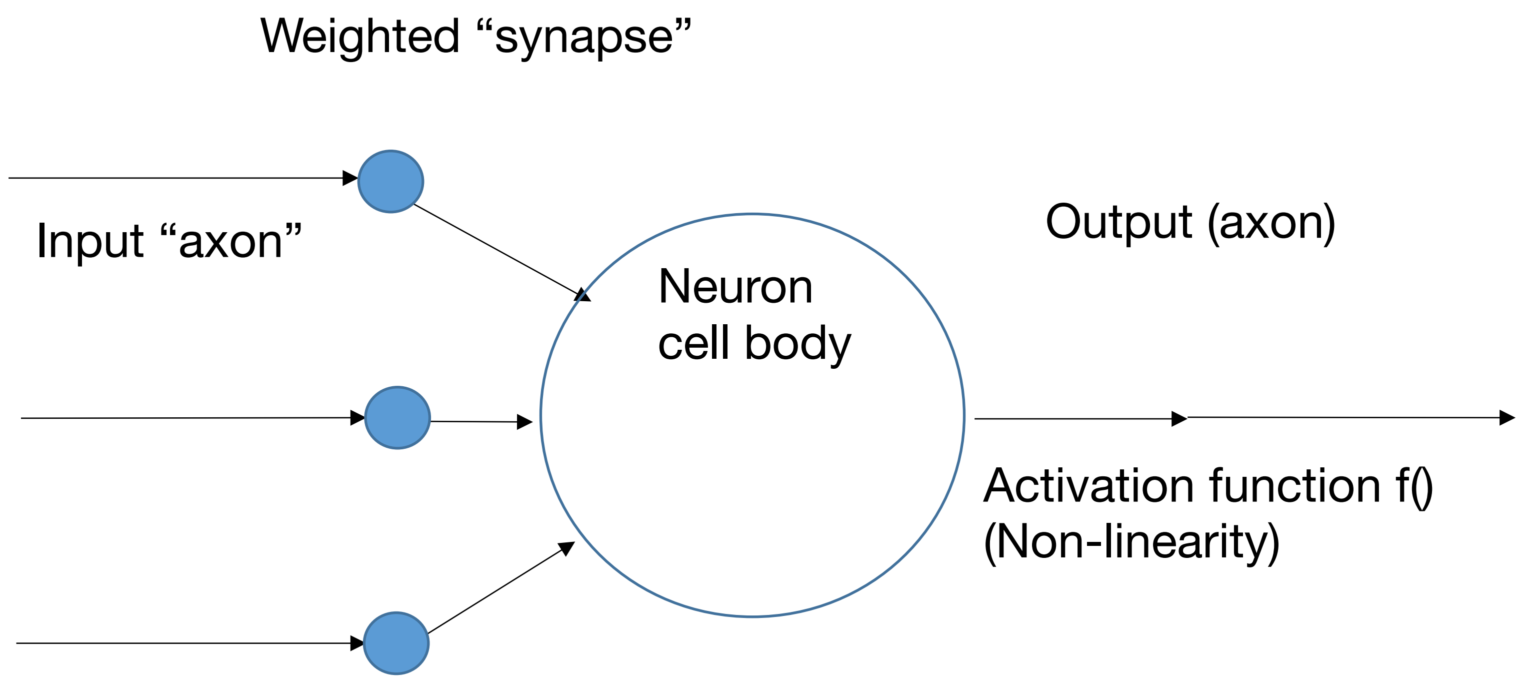
Lecture 10: Ingredients for a convolutional neural network

Machine Learning and Imaging

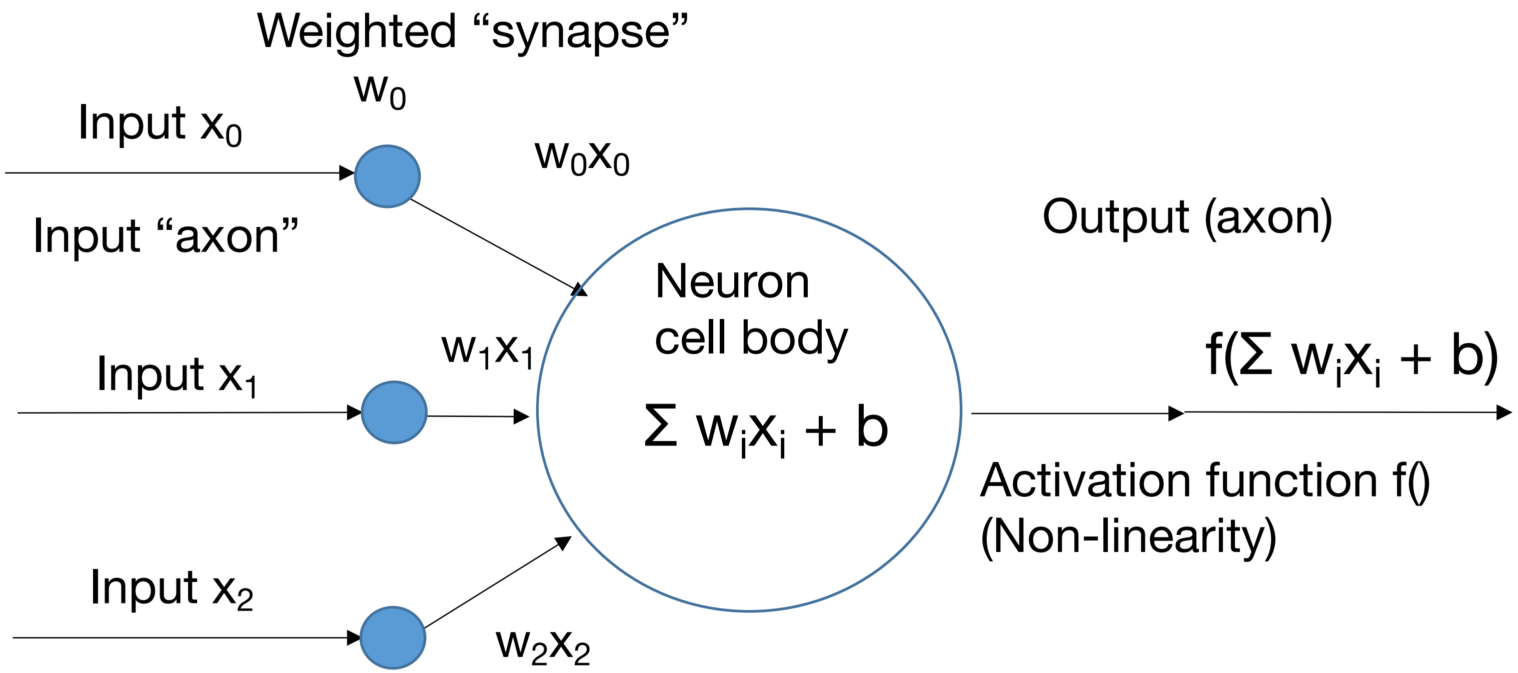
BME 548L
Roarke Horstmeyer

Note: Much material borrowed from Stanford CS231n, Lectures 4 - 10

Today we'll get into neural networks…



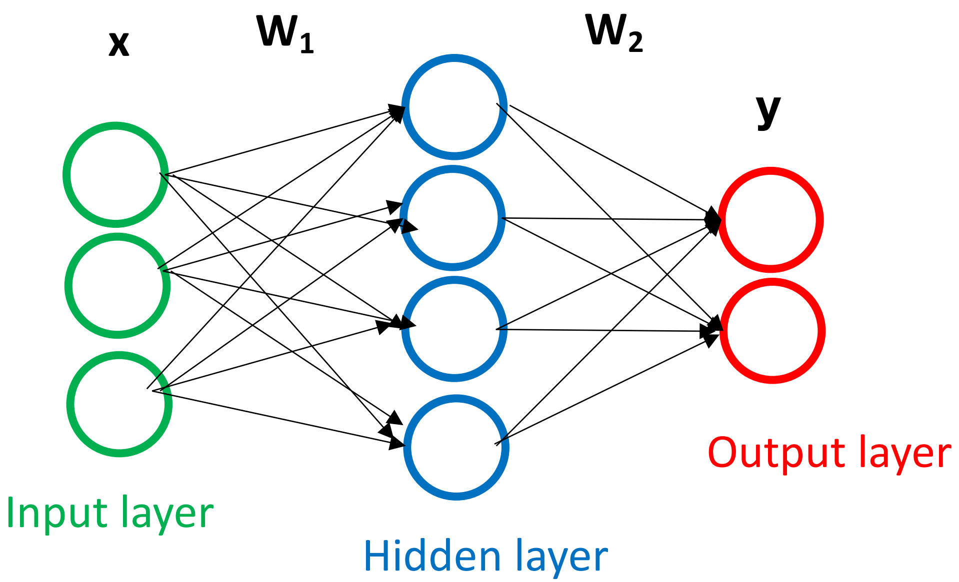
Today we'll get into neural networks…



Single "neuron": Inner product of inputs **x** with learned weights **w** & non-linearity afterwards

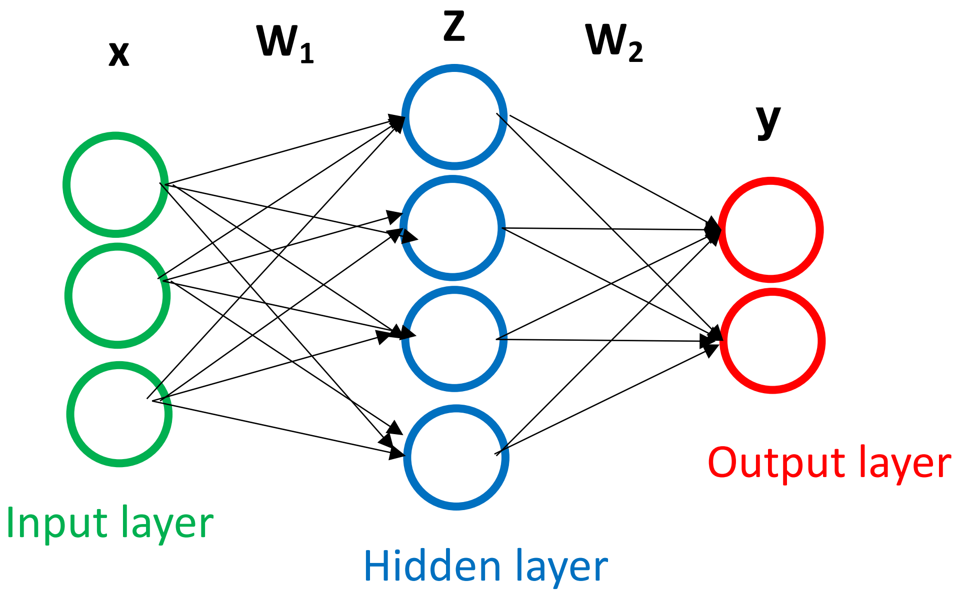
- Multiple weighted inputs: **x** -> y = $\mathbf{w}^T\mathbf{x}$ is "dendrites into cell body"
- Non-linearity f () after sum = "neuron's activation function" (loose interp.)

Today we'll get into neural networks…



- For multiple cells (units), use matrix **W** to connect inputs to outputs
- These cascade in layers

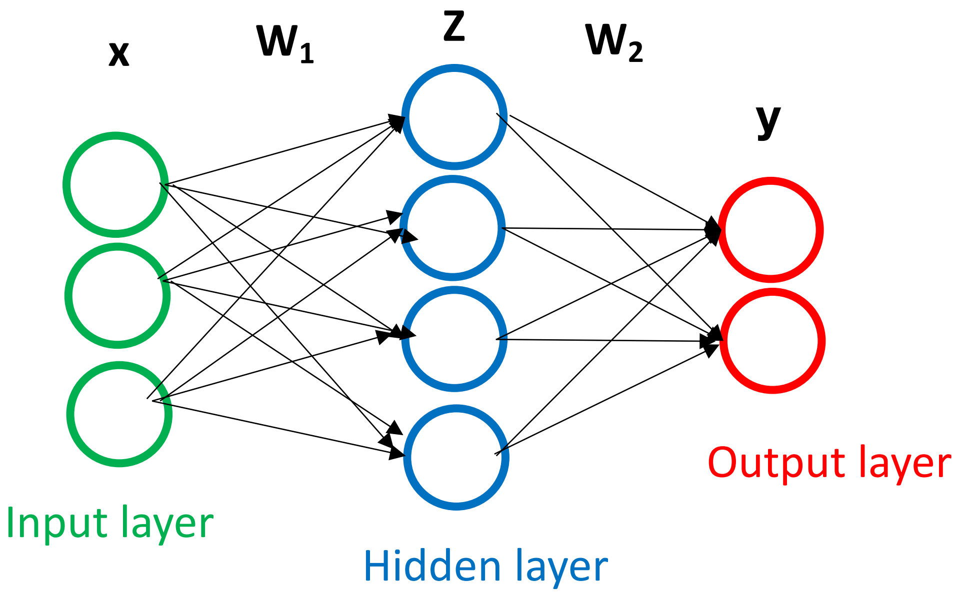
Today we'll get into neural networks…

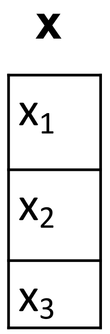


- Let's consider the first step – from input layer to hidden layer -

  Can you write a matrix expression that maps the input to hidden layer?

Today we'll get into neural networks…

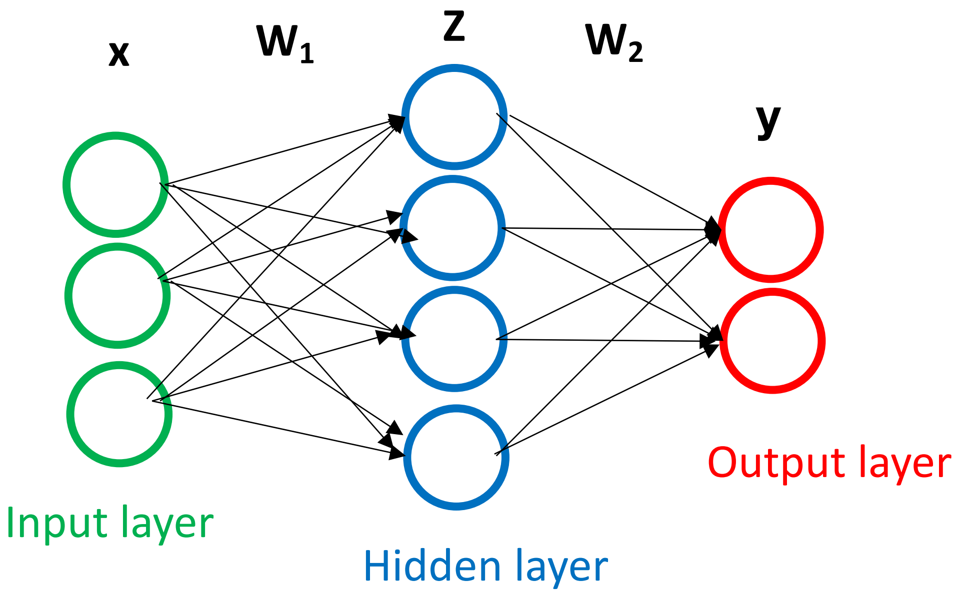




- Let's consider the first step – from input layer to hidden layer -

  Can you write a matrix expression that maps the input to hidden layer?

Today we'll get into neural networks…

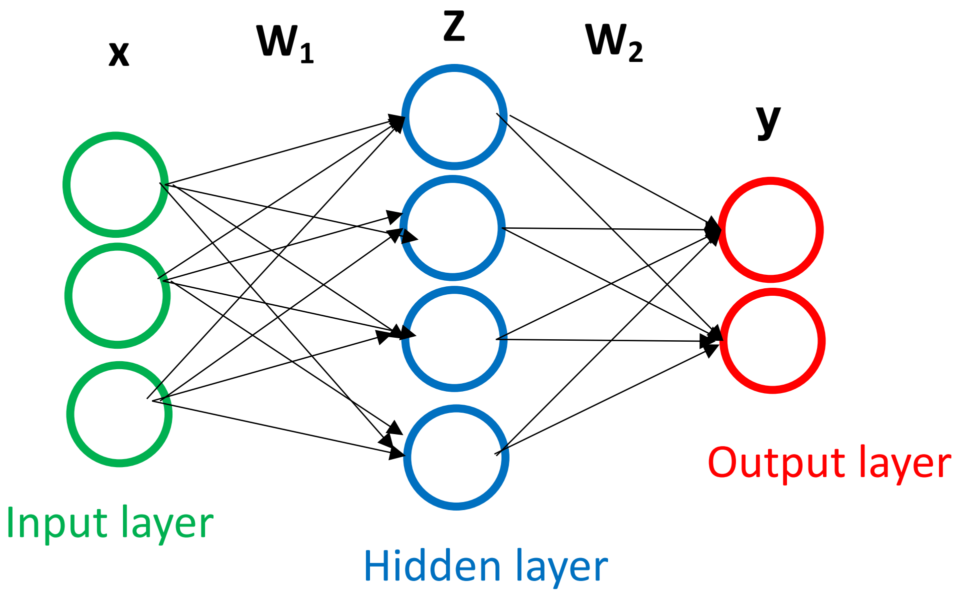


$$\begin{array}{c} \mathbf{z} \\ \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix} \end{array} = \qquad \begin{array}{c} \mathbf{x} \\ \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \end{array}$$

- Let's consider the first step – from input layer to hidden layer -

Can you write a matrix expression that maps the input to hidden layer?

Today we'll get into neural networks…



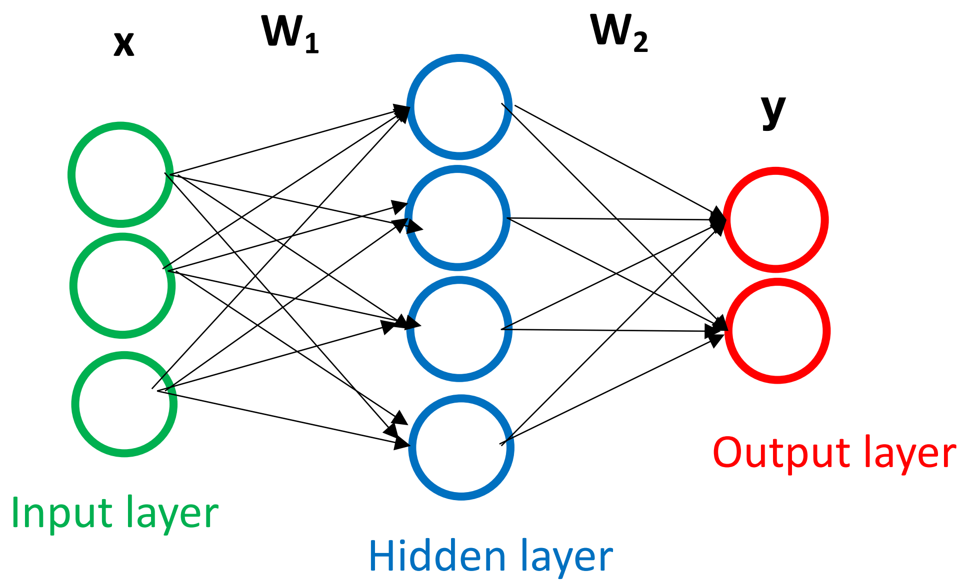
$$\overset{\mathbf{z}}{\begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix}} = \mathrm{NL}\cdot \overset{\mathbf{W_1}}{\begin{bmatrix} w_{11} & w_{21} & w_{21} \\ w_{12} & w_{22} & w_{22} \\ w_{13} & w_{23} & w_{23} \\ w_{14} & w_{24} & w_{24} \end{bmatrix}} \overset{\mathbf{x}}{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}$$

- Let's consider the first step – from input layer to hidden layer -

Can you write a matrix expression that maps the input to hidden layer?

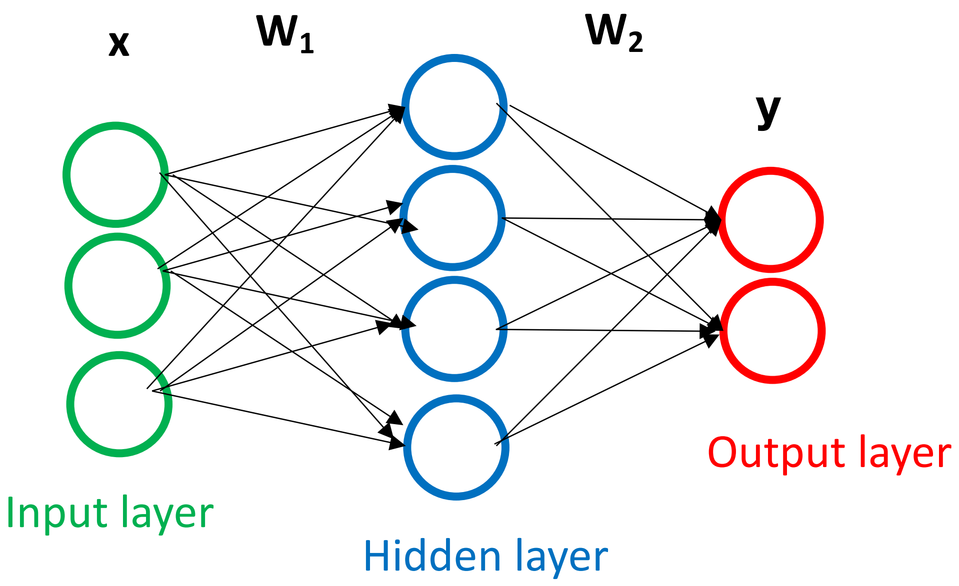
deep imaging

Today we'll get into neural networks…



- Next, let's write out the full chain from input x to output y!

Today we'll get into neural networks…

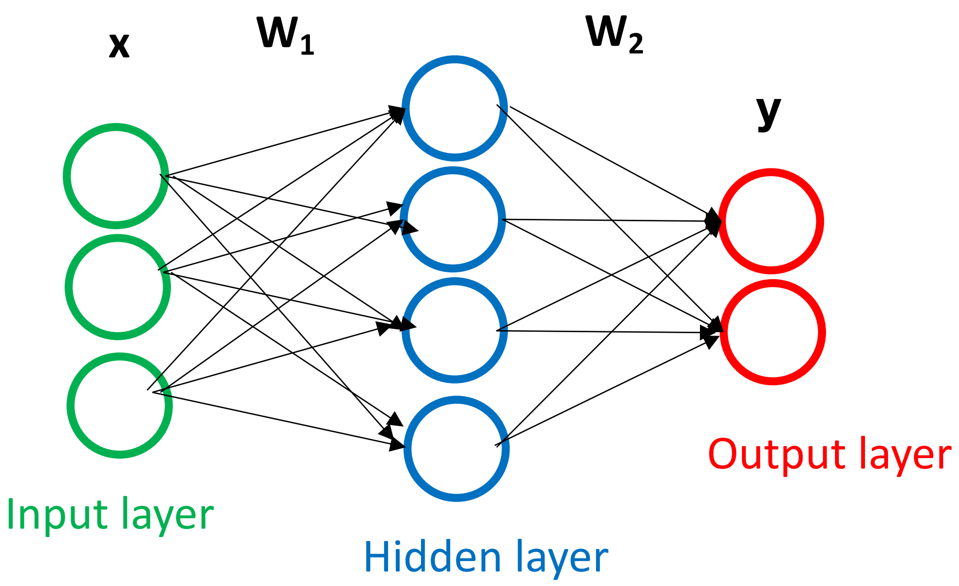


$$\mathbf{y} = NL \cdot \mathbf{W_2} \, NL \cdot \mathbf{W_1} \, \mathbf{x}$$

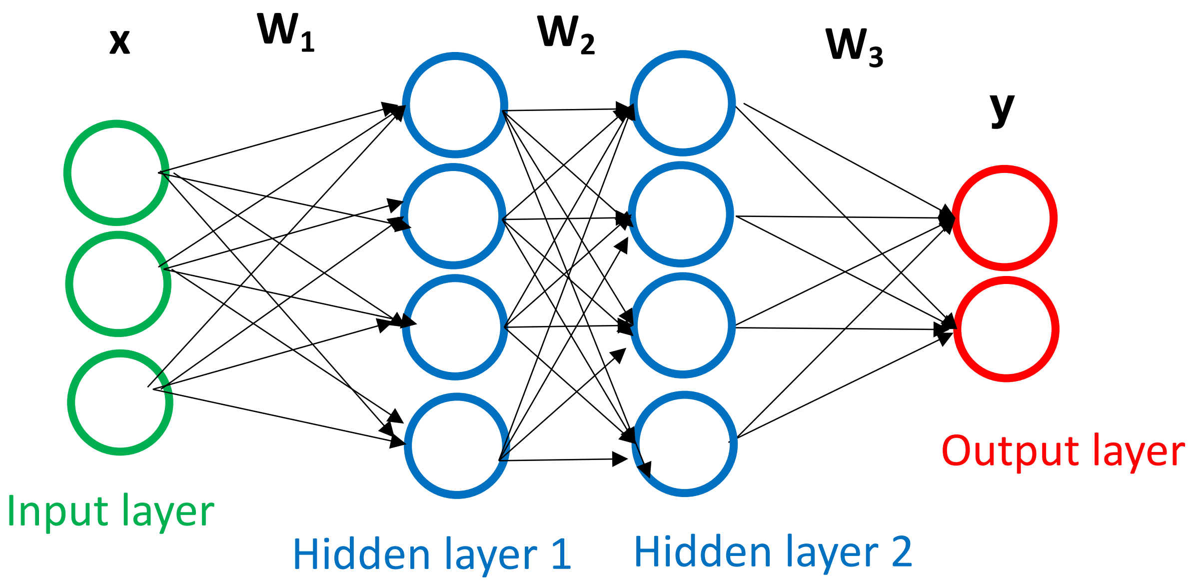
$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = NL \cdot \begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \end{bmatrix} NL \cdot \begin{bmatrix} w_{11} & w_{21} & w_{21} \\ w_{12} & w_{22} & w_{22} \\ w_{13} & w_{23} & w_{23} \\ w_{14} & w_{24} & w_{24} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Neural networks = cascaded set of multiplies and non-linearities

2-layer network:



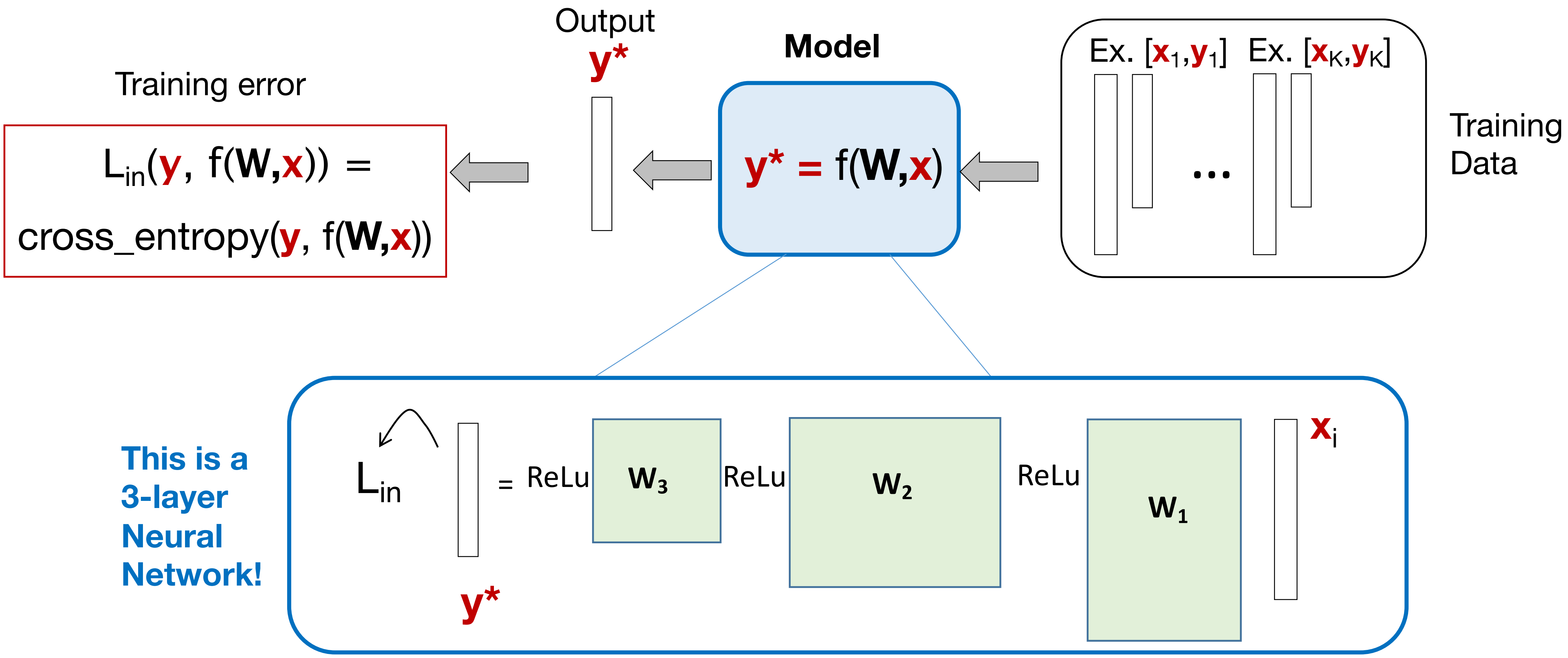
3-layer network:



or 3-layer Neural Network

$$f = W_3 \max(0, W_2 \max(0, W_1 x))$$

Our very basic convolutional neural network



Forward pass: from $\mathbf{x}_i$ and current $\mathbf{W}$'s, find L_{in}

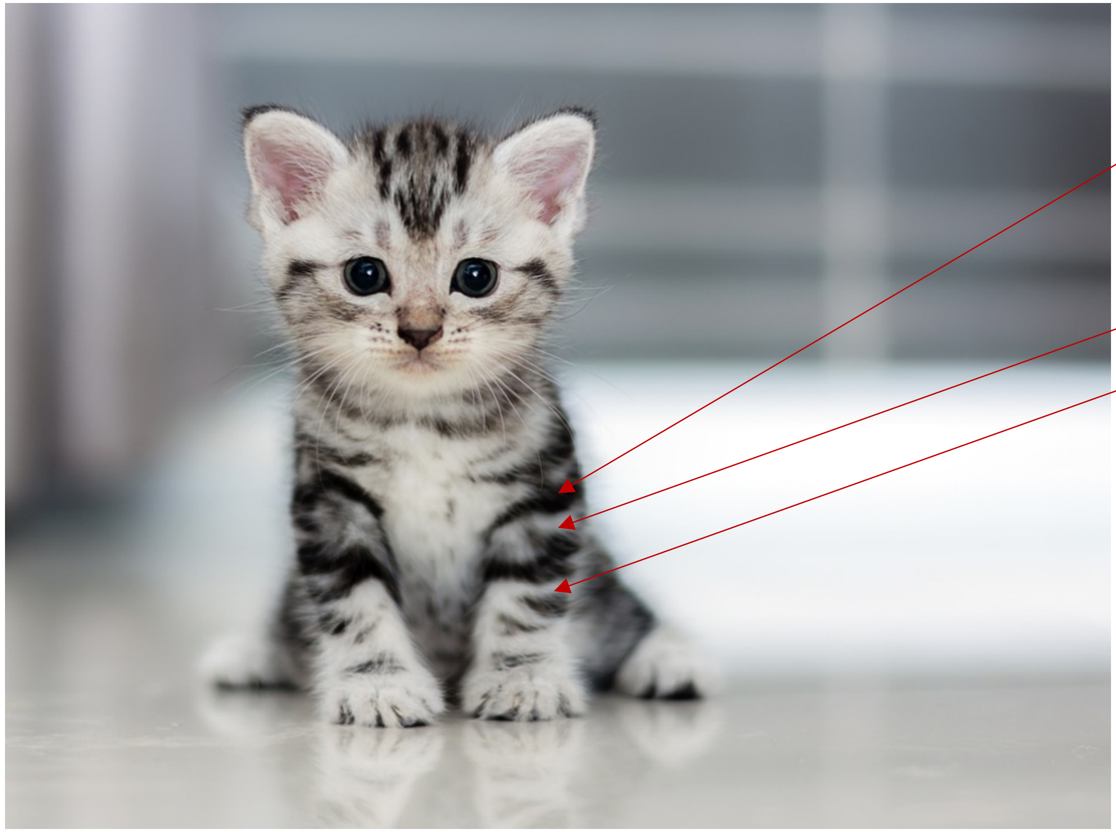
Insight: Do we really need to mix every image pixel with every other image pixel to start?

deep imaging

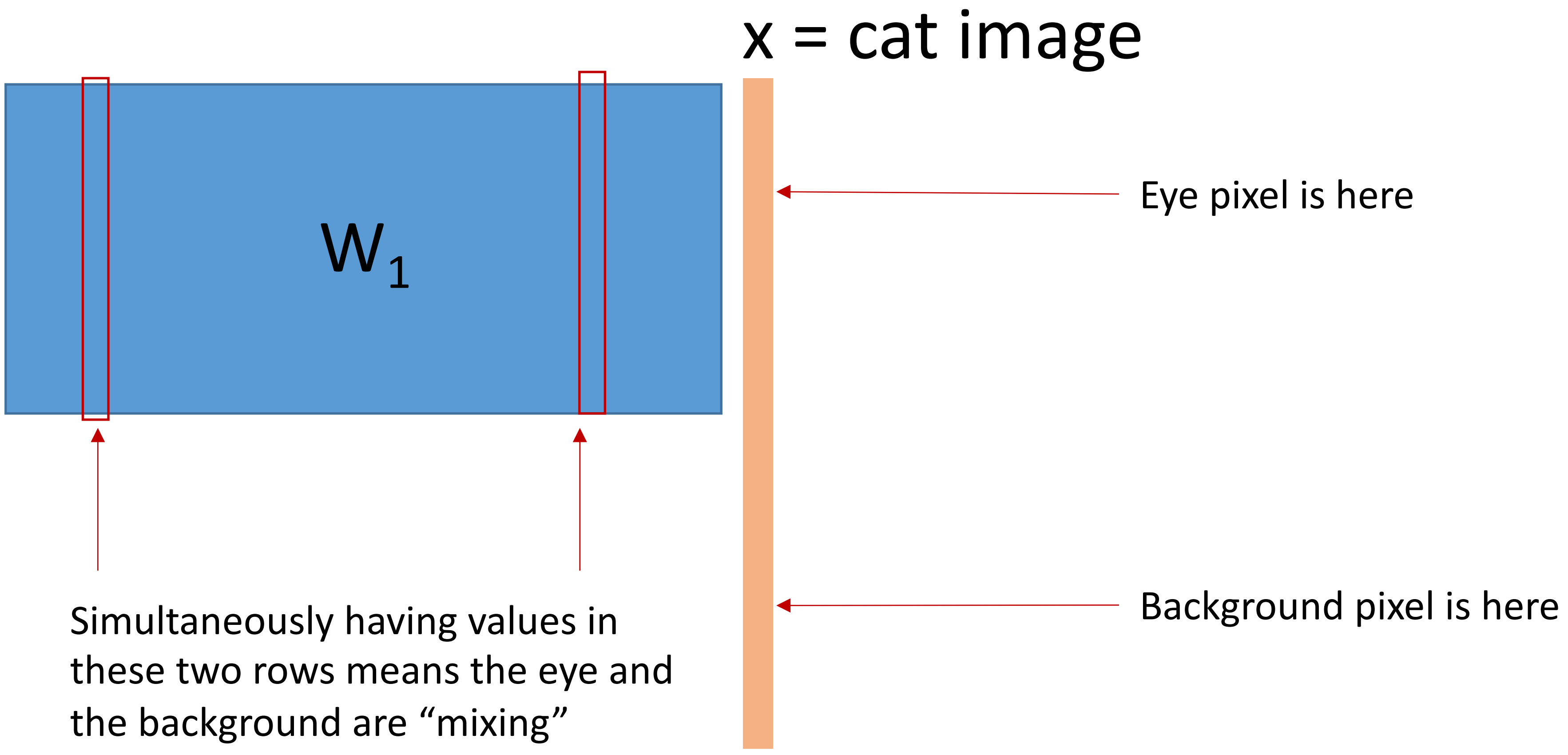
Insight: Do we really need to mix every image pixel with every other image pixel to start?

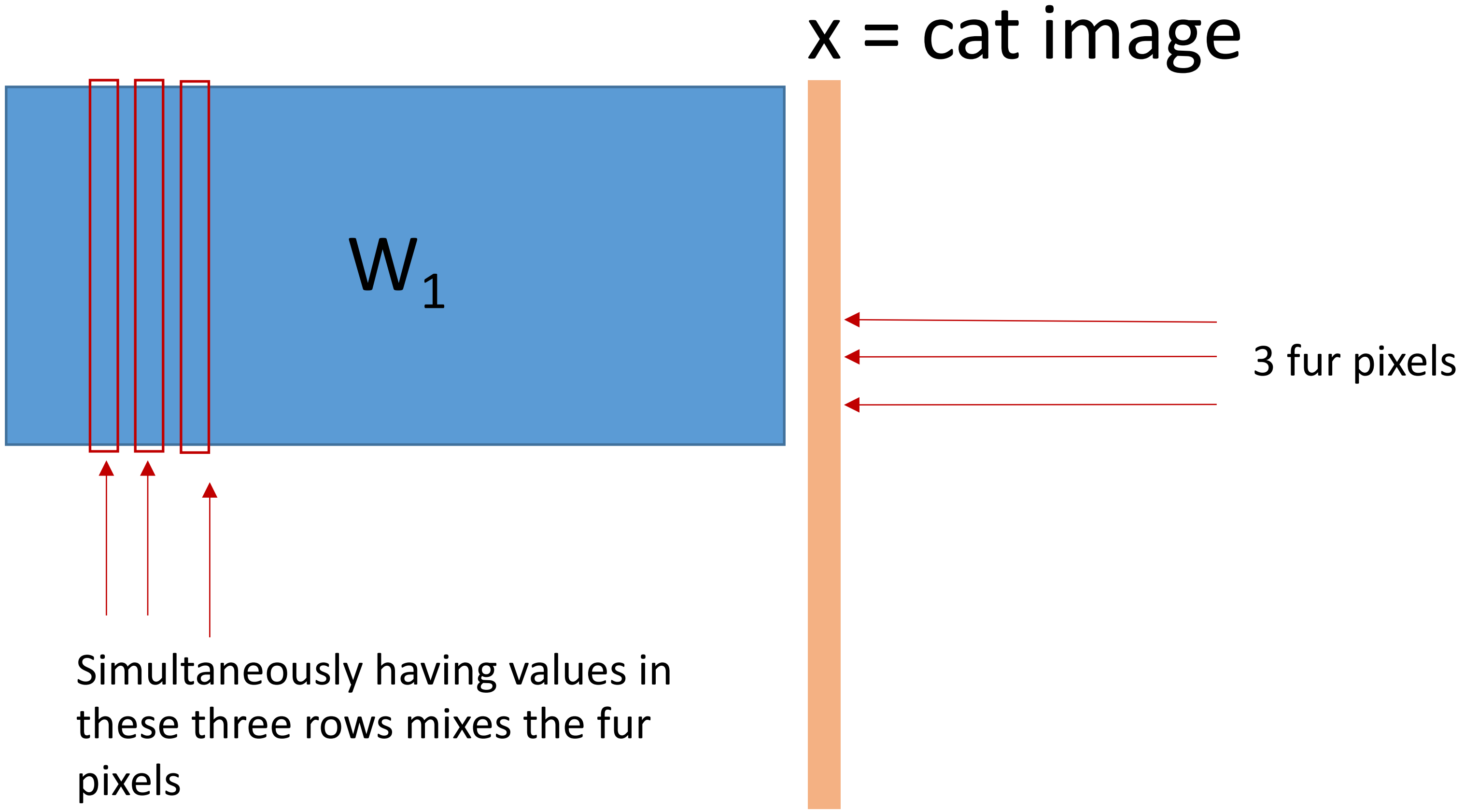
We probably don't need to mix these two pixels to figure out that this is a cat

Insight: Do we really need to mix every image pixel with every other image pixel to start?



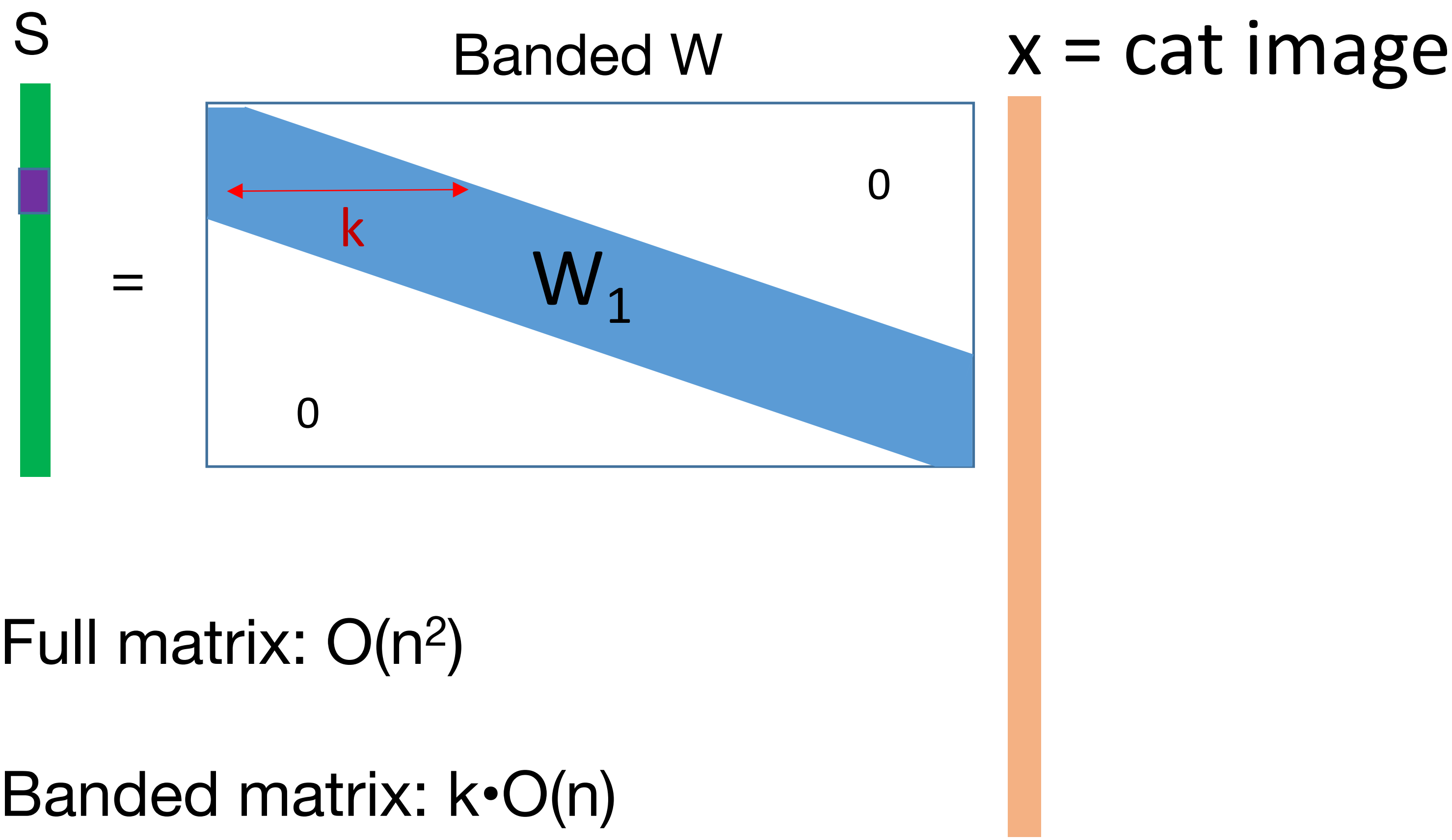
But understanding the stripes in these 3 pixels right near each other is going to be pretty helpful…


x = cat image
W1
Eye pixel is here
Background pixel is here
Simultaneously having values in these two rows means the eye and the background are "mixing"



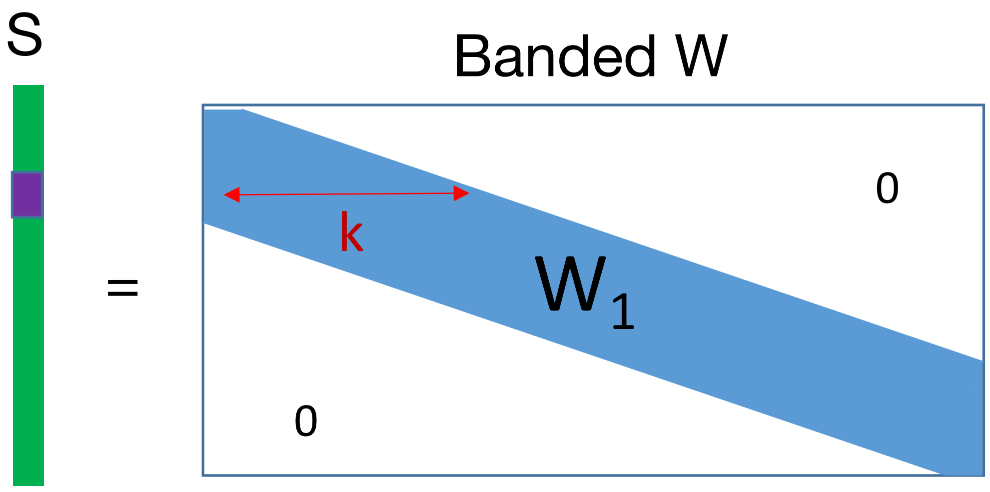
x = cat image
W1
3 fur pixels
Simultaneously having values in these three rows mixes the fur pixels

Insight from last lecture:



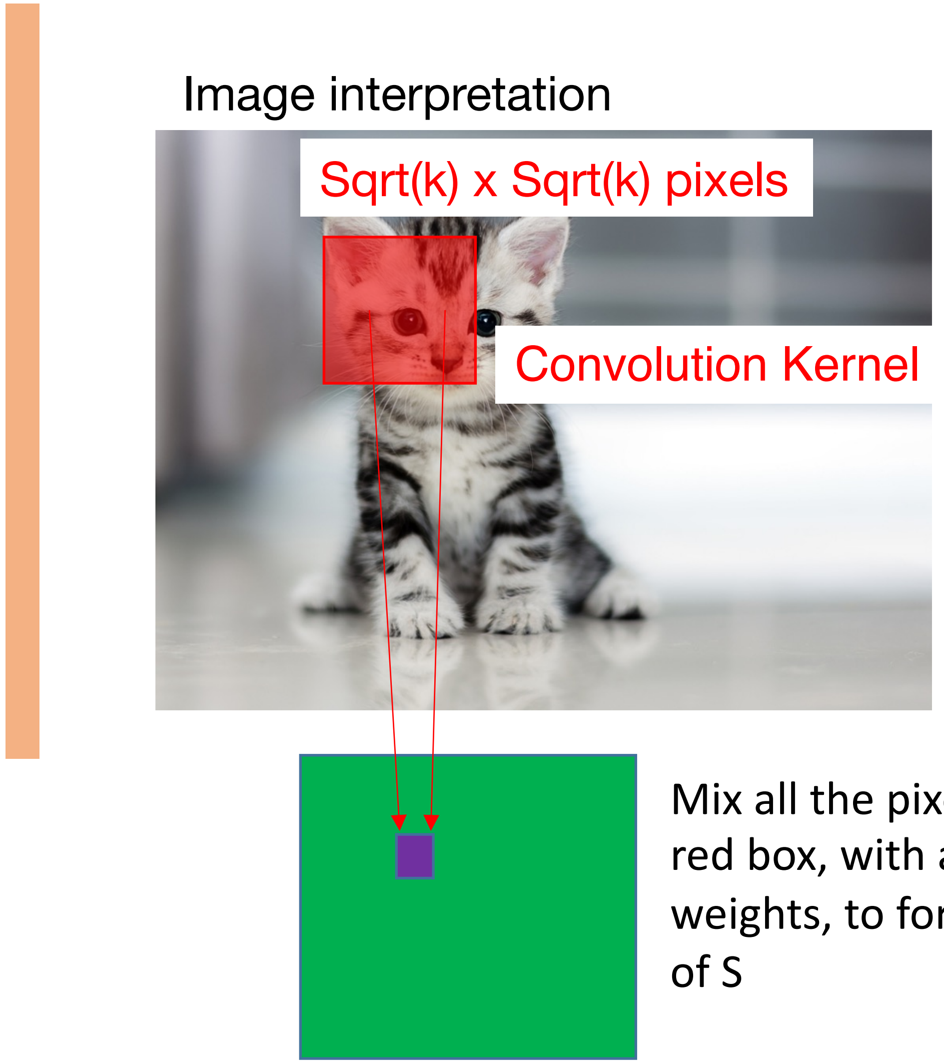
- Full matrix: $O(n^2)$
- Banded matrix: $k \cdot O(n)$
- **Banded Toeplitz matrix: k**

Insight from last lecture:



x = cat image

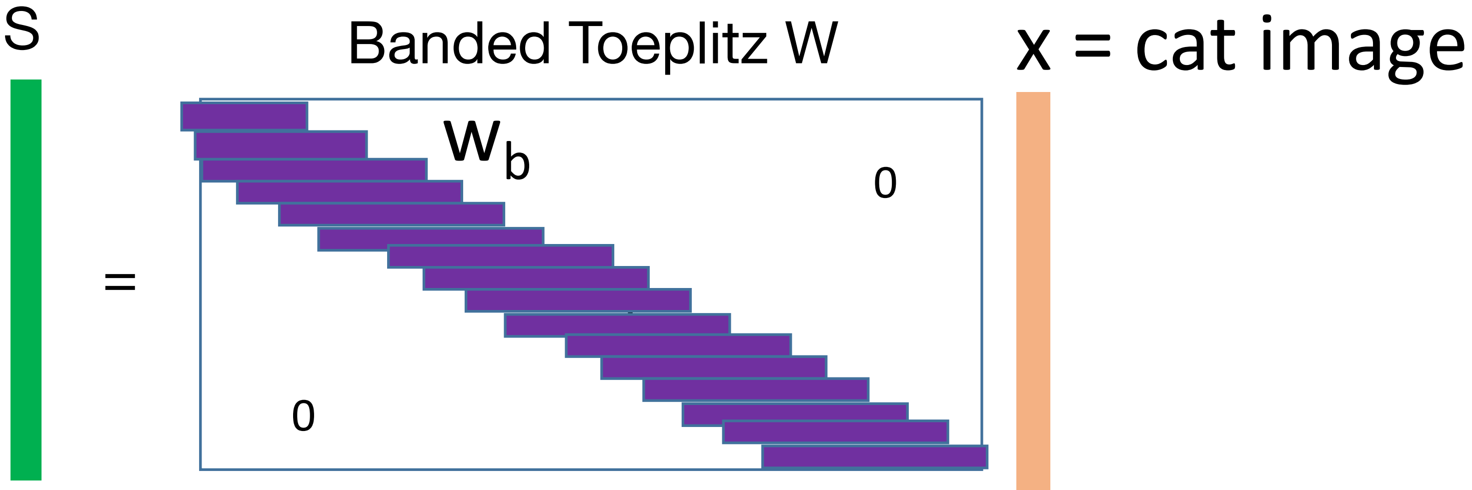
Image interpretation



Mix all the pixels in the red box, with associated weights, to form this entry of S

- Full matrix: $O(n^2)$
- Banded matrix: $k \cdot O(n)$
- **Banded Toeplitz matrix: k**

Simplification #2: Have each band be *the same weights*



This type of matrix can dramatically reduce the number of weights that are used while still allowing *local* regions to mix:

Full matrix: $O(n^2)$
Banded matrix: $k \cdot O(n)$
Banded Toeplitz matrix: k

This is the definition of a convolution

Weights "savings" via convolution

$$L(w) = \frac{1}{N}\sum_{i=1}^{N} \ln(1 + e^{-y_i \max(0, \mathbf{W}_3 \max(0, \mathbf{W}_2 \max(0, \mathbf{W}_1 x_i)))})$$

- Having "fully connected" weight matrices can produce quite a lot of weights…let's consider a binary classification task:

CIFAR10 dataset: each image is 32x32 pixels
Let's say W1 has 500 rows
Let's say W2 has 100 rows
Recall that W3 must have 2 rows

What is the total number of weights that we must optimize?

Weights “savings” via convolution

$$L(w) = \frac{1}{N}\sum_{i=1}^{N} \ln(1 + e^{-y_i \max(0, \mathbf{W}_3 \max(0, \mathbf{W}_2 \max(0, \mathbf{W}_1 x_i)))})$$

- Having “fully connected” weight matrices can produce quite a lot of weights…let’s consider a binary classification task:

CIFAR10: 32x32 images = 1024 pixels
W1 = 1024x500
W2 = 500x100
W3: 100x2

Total number of weights: 562,200

deep imaging

Weights “savings” via convolution

$$L(w) = \frac{1}{N}\sum_{i=1}^{N} \ln\left(1 + e^{-y_i \max(0, \mathbf{W}_3 \max(0, \mathbf{W}_2 \max(0, \mathbf{W}_1 x_i)))}\right)$$

- Having “fully connected” weight matrices can produce quite a lot of weights…let’s consider a binary classification task:

Total number of weights: 562,200

- What if we instead used a convolutional neural network, where $\mathbf{W_1}$ and $\mathbf{W_2}$ are now convolution operations with a 10x10 pixel convolution kernel?

Weights “savings” via convolution

$$L(w) = \frac{1}{N}\sum_{i=1}^{N} \ln(1 + e^{-y_i \max(0, \mathbf{W}_3 \max(0, \mathbf{W}_2 \max(0, \mathbf{W}_1 x_i)))})$$

- Having “fully connected” weight matrices can produce quite a lot of weights…let’s consider a binary classification task:

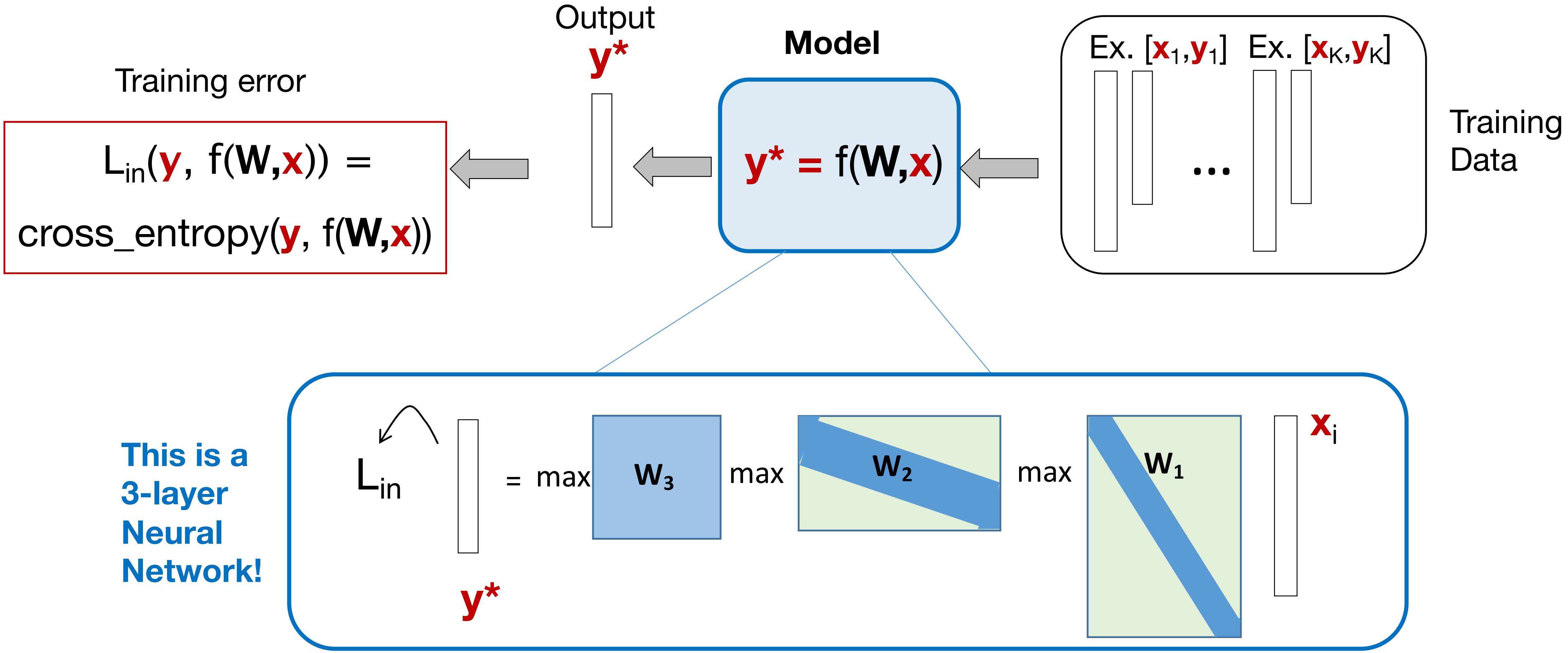
Total number of weights: 562,200

- What if we instead used a convolutional neural network, where $\mathbf{W_1}$ and $\mathbf{W_2}$ are now convolution operations with a 10x10 pixel convolution kernel?

W1 = 10x10
W2 = 10x10
W3: 1024x2

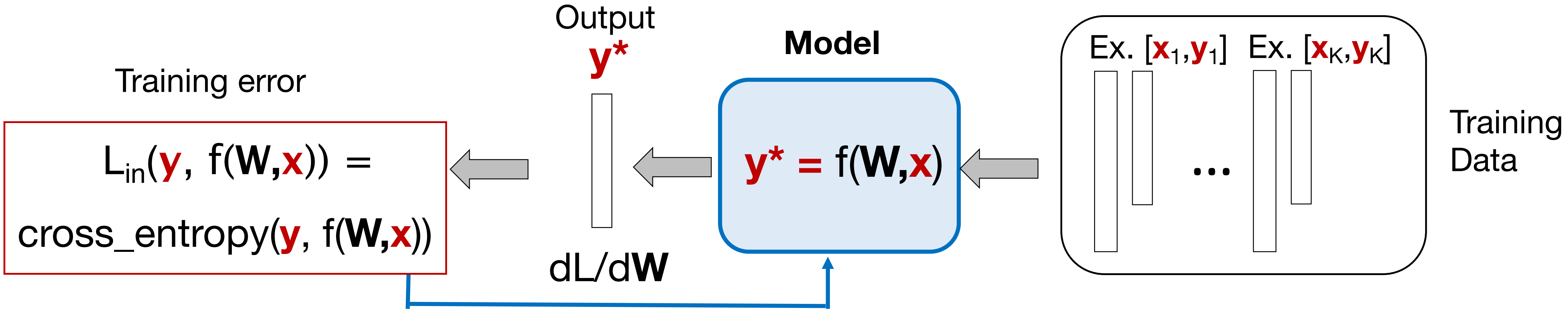
Total number of weights: 2248

Our very basic convolutional neural network



Forward pass: from $\mathbf{x}_i$ and current $\mathbf{W}$'s, find L_{in}

Our very basic convolutional neural network

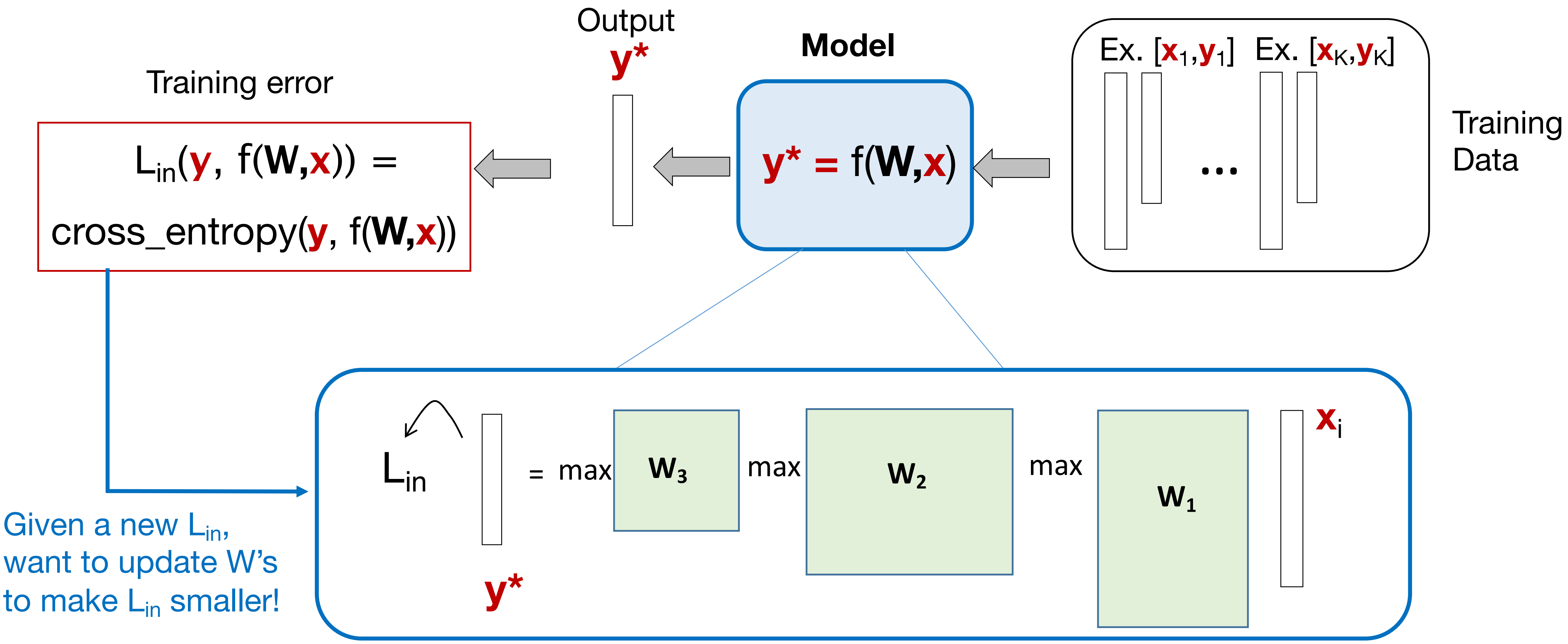


$$L(w) = \frac{1}{N}\sum_{i=1}^{N} \ln\left(1 + e^{-y_i \max(0, \mathbf{W}_3 \max(0, \mathbf{W}_2 \max(0, \mathbf{W}_1 x_i)))}\right)$$

W_1 and W_2 are banded Toeplitz matrices, W_3 is a full matrix

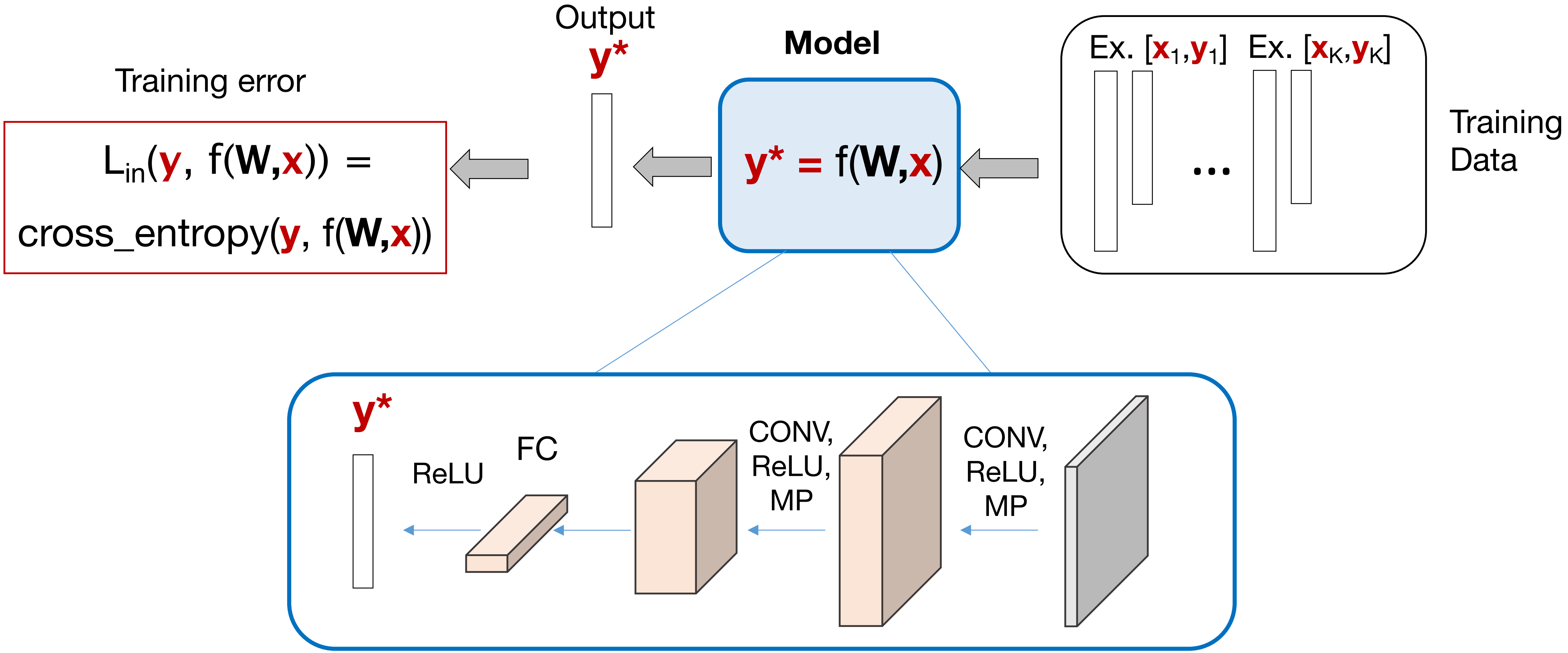
3-layer network

Our very basic convolutional neural network



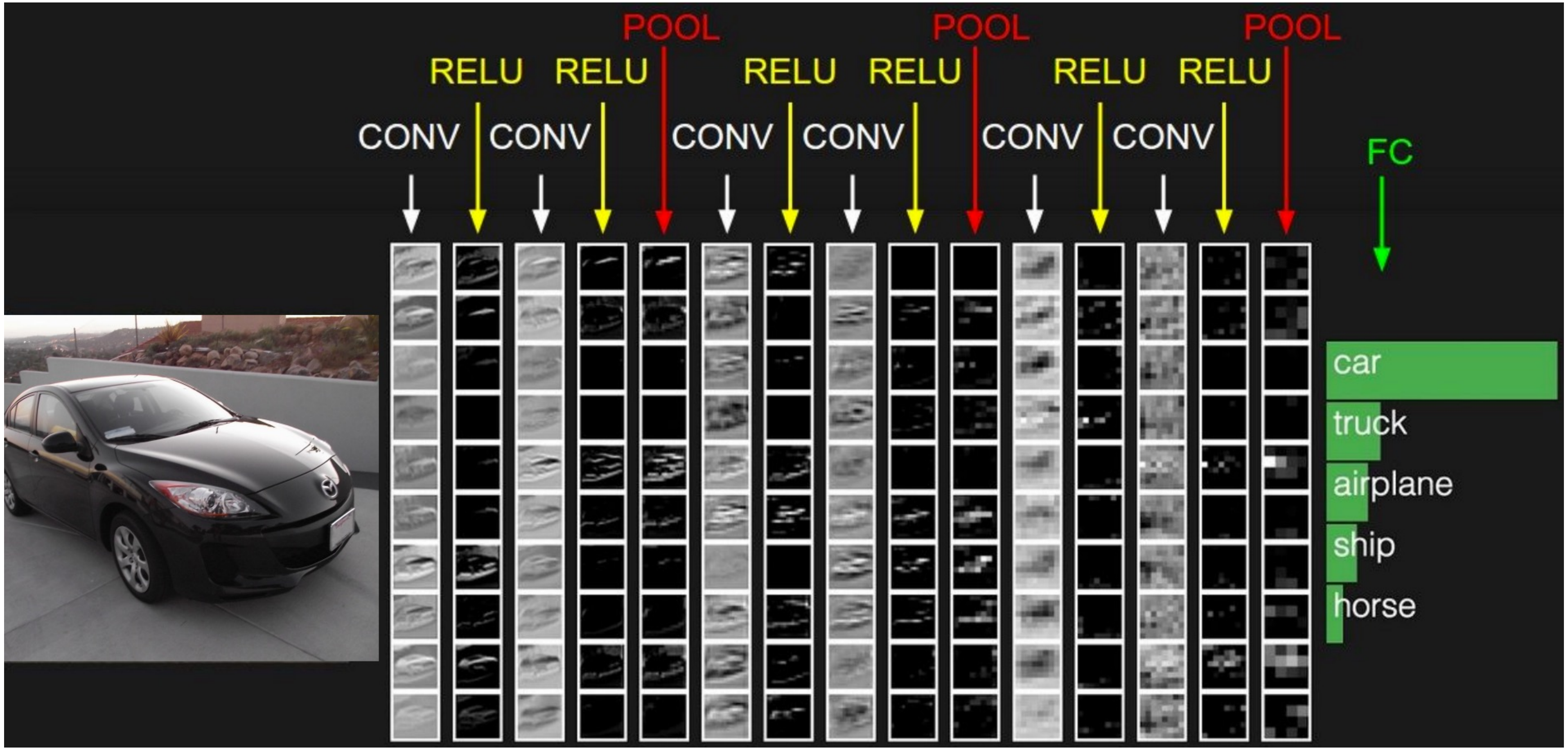
Next class: Gradient descent via L_{in} to update many **W**'s

Our very basic convolutional neural network



3-layer network for 2D images

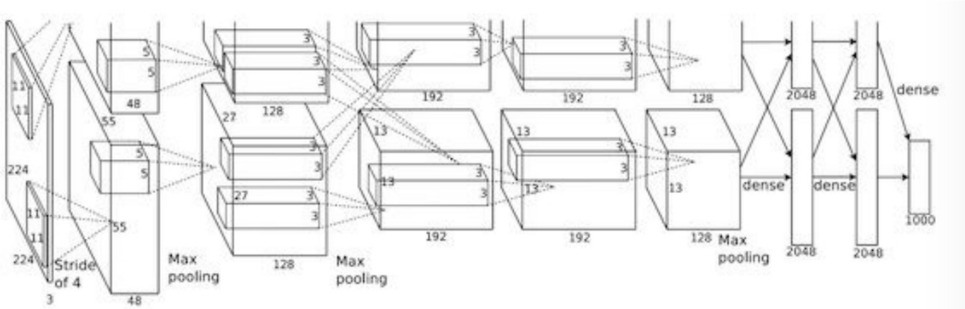
A standard CNN pipeline:



miniAlexNet, 2014

Complex networks are just an extension of this…

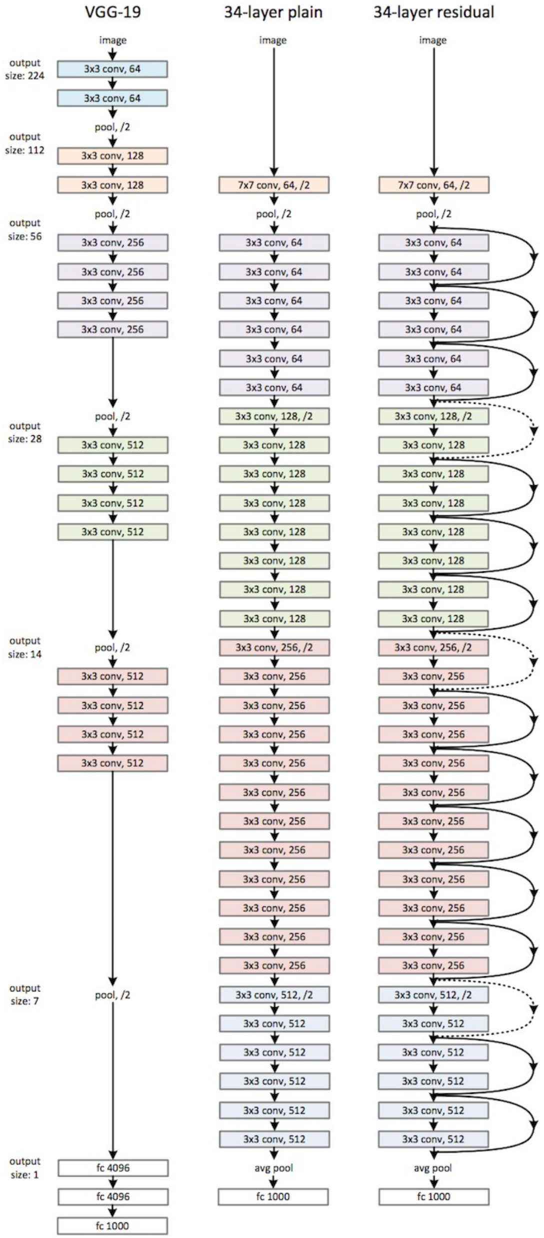
AlexNet (2012)



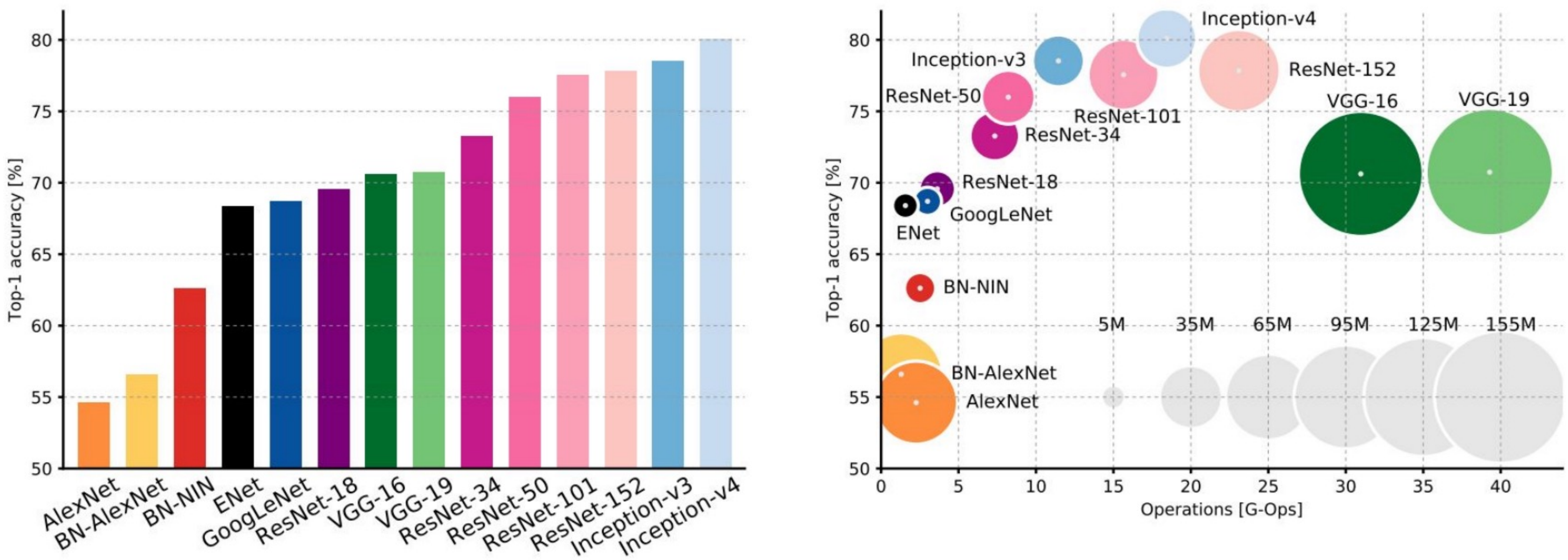
VGG (2014)

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

ResNet (2015)



Comparing complexity...



An Analysis of Deep Neural Network Models for Practical Applications, 2017.

From Stanford CS231n: http://cs231n.stanford.edu/

aging

Break here to give brief introduction to CoLab

basic_tensorflow_eager_example.ipynb

File Edit View Insert Runtime Tools Help

+ Code + Text

```
import numpy as np
import tensorflow as tf
tf.enable_eager_execution()  # if we're using tf version 1.14, then we need to call this command; if using 2.0, then
```

```
[ ]
optimizer = tf.train.GradientDescentOptimizer(learning_rate=.2)  # choose our optimizer and learning rate
x = tf.Variable(2.0)  # define a variable to optimize, with an initial value of 2

for i in range(10):  # iterative optimization loop
  with tf.GradientTape() as tape:  # gradient tape keeps track of the gradients associated with all the operations
    # define our very simple minimization problem:
    loss = x ** 2  # we're going to minimize x^2, which occurs at x=0

    # compute and apply gradients:
    gradient = tape.gradient(loss, x)
    optimizer.apply_gradients([(gradient, x)])

    # print out current iteration and loss value:
    print(i, 'loss = ' + str(loss.numpy()), 'x = ' + str(x.numpy()))
```

```
0 loss = 4.0 x = 1.2
1 loss = 1.44 x = 0.72
2 loss = 0.5184 x = 0.432
3 loss = 0.186624 x = 0.2592
4 loss = 0.06718464 x = 0.15552
5 loss = 0.024186473 x = 0.093312
6 loss = 0.008707129 x = 0.0559872
7 loss = 0.0031345668 x = 0.03359232
8 loss = 0.001128444 x = 0.020155393
9 loss = 0.00040623985 x = 0.012093236
```