

# CSE428: Image Processing

## Lecture 13

# Convolutional Neural Networks: Part 1

# Image Classification Problem

Task: Write a computer program to differentiate between **cat** and **dog** images

- Solution 1

Come up with a complicated function of pixels on your own

- Solution 2

Have the computer come up with a complicated function



dog (1)



dog (1)



dog (1)



cat (0)



dog (1)



dog (1)



cat (0)



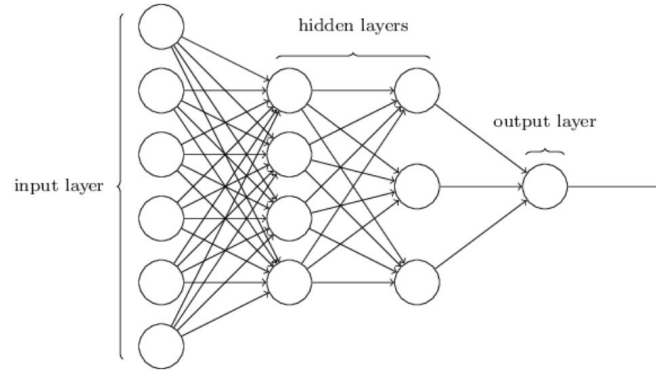
cat (0)



dog (1)

# Feed Forward Neural Network / Multilayer Perceptron

Recall: A FFNN/MLP takes an input vector and predicts its label through a series of matrix multiplication and passing it through some nonlinearities!



Powerful, but not well suited for image related computer vision tasks :(

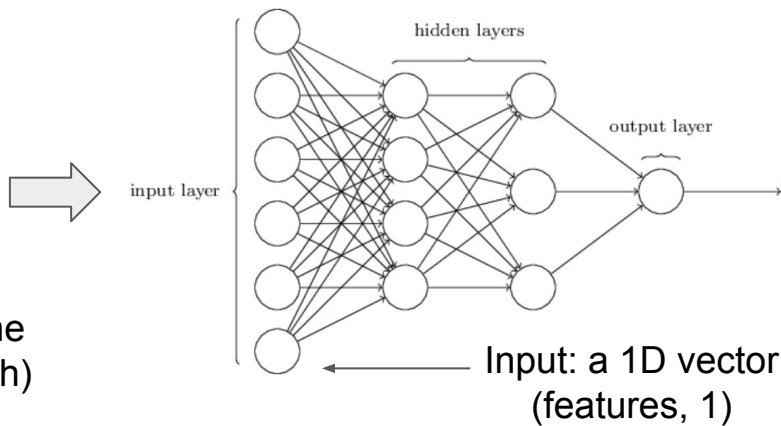
# Why not MLP?

For Computer Vision tasks feed forward fully connected neural networks (or multi layer perceptrons) are not suitable for mainly two reasons:

1. A lack of spatial reasoning
2. An explosive number of parameters



Image: a 3D volume  
(height, width, depth)

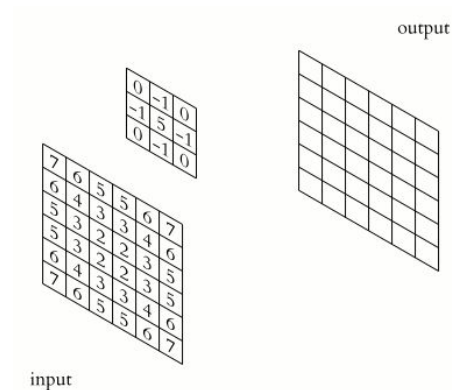


Prediction: 0 (Cat)  
Prediction: 1 (Dog)

# Image Convolutions

Remember the convolution with kernels?

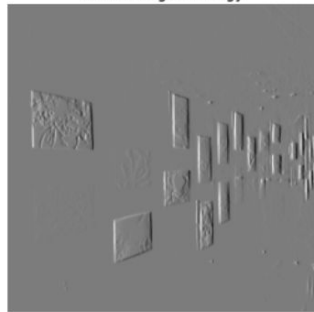
- Sliding kernel approach
- Had predefined kernels
- Works well for lower level vision tasks like edge detection
- Doesn't generalize well for higher level vision tasks like image classification



Original image

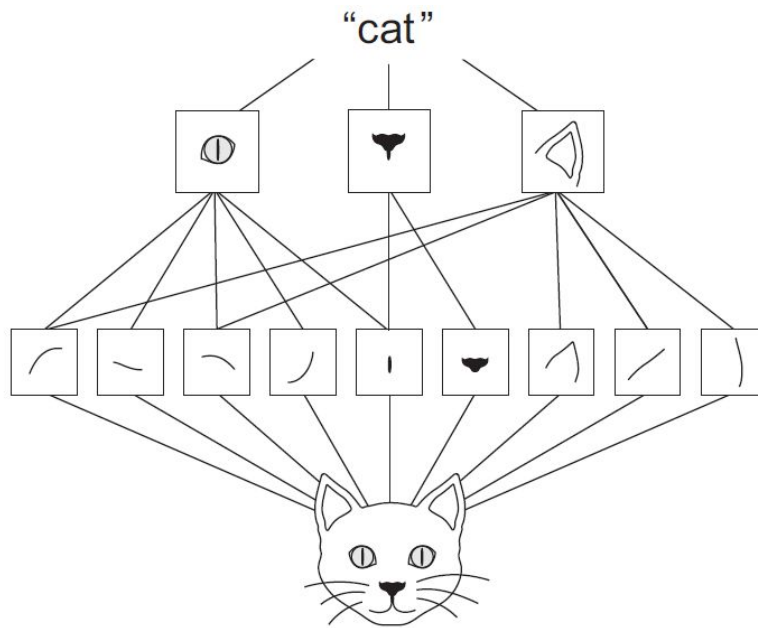


Vertical edges with gy



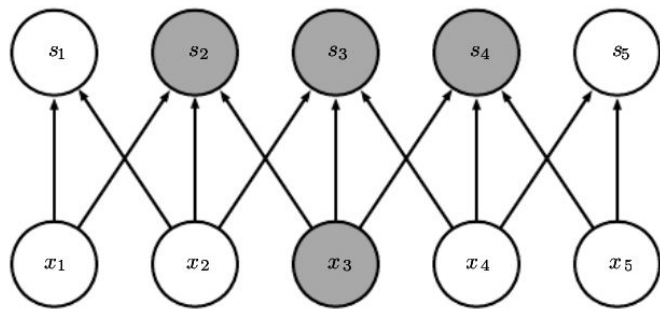
# Image Convolutions

A higher level vision task is much more complicated than just detecting edges!

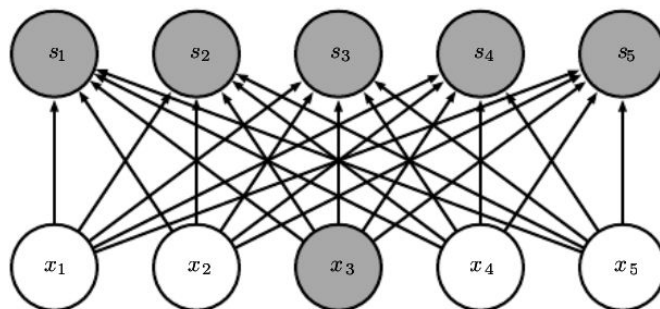


# Image Convolutions

Image convolutions provide sparse connectivity as opposed to matrix multiplication in MLP, where the connection is no longer sparse.



Convolution

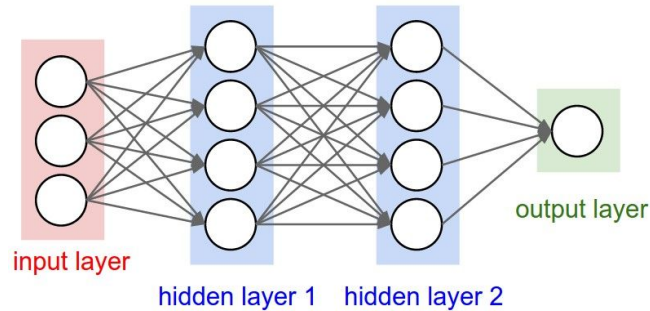


Matrix multiplication



# Image Convolutions

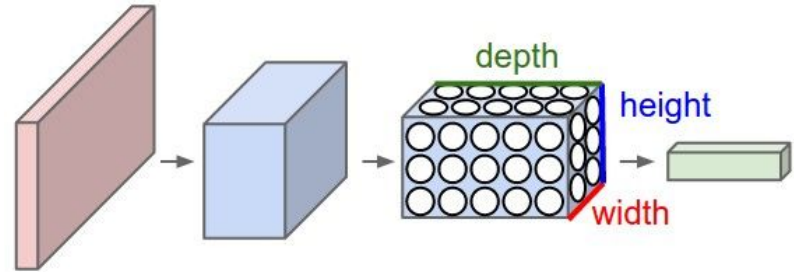
Proposal: have this ***convolution operation*** integrated with the multi layer perceptron or the feedforward neural networks!



**MLP**

**Input** to each layer: 1D Tensor  
“vector”

**Output** of each layer: 1D Tensor  
“vector”



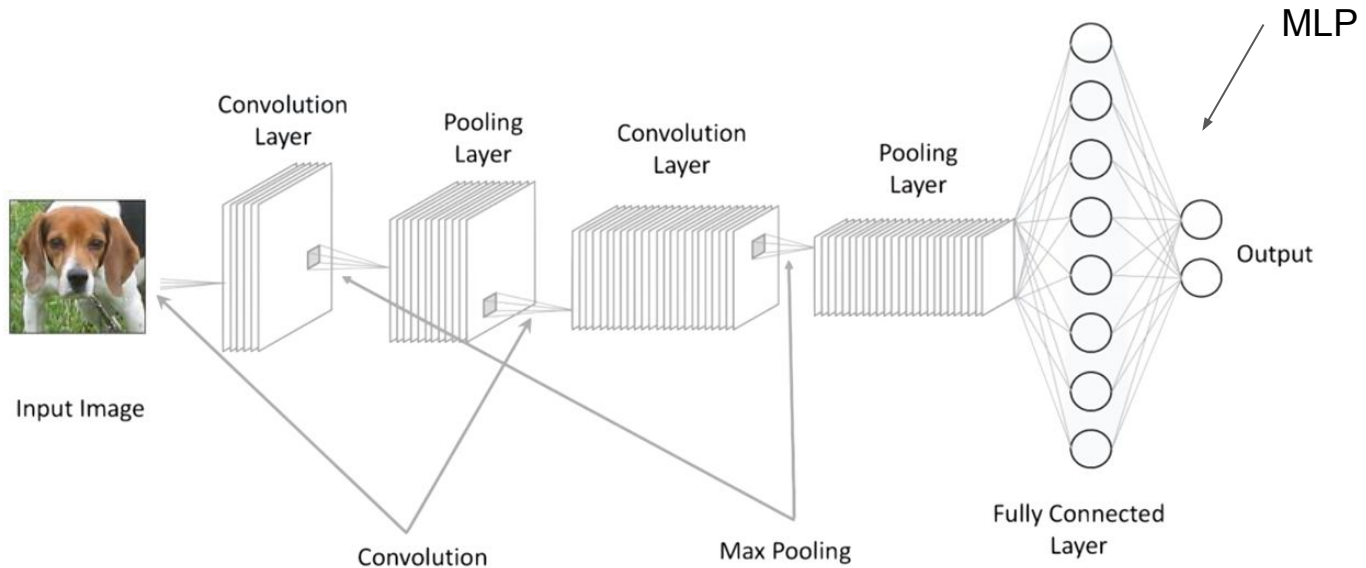
**CNN**

**Input** to each layer: 3D Tensor  
“volume”

**Output** of each layer: 3D Tensor  
“volume”

# Convolutional Neural Networks (CNNs / ConvNets)

Key idea: have this convolution operation integrated with the feedforward neural network by treating the kernel entries as weights!



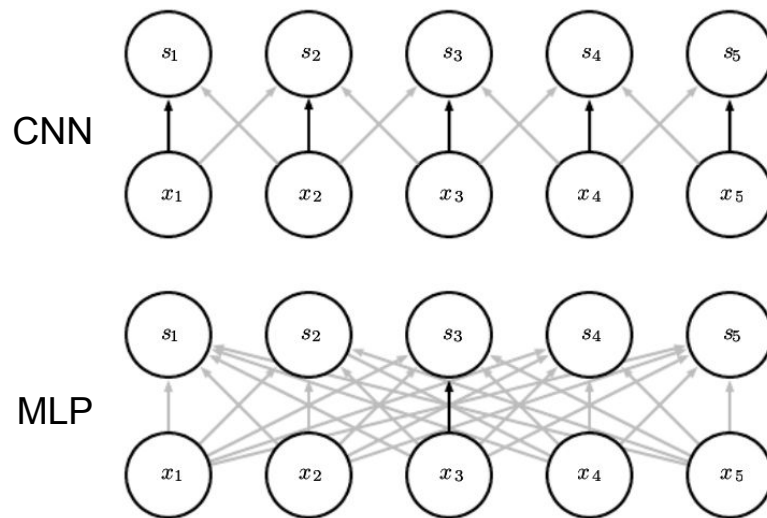
# Advantages of CNNs

CNNs have many advantages for image related vision tasks:

1. Spatial Arrangement
  - a. Focuses on local regions
2. Parameter Sharing
  - a. Very few params per layer!

Visualize CNN on your browser:

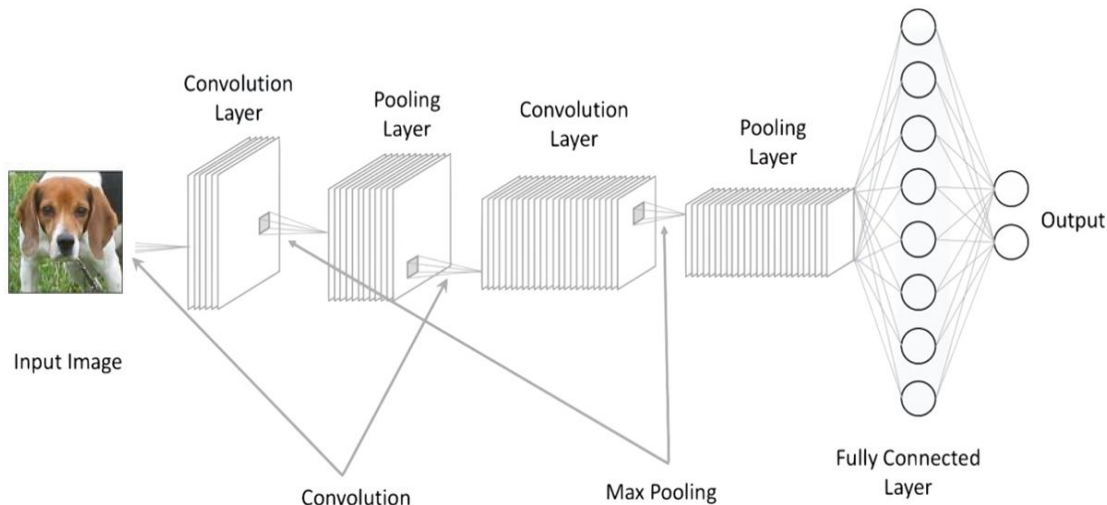
<https://www.cs.ryerson.ca/~aharley/vis/conv/>



# CNN Layers

Typically, a CNN has

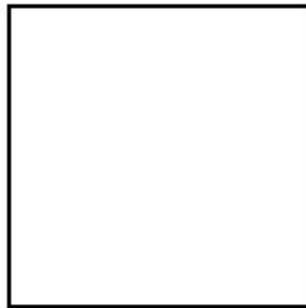
1. Input Layer
2. Convolution Layers
3. Pooling Layers
4. Fully Connected Layer
5. Output Layer



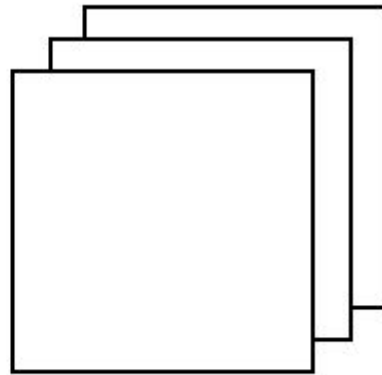
# Input Layer

## The Image!

- Grayscale Image
  - Shape: (H x W x 1)
- RGB Image
  - Shape: (H x W x 3)



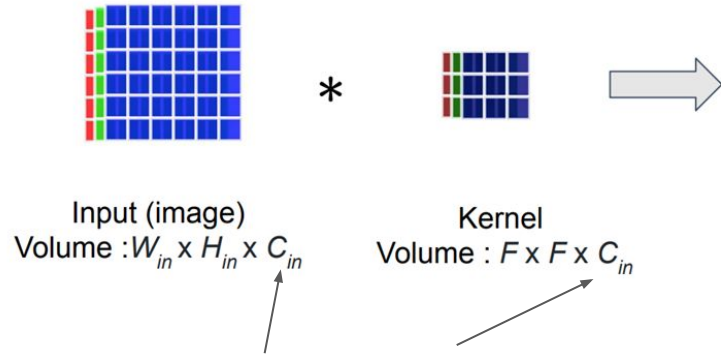
2D image



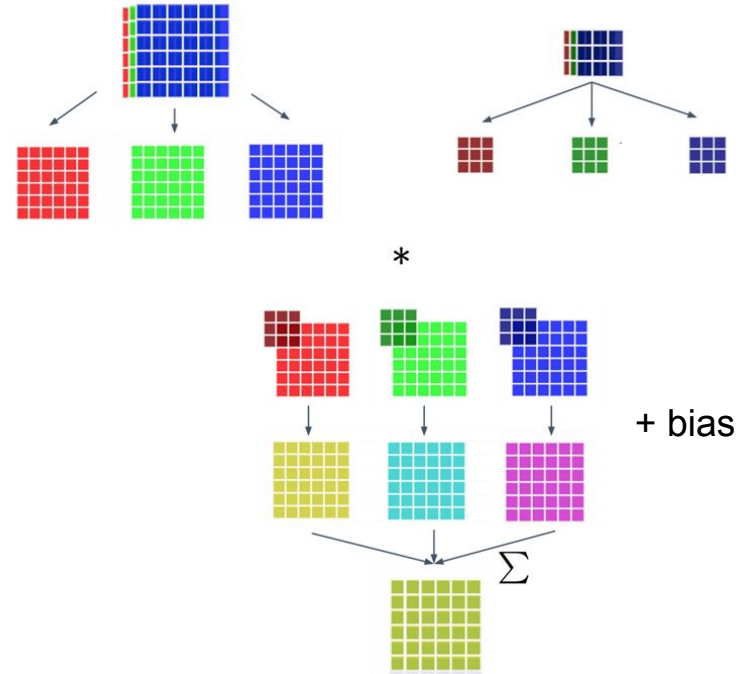
3D image

# Concept: 3D Convolution (Convolution over Volume)

Convolution operation with 1 kernel of size  $F$

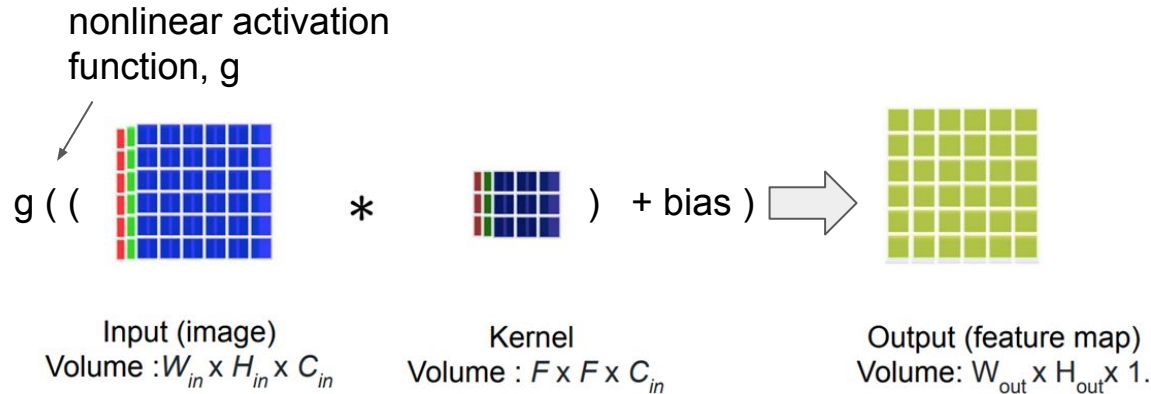


Note: here,  $C_{in(Image)} = C_{in(Kernel)}$  i.e. kernels always extend the full depth of the input volume,



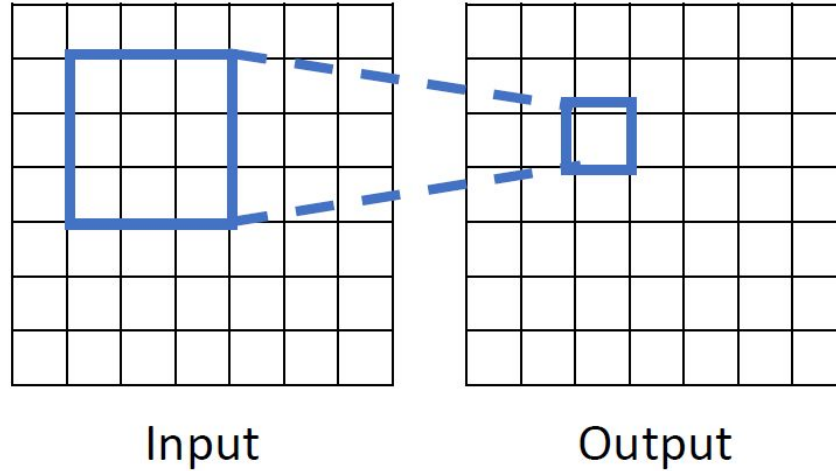
# Concept: 3D Convolution (Convolution over Volume)

3D Convolution with a single kernel produces one 2D “feature map”



# Concept: Receptive Field

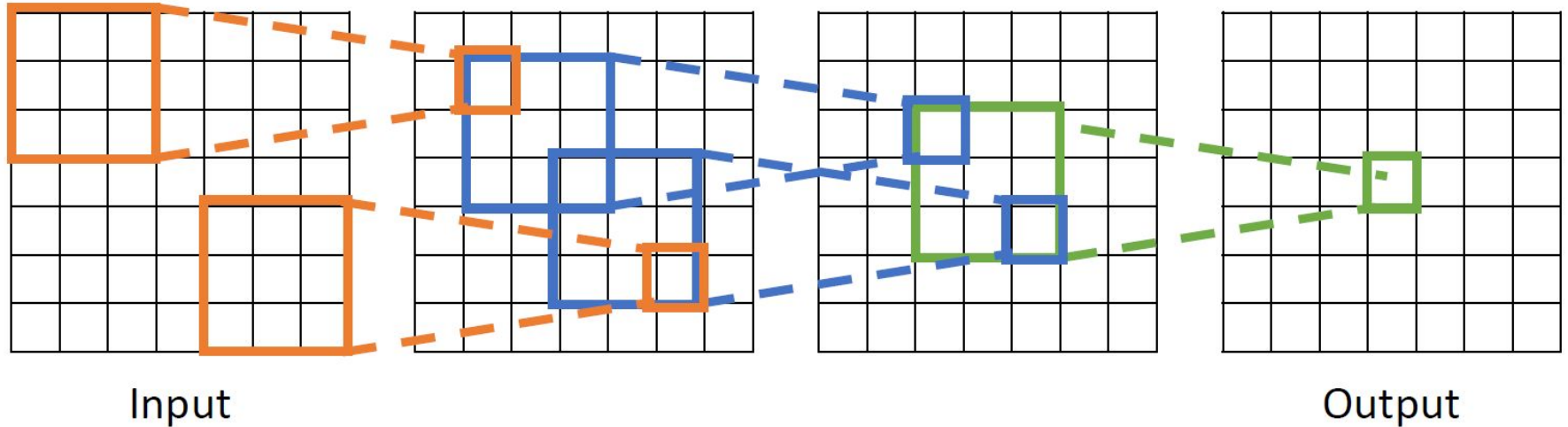
Each element in the output is the result of a  $F \times F$  “receptive field” of the input





# Concept: Receptive Field

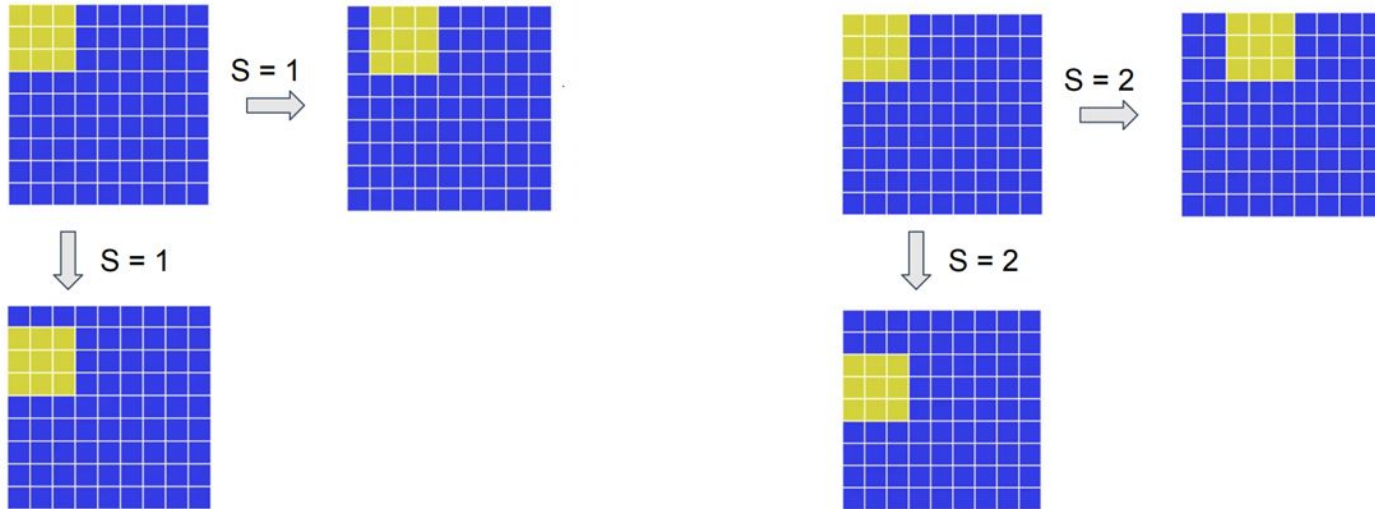
Successive convolution layer increases the apparent “receptive field” of the output



# Concept: Stride

Stride is the number of pixels the kernel moves at each step, denoted by “S”

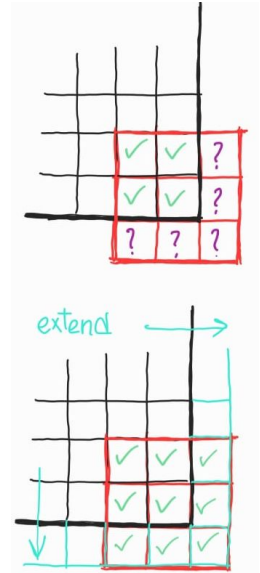
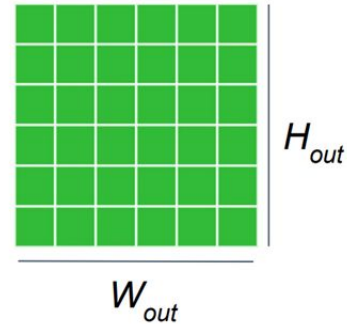
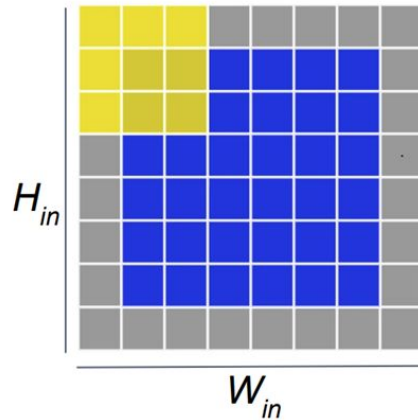
Higher value of stride increases the receptive field much faster



# Concept: Padding

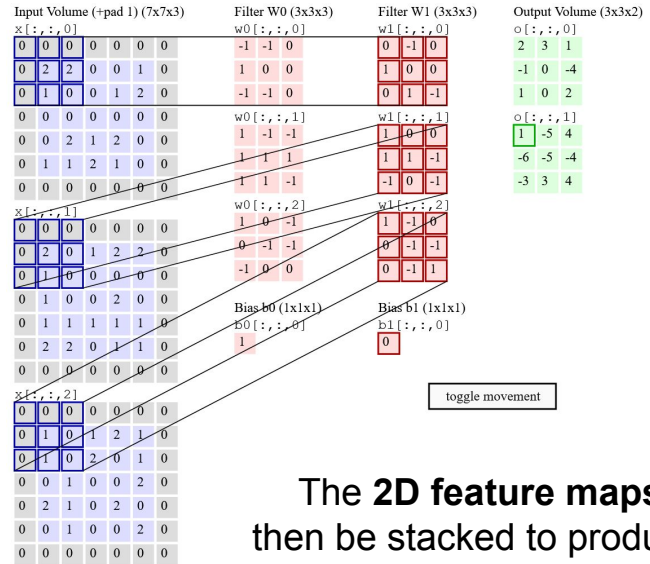
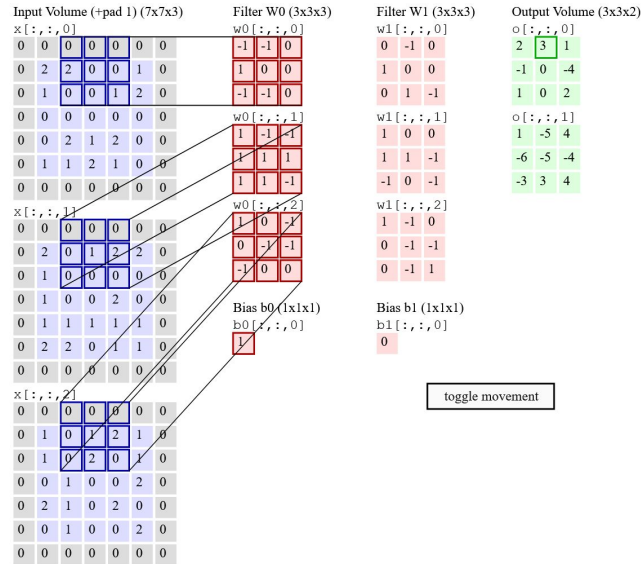
Padding: extend the borders of the original image by “P” pixels on each side:

1. To keep the output image the same size as the original image
2. To achieve any arbitrary output shape



# Concept: 3D Convolution with multiple Kernels

3D convolution with multiple kernels produces multiple 2D “feature maps” (/kernel)



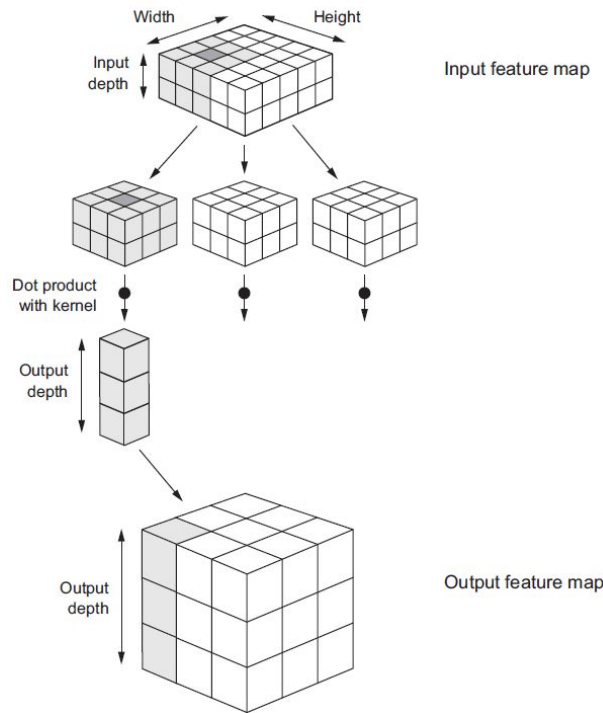
The **2D feature maps** can then be stacked to produce a **3D volume of feature maps**

# Convolution Layer

A convolution layer or conv layer in CNN takes a 3D volume as an input and produces another 3D volume output of feature maps

By performing convolution over the input volume using some kernels which extend the full depth of the input volume & passing it through a nonlinearity

- Input volume:  $H_{in} \times W_{in} \times C_{in}$
- Output volume:  $H_{out} \times W_{out} \times C_{out}$

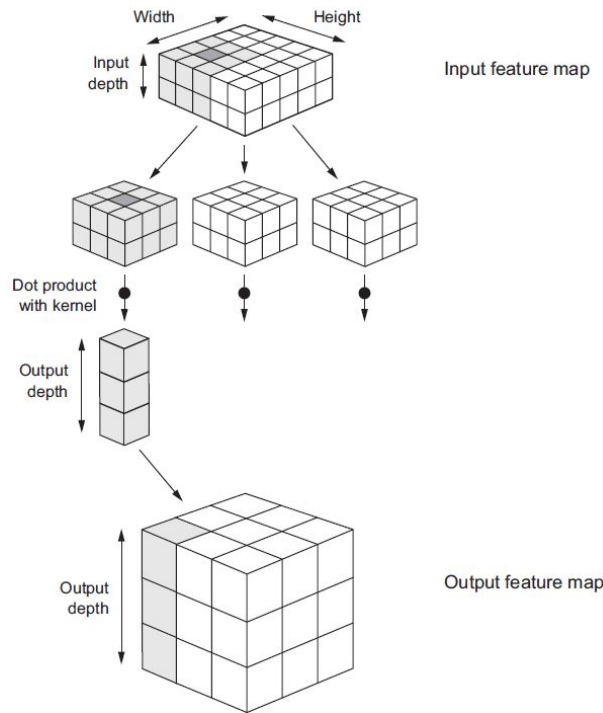


# Convolution Layer

In general for a convolution layer

- Input volume:  $H_{in} \times W_{in} \times C_{in}$
- Kernel volume (of size  $F$ ):  $F \times F \times C_{in}$
- Number of kernels:  $C_{out}$
- Padding:  $P$
- Stride:  $S$
- Then output volume:  $H_{out} \times W_{out} \times C_{out}$

$$H_{out} = \frac{H_{in} - F + 2P}{S} + 1 ; W_{out} = \frac{W_{in} - F + 2P}{S} + 1$$



# tf.keras.layers.Conv2D

```
tf.keras.layers.Conv2D(  
    filters, kernel_size, strides=(1, 1), padding='valid',  
    data_format=None, dilation_rate=(1, 1), groups=1, activation=None,  
    use_bias=True, kernel_initializer='glorot_uniform',  
    bias_initializer='zeros', kernel_regularizer=None,  
    bias_regularizer=None, activity_regularizer=None, kernel_constraint=None,  
    bias_constraint=None, **kwargs  
)
```

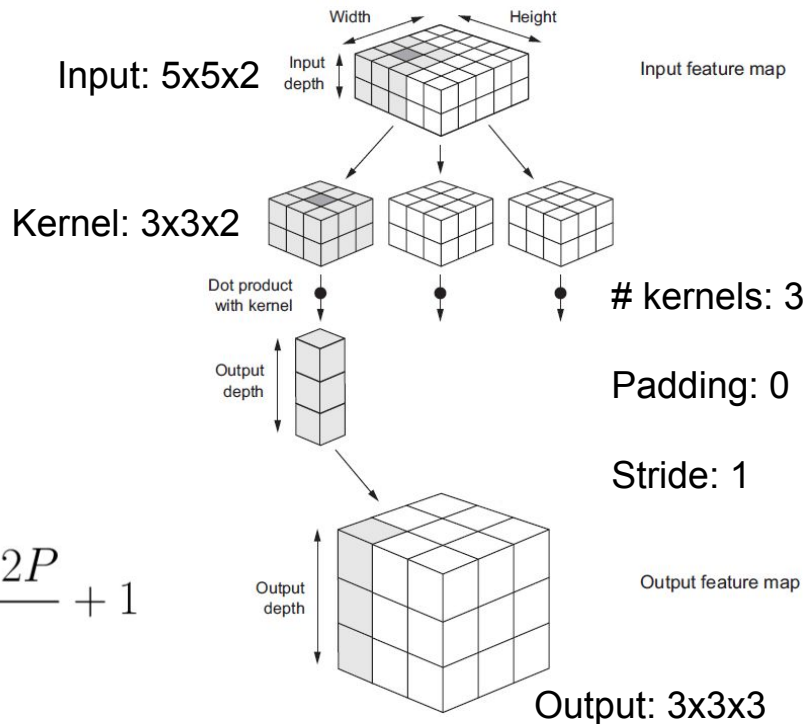


# Convolution Layer Example

## Example 1

- Input volume:  $H_{in} \times W_{in} \times C_{in}$
- Kernel volume (of size  $F$ ):  $F \times F \times C_{in}$
- Number of kernels:  $C_{out}$
- Padding:  $P$
- Stride:  $S$
- Then output volume:  $H_{out} \times W_{out} \times C_{out}$

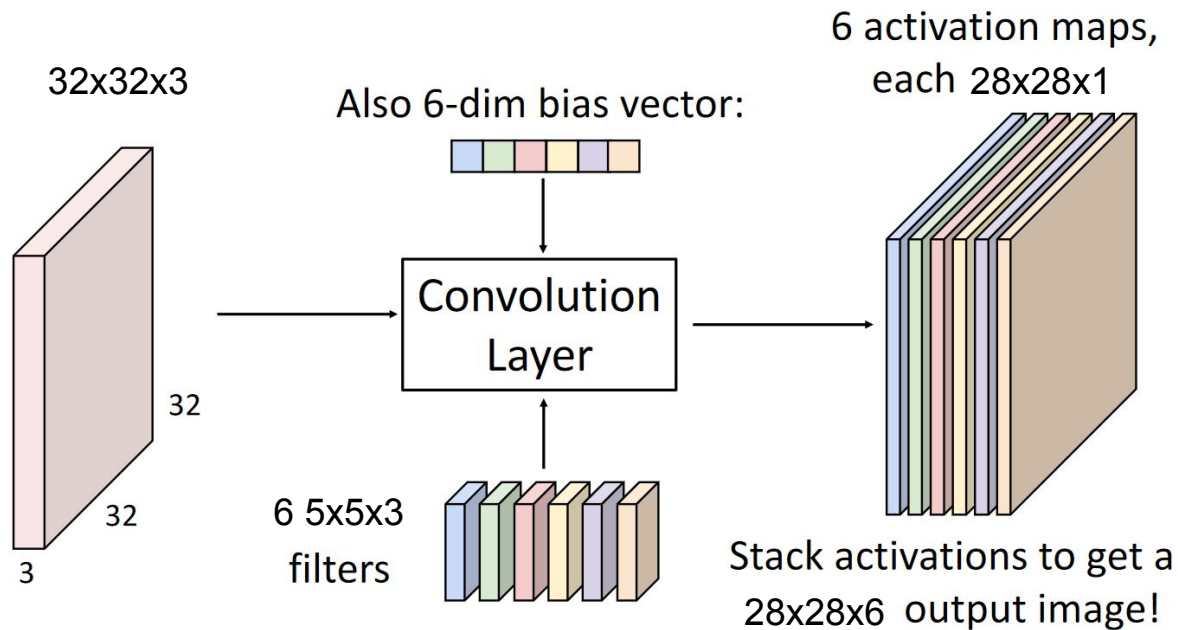
$$H_{out} = \frac{H_{in} - F + 2P}{S} + 1 ; W_{out} = \frac{W_{in} - F + 2P}{S} + 1$$





# Convolution Layer Example

## Example 2



# Convolution Layer Parameters

Hyperparameters:

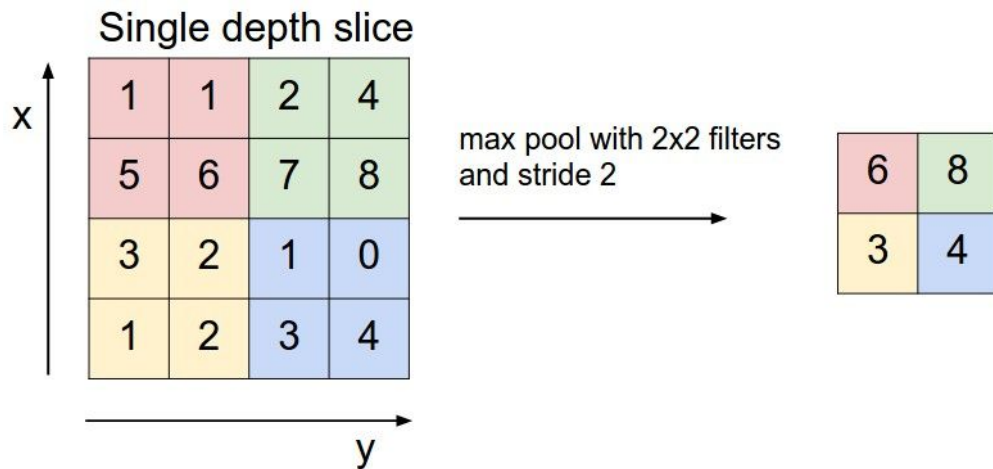
- Kernel size:  $F$
- Number of kernels:  $C_{out}$
- Padding:  $P$
- Stride:  $S$

Learnable parameters:

- Kernel weights:  $C_{out} \times F \times F \times C_{in}$
- Kernel biases:  $C_{out}$

# Pooling Layer

- Pooling layers are inserted between successive Conv layers
- Progressively reduce the spatial size while keeping the depth constant
- Operates independently on each channel or “depth” over some  $F \times F$  region

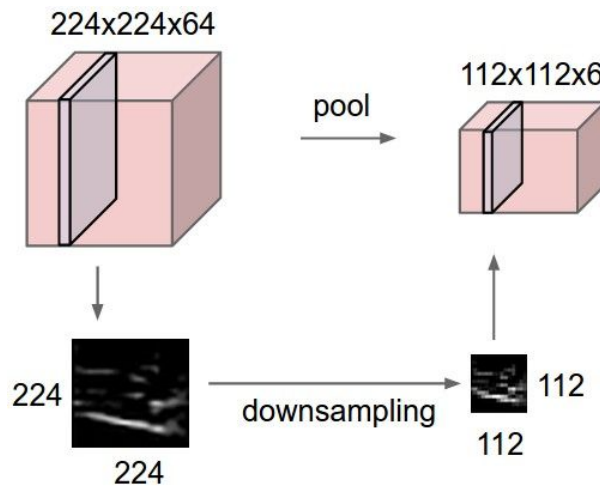


# Pooling Layer

In general for a pooling layer

- Input volume:  $H_{in} \times W_{in} \times C_{in}$
- Kernel size:  $F$
- Stride:  $S$
- Pooling function: MAX, Avg
- Then output volume:  $H_{out} \times W_{out} \times C_{in}$

$$H_{out} = \frac{H_{in} - F}{S} + 1 ; W_{out} = \frac{W_{in} - F}{S} + 1$$



# tf.keras.layers.MaxPooling2D

```
tf.keras.layers.MaxPool2D(  
    pool_size=(2, 2), strides=None, padding='valid', data_format=None,  
    **kwargs  
)
```



# Pooling Layer

Hyperparameters:

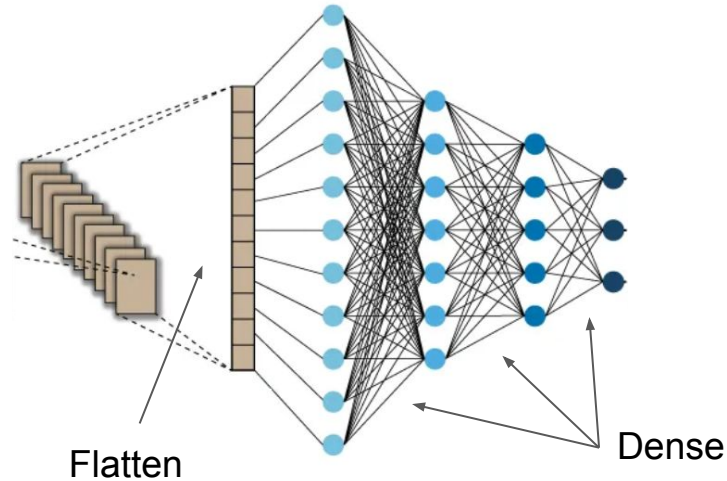
- Kernel size: ***F***
- Stride: ***S***
- Pooling function: ***MAX, Average***

Learnable parameters:

- Zero!

# Fully Connected Layer

A fully connected layer in CNN first flattens or unravels a 3D volume into a 1D vector and does similar computations like an MLP



# tf.keras.layers.Flatten

```
tf.keras.layers.Flatten(  
    data_format=None, **kwargs  
)
```





# tf.keras.layers.Dense

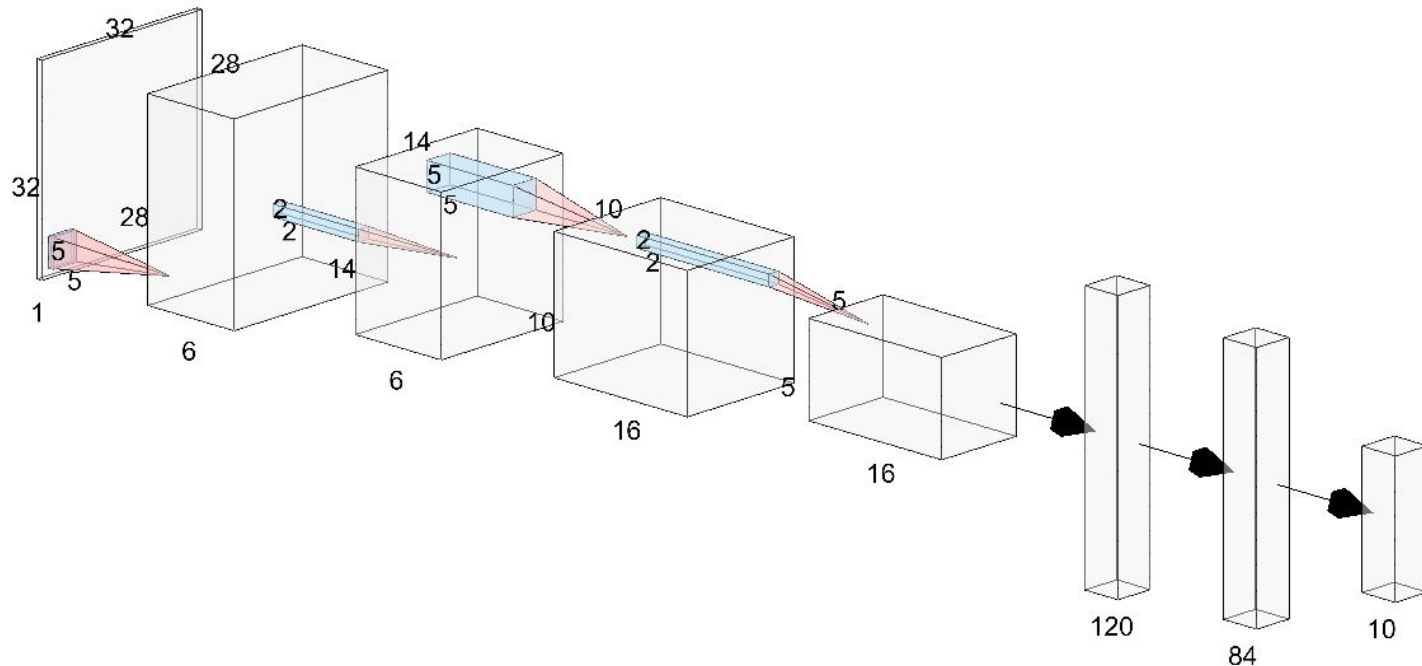
```
tf.keras.layers.Dense(  
    units, activation=None, use_bias=True,  
    kernel_initializer='glorot_uniform',  
    bias_initializer='zeros', kernel_regularizer=None,  
    bias_regularizer=None, activity_regularizer=None, kernel_constraint=None,  
    bias_constraint=None, **kwargs  
)
```



# The LeNet-5 CNN Architecture

Layers:

1. Input
2. Conv-1
3. Pool-1
4. Conv-2
5. Pool-2
6. FC-1
7. FC-2
8. Output



# Resources

1. <https://cs231n.github.io/convolutional-networks/>
2. <https://web.eecs.umich.edu/~justincj/teaching/eecs498/FA2020/>
3. [https://www.tensorflow.org/api\\_docs/python/tf/keras](https://www.tensorflow.org/api_docs/python/tf/keras)
4. Deep Learning with Python Book by François Chollet
5. <https://www.deeplearningbook.org/>
6. Hands-on Computer Vision with TensorFlow 2 by Eliot Andres & Benjamin Planche (Packt Pub.)

# Colab Notebooks

## 1. NN Demo

[https://colab.research.google.com/drive/1Zrb2f\\_xNSfZ1QXASa9vnbFFub56J9opl?usp=sharing](https://colab.research.google.com/drive/1Zrb2f_xNSfZ1QXASa9vnbFFub56J9opl?usp=sharing)

## 1. CNN Demo

<https://colab.research.google.com/drive/1w0ELsNaI47SszoBKUGa736V7JWF eYIzJ?usp=sharing>