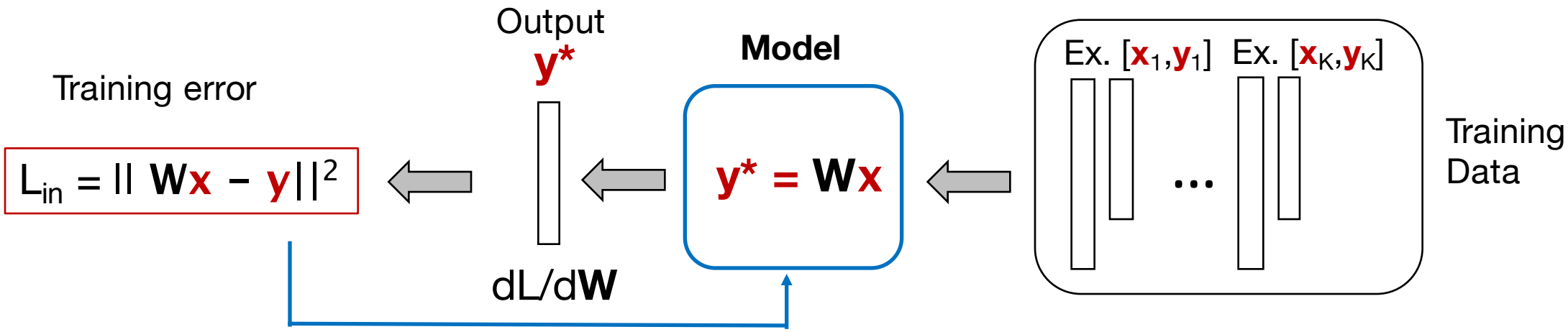
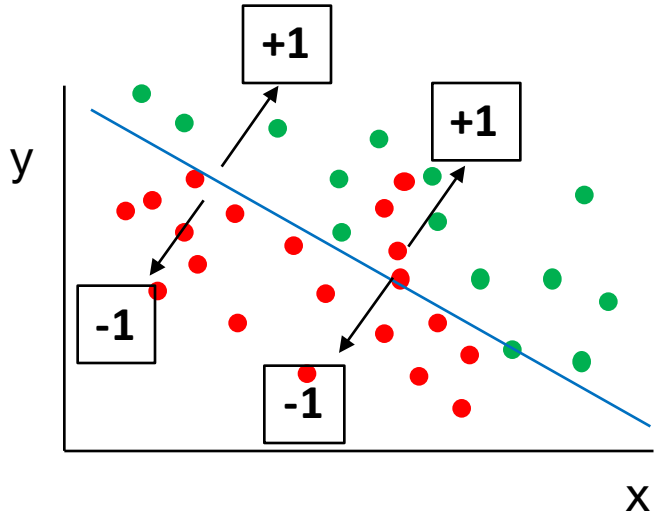
Machine Learning in Imaging

BME 590L
Roarke Horstmeyer

Lecture 7: Increasing model capacity and convolutional networks

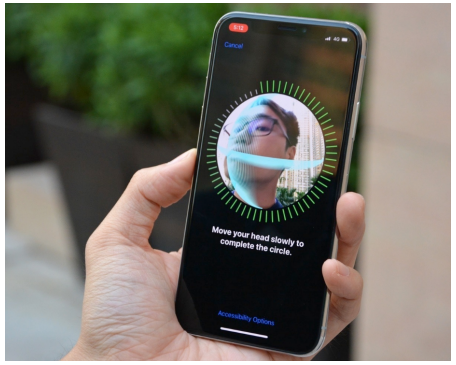
The linear classification model – what's not to like?



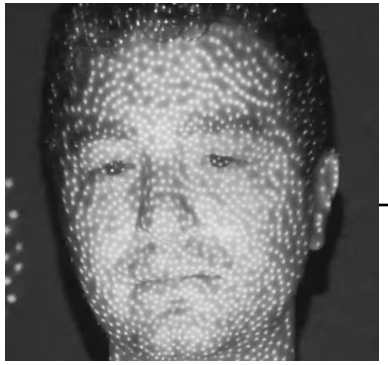


1. Can only separate data with lines (hyper-planes)…
2. We only allowed for binary labels (y = +/- 1)
3. Error function L_{in} inherently makes assumptions about statistical distribution of data

Cost functions matter: a simple example



x



f(x, W) → +1 = You, -1 = Bad guy

Two types of error: false accept and false reject

On a standard phone, what's a good cost function?

$$L_{in} = \text{ReLU}[f(x, W)-y] + \mathbf{10}\ \text{ReLU}[y-f(x, W)]$$

Penalty for intruder

Large penalty for annoyance…

f(x, W)

y \ f(x, W)	+1	-1
+1	No Error	False reject
-1	False accept	No Error

It's you, but you can't get in…

Letting an intruder in

Linear classification is the maximum likelihood for Gaussian data

For what $\mathbf{w}$ is $\prod_{i=1}^{N} p(\mathbf{y}_i, |\mathbf{x}_i, \mathbf{w})$ maximized?

For what $\mathbf{w}$ is $\prod_{i=1}^{N} \exp\left(\frac{-(\mathbf{y}_i - \mathbf{w}^T\mathbf{x})^2}{2\sigma^2}\right)$ maximized?

For what $\mathbf{w}$ is $\sum_{i=1}^{N} \frac{-(\mathbf{y}_i - \mathbf{w}^T\mathbf{x})^2}{2\sigma^2}$ maximized?

For what $\mathbf{w}$ is $\sum_{i=1}^{N} (\mathbf{y}_i - \mathbf{w}^T\mathbf{x})^2$ minimized?

Summary: Linear classification assumes labels are a Gaussian random process, and then thresholds them to ether -1 or +1

How to minimize L_{in}

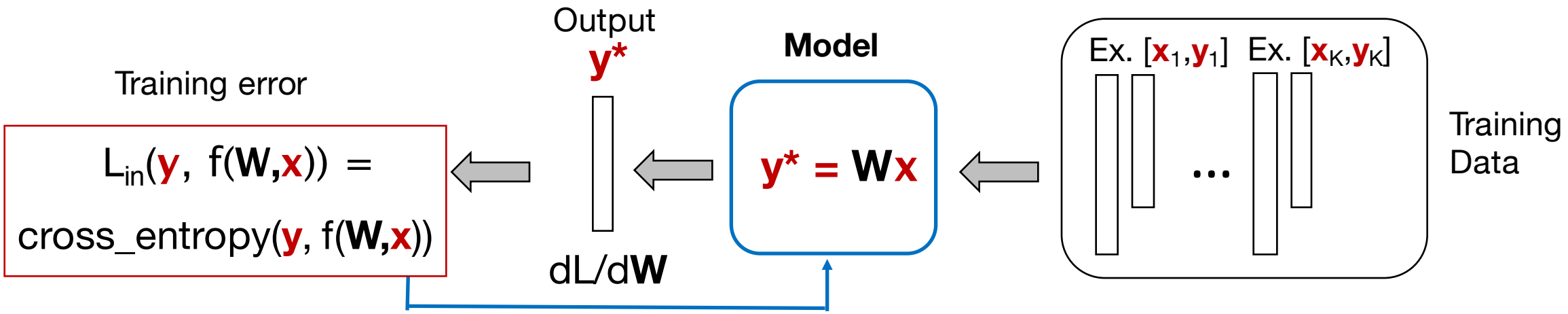
For logistic regression,

$$L_{\text{in}}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} \ln\left(1 + e^{-y_n \mathbf{w}^{\mathsf{T}}\mathbf{x}_n}\right) \quad \longleftarrow \textbf{iterative} \text{ solution}$$

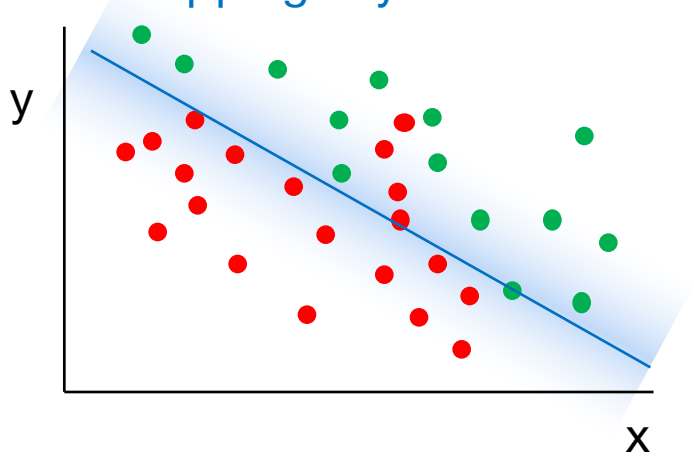
Compare to linear regression:

$$L_{\text{in}}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} \left(\mathbf{w}^{\mathsf{T}}\mathbf{x}_n - y_n\right)^2 \quad \longleftarrow \text{closed-form solution}$$

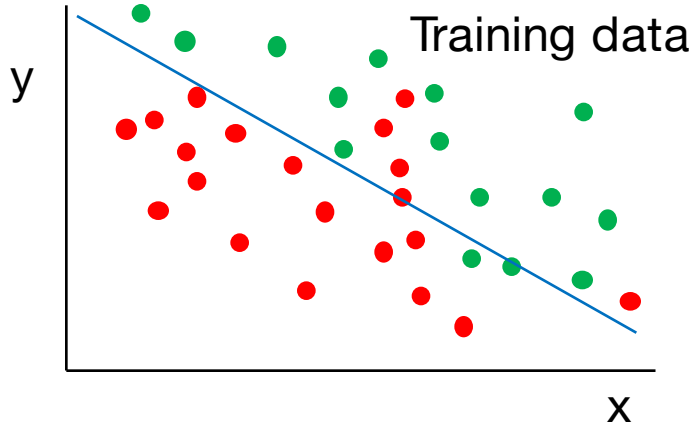
The linear classification model – what's not to like?



Probabilistic mapping to y

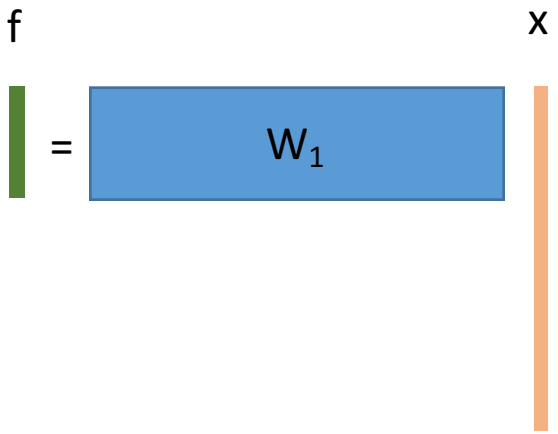


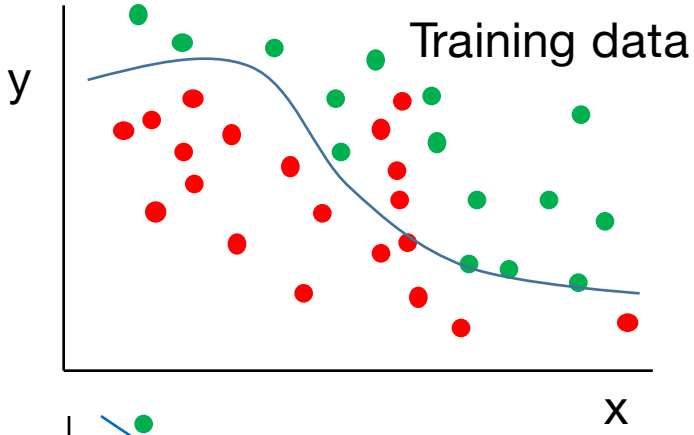
1. Can only separate data with lines (hyper-planes)…
2. We only allowed for binary labels (y = +/- 1)
3. Error function L_{in} inherently makes assumptions about statistical distribution of data



$$f = W_1 x$$

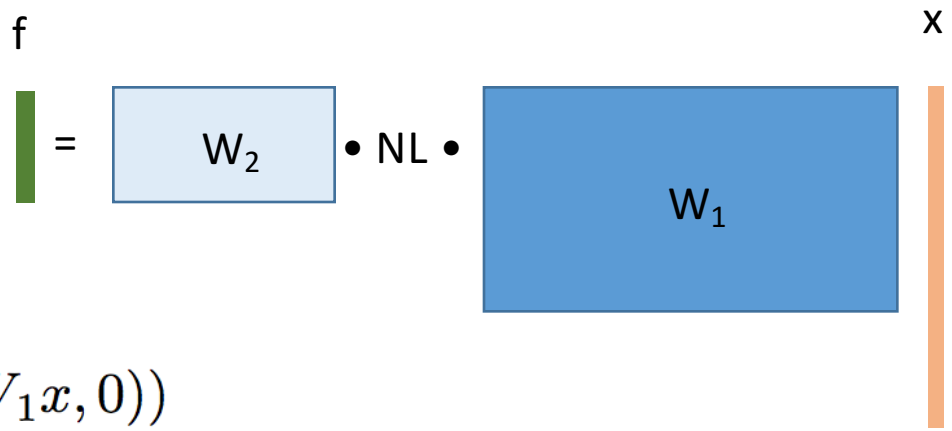
Learned *f:* not flexible

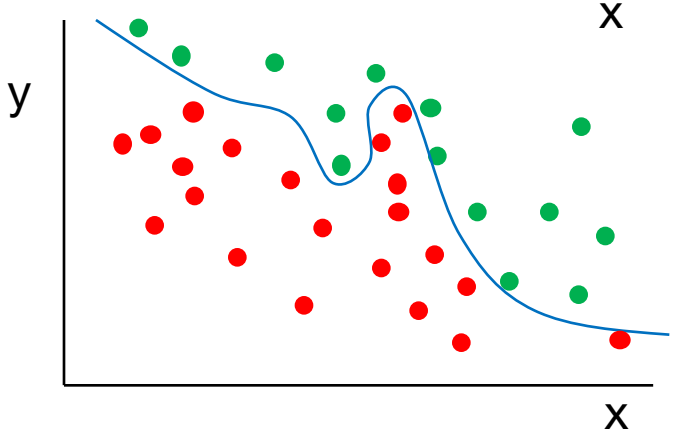




$$f = W_2 \max(W_1 x, 0)$$

Learned *f:* a bit flexible





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

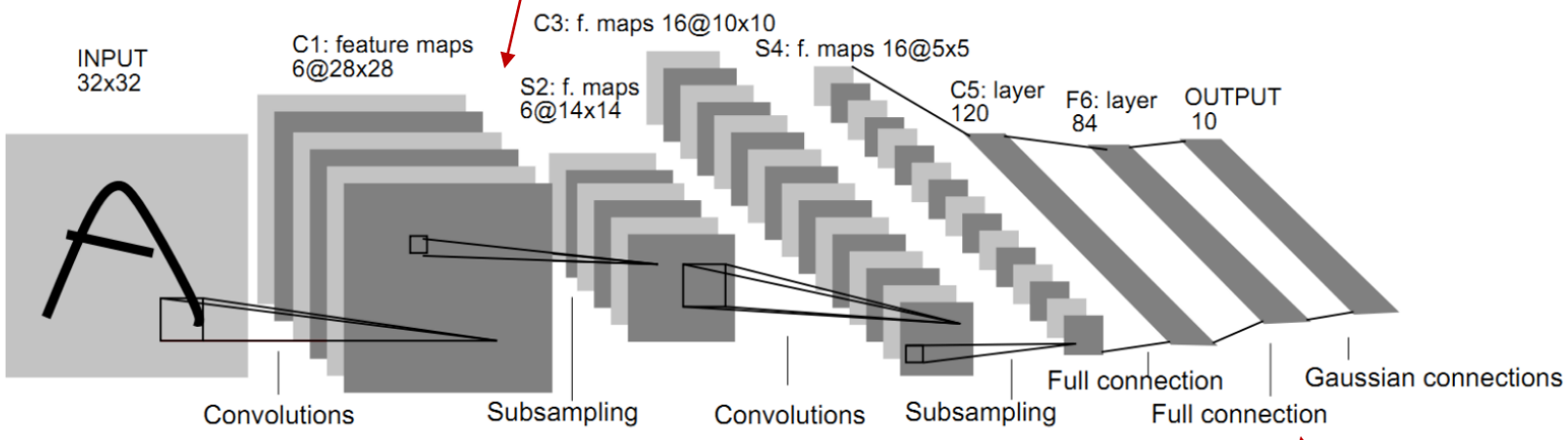
Learned *f:* more flexible

Does it generalize???

We can keep adding these “layers”…

Getting us to Convolutional Neural Networks

After, apply non-linearity and sub-sampling

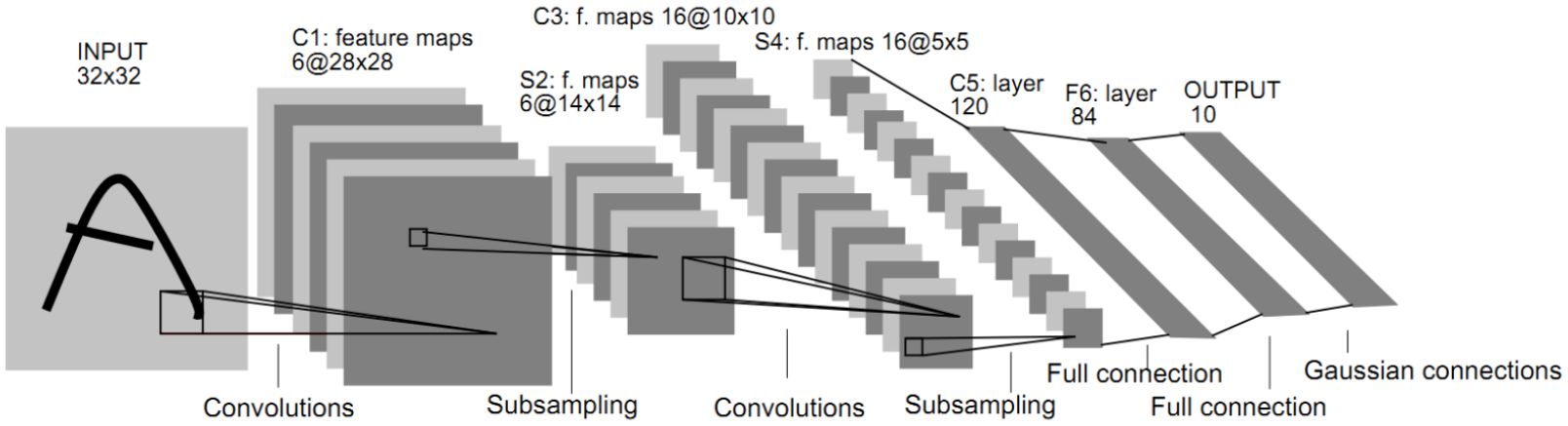


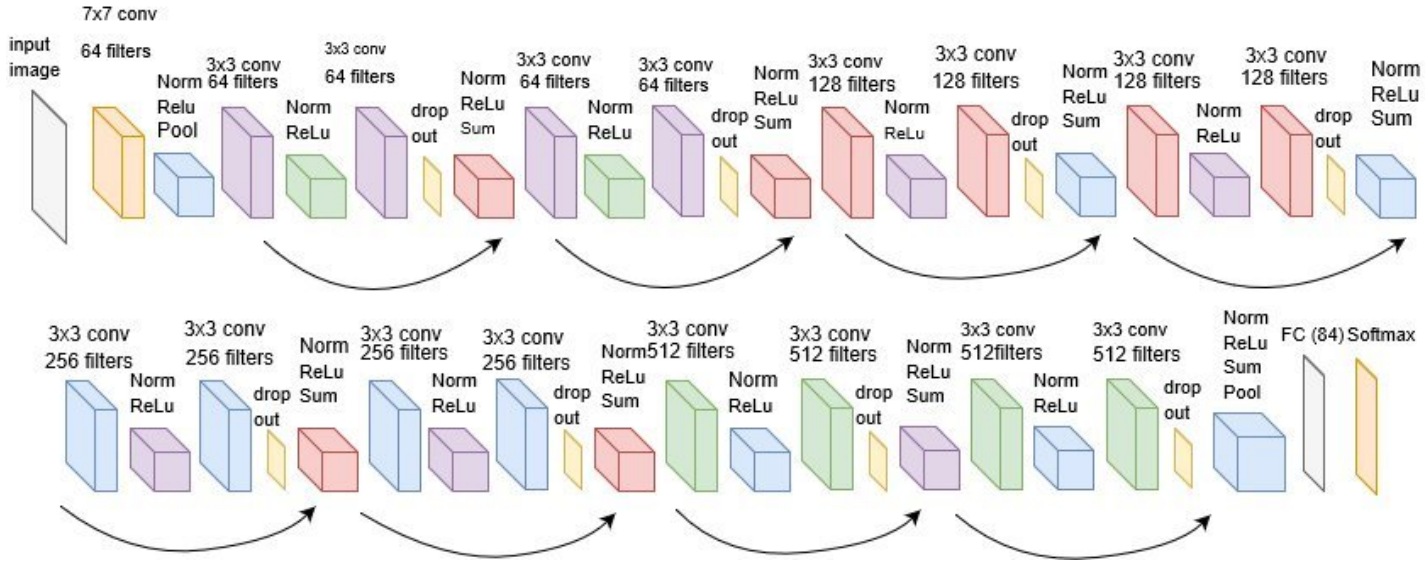
Each matrix W is a convolution matrix

Repeat a few times

At the end, use a full W for a final matrix multiplication

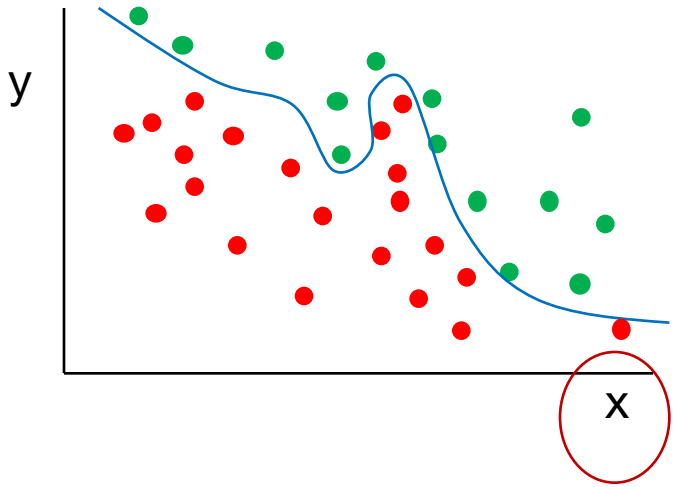
Getting us to Convolutional Neural Networks





Aside #1 before convolutional neural network details

Q: Can we try to avoid making these learning models too complicated?

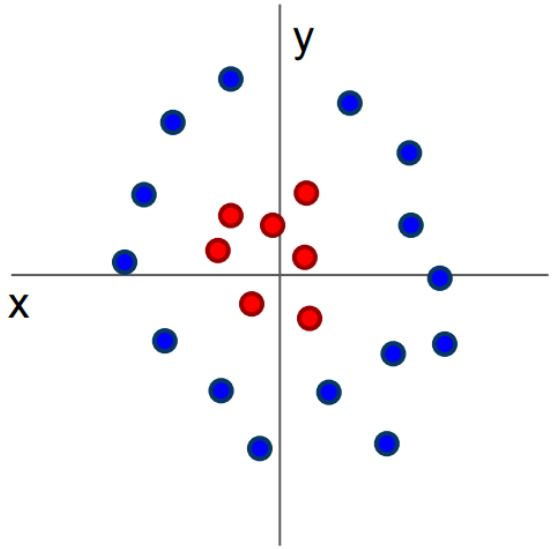


Learned *f:* more flexible

Does it generalize???

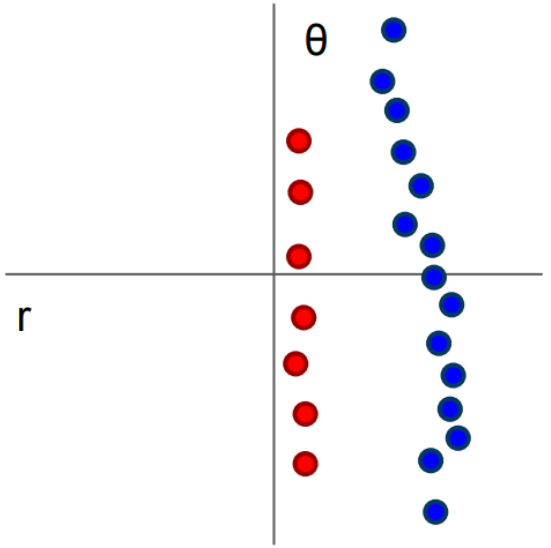
A: Yes, by transforming the data coordinates *before* classification

Image Features: Motivation



Cannot separate red and blue points with linear classifier

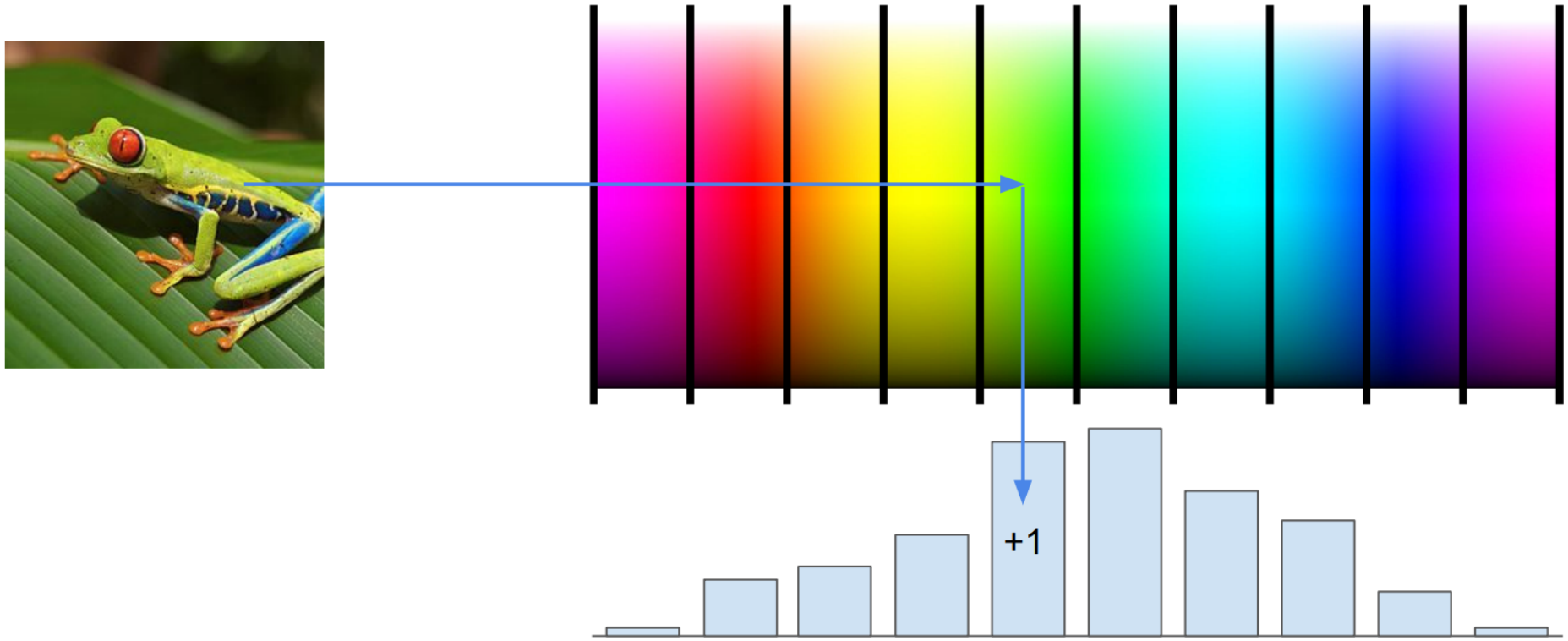
$f(x, y) = (r(x, y), \theta(x, y))$



After applying feature transform, points can be separated by linear classifier

From Stanford CS231

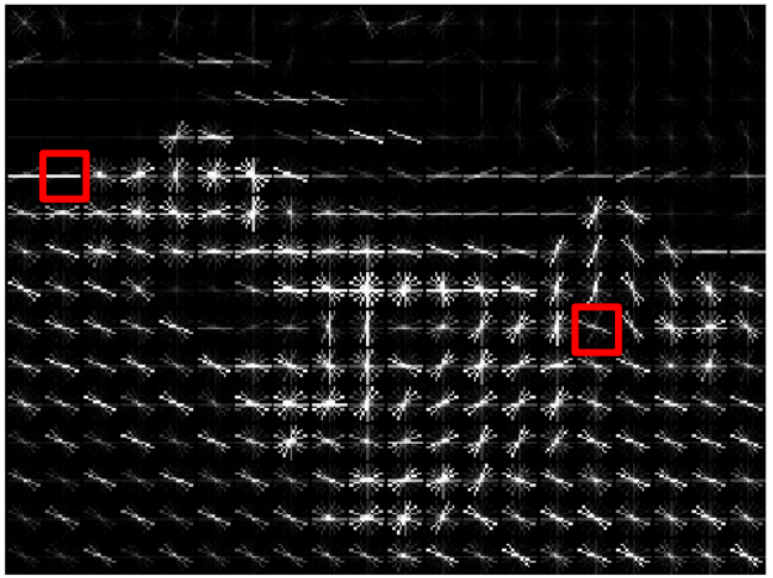
Example: Color Histogram



From Stanford CS231

Example: Histogram of Oriented Gradients (HoG)

Divide image into 8x8 pixel regions
Within each region quantize edge direction into 9 bins



Example: 320x240 image gets divided into 40x30 bins; in each bin there are 9 numbers so feature vector has 30*40*9 = 10,800 numbers

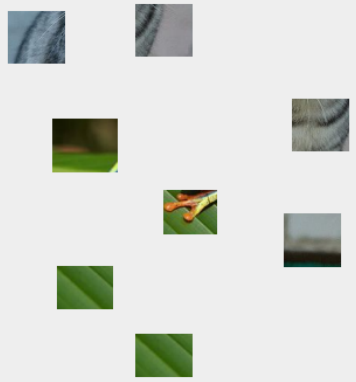
Lowe, “Object recognition from local scale-invariant features”, ICCV 1999
Dalal and Triggs, "Histograms of oriented gradients for human detection," CVPR 2005

From Stanford CS231

Example: Bag of Words

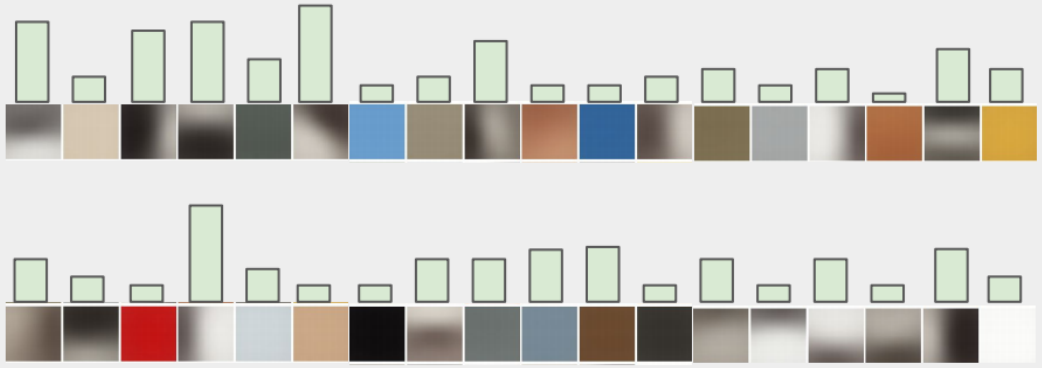
Step 1: Build codebook

Extract random patches



Cluster patches to form “codebook” of “visual words”

Step 2: Encode images

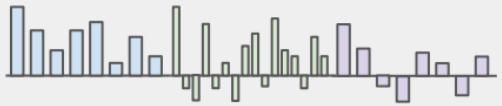


Fei-Fei and Perona, “A bayesian hierarchical model for learning natural scene categories”, CVPR 2005

From Stanford CS231

Image features vs ConvNets

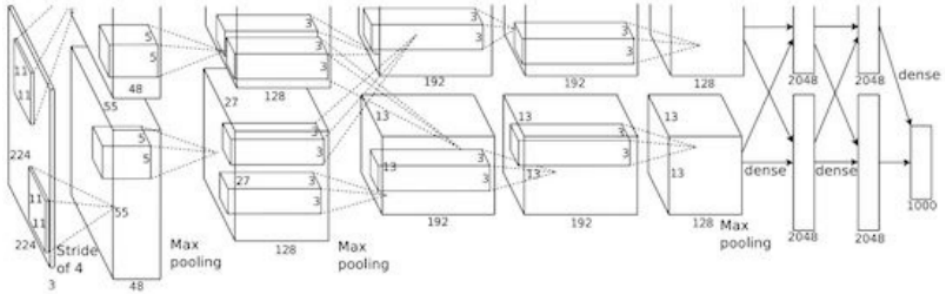
Feature Extraction



f

10 numbers giving scores for classes

training



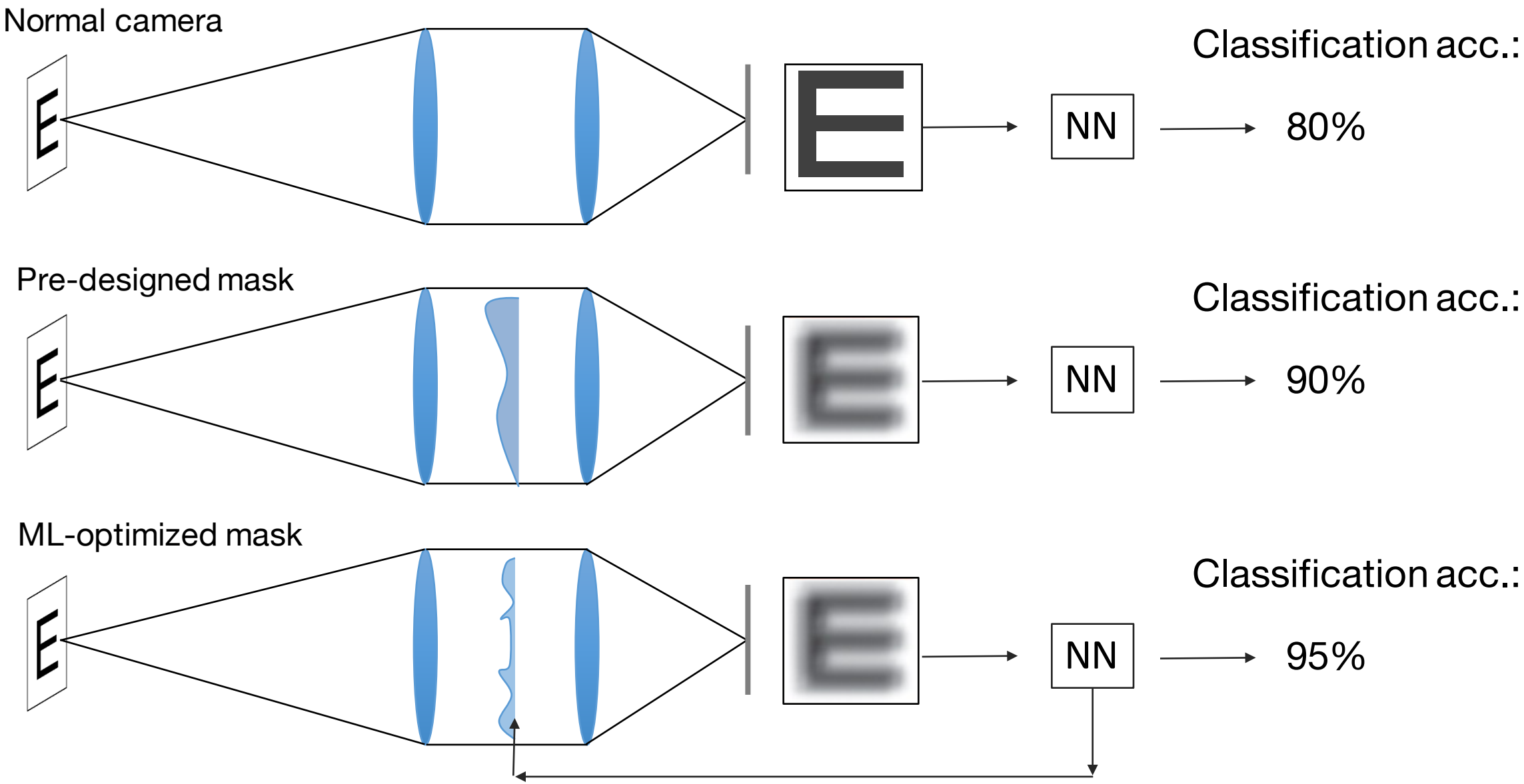
Krizhevsky, Sutskever, and Hinton, "Imagenet classification with deep convolutional neural networks", NIPS 2012. Figure copyright Krizhevsky, Sutskever, and Hinton, 2012. Reproduced with permission.

10 numbers giving scores for classes

training

From Stanford CS231

Hand-crafted versus learned features also applies to imaging



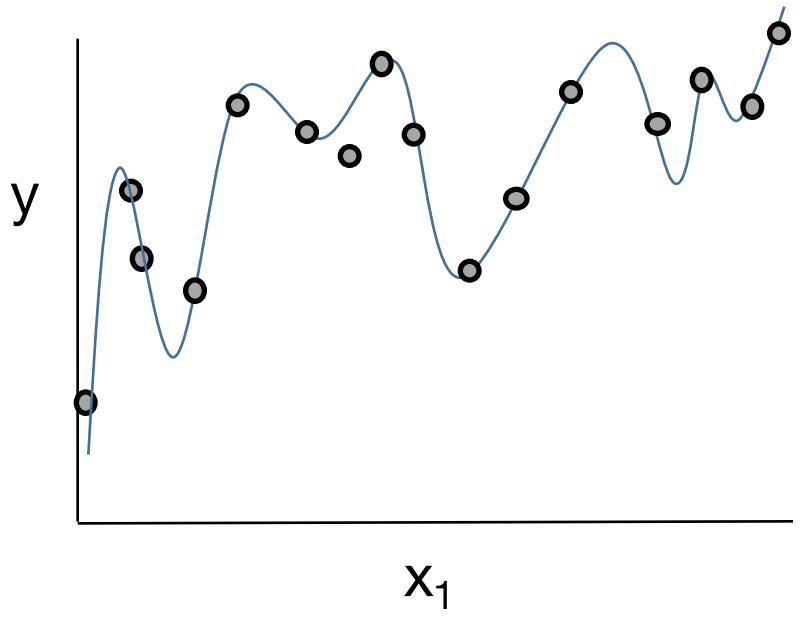
Aside #2 before CNN's: machine learning has two competing goals

1. Can we make sure the in-sample error $L_{in}(y, f(x,W))$ is small enough?
   - Appropriate cost function
   - "complex enough" model

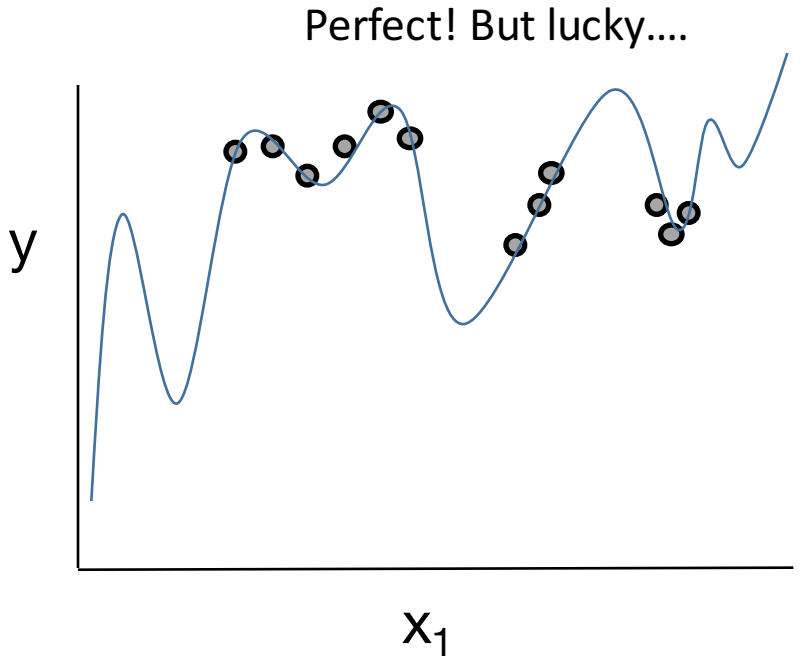
2. Can we make sure that $L_{out}(y, f(x,W))$ is close enough to $L_{in}(y, f(x,W))$?
   - Probabilistic analysis says yes!
   - $|L_{in} - L_{out}|$ bounded from above
   - Bound grows with model capacity (bad)
   - Bound shrinks with # of training examples (good)

Model overfitting versus underfitting – a thought exercise

Let's fit these "training" data points:

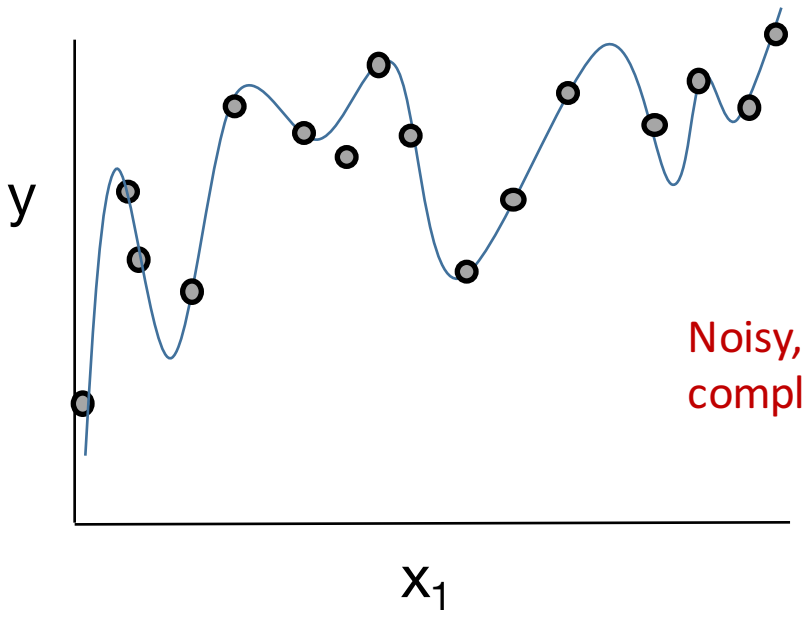


And then here's our testing dataset – good?

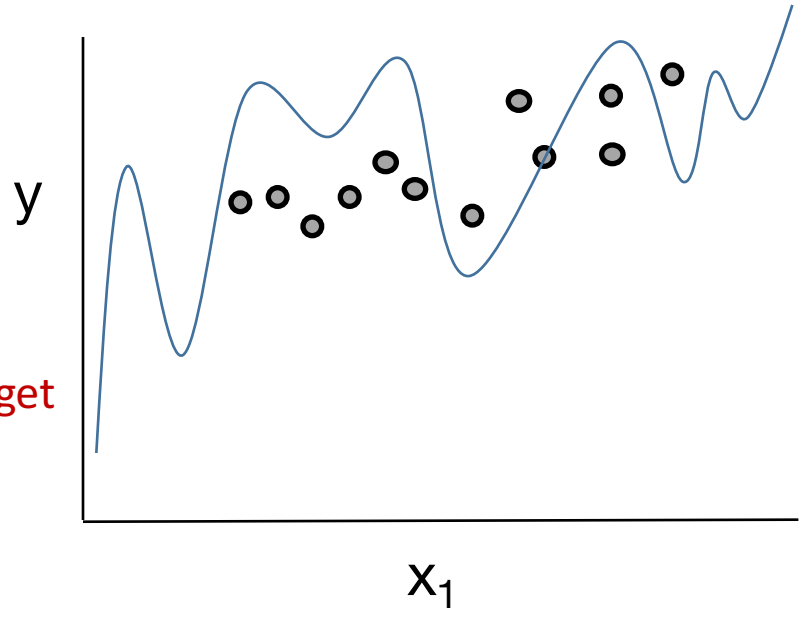


Model overfitting versus underfitting – a thought exercise

Let's fit these "training" data points:



What if our test dataset was this :

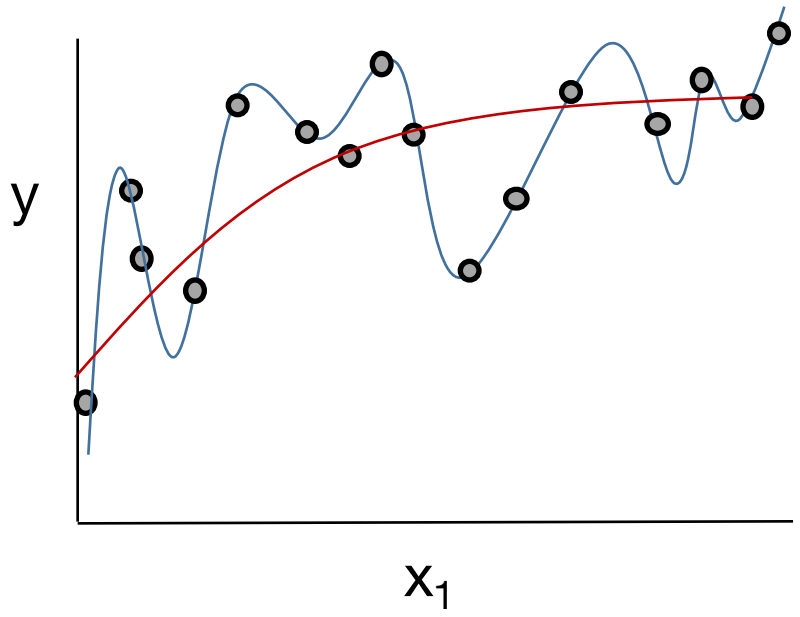


Noisy, low complexity target

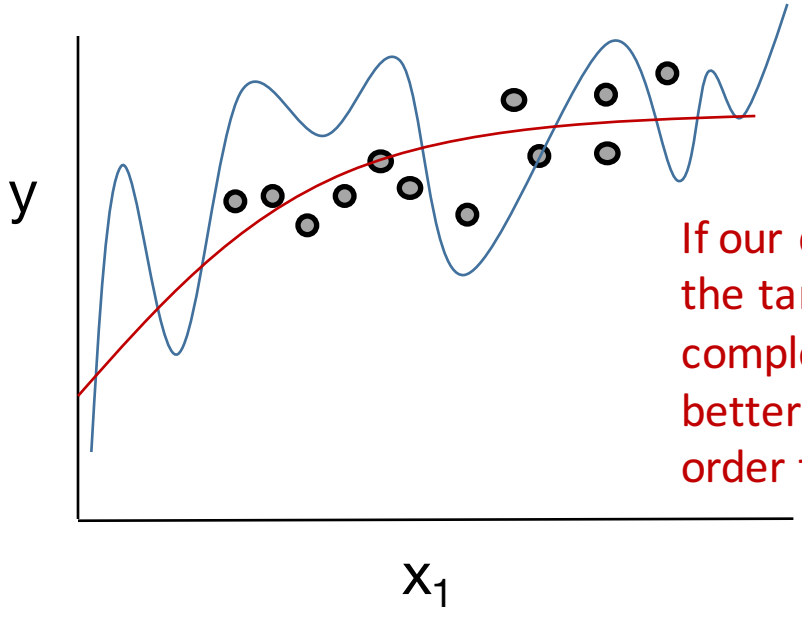
10th order Polynomial Fit

Model overfitting versus underfitting – a thought exercise

Let's fit these "training" data points:

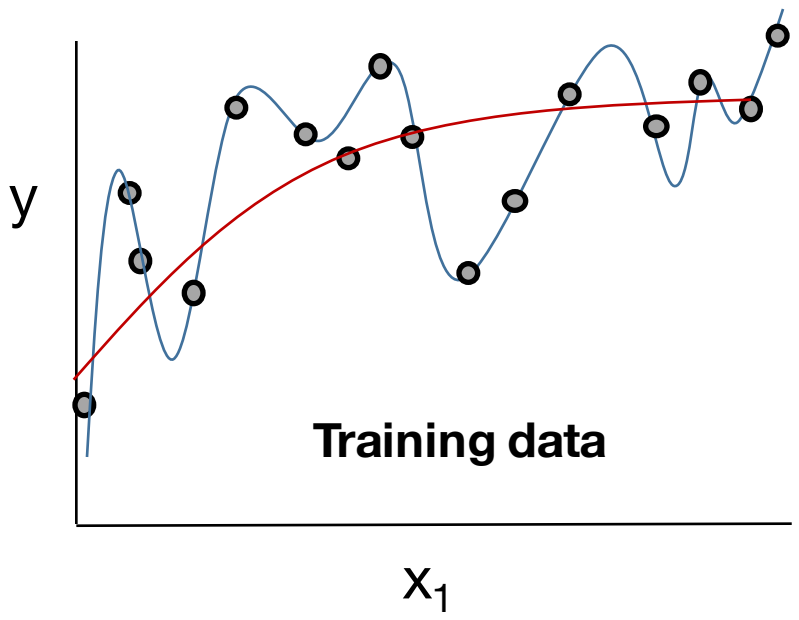


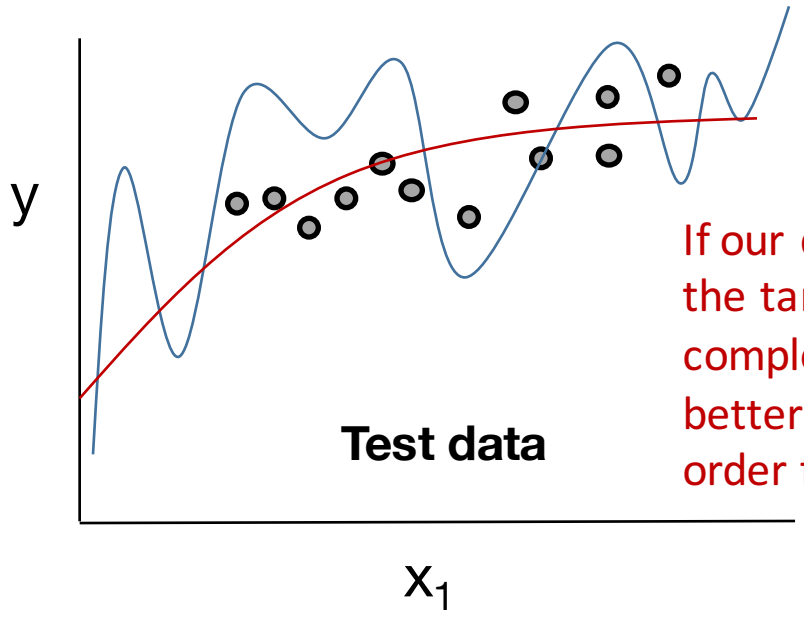
What if our test dataset was this :



If our data was noisy and the target followed a low-complexity model, we'd be better off with a second order fit!

Model overfitting versus underfitting



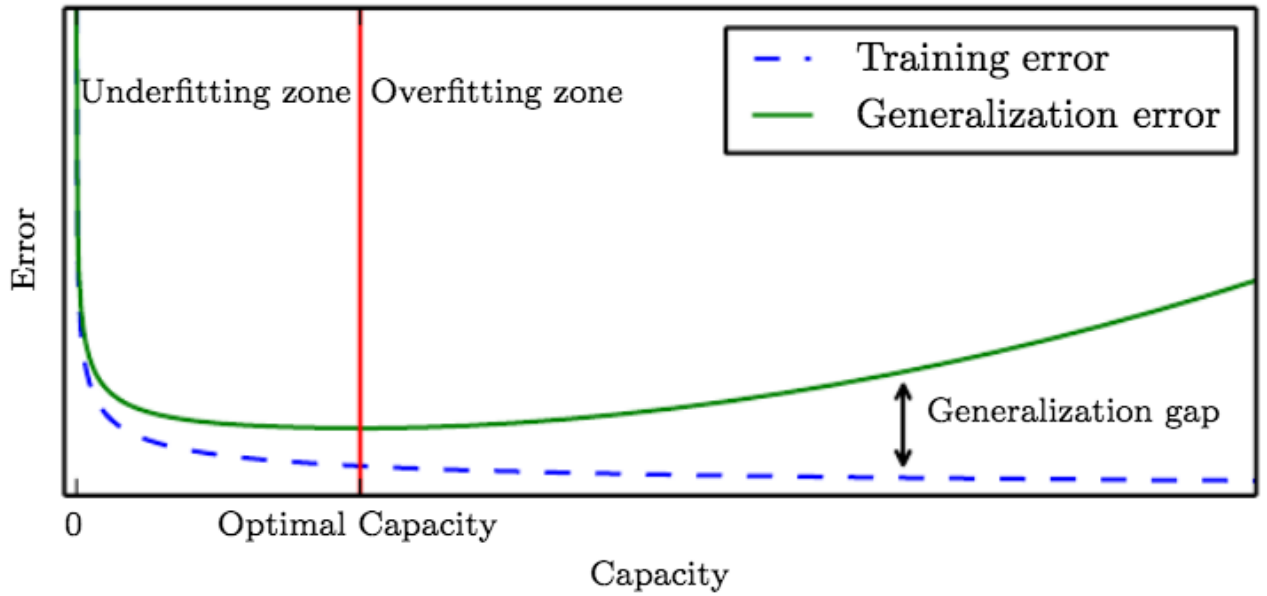


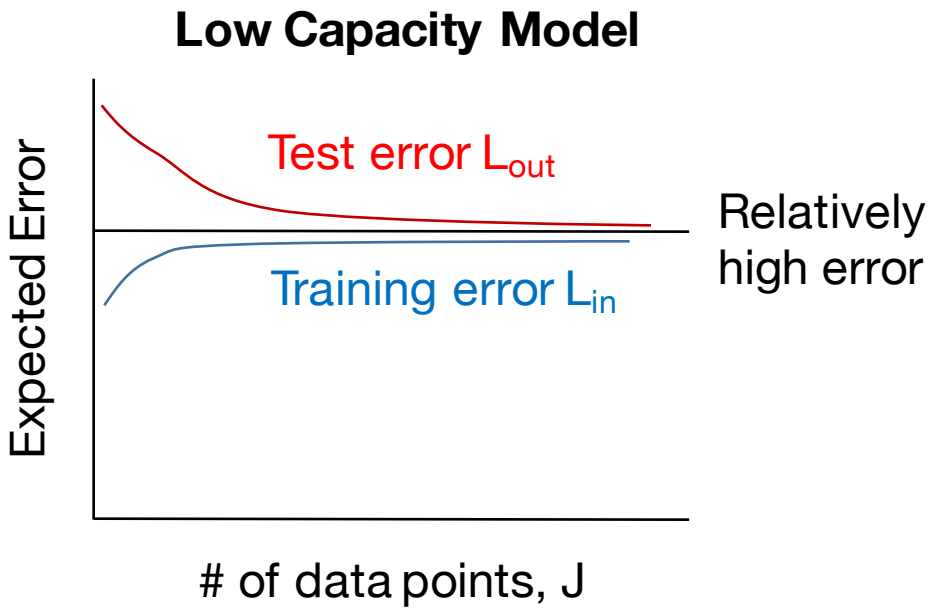
If our data was noisy and the target followed a low-complexity model, we'd be better off with a second order fit!

Model capacity: ability to fit a wide range of functions

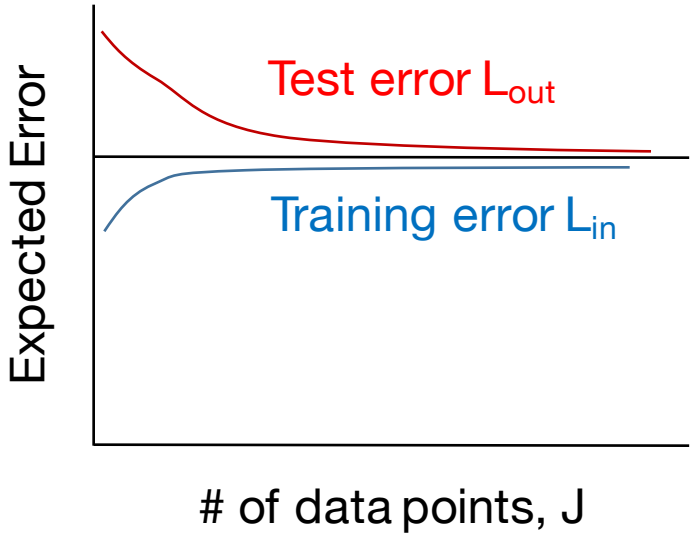
Control capacity through model's hypothesis space (set of functions model can take)

Hard to know ahead of time!



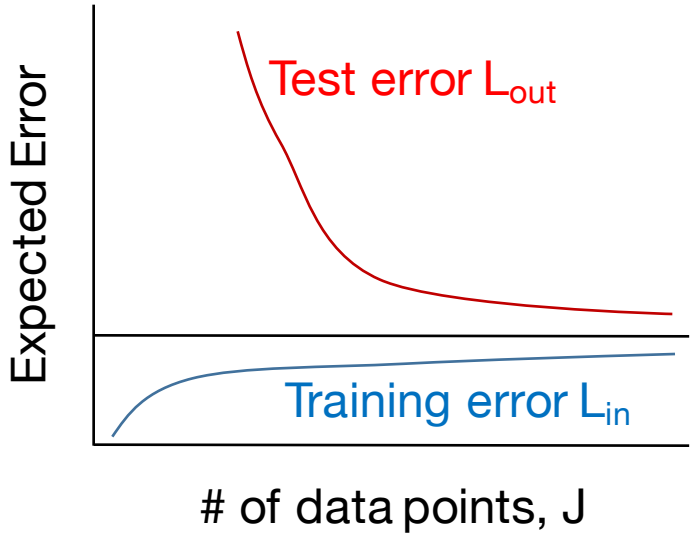

Low Capacity Model
Test error L_{out}
Training error L_{in}
Relatively high error
Expected Error
# of data points, J

Low Capacity Model



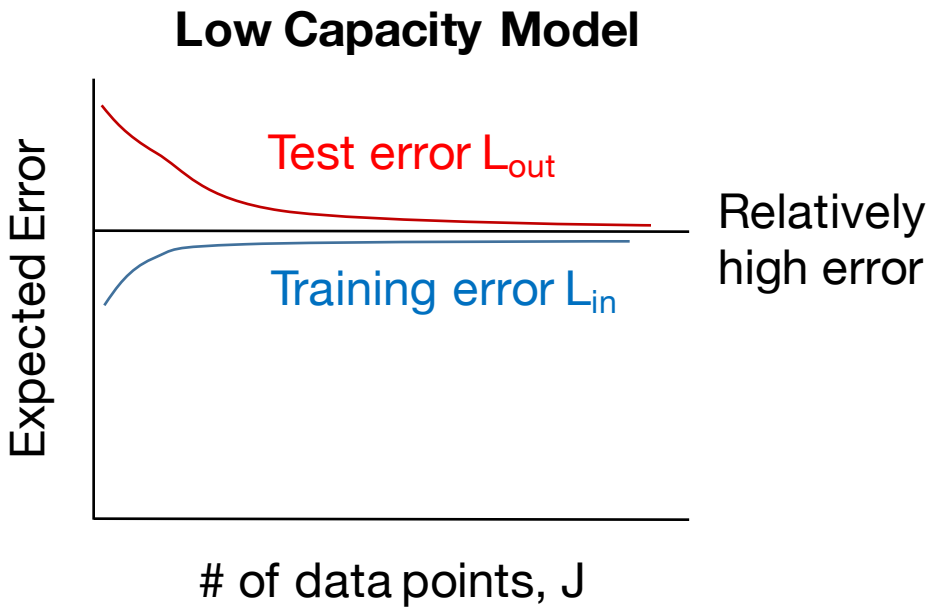
Relatively high error

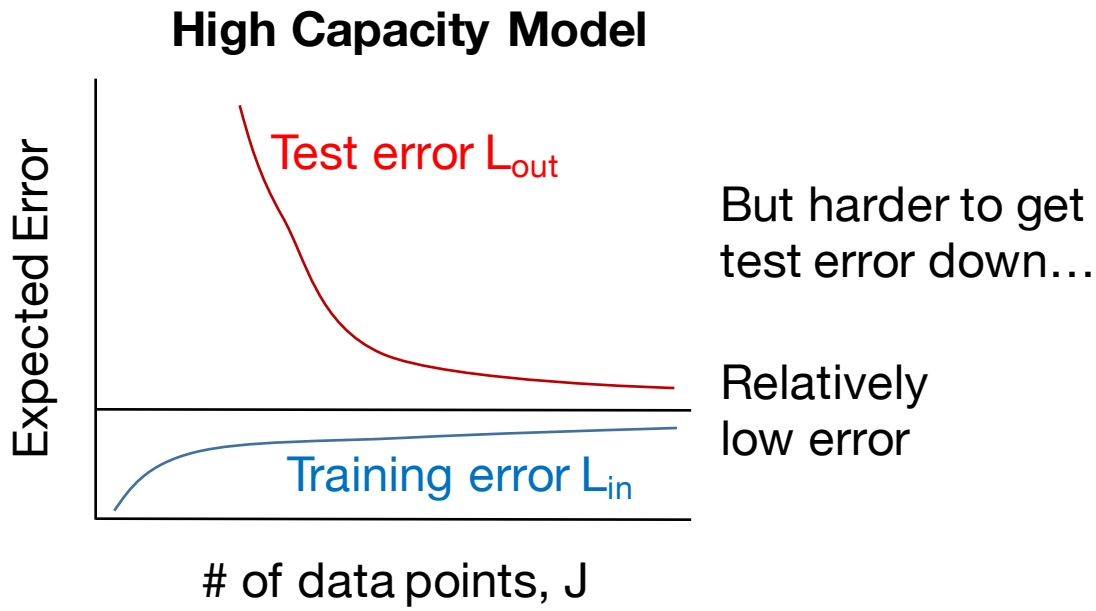
High Capacity Model



But harder to get test error down…

Relatively low error

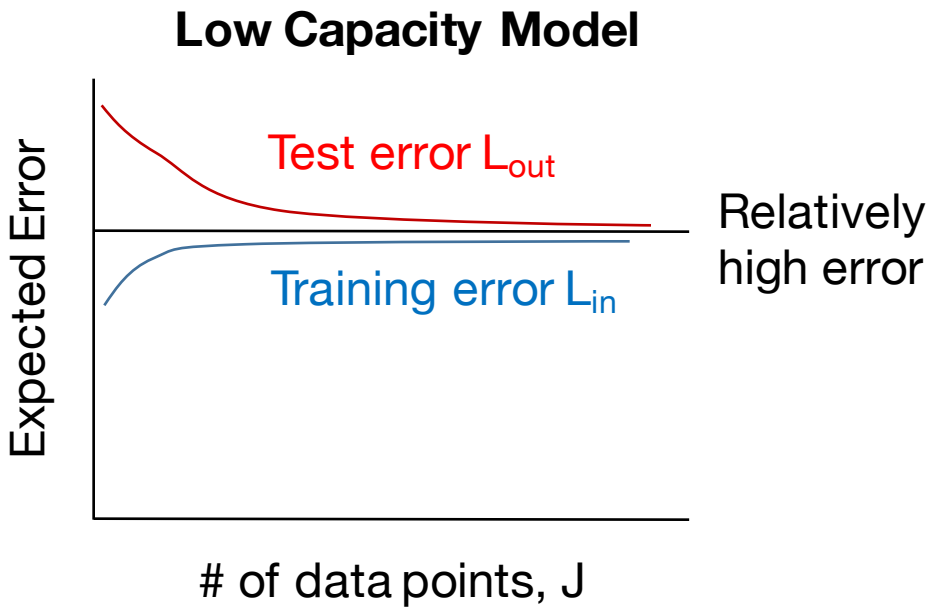




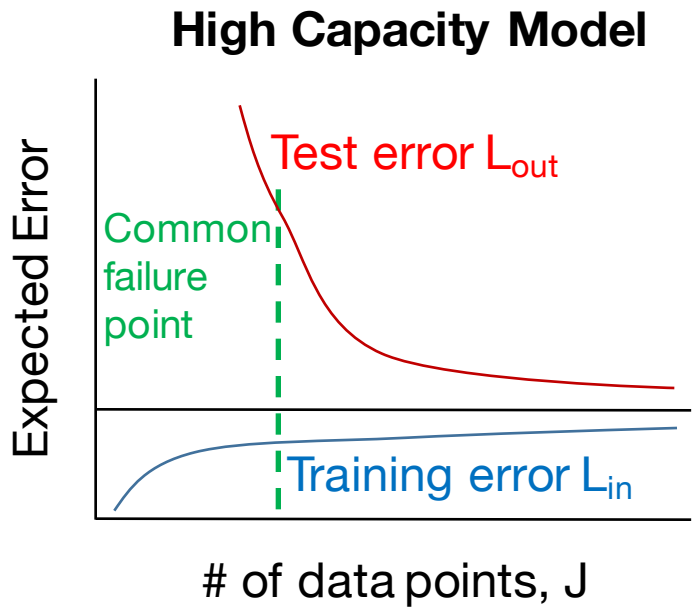
CNN's are this type of model

Take away concepts:

- Can't ever really expect test error to be less than training error
- Complicated models tend to appear to "do better" during training, before trying test data
- When the model gets complicated and you don't have enough data, challenging to get test error down



Relatively high error



But harder to get test error down…

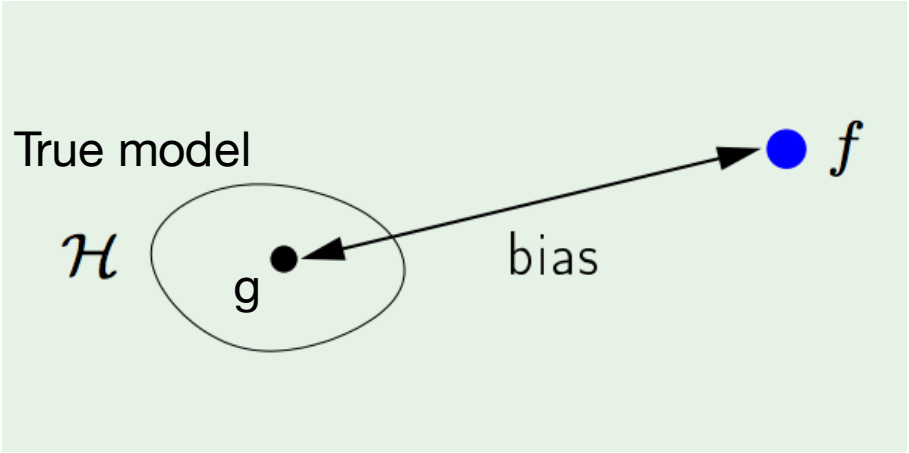
Relatively low error

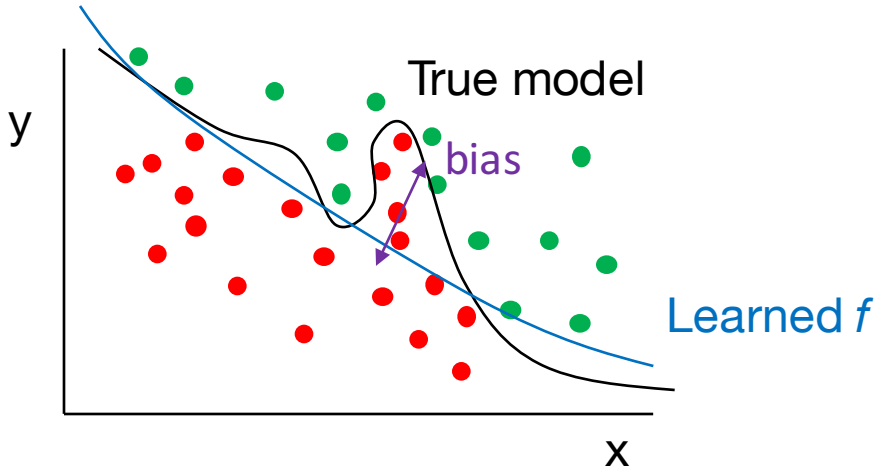
CNN's are this type of model

Take away concepts:

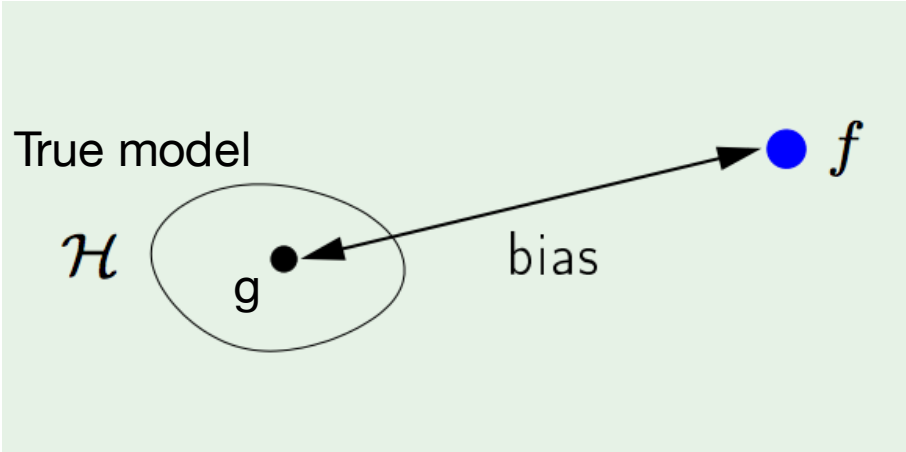
- Can't ever really expect test error to be less than training error
- Complicated models tend to appear to "do better" during training, before trying test data
- When the model gets complicated and you don't have enough data, challenging to get test error down

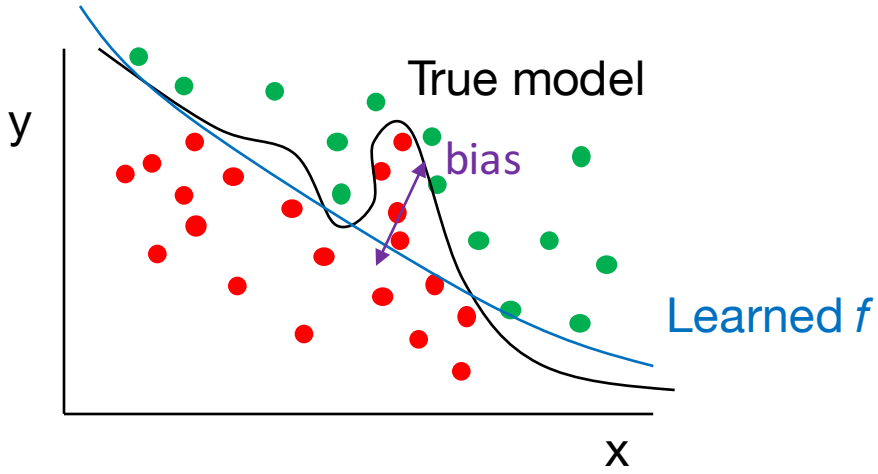
Model bias versus variance





Model bias versus variance



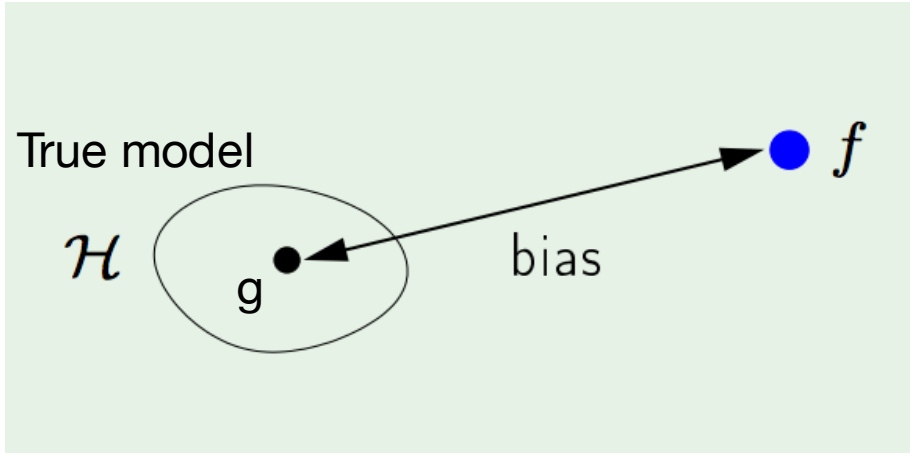


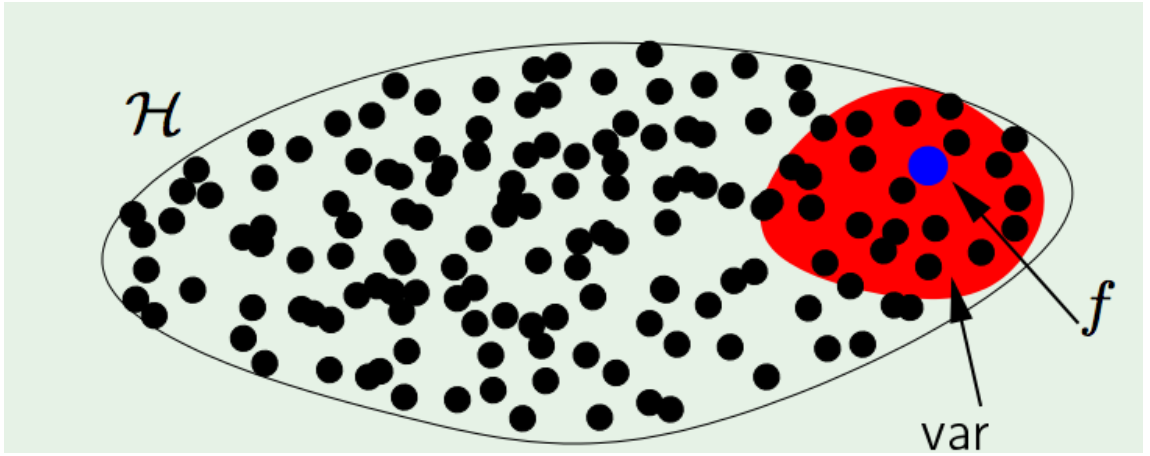
Models that tend to be “a bit too simple” are biased away from ”true” model

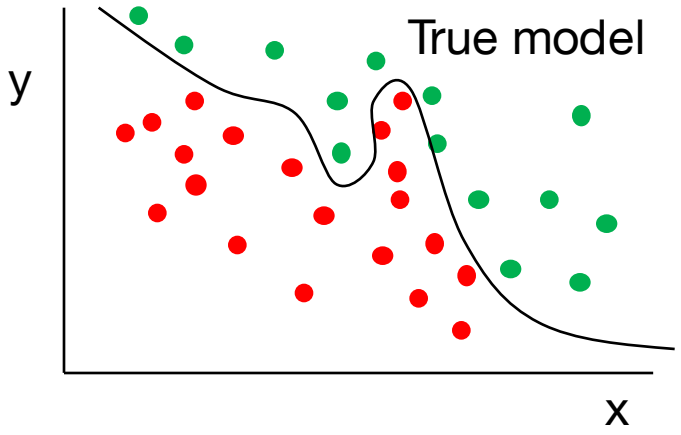
$$\text{Bias} = (g(\mathbf{x}) - f(\mathbf{x}))^2$$

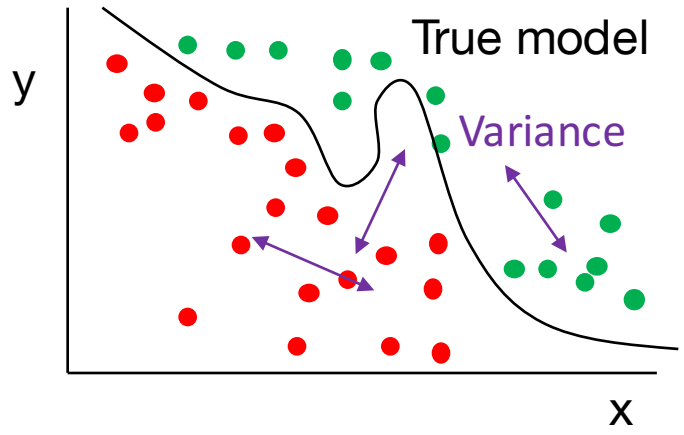
Measures how far our learning model f is biased away from target function g (for perfect training data classification)

Model bias versus variance





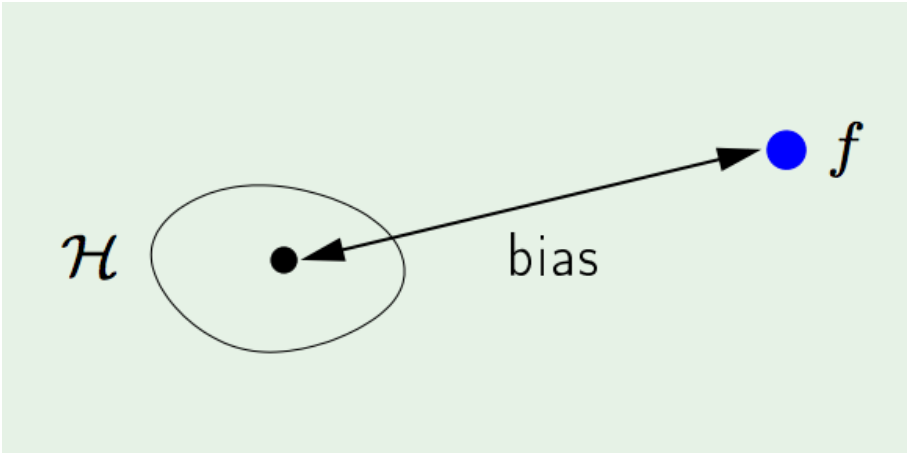


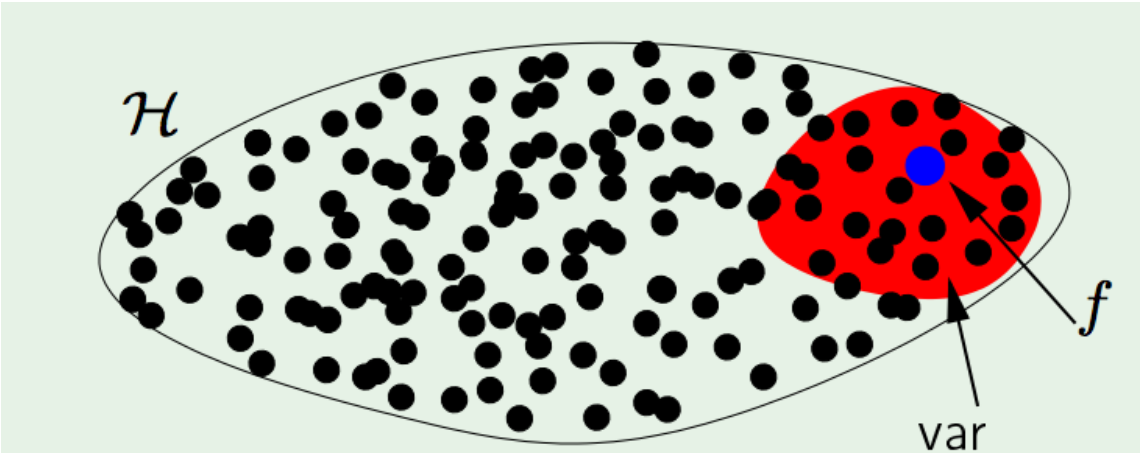


Variance = Var[g(**x**)]

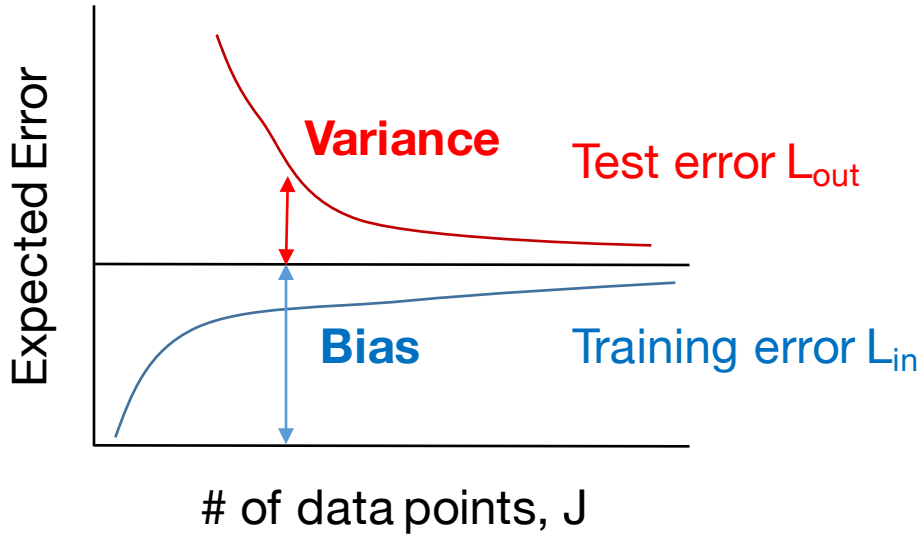
More complicated datasets exhibit lots of variance between training and test set

Model bias versus variance





Error vs. # data points



Test Error is sum of model bias and variance!

Goal is to find a model *f* that balances between these two quantities for a given dataset

How to formally define capacity and complexity?

- Short answer: it's complicated…
- Related to something called the VC Dimension
  - Can provide theoretical bounds on performance
  - Tend to be dimensional bounds rather than scalar bounds…
- I decided not to go into it, but please let me know if you'd like me to!

Conclusions from statistical machine learning

- Conclusion: you want a model that is complex enough to capture variations within high-dimensional space, but not too complex such that it overfits the data

- Want a model with a high capacity, but can still *generalize* to data outside training set
  - More data -> less overfitting, complex target -> more overfitting

- For simple models, we can measure complexity via degrees of freedom, the VC bound and so-on to help us nail down ideal models that can generalize well

Conclusions from statistical machine learning

- Conclusion: you want a model that is complex enough to capture variations within high-dimensional space, but not too complex such that it overfits the data

- Want a model with a high capacity, but can still *generalize* to data outside training set
  - More data -> less overfitting, complex target -> more overfitting

- For simple models, we can measure complexity via degrees of freedom, the VC bound and so-on to help us nail down ideal models that can generalize well

- **For DL models**: this will get too hard…here's a few counter-intuitive properties:

1. A fixed DL *architecture* exhibits data-dependent complexities
   - e.g., "good" DL networks achieve 0 training error on images with random labels, so cannot generalize at all in this case, and are too complex

2. DL networks with more hidden units leads to *better* generalization (the main finding of the last few years). So deeper models tend to be less complex, actually…

3. Complexity depends upon loss function and optimization method…

Important to remember: “No Free Lunch Theorem”

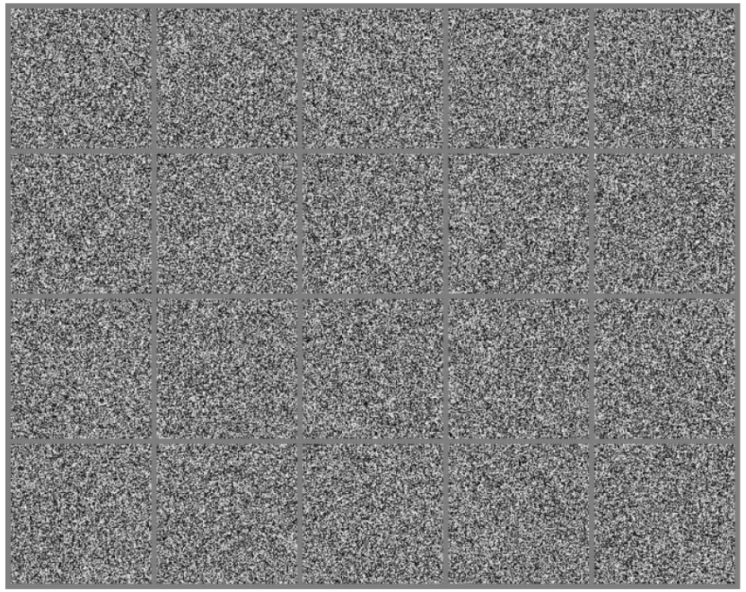
- *“Averaged over all possible data-generating distributions, every classification algorithm has the same error rate when classifying previously unobserved points.”*

- The most sophisticated DL algorithm has save average performance (averaged over all possible tasks) as the simplest.

Important to remember: “No Free Lunch Theorem”

- *“Averaged over all possible data-generating distributions, every classification algorithm has the same error rate when classifying previously unobserved points.”*
- The most sophisticated DL algorithm has save average performance (averaged over all possible tasks) as the simplest.
- Must make assumptions about probability distributions of inputs we’ll encounter in real-world

Set of 20 “images”, random Gaussian distribution

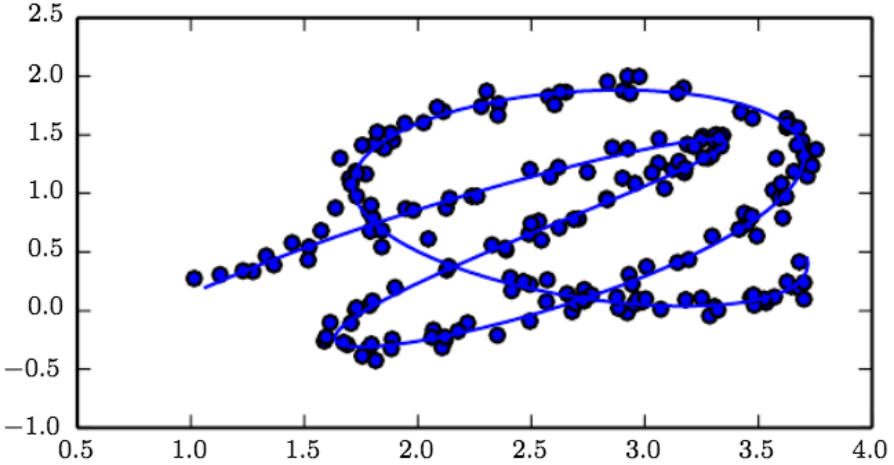


Face at different orientations = manifold n-D space

Important to remember: “No Free Lunch Theorem”

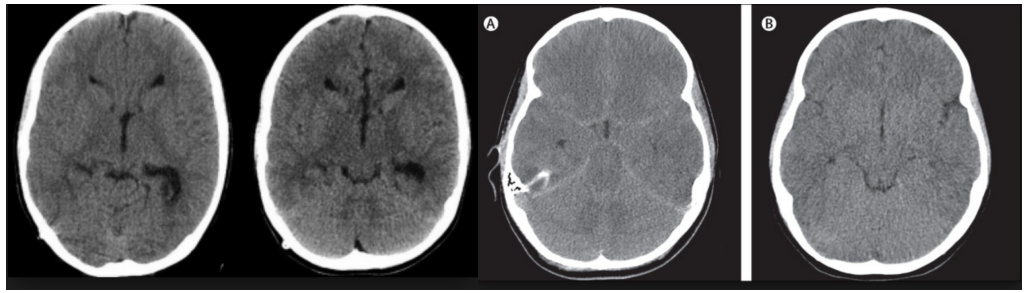
- *“Averaged over all possible data-generating distributions, every classification algorithm has the same error rate when classifying previously unobserved points.”*
- The most sophisticated DL algorithm has save average performance (averaged over all possible tasks) as the simplest.
- Must make assumptions about probability distributions of inputs we’ll encounter in real-world

1D Manifold in 2D space



Manifold Hypothesis

CT reconstructions of every brain in the world = kD manfold in nD space?

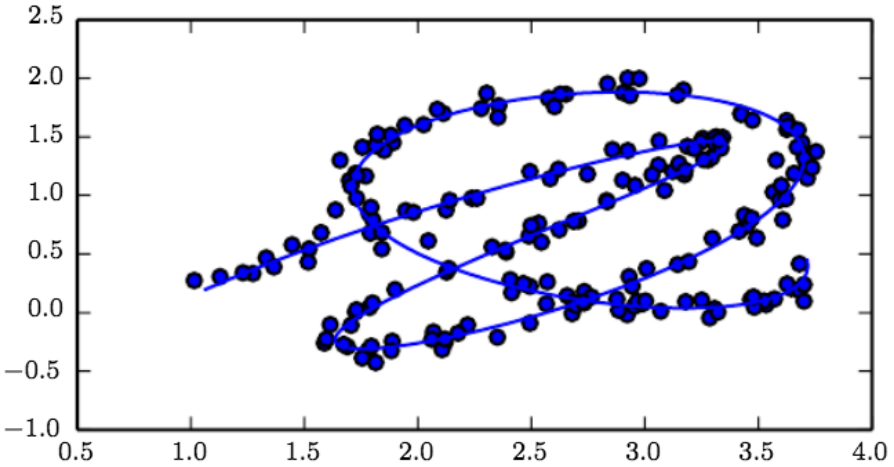


...

Important to remember: “No Free Lunch Theorem”

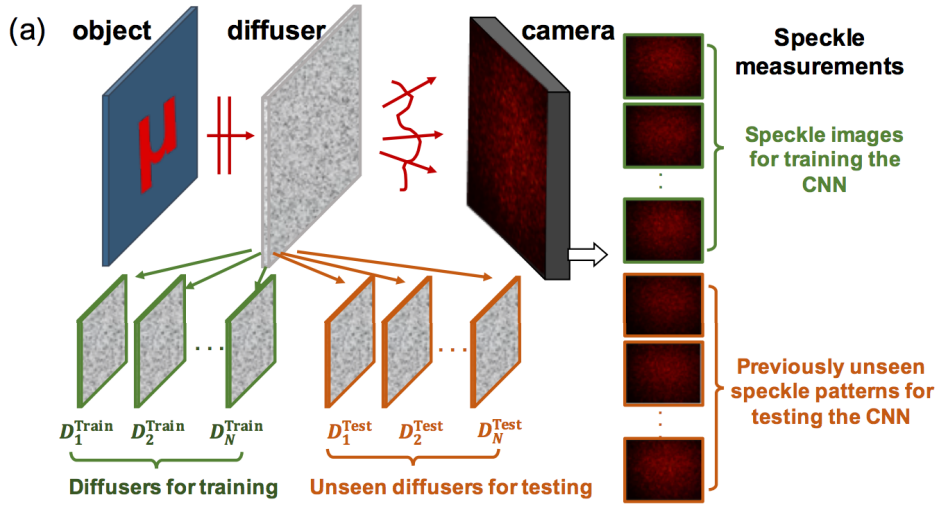
- *“Averaged over all possible data-generating distributions, every classification algorithm has the same error rate when classifying previously unobserved points.”*
- The most sophisticated DL algorithm has save average performance (averaged over all possible tasks) as the simplest.
- Must make assumptions about probability distributions of inputs we’ll encounter in real-world

1D Manifold in 2D space



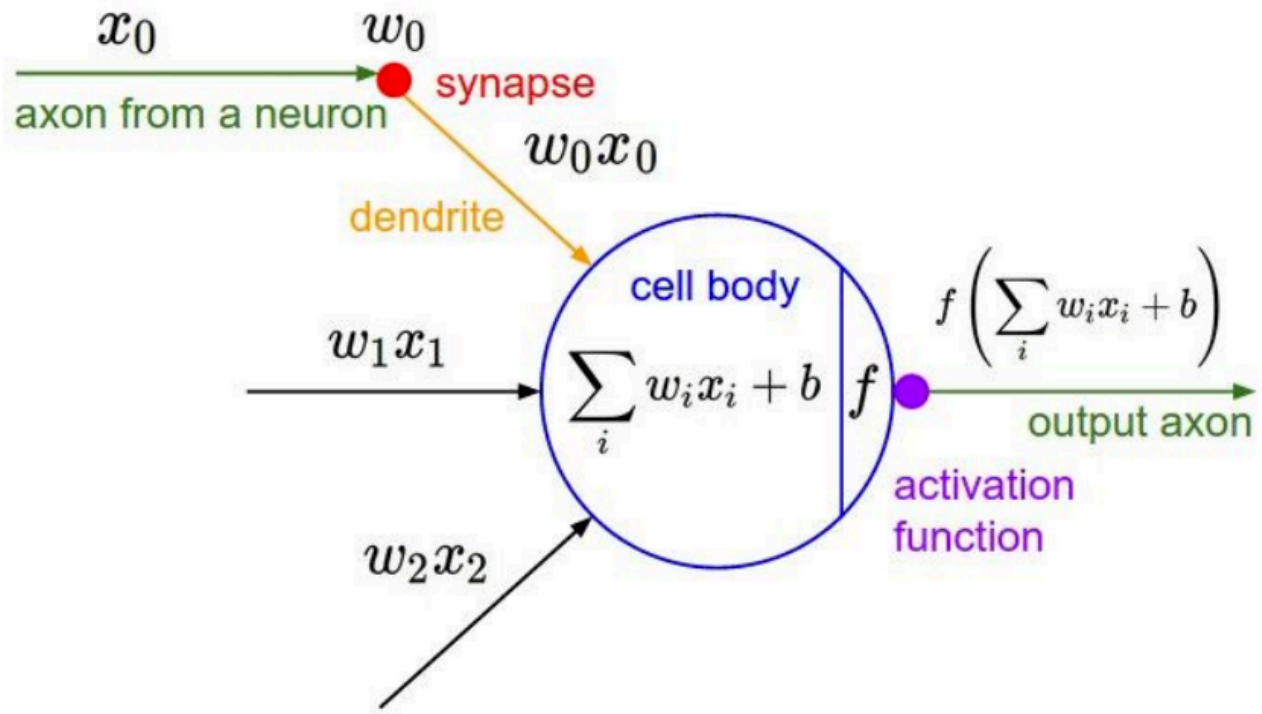
Manifold Hypothesis

CT reconstructions of every brain in the world = kD manfold in nD space?



. . .

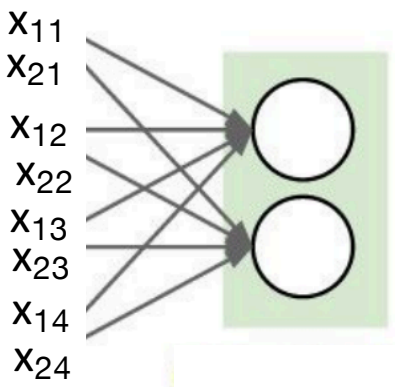
No more asides! Now lets get into neural networks…



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.)

No more asides! Now lets get into neural networks…



NL•

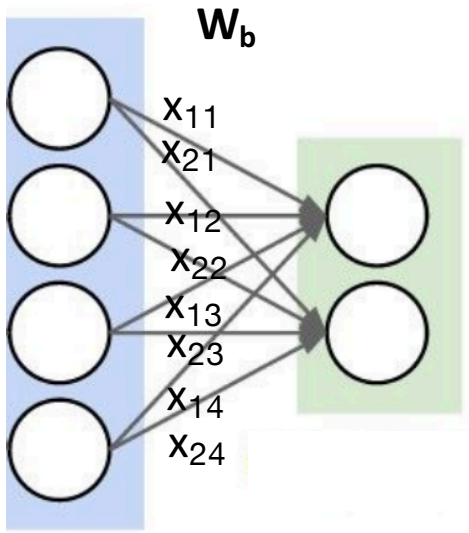
W

w_{11}	w_{12}	w_{13}	w_{14}
w_{21}	w_{22}	w_{23}	w_{24}

x_1	**x_2**
x_{11}	x_{21}
x_{12}	x_{22}
x_{13}	x_{23}
x_{14}	x_{24}

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

No more asides! Now lets get into neural networks…



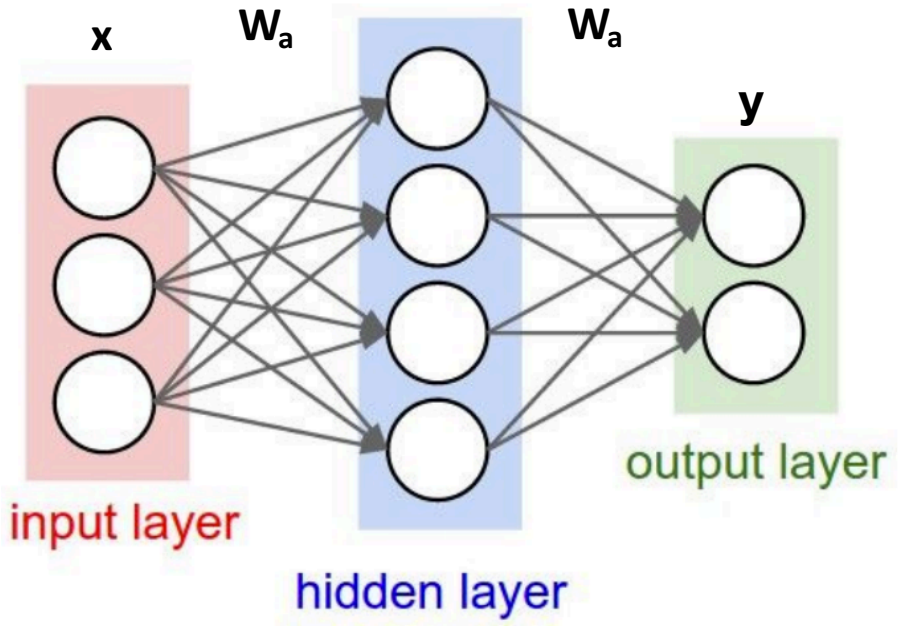
NL•

W_b			
w_{11}	w_{12}	w_{13}	w_{14}
w_{21}	w_{22}	w_{23}	w_{24}

x_1	x_2
x_{11}	x_{21}
x_{12}	x_{22}
x_{13}	x_{23}
x_{14}	x_{24}

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

No more asides! Now lets get into neural networks…



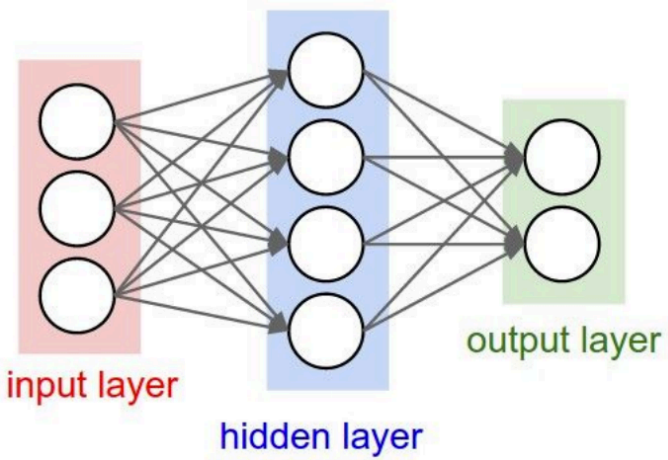
$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = NL\bullet \begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \end{bmatrix} NL\bullet \begin{bmatrix} w_{11} & w_{21} & w_{21} \\ w_{12} & w_{22} & w_{22} \\ w_{13} & w_{23} & w_{23} \\ w_{14} & w_{24} & w_{24} \end{bmatrix}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

$\mathbf{y}$ $\mathbf{W_b}$ $\mathbf{W_a}$ $\mathbf{x}$

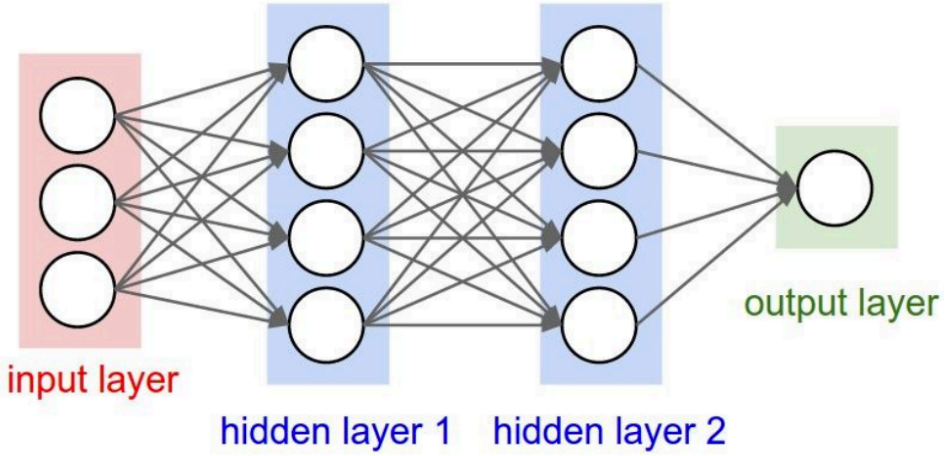
- For multiple cells (units), use matrix **W** to connect inputs to outputs
- These cascade in layers

Next step: Neural networks! Pretty much the same thing…

2-layer network:



3-layer network:



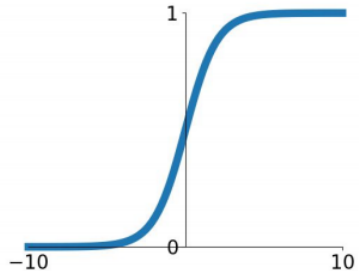
or 3-layer Neural Network

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

Non-linear “activation” functions besides max(·)

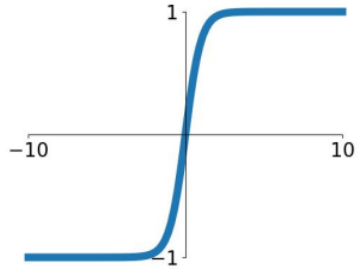
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



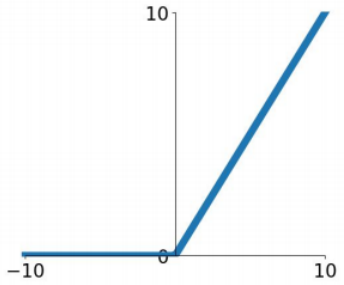
tanh

$$\tanh(x)$$



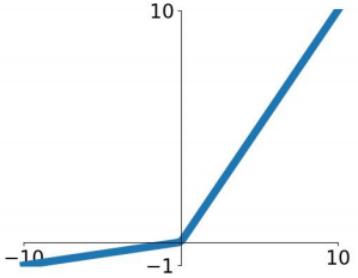
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

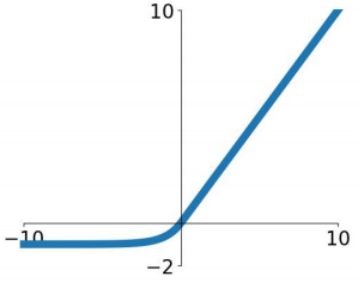


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



From Stanford CS231

What this boils down to: our simple loss function will continue to get more complex

$$L = \frac{1}{N}\sum_{i=1}^{N}(w^T x_i - y_i)^2$$

$$L(w) = \frac{1}{N}\sum_{i=1}^{N}(\max(w^T x_i, 0) - y_i)^2$$

1. Add a non-linear threshold to the linear model

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

2. Cross-entropy loss function

$$L(w) = \frac{1}{N}\sum_{i=1}^{N} ln(1 + e^{-y_i W_2 \max(W_1 x_i, 0)})$$

3. Add another layer

4. What else? More layers, convolutions…Regularization!

One last ingredient: regularization:

λ = regularization strength (hyperparameter)

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{Data loss}} + \underbrace{\lambda R(W)}_{\text{Regularization}}$$

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too* well on training data

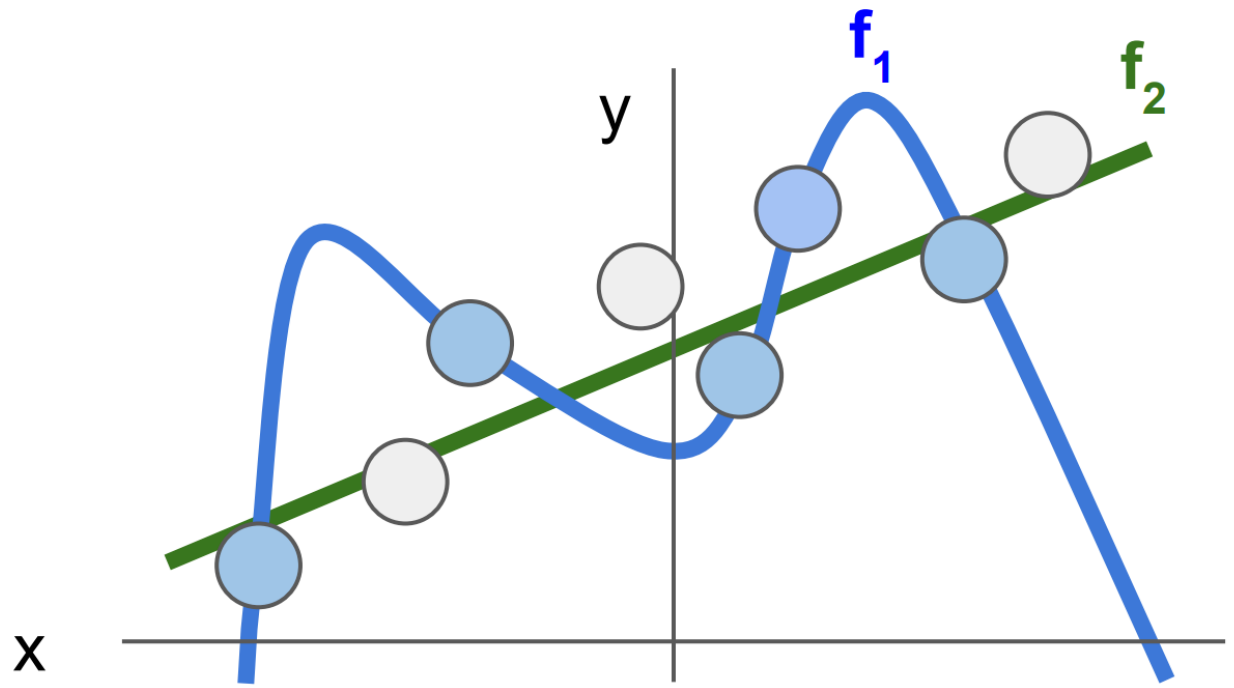
Simple examples

L2 regularization: $R(W) = \sum_k \sum_l W_{k,l}^2$

L1 regularization: $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2): $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

Regularization: Prefer Simpler Models



Regularization pushes against fitting the data *too* well so we don't fit noise in the data

Take away: Useful to add an additional weight-dependent term to the loss function to control capacity and prevent overfitting

From Stanford CS231

The final form of our two-layer neural network:

$$L(w) = \frac{1}{N}\sum_{i=1}^{N} ln(1 + e^{-y_i W_2 \max(W_1 x_i, 0)}) + \lambda(||W_1||_2 + ||W_2||_2)$$

Q: How do we determine the best weights W_1 and W_2 to use from this model?

A: Gradient descent!

Q: How do we figure out the gradients for dL/dW_1 and dL/dW_2

A: Chain rule! (next lectures)

Linear matrices and convolutions

$$L(w) = \frac{1}{N}\sum_{i=1}^{N} ln(1 + e^{-y_i W_2 \max(W_1 x_i, 0)}) + \lambda(||W_1||_2 + ||W_2||_2)$$

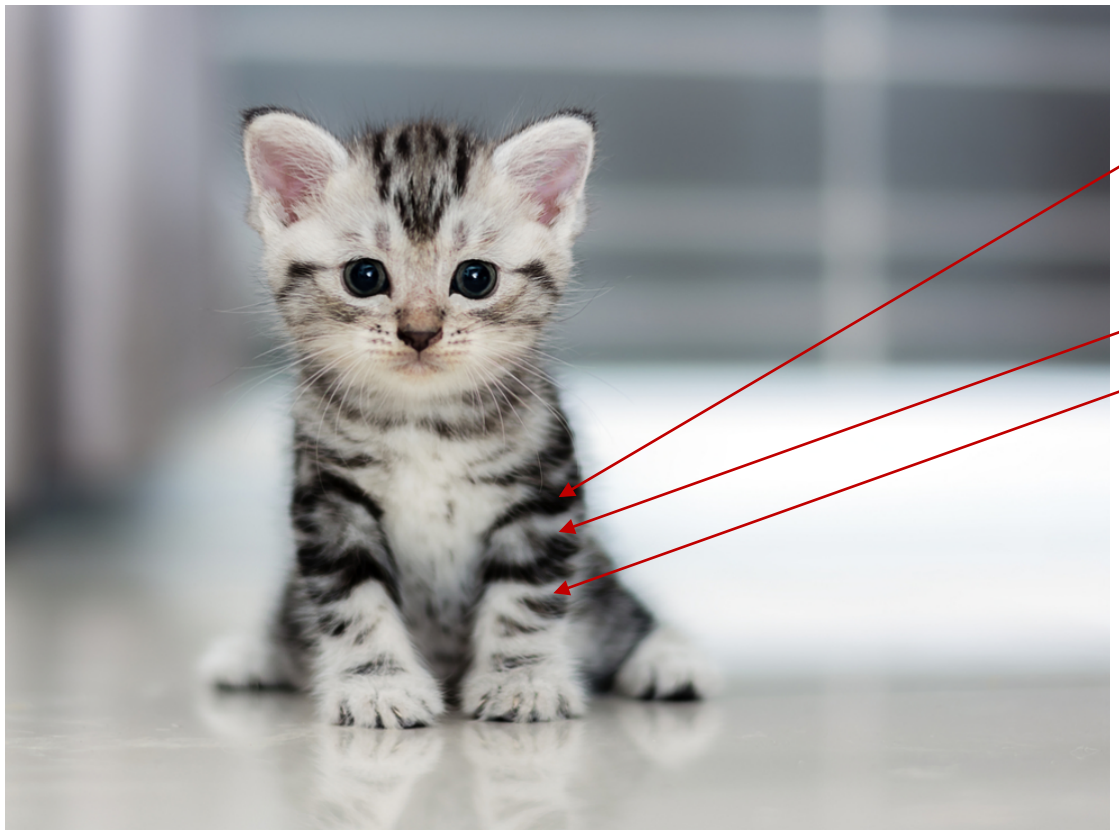
- This is all well and good, but what happens with “large” inputs = small images?
- Having “fully connected” weight matrices can produce quite a lot of weights…

CIFAR10: 32x32 images = 1024 pixels
W1 = 1024x512
W2 = 512x12

Total number of weights: 530,008

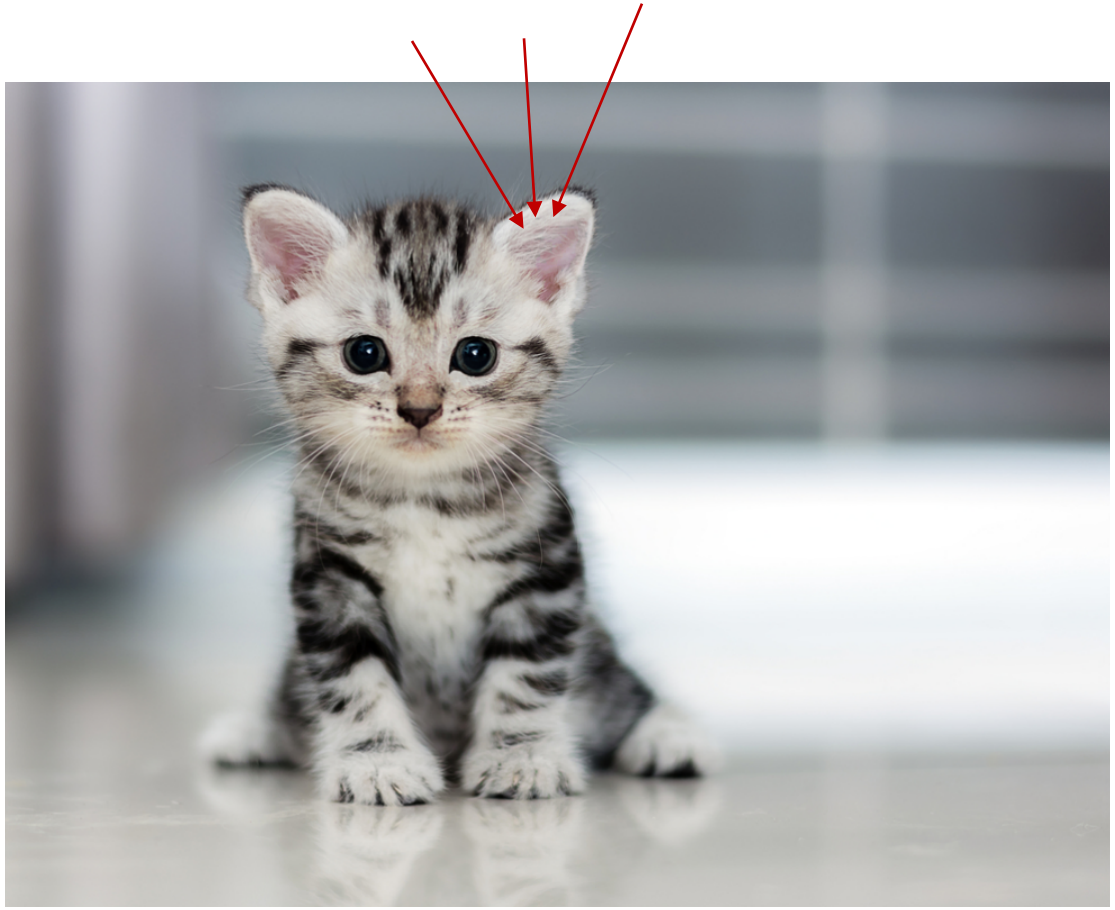
- In addition, classification information is often “local”!

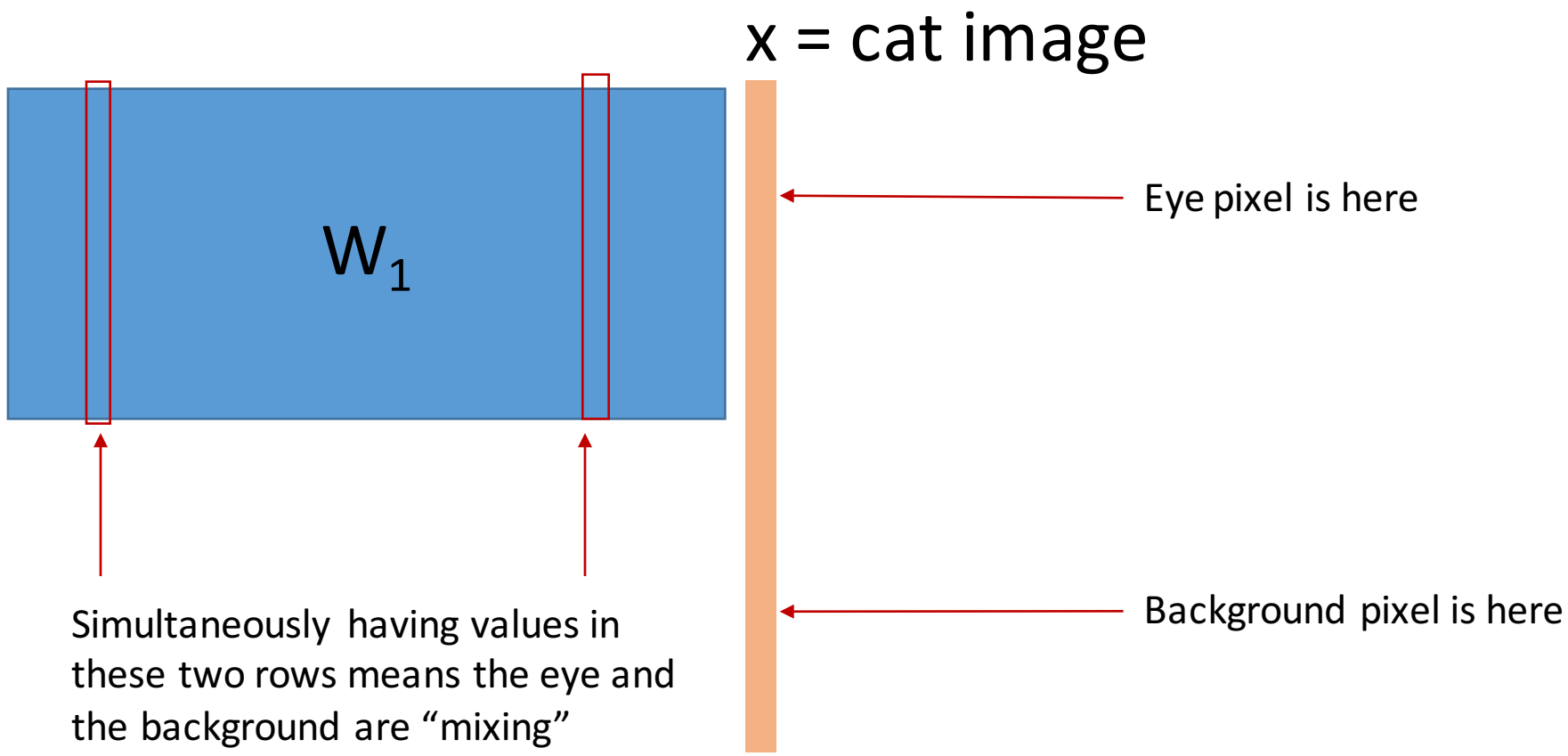
We probably don’t need to mix these two pixels to figure out that this is a cat

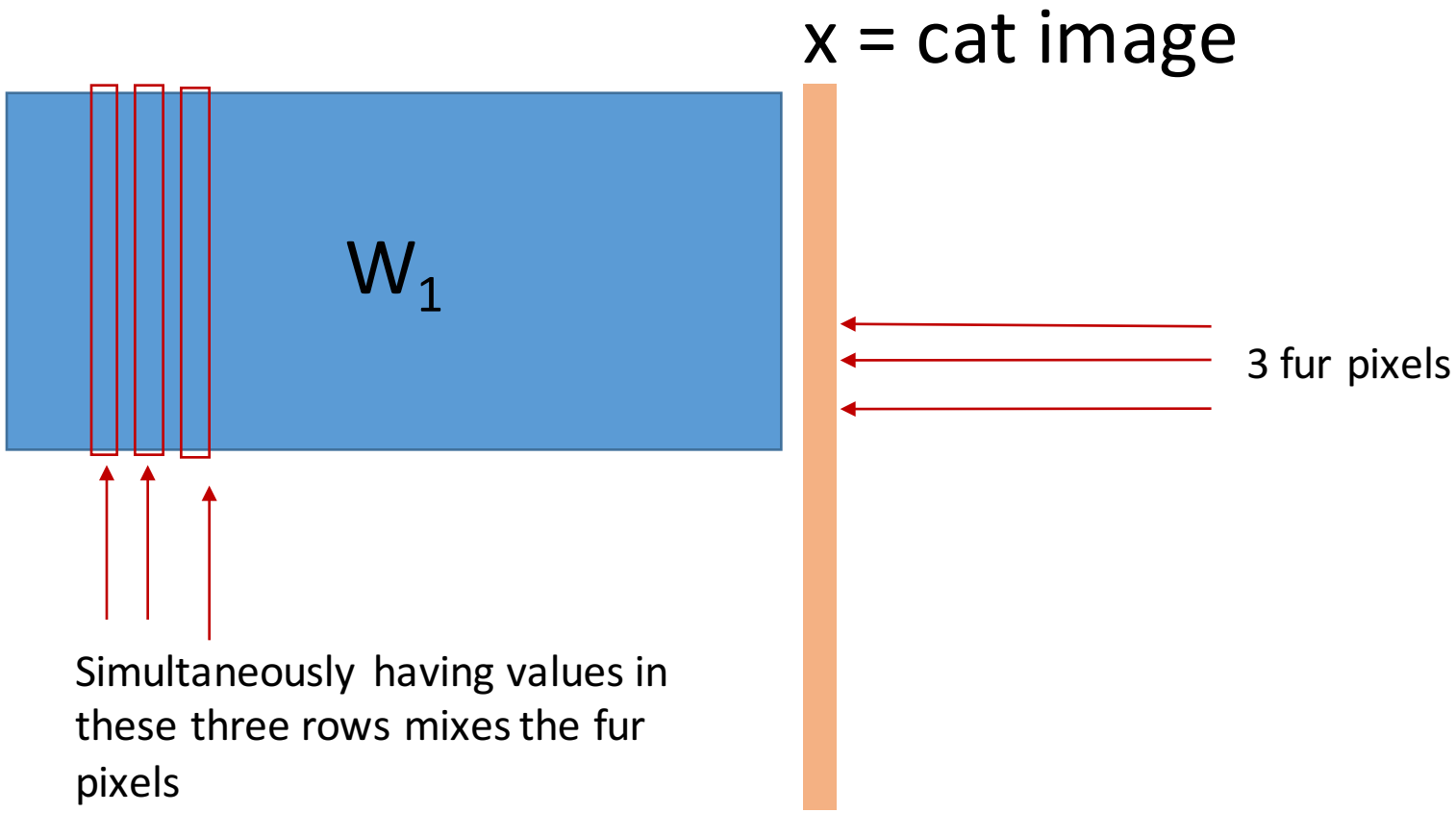


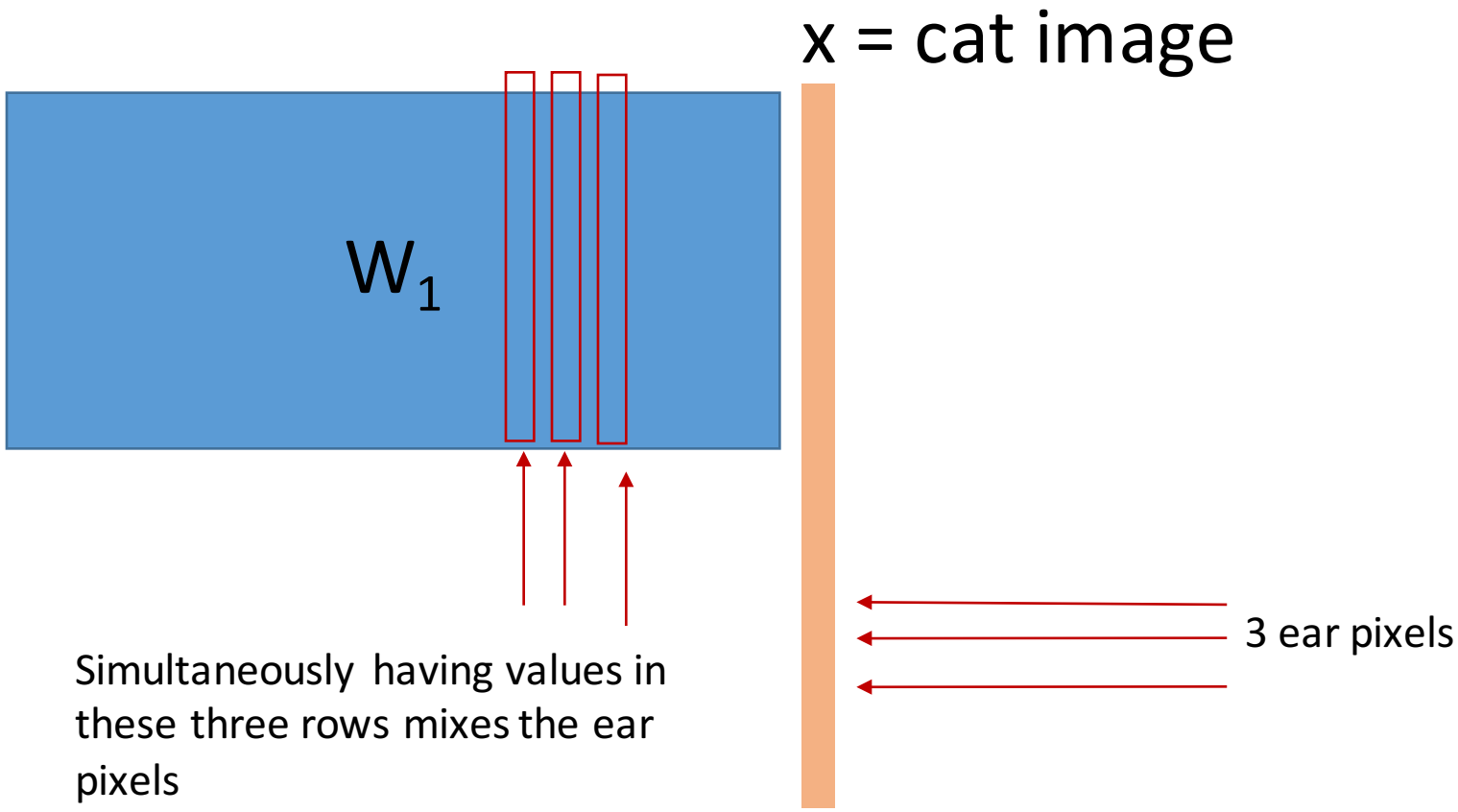
But understanding the stripes in these 3 pixels right near each other is going to be pretty helpful…

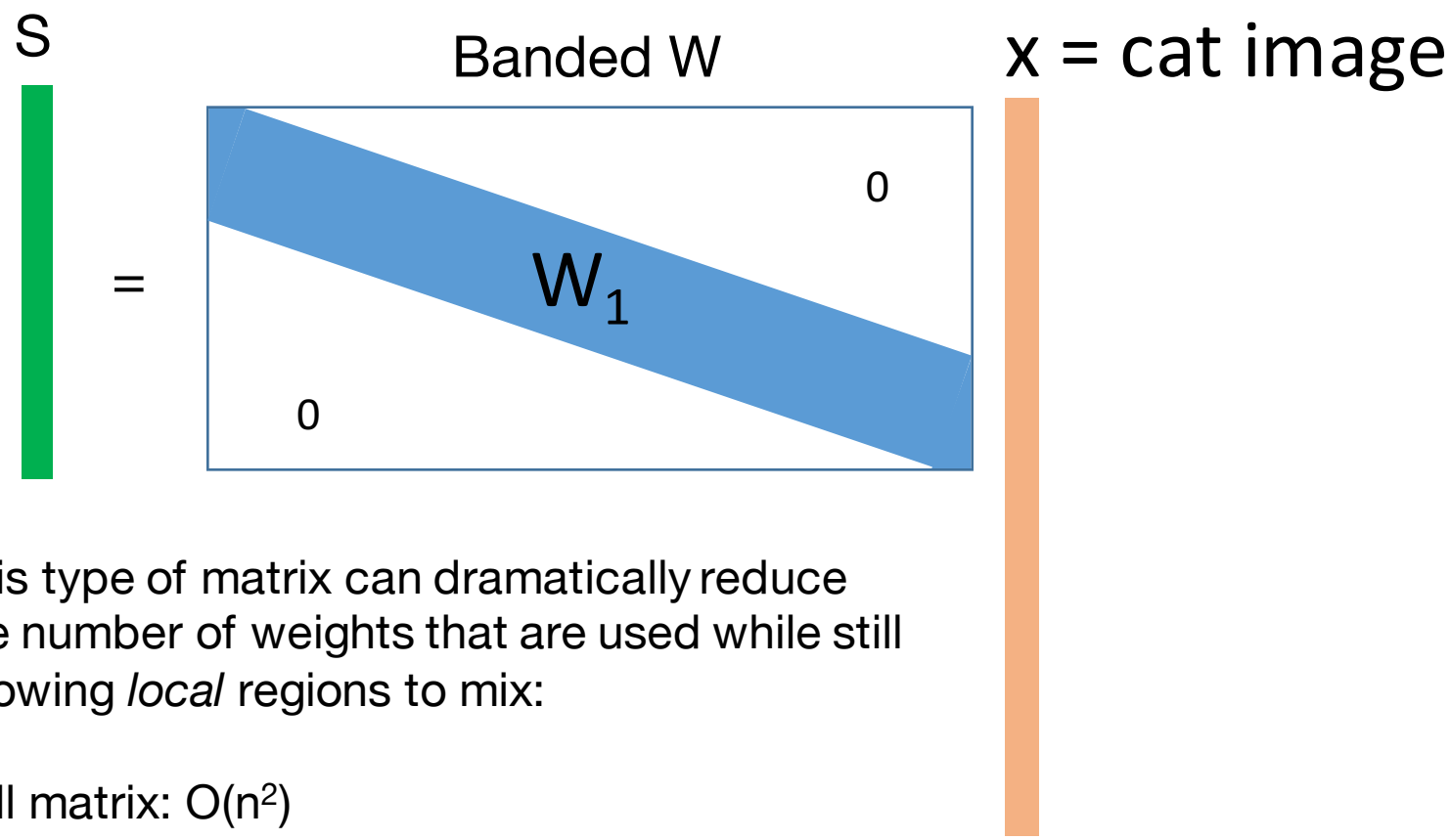
Understanding the round shape of this ear will also be helpful




x = cat image
W_1
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
W_1
3 fur pixels
Simultaneously having values in these three rows mixes the fur pixels

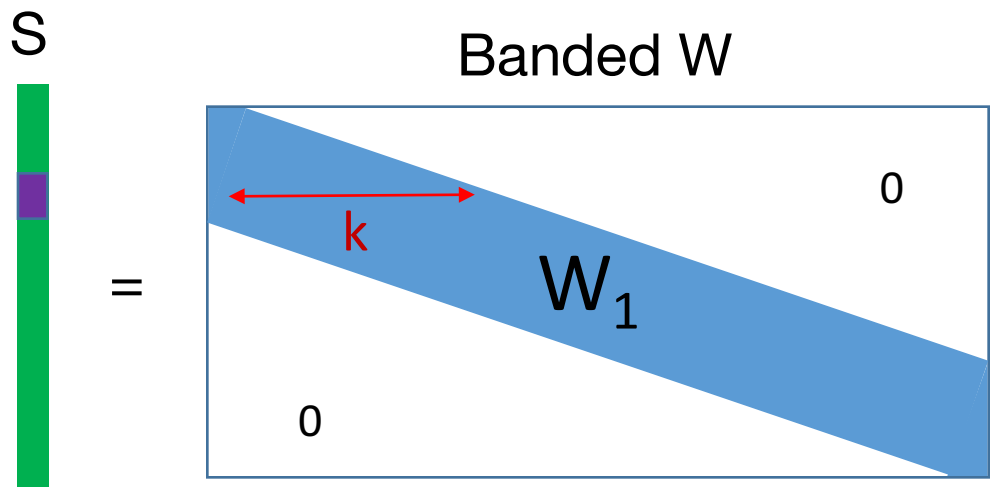

x = cat image
W_1
3 ear pixels
Simultaneously having values in these three rows mixes the ear pixels



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·O(n)



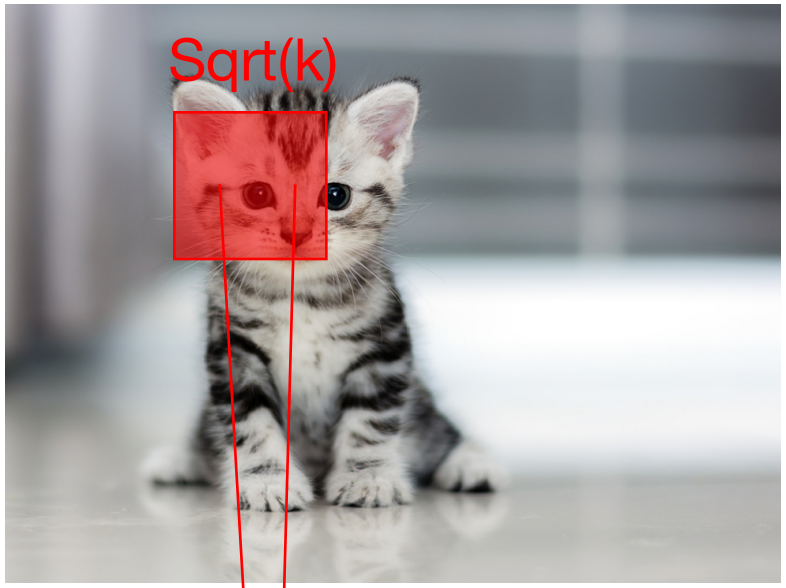
x = cat image

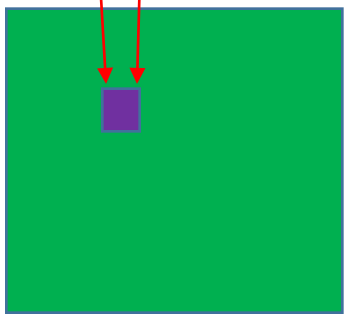
This type of matrix can dramatically reduce the number of weights that are used while still allowing *local* regions to mix:

Full matrix: $O(n^2)$

Banded matrix: k·O(n)

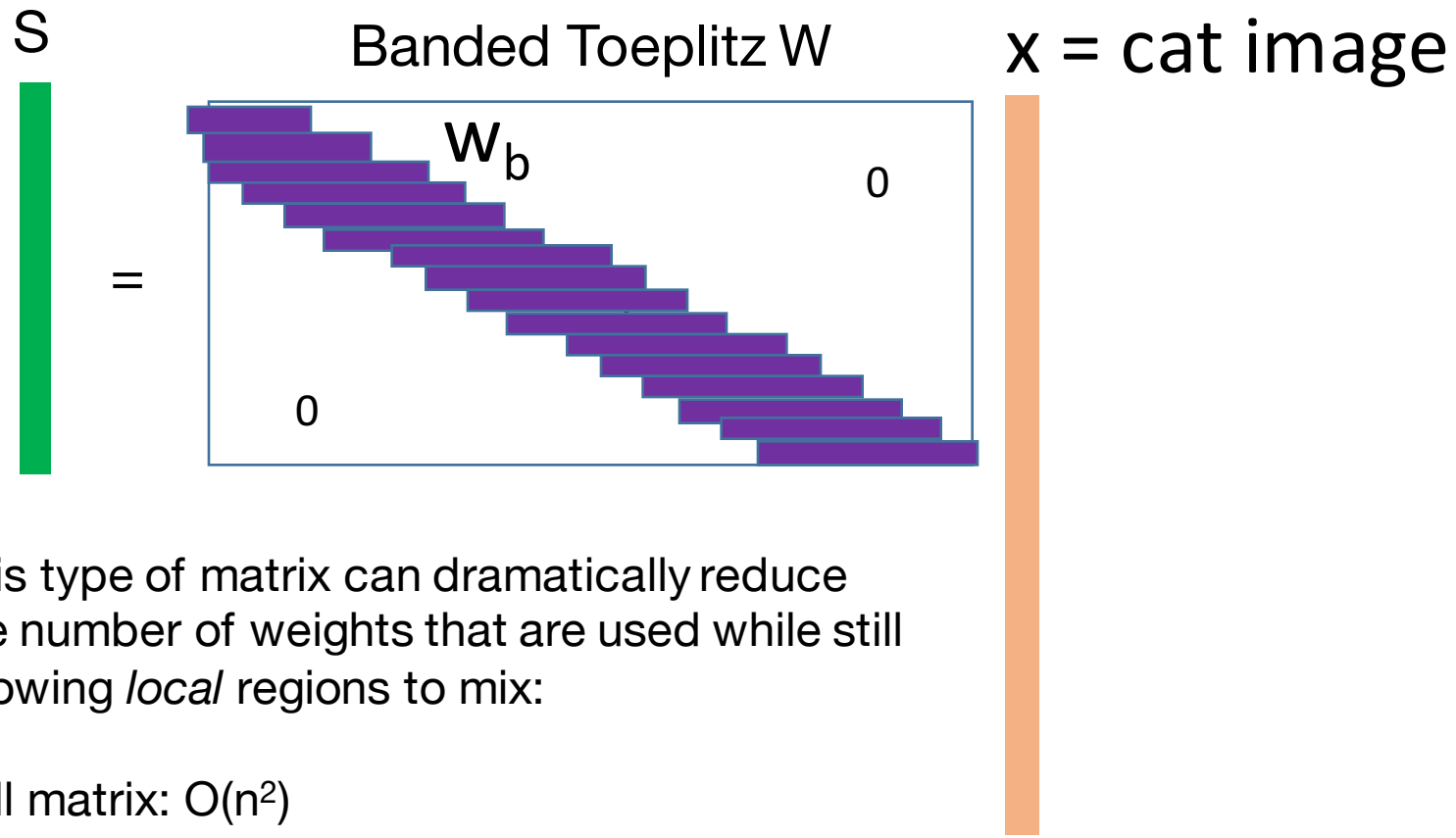
Image interpretation





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

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



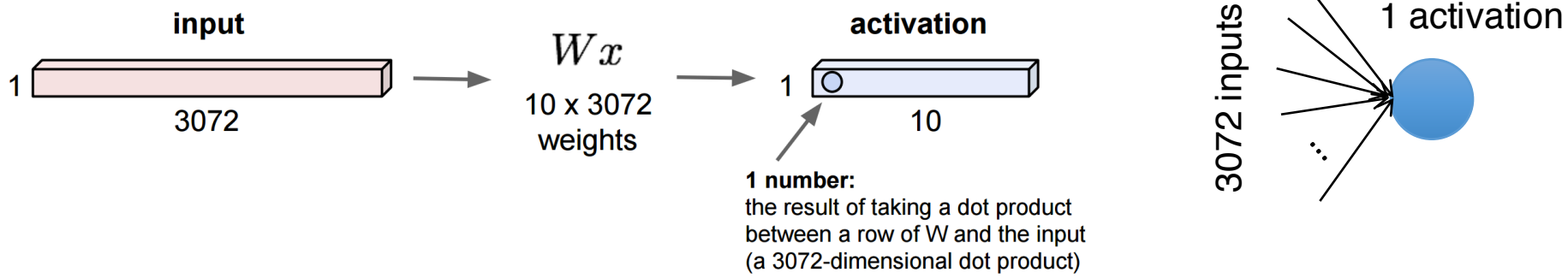
This type of matrix can dramatically reduce the number of weights that are used while still allowing *local* regions to mix:

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

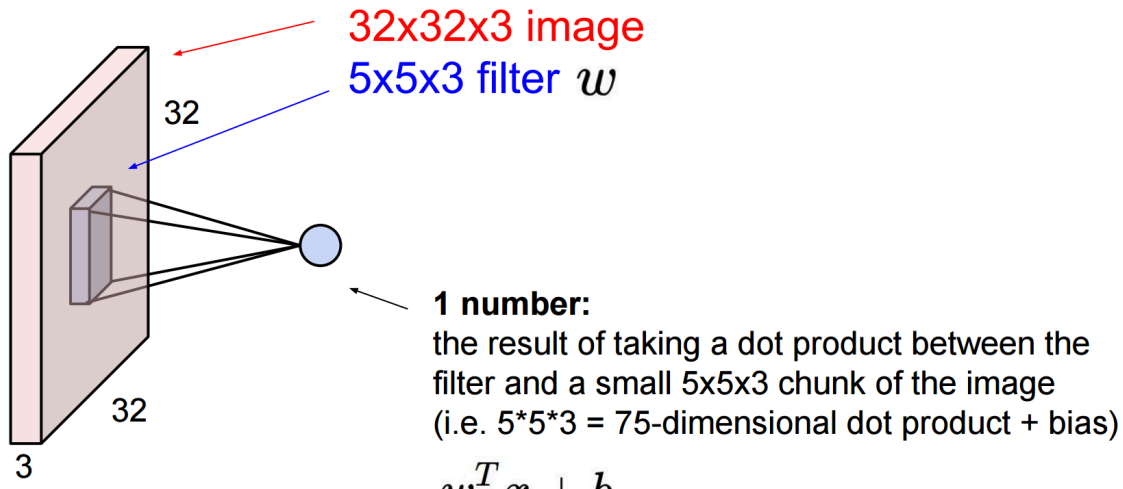
This is the definition of a convolution

Matrix multiplication is a "fully connected layer"

32x32x3 image -> stretch to 3072 x 1



Convolution takes inputs from a smaller spatial area – preserves structure



Our first convolutional neural network cost function!

$$L(w) = \frac{1}{N}\sum_{i=1}^{N} ln(1 + e^{-y_i W_2 \max(W_1 x_i, 0)}) + \lambda(||W_1||_2 + ||W_2||_2)$$

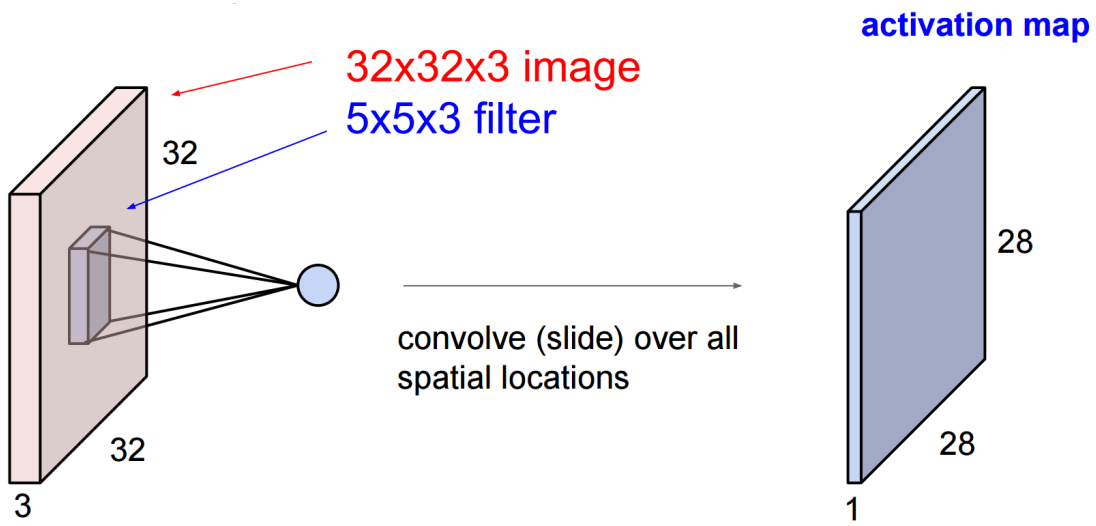
- Having “fully connected” weight matrices can produce quite a lot of weights…

  CIFAR10: 32x32 images = 1024 pixels
  W1 = Convolution matrix, K = 5^2 elements
  W2 = 1024 x 10

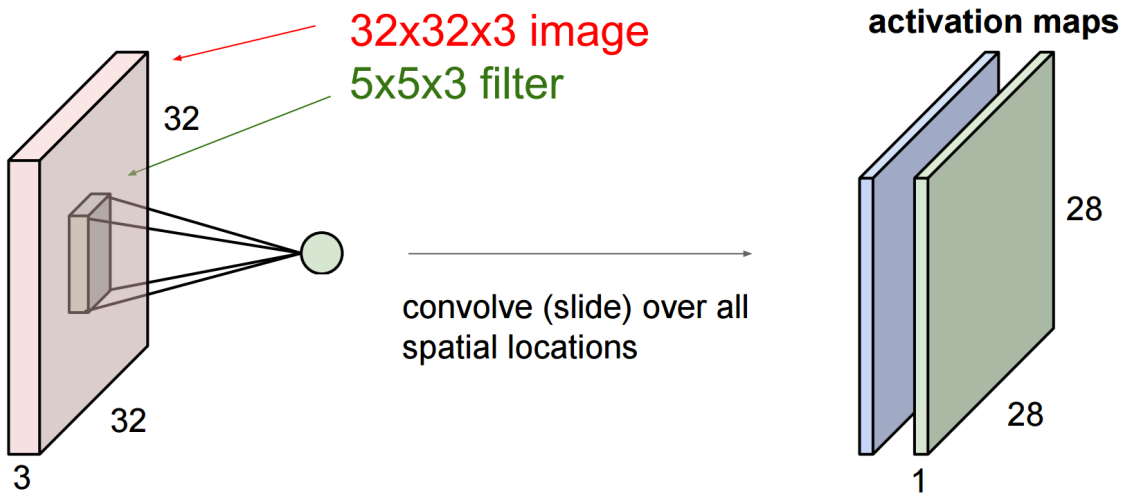
  Total number of weights: 10,240 + 25…better…

- In general, we’re going to want to have more than one convolutional filter per layer

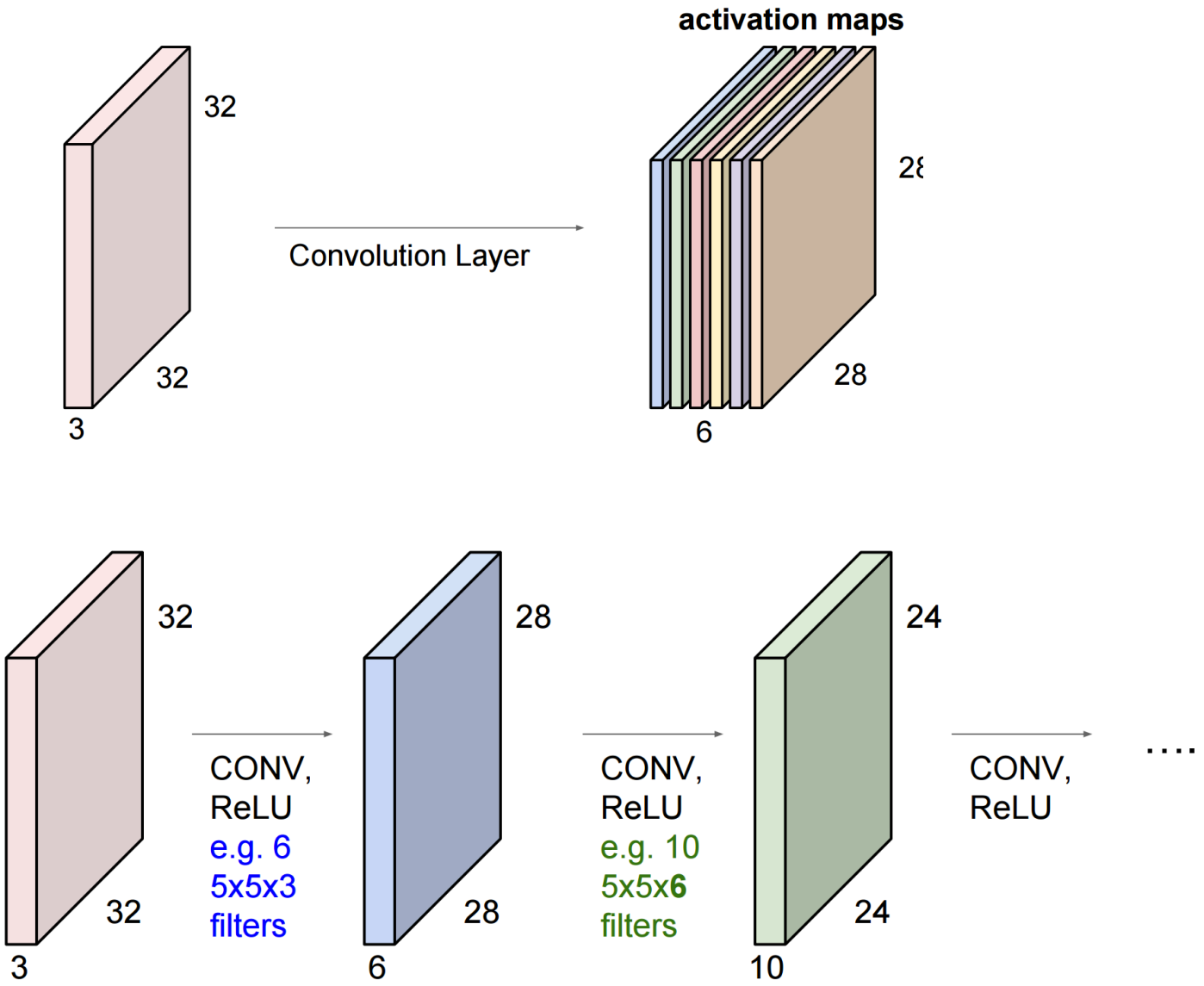
Convolution layer: learn multiple filters



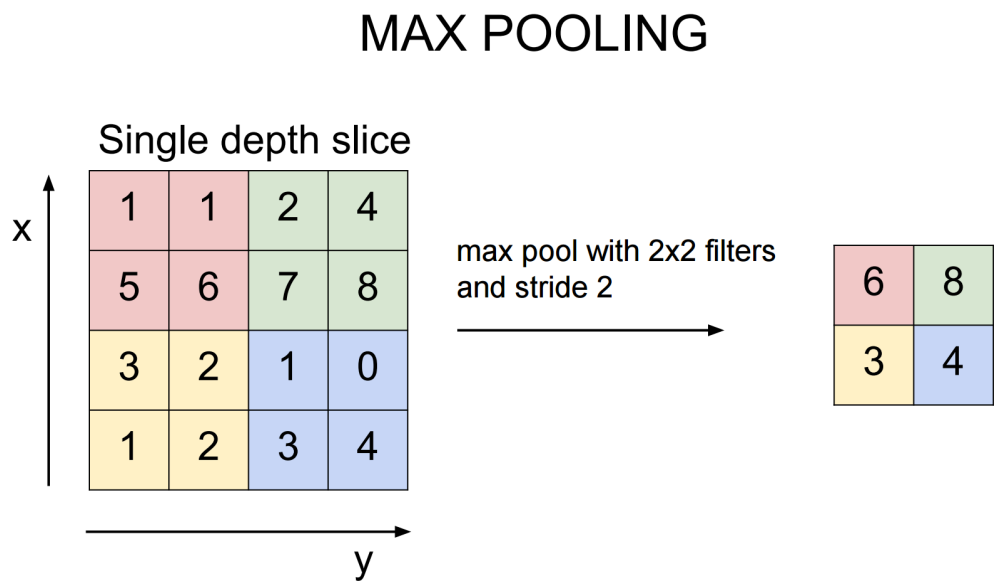
Convolution layer: learn multiple filters



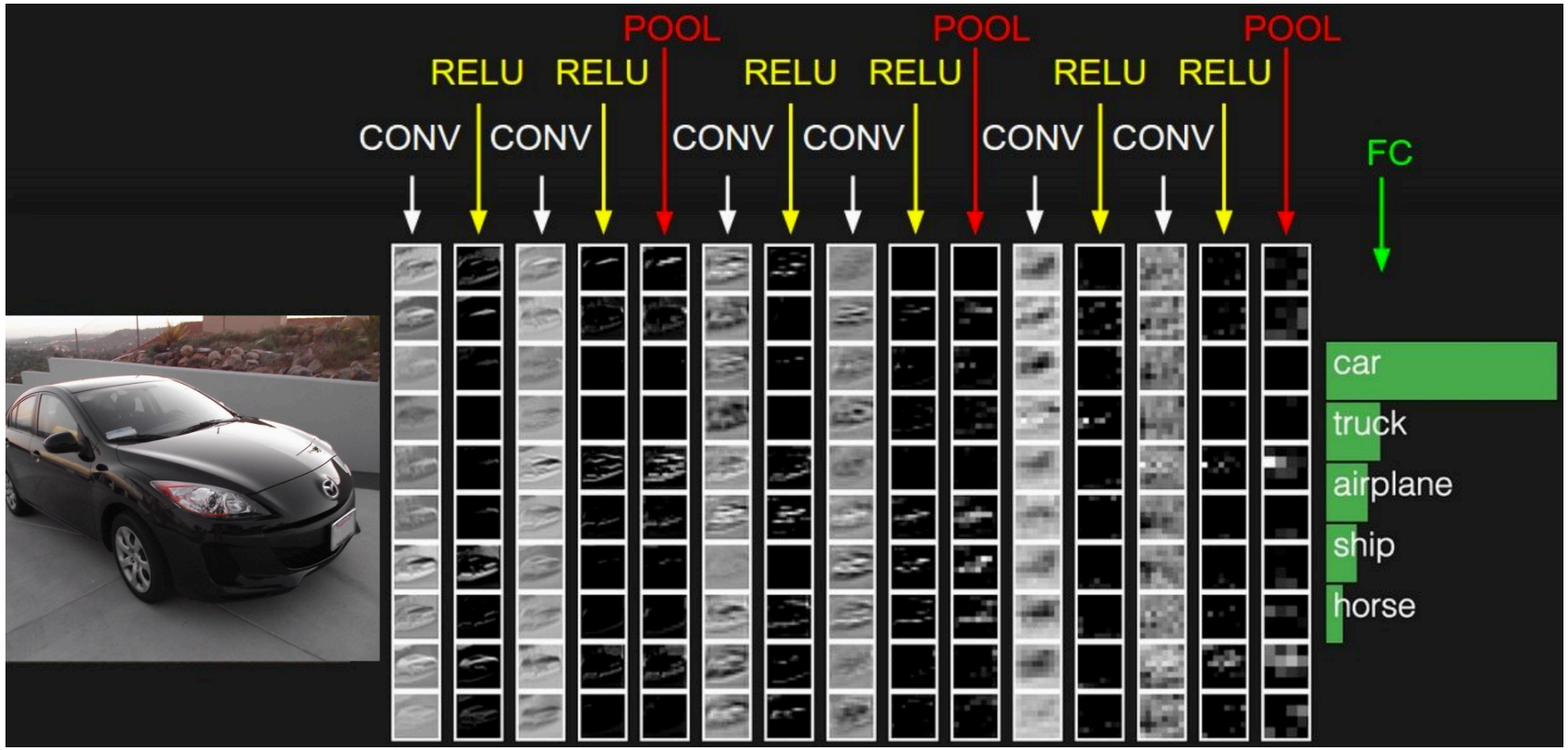
Convolution layer: learn multiple filters



Pooling operation – reduce the size of data cubes along space

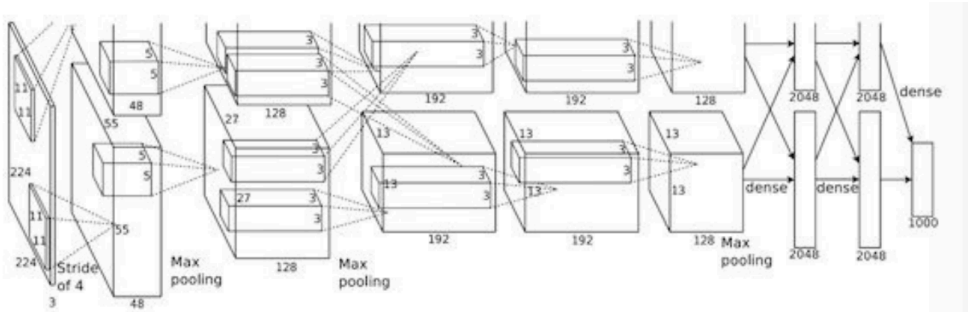


A standard CNN pipeline:



Things are getting more complex quickly…

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)