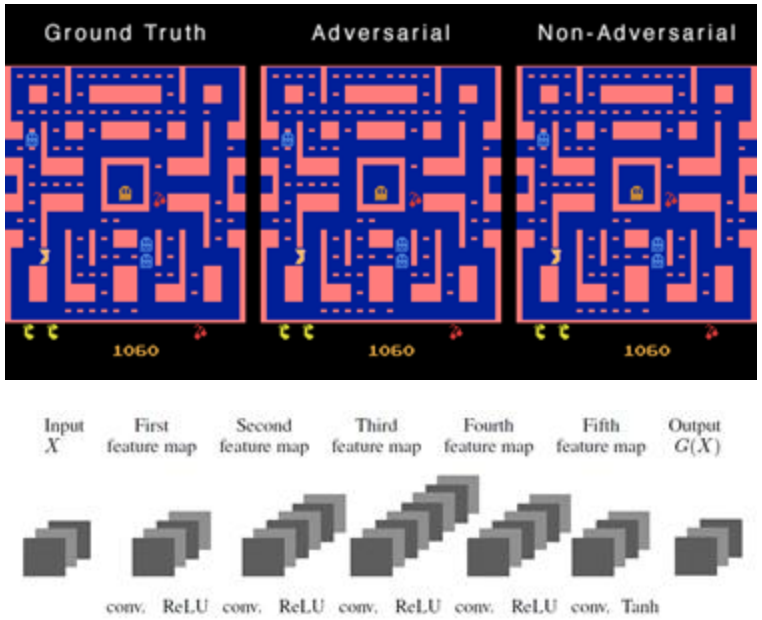
Generated Images



Conditional GAN

Conditional GAN

Video prediction AI model (CNN + GAN)

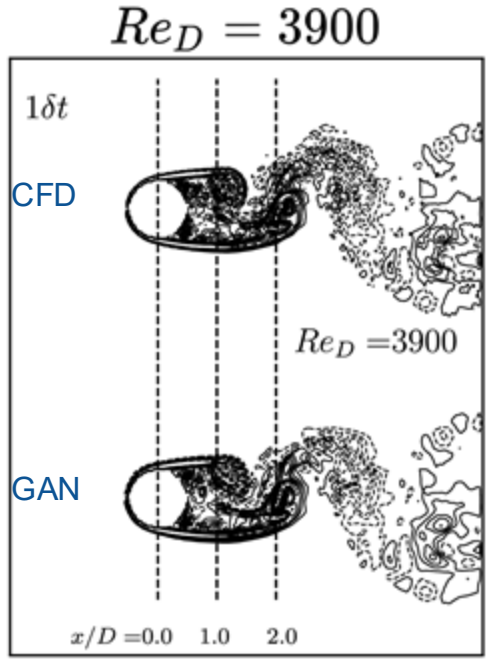


Deep Multi-Scale Video Prediction Beyond Mean Square Error (2016 ICLR)

2D unsteady flows

Lee & You (2017 arXiv, **2019 JFM**)

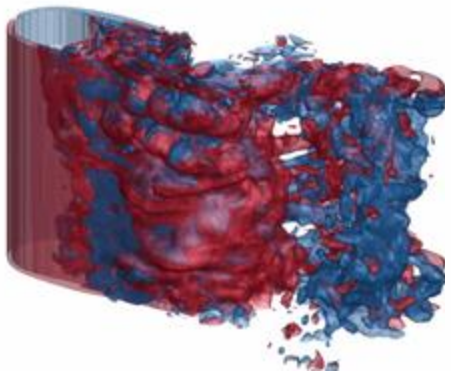
Input: Past 4 time-steps
Output: Future prediction
Train: e.g., Re 300, 500
Test: e.g., Re 3900
+ Physical loss functions
+ **Mass & Momentum**



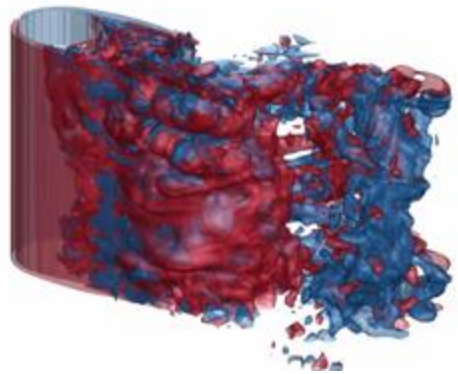
3D unsteady flows

Lee & You (2019 arXiv, 2021 POF)

CFD



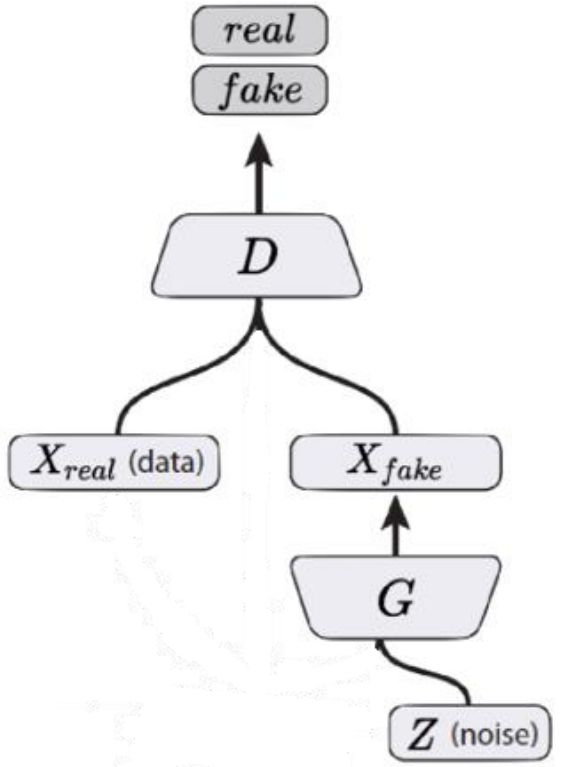
CNN



Conditional GAN

- Simple modification to the original GAN framework that conditions the model on additional information for better multi-modal learning

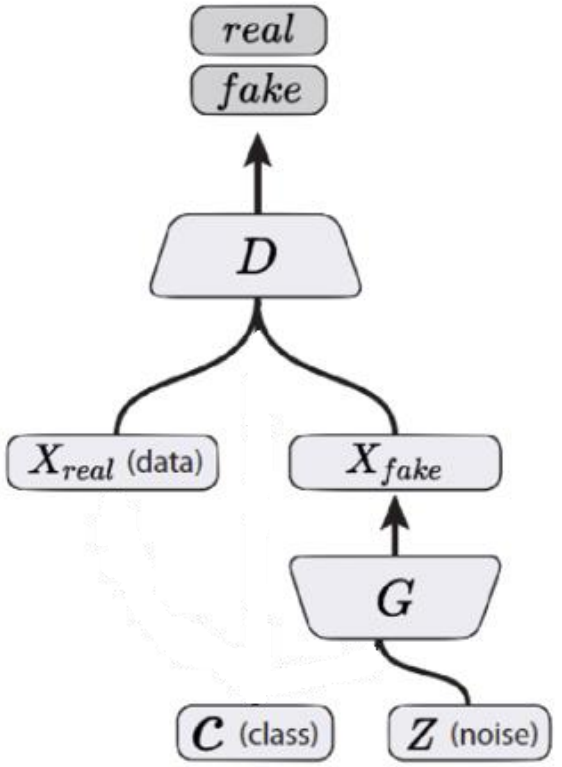
- Many practical applications of GANs when we have explicit supervision available



Conditional GAN

- Simple modification to the original GAN framework that conditions the model on additional information for better multi-modal learning

- Many practical applications of GANs when we have explicit supervision available

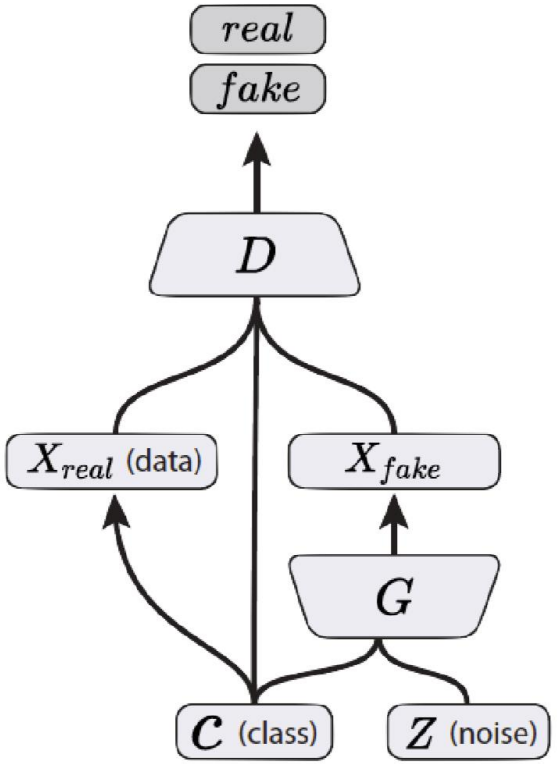


Conditional GAN
(Mirza & Osindero, 2014)

Conditional GAN

- Simple modification to the original GAN framework that conditions the model on additional information for better multi-modal learning

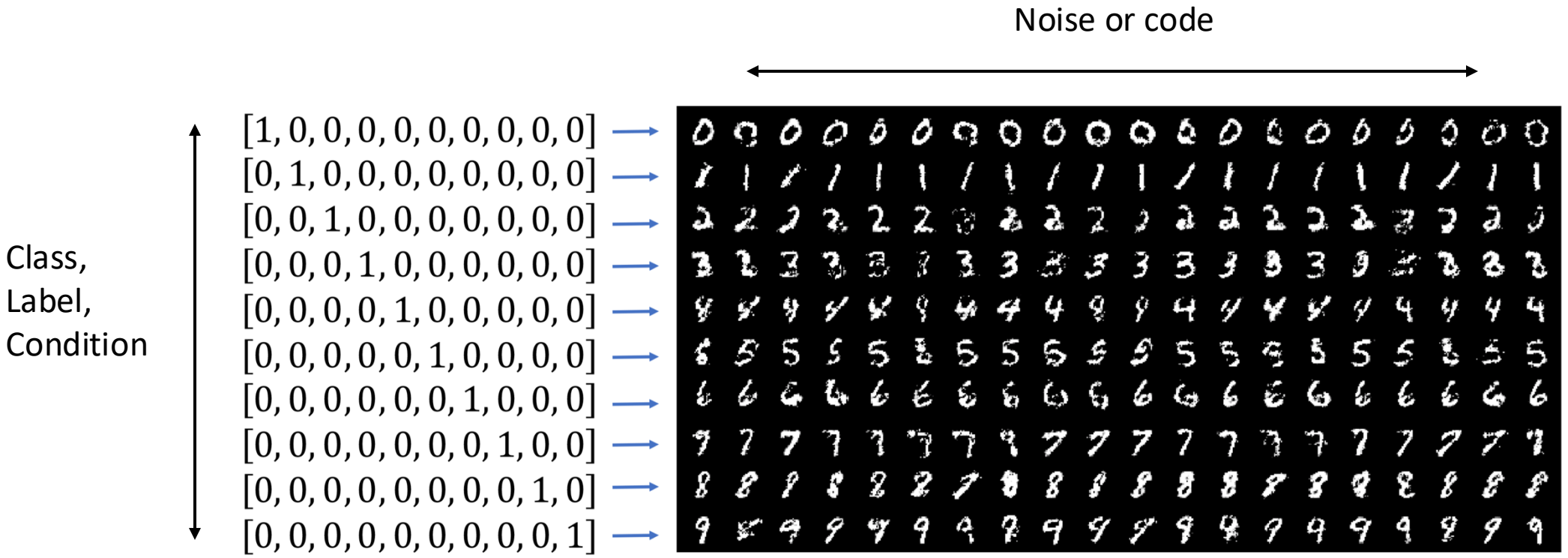
- Many practical applications of GANs when we have explicit supervision available



Conditional GAN
(Mirza & Osindero, 2014)

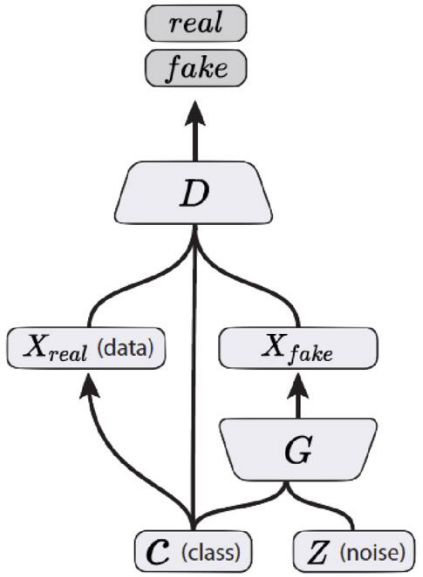
Conditional GAN

- MNIST digits generated conditioned on their class label
- https://drive.google.com/file/d/13sH-PUQavPFJGZR_jC87Pf0zowAQGnzN/view?usp=sharing



Generator

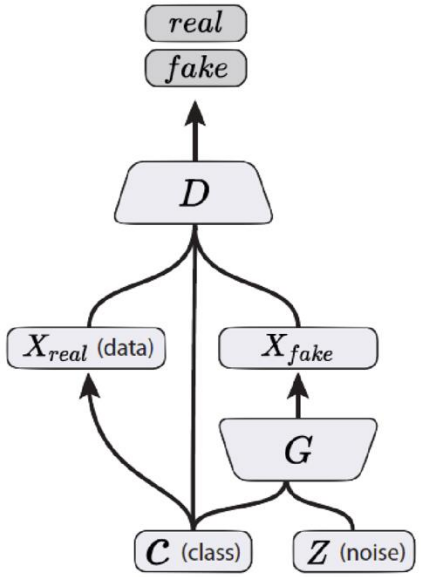
```
generator_model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(units = 256, activation = 'relu', input_dim = 138),
    tf.keras.layers.Dense(units = 784, activation = 'sigmoid')
])

noise = tf.keras.layers.Input(shape = (128,))
label = tf.keras.layers.Input(shape = (10,))

model_input = tf.keras.layers.concatenate([noise, label], axis = 1)
generated_image = generator_model(model_input)

generator = tf.keras.models.Model(inputs = [noise, label], outputs = generated_image)
```



Conditional GAN
(Mirza & Osindero, 2014)

Discriminator

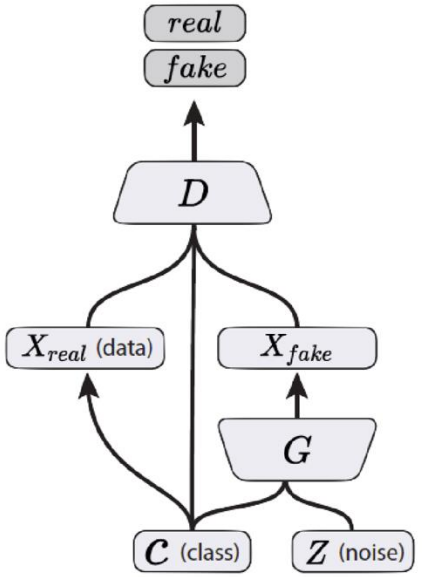
```
discriminator_model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(units = 256, activation = 'relu', input_dim = 794),
    tf.keras.layers.Dense(units = 1, activation = 'sigmoid')
])

input_image = tf.keras.layers.Input(shape = (784,))
label = tf.keras.layers.Input(shape = (10,))

model_input = tf.keras.layers.concatenate([input_image, label], axis = 1)
validity = discriminator_model(model_input)

discriminator = tf.keras.models.Model(inputs = [input_image, label], outputs = validity)
```



Conditional GAN
(Mirza & Osindero, 2014)

Combined

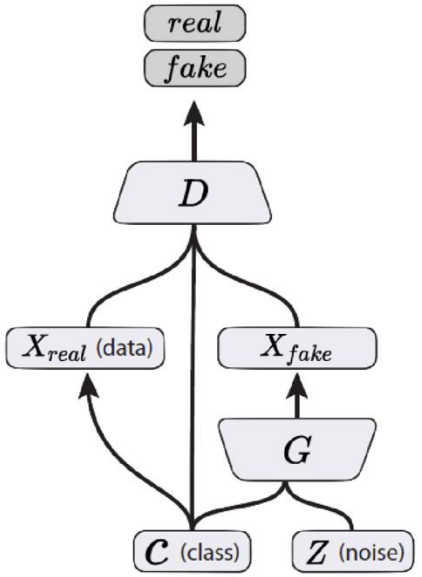
```
discriminator.trainable = False

noise = tf.keras.layers.Input(shape = (128,))
label = tf.keras.layers.Input(shape = (10,))

generated_image = generator([noise, label])
validity = discriminator([generated_image, label])

combined = tf.keras.models.Model(inputs = [noise, label], outputs = validity)
```



Conditional GAN
(Mirza & Osindero, 2014)

CGAN Implementation

```
valid = np.ones(batch_size)
fake = np.zeros(batch_size)

for i in range(n_iter):

    # Train Discriminator
    idx = np.random.randint(0, train_x.shape[0], batch_size)
    real_images, labels = train_x[idx], train_y[idx]

    noise = create_noise(batch_size)
    generated_images = generator.predict([noise, labels], verbose = 0)

    d_loss_real = discriminator.train_on_batch([real_images, labels], valid)
    d_loss_fake = discriminator.train_on_batch([generated_images, labels], fake)
    d_loss = d_loss_real + d_loss_fake

    # Train Generator
    noise = create_noise(batch_size)
    labels = np.random.randint(0, 10, batch_size)
    labels_onehot = np.eye(10)[labels]

    g_loss = combined.train_on_batch([noise, labels_onehot], valid)
```



Conditional GAN
(Mirza & Osindero, 2014)

Fake MNIST Images Generated by CGAN