
# Workshop 6

COMP90051 Statistical Machine Learning

Semester 1, 2021

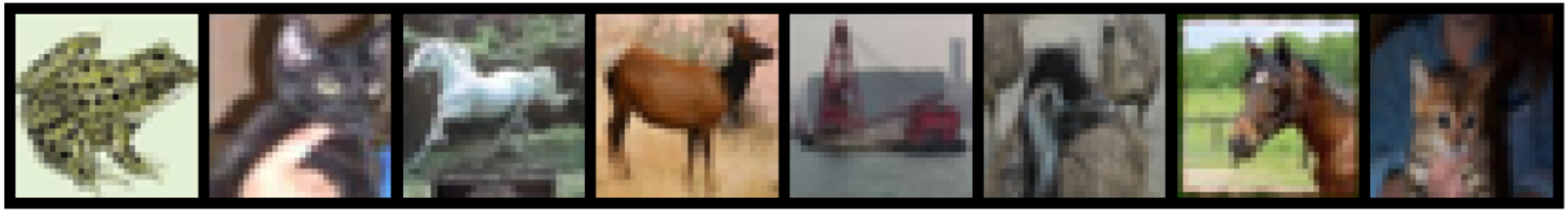
# Learning Outcomes

At the end of this workshop you should:

1. Be familiar with Convolutional Neural Network and be able to implement it.
2. Be able to implement autoencoder

# Image classification task

- CIFAR 10 - Image classification
- 10 labels
  - Multi-class
  - classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck')



# How is data represented in neural nets?

- Primary data structure is the tensor—a fancy name for a multi-dimensional array
- Can be used to represent trainable weights, hyperparameters, data flowing through the network, etc.
- E.g. an image can be represented as a 3d-tensor: 2 dimensions for horizontal/vertical pixels + 1 dimension for the RGB channels

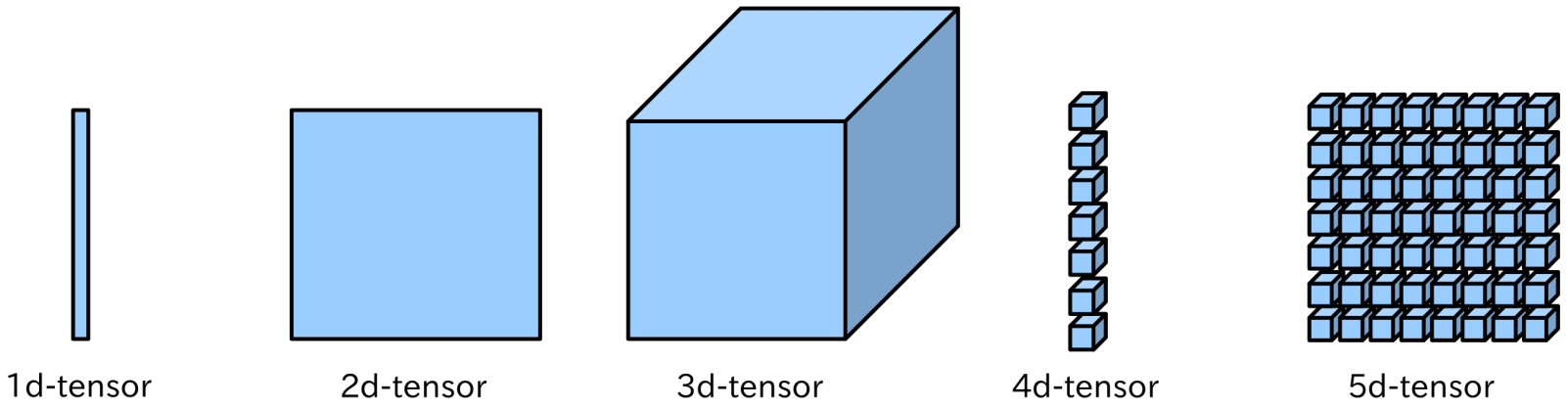
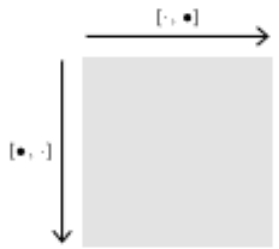


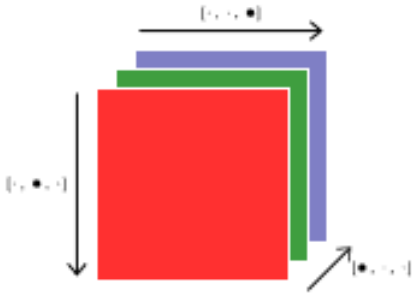

Image source: Tatsuya Yokota (2012). *Tensor Analysis for EEG data.*

# Image Tensor

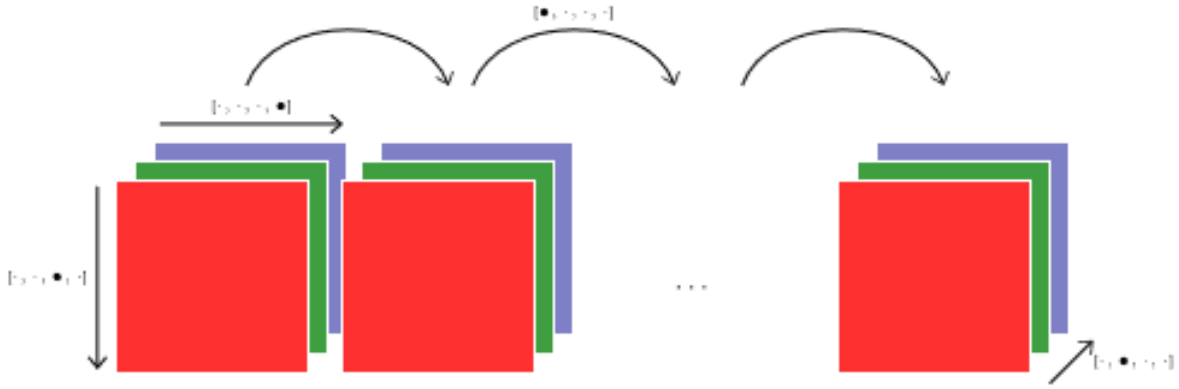
2d tensor (*e.g.* grayscale image)



3d tensor (*e.g.* rgb image)



4d tensor (*e.g.* sequence of rgb images)



François Fleuret - Autumn 2019 Lecture slides.

# Feature for logistic regression

```
tensor([[0.3647, 0.3608, 0.3647,  ..., 0.3804, 0.3882, 0.3843],
        [0.3725, 0.3608, 0.3608,  ..., 0.4000, 0.3882, 0.3529],
        [0.4000, 0.3843, 0.3765,  ..., 0.4275, 0.4039, 0.3608],
        ...,
        [0.7961, 0.7882, 0.8000,  ..., 0.8157, 0.8314, 0.8235],
        [0.8000, 0.7922, 0.7882,  ..., 0.8431, 0.8314, 0.7804],
        [0.7804, 0.7765, 0.7765,  ..., 0.7961, 0.7725, 0.7451]])
```

```
tensor([[0.4431, 0.4392, 0.4392,  ..., 0.4784, 0.4627, 0.4431],
        [0.4588, 0.4588, 0.4627,  ..., 0.4980, 0.4824, 0.4392],
        [0.4784, 0.4863, 0.4863,  ..., 0.5137, 0.5020, 0.4549],
        ...,
        [0.7725, 0.7647, 0.7765,  ..., 0.7882, 0.8039, 0.7922],
        [0.7765, 0.7686, 0.7647,  ..., 0.8157, 0.8039, 0.7529],
        [0.7569, 0.7529, 0.7529,  ..., 0.7686, 0.7451, 0.7176]])
```

```
tensor([[0.7176, 0.7098, 0.7059,  ..., 0.7294, 0.7255, 0.7020],
        [0.7412, 0.7098, 0.7059,  ..., 0.7608, 0.7255, 0.6902],
        [0.6980, 0.6863, 0.7216,  ..., 0.7176, 0.6824, 0.6745],
        ...,
        [0.7059, 0.7020, 0.7098,  ..., 0.7255, 0.7412, 0.7294],
        [0.7137, 0.7059, 0.7020,  ..., 0.7529, 0.7412, 0.6902],
        [0.6941, 0.6902, 0.6902,  ..., 0.7059, 0.6824, 0.6549]])
```

32*32 matrix

32*32 matrix

32*32 matrix

3*32*32 tensor

flatten

```
tensor([[0.3647, 0.3608, 0.3647,  ..., 0.7059, 0.6824, 0.6549]])
```

1*3072 vector

# Feature for CNN

```
tensor([[[0.3647, 0.3608, 0.3647,  ..., 0.3804, 0.3882, 0.3843],
         [0.3725, 0.3608, 0.3608,  ..., 0.4000, 0.3882, 0.3529],
         [0.4000, 0.3843, 0.3765,  ..., 0.4275, 0.4039, 0.3608],
         ...,
         [0.7961, 0.7882, 0.8000,  ..., 0.8157, 0.8314, 0.8235],
         [0.8000, 0.7922, 0.7882,  ..., 0.8431, 0.8314, 0.7804],
         [0.7804, 0.7765, 0.7765,  ..., 0.7961, 0.7725, 0.7451]])
```

32*32 matrix

```
tensor([[[0.4431, 0.4392, 0.4392,  ..., 0.4784, 0.4627, 0.4431],
         [0.4588, 0.4588, 0.4627,  ..., 0.4980, 0.4824, 0.4392],
         [0.4784, 0.4863, 0.4863,  ..., 0.5137, 0.5020, 0.4549],
         ...,
         [0.7725, 0.7647, 0.7765,  ..., 0.7882, 0.8039, 0.7922],
         [0.7765, 0.7686, 0.7647,  ..., 0.8157, 0.8039, 0.7529],
         [0.7569, 0.7529, 0.7529,  ..., 0.7686, 0.7451, 0.7176]])
```

32*32 matrix

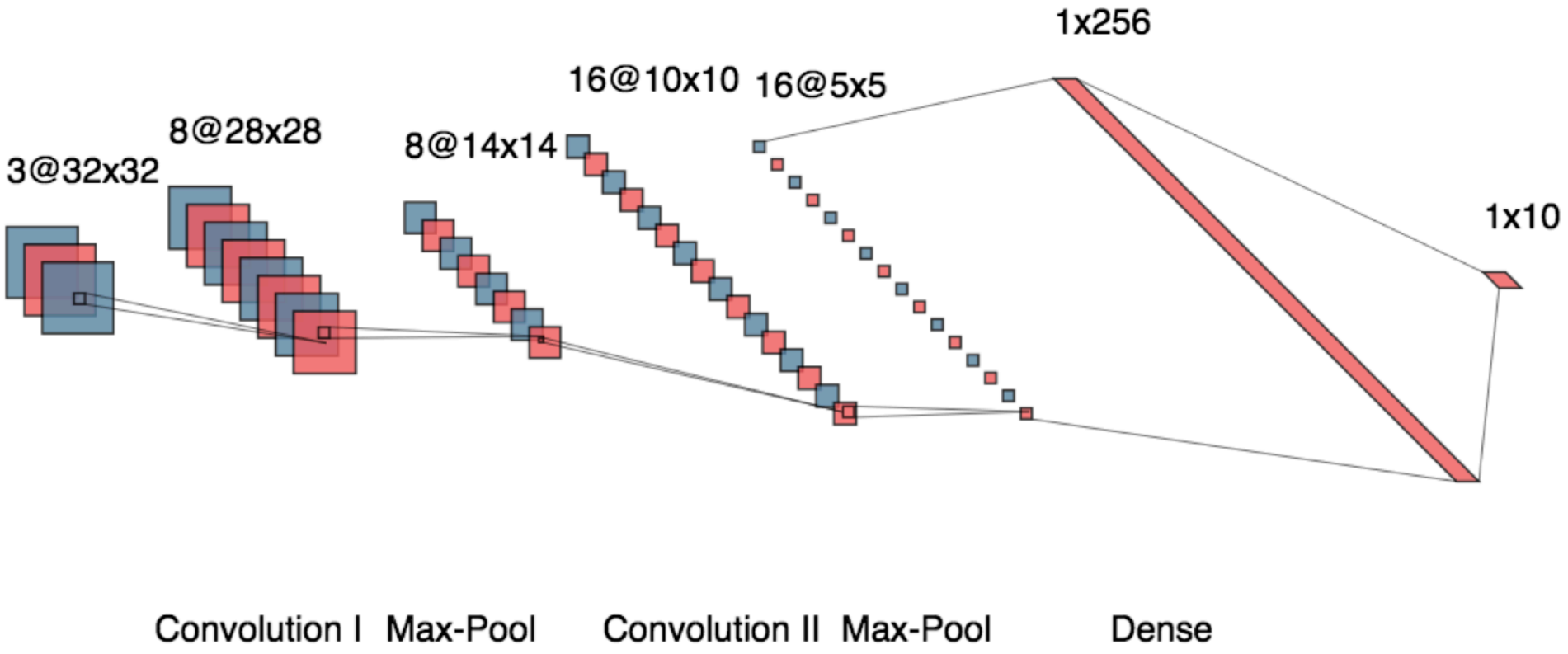
```
tensor([[[0.7176, 0.7098, 0.7059,  ..., 0.7294, 0.7255, 0.7020],
         [0.7412, 0.7098, 0.7059,  ..., 0.7608, 0.7255, 0.6902],
         [0.6980, 0.6863, 0.7216,  ..., 0.7176, 0.6824, 0.6745],
         ...,
         [0.7059, 0.7020, 0.7098,  ..., 0.7255, 0.7412, 0.7294],
         [0.7137, 0.7059, 0.7020,  ..., 0.7529, 0.7412, 0.6902],
         [0.6941, 0.6902, 0.6902,  ..., 0.7059, 0.6824, 0.6549]])
```

32*32 matrix

3*32*32 tensor

Then?

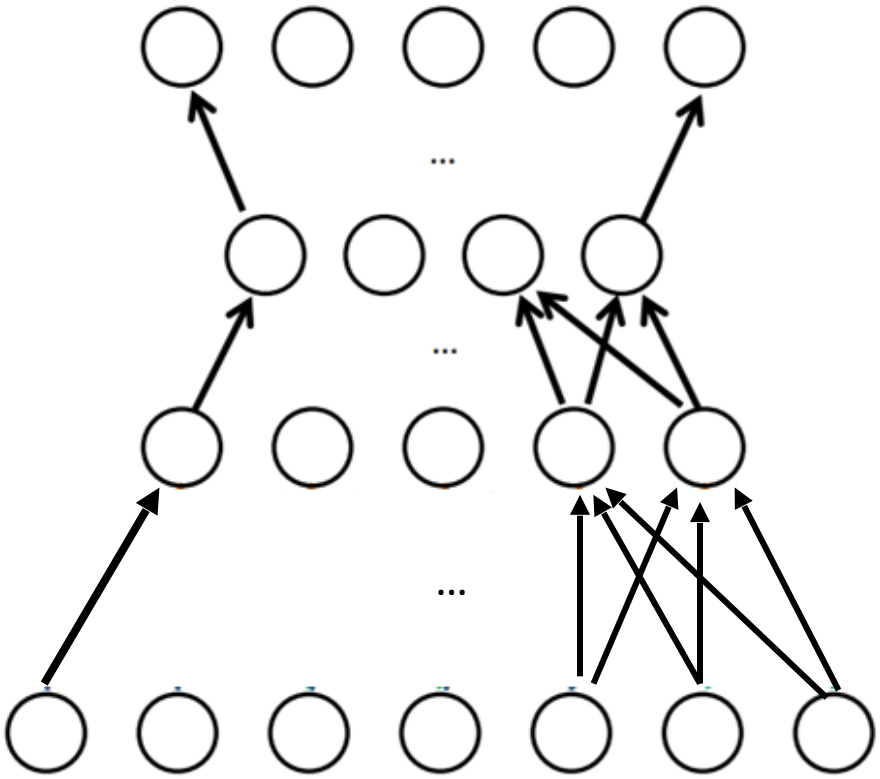
# Architecture of our model



# Components of a CNN

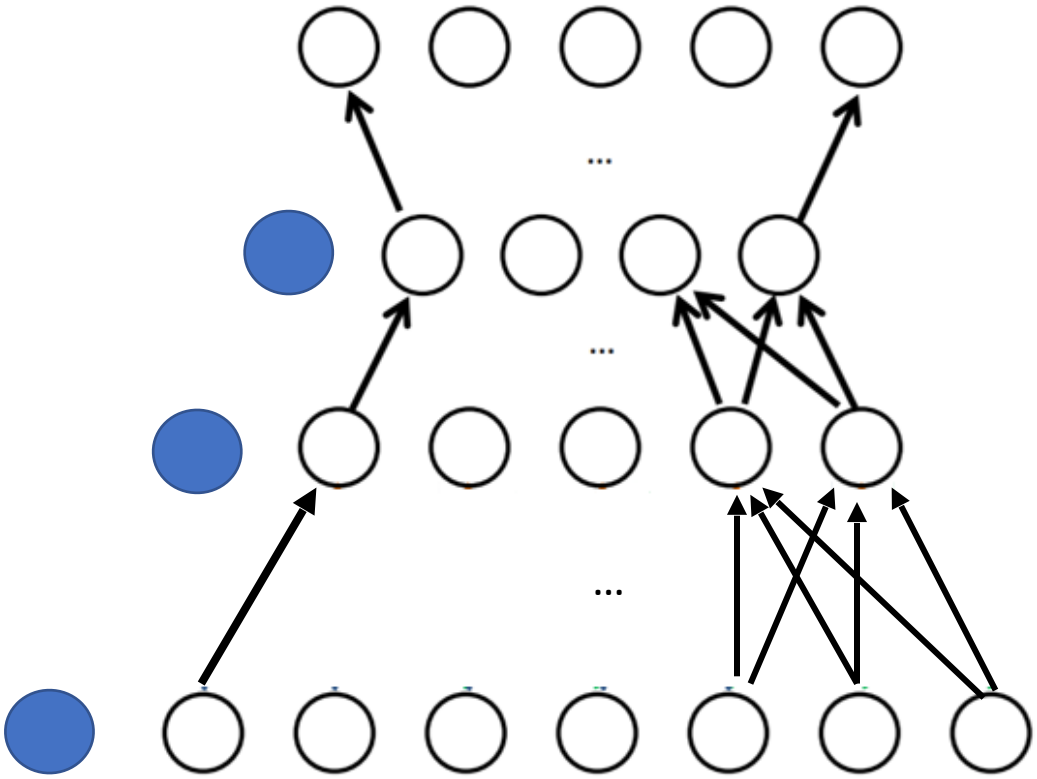
- Convolutional layers
  - Complex input representations based on convolution operation
  - Filter weights are learned from training data
- Downsampling, usually via Max Pooling
  - Re-scales to smaller resolution, limits parameter explosion
- Fully connected parts and output layer
  - Merges representations together

# Number of Parameters



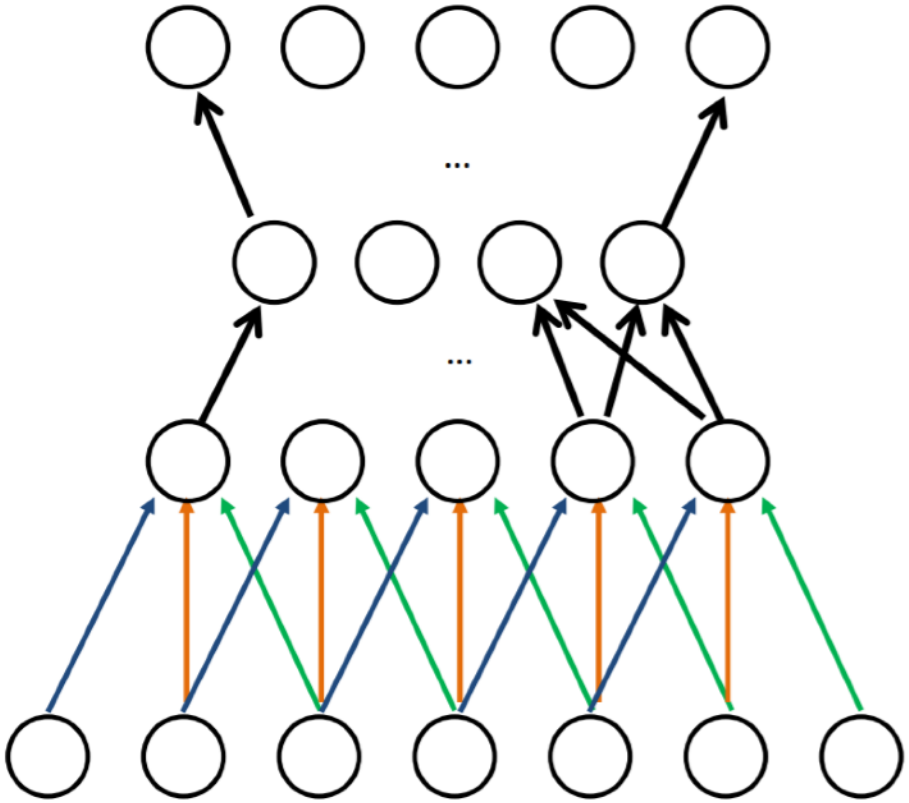
- ANN
- How many weights?

# Number of Parameters



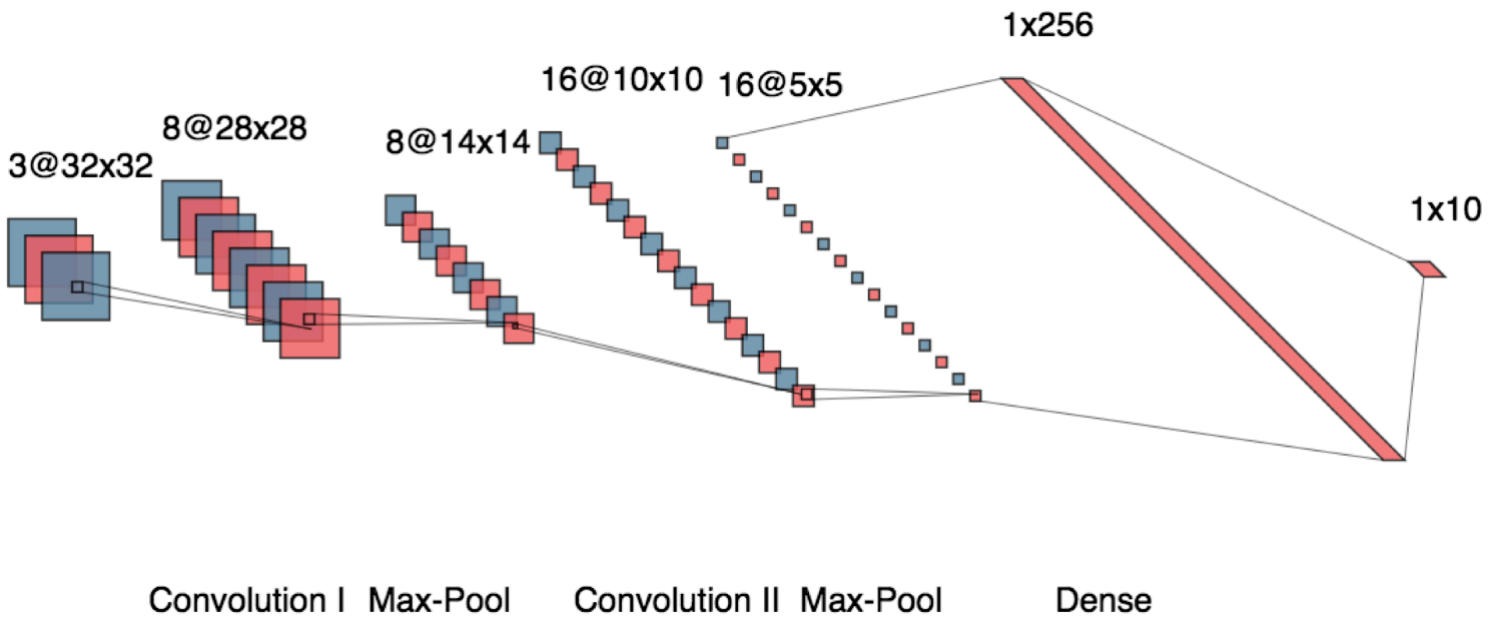
- ANN
- Consider bias
- How many weights?

# Number of Parameters



- CNN
- 1D Kernel size = 3
- How many weights?

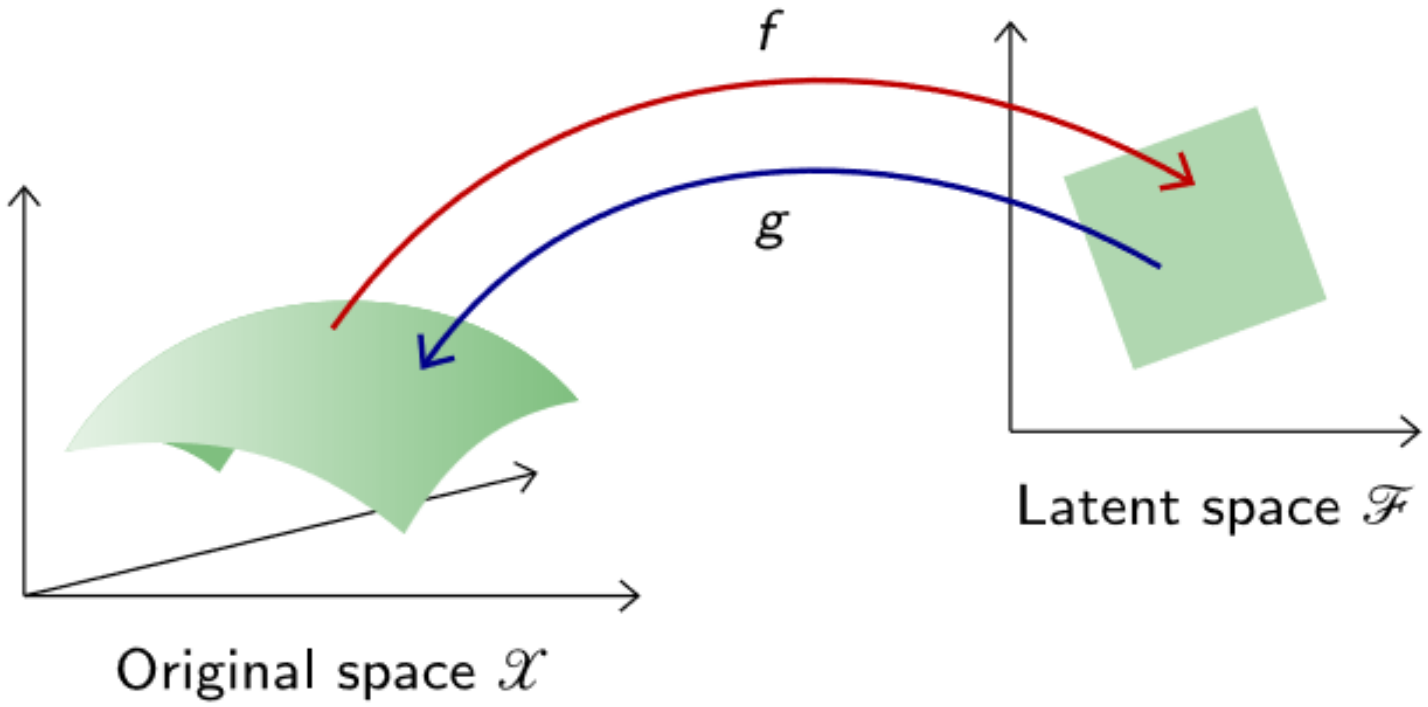
# Number of Parameters




**Question (Extension):** Calculate the number of parameters in the logistic regression model and the above convnet.
The diagram above may be a useful guide. You may also want to look at the model.parameters() method for each model.

# CNN vs Logistic Regression

# Autoencoder




François Fleuret - Autumn 2019 Lecture slides.

# Autoencoder

- Learn encoder $f : \mathscr{X} \to \mathscr{F}$ from original to latent space.
- Learn decoder $g : \mathscr{F} \to \mathscr{X}$ from latent to original space.
- Unsupervised mapping. $g \circ f$ should be close to identity. Minimize quadratic loss over data:

$$\min_{f,g} \sum_{k} \|x_k - g \circ f(x_k)\|^2 \tag{1}$$