Deep Learning Agenda

11:15-11:40	– DL1: Intro to Neural Networks and CNNs
11:40 - 12:10	– DL2: MNIST CNN Hands-on tutorial
12:10 - 12:25	– Break
12:25 - 1:10	– DL3: CNN Transfer Learning & Hands-On
1:10 - 1:30	– DL4: Object Detection
1:30 - 1:55	– DL5: Other Deep Learning Architectures
1:55 - 2:00	– Wrap-Up

SDSC Summer Institute 2020

DL1 – Introduction to Neural Networks, Convolution Networks (CNNs)

Paul Rodriguez

08/06/20

Location: Breakout Room

Gitter (session support): Breakout Room

Gitter (general system support): Help Desk

Outline

- What is Deep Learning
- Neural Networks
- Convolution Neural Networks
- Tutorial
- Summary

Deep Learning

- **3 characterizations:**

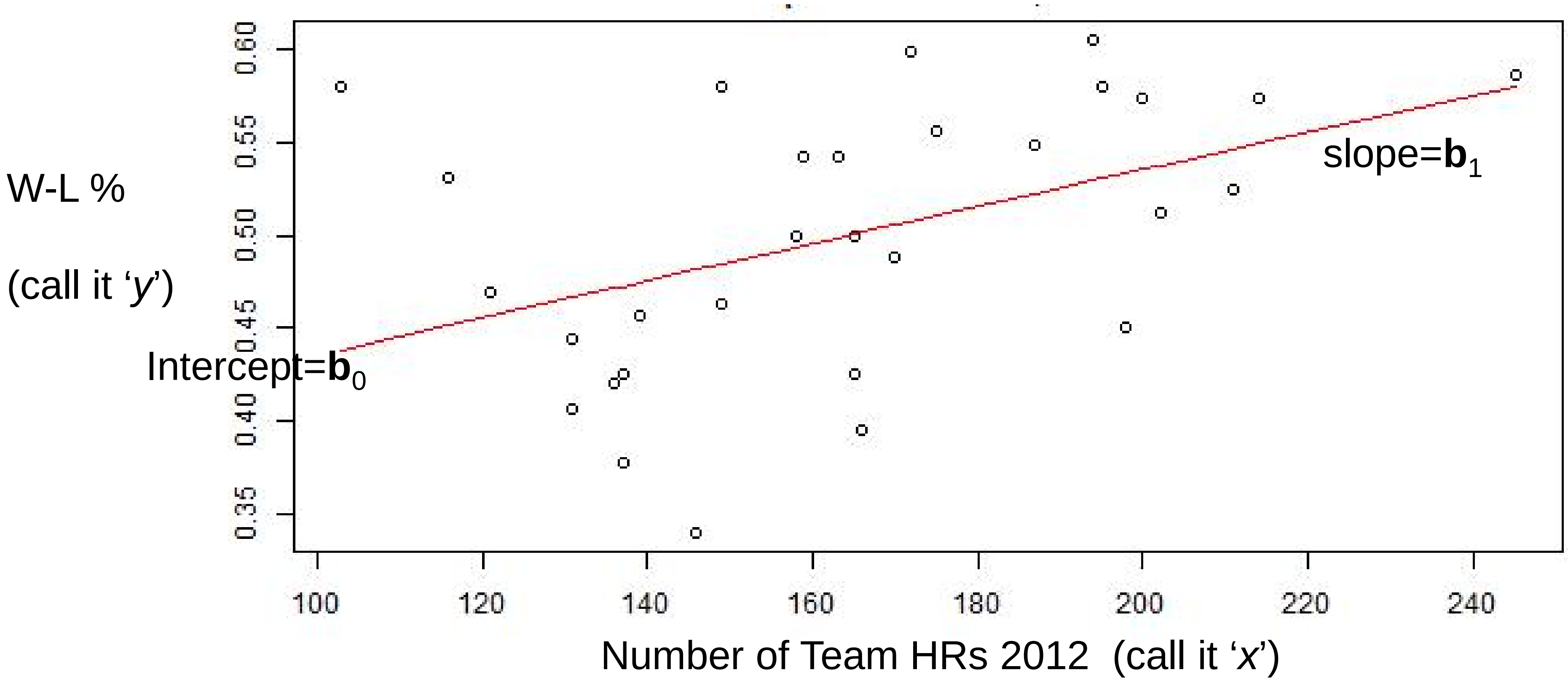
  1. Learning complicated interactions about input

  2. Discovering complex non-linear feature transformations

  3. Using neural networks with many layers

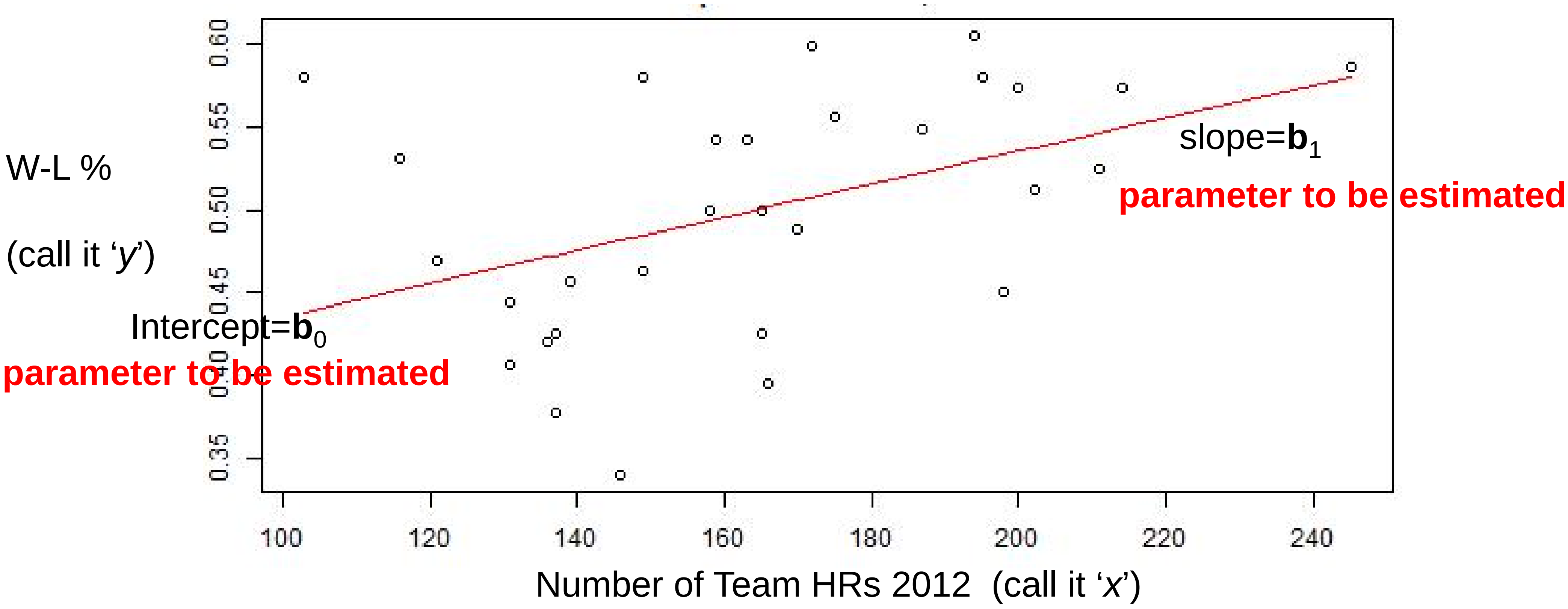
Recall Linear Regression is Fitting a Line

the Model: $y = f(x, b) = bo * 1 + b_1 * x$



Recall Linear Regression is Fitting a Line

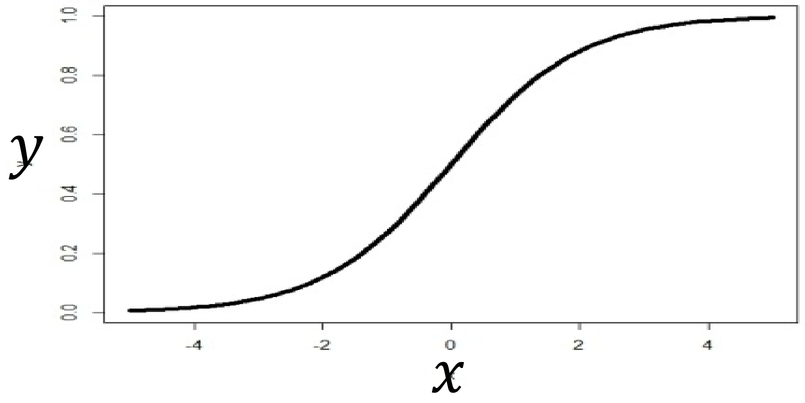
$$\textbf{the Model}: y = f(x, b) = bo * 1 + b_1 * x$$



to get neural network:

1 Consider nonlinear Logistic Function

$$f(x_i) = \frac{1}{1 + exp^{(-(b+wx))}}$$



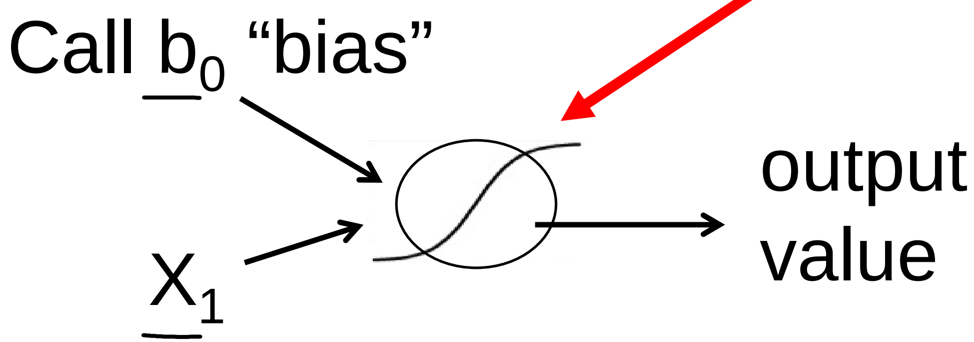
$b = 0 \;\;,\; w_1 = 1$

2 Now use a graphical network description

$$f(x_i) = \frac{1}{1 + exp^{(-(b+wx))}}$$

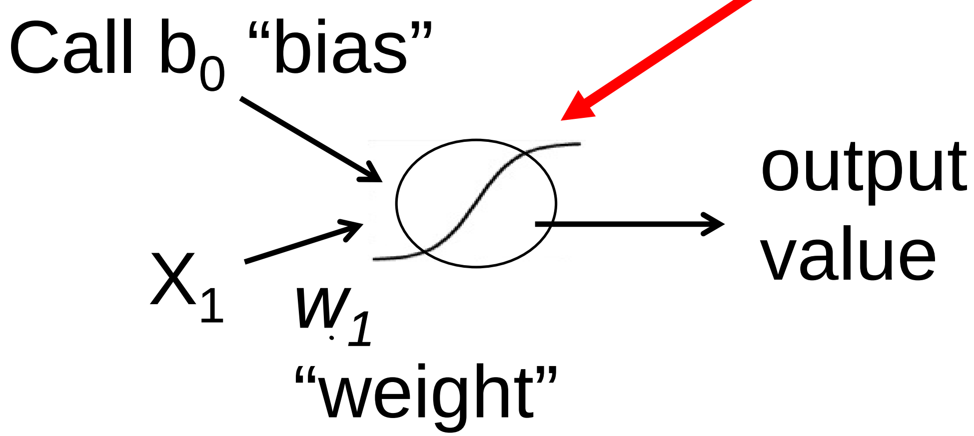
2 Now use a graphical network description

$$f(x_i) = \frac{1}{1 + exp^{(-(b+wx))}}$$



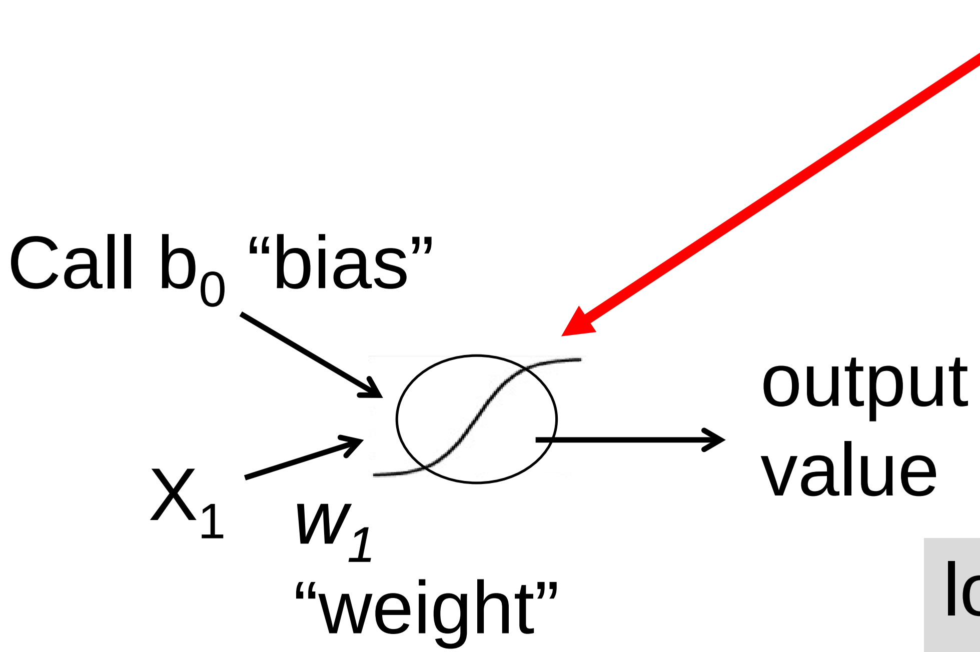
2 Now use a graphical network description

$$f(x_i) = \frac{1}{1 + exp^{(-(b+wx))}}$$



2 Now use a graphical network description

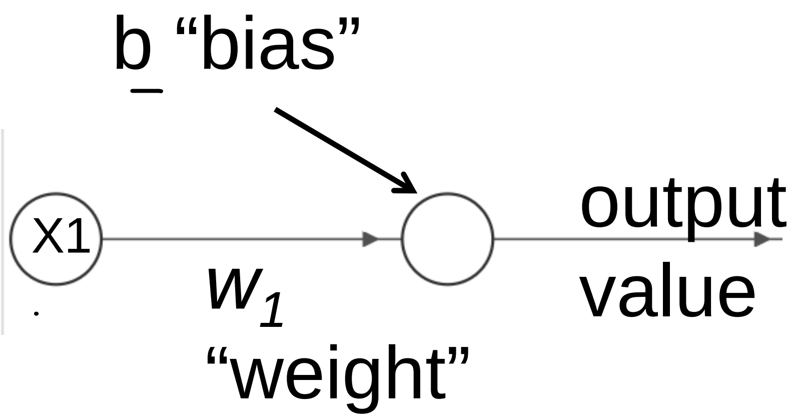
$$f(x_i) = \frac{1}{1 + exp^{(-(b+wx))}}$$

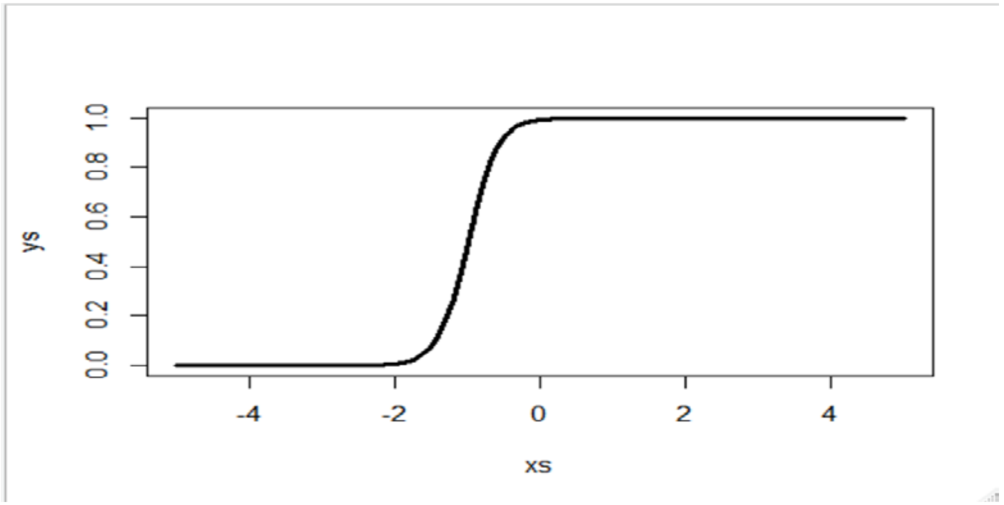


logistic function will transform input to output – call it the 'activation' function

How does changing parameters affect function?

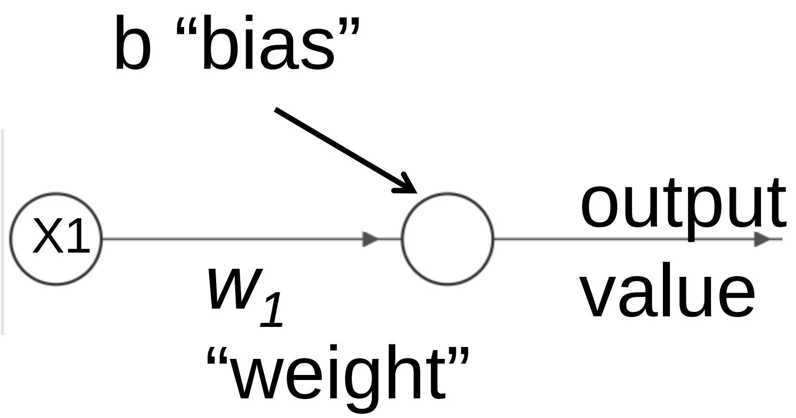
$$f(x_i) = \frac{1}{1 + exp^{(-(b+wx))}}$$

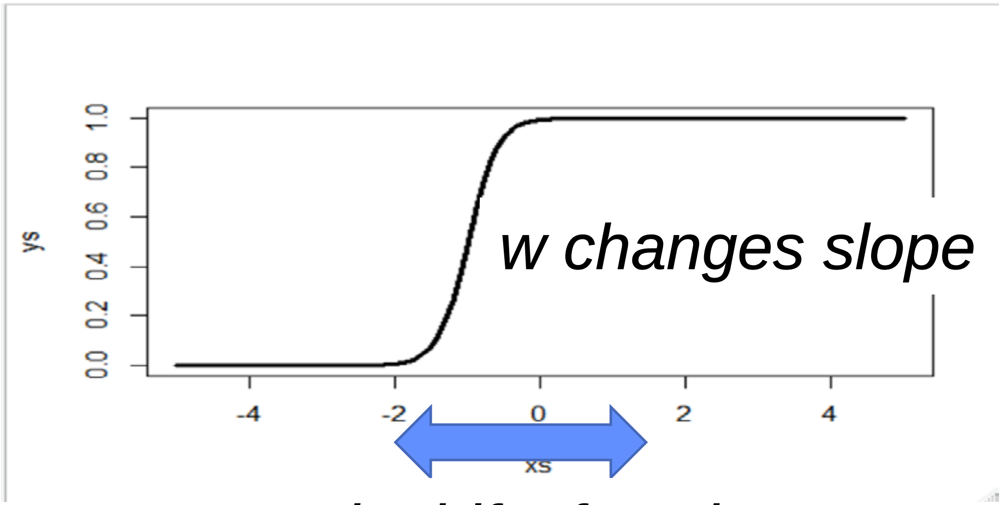




How does changing parameters affect function?

$$f(x_i) = \frac{1}{1 + exp^{(-(b+wx))}}$$

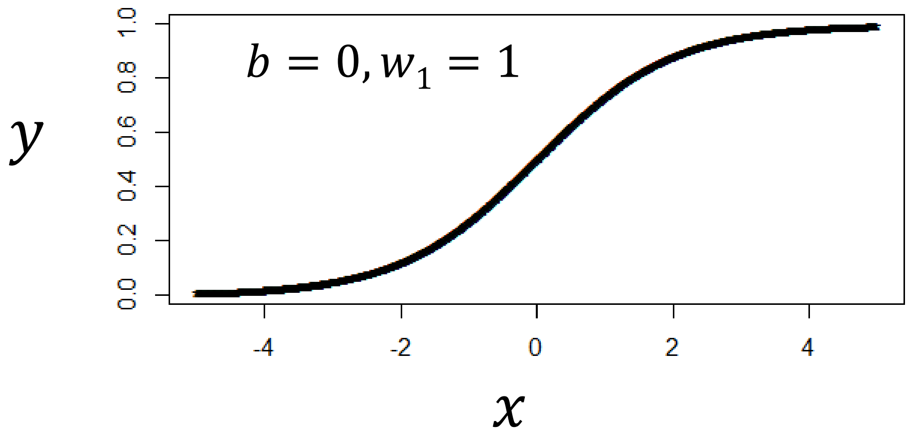




b shifts function

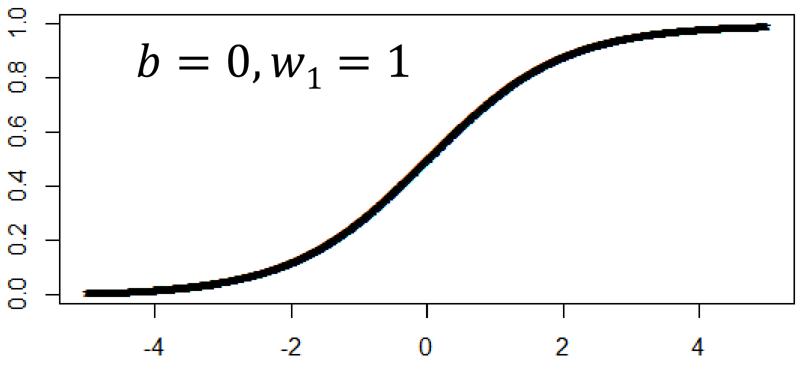
Logistic function w/various weights

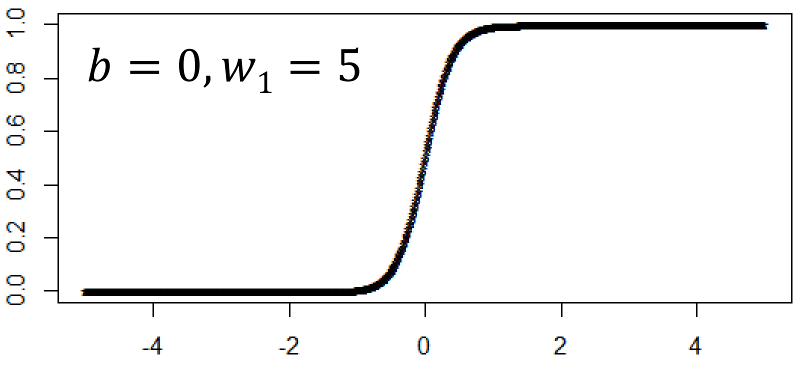
$$for\ y = 1/(1+exp(-(b+w_1 * x))$$



Logistic function w/various weights

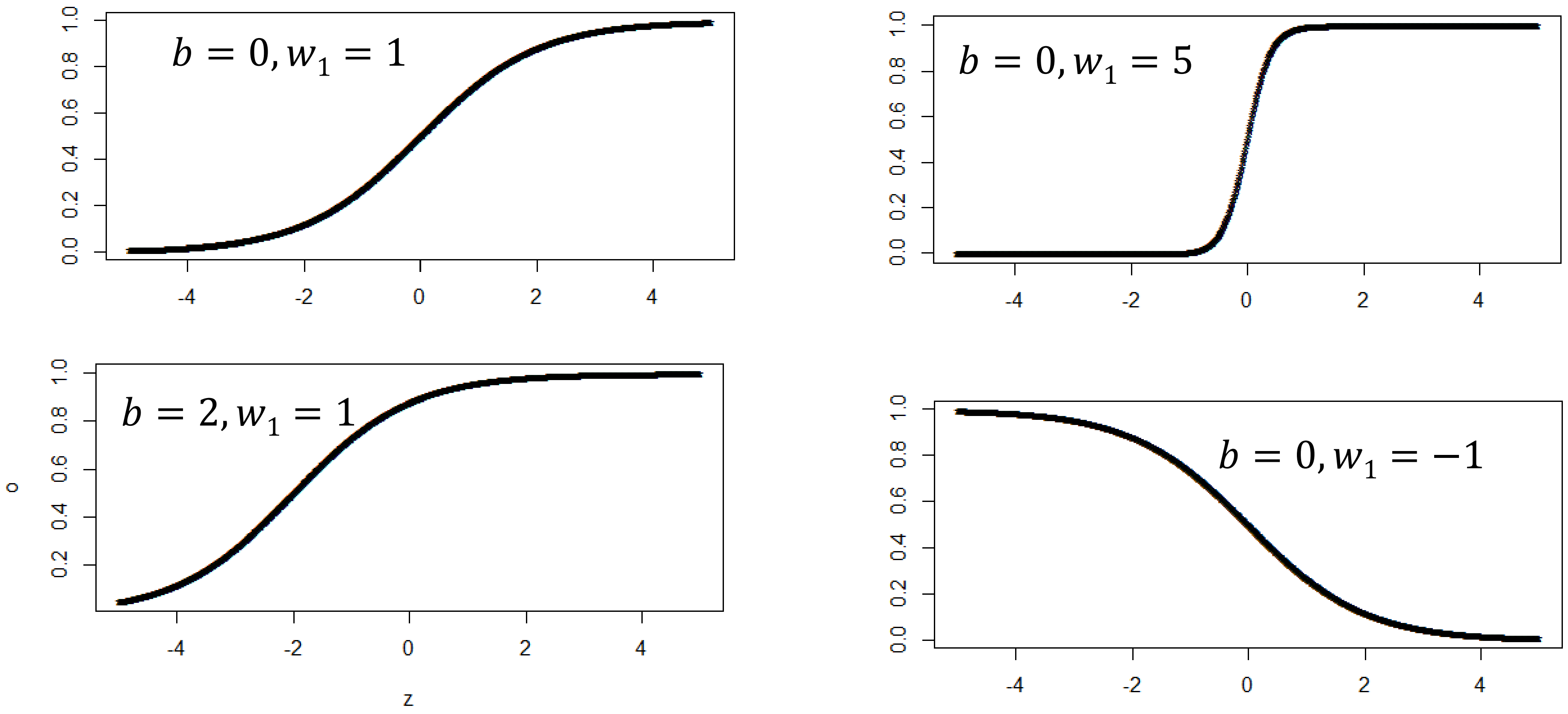
$$for\ y = 1/(1+exp(-(b+w_1*x)))$$



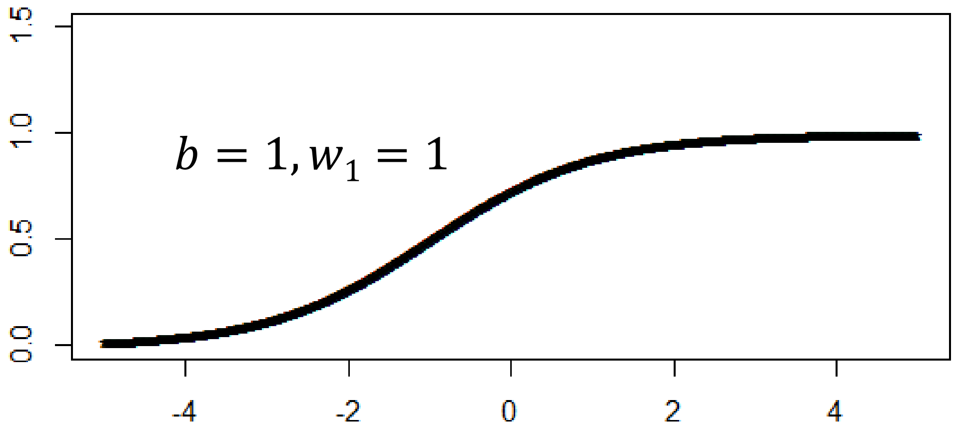


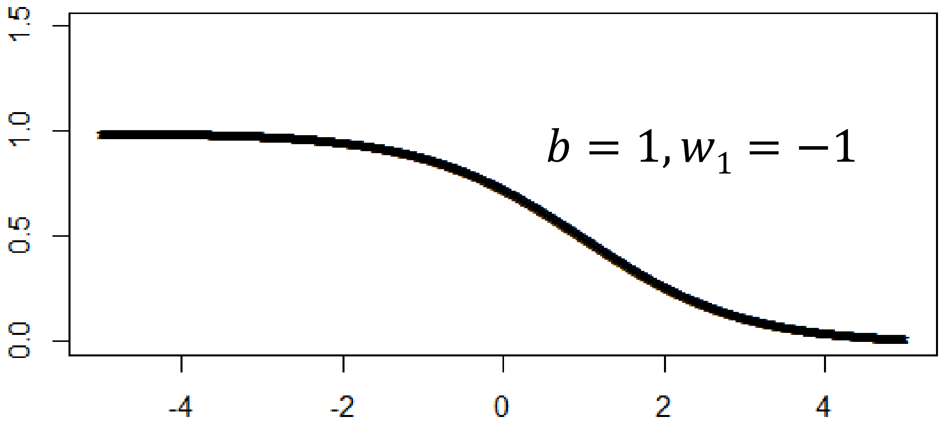
Logistic function w/various weights

$$for\ y = 1/(1+exp(-(b+w_1*x))$$



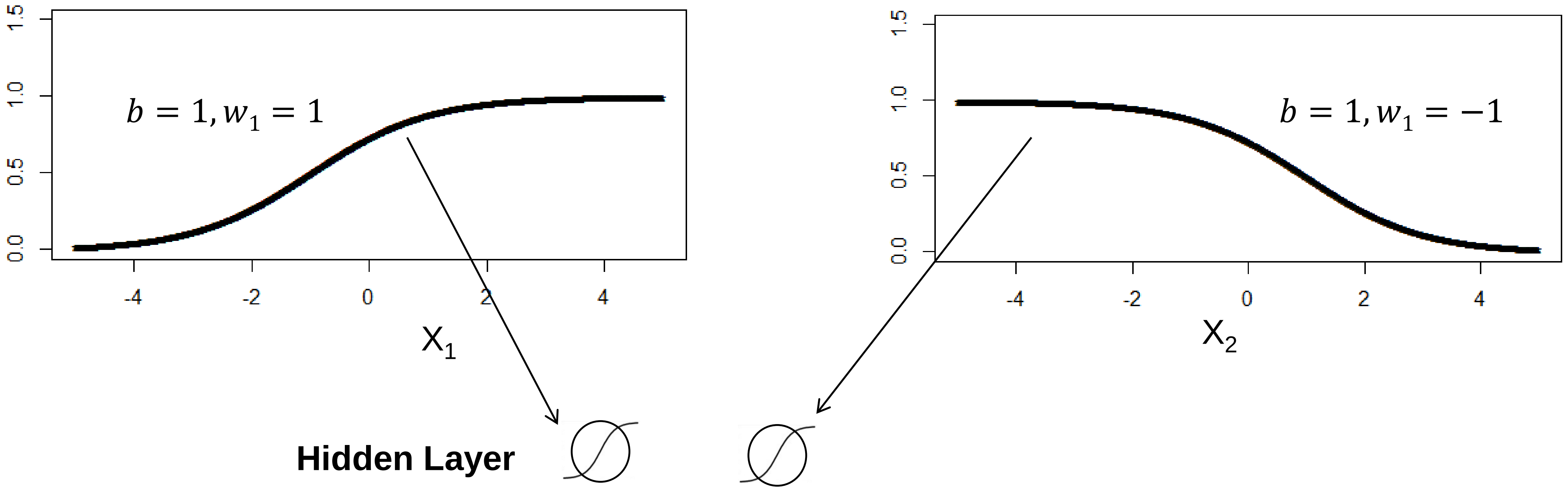
So combinations are highly flexible and nonlinear



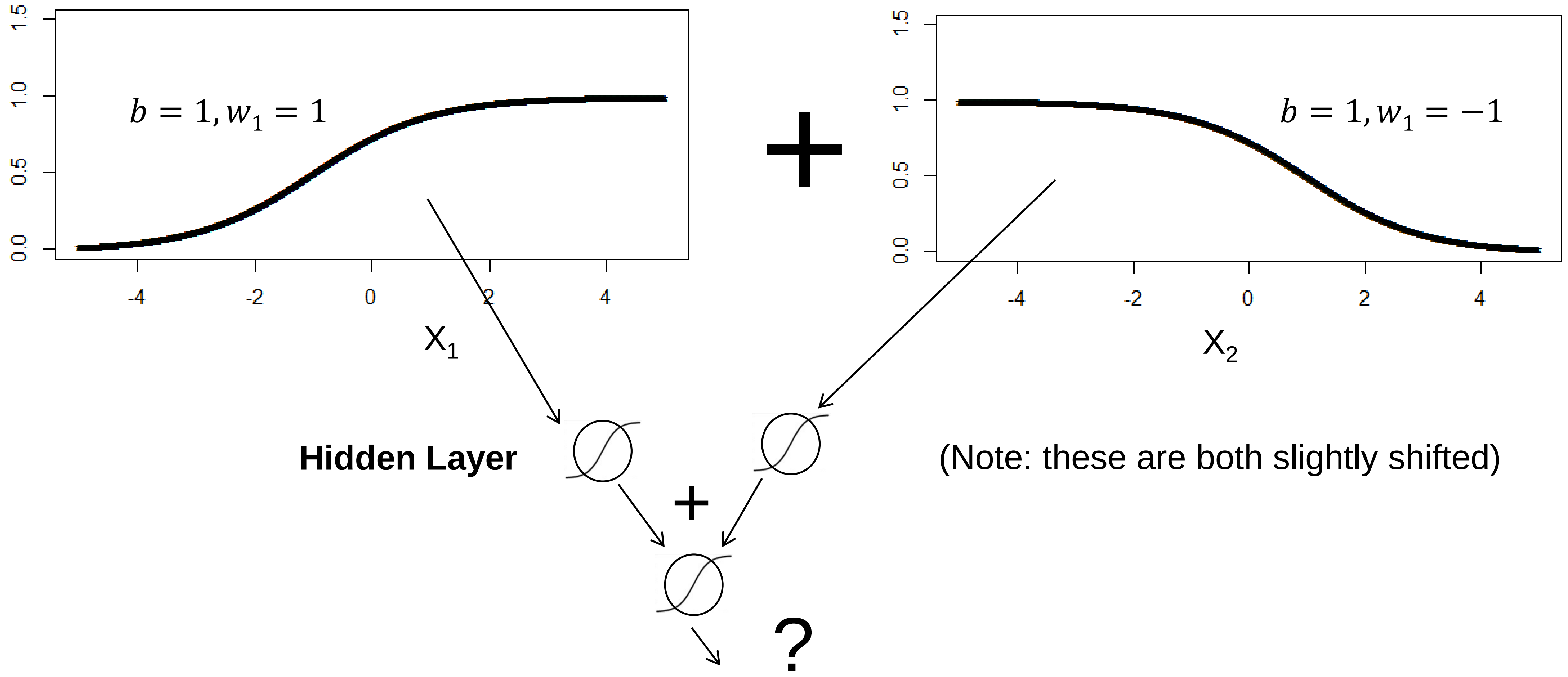


(Note: these are both slightly shifted)

So combinations are highly flexible and nonlinear

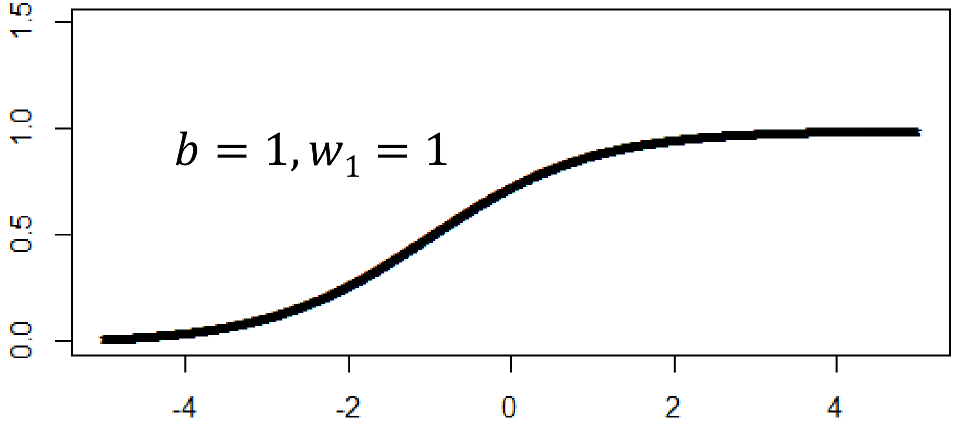


So combinations are highly flexible and nonlinear

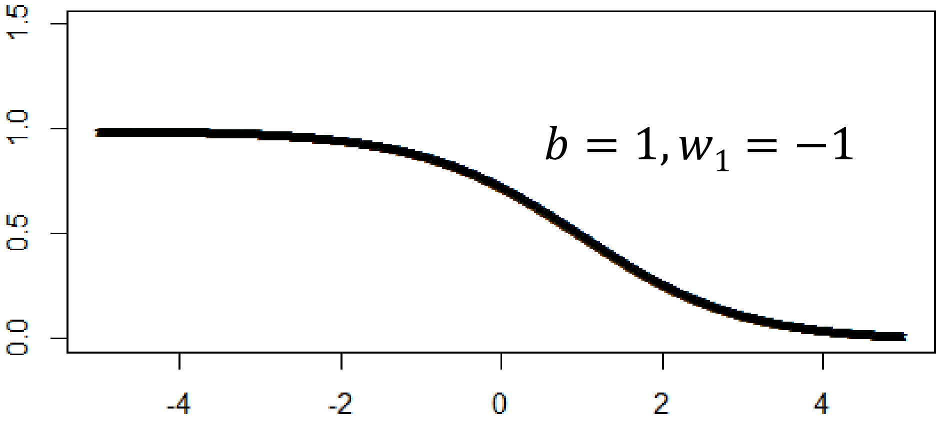


(Note: these are both slightly shifted)

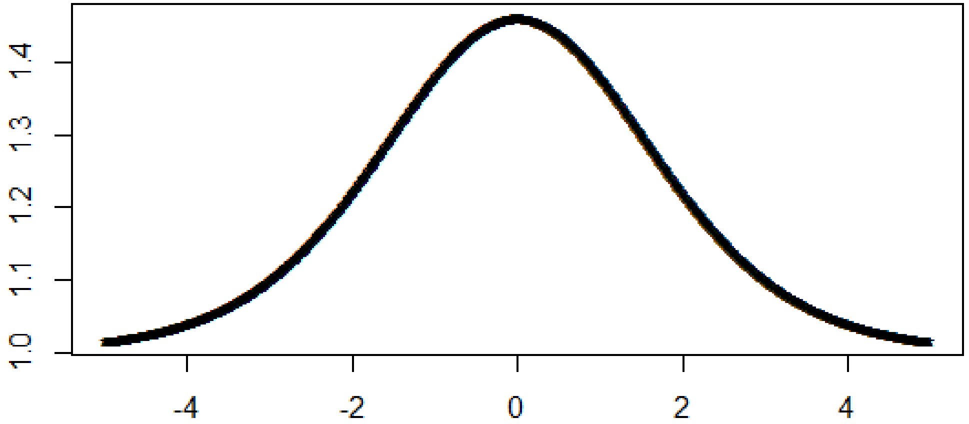
So combinations are highly flexible and nonlinear



+

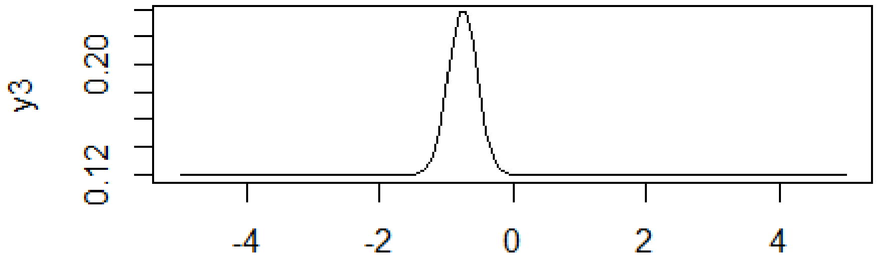


=

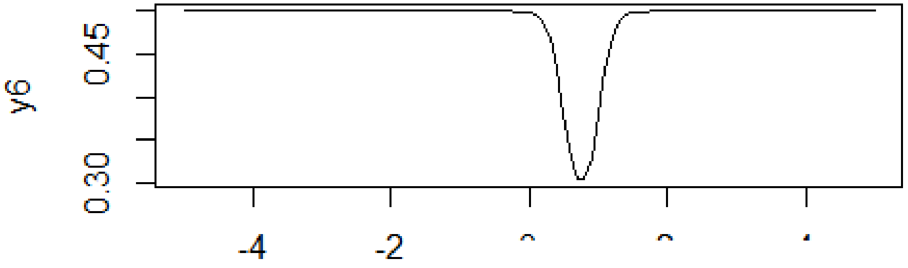


Higher level function combinations

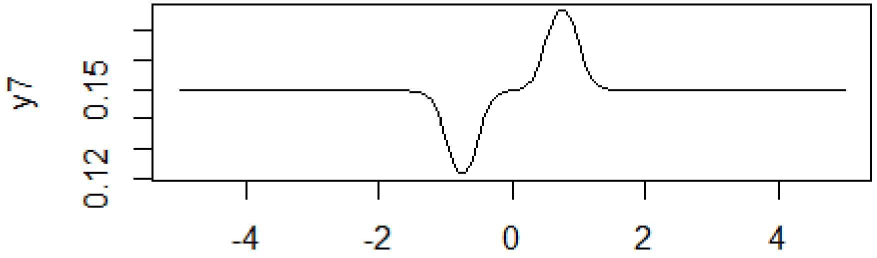
```
x=seq(-5,5,.1)
y1=1/(1+exp(10+   10*x))
y2=1/(1+exp(-5+(-10)*x))
y3=1/(1+exp(1+1*y1+1*y2))
plot(x,y3,type="l")
```




```
y4=1/(1+exp(10+ (-10)*x))
y5=1/(1+exp(-5+(10)*x))
y6=1/(1+exp(1-1*y4-1*y5))
plot(x,y6,type="l")
```

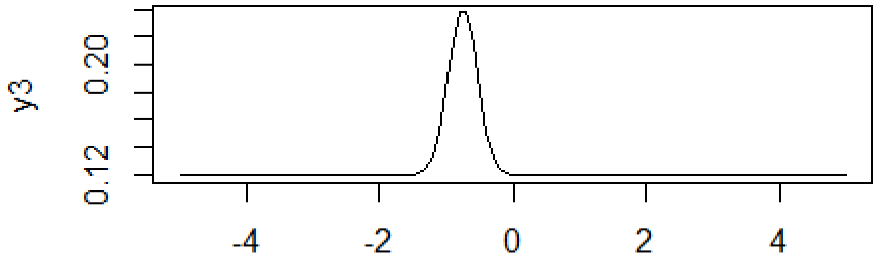



```
y7=1/(1+exp(1+2*y3+1*y6))
plot(x,y7,type="l")
```



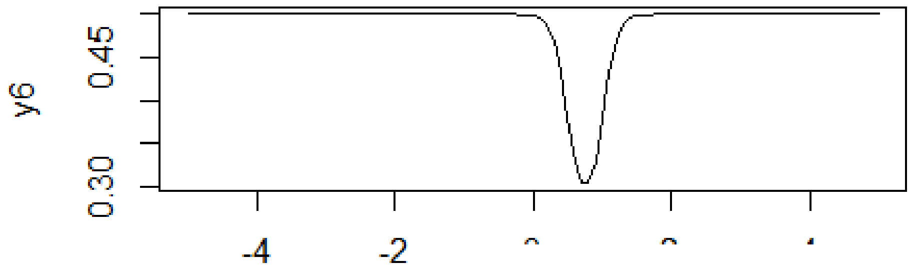
Higher level function combinations

```
x=seq(-5,5,.1)
y1=1/(1+exp(10+   10*x))
y2=1/(1+exp(-5+(-10)*x))
y3=1/(1+exp(1+1*y1+1*y2))
plot(x,y3,type="l")
```

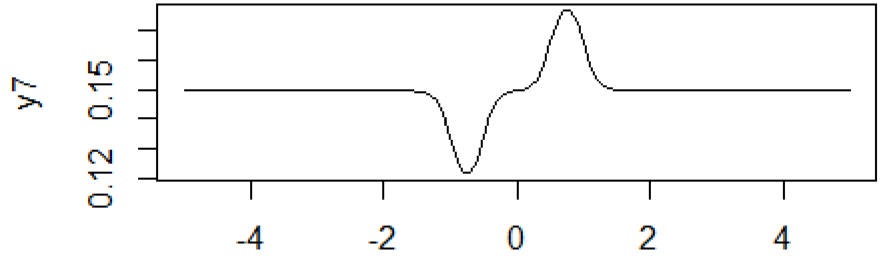


Multiple layer networks can represent any logical or real-valued functions (unbiased, but potential to overfit)

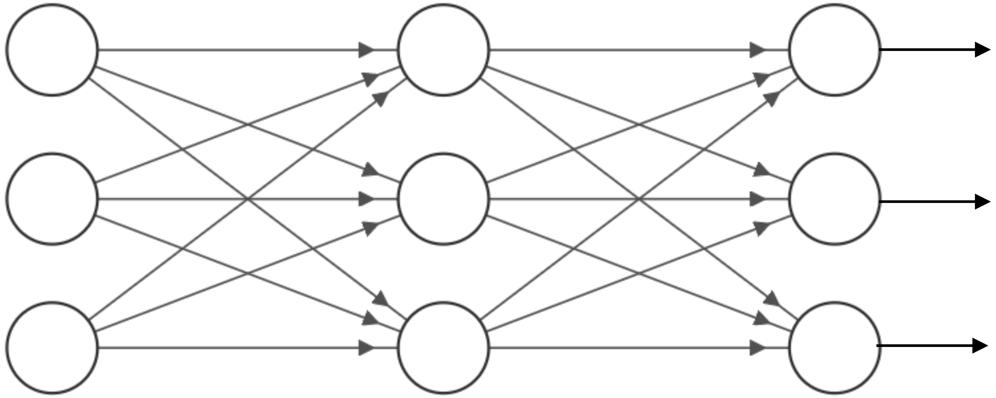
```
y4=1/(1+exp(10+ (-10)*x))
y5=1/(1+exp(-5+(10)*x))
y6=1/(1+exp(1-1*y4-1*y5))
plot(x,y6,type="l")
```




```
y7=1/(1+exp(1+2*y3+1*y6))
plot(x,y7,type="l")
```



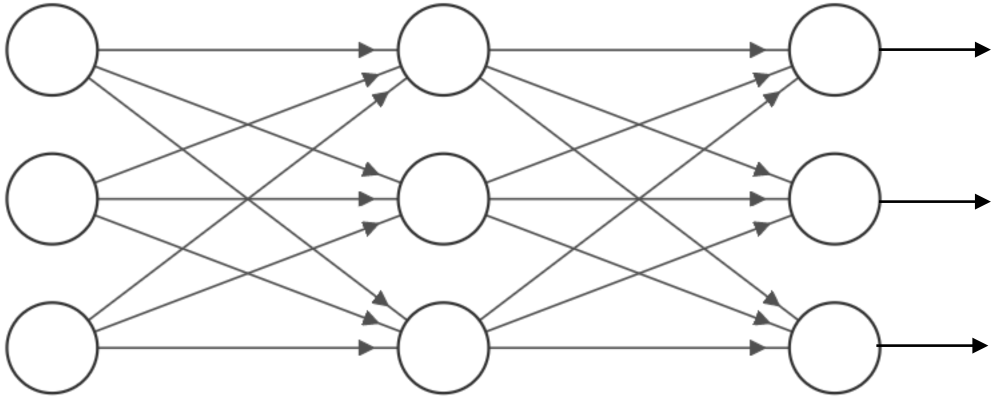
More generally we can add layers and nodes



More generally we can add layers and nodes

Multilayer Perceptron

1 Input layer 1 Hidden layer 1 Output

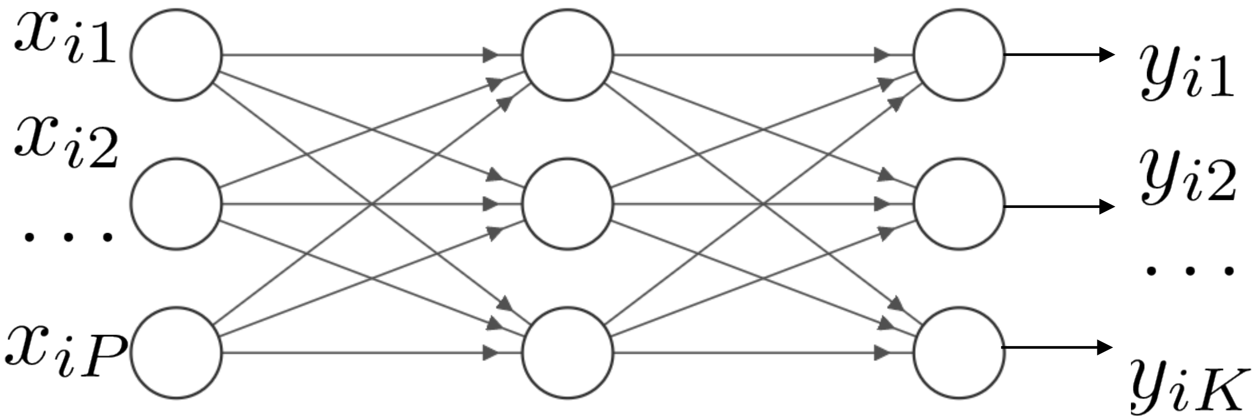


More generally we can add layers and nodes

Multilayer Perceptron

1 Input layer　　1 Hidden layer　　1 Output

P input units



K output units

More generally we can add layers and nodes

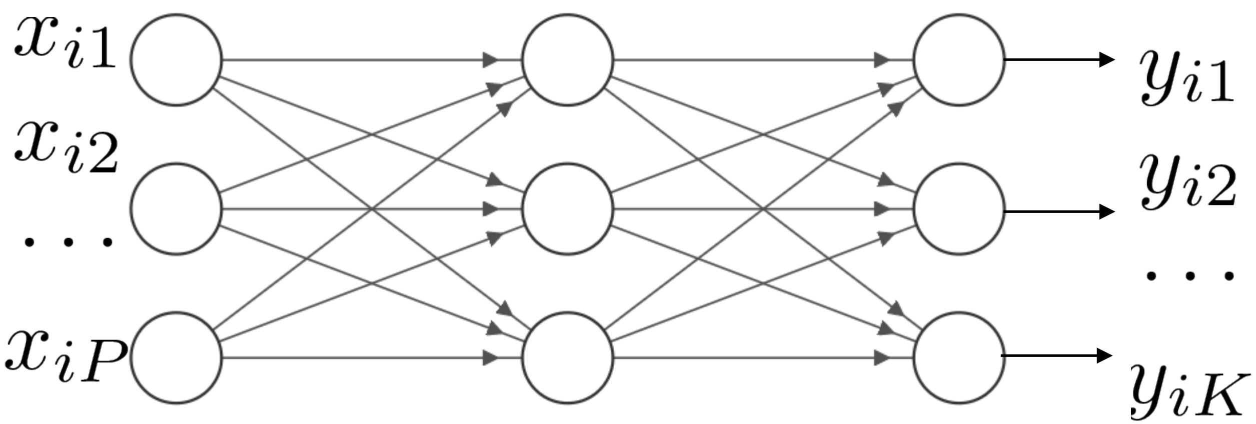
Multilayer Perceptron

1 Input layer

1 Hidden layer

1 Output

input units



output units

hidden units

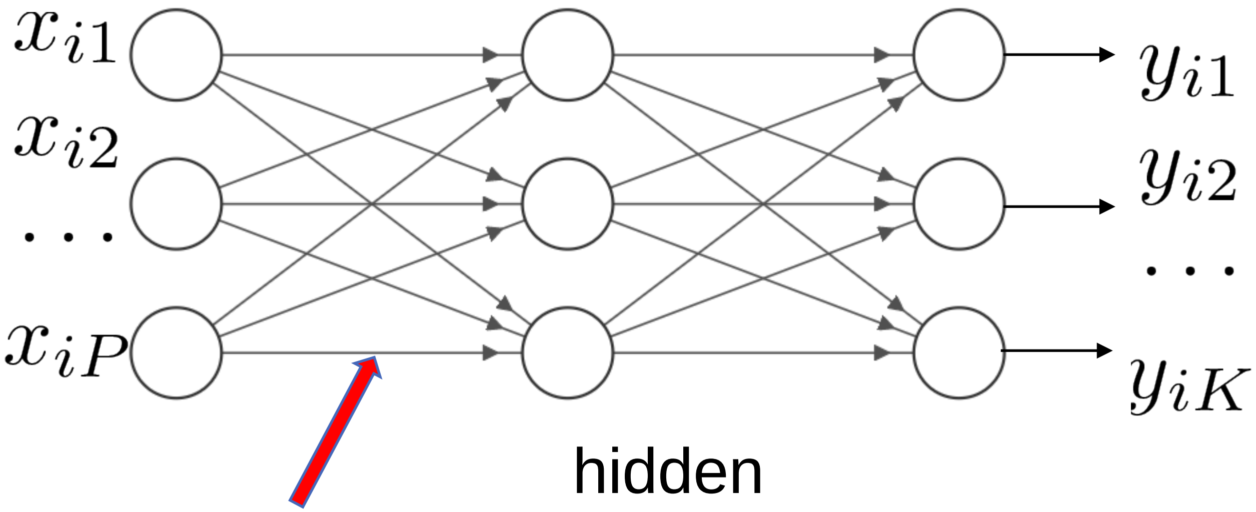
All units have a bias parameter (not shown)

More generally we can add layers and nodes

Multilayer Perceptron

1 Input layer | 1 Hidden layer | 1 Output

input units



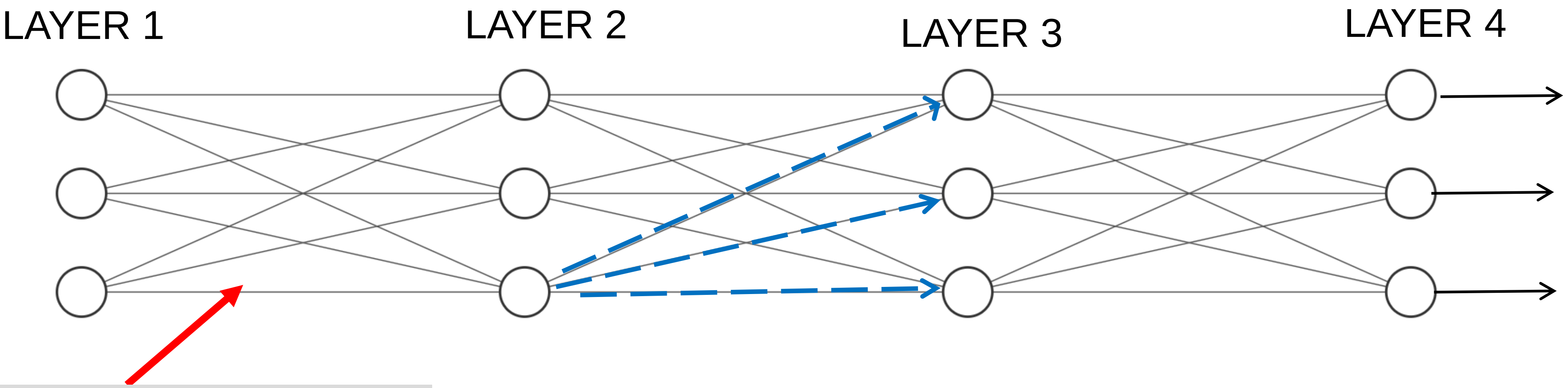
output units

1 weight parameter for each connection

hidden units

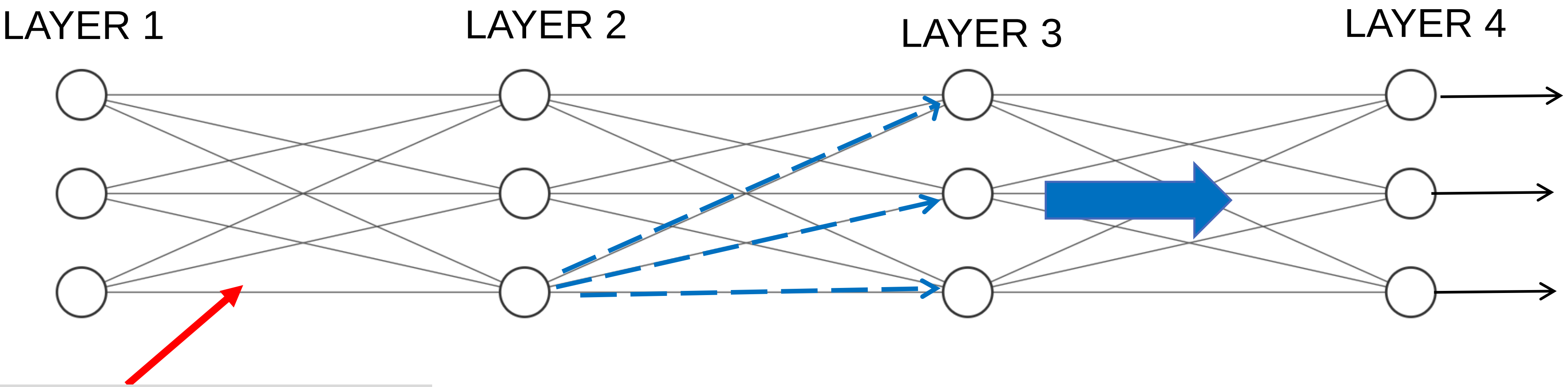
All units have a bias parameter (not shown)

Propagation



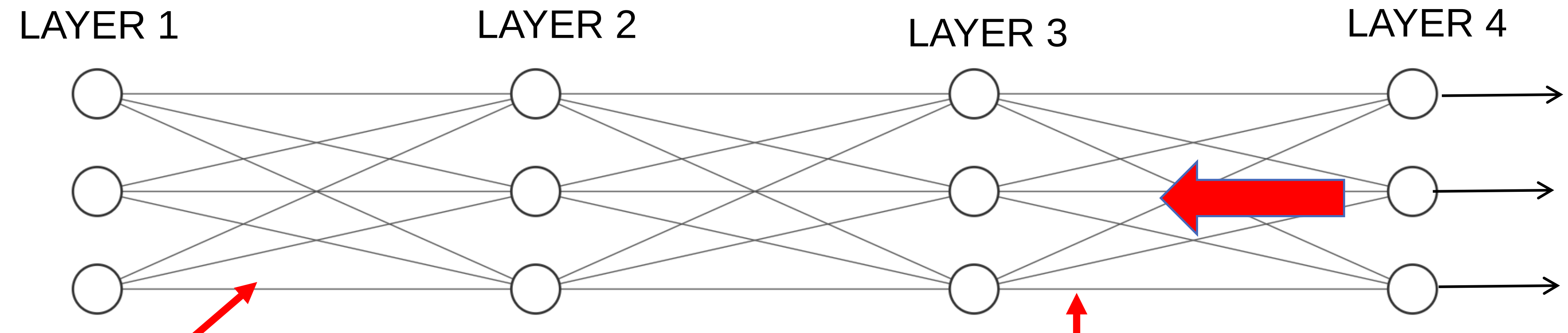
Forward Propagation
An input weight here contributes to all hidden units in layer 3

Propagation



Forward Propagation
An input weight here contributes to all hidden units in layer 3, and all outputs in layer 4

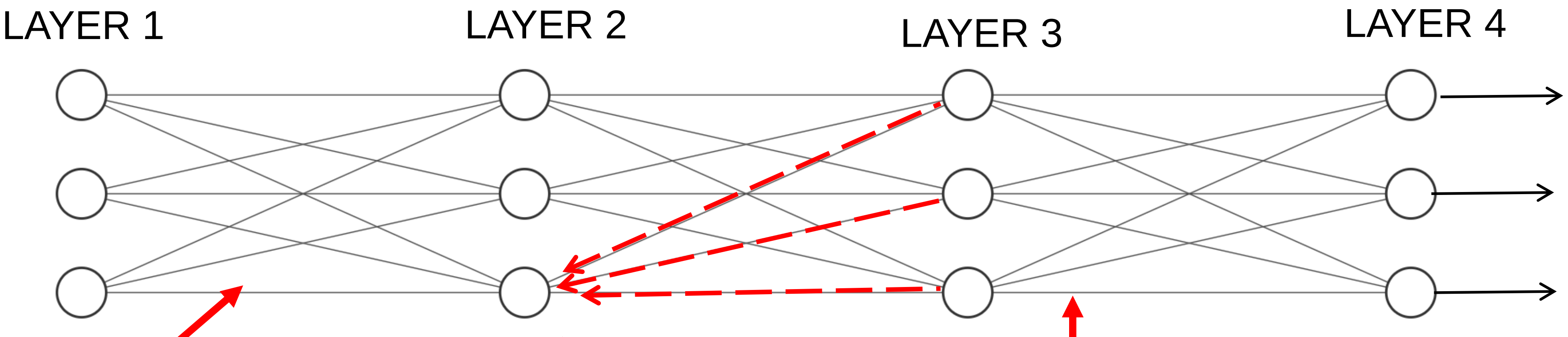
Propagation



Forward Propagation
An input weight here contributes to all hidden units in layer 3, and all outputs in layer 4

Backpropagation:
Take derivatives of error (dE/dY) at output units to update weights by gradient descent

Propagation



Forward Propagation
An input weight here contributes to all hidden units in layer 3, and all outputs in layer 4

Backpropagation:
Take derivatives of error (dE/dY) at output units to update weights by gradient descent

For more layers just need keep extending chain rule and keep passing back error information (dE/dY dY/dH_3 dH_3/dH_2 etc…)

The Neural Network Algorithm

LOOP until stopping criterion:

FORWARD PROPAGATION: apply input data x_i , calculate all node activations

BACKWARD PROPAGATION: calculate all error derivatives to ***minimize Loss***

UPDATE WEIGHTS: add in derivatives (stochastic gradient descent)

STOP: when validation error reaches minimum or converges

terminology and cheat sheet on output activations:

Type of Problem	Y outputs	Output Activation Function (this gives a SCORE= $\hat{Y}$)	Output PREDICTION (what you decide to predict)	Output Loss Function	Evaluative Measure
Regression: map into to real valued prediction	if $Y \in (-\infty, +\infty)^K$	$\hat{Y} = XW$	$\hat{Y}$	Sum Squared Error (SSE)	Root Mean Squared Error (RMSE)
Multivariate output of 0's and 1's	if $Y \in [0,1]^K$	$\hat{Y} = \frac{1}{1+exp^{-(XW)}}$	1 or 0	SSE	RMSE
Binary Classification	if $Y \in \{0,1\}$	$\hat{Y} = \frac{1}{1+exp^{-(XW)}}$	A probability given by $\hat{Y}$ $P(y=1\|x)$	Cross Entropy $L = -ylog(\hat{y}) - (1-y)(log(\hat{y})$	Accuracy, ROC
Multiclassification	if $Y \in \{0,1\}^K$	$\hat{Y}_k = \frac{exp^{-(XW_k)}}{\sum_k exp^{-(XW_k)}}$	Max class	Cross Entropy $L = -\sum_k y_k log(\hat{y_k})$	Accuracy

Summary:

Pro:

- Multilayer Perceptron, and Neural Networks in general, are flexible powerful learners
- Hidden layers learn a nonlinear transformation of input

Summary:

Pro:

- Multilayer Perceptron, and Neural Networks in general, are flexible powerful learners
- Hidden layers learn a nonlinear transformation of input

Con:

- Lots of parameters
- Hard to interpret
- Needs more data

pause

onto Convolution Networks

Image features

- **MNIST - A database of handwritten printed digits**

  (National Inst. of Standards and Technology)

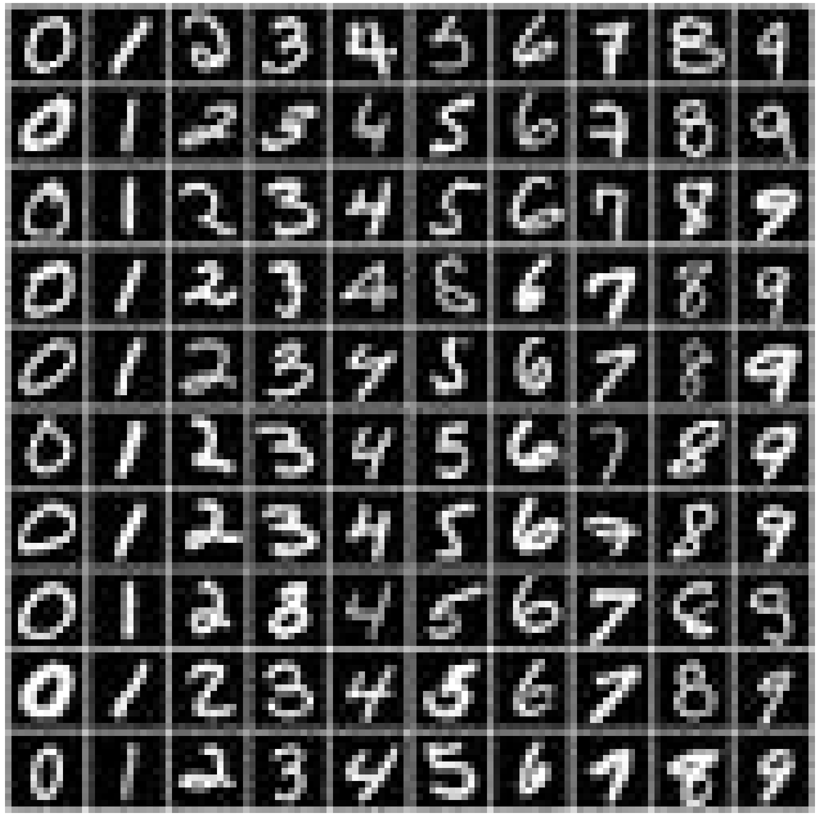
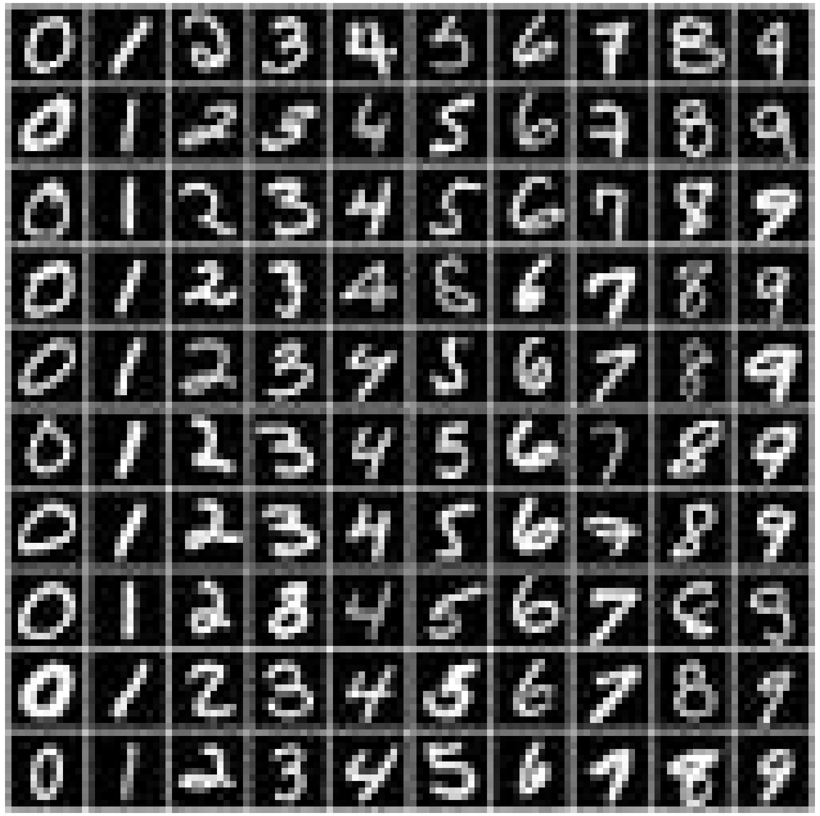


Image features

- **MNIST - A database of handwritten printed digits**

  (National Inst. of Standards and Technology)

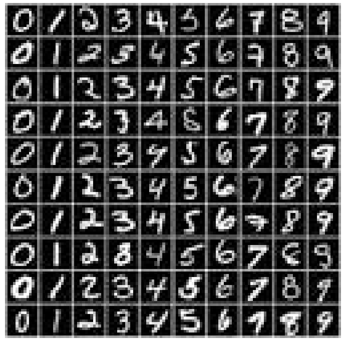


How to classify digits?

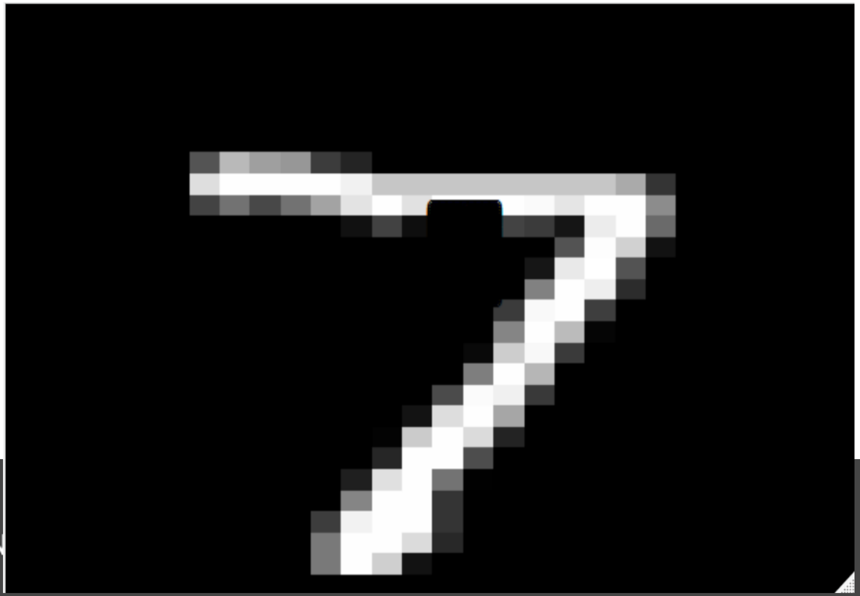
Image features

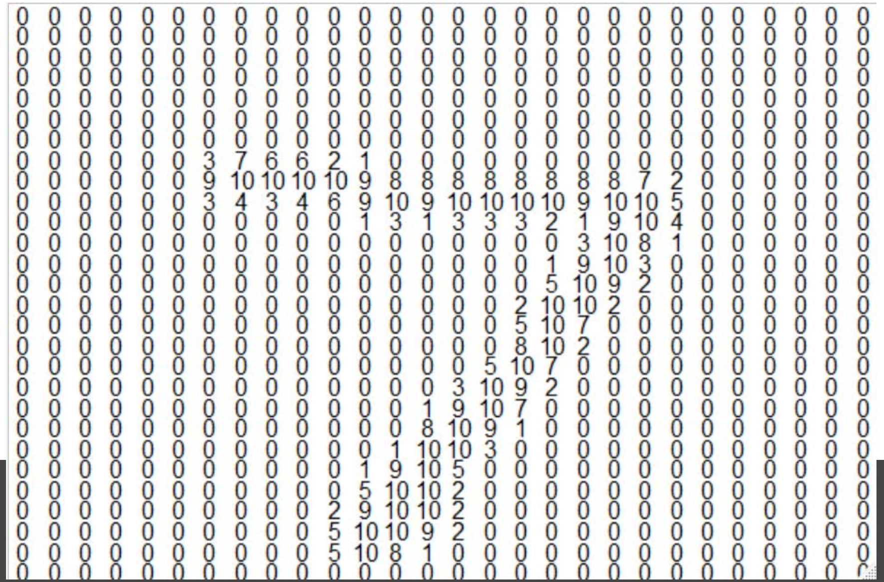
- **MNIST - A database of handwritten printed digits**

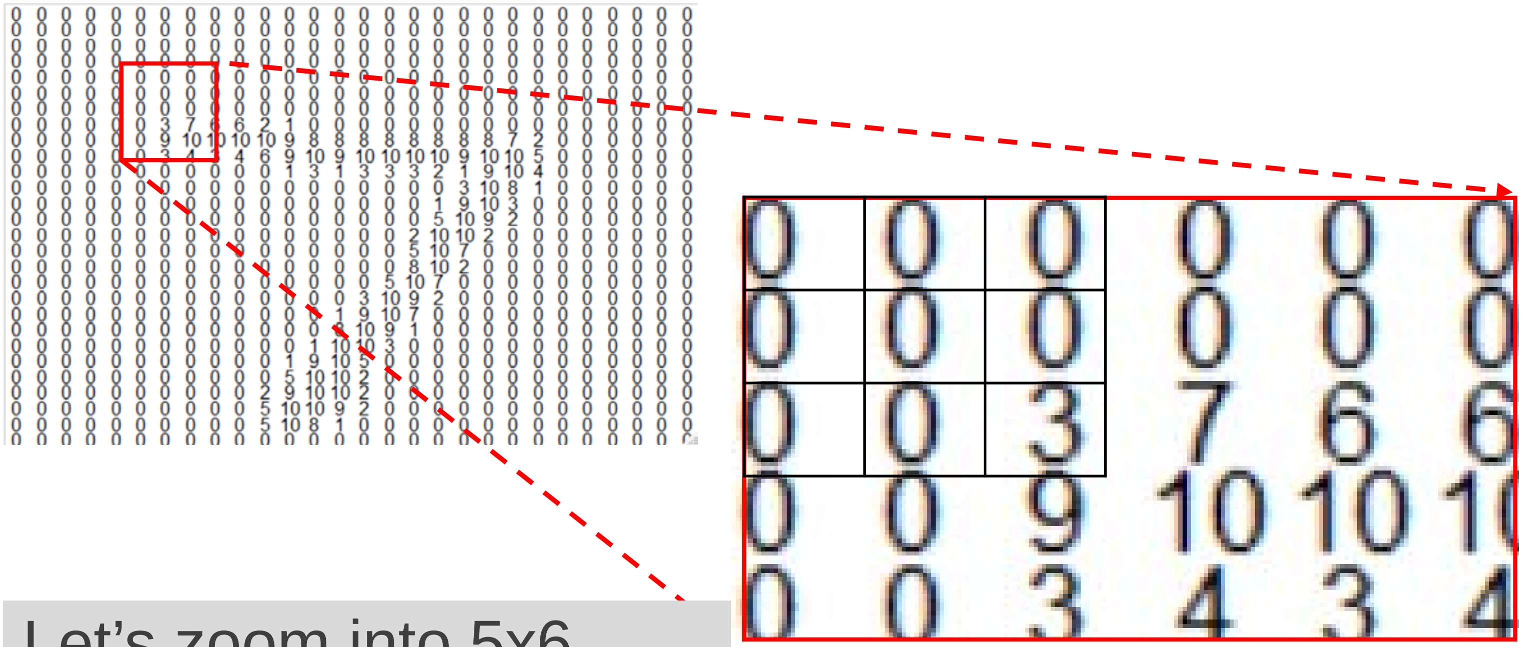
  (National Inst. of Standards and Technology)



How to classify digits?

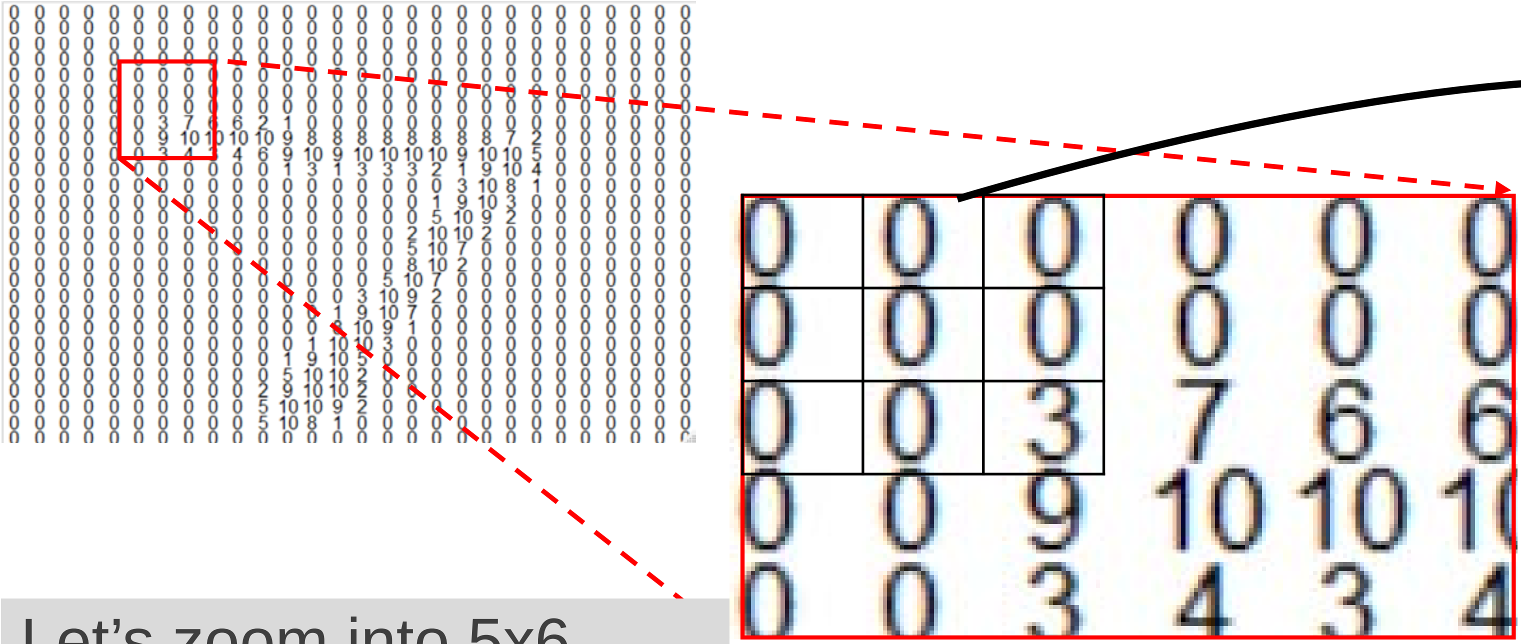






Let’s zoom into 5x6 window of pixels near the tip of ‘7’

Take a 3x3 patch of pixels and apply a ‘filter’ template – designed to find an edge



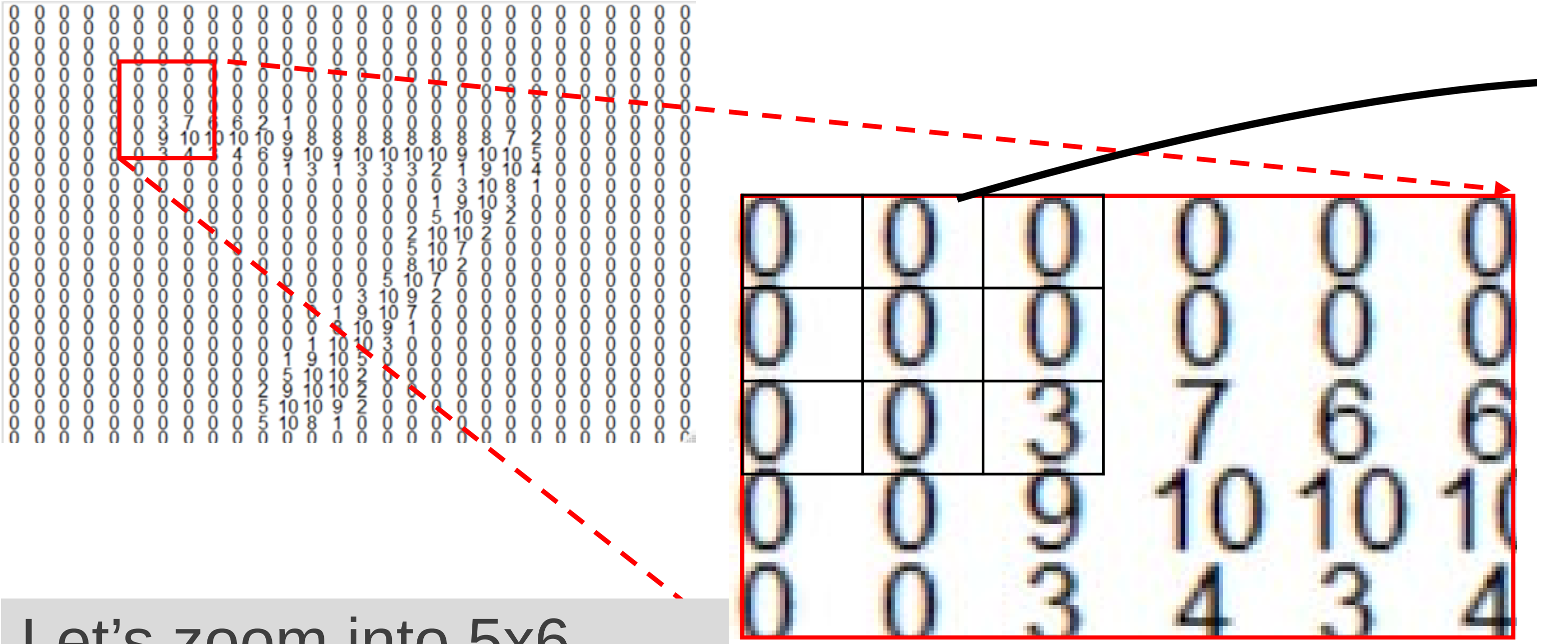
X

-1	0	+1
-1	0	+1
-1	0	+1

1. Multiply 3x3 patch of pixels with 3x3 filter

Let’s zoom into 5x6 window of pixels near the tip of ‘7’

Take a 3x3 patch of pixels and apply a ‘filter’ template – designed to find an edge



These are W parameters

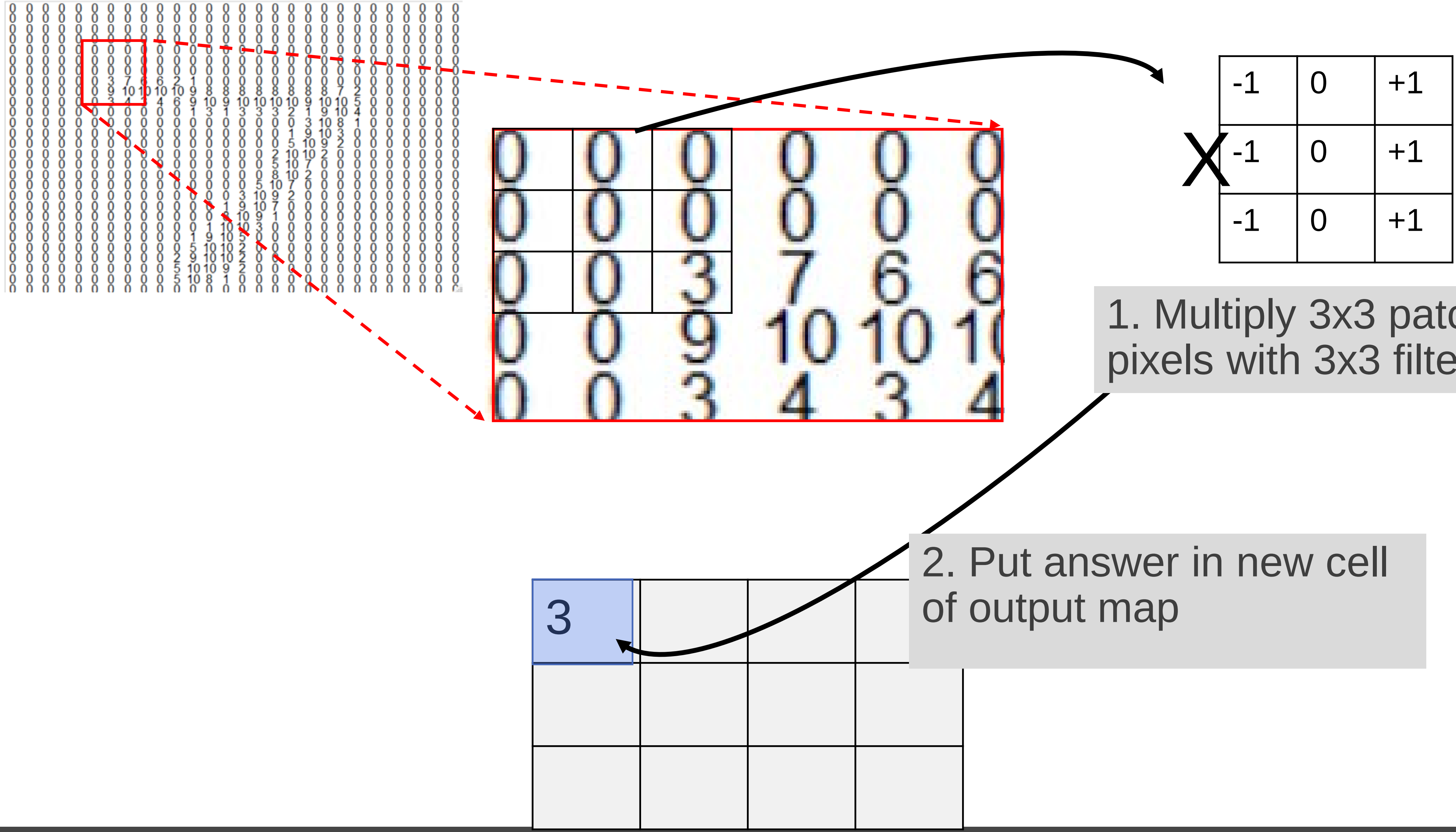
X

-1	0	+1
-1	0	+1
-1	0	+1

1. Multiply 3x3 patch of pixels with 3x3 filter

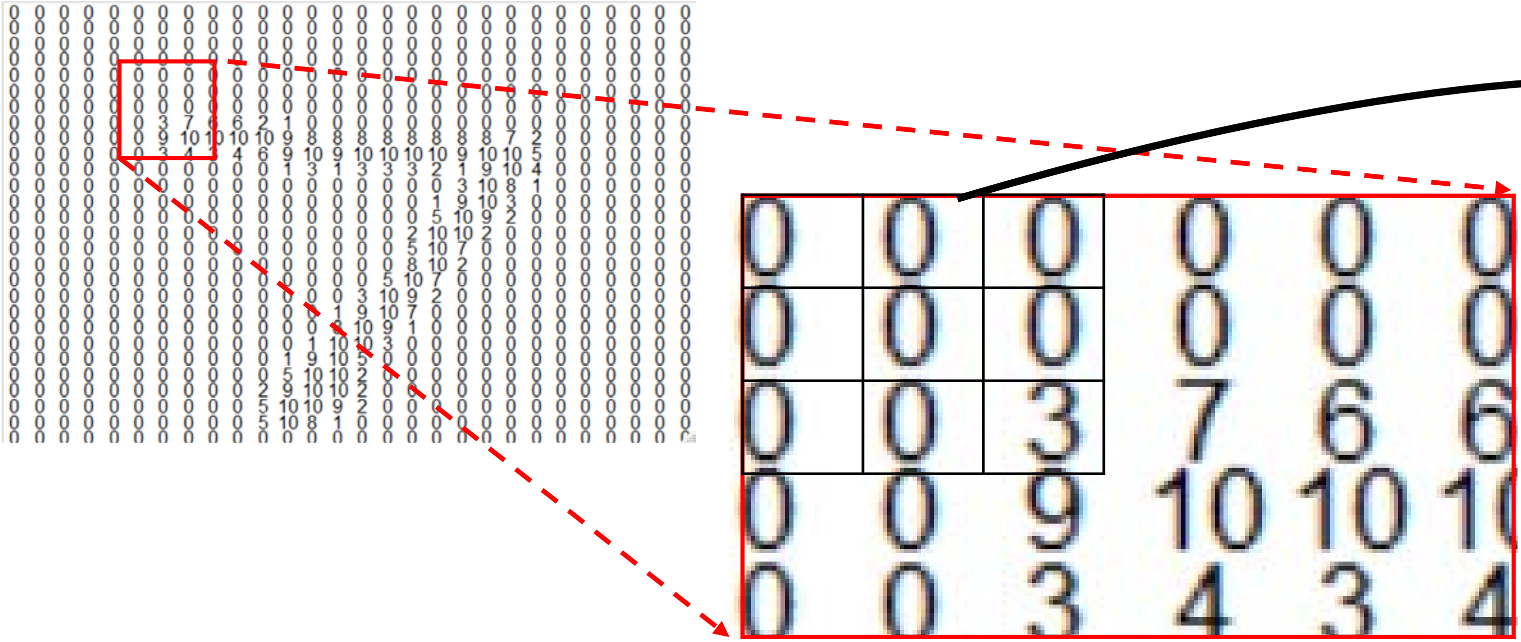
Let’s zoom into 5x6 window of pixels near the tip of ‘7’

Take a 3x3 patch of pixels and apply a ‘filter’ template – designed to find an edge



1. Multiply 3x3 patch of pixels with 3x3 filter

2. Put answer in new cell of output map



X

-1	0	+1
-1	0	+1
-1	0	+1

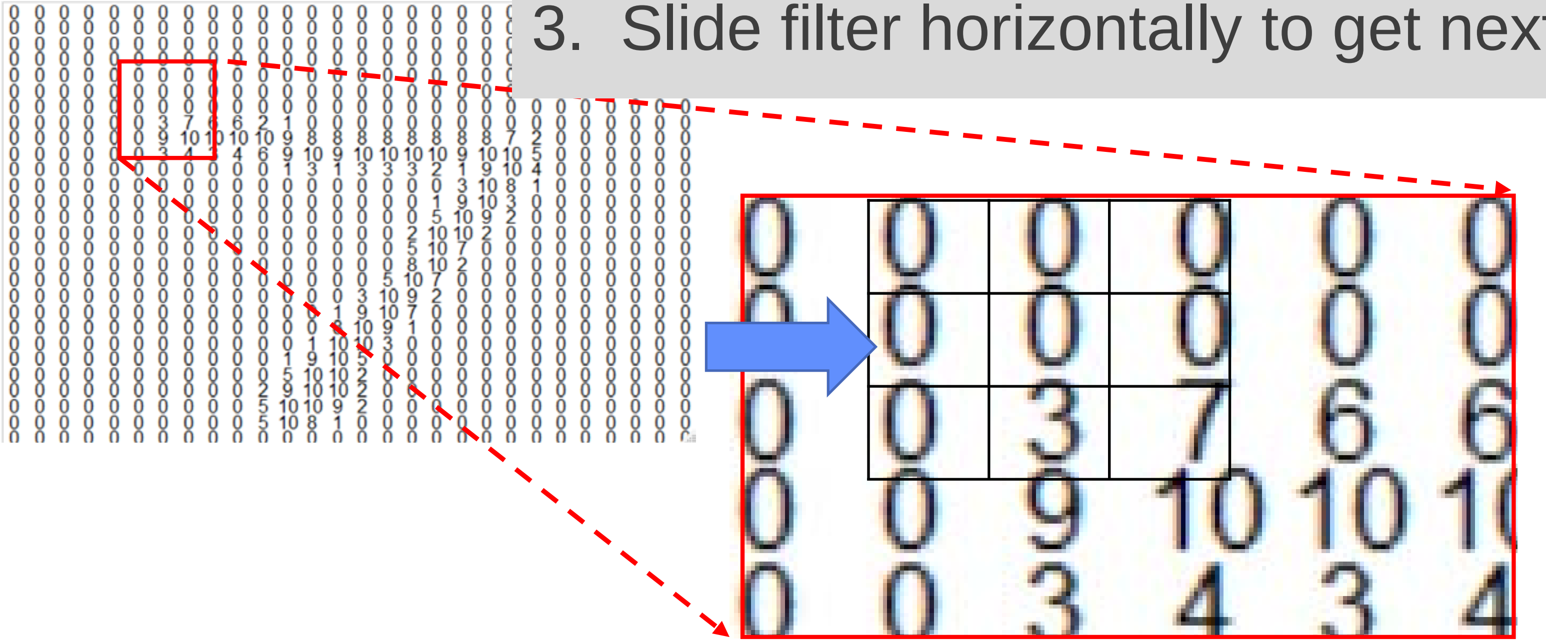
1. Multiply 3x3 patch of pixels with 3x3 filter

This is $X * W$

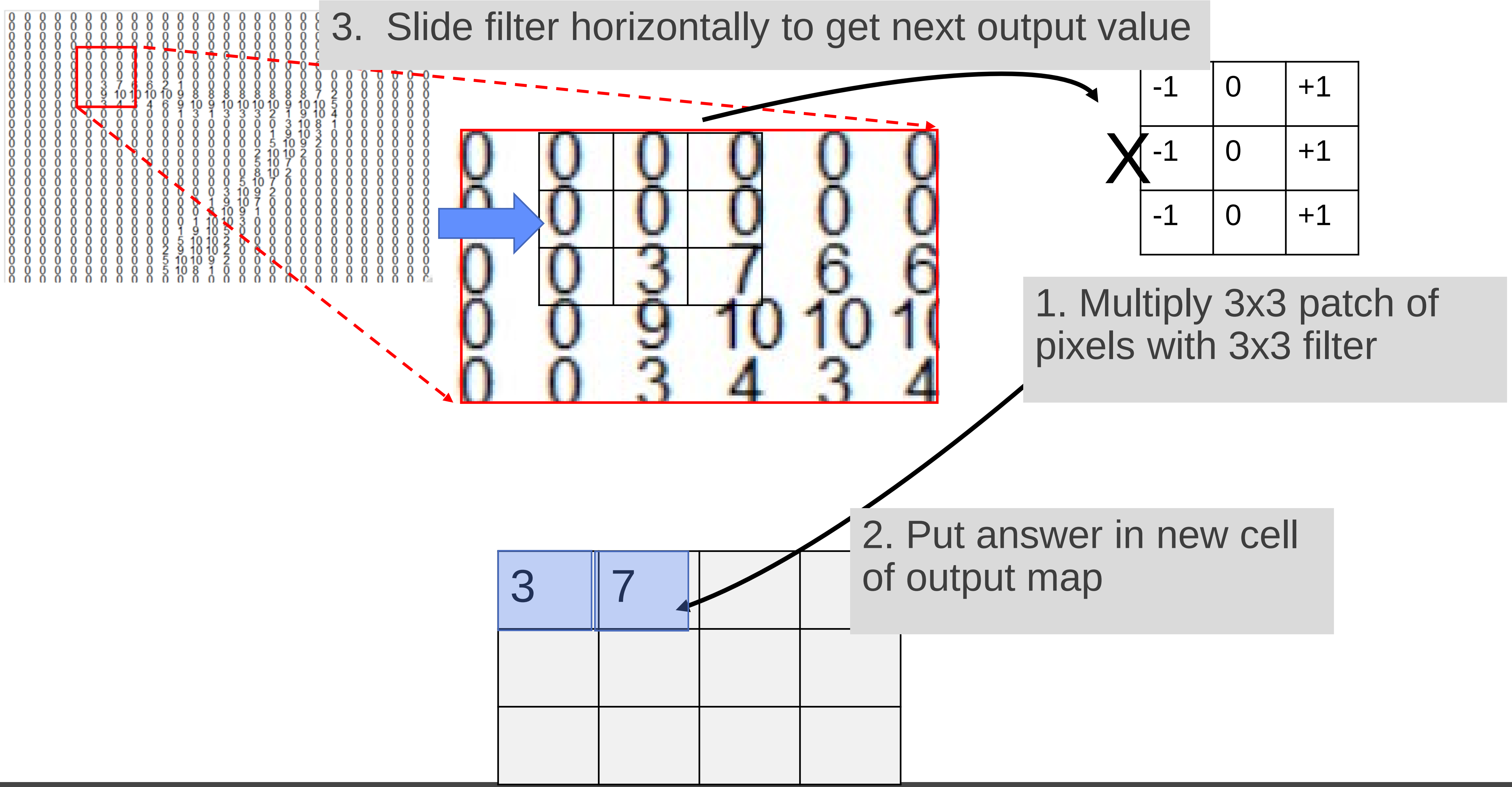
3			

2. Put answer in new cell of output map

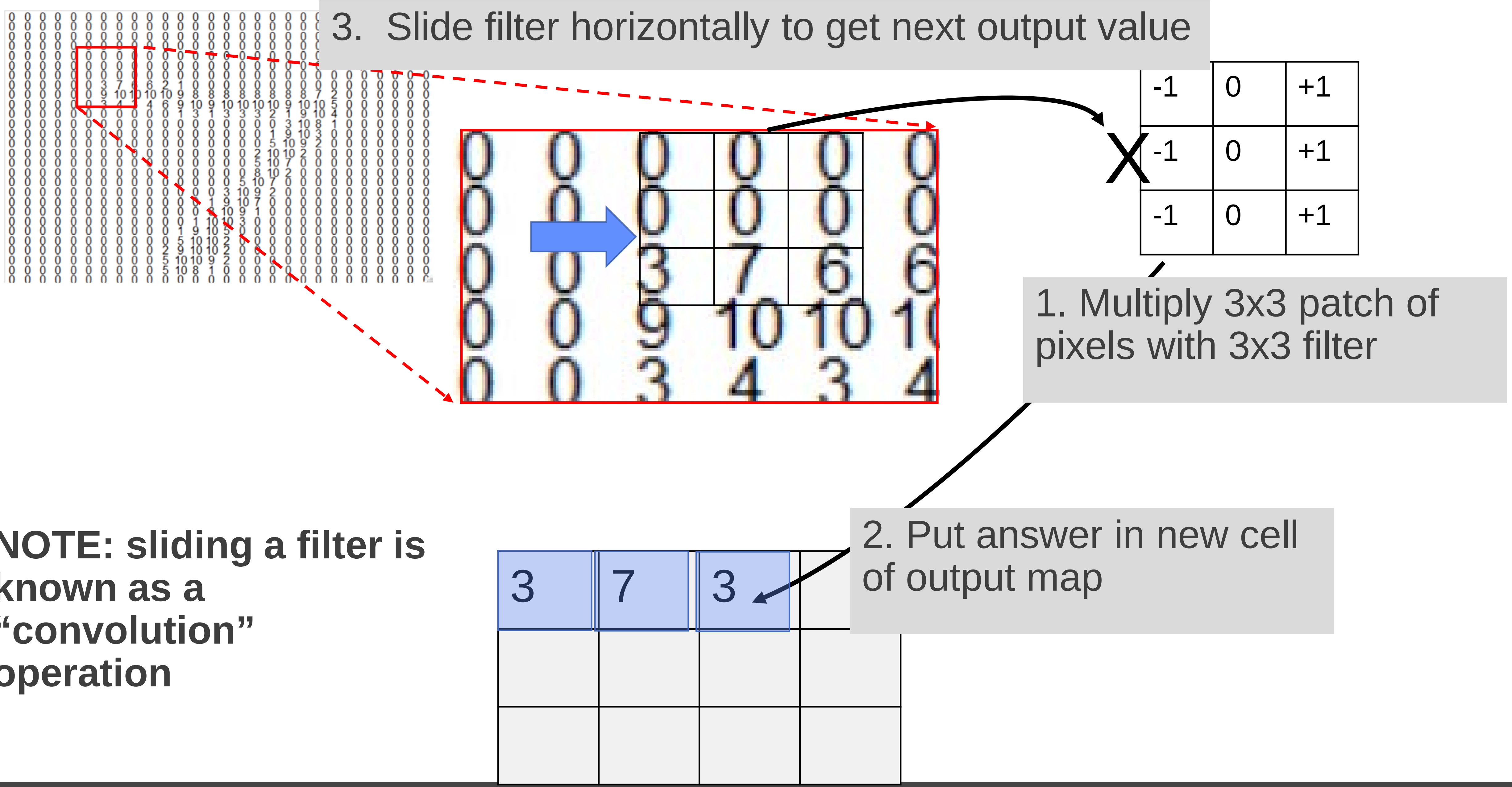
3. Slide filter horizontally to get next output value



3	7		

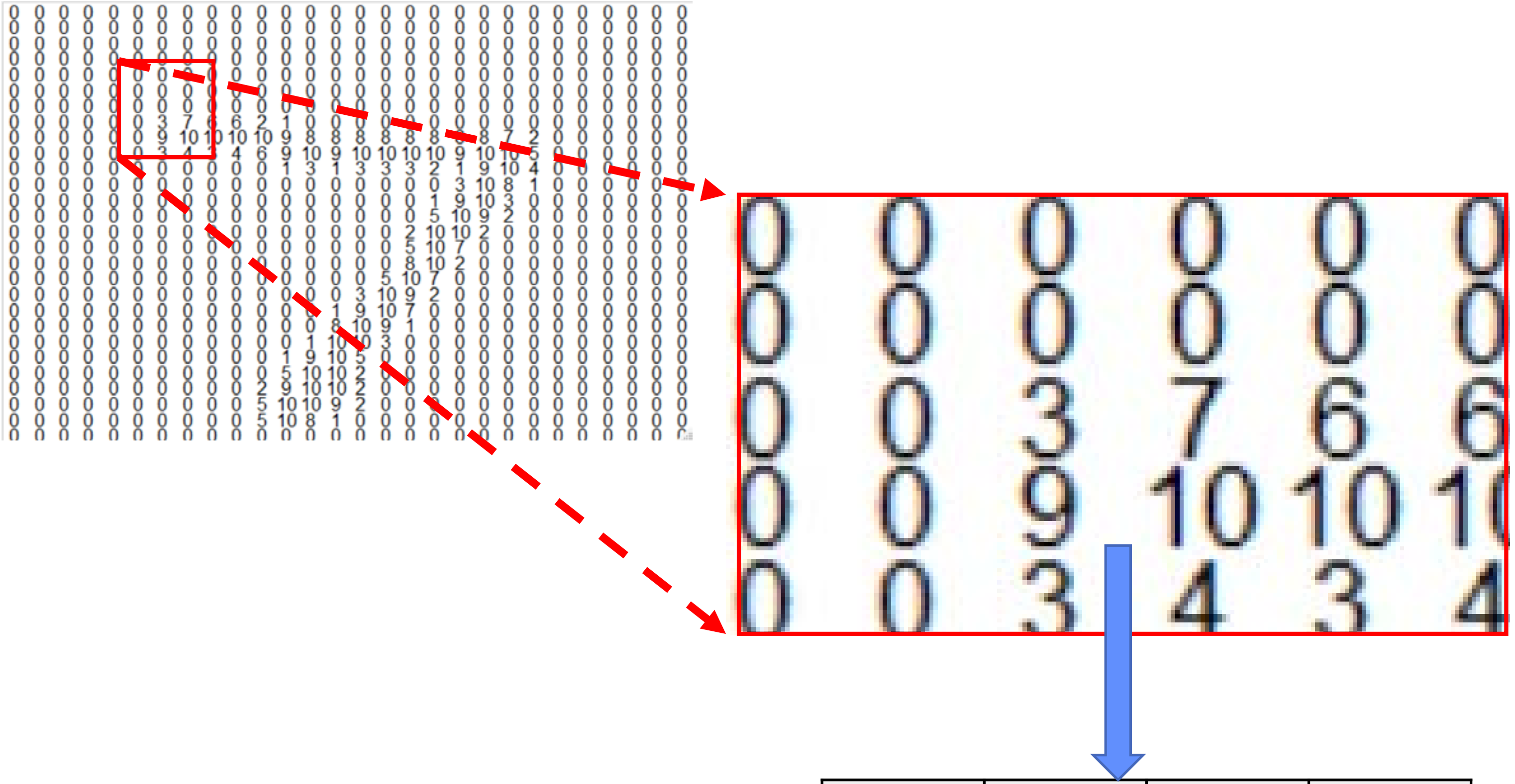


3. Slide filter horizontally to get next output value
X
-1 0 +1
-1 0 +1
-1 0 +1
1. Multiply 3x3 patch of pixels with 3x3 filter
2. Put answer in new cell of output map
3
7

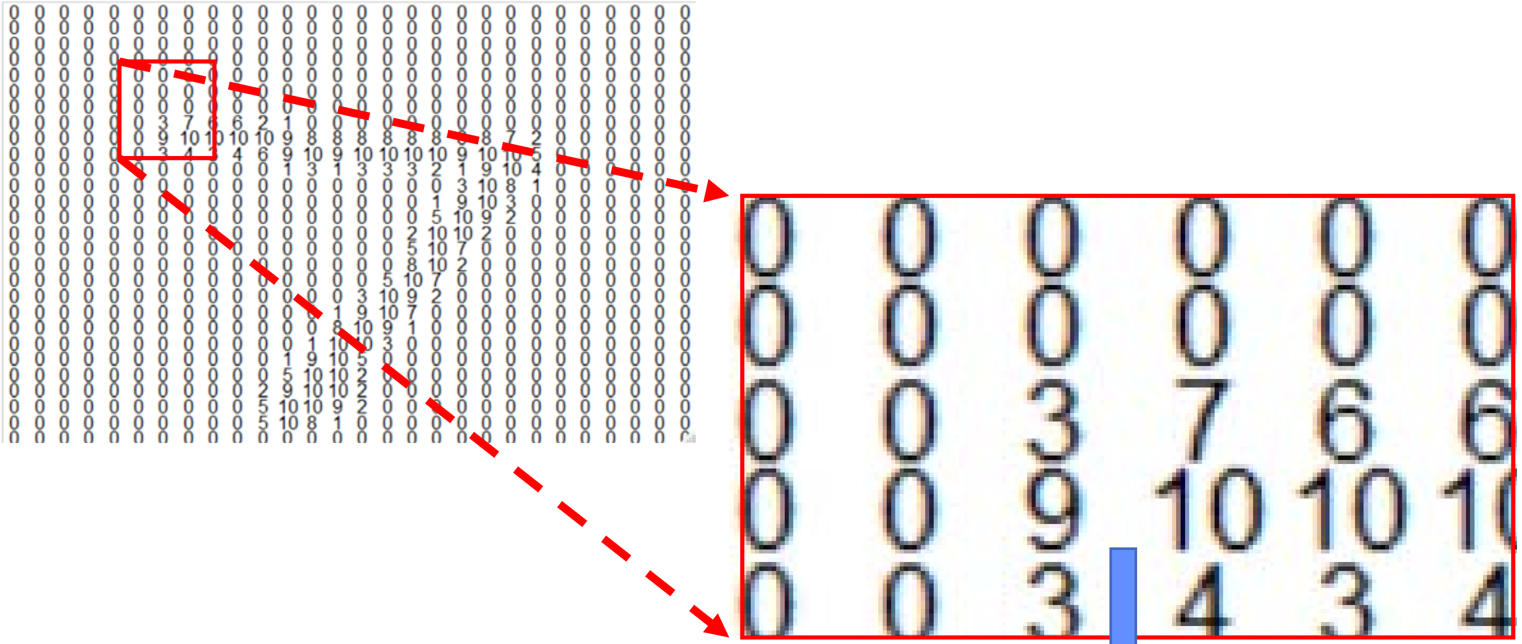


NOTE: sliding a filter is known as a "convolution" operation

After vertical and horizontal sliding the 5x6 patch is now a 3x5 ***feature map.***



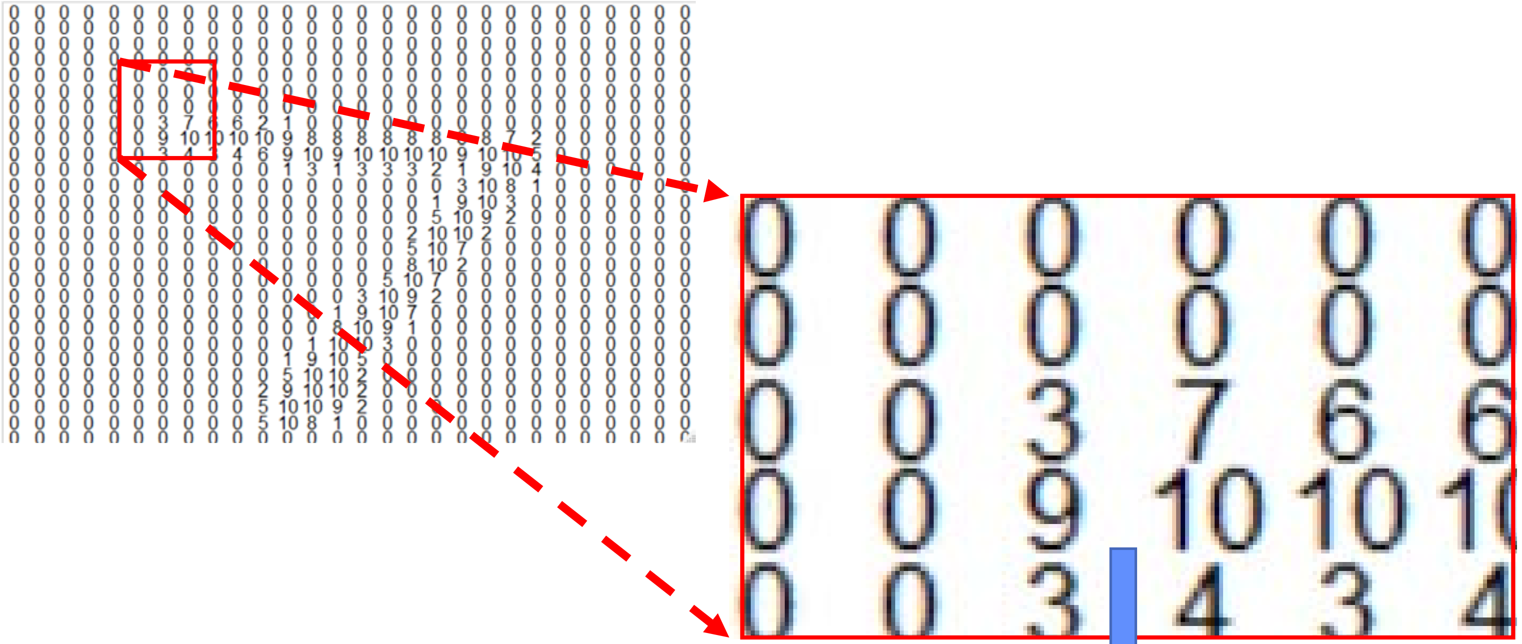
3	7	3	-1
12	17	4	-1
15	21	4	-1



3	7	3	-1
12	17	4	-1
15	21	4	-1

After vertical and horizontal sliding the 5x6 patch is now a 3x5 ***feature map.***

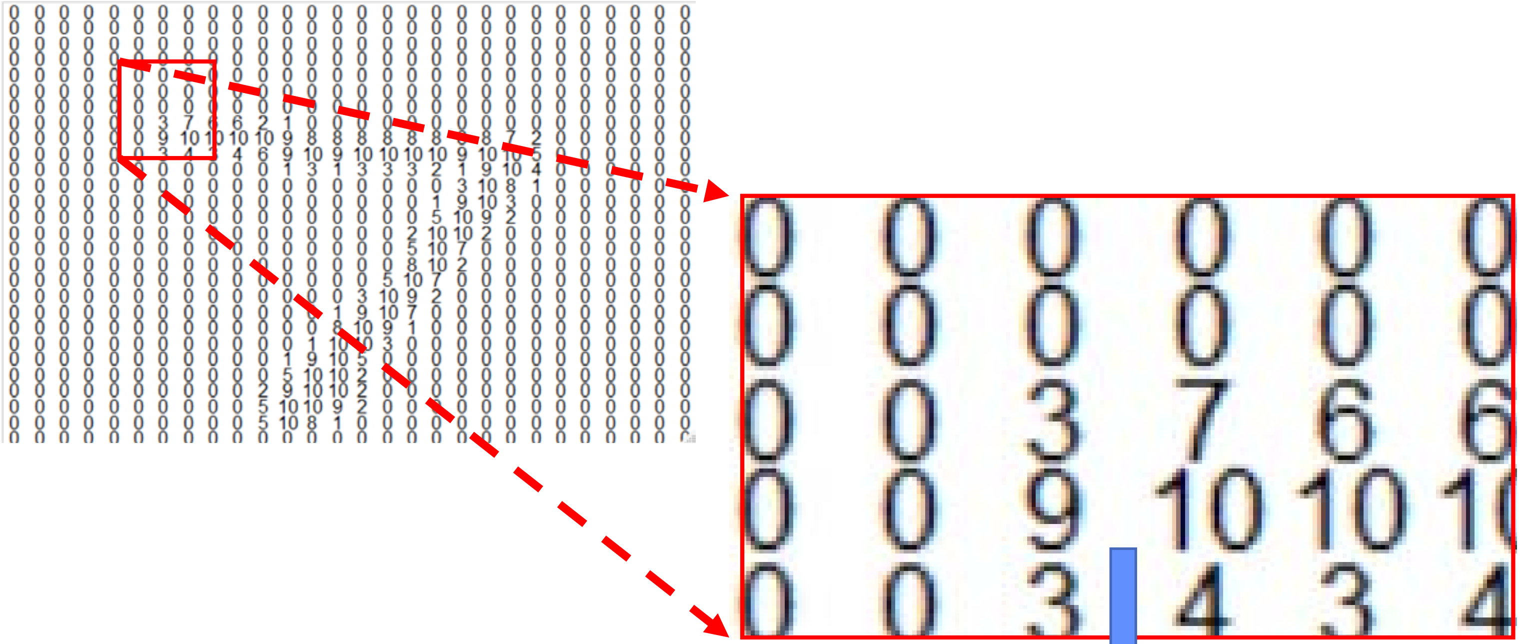
What do the highest values in the feature map represent?



3	7	3	-1
12	17	4	-1
15	21	4	-1

Optional next step:

Use another filter, and take maximum over elements - “max pooling”



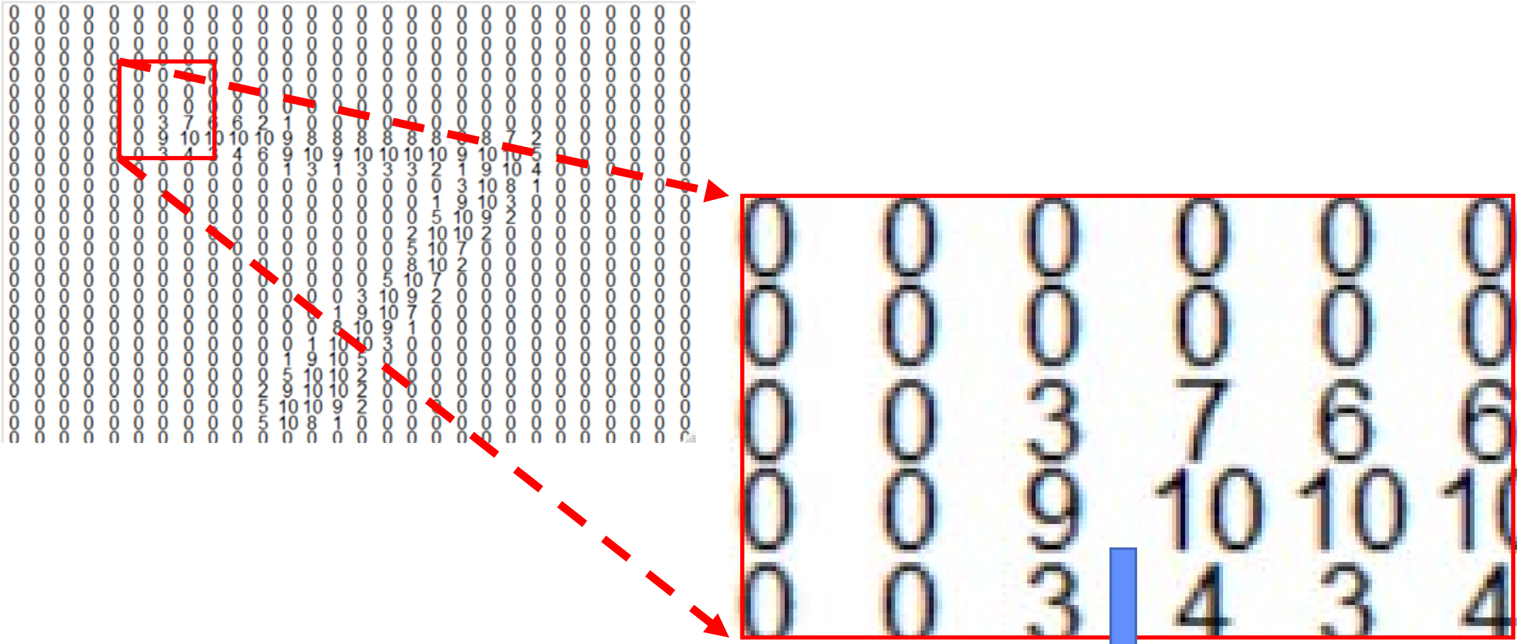
3	7	3	-1
12	17	4	1
15	21	4	-1

Optional next step:

Use another filter, and take maximum over elements - “max pooling”

2x2 filter has max=17

17		



3	7	3	-1
12	17	4	-1
15	21	4	-1

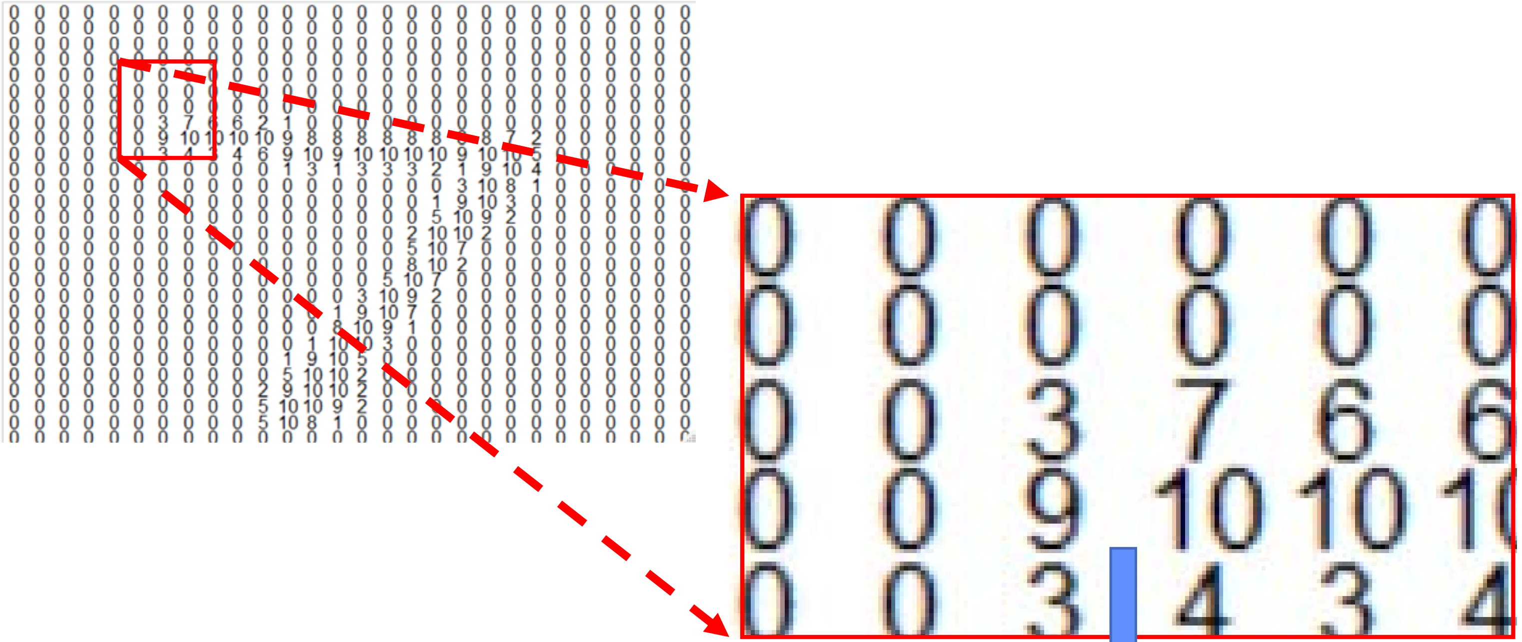
Optional next step:

Use another filter, and take maximum over elements - "max pooling"

Slide filter …

17	17	4
21	21	4

After convolution and pooling 5x6 patch is ***transformed*** *into a 2x3 feature map of ‘edge gradients’*



3	7	3	-1
12	17	4	-1
15	21	4	-1

Slide filter …

17	17	4
21	21	4

Feature engineering

In Computer Vision there are many kinds of edge detectors and many ways to scale them

-1	0	+1
-1	0	+1
-1	0	+1

But building features is hard, so if you have enough data …

Convolution Neural Network (CNN)

In CNNs the filter values are weight parameters that are learned (feature discovery)

W_{11}	W_{12}	W_{13}
W_{21}	W_{22}	W_{23}
W_{31}	W_{32}	W_{33}

Convolution Neural Network (CNN)

In CNNs the filter values are weight parameters that are learned (feature discovery)

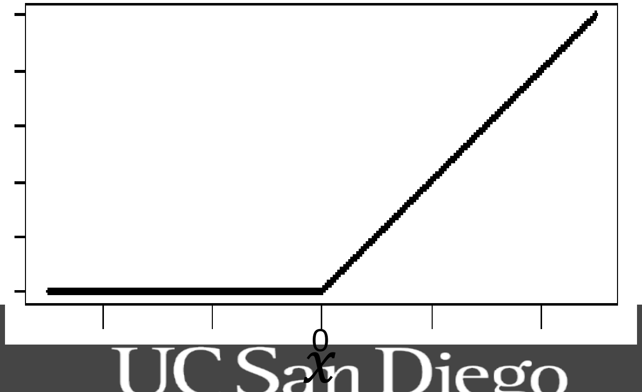
W_{11}	W_{12}	W_{13}
W_{21}	W_{22}	W_{23}
W_{31}	W_{32}	W_{33}

The activation function is often a RELU

RELU (rectified linear

$$f(a) = \begin{cases} a & a > 0 \\ 0 & a <= 0 \end{cases}$$

where $a = XW$



Convolution Neural Network (CNN)

In CNNs the filter values are weight parameters that are learned (feature discovery)

W_{11}	W_{12}	W_{13}
W_{21}	W_{22}	W_{23}
W_{31}	W_{32}	W_{33}

The activation function is often a RELU

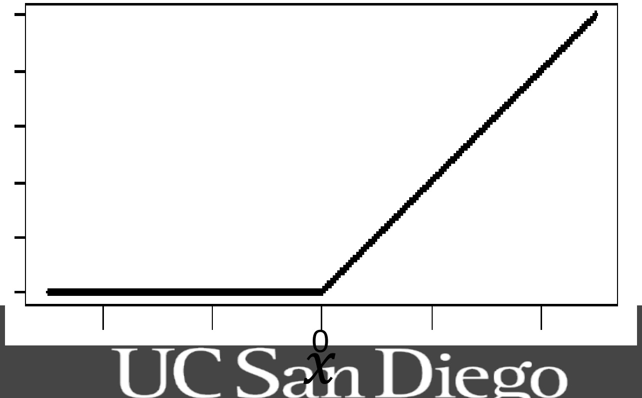
It is unscaled (bad!)

But *df/da* is constant (good!)

$$f(a) = \begin{cases} a & a > 0 \\ 0 & a <= 0 \end{cases}$$

where $a = XW$

RELU (rectified linear



Convolution Neural Network (CNN)

In CNNs the filter values are weight parameters that are learned (feature discovery)

W_{11}	W_{12}	W_{13}
W_{21}	W_{22}	W_{23}
W_{31}	W_{32}	W_{33}

The activation function is often a RELU

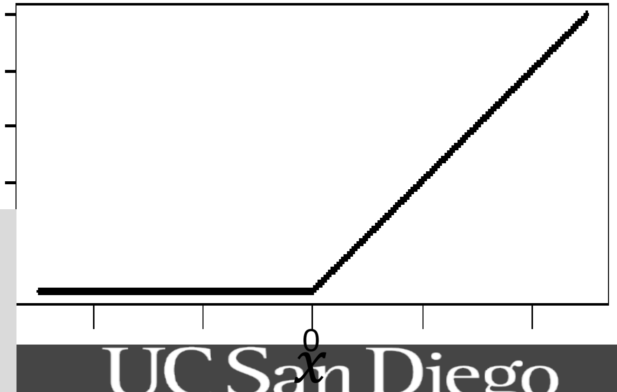
It is unscaled (bad!)

But *df/da* is constant (good!)

$$f(a) = \begin{cases} a & a > 0 \\ 0 & a <= 0 \end{cases}$$

where $a = XW$

RELU (rectified linear



A convolution layer is a set of feature maps, where each map is derived from convolution of 1 filter with input.

Convolution Neural Network (CNN)

More hyperparameters:

Size of filter (smaller is more general)

Convolution Neural Network (CNN)

More hyperparameters:

Size of filter (smaller is more general)

Number of pixels to slide over (1 or 2 is usually fine)

Convolution Neural Network (CNN)

More hyperparameters:

Size of filter (smaller is more general)

Number of pixels to slide over (1 or 2 is usually fine)

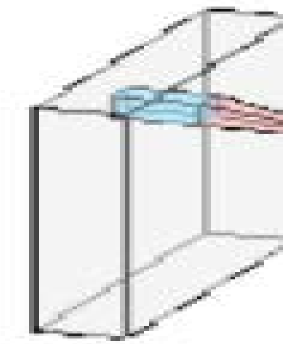
Number of filters (depends on the problem!)

Max pooling or not (usually some pooling layers)

Convolution with image

- **Make 1 layer, using HxWx3 image (3 for Red,Green,Blue channels)**

Crop or resize image as needed

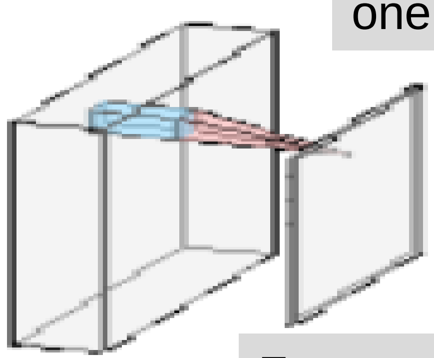


224x224x3

Convolution with image

- **Make 1 layer, using HxWx3 image (3 for RGB channels)**

Crop or resize image as needed



one 5x5x3 filter

224x224x3

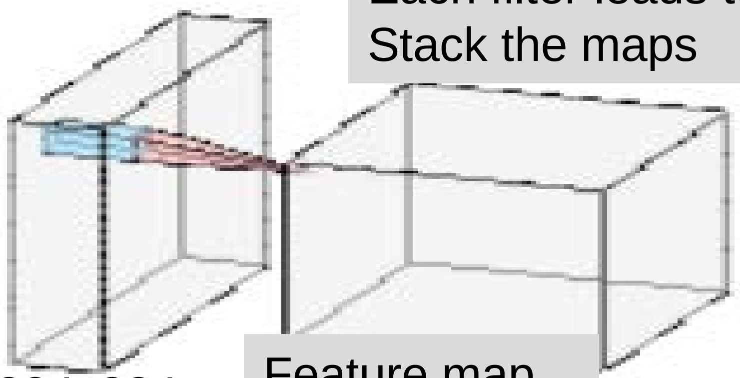
Feature map
Size depends on filter size and slide parameter

Convolution with image

- **Make 1 layer, using HxWx3 image (3 for RGB channels)**

Crop or resize image as needed

Repeat for many filters, all 5x5x3.
Each filter leads to 1 feature map
Stack the maps



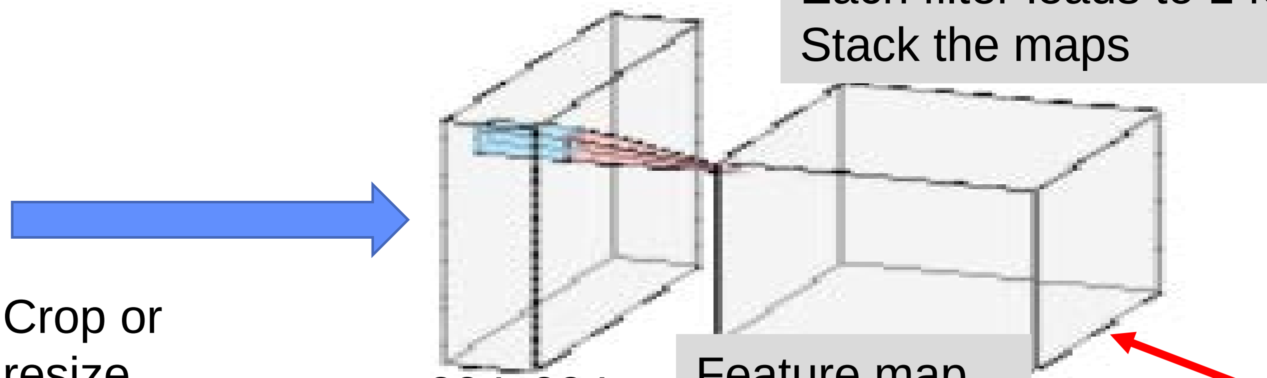
224x224x3

Feature map
Size depends on filter size and slide parameter

Convolution with image

- **Make 1 layer, using HxWx3 image (3 for RGB channels)**

Crop or resize image as needed



Repeat for many filters, all 5x5x3.
Each filter leads to 1 feature map
Stack the maps

224x224x3

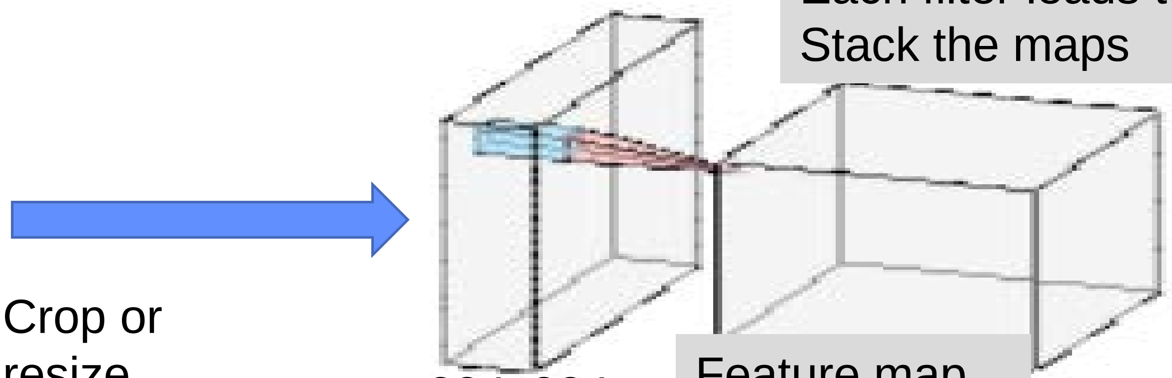
Feature map
Size depends on filter size and slide parameter

This is a "convolution layer"

Convolution with image

- **Make 1 layer, using HxWx3 image (3 for RGB channels)**

Crop or
resize
image as
needed



224x224x
3

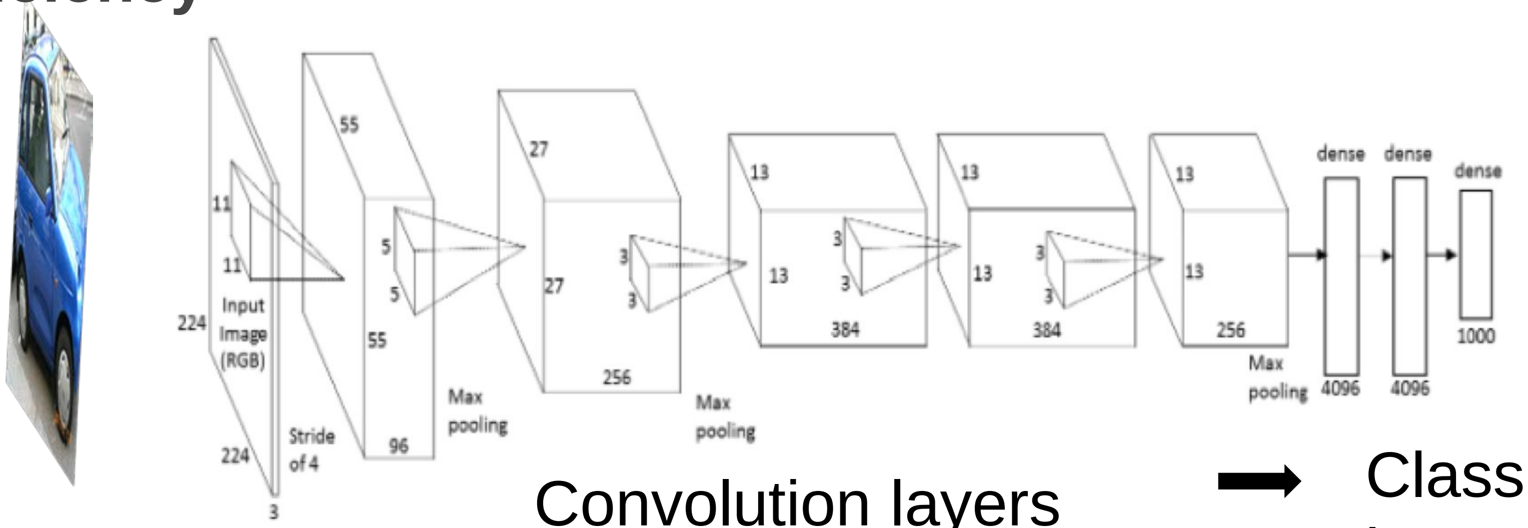
Repeat for many filters, all 5x5x3.
Each filter leads to 1 feature map
Stack the maps

Feature map
Size depends
on filter size
and slide
parameter

Add a layer of max-pool to
down sample each feature
map
OR
Add a next convolution layer

Large Scale Versions

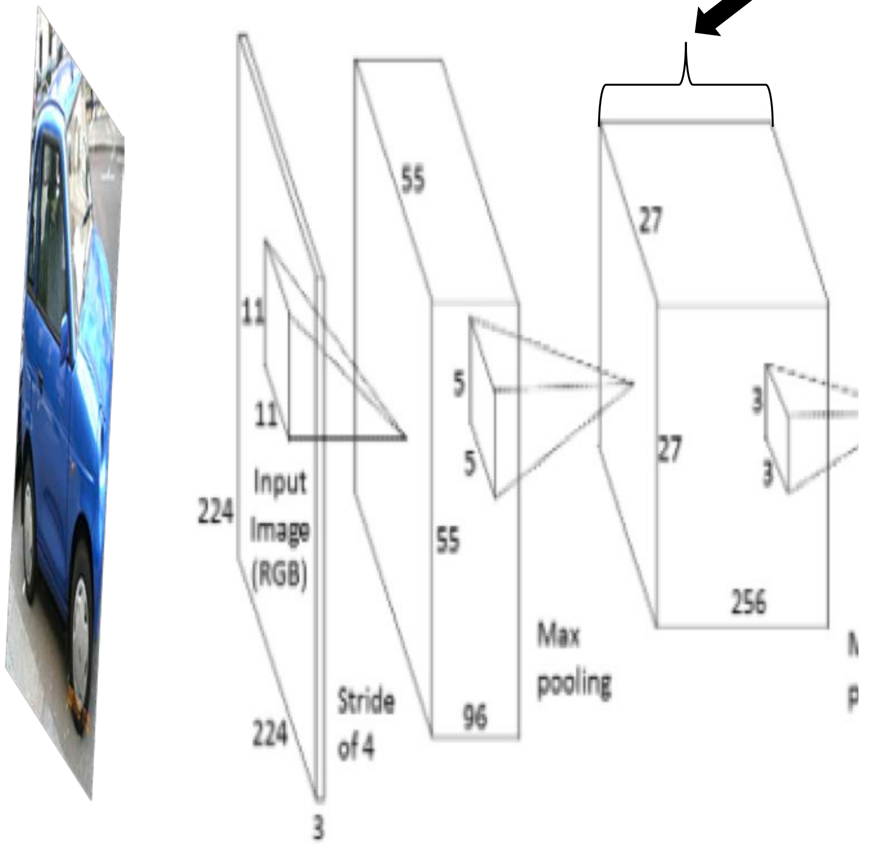
- Large (deep) Convolution Networks are turning out to be feasible with GPUs (some are 100+ layers)
- Need large amounts of data and many heuristics to avoid overfitting and increase efficiency



Convolution layers ➡ Classification layers and output

Large Scale Versions

- Zooming in: Convolution layers

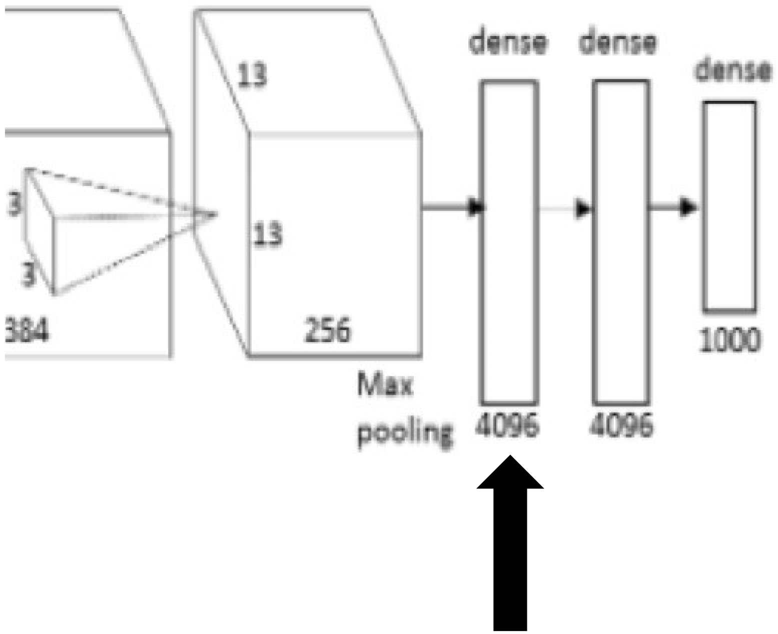


The thickness is the number of different feature maps, sometimes called ‘channels’ or ‘number of filters’

Large Scale Versions

- **Zooming in:**

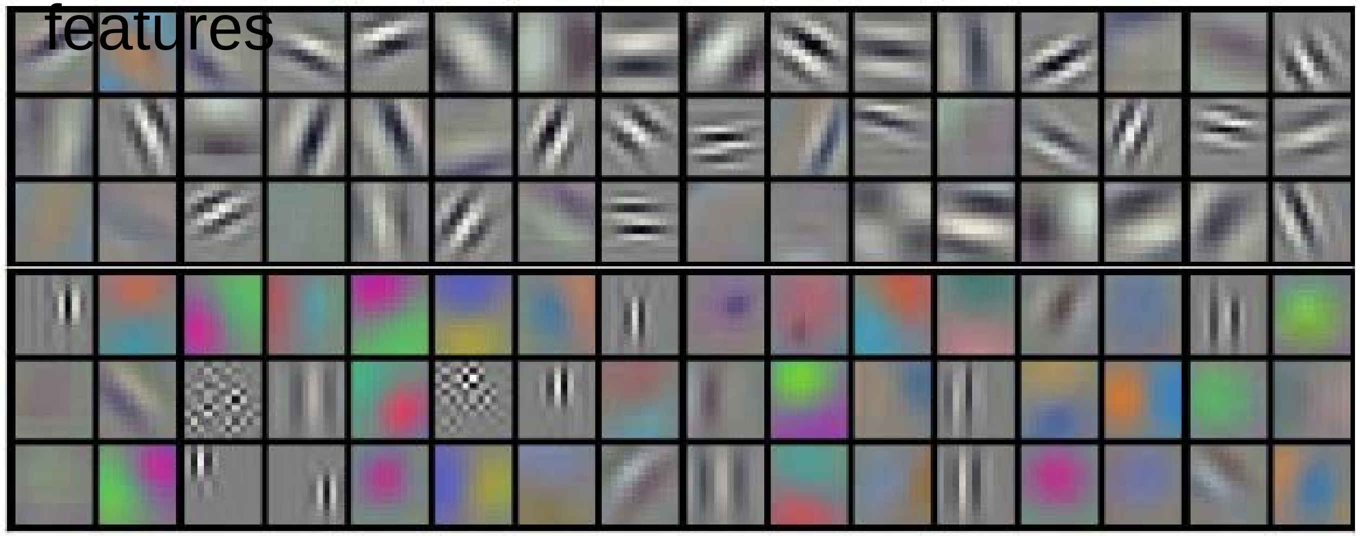
Classification layers



Last convolution layer is laid out as a vector followed by fully connected, hidden layers and output layer.

What Learned Convolutions Look Like

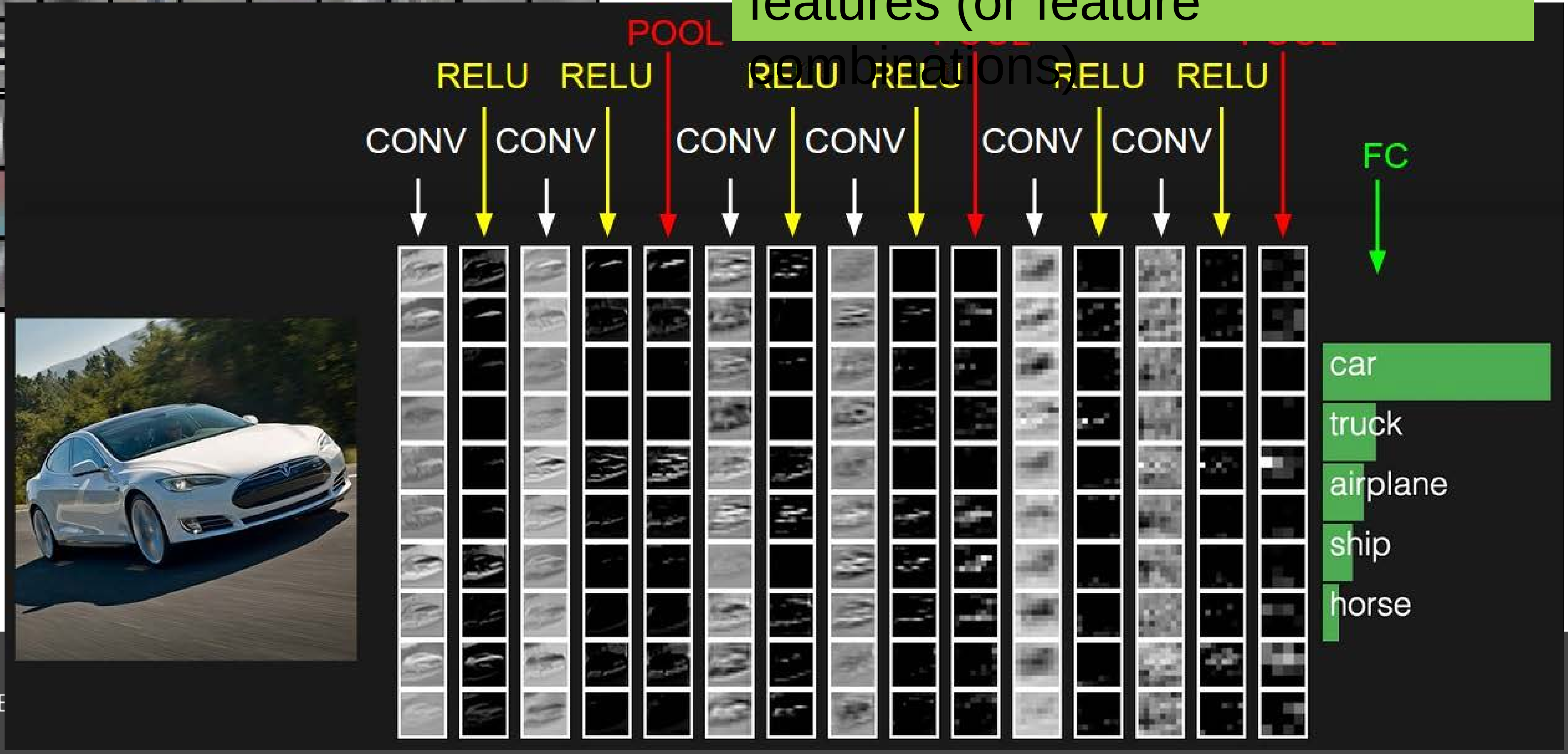
First convolution layer filters are simple features



What Learned Convolutions

First convolution layer filters are simple features

Higher layers are more abstract features (or feature combinations)



General deep learning features for object recognition

CNNs works because convolution layers have a special architecture and function – it is biased to do certain kind of transformations

Low layers have less filters that represent simple local features for all classes

Higher layers have more filters that cover large regions that represent object class features

General deep learning features for object recognition

CNNs works because convolution layers have a special architecture and function – it is biased to do certain kind of transformations

Low layers have less filters that represent simple local features for all classes

Higher layers have more filters that cover large regions that represent object class features

Pause for questions

Tutorial

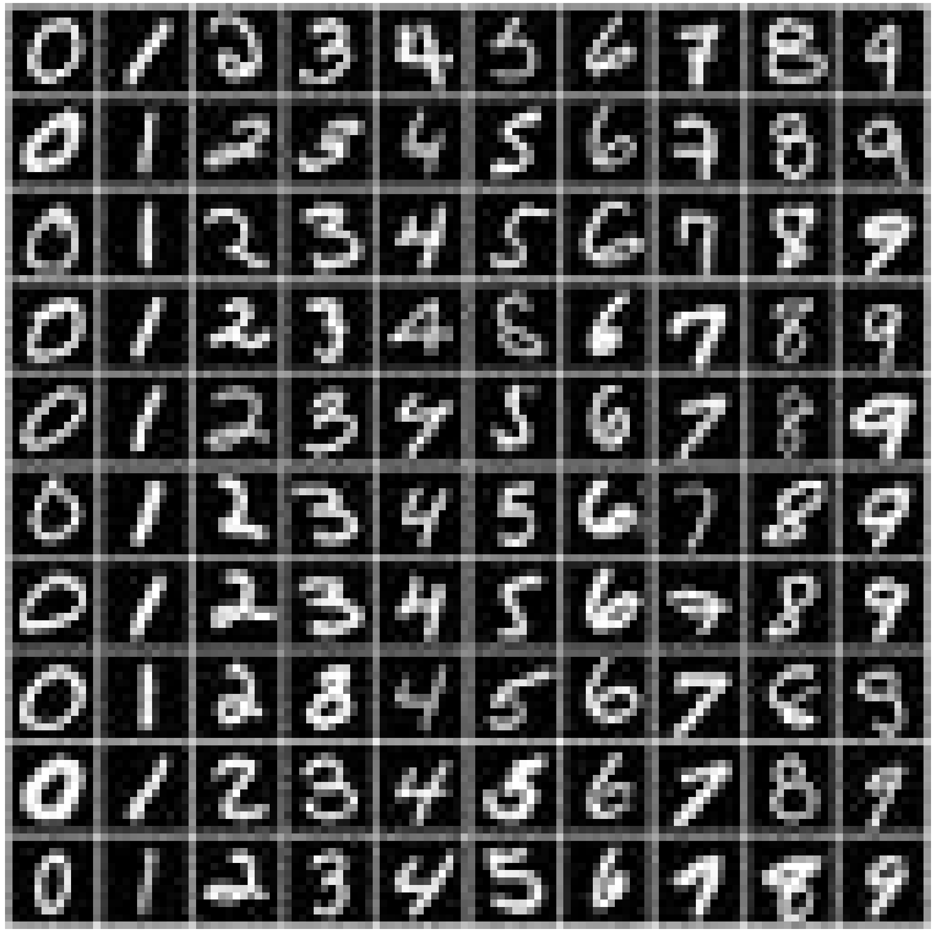
- The ‘hello world’ of Conv. Neural Networks

- Use Keras front end to Tensorflow engine (neural math operations)

- Works with GPU or CPUs

- Start jupyter notebook and see exercise in deep learning folder

(the “<<<< ----” points to parameters to try changing)



Feature Coding vs Discovery

- Edge detection with Support Vector Machine

  OR

  Convolution Neural Network?

- With small datasets and reasonable features, SVMs can work wel

- But building features is hard, and large classification problems can benefit from common features that CNNs can discover

The Zoo

- Machine learning/convolution network frameworks:

  Tensorflow, pyTorch (libraries and API to build graphs of networks and processing)

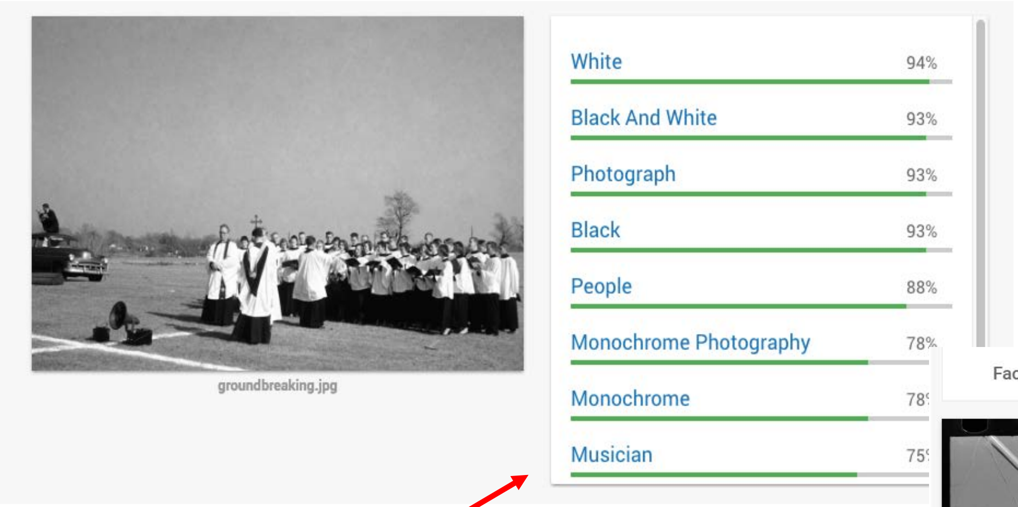
  Keras - higher level CNN library with tensorflow (best for learning)
  Caffe – C/C++ library with many pretrained models
  Caffe2 – Facebook tookover Caffe, Pytorch (has a good model for people detection)
  YOLO/Darknet – A C++ library, with object detection
  Matlab – CNN functions, and pretrained networks

- Many networks pretrained on large or particular object classes are available: AlexNet, VGG19, Googlenet, Detectron

- Big Tech have online services (see next page)

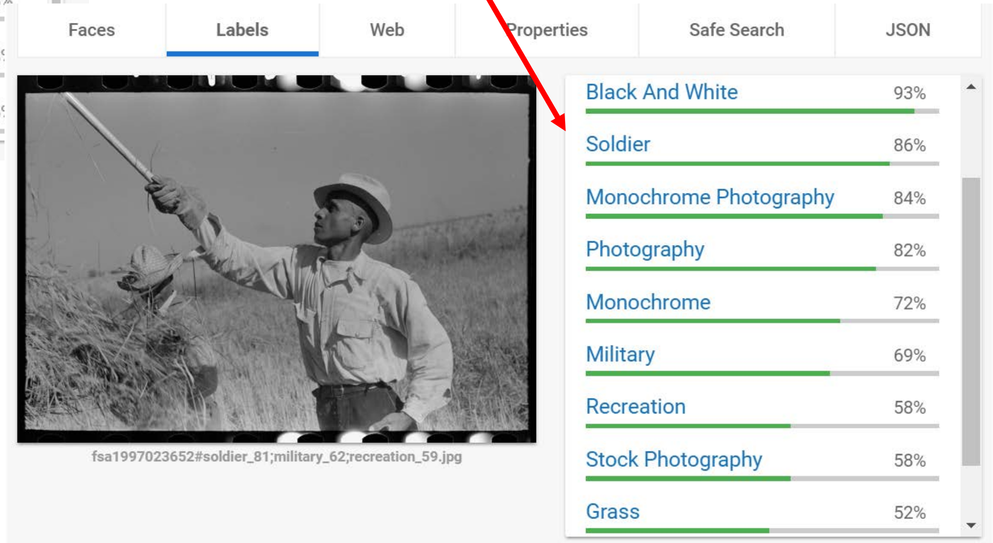
Google tool for objects, faces, text

- Google Vision api – object recognition network



gets “Musician”

fails on “Soldier”



References

- **Book: https://mitpress.mit.edu/books/deep-learning**
- **Documentation: https://keras.io/**
- **Tutorials I used (borrowed):**
  - http://cs231n.github.io/convolutional-networks/
  - https://hackernoon.com/visualizing-parts-of-convolutional-neural-networks-using-keras-and-cats-5cc01b214e59
  - https://github.com/julienr/ipynb_playground/blob/master/keras/convmnist/keras_cnn_mnist.ipynb