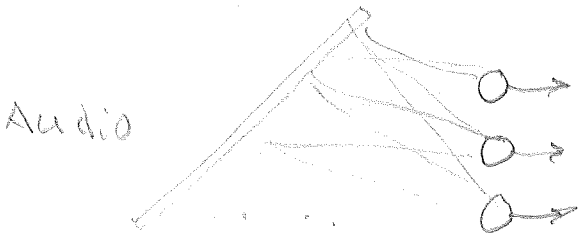
RECAP SO FAR

* Model of a single neuron (perceptron)

* Gradient descent learning (GD)
* Multi-layer Perceptron (MLP)
* GD for MLP

⇒ can learn anything from sufficient data

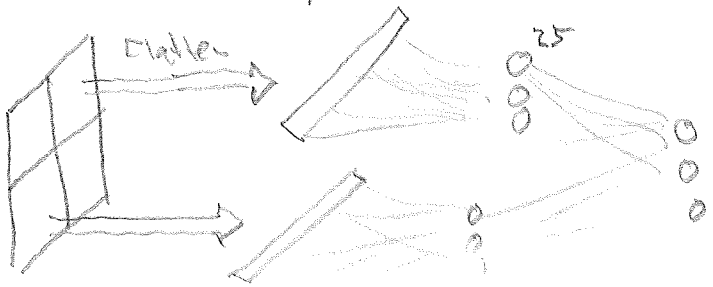


For 100×100 image w/ 100 hidden layer neurons + 3 output neurons

$$\underbrace{100\times100\times(100+1)}_{\text{hidden weights}} + (100+1)\times 3 = 1\text{M weights}$$

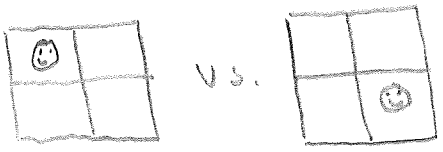
Flatten

Decrease # of weights 1: Localized detectors

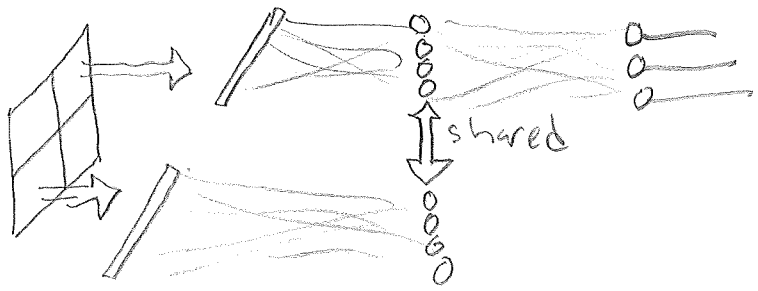


$$\underbrace{25\times25\times(25+1)}_{\text{one location}}\times 4 + (100+1)\times 3 = 65\text{k weights}$$

Prob.

* lack of invariance  vs.

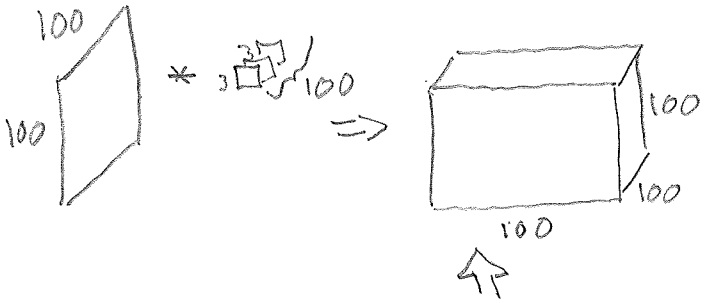
Extension to solution 1: Shared weights



$25 \times 25 \times (25+1) + (100+1) \times 3 \simeq 16k$ weights

By calculating extension in all locations we essentially have a convolutional neural network. (CNN).

CNN Learning Tricks

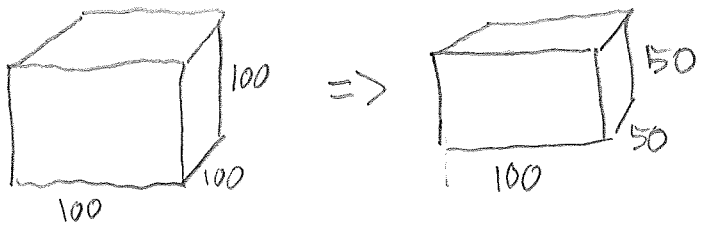


every location produces gradient and only few are meaningful
=> a lot of noise

Solution 1: stride

We jump over every Nth pixel (stride)

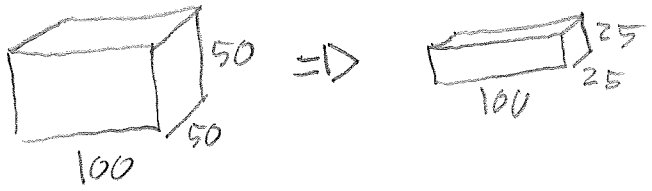
Stride 2



Solution 2: (Max) Pooling

Only the maximum response in certain neighborhood is retained

2x2 max pooling



After stride and pooling only 6.25% of the locations produce gradient update. In practice even less as only high response channels are selected => Filters specialize

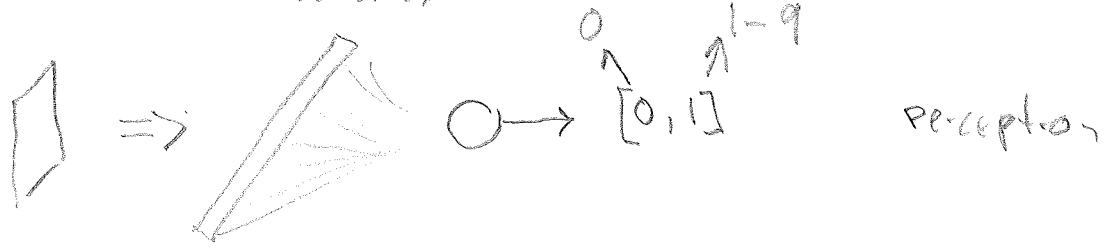
Lower levels learn elementary features and higher levels more advance feature (similar to human visual system)

[Video: Deep Dreams]

[notebook]

MNIST – Digits 0-9

Tested architectures



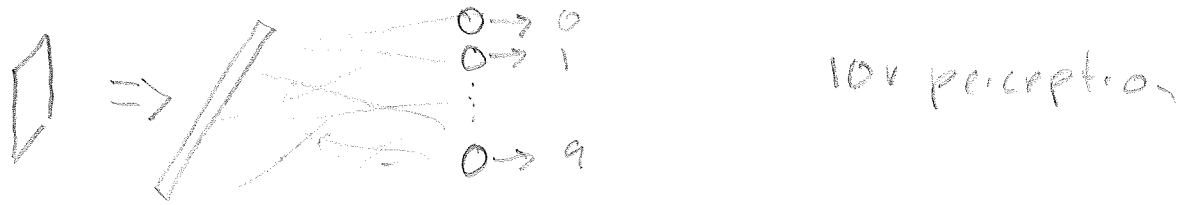
Perceptron

model =
tf.keras.models.Sequential()
* add layers one by one

model.compile()
* makes computation graph

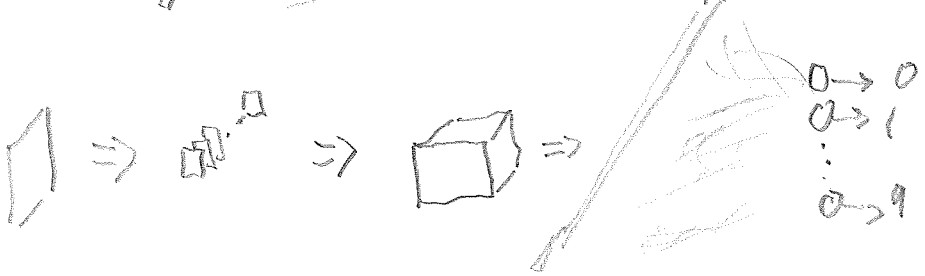
model.fit()
* training (batch_size, learning_rate)

model.predict()
* use model



10x perceptron

MLP



+ multiple conv + max pooling [homework]