Neural Networks

Textbook

This book has a lot of material:

Ian Goodfellow and Yoshua Bengio and Aaron Courville

Deep Learning

MIT Press, 2016

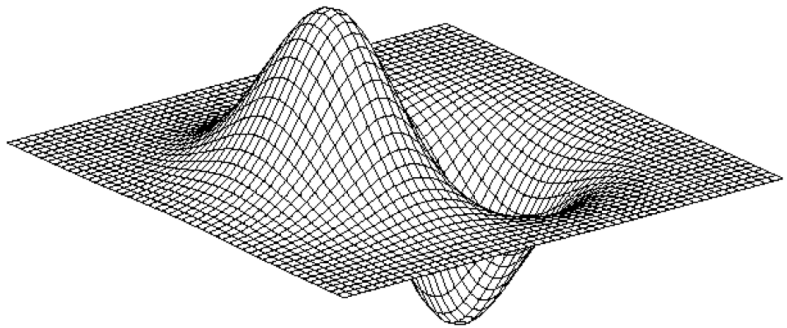
Online Available: `https://www.deeplearningbook.org`

What we already know

- Before we proceed, let's first give a shot to the techniques we already know
- Can we try edge detection with this image?

What we already know

- Before we proceed, let's first give a shot to the techniques we already know
- Can we try edge detection with this image?
- Well, let's run Canny edge detection!!!!



Derivative of Gaussian (x)

$\frac{\partial}{\partial x} h_\sigma(u, v)$

What we already know

- Before we proceed, let's first give a shot to the techniques we already know
- Can we try edge detection with this image?
- Well, let's run Canny edge detection!!!

Figure: Results from Canny Edge Detection.

What we already know

- Before we proceed, let's first give a shot to the techniques we already know
- Can we try edge detection with this image?
- Well, let's run Canny edge detection!!!
- Hold on a second, what about these images?


Figure: Objects have similar color with background.

What we already known

- Before we proceed, let's first give a shot to the techniques we already know
- Can we try edge detection with this image?
- Well, let's run Canny edge detection!!!
- Hold on a second, what about these images?


Figure: Over exposure. Under exosure. Hard illumination

What if we want to do a harder task?

- What's the category for one image?
- Can you manually write down a kernel to do it?






[Pic from: S. Lazebnik]

What if we want to do an even harder task?

- Tones of classes
- Can you manually write down a kernel to do it?

Neural Network

- What if I tell you that you can do all these tasks with fantastic accuracy (enough to get a D+ in Papert's class) with a single concept?

Neural Network

- What if I tell you that you can do all these tasks with fantastic accuracy (enough to get a D+ in Papert's class) with a single concept?
- This concept is called **Neural Networks**
- And it is quite simple.

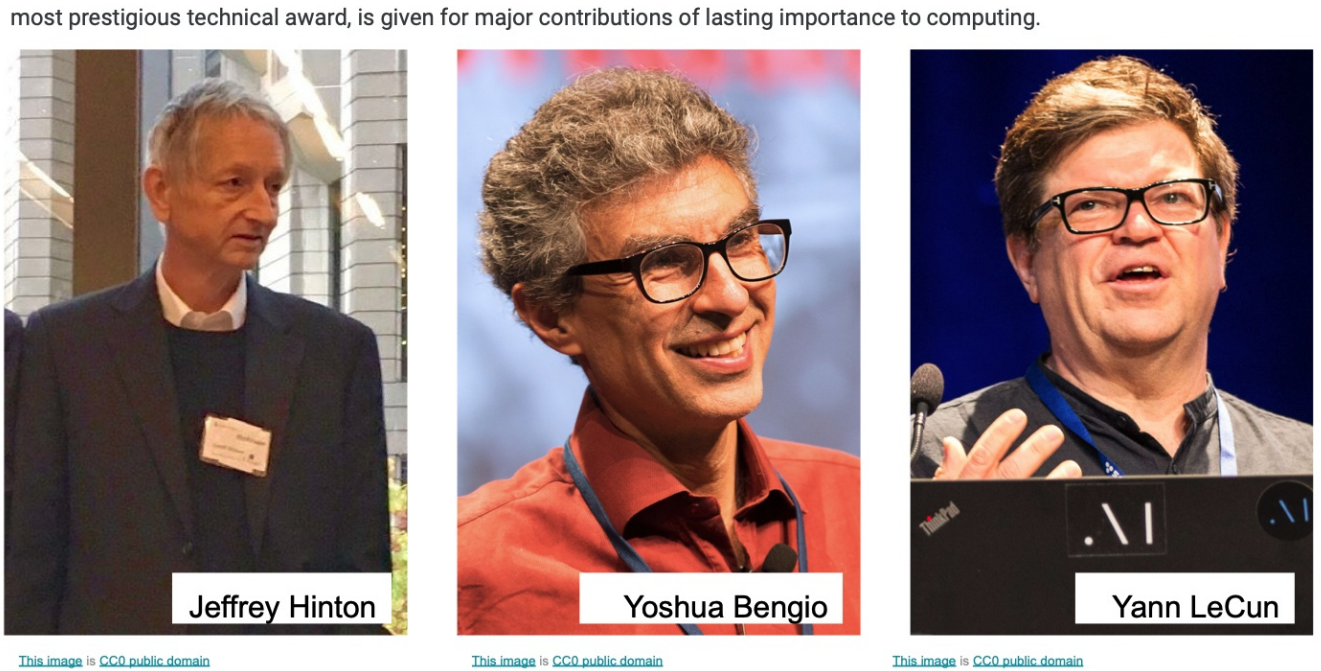


Figure: 2018 Turing Award for Deep Learning

Inspiration: The Brain

- Many machine learning methods inspired by biology, eg the (human) brain
- Our brain has $\sim 10^{11}$ neurons, each of which communicates (is connected) to $\sim 10^4$ other neurons

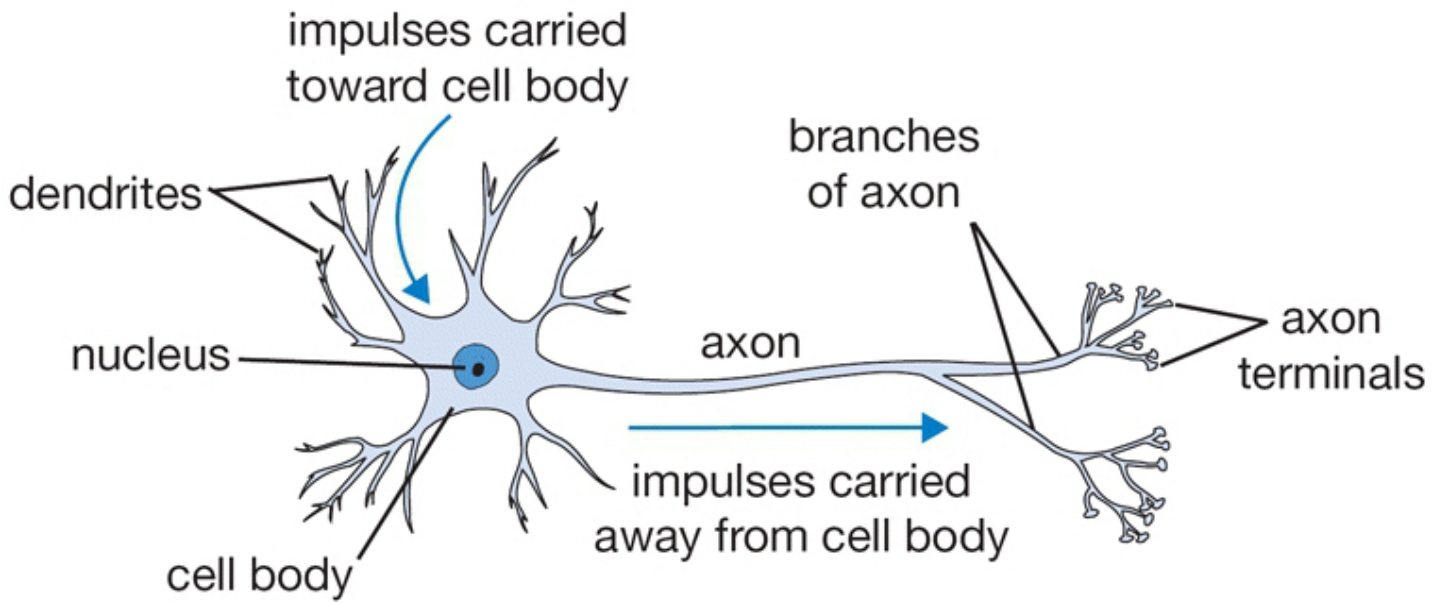


Figure: The basic computational unit of the brain: Neuron

[Pic credit: http://cs231n.github.io/neural-networks-1/]

Mathematical Model of a Neuron

- Neural networks define functions of the inputs (*hidden features*), computed by neurons
- Artificial neurons are called *units*
  - Input: x_0, x_1, x_2. Information from other neurons
  - Weights: w_0, w_1, w_2.
  - Output: $o = f(\sum_i w_i x_i + b)$. Output of this neuron

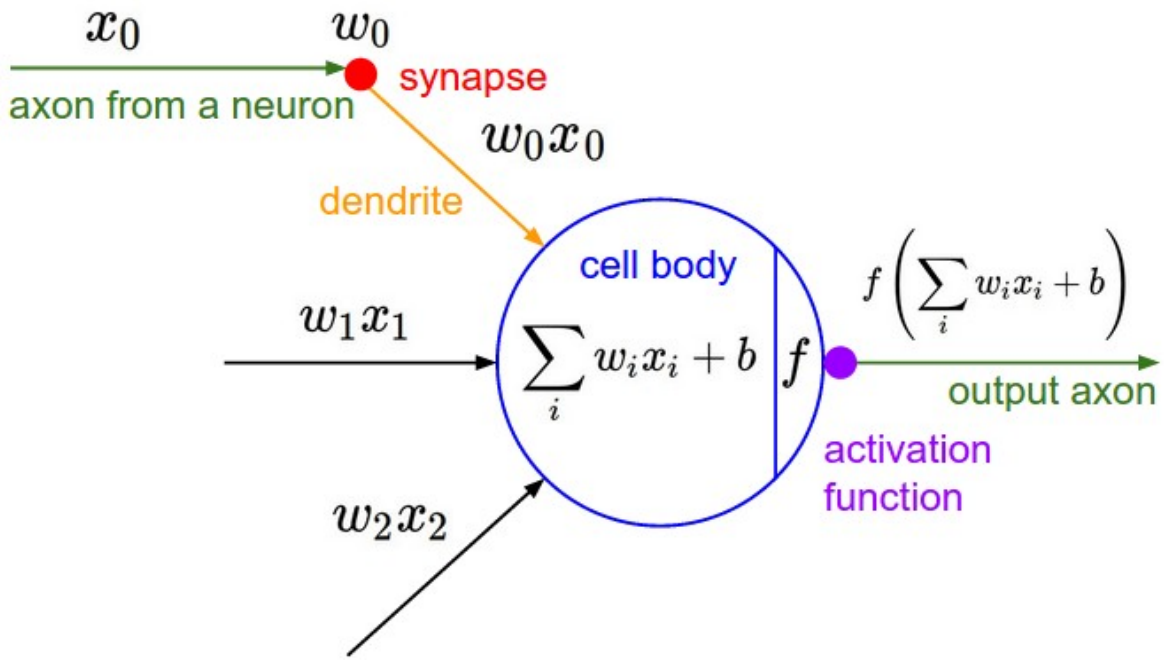
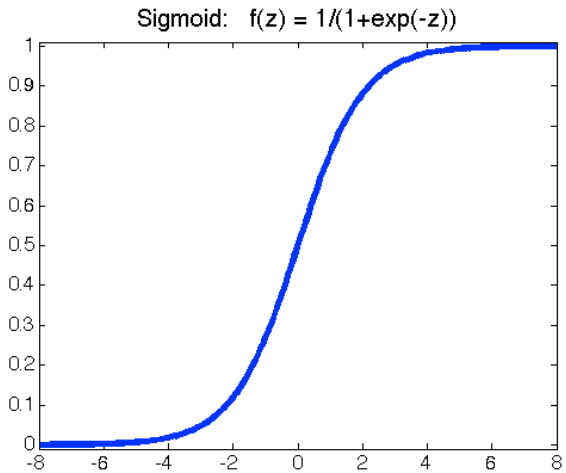


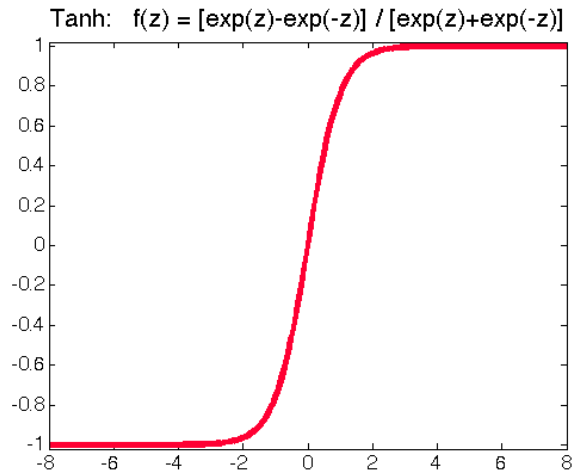
Figure: A mathematical model of the neuron in a neural network

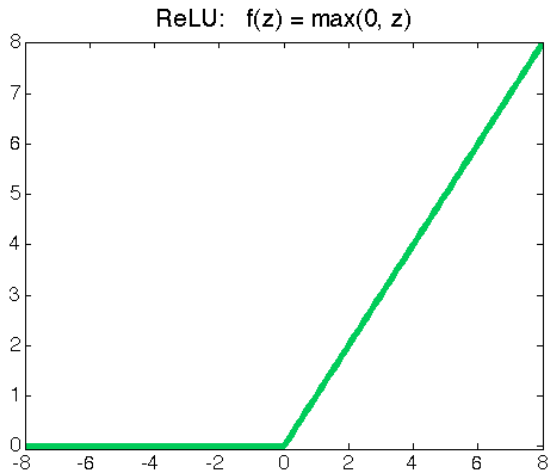
Activation Functions

Most commonly used activation functions:

- Suppose $z = \sum_i w_i x_i + b$
- Sigmoid: $\sigma(z) = \frac{1}{1+\exp(-z)}$
- Tanh: $\tanh(h) = \frac{\exp(z)-\exp(-z)}{\exp(z)+\exp(-z)}$
- ReLU (Rectified Linear Unit): $\mathrm{ReLU}(z) = \max(0, z)$
- Thinking: why do we need activation function? (we will talk later)







Neuron in Python

- Example in Python of a neuron with a sigmoid activation function

```
class Neuron(object):
  # ...
  def forward(inputs):
    """ assume inputs and weights are 1-D numpy arrays and bias is a number """
    cell_body_sum = np.sum(inputs * self.weights) + self.bias
    firing_rate = 1.0 / (1.0 + math.exp(-cell_body_sum)) # sigmoid activation function
    return firing_rate
```

Figure: Example code for computing the activation of a single neuron

[http://cs231n.github.io/neural-networks-1/]

A layer of neurons

- We have multiple neurons for output layer

A layer of neurons

- We have multiple neurons for output layer

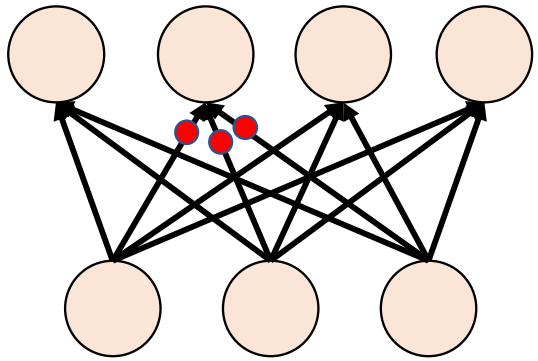


Figure: A mathematical model of a layer in a neural network

- Each unit computes its value based on linear combination of values of units that point into it, and an activation function
- $o_0 = f(w_{0,0}x_0 + w_{0,1}x_1 + w_{0,2}x_2 + b_0)$
- $o_1 = f(w_{1,0}x_0 + w_{1,1}x_1 + w_{1,2}x_2 + b_1)$
- $o_2 = f(w_{2,0}x_0 + w_{2,1}x_1 + w_{2,2}x_2 + b_2)$
- $o_3 = f(w_{3,0}x_0 + w_{3,1}x_1 + w_{3,2}x_2 + b_3)$

A layer of neurons

- We have multiple neurons for output layer

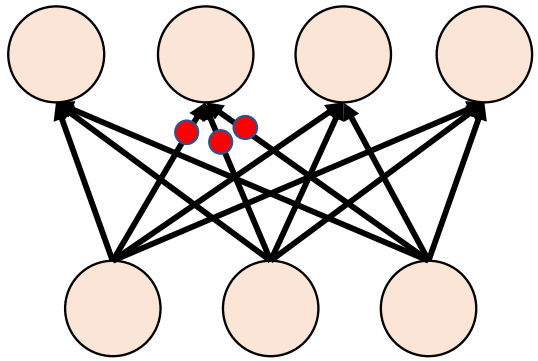


Figure: A mathematical model of a layer in a neural network

- Each unit computes its value based on linear combination of values of units that point into it, and an activation function
- Written in a matrix form:
  - $\mathbf{o} = f(\mathbf{wx} + \mathbf{b})$:
  - Size of each element: $\mathbf{o} : 4 \times 1, \mathbf{w} : 4 \times 3, \mathbf{x} : 3 \times 1, \mathbf{b} : 4 \times 1$

One layer in Python

- Example in Python of one layer with a sigmoid activation function

```
f = lambda x: 1.0 / (1.0 + np.exp(-x)) # Sigmod activation function
Input_dim, Output_dim = 3, 4
class OneLayer(object):
    def init():
        self.W = np.randn(Output_dim, Input_dim) # Random initialize the weight
        self.b = np.zeros(Output_dim, 1) # Bias
    def forward(x):
        # Input data x: Input_dim x 1
        o = f(np.dot(self.W, x) + self.b) # sigmoid(W1 * x + b1)
        return o
```

Figure: Example code for computing the activation of one layer neurons

Neural Network Architecture (Multi-Layer Perceptron)

- Network with one layer of four hidden units:

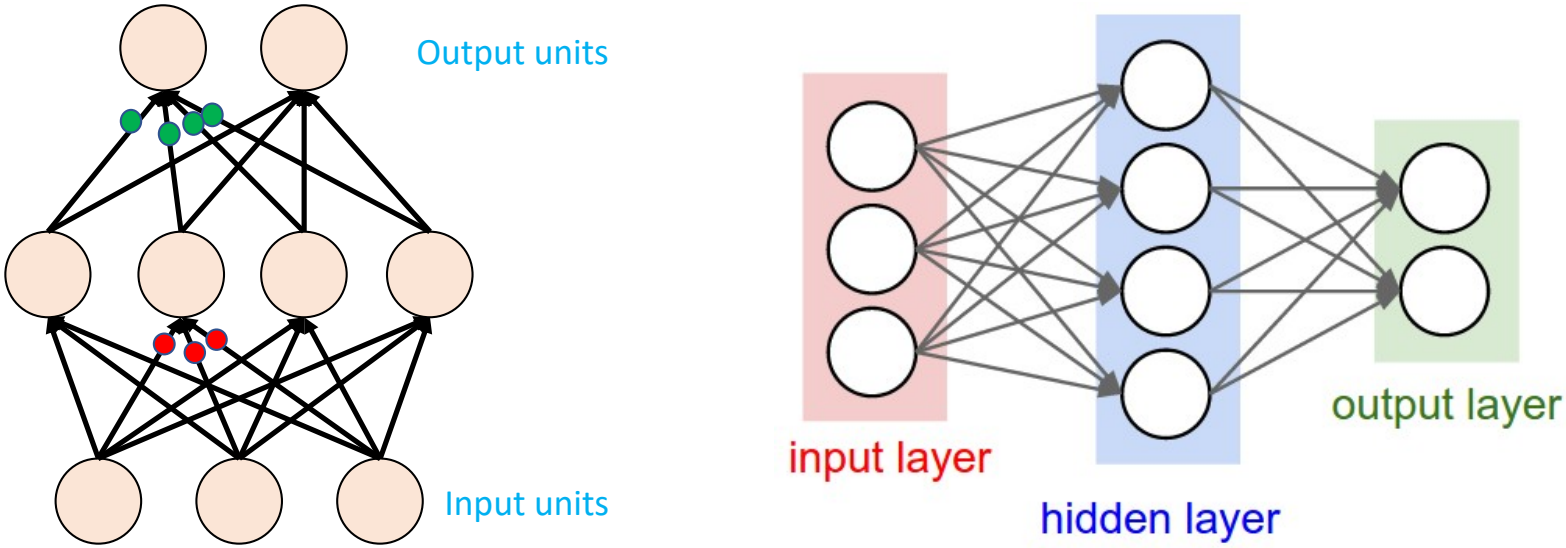


Figure: Two different visualizations of a 2-layer neural network. In this example: 3 input uni
4 hidden units and 2 output units

- Naming conventions; a 2-layer neural network:
  - One layer of hidden units
  - One output layer
    (we do not count the inputs as a layer)

[http://cs231n.github.io/neural-networks-1/]

Neural Network Architecture (Multi-Layer Perceptron)

- Going deeper: a 3-layer neural network with two layers of hidden units

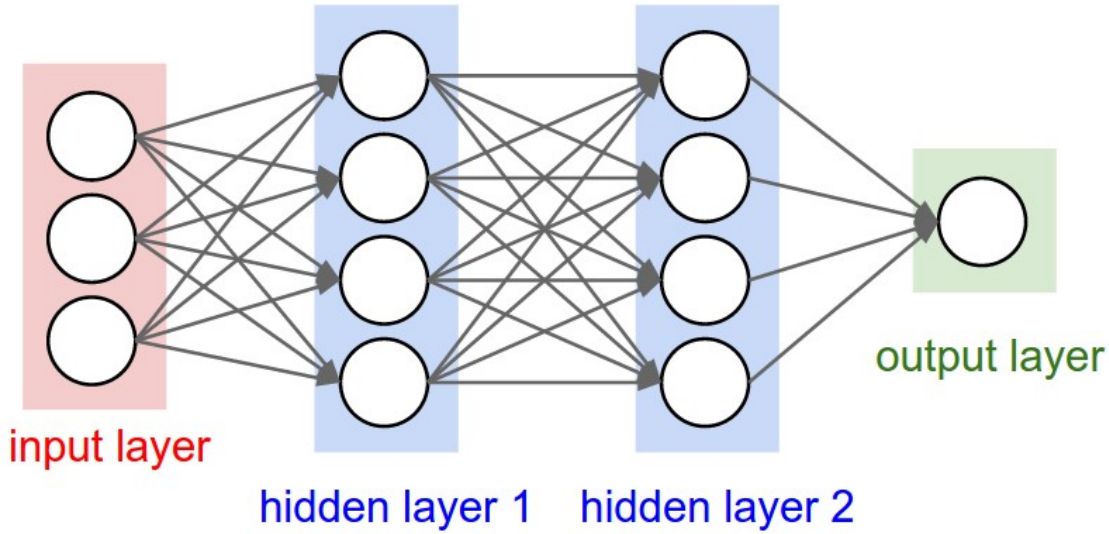


Figure: A 3-layer neural net with 3 input units, 4 hidden units in the first and second hidden layer and 1 output unit

- Naming conventions; a N-layer neural network:
  - $N - 1$ layers of hidden units
  - One output layer

[http://cs231n.github.io/neural-networks-1/]

Two layers in Python

- Example in Python of two layers with a sigmoid activation function

```
f = lambda x: 1.0 / (1.0 + np.exp(-x)) # Sigmod activation function
Input_dim, Hidden_dim, Output_dim = 3, 4, 2
class TwoLayerMLP(object):
    def init():
        self.W1 = np.randn(Hidden_dim, Input_dim) # Random init weight for the 1st laye
        self.b1 = np.zeros(Hidden_dim, 1) # Bias for the 1st layer
        self.W2 = np.randn(Output_dim, Hidden_dim) # Random init weight for the 2nd lay
        self.b2 = np.zeros(Output_dim, 1) # Bias for the 2nd layer
    def forward(x):
        # Input data x: Input_dim x 1
        h = f(np.dot(self.W1, x) + self.b1) # sigmoid(W1 * x + b1)
        output = f(np.dot(self.W2, h) + self.b2) # Sigmoid (W2 * h + b2)
        return output
```

Figure: Example code for computing the activation of two layer neurons

Why we need activation functions?

- What if we do not have activation function f in the case of deep network?
- One-layer neural network: $\mathbf{o} = \mathbf{w}_1\mathbf{x} + \mathbf{b}_1$

Why we need activation functions?

- What if we do not have activation function f in the case of deep network?
- One-layer neural network: $\mathbf{o} = \mathbf{w}_1\mathbf{x} + \mathbf{b_1}$
- Two-layer neural network: $\mathbf{o} = \mathbf{w}_2(\mathbf{w}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2$

Why we need activation functions?

- What if we do not have activation function f in the case of deep network?
- One-layer neural network: $\mathbf{o} = \mathbf{w}_1\mathbf{x} + \mathbf{b_1}$
- Two-layer neural network: $\mathbf{o} = \mathbf{w}_2(\mathbf{w}_1\mathbf{x} + \mathbf{b_1}) + \mathbf{b_2}$
- N-layer neural network: $\mathbf{o} = \mathbf{w}_N(\cdots(\mathbf{w}_1\mathbf{x} + \mathbf{b_1}) + \mathbf{b_2}) + ...)$

Why we need activation functions?

- What if we do not have activation function f in the case of deep network?
- One-layer neural network: $\mathbf{o} = \mathbf{w}_1\mathbf{x} + \mathbf{b_1}$
- Two-layer neural network: $\mathbf{o} = \mathbf{w}_2(\mathbf{w}_1\mathbf{x} + \mathbf{b_1}) + \mathbf{b_2}$
- N-layer neural network: $\mathbf{o} = \mathbf{w}_N(\cdots(\mathbf{w}_1\mathbf{x} + \mathbf{b_1}) + \mathbf{b_2}) + ...)$
- Even if we have infinite layers, it will still be a linear function :(

Why we need activation functions?

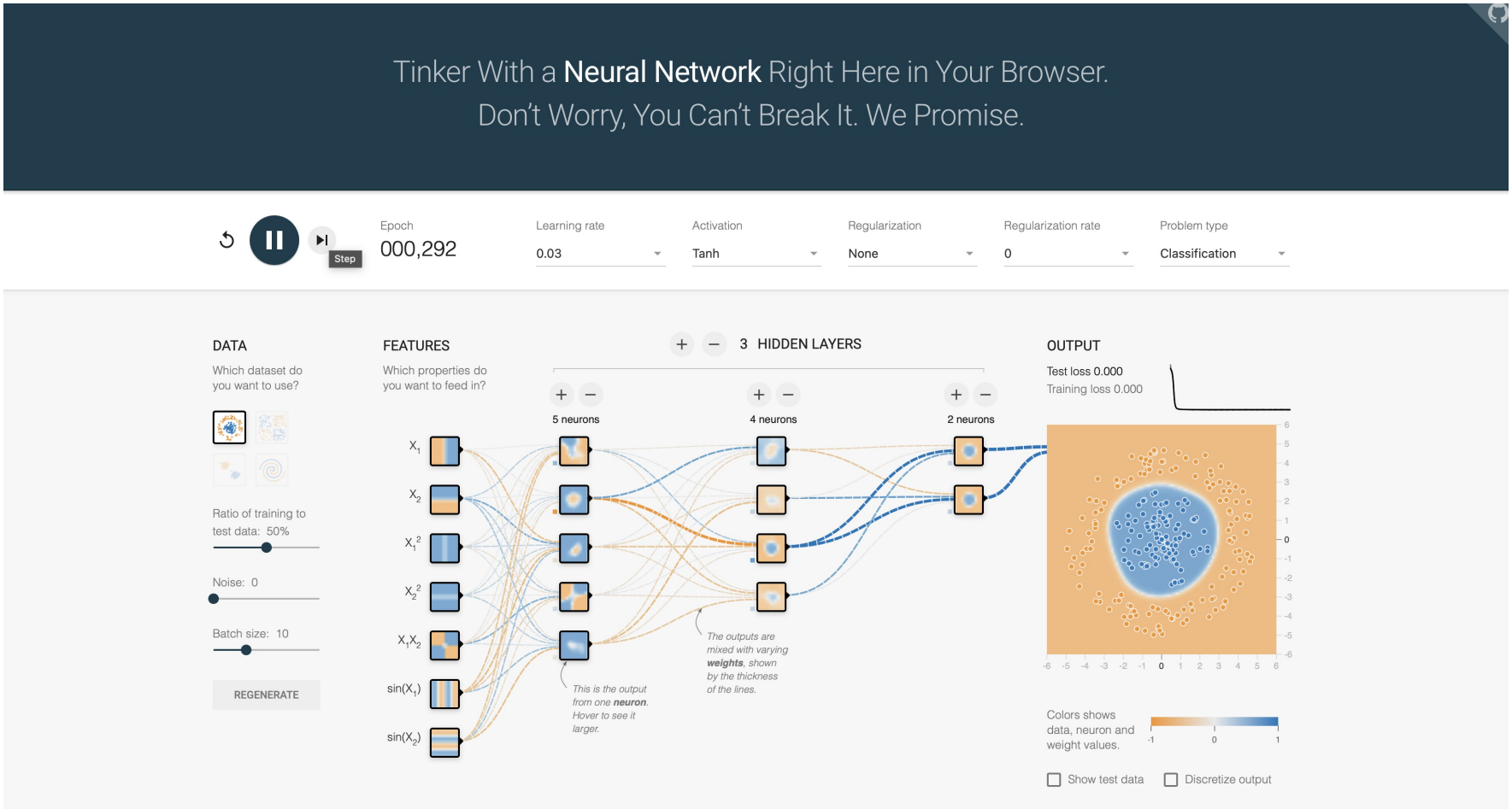
- What if we do not have activation function f in the case of deep network?
- One-layer neural network: $\mathbf{o} = \mathbf{w}_1\mathbf{x} + \mathbf{b_1}$
- Two-layer neural network: $\mathbf{o} = \mathbf{w}_2(\mathbf{w}_1\mathbf{x} + \mathbf{b_1}) + \mathbf{b_2}$
- N-layer neural network: $\mathbf{o} = \mathbf{w}_N(\cdots(\mathbf{w}_1\mathbf{x} + \mathbf{b_1}) + \mathbf{b_2}) + ...)$
- Even if we have infinite layers, it will still be a linear function :(

Why we need activation functions?

- What if we have activation function f in the case of deep network?
- One-layer neural network: $\mathbf{o} = f(\mathbf{w}_1\mathbf{x} + \mathbf{b_1})$
- Two-layer neural network: $\mathbf{o} = f(\mathbf{w}_2 f(\mathbf{w}_1\mathbf{x} + \mathbf{b_1}) + \mathbf{b_2})$
- N-layer neural network: $\mathbf{o} = f(\mathbf{w}_N f(\cdots f(\mathbf{w}_1\mathbf{x} + \mathbf{b_1}) + \mathbf{b_2}) + ...)$
- More non-linearity with more layers.

Let's play with it Online!

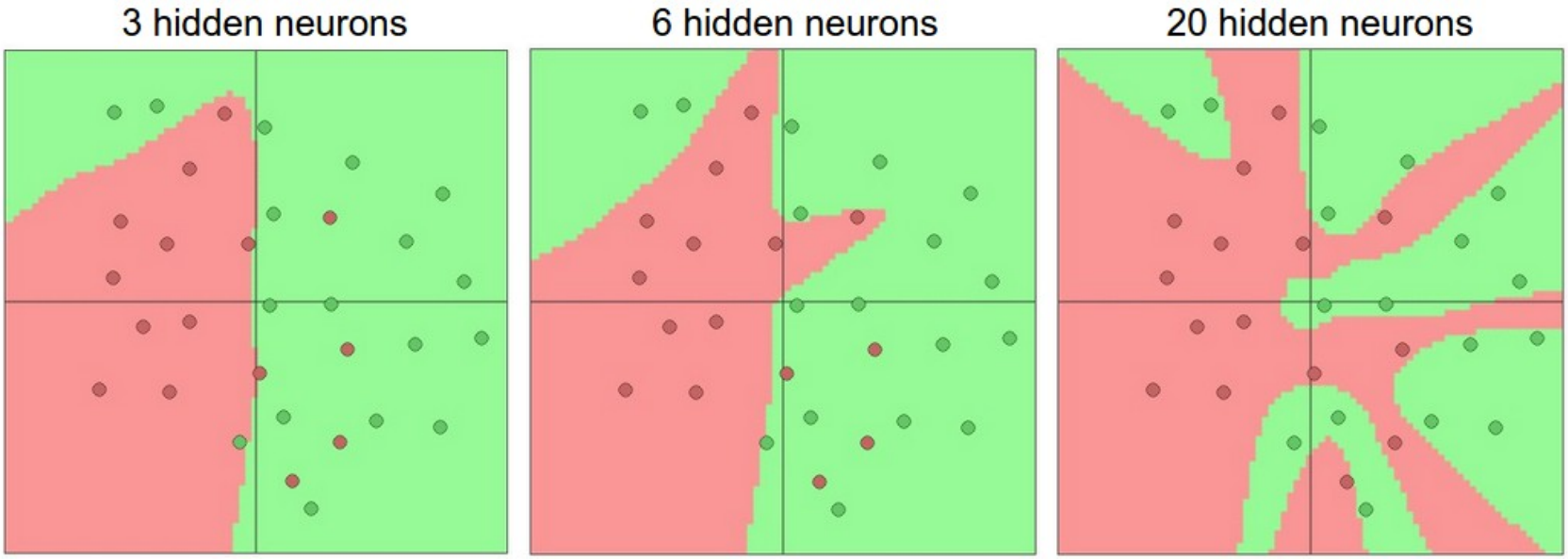
- A Neural Network Playground: https://playground.tensorflow.org



Representational Power

- Neural network with at **least one hidden layer** is a universal approximator (can represent any function).

  Proof in: Approximation by Superpositions of Sigmoidal Function, Cybenko, paper

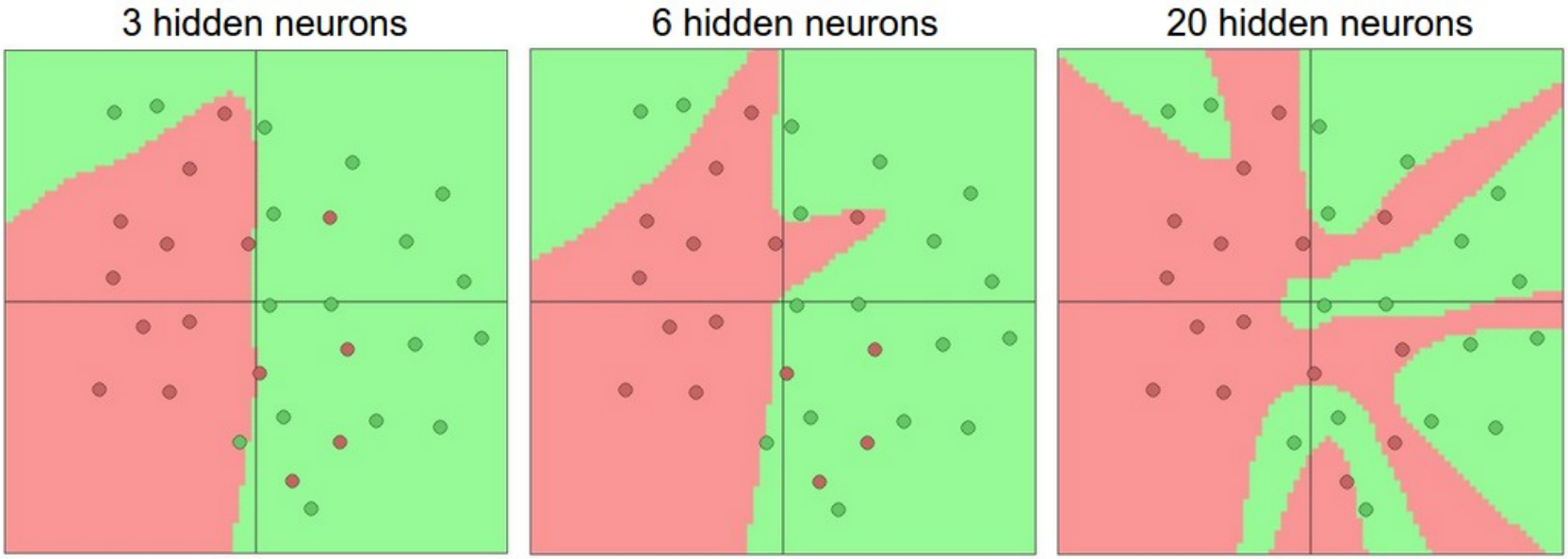


- The capacity of the network increases with more hidden units and more hidden layers

Representational Power

- Neural network with at **least one hidden layer** is a universal approximator (can represent any function).

  Proof in: Approximation by Superpositions of Sigmoidal Function, Cybenko, paper



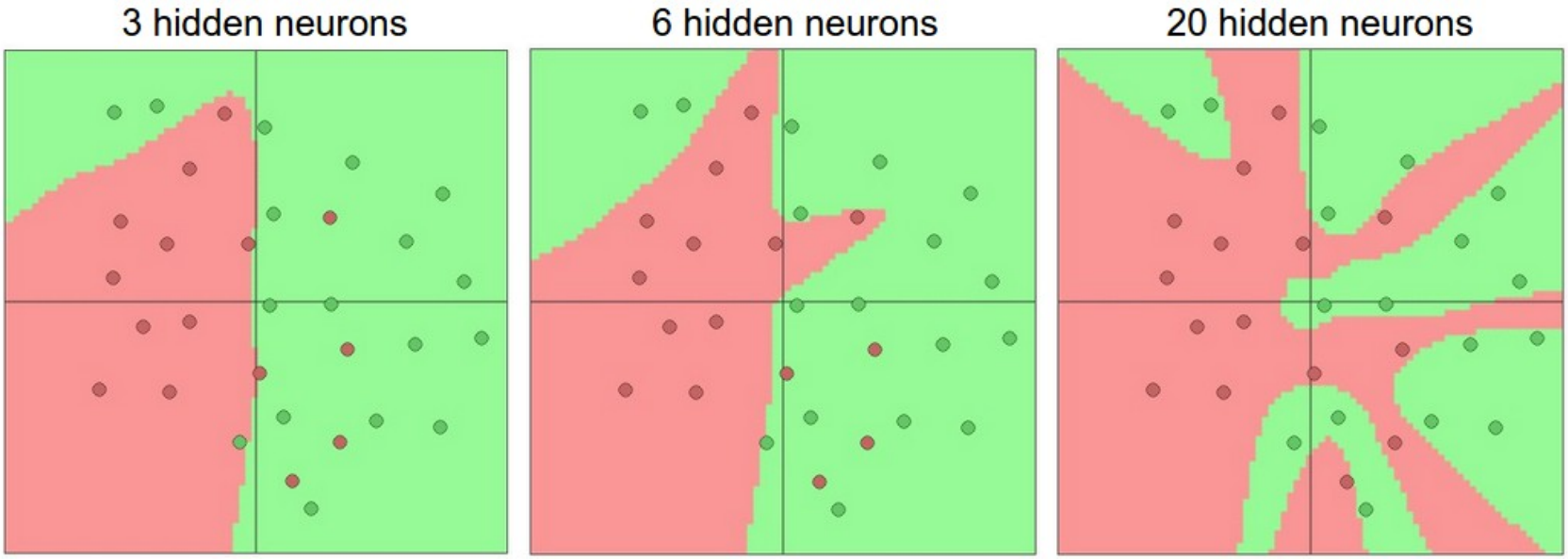
- The capacity of the network increases with more hidden units and more hidden layers
- Why go deeper? Read eg: Do Deep Nets Really Need to be Deep? Jimmy Ba, Rich Caruana, Paper: paper]

[http://cs231n.github.io/neural-networks-1/]

Representational Power

- Neural network with at **least one hidden layer** is a universal approximator (can represent any function).

  Proof in: Approximation by Superpositions of Sigmoidal Function, Cybenko, paper



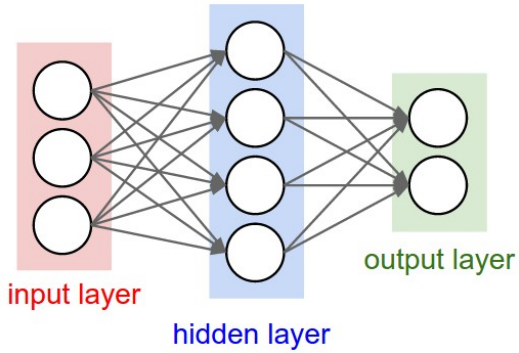
- The capacity of the network increases with more hidden units and more hidden layers
- Why go deeper? Read eg: Do Deep Nets Really Need to be Deep? Jimmy Ba, Rich Caruana, Paper: paper]

[http://cs231n.github.io/neural-networks-1/]

Neural Networks

- We only need to know two algorithms
  - *Forward pass:* performs inference
  - *Backward pass:* performs training

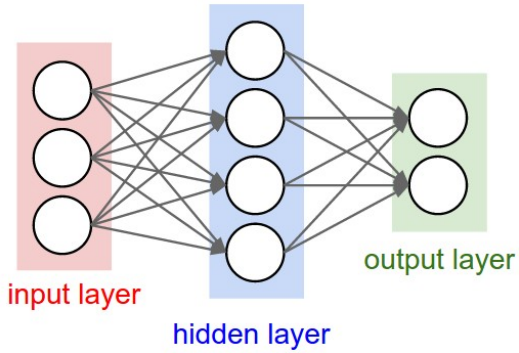
Forward Pass: What does the Network Compute?



- Output of the network can be written as:

$$h_j(\mathbf{x}) = f(b_j^1 + \sum_{i=1}^{D} x_i w_{ji}^1)$$

Forward Pass: What does the Network Compute?



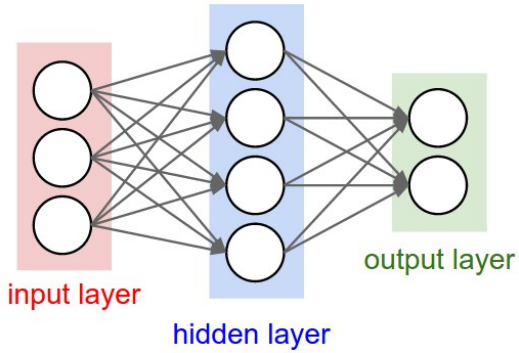
- Output of the network can be written as:

$$h_j(\mathbf{x}) = f(b_j^1 + \sum_{i=1}^{D} x_i w_{ji}^1)$$

$$o_k(\mathbf{x}) = g(b_k^2 + \sum_{j=1}^{J} h_j(\mathbf{x}) w_{kj}^2)$$

(j indexing hidden units, k indexing the output units, D number of inputs)

Forward Pass: What does the Network Compute?



- Output of the network can be written as:

$$h_j(\mathbf{x}) = f(b_j^1 + \sum_{i=1}^{D} x_i w_{ji}^1)$$

$$o_k(\mathbf{x}) = g(b_k^2 + \sum_{j=1}^{J} h_j(\mathbf{x}) w_{kj}^2)$$

(j indexing hidden units, k indexing the output units, D number of inputs)

Training Neural Networks

- Dataset: $\{(\mathbf{x}^{(1)}, \mathbf{t}^{(1)}), (\mathbf{x}^{(2)}, \mathbf{t}^{(2)}), ..., (\mathbf{x}^{(N)}, \mathbf{t}^{(N)})\}$
- Find weights:

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{n=1}^{N} \operatorname{loss}(\mathbf{o}^{(n)}, \mathbf{t}^{(n)})$$

where $\mathbf{o} = f(\mathbf{x}; \mathbf{w})$ is the output of a neural network, $\mathbf{t}$ is ground-truth

Defining a loss function

- Continuous labels: Squared loss: $\sum_k^K \frac{1}{2}(o_k^{(n)} - t_k^{(n)})^2$
  - Next day temperature (celcius)?
  - Price of a house (dollars)?
  - How many oils from olive?
  - Other loss function: L1 loss $\sum_k^K \frac{1}{2}|o_k^{(n)} - t_k^{(n)}|$

Figure: How many oil can you get from these olives?

Defining a loss function

- Discrete labels: Cross-entropy loss: $-\sum_k^K t_k^{(n)} \log o_k^{(n)}$
  - Image category (dog or cat?)
  - Edges (Edge or not?)
  - Note: target label $t_k^{(n)}$ is one-hot.
  - Maximize the log probability of target category

Figure: Image classification: is this a dog or cat?

Defining a loss function

- Discrete labels: Cross-entropy loss: $-\sum_k^K t_k^{(n)} \log o_k^{(n)}$
  - Constraints: $\sum_k^K o_k^{(n)} = 1$, & $0 < o_k^{(n)} < 1$
  - Softmax $o_k^{(n)} = \frac{exp(z_k^{(n)})}{\sum_{j=1}^K exp(z_j^{(n)})}$

Figure: Image classification: is this a dog or cat?

Defining a loss function

- Discrete labels: Cross-entropy loss: $-\sum_k^K t_k^{(n)} \log o_k^{(n)}$
  - Constraints: $\sum_k^K o_k^{(n)} = 1$, & $0 < o_k^{(n)} < 1$
  - Softmax $o_k^{(n)} = \frac{exp(z_k^{(n)})}{\sum_{j=1}^K exp(z_j^{(n)})}$

Figure: Image classification: is this a dog or cat?

Training Neural Networks

- Find weights:

$$\mathbf{w}^* = \underset{\mathbf{w}}{\text{argmin}} \sum_{n=1}^{N} \text{loss}(\mathbf{o}^{(n)}, \mathbf{t}^{(n)})$$

  where $\mathbf{o} = f(\mathbf{x}; \mathbf{w})$ is the output of a neural network, $\mathbf{t}$ is ground-truth

- Define the loss function: discrete labels, continuous labels.

Training Neural Networks

- Find weights:

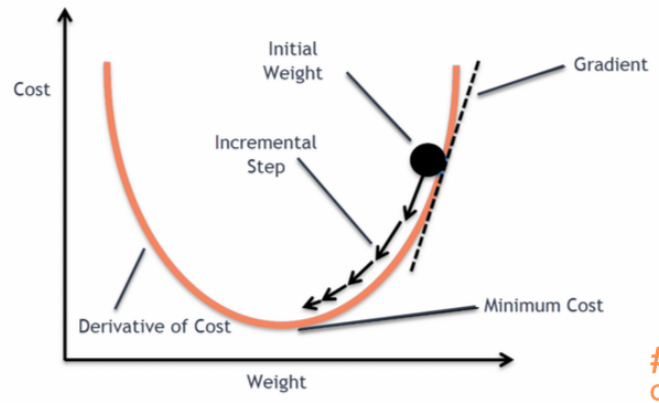
$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{n=1}^{N} \operatorname{loss}(\mathbf{o}^{(n)}, \mathbf{t}^{(n)})$$

  where $\mathbf{o} = f(\mathbf{x}; \mathbf{w})$ is the output of a neural network, $\mathbf{t}$ is ground-truth

- Gradient Descent!
  - Start: Random initialization: $\mathbf{w}^0$
  - Every iteration:

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \frac{\partial L}{\partial \mathbf{w}^t}$$

  where η is the learning rate (and L is error/loss)



Wait! How to compute the gradient for NN?

- Find weights:

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{n=1}^{N} \operatorname{loss}(\mathbf{o}^{(n)}, \mathbf{t}^{(n)})$$

where $\mathbf{o} = f(\mathbf{x}; \mathbf{w})$ is the output of a neural network, $\mathbf{t}$ is ground-truth

- Gradient Descent!

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \frac{\partial L}{\partial \mathbf{w}^t}$$

where η is the learning rate (and L is error/loss)

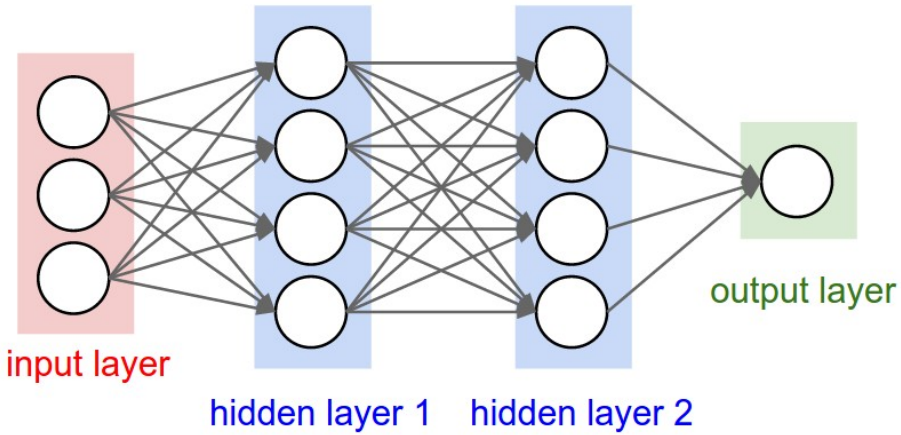
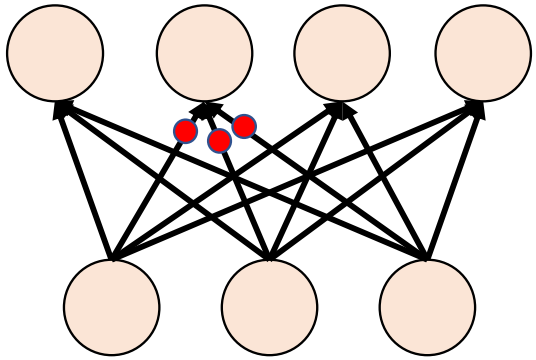


Figure: Neural Network

Wait! How to compute the gradient for NN?

- Let's start with one-layer neural network with sigmoid activation & squared loss



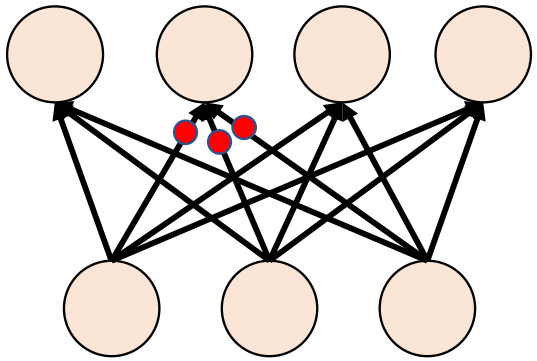
Wait! How to compute the gradient for NN?

- Let's start with one-layer neural network with sigmoid activation & squared loss

$$\mathbf{z}^{(n)} = \mathbf{w}\mathbf{x}^{(\mathbf{n})}, \qquad \mathbf{o}^{(n)} = \text{sigmoid}(\mathbf{z}^{(n)}) = \frac{1}{1 + \exp(-\mathbf{z}^{(\mathbf{n})})}$$

$$L = \sum_{n=1}^{N} \frac{1}{2}(\mathbf{o}^{(n)} - \mathbf{t}^{(n)})^2, \qquad \mathbf{w}^* = \underset{\mathbf{w}}{\text{argmin}}\, L$$

where $\mathbf{o} = f(\mathbf{x}; \mathbf{w})$ is the output of a neural network, $\mathbf{t}$ is ground-truth



- How to compute $\frac{\partial L}{\partial \mathbf{w}}$?

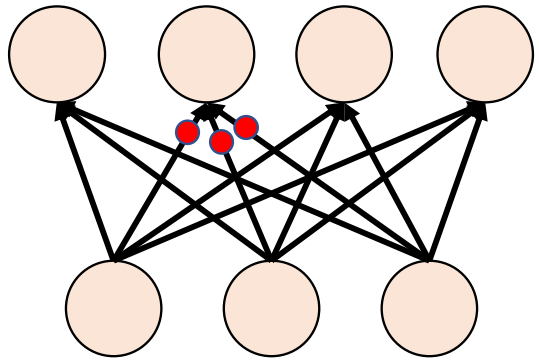
Wait! How to compute the gradient for NN?

- Let's start with one-layer neural network with sigmoid activation & squared loss

$$\mathbf{z}^{(n)} = \mathbf{w}\mathbf{x}^{(\mathbf{n})}, \qquad \mathbf{o}^{(n)} = \text{sigmoid}(\mathbf{z}^{(n)}) = \frac{1}{1 + \exp(-\mathbf{z}^{(\mathbf{n})})}$$

$$L = \sum_{n=1}^{N} \frac{1}{2}(\mathbf{o}^{(n)} - \mathbf{t}^{(n)})^2, \qquad \mathbf{w}^* = \underset{\mathbf{w}}{\text{argmin}}\, L$$

where $\mathbf{o} = f(\mathbf{x}; \mathbf{w})$ is the output of a neural network, $\mathbf{t}$ is ground-truth



- How to compute $\frac{\partial L}{\partial \mathbf{w}}$?

Wait! How to compute the gradient for NN?

- Let's start with one-layer neural network with sigmoid activation & squared loss

$$\mathbf{z}^{(n)} = \mathbf{w}\mathbf{x}^{(\mathbf{n})}, \qquad \mathbf{o}^{(n)} = \text{sigmoid}(\mathbf{z}^{(n)}) = \frac{1}{1 + \exp(-\mathbf{z}^{(\mathbf{n})})}$$

$$L = \sum_{n=1}^{N} \frac{1}{2}(\mathbf{o}^{(n)} - \mathbf{t}^{(n)})^2, \qquad \mathbf{w}^* = \underset{\mathbf{w}}{\text{argmin}}\, L$$

- How to compute $\frac{\partial L}{\partial \mathbf{w}}$?
- This is a composition function! Let's use chain rule!

$$\frac{\partial L}{\partial \mathbf{w}} = \sum_{n=1}^{N} \frac{\partial L}{\partial \mathbf{o}^{(n)}} * \frac{\partial \mathbf{o}^{(n)}}{\partial \mathbf{w}}$$

$$\frac{\partial L}{\partial \mathbf{o}^{(n)}} = (\mathbf{o}^{(n)} - \mathbf{t}^{(n)})$$

Wait! How to compute the gradient for NN?

- Let's start with one-layer neural network with sigmoid activation & squared loss

$$\mathbf{z}^{(n)} = \mathbf{w}\mathbf{x}^{(\mathbf{n})}, \qquad \mathbf{o}^{(n)} = \text{sigmoid}(\mathbf{z}^{(n)}) = \frac{1}{1+\exp(-\mathbf{z}^{(\mathbf{n})})}$$

$$L = \sum_{n=1}^{N} \frac{1}{2}(\mathbf{o}^{(n)} - \mathbf{t}^{(n)})^2, \qquad \mathbf{w}^* = \underset{\mathbf{w}}{\text{argmin}}\, L$$

- How to compute $\frac{\partial L}{\partial \mathbf{w}}$?
- This is a composition function! Let's use chain rule! (Backprogapation)

$$\frac{\partial L}{\partial \mathbf{w}} = \sum_{n=1}^{N} \frac{\partial L}{\partial \mathbf{o}^{(n)}} * \frac{\partial \mathbf{o}^{(n)}}{\partial \mathbf{w}} = \sum_{n=1}^{N} \frac{\partial L}{\partial \mathbf{o}^{(n)}} * \frac{\partial \mathbf{o}^{(n)}}{\partial \mathbf{z}^{(n)}} * \frac{\partial \mathbf{z}^{(n)}}{\partial \mathbf{w}}$$

$$\frac{\partial L}{\partial \mathbf{o}^{(n)}} = (\mathbf{o}^{(n)} - \mathbf{t}^{(n)})$$

Wait! How to compute the gradient for NN?

- Let's try with two-layer neural network with squared loss

$$\mathbf{z}^{(n)} = \mathbf{w_2} * f(\mathbf{w_1}\mathbf{x}^{(n)}), \qquad \mathbf{o}^{(n)} = f(\mathbf{z}^{(n)})$$

$$L = \sum_{n=1}^{N} \frac{1}{2}(\mathbf{o}^{(n)} - \mathbf{t}^{(n)})^2, \qquad \mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}}\, L$$

- How to compute $\frac{\partial L}{\partial \mathbf{w_2}}$?
- This is a composition function! Let's use chain rule!

$$\frac{\partial L}{\partial \mathbf{w_2}} = \sum_{n=1}^{N} \frac{\partial L}{\partial \mathbf{o}^{(n)}} * \frac{\partial \mathbf{o}^{(n)}}{\partial \mathbf{w_2}} = \sum_{n=1}^{N} \frac{\partial L}{\partial \mathbf{o}^{(n)}} * \frac{\partial \mathbf{o}^{(n)}}{\partial \mathbf{z}^{(n)}} * \frac{\partial \mathbf{z}^{(n)}}{\partial \mathbf{w_2}}$$

Wait! How to compute the gradient for NN?

- Let's try with two-layer neural network with squared loss

$$\mathbf{z}^{(n)} = \mathbf{w_2} * f(\mathbf{w_1}\mathbf{x}^{(n)}), \qquad \mathbf{o}^{(n)} = f(\mathbf{z}^{(n)})$$

$$L = \sum_{n=1}^{N} \frac{1}{2}(\mathbf{o}^{(n)} - \mathbf{t}^{(n)})^2, \qquad \mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}}\, L$$

- How to compute $\frac{\partial L}{\partial \mathbf{w_2}}$?
- This is a composition function! Let's use chain rule!

$$\frac{\partial L}{\partial \mathbf{w_2}} = \sum_{n=1}^{N} \frac{\partial L}{\partial \mathbf{o}^{(n)}} * \frac{\partial \mathbf{o}^{(n)}}{\partial \mathbf{w_2}} = \sum_{n=1}^{N} \frac{\partial L}{\partial \mathbf{o}^{(n)}} * \frac{\partial \mathbf{o}^{(n)}}{\partial \mathbf{z}^{(n)}} * \frac{\partial \mathbf{z}^{(n)}}{\partial \mathbf{w_2}}$$

How to train the neural network efficiently?

- It's hard to put all the data into memory and get gradient :(
- Stochatic Gradient Descent (SGD): random sample a batch of data at every iteration

```
for epoch in range(100):# Each epoch means running through the whole dataset
    for data in dataloader:
        x, t = data # Randomly fetch a batch of data (e.g. 32 data points)
        output = nn.forward(x) # Forward through the network to get output
        loss = np.sum((output - t) ** 2) # Calculate loss function
        loss.backward() # Backpropagation to get the gradients
        optimizer.step() # Gradient descent
```

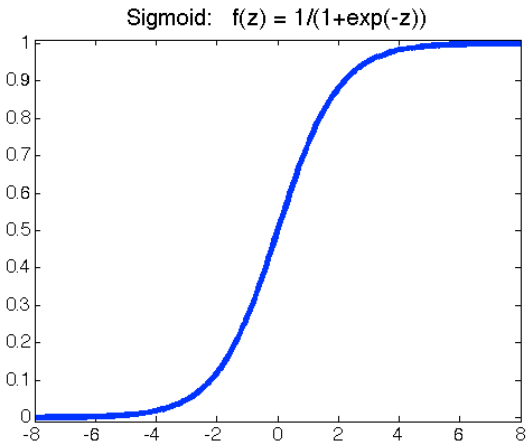
- Other optimizers: Adam, Adagrad, RMSprop

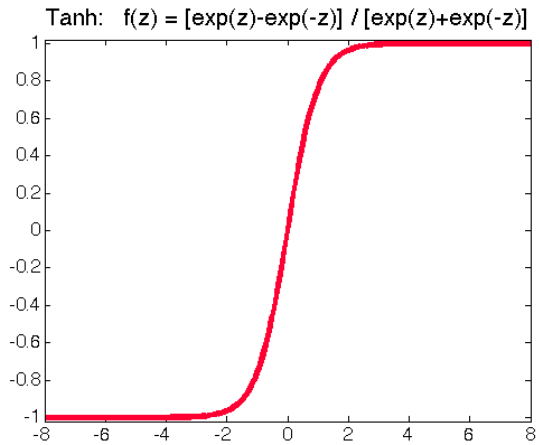
Train one neural network in Python

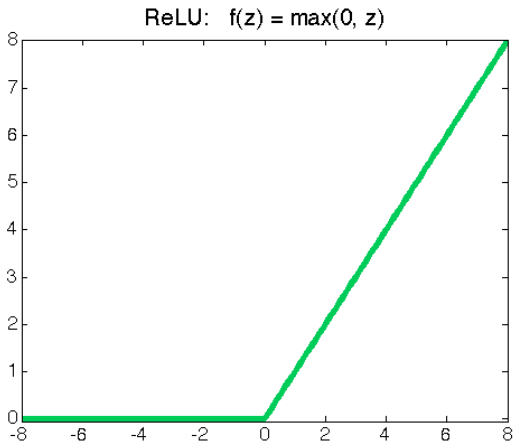
```
f = lambda x: 1.0 / (1.0 + np.exp(-x)) # Sigmod activation function
Input_dim, Hidden_dim, Output_dim = 3, 4, 2
class TwoLayerMLP(object):
    def init():
        self.W1 = np.randn(Hidden_dim, Input_dim) # Random init weight for the 1st laye
        self.b1 = np.zeros(Hidden_dim, 1) # Bias for the 1st layer
        self.W2 = np.randn(Output_dim, Hidden_dim) # Random init weight for the 2nd lay
        self.b2 = np.zeros(Output_dim, 1) # Bias for the 2nd layer
    def forward(x):
        # Input data x: Input_dim x 1
        h = f(np.dot(self.W1, x) + self.b1) # sigmoid(W1 * x + b1)
        output = f(np.dot(self.W2, h) + self.b2) # Sigmoid (W2 * h + b2)
        return output

nn = TwoLayerMLP()
for epoch in range(100): # Each epoch means running through the whole dataset
    for data in dataloader:
        x, t = data # Fetch a batch of data from your dataset
        output = nn.forward(x) # Forward through the network to get output
        loss = np.sum((output - t) ** 2) # Calculate loss function
        loss.backward() # Backpropagation to get the gradients
        optimizer.step() # Gradient descent
```

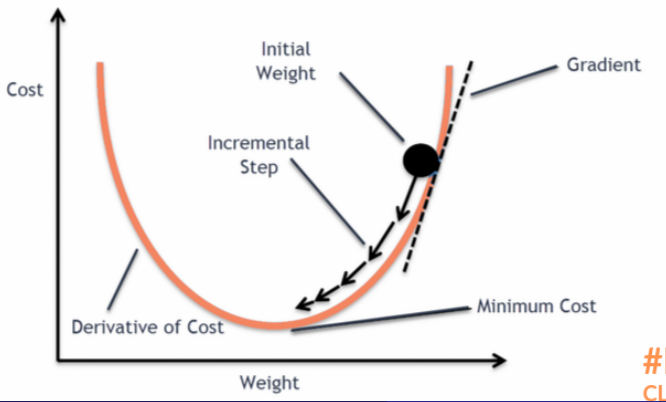
Some advices on training neural network

- Initialization matters (be careful about the scale → Normalizations)
- Avoid gradient vanishing problem
- Be careful about the learning rates









#MLmuse
CLAIRVOYANT

Convolutional Neural Networks (CNN)

- To work with images we typically use NN with special architecture
- Any specical properties we can think of for an image?

Convolutional Neural Networks (CNN)

- To work with images we typically use NN with special architecture
- Any specical properties we can think of for an image?
- Depending on the tasks, we prefer different properties.
- Translation Equivariance :
  - If I translate the input, the output will be translated by the same amount.
  - E.g. Edge detection

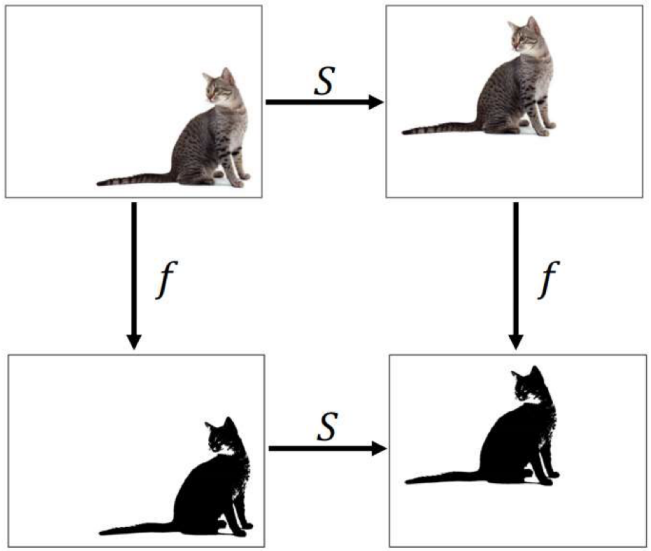


Figure: $\mathcal{S}(f(I)) = f(\mathcal{S}(I))$

Convolutional Neural Networks (CNN)

- To work with images we typically use NN with special architecture
- Any specical properties we can think of for an image?
- Depending on the tasks, we prefer different properties.
- Translation Equivariance
- Translation Invariance :
  - If I translate the input, the output will be the same.
  - E.g. Image classification

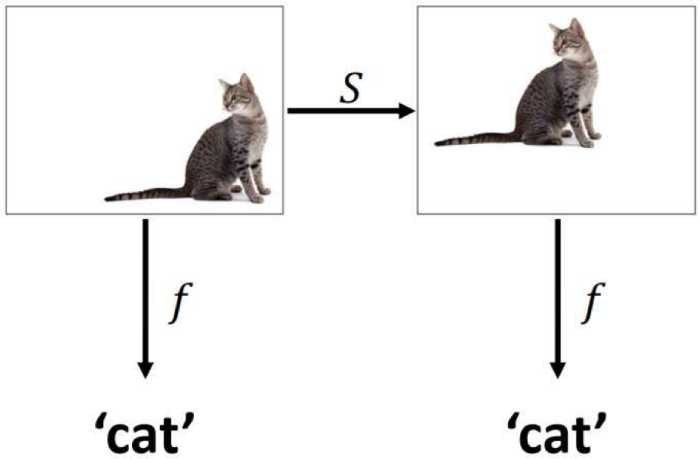


Figure: $f(I) = f(\mathcal{S}(I))$

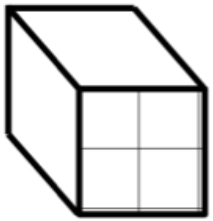
[Adapted from Bernhard Kainz]

Convolutional Neural Networks (CNN)

- Remember our Lecture 2 about filtering?

Input “image”

Filter

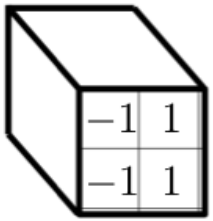


Convolutional Neural Networks (CNN)

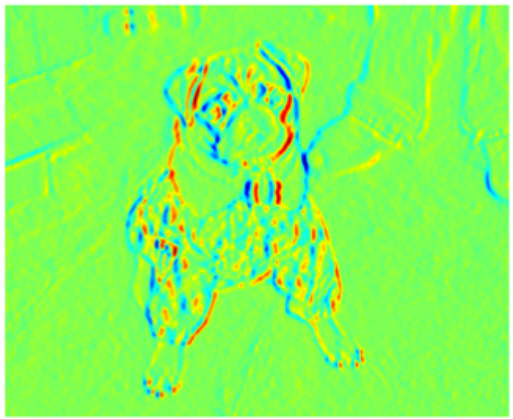
- If our filter was $[-1, 1]$, we got a vertical edge detector

Input "image"

Filter

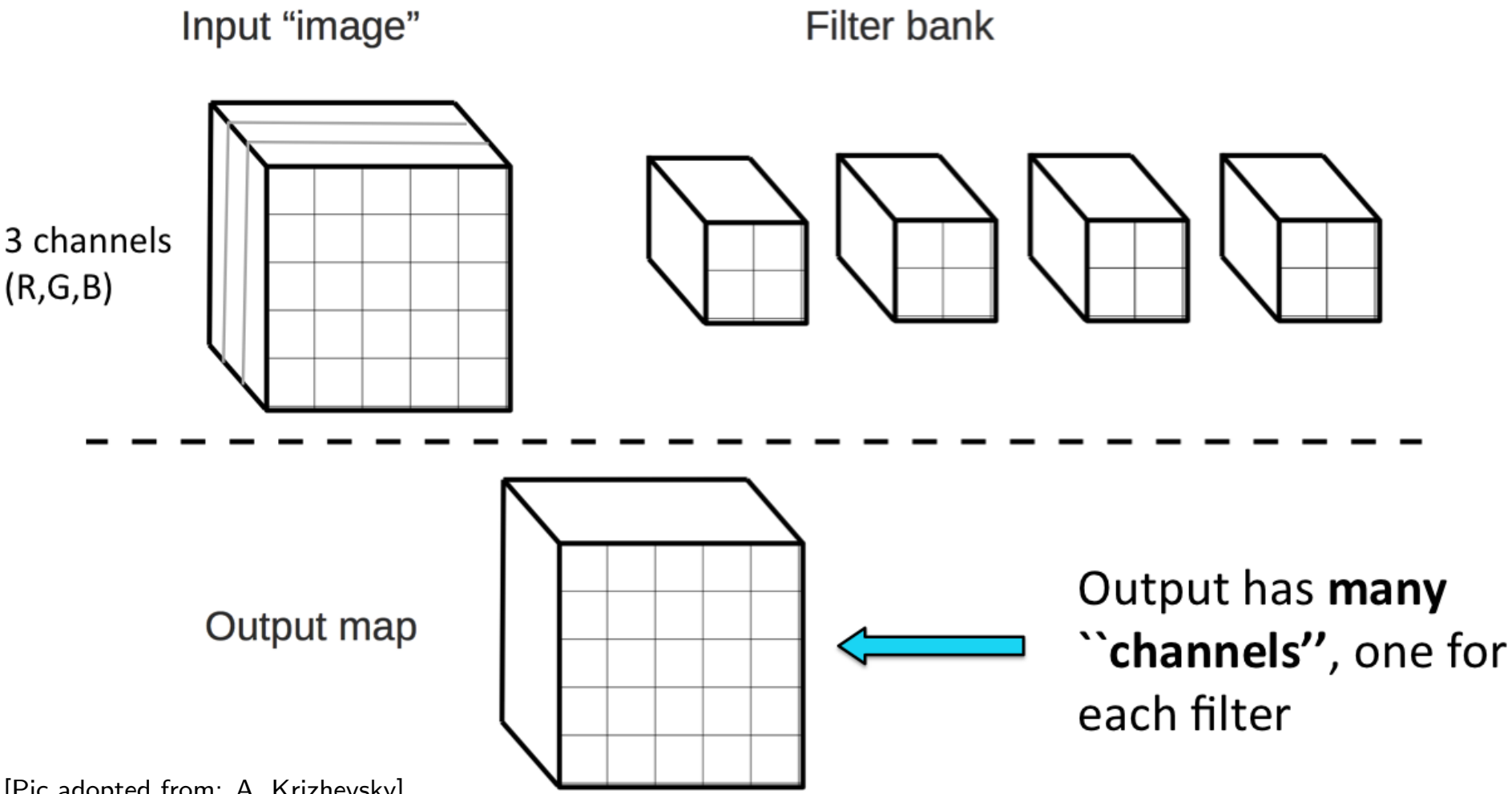


Output map



Convolutional Neural Networks (CNN)

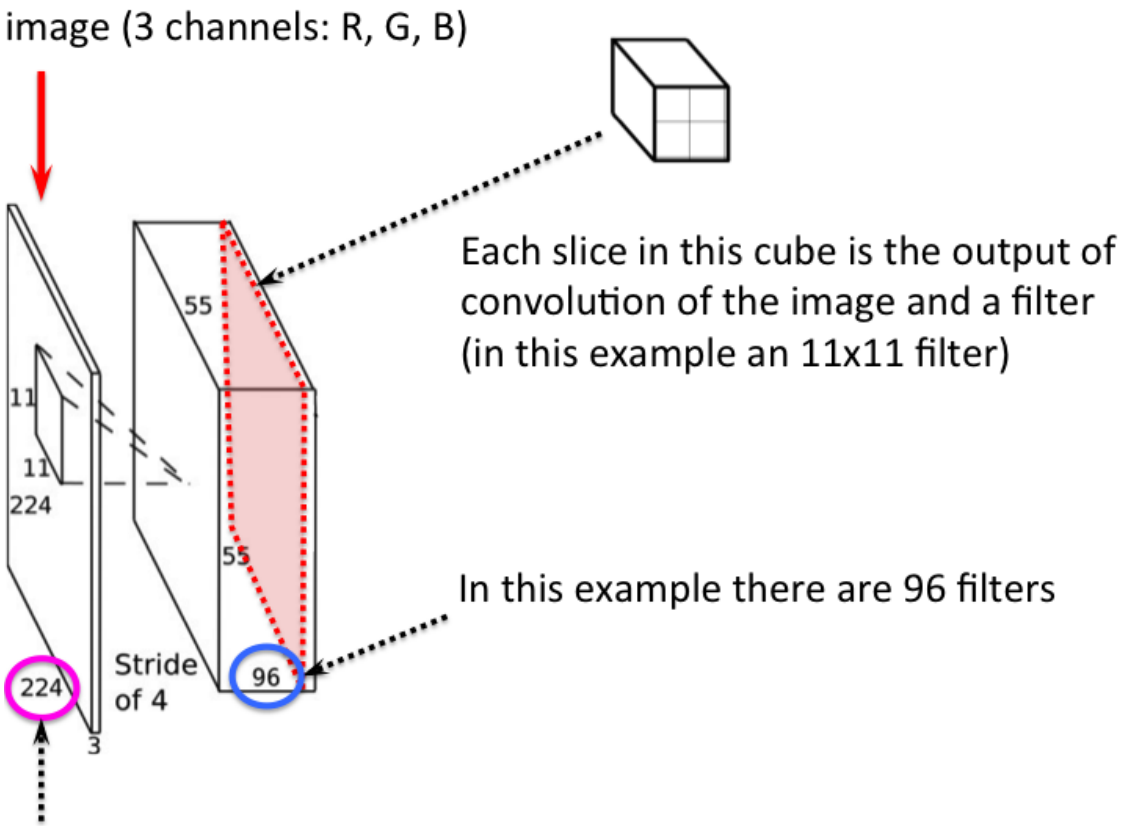
- Now imagine we didn't only want a vertical edge detector, but also a horizontal one, and one for corners, one for dots, etc. We would need to take many filters. A **filterbank**.



[Pic adopted from: A. Krizhevsky]

Convolutional Neural Networks (CNN)

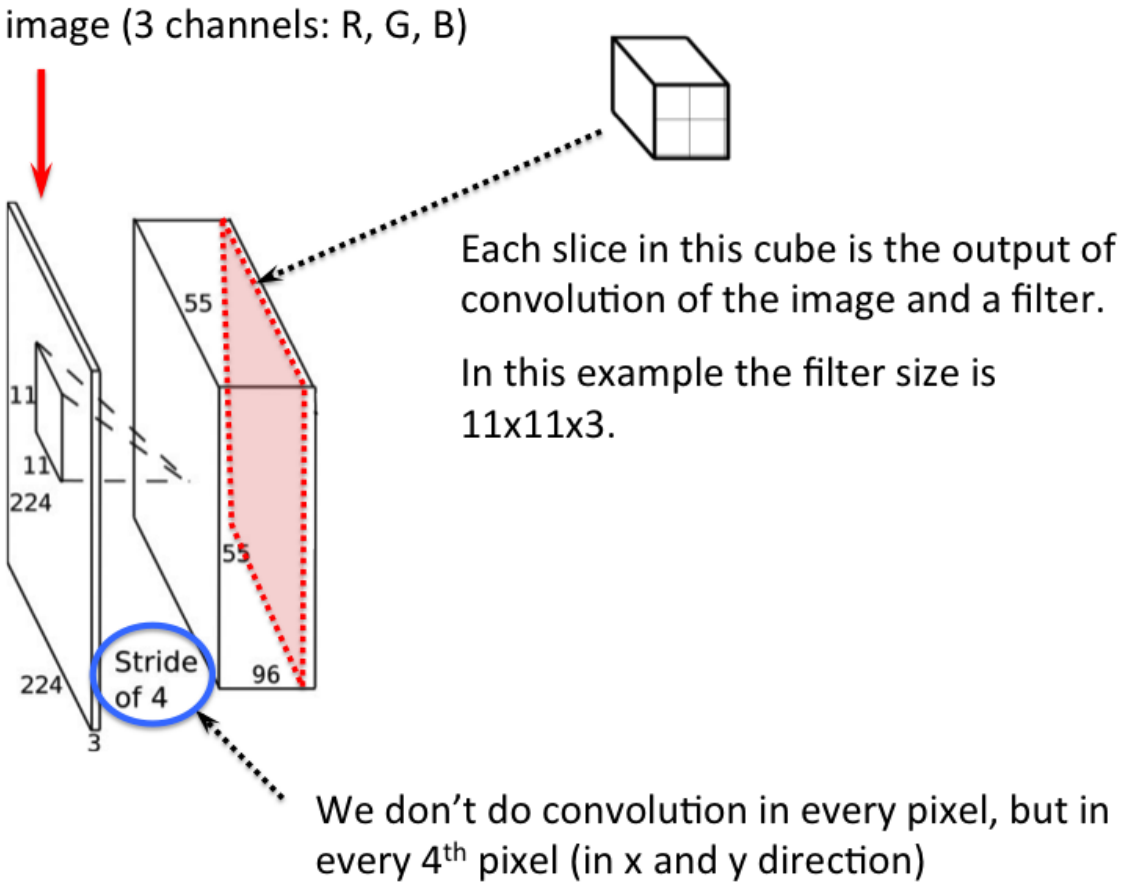
- Applying a filterbank to an image.
- Instead of using predefined convolution kernel, we use the data to learn it!



[Pic adopted from: A. Krizhevsky]

Convolutional Neural Networks (CNN)

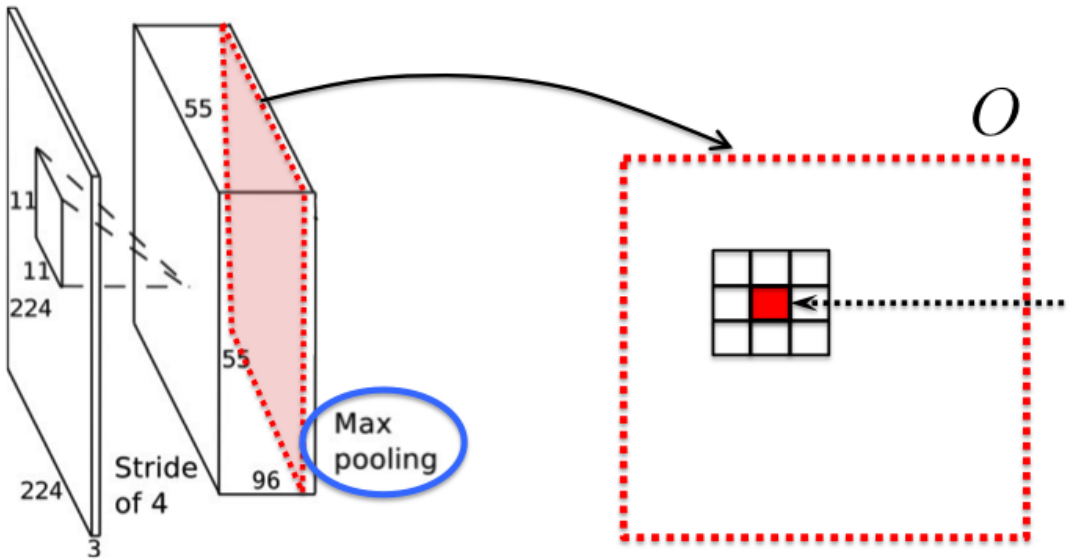
- Instead of using predefined convolution kernel, we use the data to learn it.
- Convolution vs. MLP?



[Pic adopted from: A. Krizhevsky]

Convolutional Neural Networks (CNN)

- Do some additional tricks. A popular one is called **max pooling**: Only care about the maximum within certain region.
- Any idea why you would do this? (Why didn't do convlution with stride?)



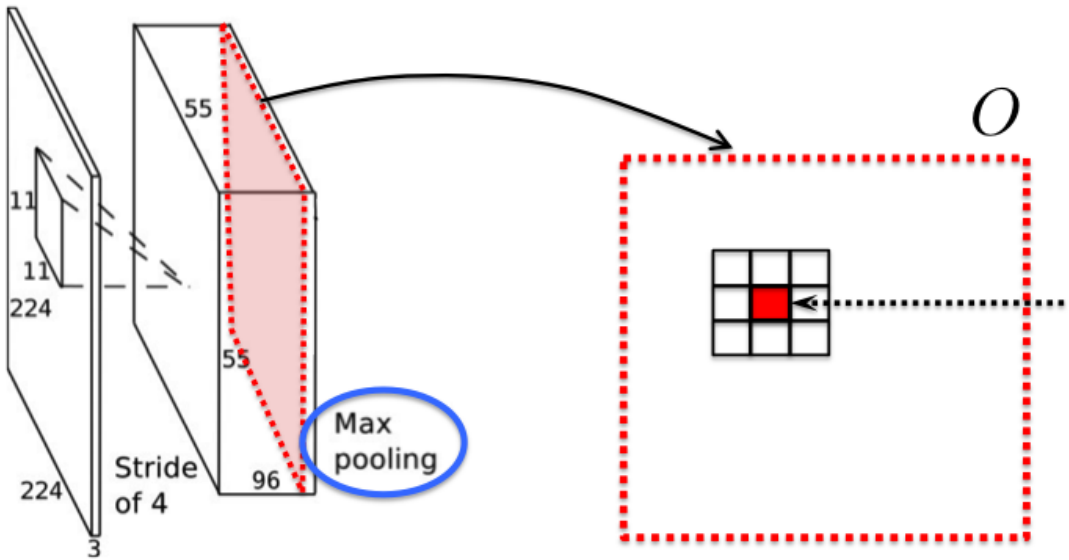
$$O(i,j) = \max_{\substack{k\in\{i-1,i,i+1\}\\ l\in\{j-1,j,j+1\}}} O(k,l)$$

Take each slice in the output cube, and in each pixel compute a max over a small patch around it. This is called **max pooling**.

[Pic adopted from: A. Krizhevsky]

Convolutional Neural Networks (CNN)

- Do some additional tricks. A popular one is called **max pooling**.
- Any idea why you would do this? To get **invariance to small shifts in position**.



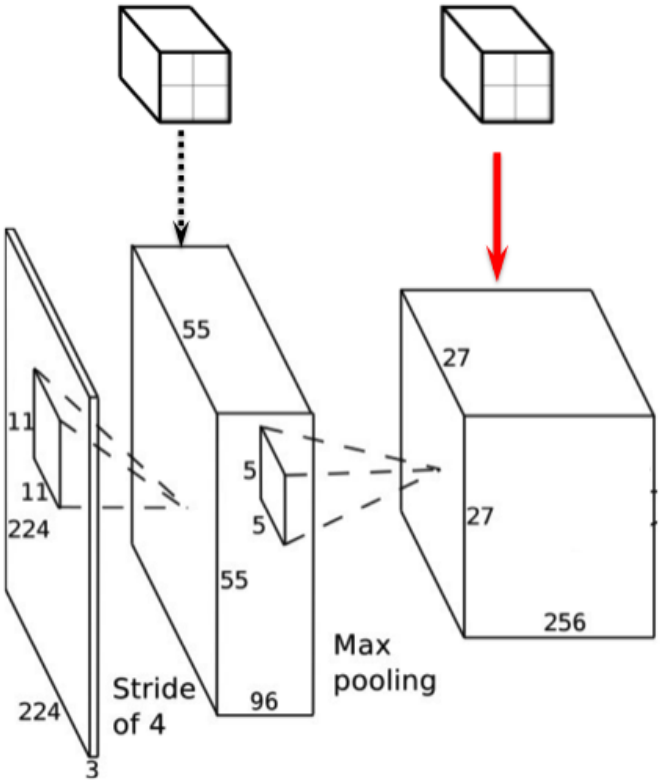
$$O(i,j) = \max_{\substack{k \in \{i-1,i,i+1\} \\ l \in \{j-1,j,j+1\}}} O(k,l)$$

Take each slice in the output cube, and in each pixel compute a max over a small patch around it. This is called **max pooling**.

[Pic adopted from: A. Krizhevsky]

Convolutional Neural Networks (CNN)

- Now add another "layer" of filters. For each filter again do convolution, but this time with the output cube of the previous layer.



Add one more layer of filters

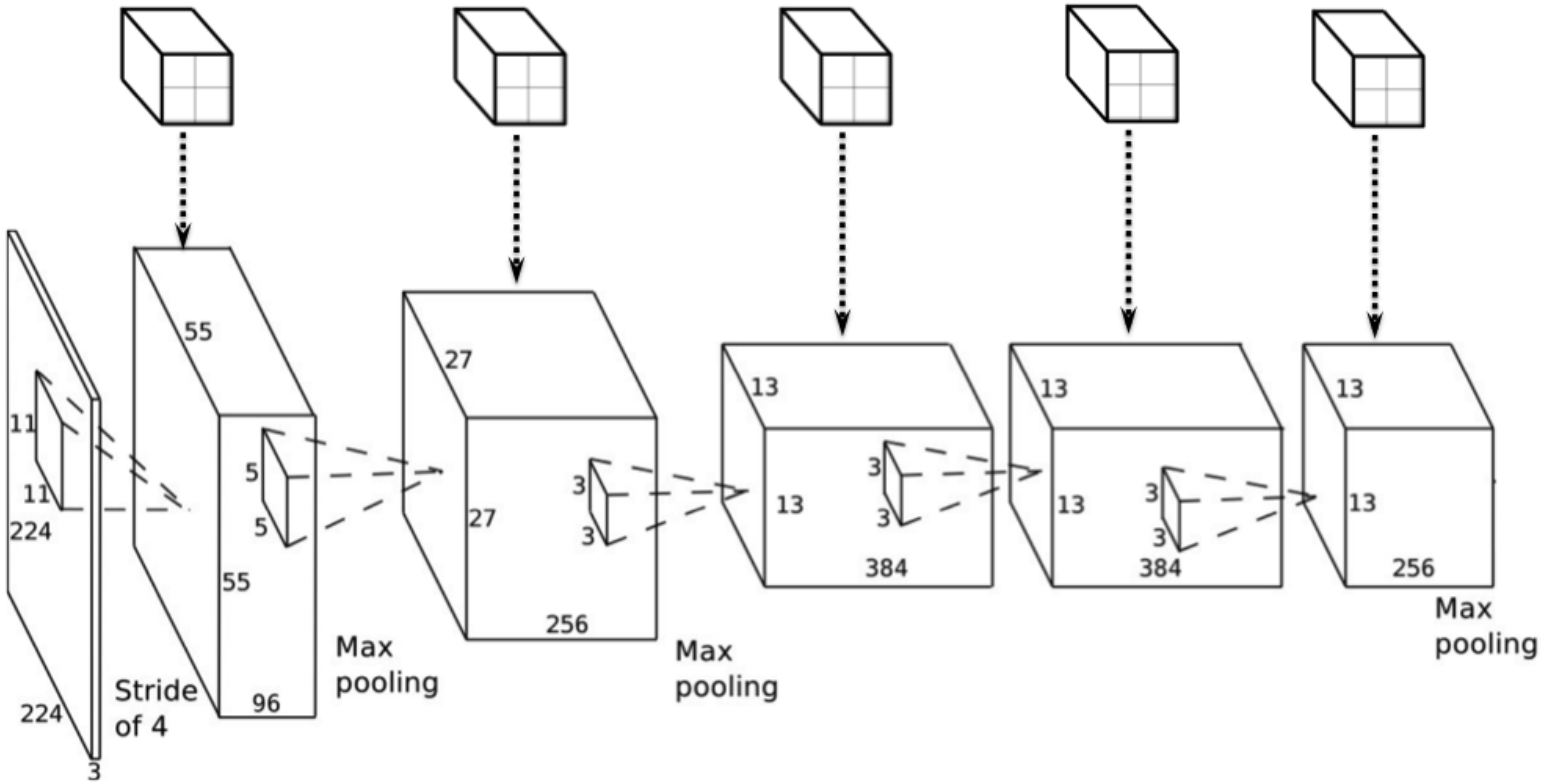
These filters are convolved with the output of the previous layer. The results of each convolution is again a slice in the cube on the right.

What is the dimension of each of these filters?

[Pic adopted from: A. Krizhevsky]

Convolutional Neural Networks (CNN)

- Keep adding a few layers. Any idea what's the purpose of more layers? Why can't we just have a full bunch of filters in one layer?



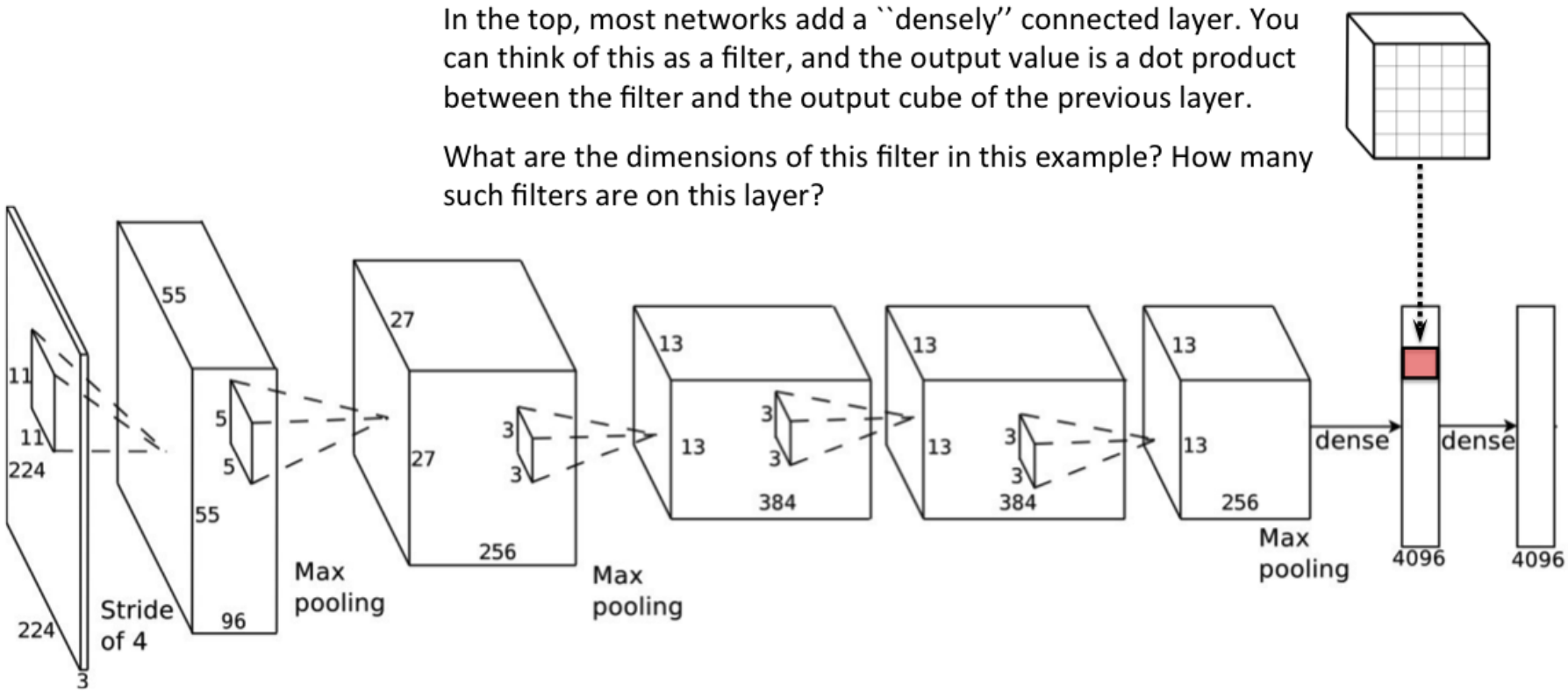
Do it recursively

Have multiple ``layers''

[Pic adopted from: A. Krizhevsky]

Convolutional Neural Networks (CNN)

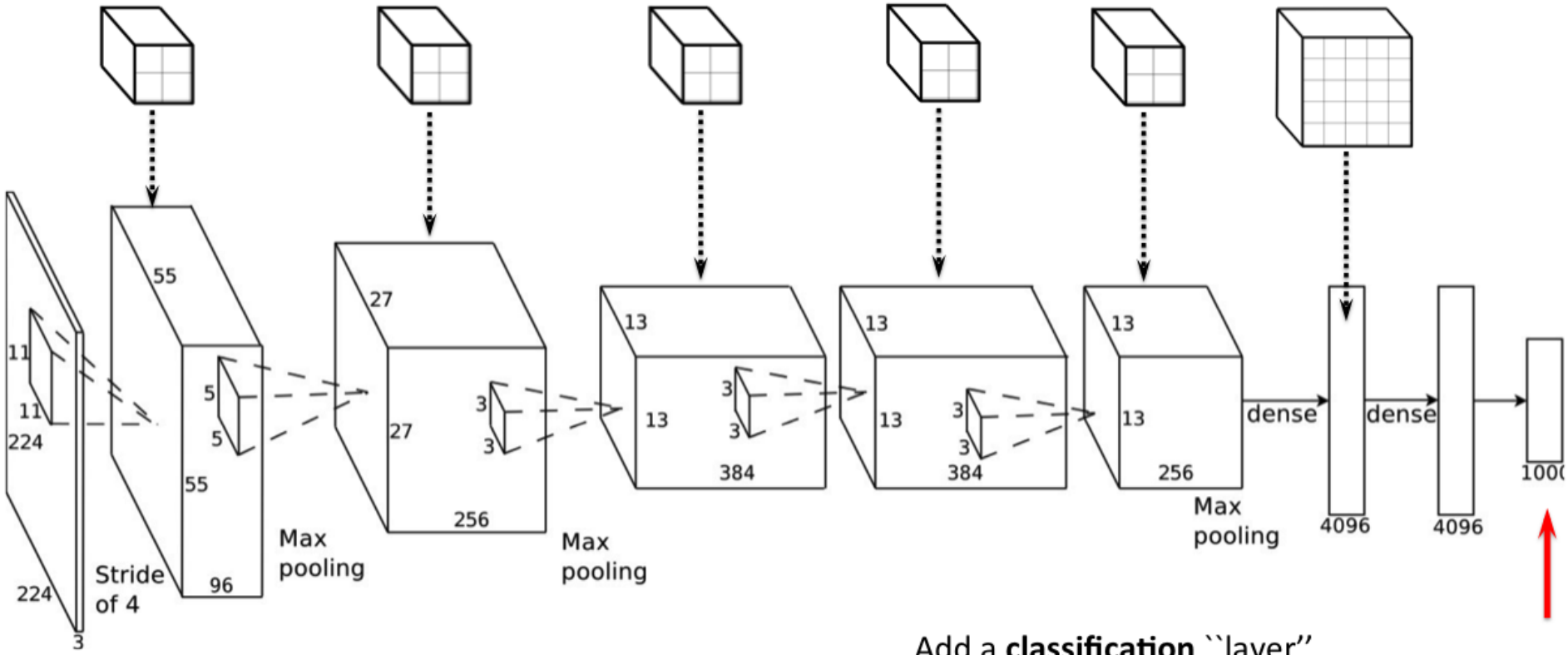
- In the end add one or two **fully** (or **densely**) connected layers. In this layer, we don't do convolution we just do a dot-product between the "filter" and the output of the previous layer.



[Pic adopted from: A. Krizhevsky]

Convolutional Neural Networks (CNN)

- Add one final layer: a **classification** layer. Each dimension of this vector tells us the probability of the input image being of a certain class.



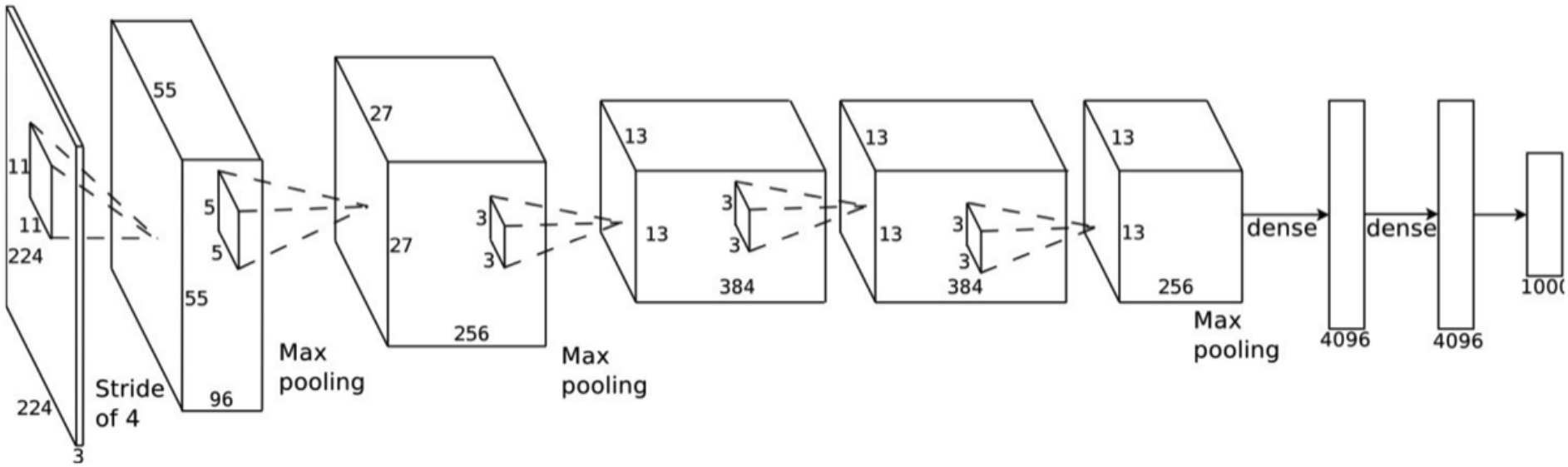
Add a **classification** ``layer''.

For an input image, the value in a particular dimension of this vector tells you the probability of the corresponding object class.

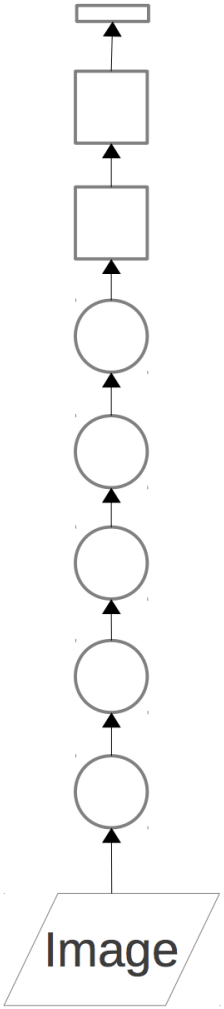
[Pic adopted from: A. Krizhevsky]

Convolutional Neural Networks (CNN)

- This fully specifies a network. The one below has been a popular choice in the fast few years. It was proposed by UofT guys: A. Krizhevsky, I. Sutskever, G. E. Hinton, *ImageNet Classification with Deep Convolutional Neural Networks*, NIPS 2012. This network won the Imagenet Challenge of 2012, and revolutionized computer vision.
- How many parameters (weights) does this network have?



Convolutional Neural Networks (CNN)



- Trained with stochastic gradient descent on two NVIDIA GPUs for about a week
- 650,000 neurons
- 60,000,000 parameters
- 630,000,000 connections
- **Final feature layer:** 4096-dimensional

Convolutional layer: convolves its input with a bank of 3D filters, then applies point-wise non-linearity

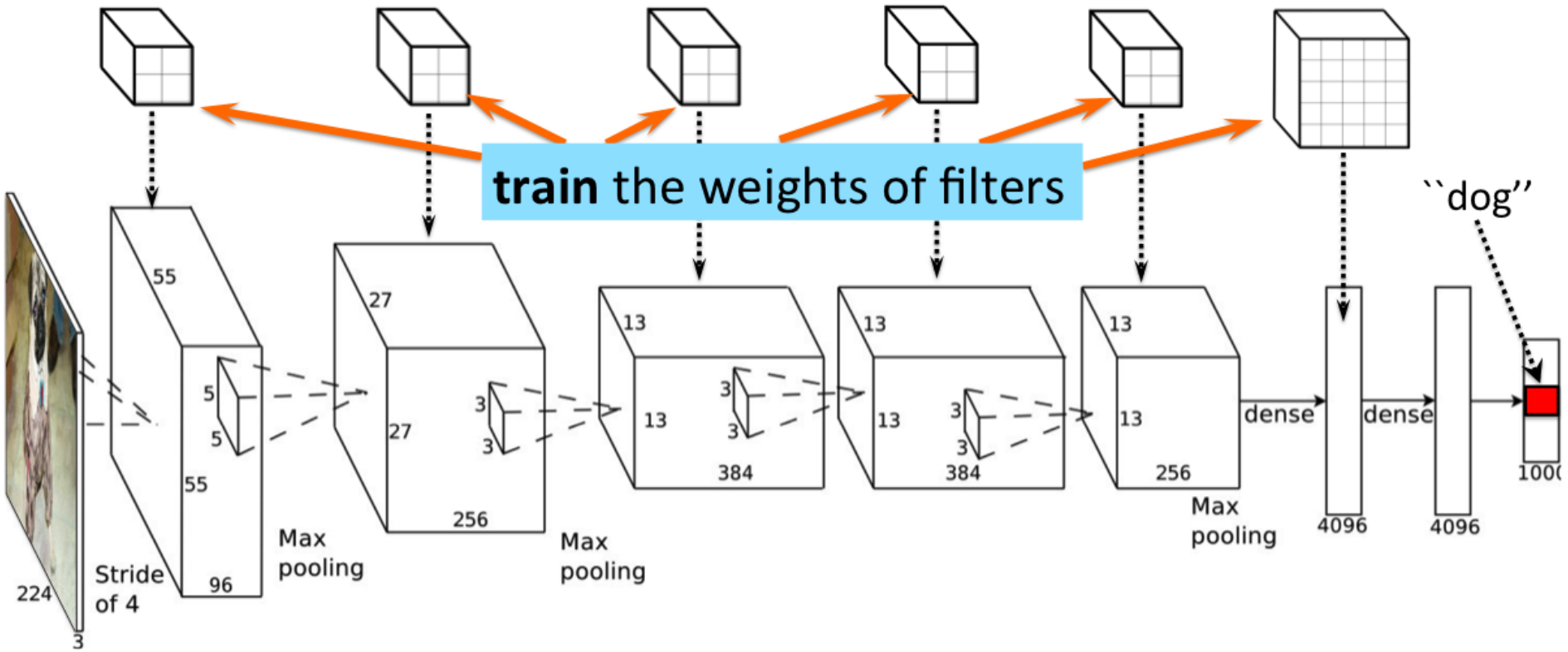
Fully-connected layer: applies linear filters to its input, then applies point-wise non-linearity

Figure: From: `http://www.image-net.org/challenges/LSVRC/2012/supervision.pdf`

[Pic adopted from: A. Krizhevsky]

Convolutional Neural Networks (CNN)

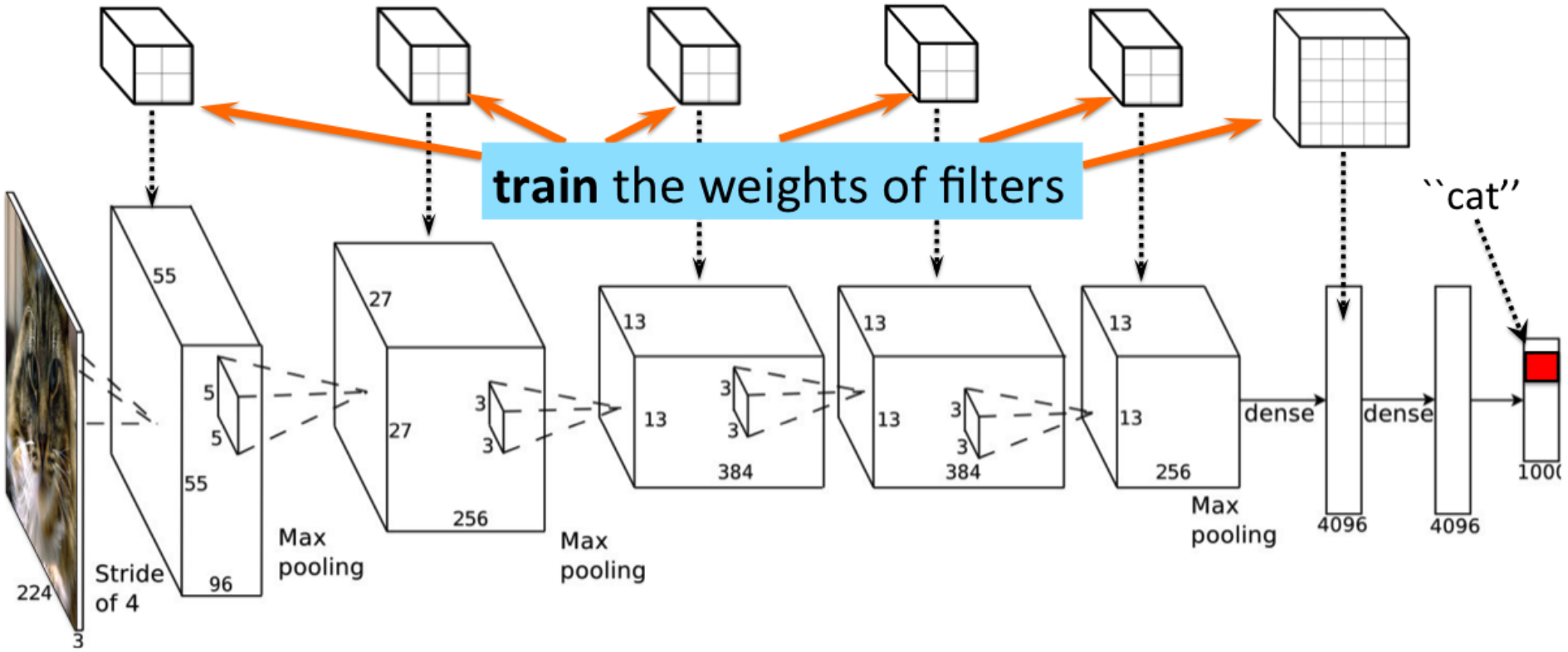
- The trick is to not hand-fix the weights, but to **train** them. Train them such that when the network sees a picture of a dog, the last layer will say "dog".



[Pic adopted from: A. Krizhevsky]

Convolutional Neural Networks (CNN)

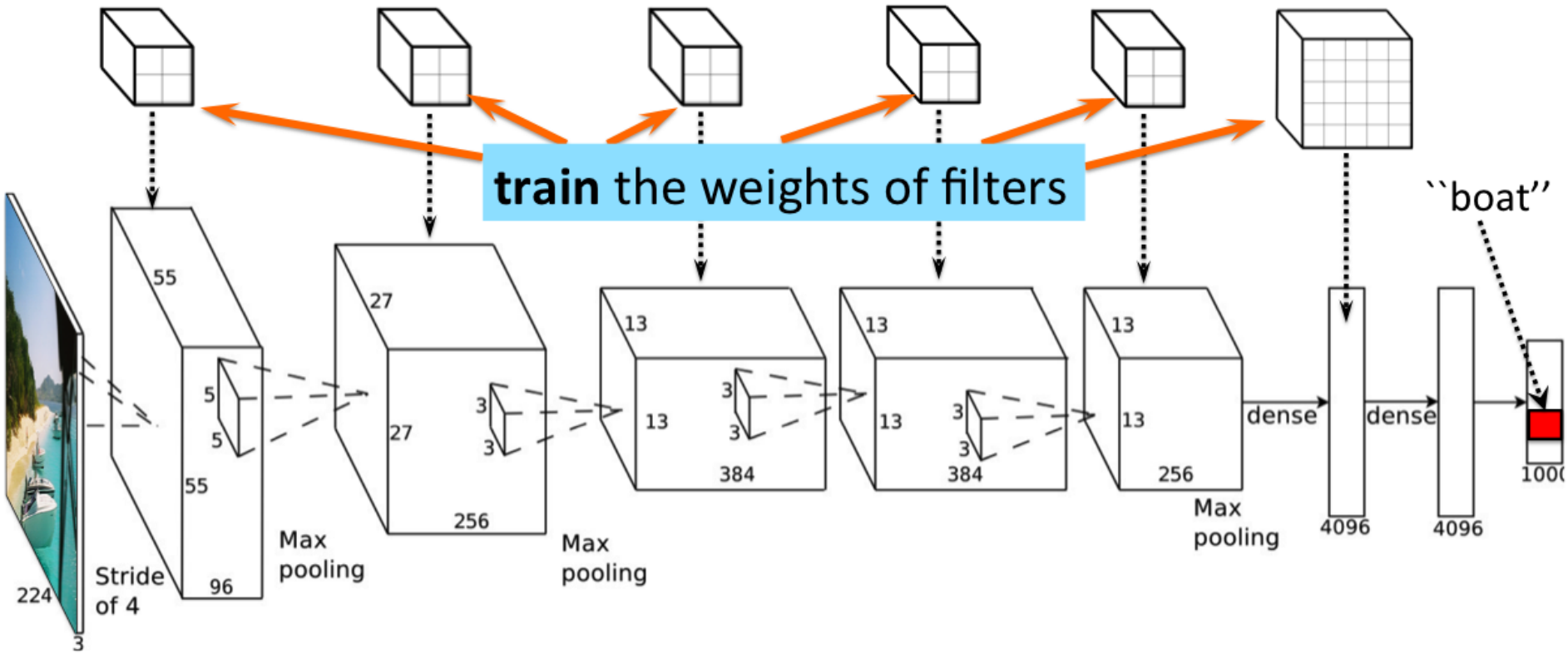
- Or when the network sees a picture of a cat, the last layer will say "cat".



[Pic adopted from: A. Krizhevsky]

Convolutional Neural Networks (CNN)

- Or when the network sees a picture of a boat, the last layer will say "boat"... The more pictures the network sees, the better.



Train on **lots** of examples. Millions. Tens of millions. Wait a week for training to finish.

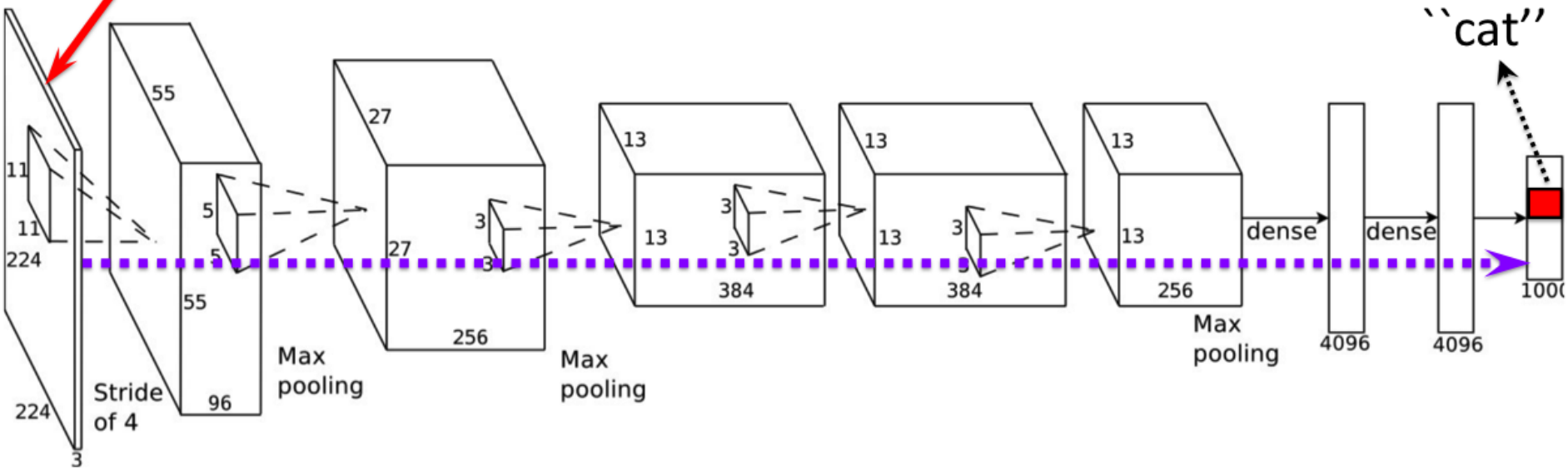
Share your network (the weights) with others who are not fortunate enough with GPU power.

[Pic adopted from: A. Krizhevsky]

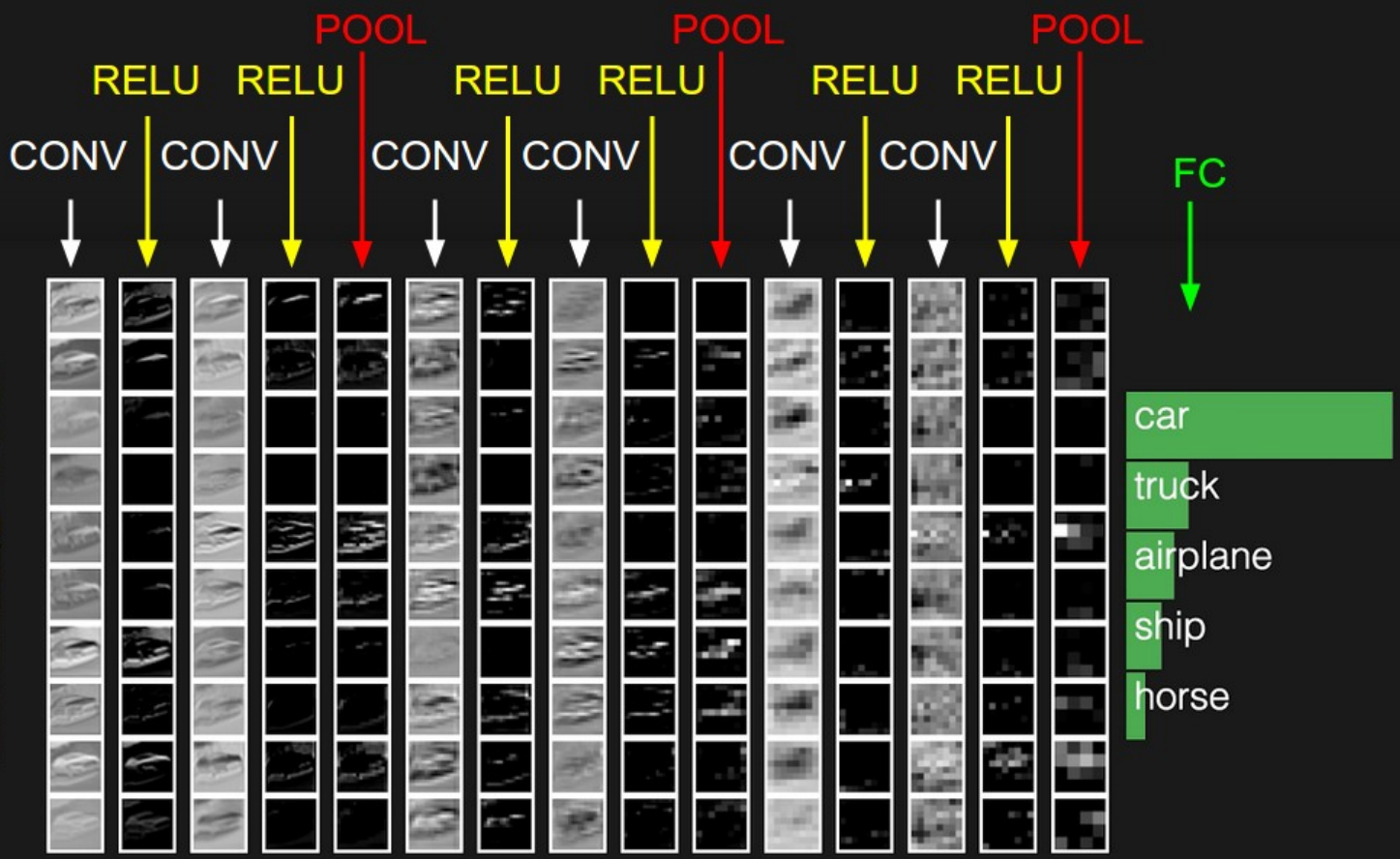
Classification

- Once trained we can do classification. Just feed in an image or a crop of the image, run through the network, and read out the class with the highest probability in the last (classification) layer.

What's the class of this object?



Example



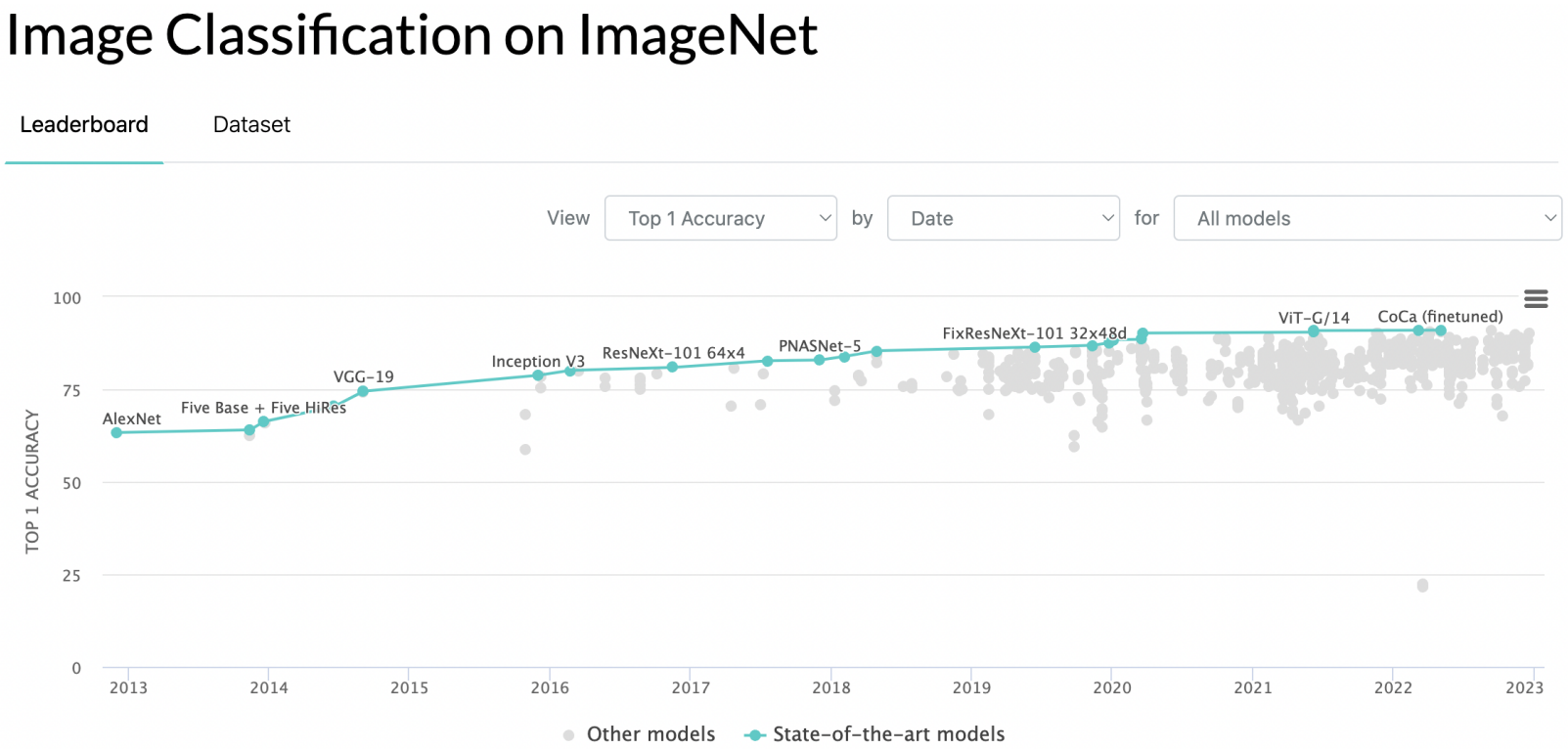
[http://cs231n.github.io/convolutional-networks/]

Classification Performance

- Imagenet, main challenge for object classification: `http://image-net.org/`
- 1000 classes, 1.2M training images, 150K for test

Classification Performance in 2023

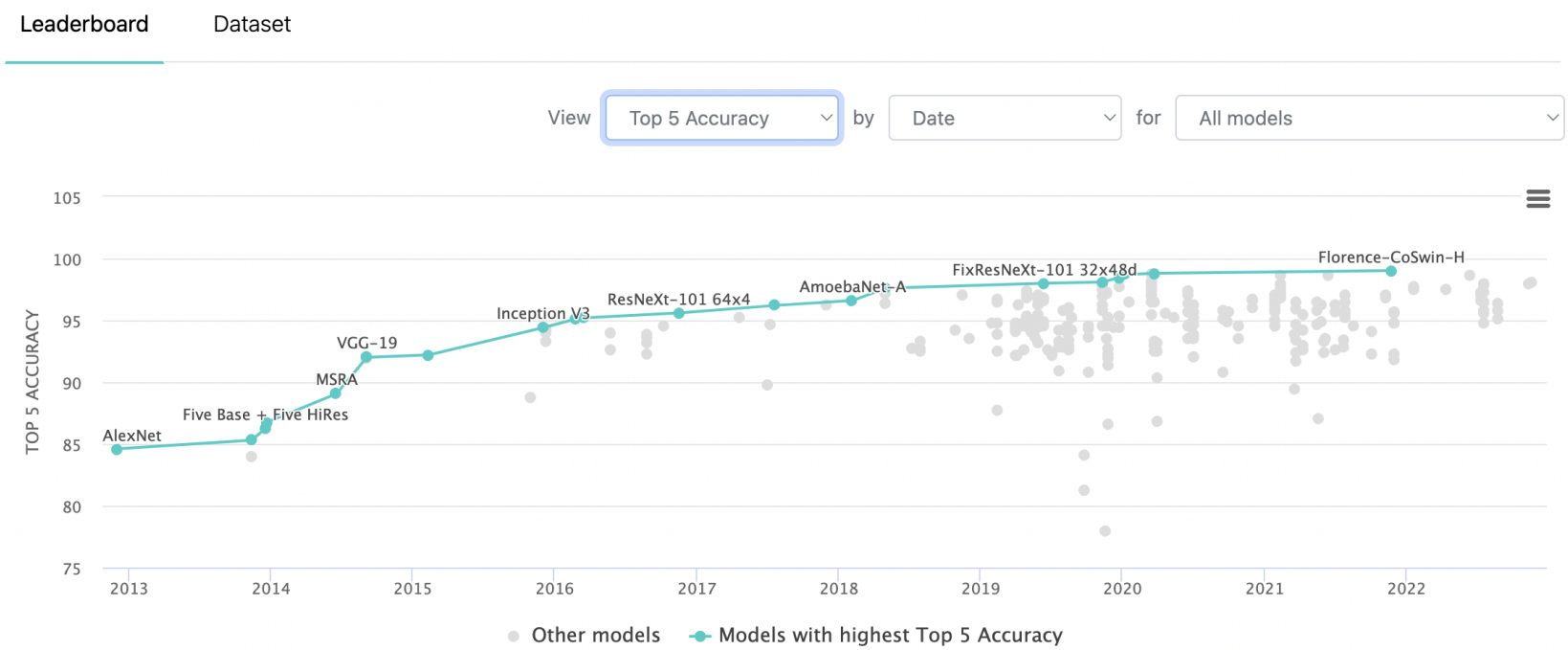
- Top 1 accuracy: 90+%



[Source: https://paperswithcode.com/sota/image-classification-on-imagenet]

Classification Performance in 2023

- Top 5 accuracy: 99+%



[Source: https://paperswithcode.com/sota/image-classification-on-imagenet]

Performance of CNN in 2023

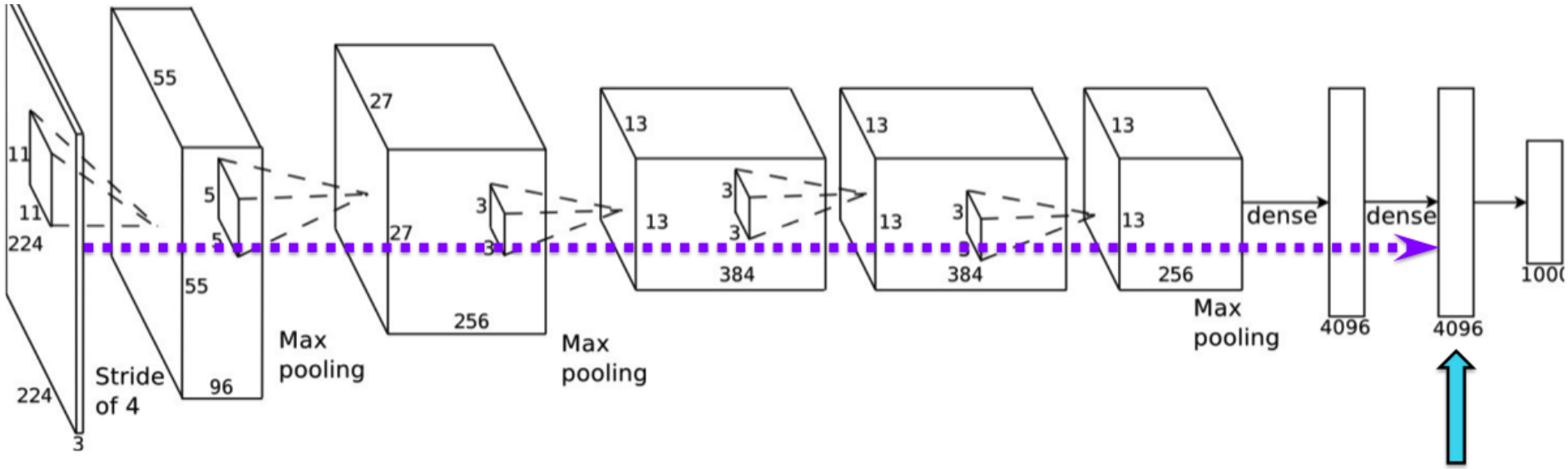
- Stable Diffusion

Figure: Text prompt to 2D Images (Training data: 5 billion images)

[Source: https://www.youtube.com/watch?v=nVhmFski3vg]

Neural Networks as Descriptors

- What vision people like to do is take the already trained network (avoid one week of training), and remove the last classification layer. Then take the top remaining layer (the 4096 dimensional vector here) and use it as a descriptor (feature vector).

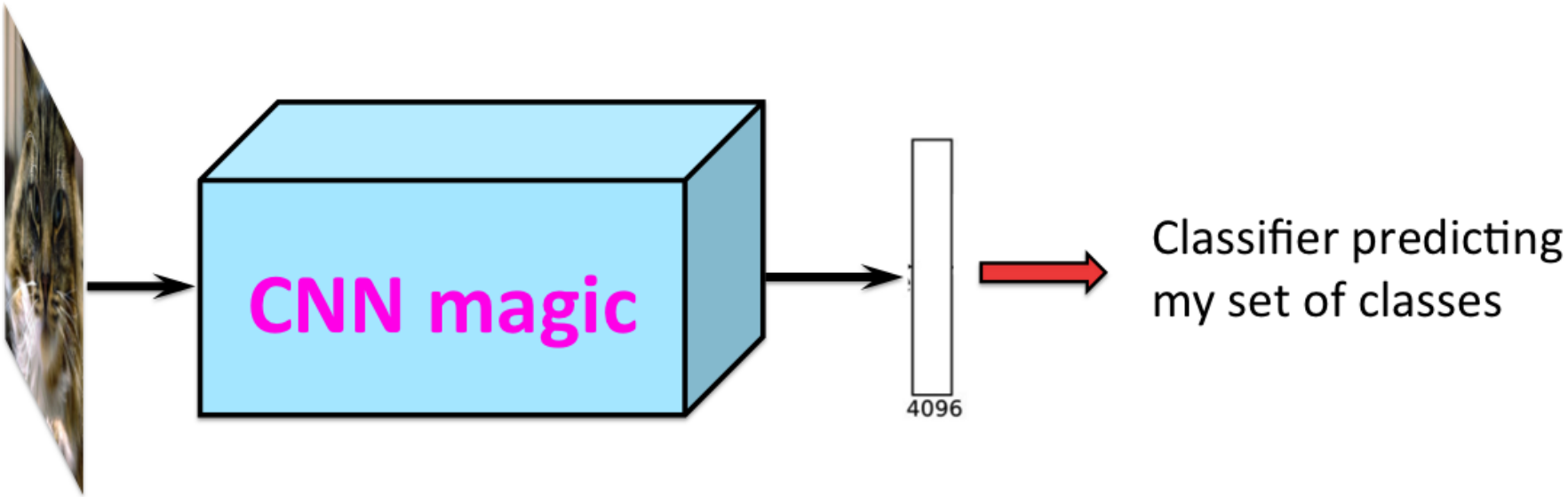


Vision people are mainly interested in this vector. **You can use it as a descriptor**. A much better descriptor than SIFT, etc.

Train your own classifier on top for your choice of classes.

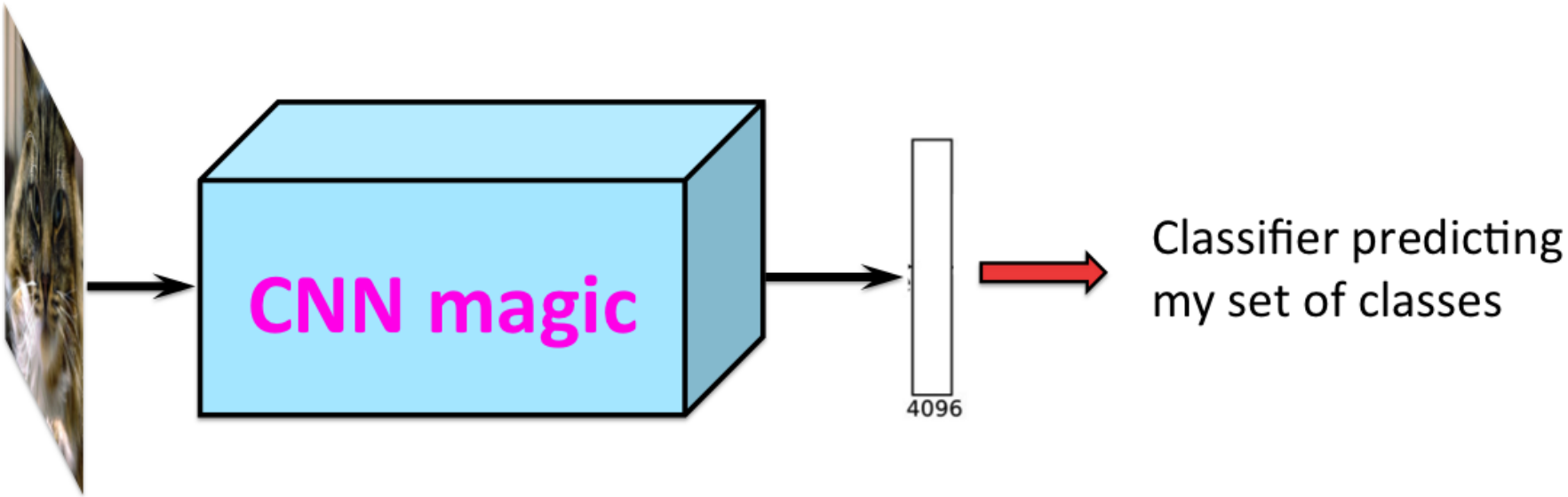
Neural Networks as Descriptors

- What vision people like to do is take the already trained network, and remove the last classification layer. Then take the top remaining layer (the 4096 dimensional vector here) and use it as a descriptor (feature vector).
- Now train your own classifier on top of these features for arbitrary classes.



Neural Networks as Descriptors

- What vision people like to do is take the already trained network, and remove the last classification layer. Then take the top remaining layer (the 4096 dimensional vector here) and use it as a descriptor (feature vector).
- Now train your own classifier on top of these features for arbitrary classes.



So Neural Networks are Great

- So networks turn out to be great.
- At this point Google, Facebook, Microsoft, Baidu "steal" most neural network professors from academia.

So Neural Networks are Great

- But to train the networks you need quite a bit of computational power. So what do you do?

So Neural Networks are Great

- Buy even more.

So Neural Networks are Great

- And train **more layers**. 16 instead of 7 before. 144 million parameters.

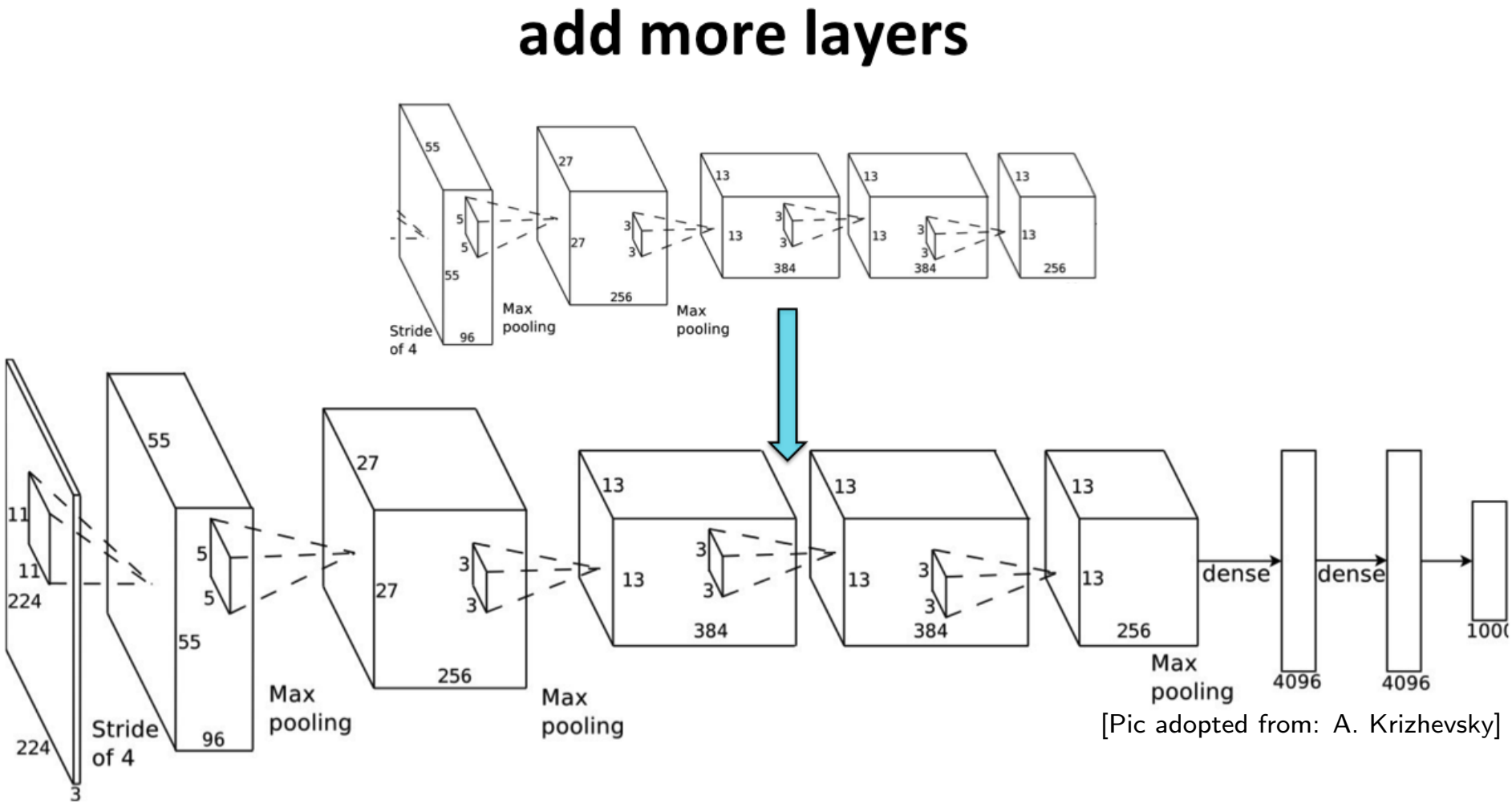
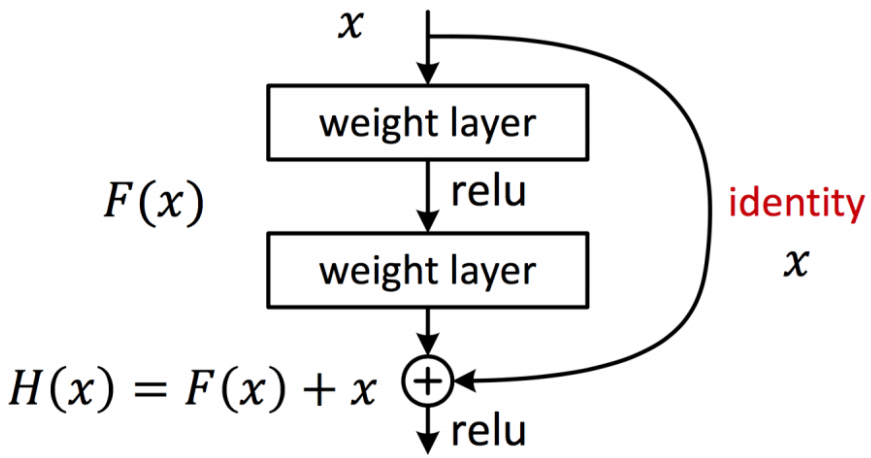
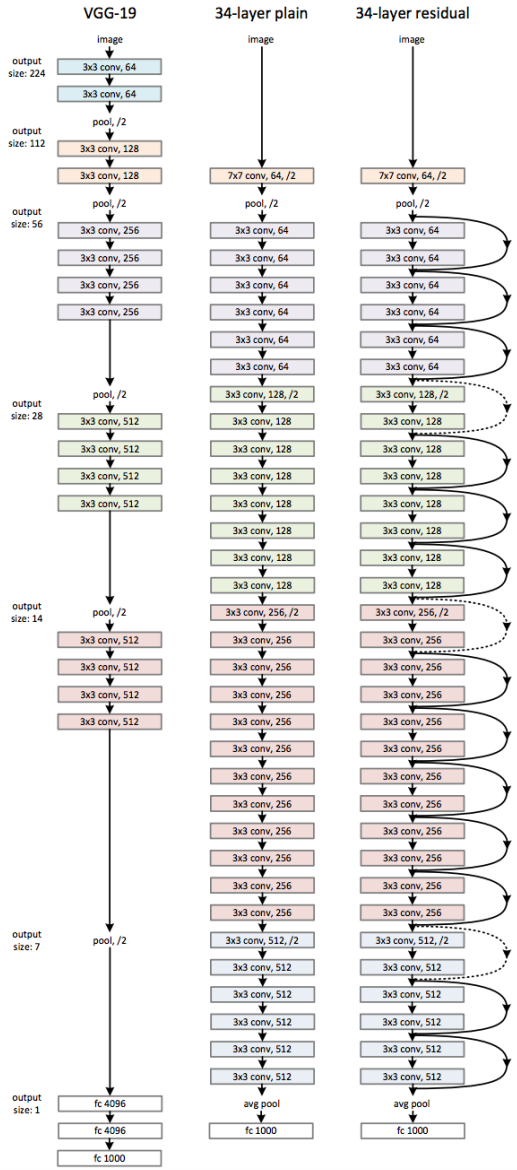


Figure: K. Simonyan, A. Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition. ICLR 2015

150 Layers!

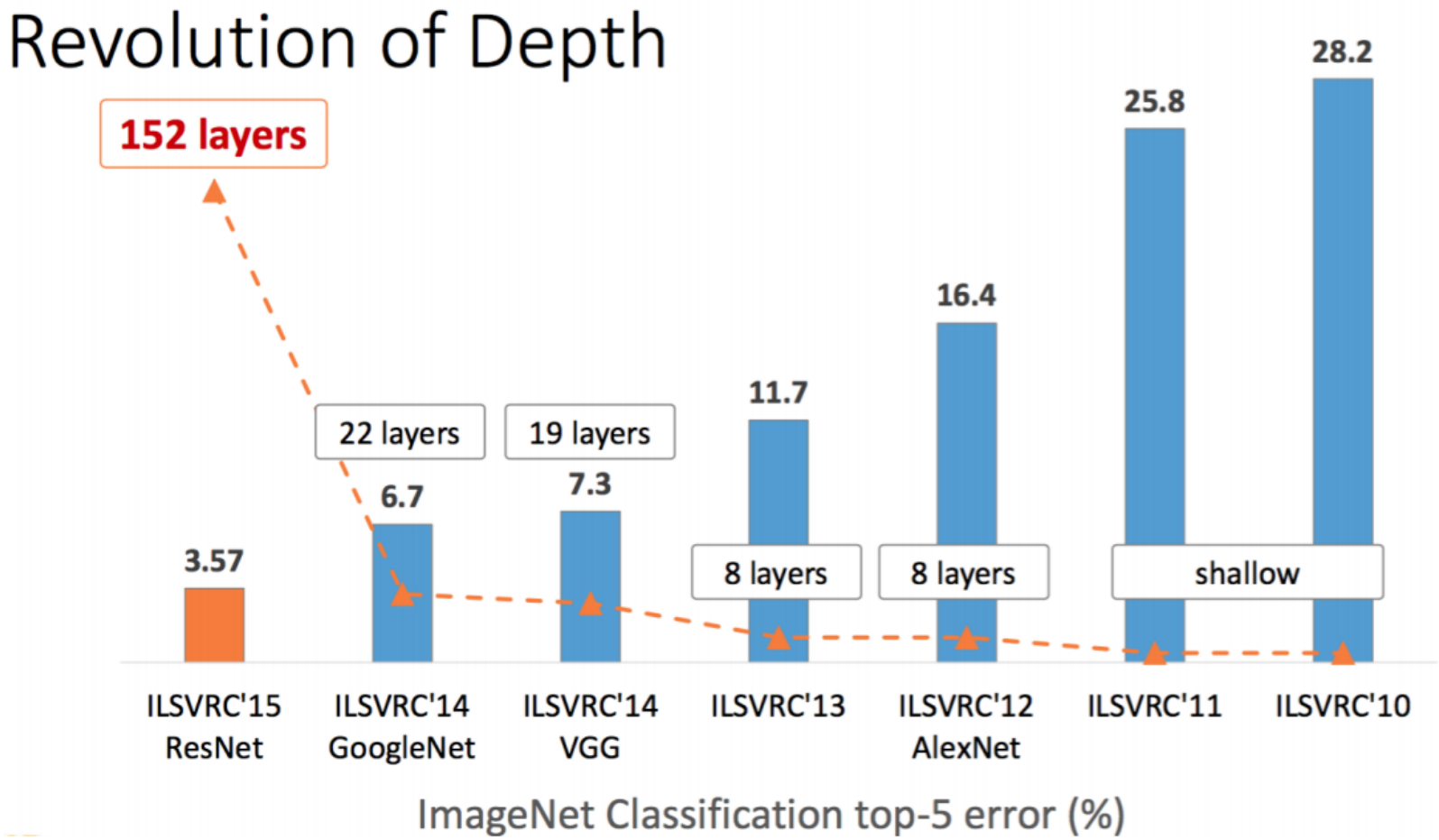
- Networks are now at 150 layers
- They use a skip connections with special form
- In fact, they don't fit on this screen
- Amazing performance!
- A lot of "mistakes" are due to wrong ground-truth





[He, K., Zhang, X., Ren, S. and Sun, J., 2015. Deep Residual Learning for Image Recognition. CVPR 2016 Best paper]

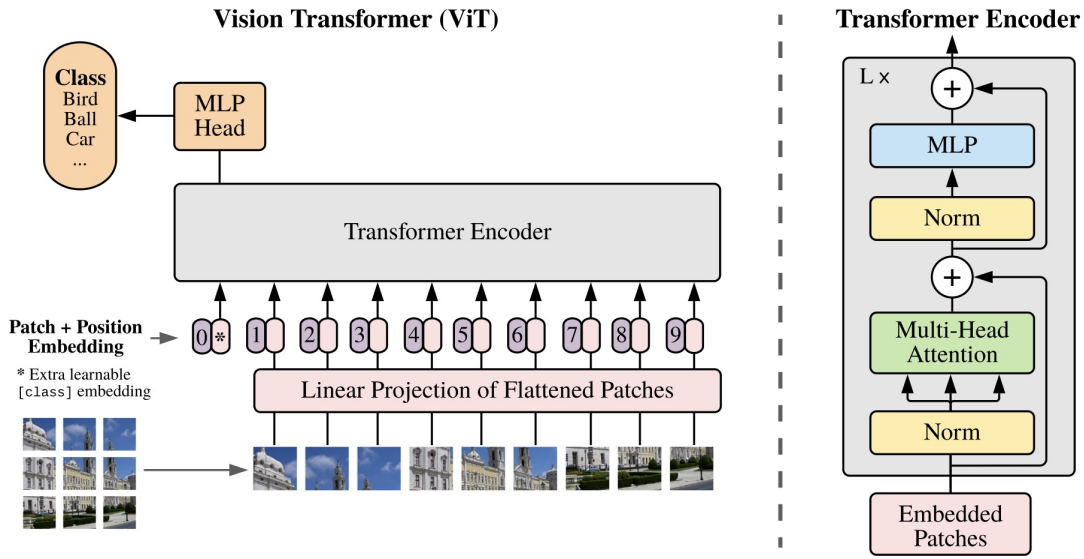
Results: Object Classification



Slide: R. Liao, Paper: [He, K., Zhang, X., Ren, S. and Sun, J., 2015. Deep Residual Learning for Image Recognition. CVPR 2016]

What it looks like in 2023?

- Vision Transformer!
- Self-attention among patches of an image.
- Better performance!



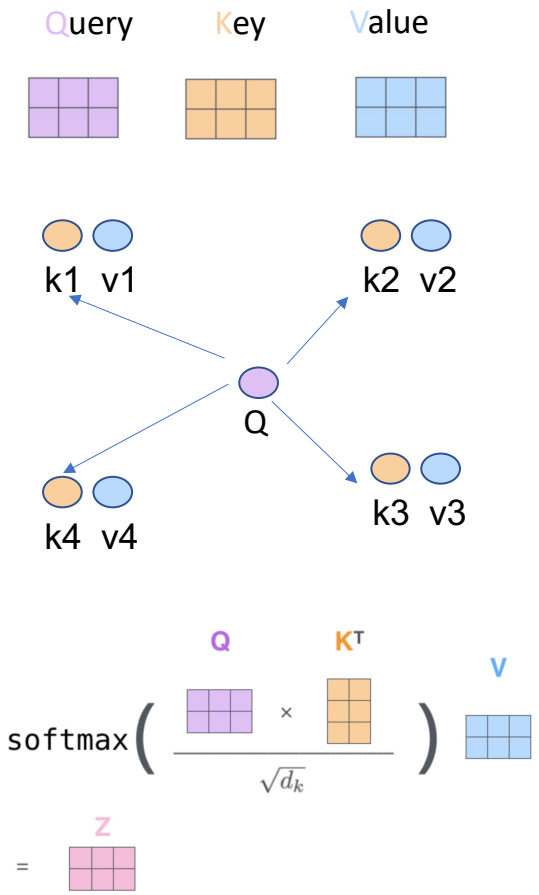


Figure: Self attention mechanism

[An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale, ICLR 2021]

What do CNNs Learn?

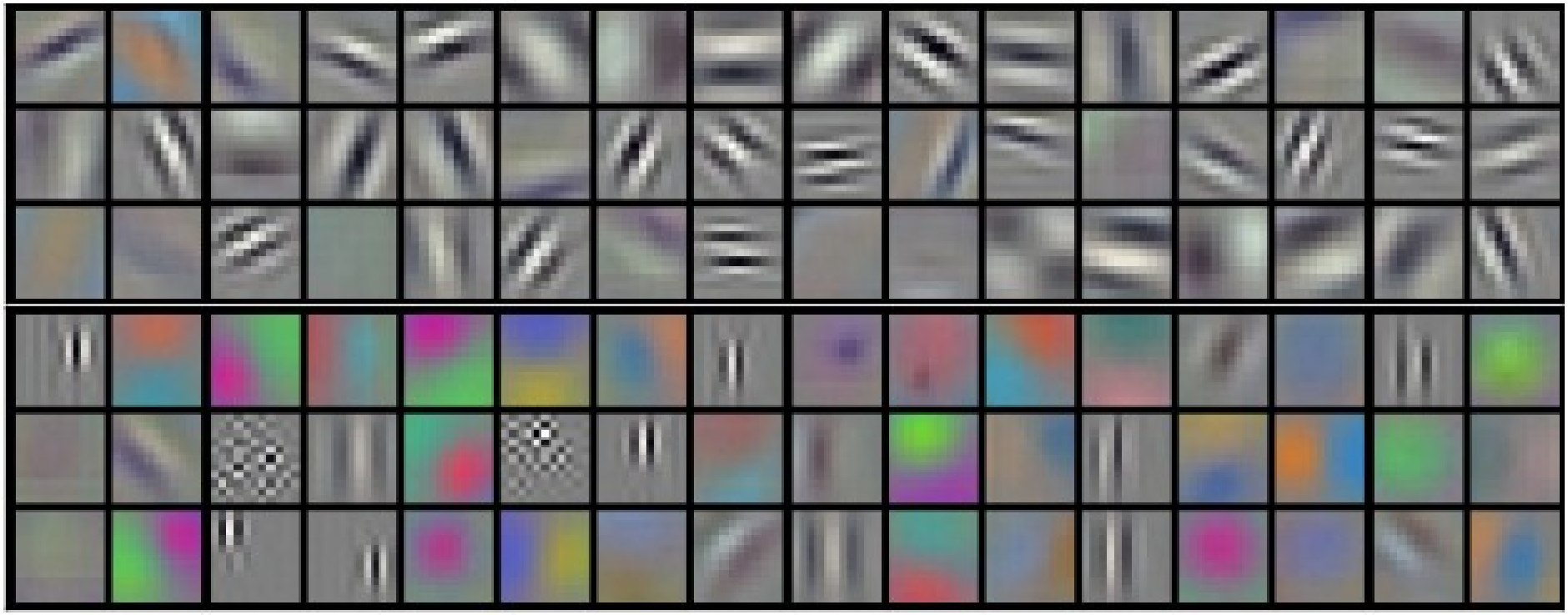


Figure: Filters in the first convolutional layer of Krizhevsky et al

[Matthew D. Zeiler, Rob Fergus, Visualizing and Understanding Convolutional Networks, ECCV 2014]

What do CNNs Learn?

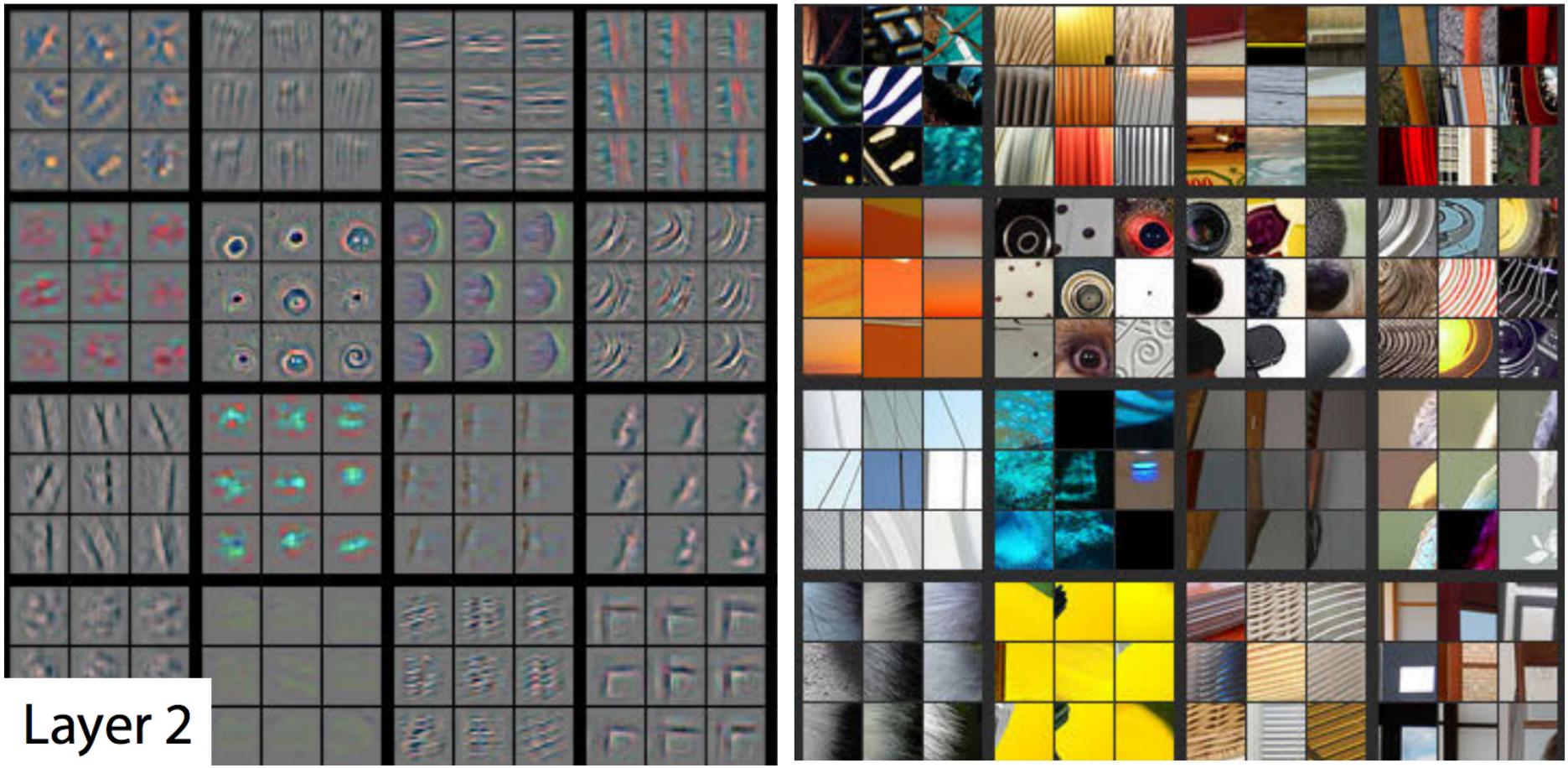


Figure: Filters in the second layer

[Matthew D. Zeiler, Rob Fergus, Visualizing and Understanding Convolutional Networks, ECCV 2014]

What do CNNs Learn?

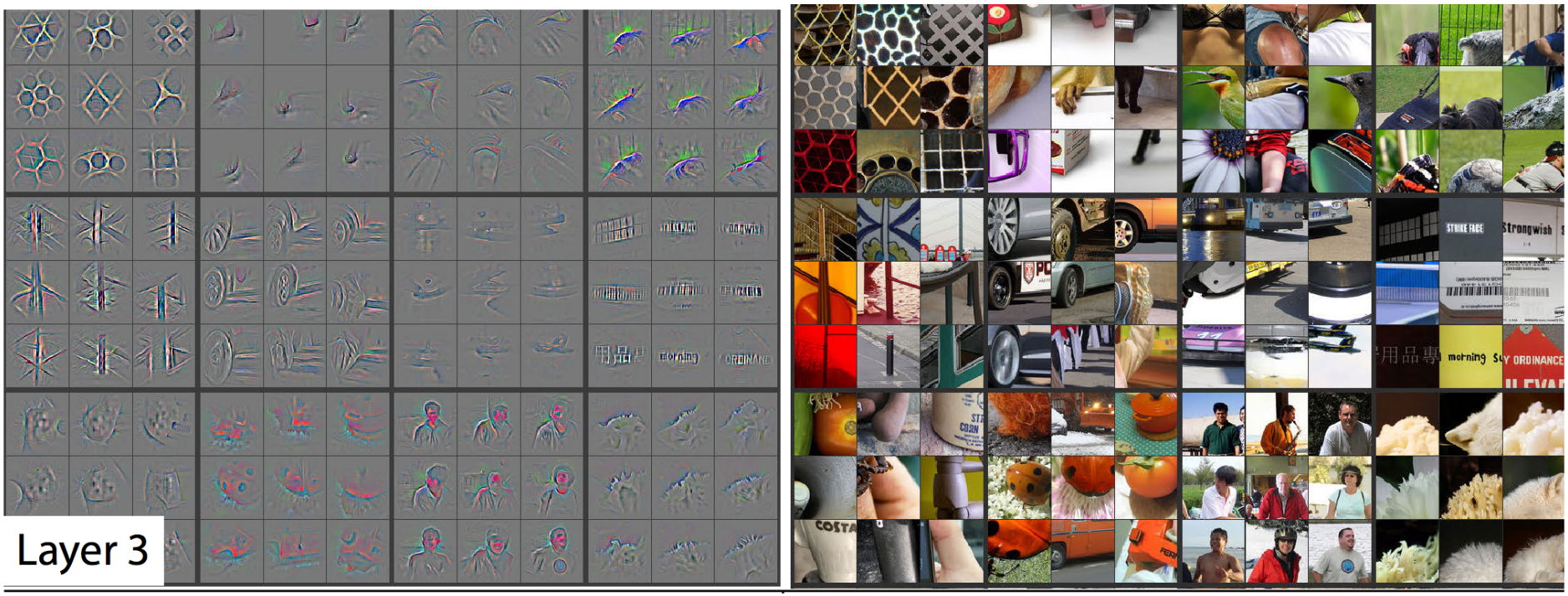
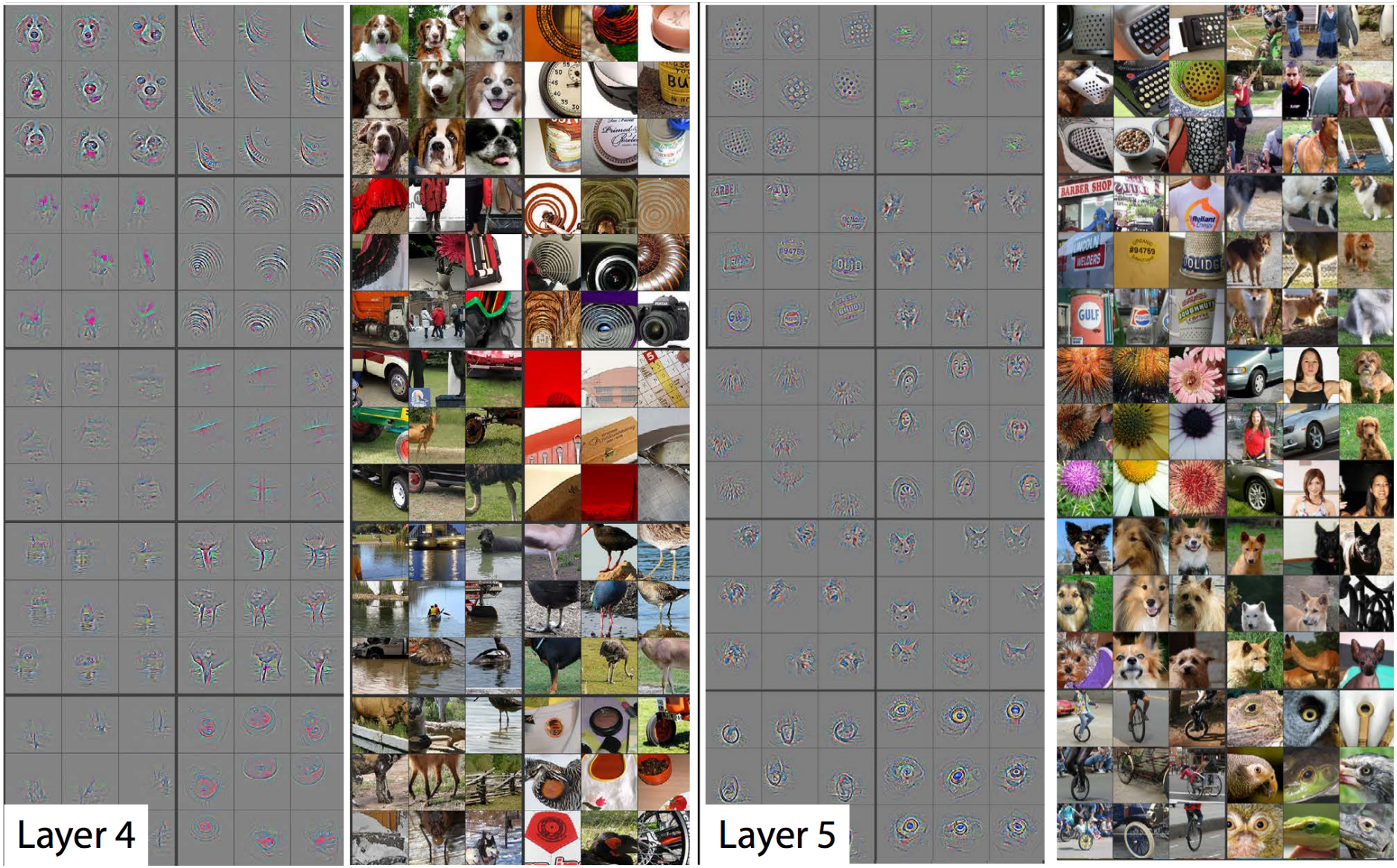


Figure: Filters in the third layer

[Matthew D. Zeiler, Rob Fergus, Visualizing and Understanding Convolutional Networks, ECCV 2014]

What do CNNs Learn?



[Matthew D. Zeiler, Rob Fergus, Visualizing and Understanding Convolutional Networks, ECCV 2014]

Neural Networks – Can Do Anything

- Classification / annotation
- Detection
- Segmentation
- Stereo
- Optical flow

How would you use them for these tasks?

But !!!

- Does this mean we should throw away traditional computer vision techniques?
- Understanding how images are composed or captured paves the way to design better Neural network architecture for learning.
- Great insights from traditional techniques when the computing resources are not enough.
- Even right now, some traditional methods are still important. (stereo matching, keypoints detection, etc.)
- Role of neural network: better feature extractor, function approximator

Neural Networks – Why Do They Work?

- Some cool tricks in design and training:

  A. Krizhevsky, I. Sutskever, G. E. Hinton, *ImageNet Classification with Deep Convolutional Neural Networks*, NIPS 2012

- Computational resources and tones of data
- NNs can **train millions** of parameters from tens of **millions of examples**

Figure: The Imagenet dataset: Deng et al. 14 million images, 1000 classes

Code

Main code:

- Neural network packages:

  Tensorflow, PyTorch

- Object detection:

  `https://github.com/rbgirshick/rcnn`

  `https://github.com/weiliu89/caffe/tree/ssd`

- Semantic Segmentation:

  `https://github.com/open-mmlab/mmsegmentation`

  `https://github.com/CSAILVision/semantic-segmentation-pytorch`

Summary – Stuff Useful to Know

- Basic operations in neural network, including MLP, activation function, Convolutional Layer, loss functions, gradient descnt, back propagation.
- Neural Networks are currently the best feature extractor in computer vision, still active research!
- Mainly because they have multiple layers of nonlinear classifiers, and because they can train from millions of examples efficiently.
- Going forward design computationally less intense solutions with higher generalization power that will beat 1000 layers that Google can afford to do.