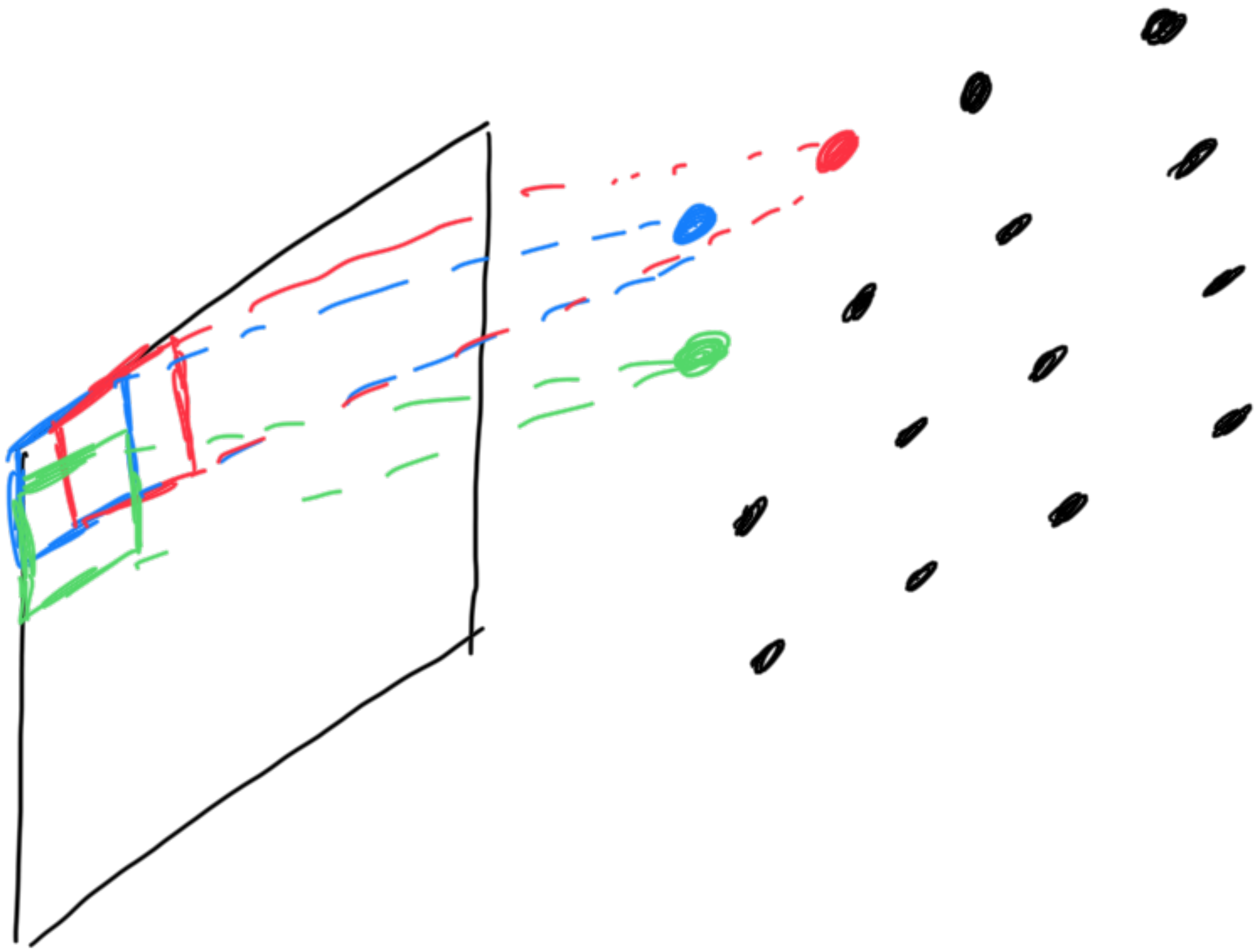
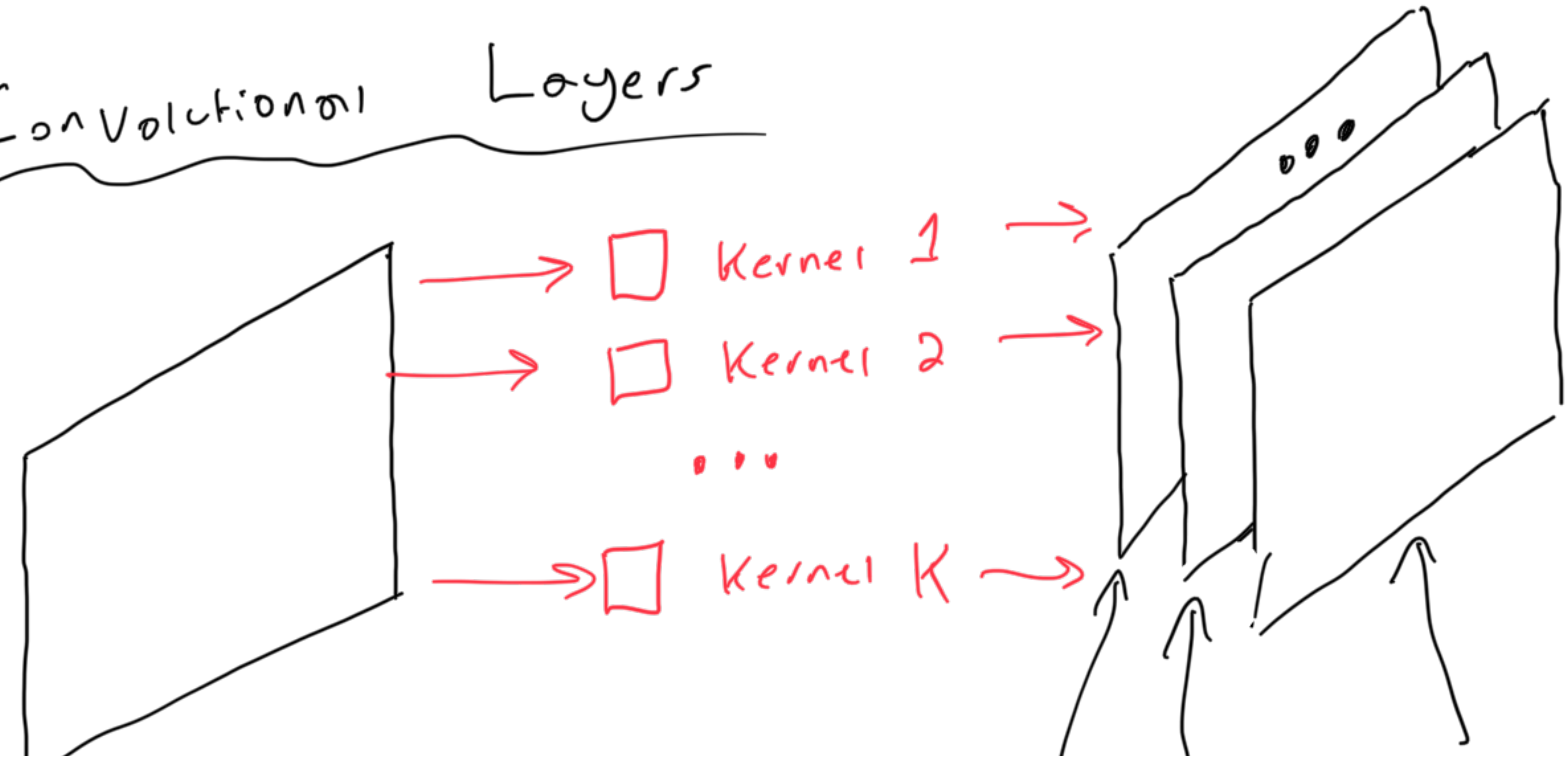
Day 21: Convolutional Neural Networks



Key ideas of CNNs:

1. use several kernels
2. learn the best kernels to use

Convolutional Layers



feature map K … feature map 2 feature map 1

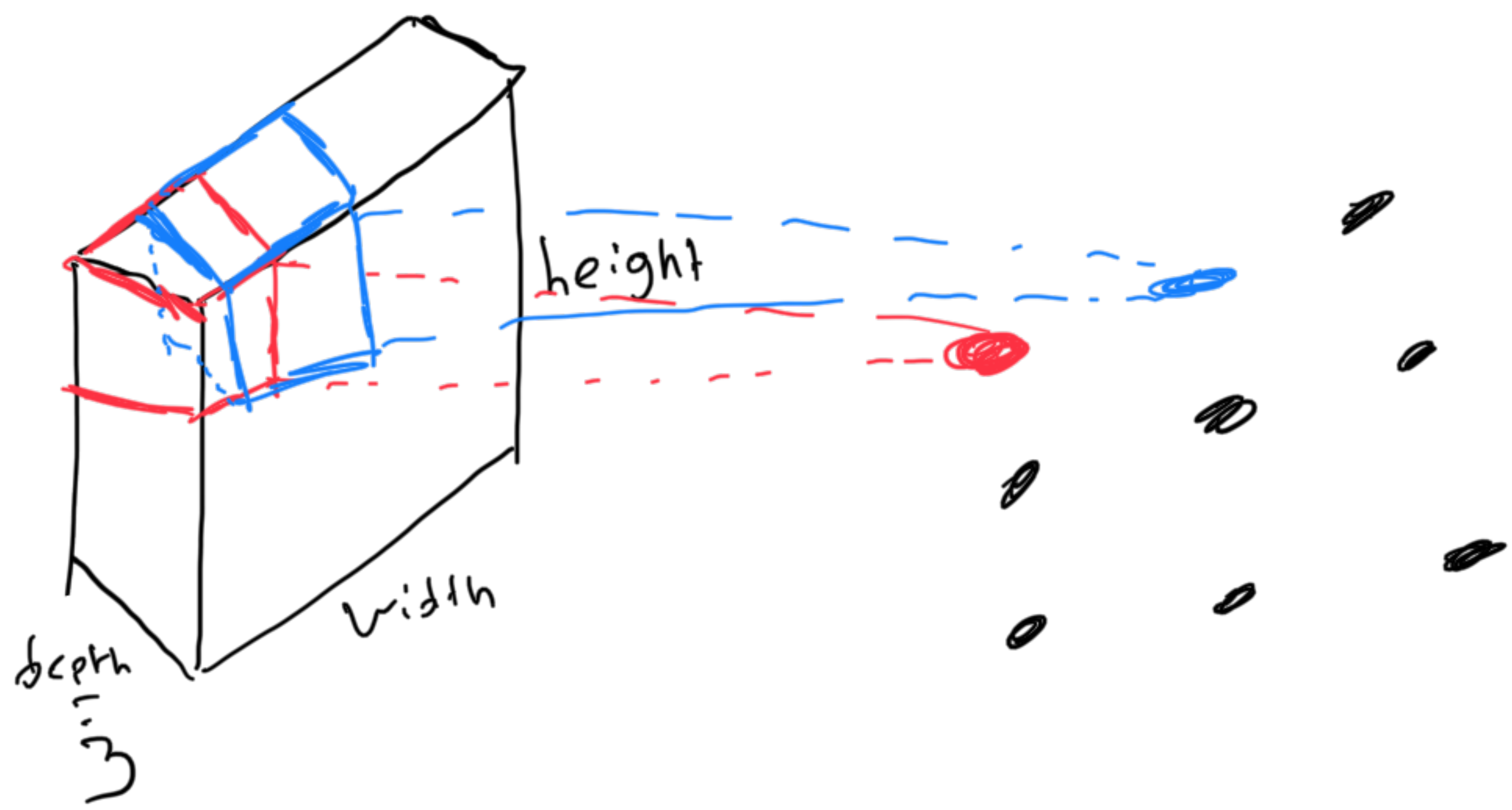
feature/activation map i is formed by convolving kernel i across the image

Recall: images are represented as RGB

$$\left[\begin{bmatrix} 255 & 255 & 255 \\ 0 & 255 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 255 & 255 & 255 \\ 0 & 255 & 0 \end{bmatrix} \right]$$

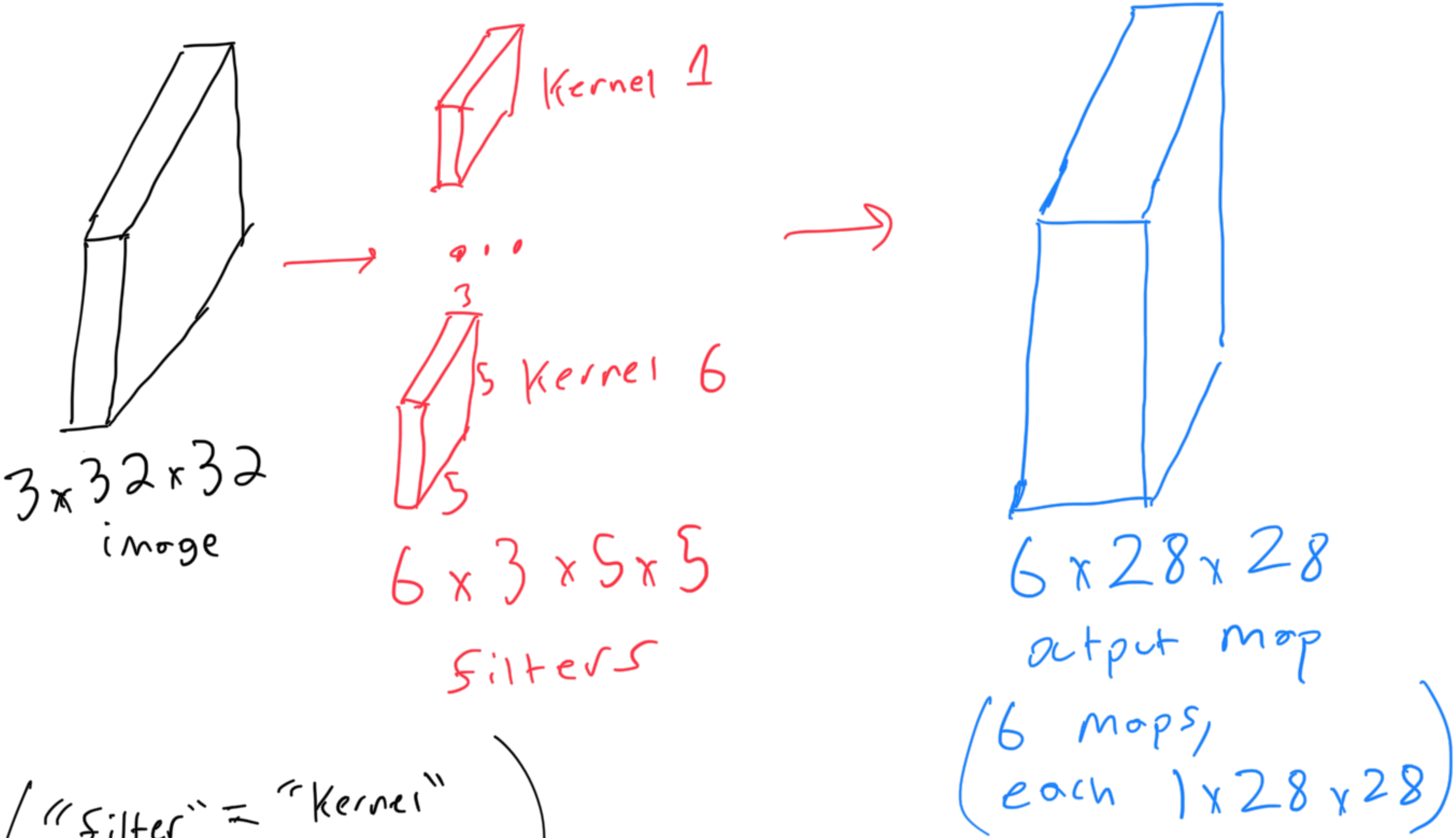
R G B

So for color images:



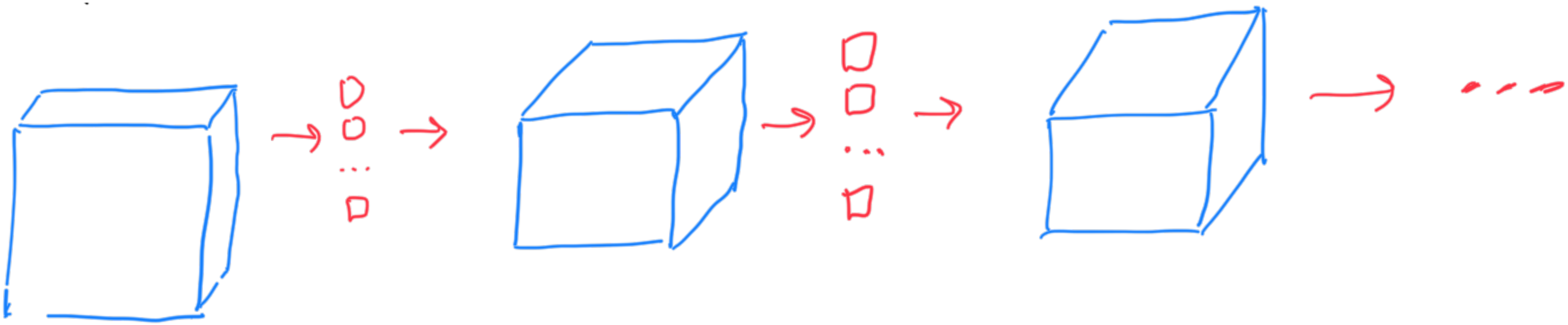
The weights/parameters are the kernels themselves

We stack the feature maps to get the input to the next layer:



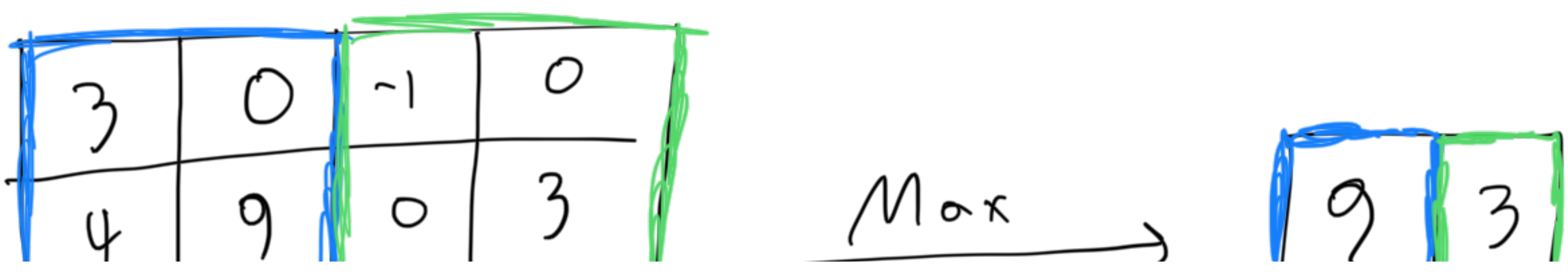
("filter" = "kernel"
"feature map" = "activation map")

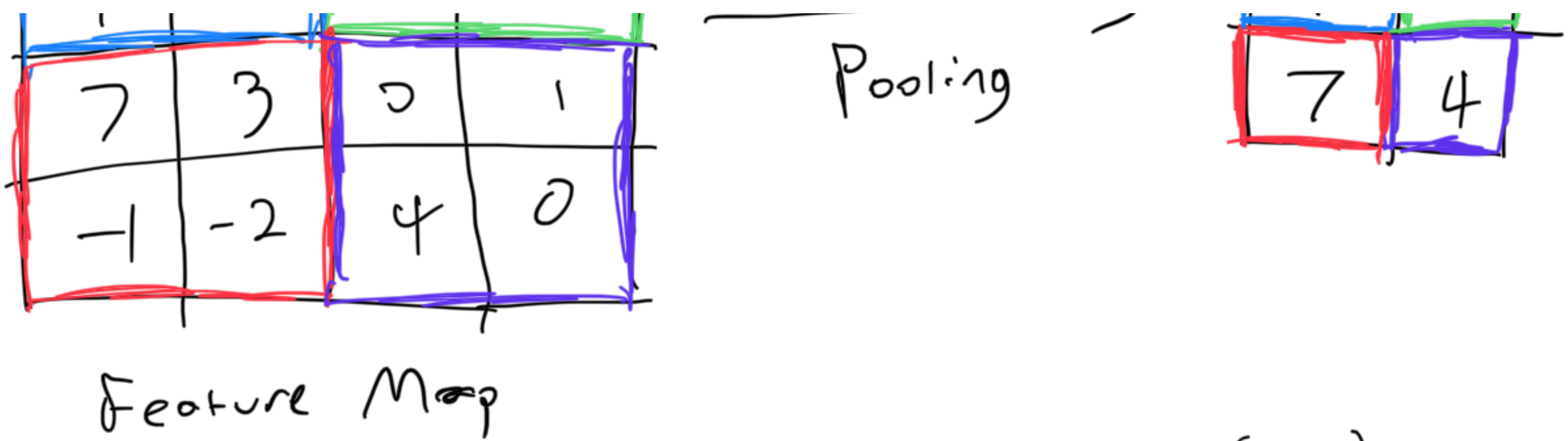
A CNN starts with several convolutional layers:



In order to learn higher-level (more "abstract") representations, we can downsample the feature maps to extract the most salient information:

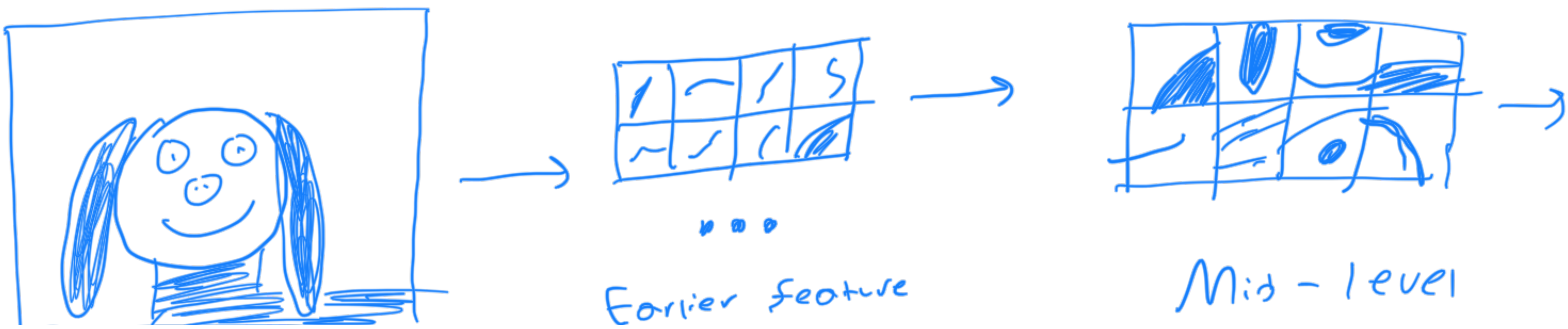
Max Pooling Layers





Take max element in $N \times N (\times N)$ window

Effect = each pixel in progressive layers represents larger portion of the original image



Input Image

maps: low-level features

features

High-Level Features

Putting it all together:

The "prototypical" CNN:

Input → Conv Layer (Repeat multiple times) → Max Pooling → ("flattening" the 3D tensor) Dense → Output Prediction

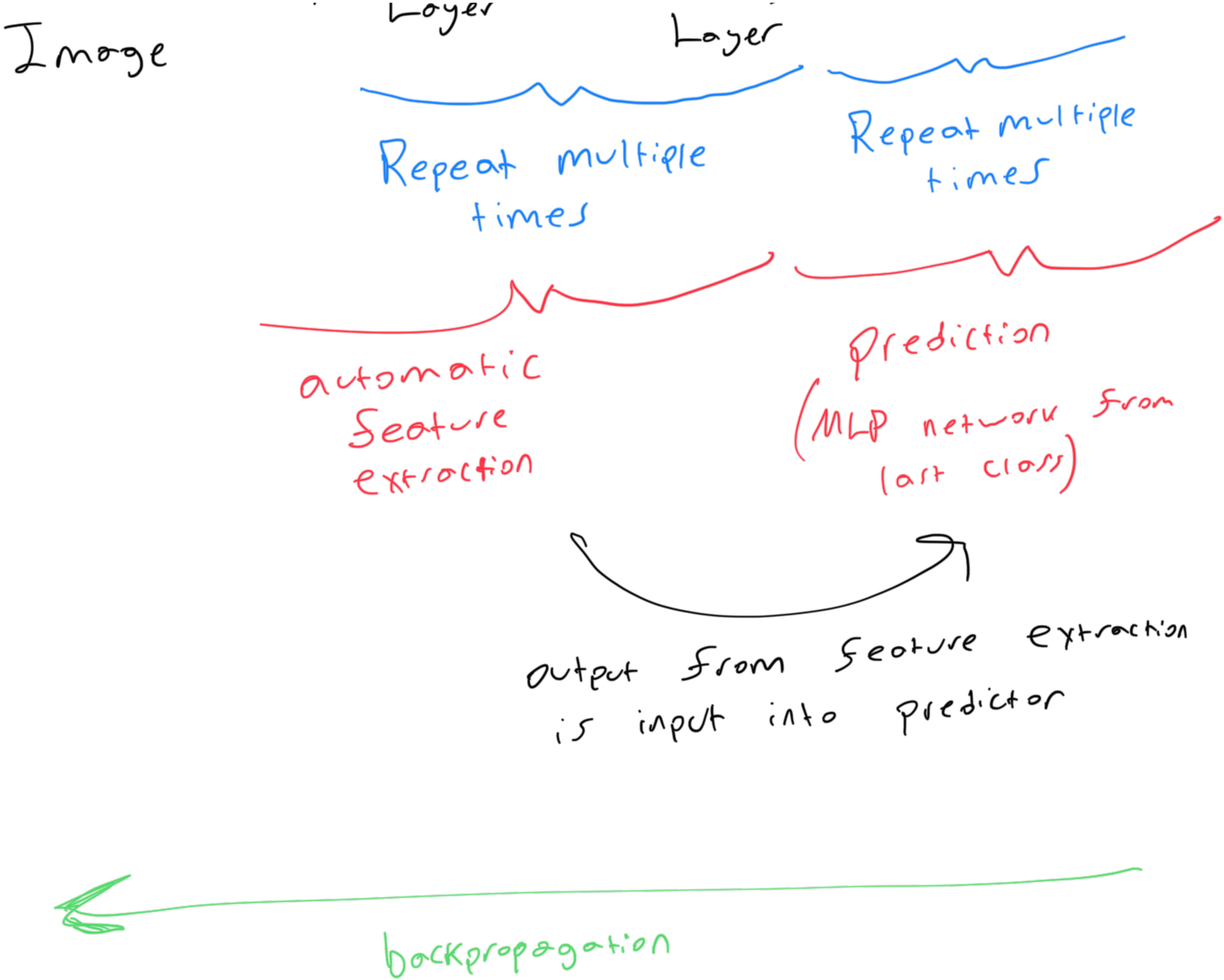


Image
Layer
Layer
Repeat multiple times
Repeat multiple times
automatic feature extraction
prediction (MLP network from last class)
output from feature extraction is input into predictor
backpropagation

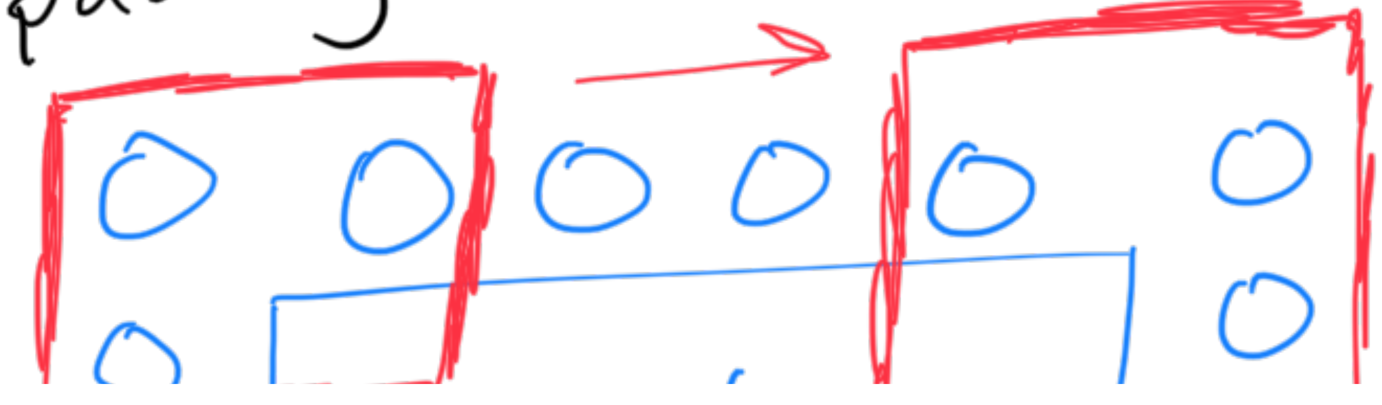
Common Hyperparameters in CNNs:

Dense Layers

- # of nodes
- non-linear activation function

Convolutional Layers

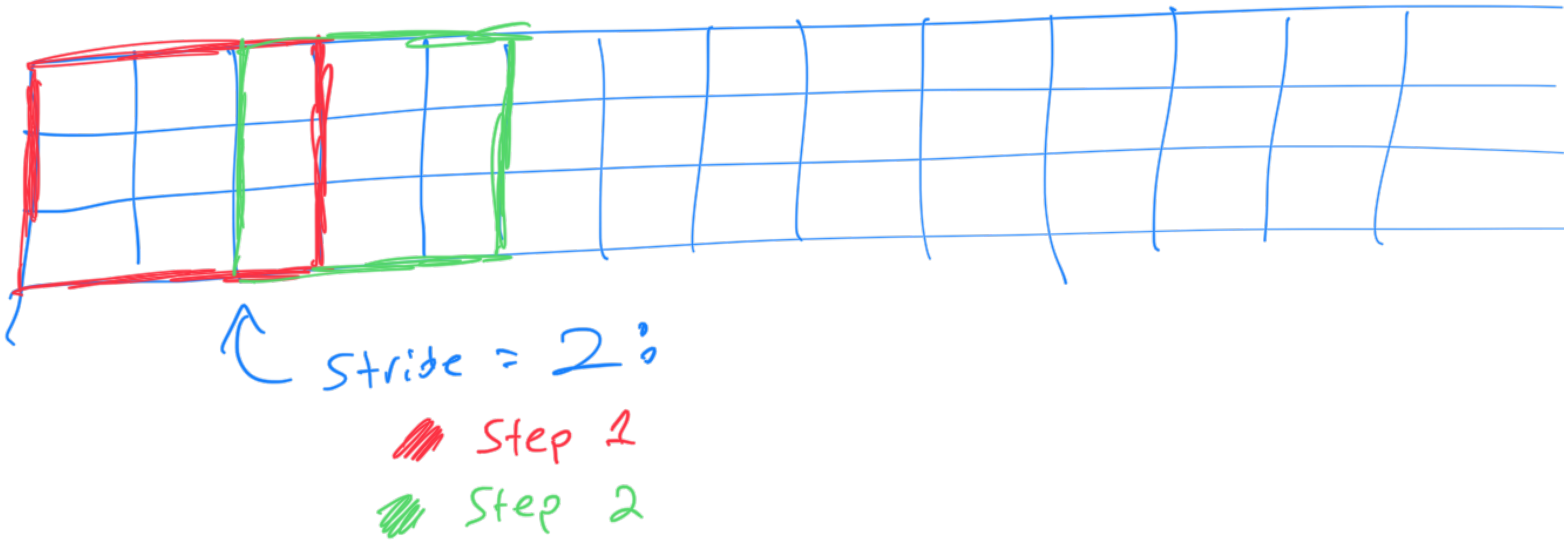
- # Kernels
- Kernel size
- padding



(can help ensure that output dimension =

input dimension)

• Stride
↳ how much to move the filter



Pooling Layers

- window size
- padding
- stride

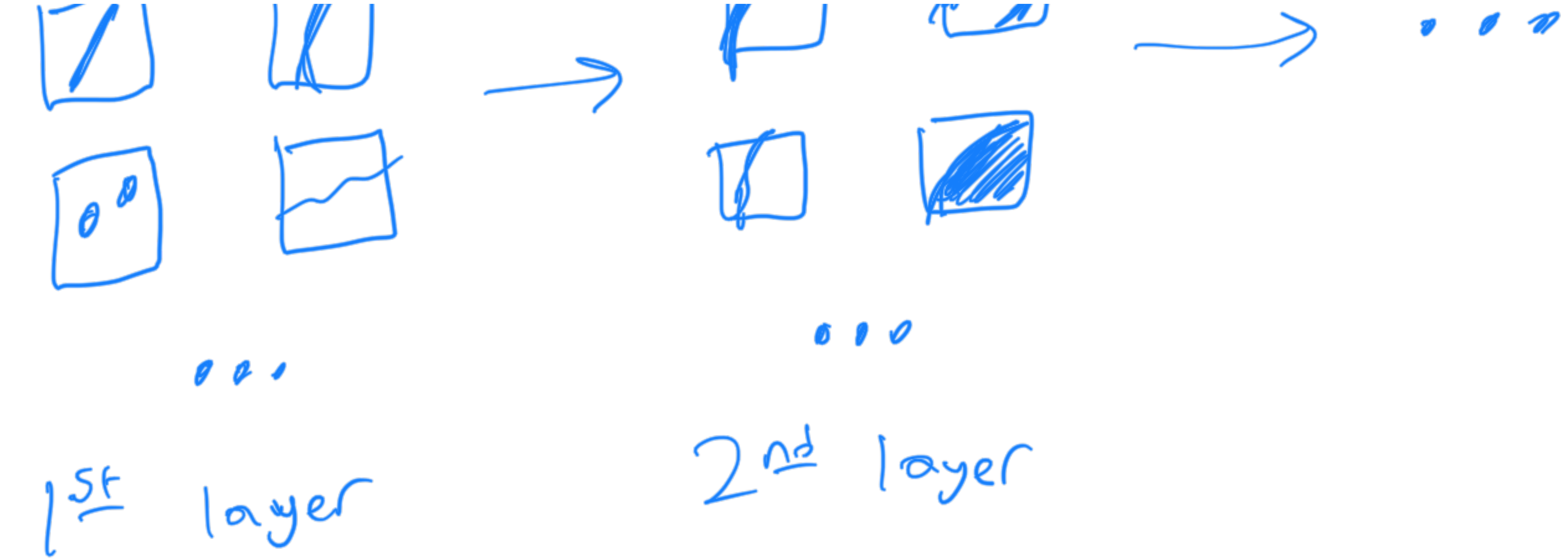
Transfer Learning

Many lower-level features can be re-used for a variety of purposes

Example:

Train a CNN to predict cat vs dog.

Feature maps:



The above features are useful to many prediction tasks besides cat vs. dog.

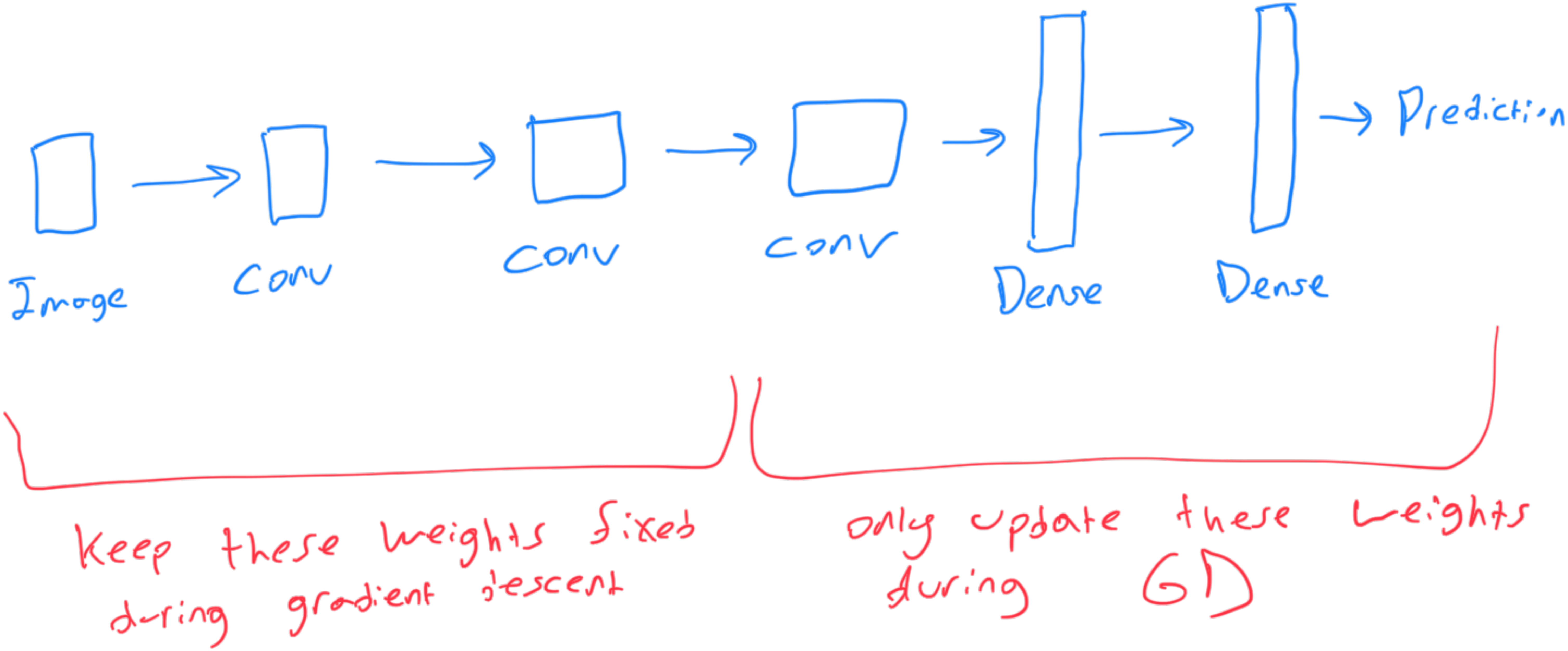
Steps of transfer learning:

(any dataset)

1. "Pre-train" a CNN on a dataset
2. "Freeze" the weights of the first N

convolutional layers

③ Train (aka transfer learn) the remaining weights on dataset of interest



Most common pre-training dataset is ImageNet:

ImageNet

- > 14 million images
- > 20,000 classes
- each class is a common object (e.g, Chair, remote control, grasshopper, ...)

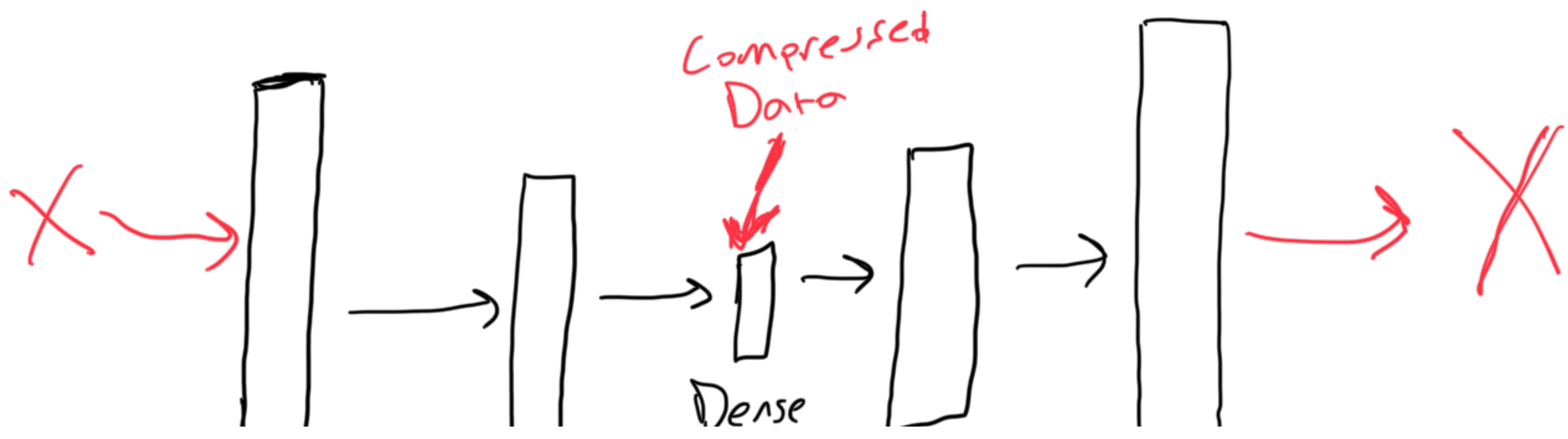
Many CNN architectures pre-trained on ImageNet, are readily available

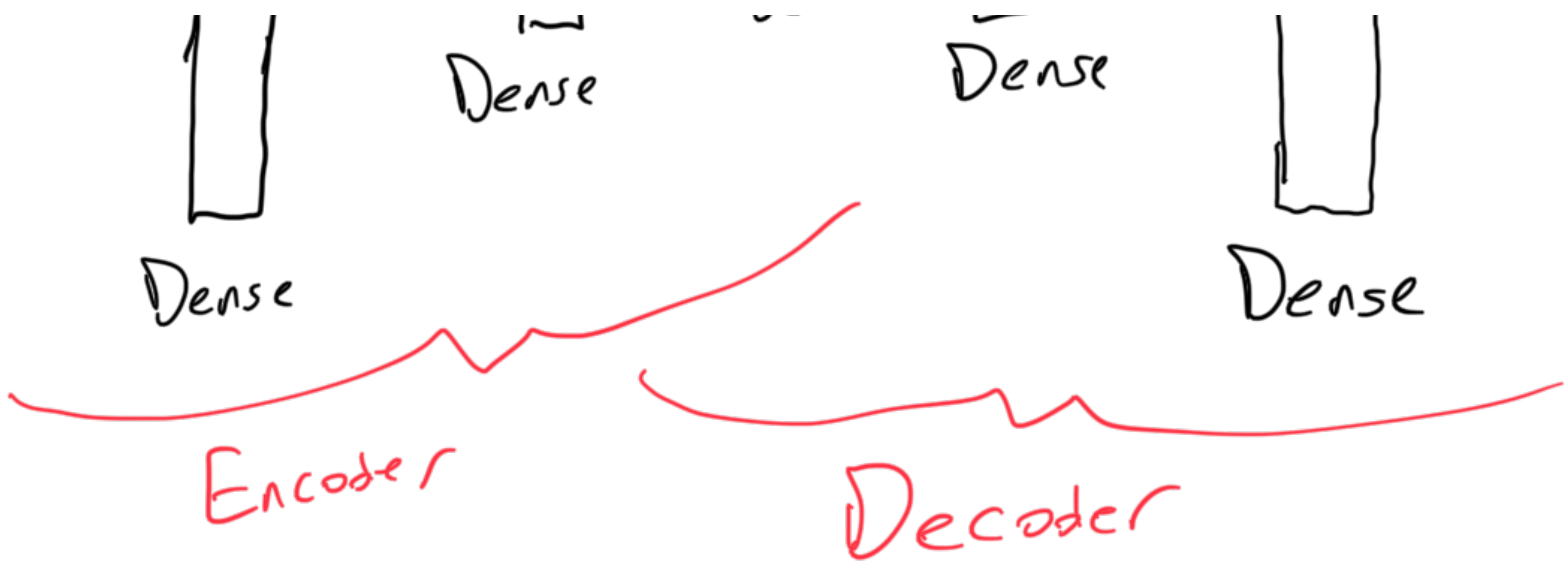
Starting with ImageNet weights is common practice

DL enables way more than just supervised learning

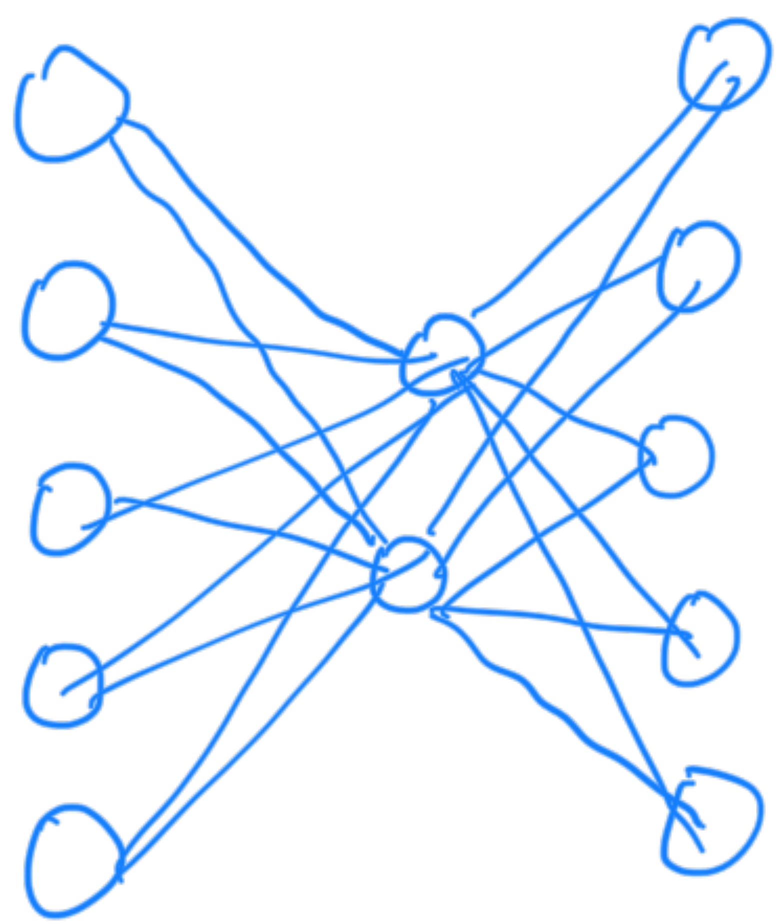
1st class example of many:

Dimensionality Reduction with Neural Networks: Autoencoders





Example:



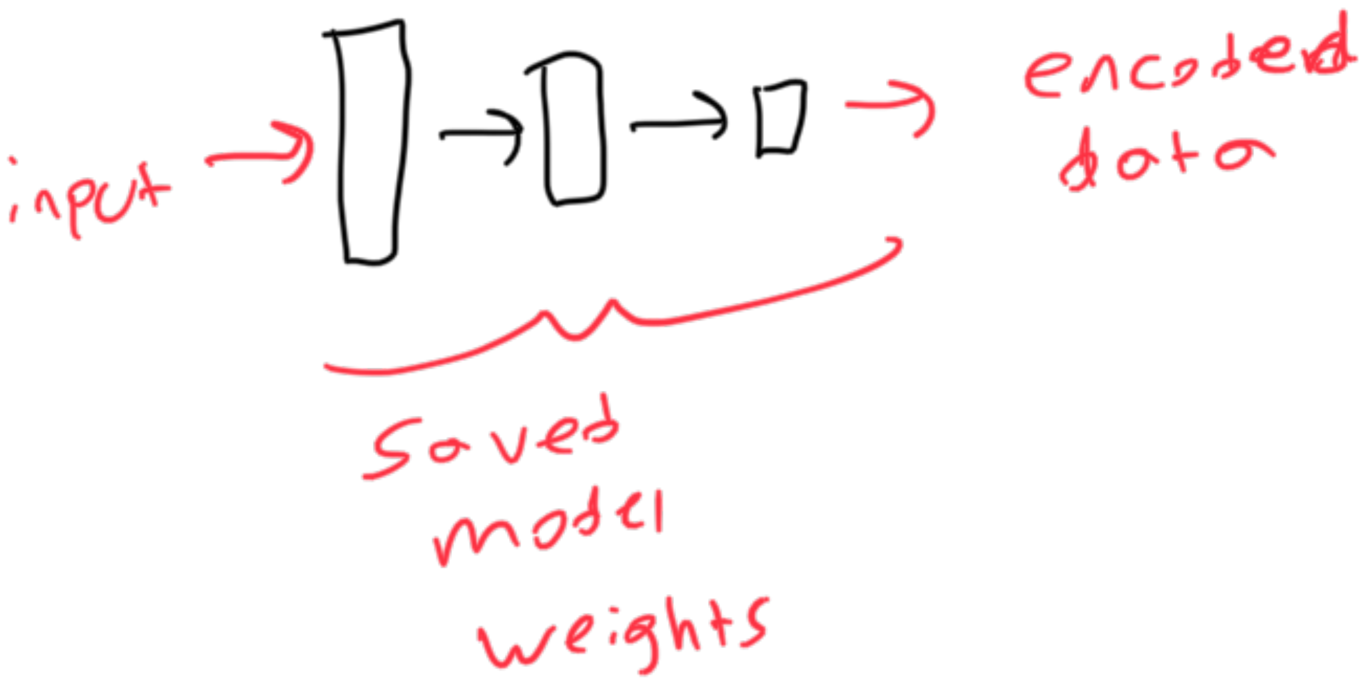
Reconstruction Loss: how similar is

Reconstruction ...

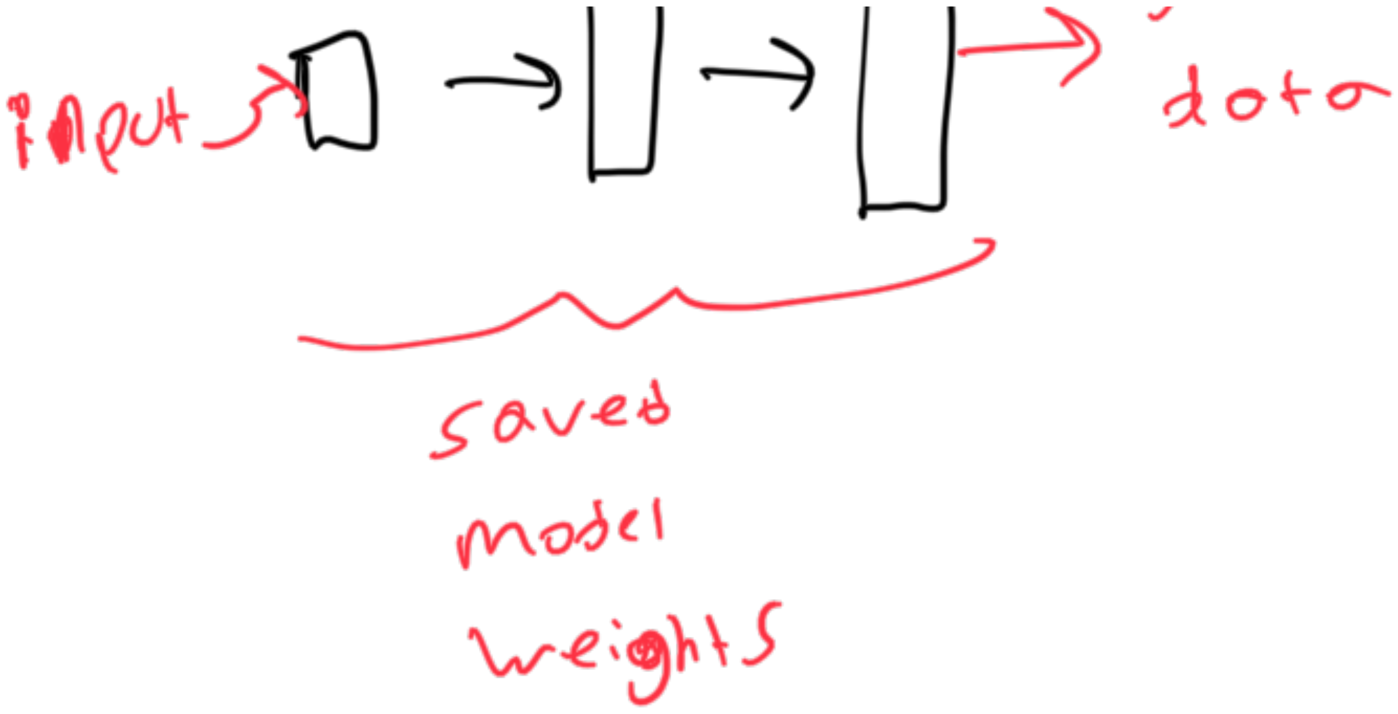
X_train to X_train?

Cross Entropy(X_train, X_train)

To encode data:



To decode data:



Convolutional Autoencoders work well w/ image data: