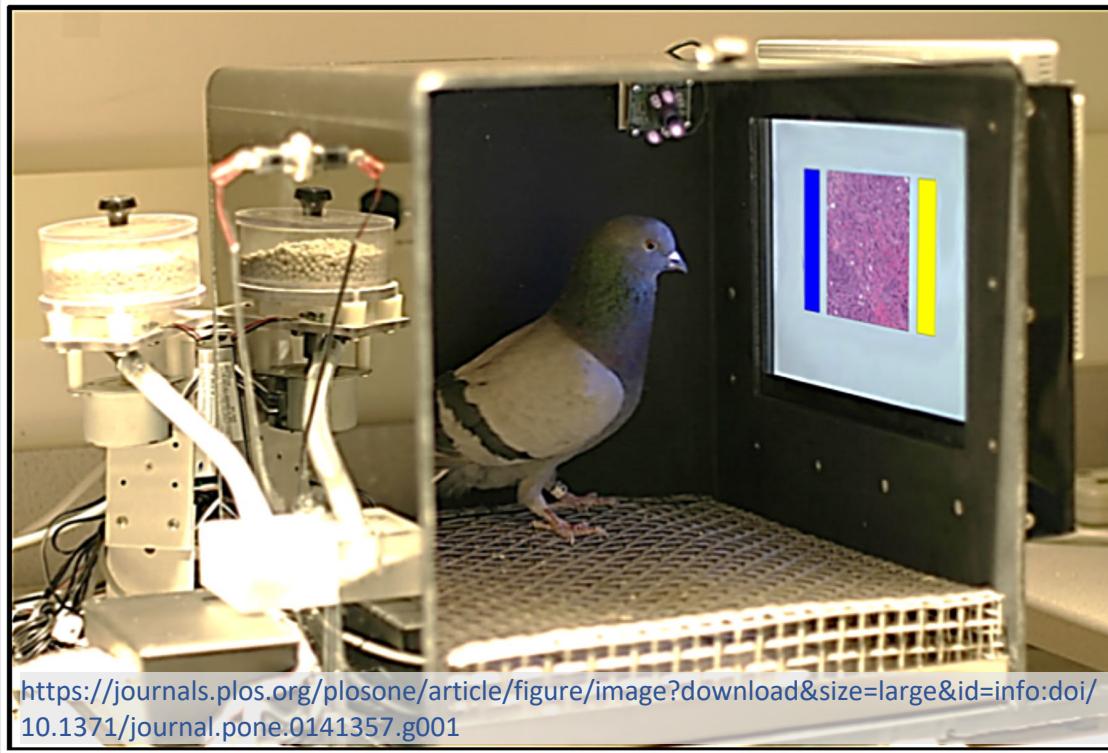


Lecture 7: Neural Networks



Haiping Lu

YouTube Playlist: <https://www.youtube.com/c/HaipingLu/>
[COM4059/6059: MLAI20@The University of Sheffield](#)

Week 7 Contents / Objectives

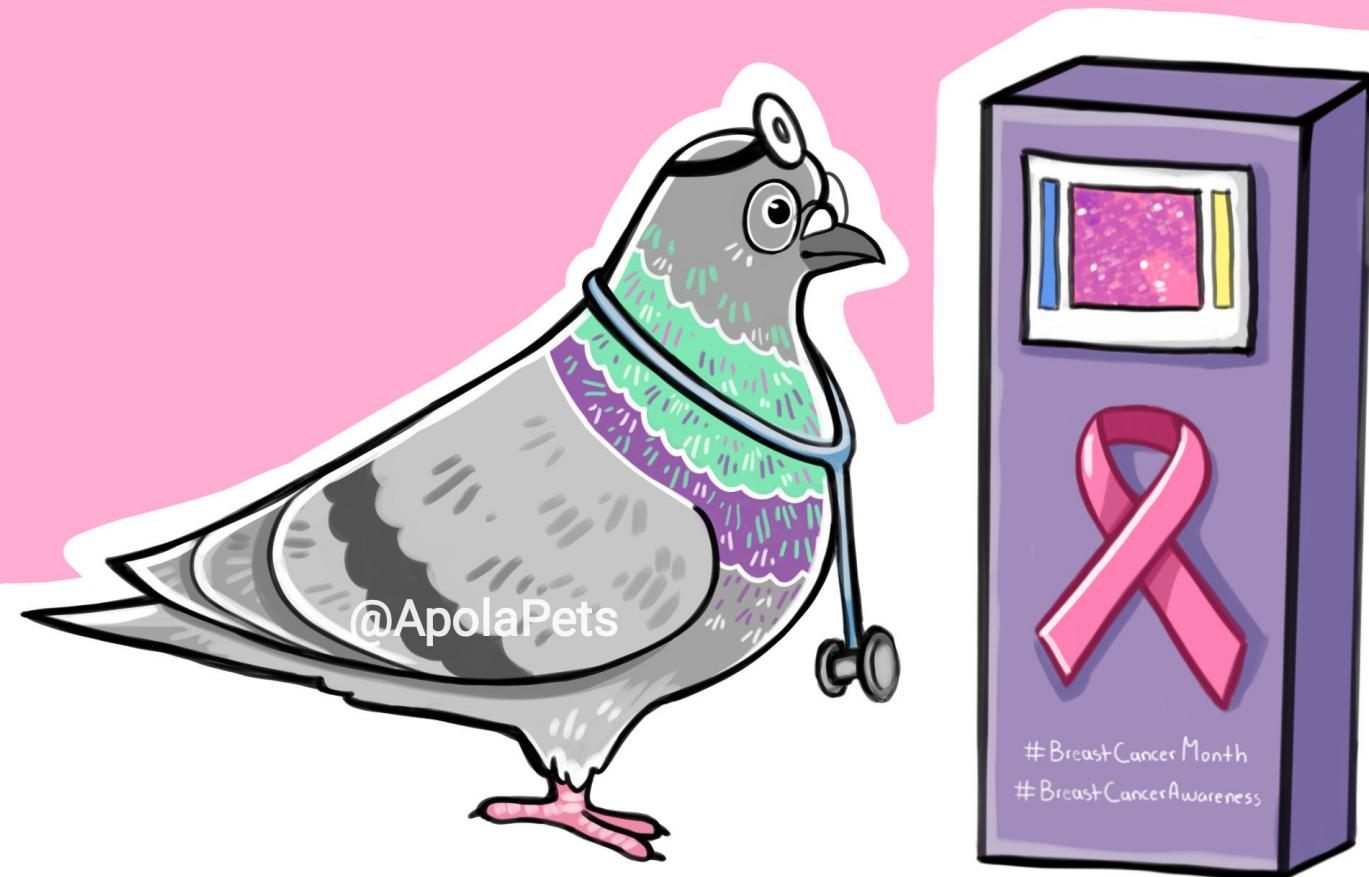
- Learning with Neurons
- Neural Networks (NNs)
- NN Decision Boundary & Features
- Convolutional NN Basics
- Convolutional NN Unboxing

Week 7 Contents / Objectives

- **Learning with Neurons**
- Neural Networks (NNs)
- NN Decision Boundary & Features
- Convolutional NN Basics
- Convolutional NN Unboxing

Did you know ?

Pigeons identify Breast Cancer



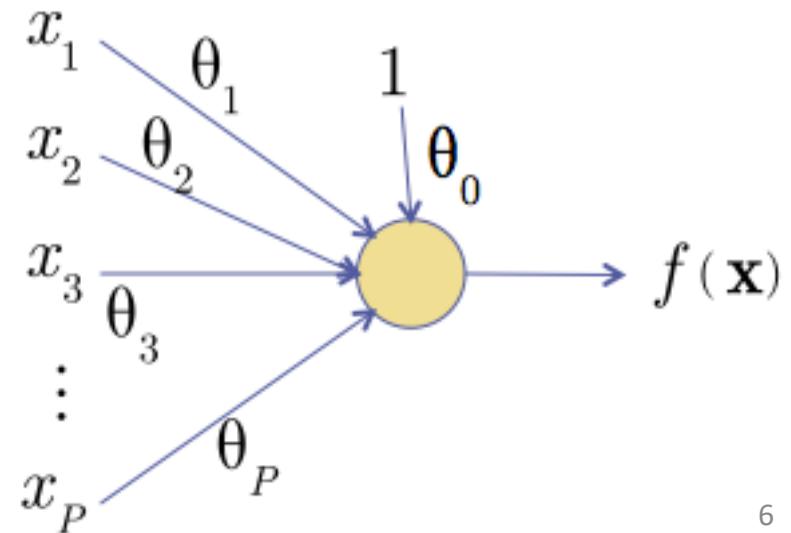
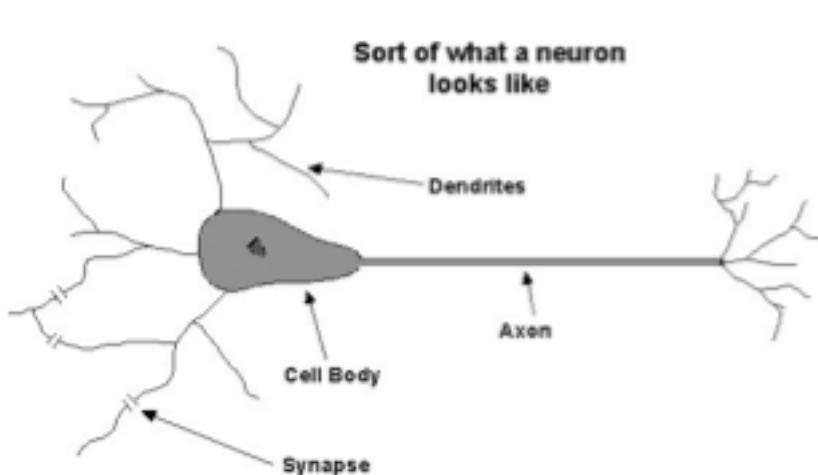
Just as well as radiologists



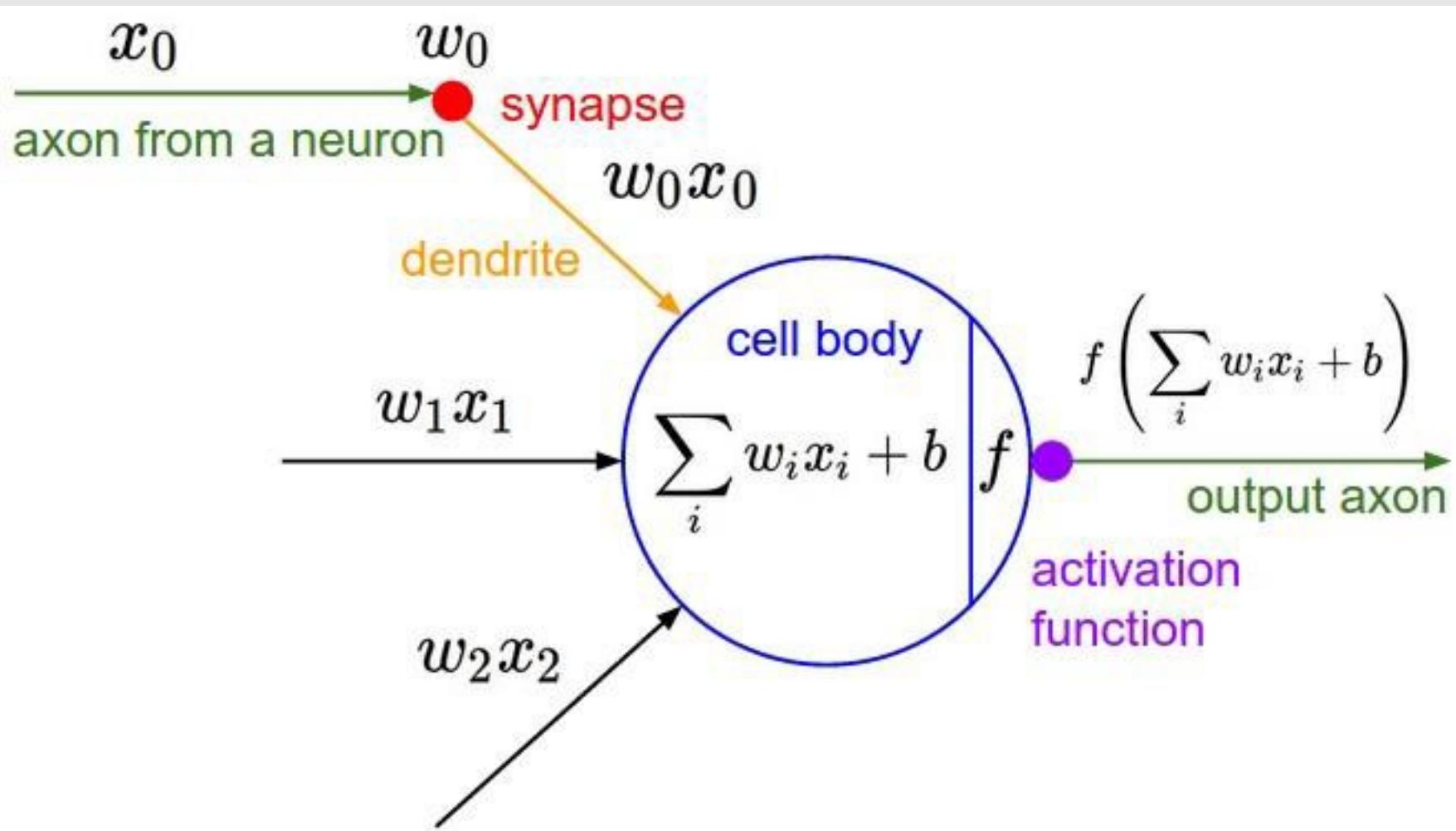
**Three correct trials with benign images follow
(the yellow button is correct).**

The Neuron Metaphor

- Neurons
 - Accept information from multiple inputs
 - Transmit information to other neurons
- Multiply inputs by weights along edges
- Apply some function to the inputs at each node

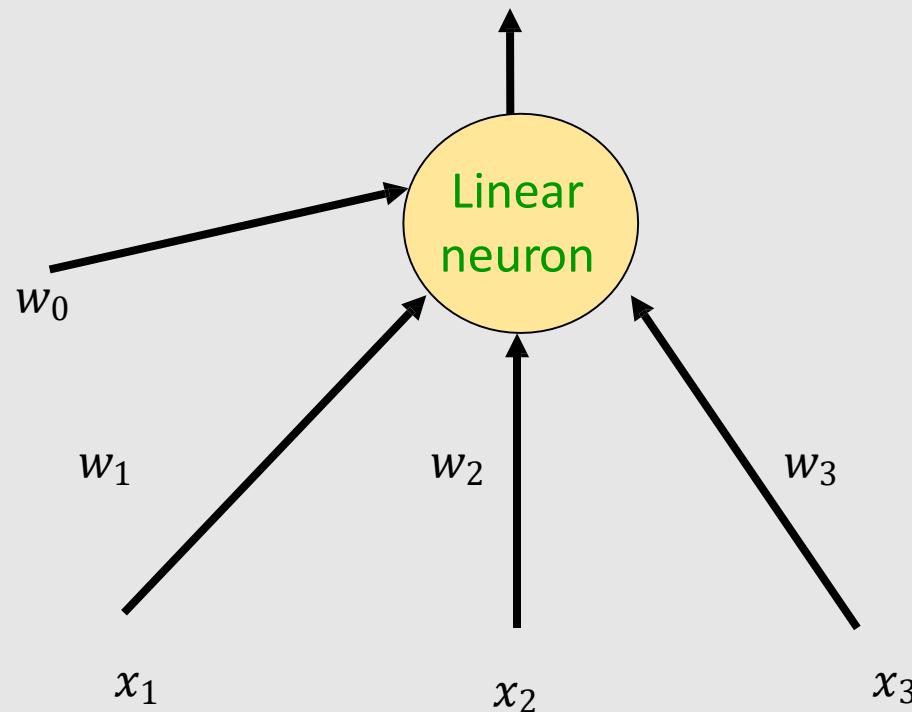


A Neuron Analogous to the Brain



Neural Network

- Network of neurons
- Linear neuron $w_0 + w_1x_1 + w_2x_2 + w_3x_3$



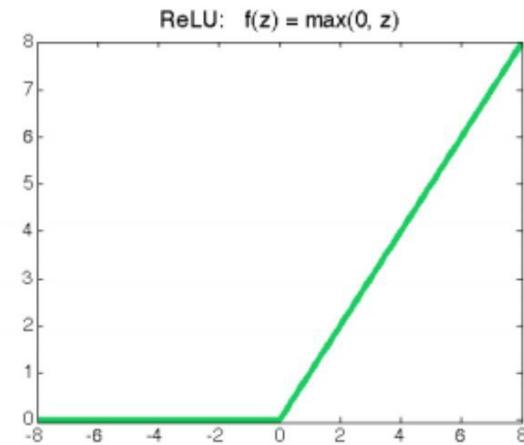
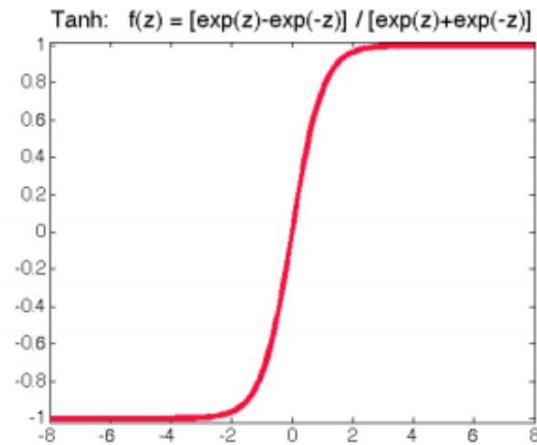
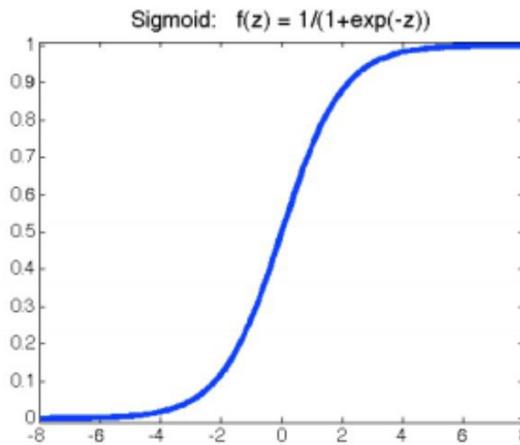
Activation Functions

- Commonly used activation functions

- Sigmoid: $\sigma(z) = \frac{1}{1+\exp(-z)}$

- Tanh: $\tanh(z) = \frac{\exp(z)-\exp(-z)}{\exp(z)+\exp(-z)}$

- ReLU (Rectified Linear Unit): $\text{ReLU}(z) = \max(0, z)$



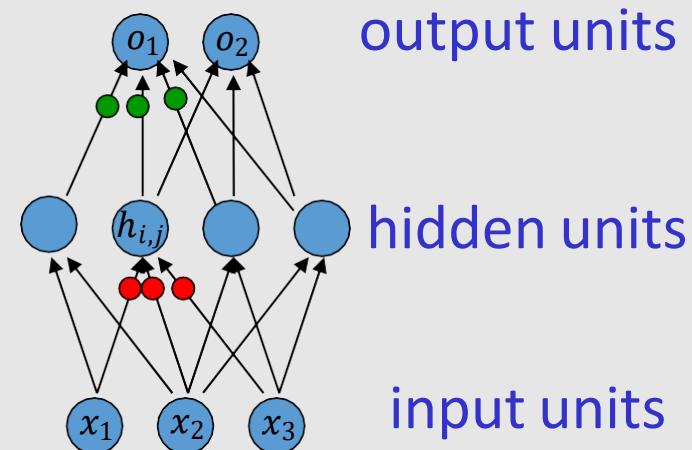
Computation in Neural Networks

- Forward pass

- Making predictions (decisions)
- Plug in the input x , get the output y

$$\mathbf{o} = g \left((W^{(2)})^T \mathbf{h} + b^{(2)} \right)$$

$$\mathbf{h} = g \left((W^{(1)})^T \mathbf{x} + b^{(1)} \right)$$



- Backward pass (backpropagation for optimisation)

- Compute the gradient of the **cost (loss/error)** function with respect to the weights to find good values for weights

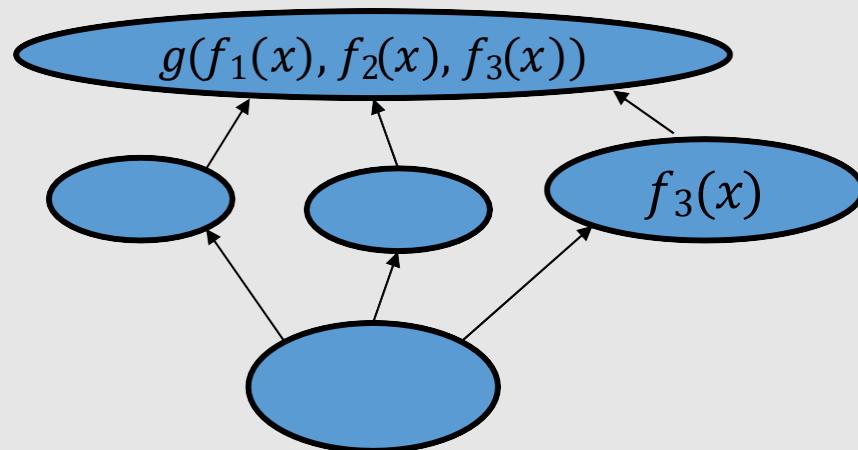
Autograd: Chain Rule

- **Univariate** chain rule

$$\frac{d}{dt} g(f(t)) = \frac{dg}{df} \cdot \frac{df}{dt}$$

- **Multivariate** chain rule

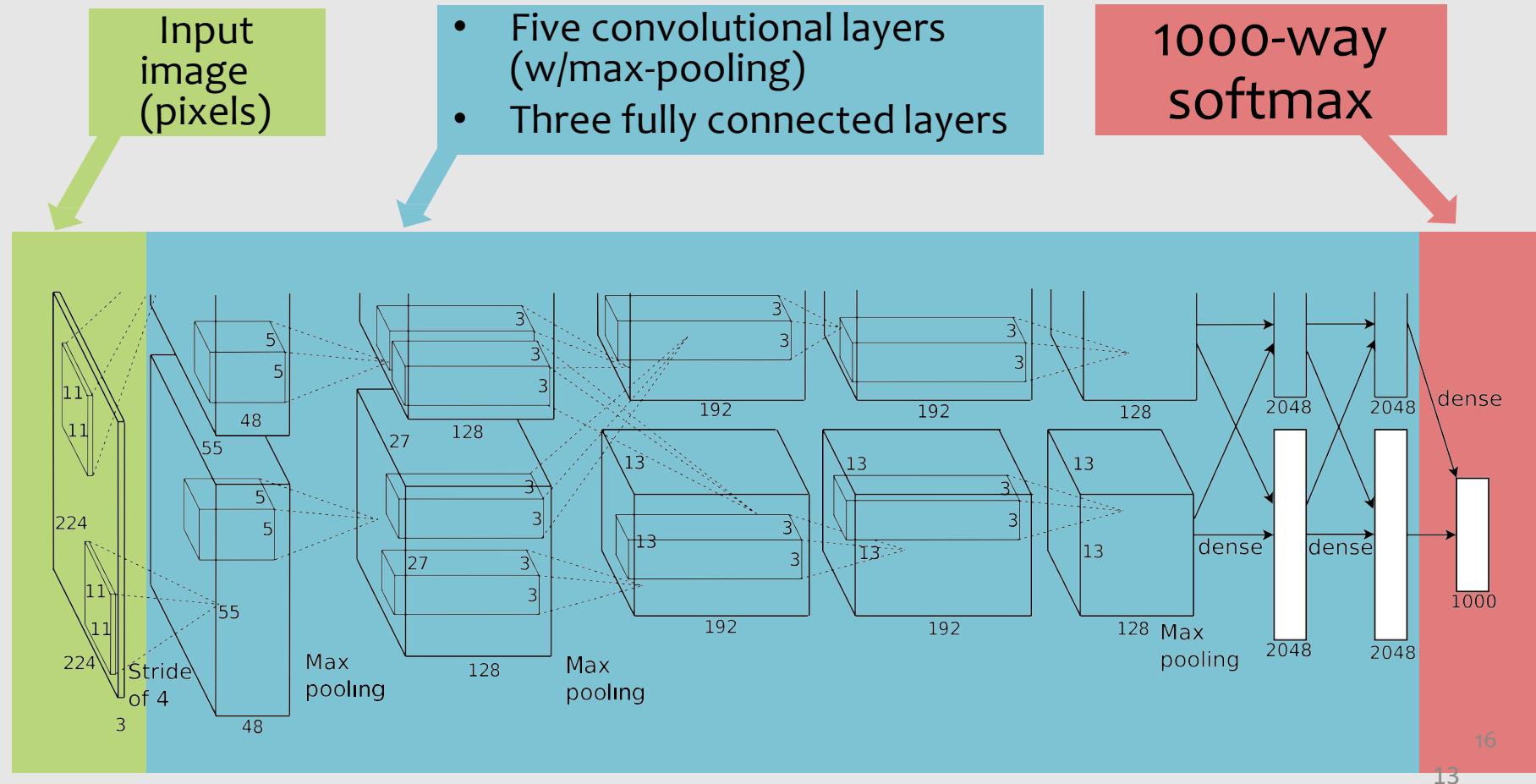
$$\frac{\partial g}{\partial x} = \sum_i \frac{\partial g}{\partial f_i} \frac{\partial f_i}{\partial x}$$



Week 7 Contents / Objectives

- Learning with Neurons
- **Neural Networks (NNs)**
- NN Decision Boundary & Features
- Convolutional NN Basics
- Convolutional NN Unboxing

AlexNet for ImageNet LSVRC-2010



Data, Model, Metric for Learning

1. Given training data:

$$\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N$$

2. Choose each of these:

– Decision function/model

$$\hat{\mathbf{y}} = f_{\theta}(\mathbf{x}_i)$$

– Loss function/metric

$$\ell(\hat{\mathbf{y}}, \mathbf{y}_i) \in \mathbb{R}$$

Face



Face



Not a face



Examples: Linear regression,
Logistic regression, Neural
Network

Examples: Mean-squared
error, Cross Entropy

Data, Model, Metric, Optimisation

1. Given training data:

$$\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N$$

2. Choose each of these:

- Decision function/model

$$\hat{\mathbf{y}} = f_{\boldsymbol{\theta}}(\mathbf{x}_i)$$

- Loss function/metric

$$\ell(\hat{\mathbf{y}}, \mathbf{y}_i) \in \mathbb{R}$$

3. Define goal:

- Objective function

$$\boldsymbol{\theta}^* = \arg \min_{\boldsymbol{\theta}} \sum_{i=1}^N \ell(f_{\boldsymbol{\theta}}(\mathbf{x}_i), \mathbf{y}_i)$$

4. Train/optimize with SGD: (take small steps opposite the gradient)

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\mathbf{x}_i), \mathbf{y}_i)$$

Data, Model, Metric, Optimisation

1. Given training data:

$$\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N$$

2. Choose each of these:

- Decision function/model

$$\hat{\mathbf{y}} = f_{\boldsymbol{\theta}}(\mathbf{x}_i)$$

- Loss function/metric

$$\ell(\hat{\mathbf{y}}, \mathbf{y}_i) \in \mathbb{R}$$

3. Define goal:

- Objective function

Compute **gradients**
via backpropagation
Using automatic
differentiation

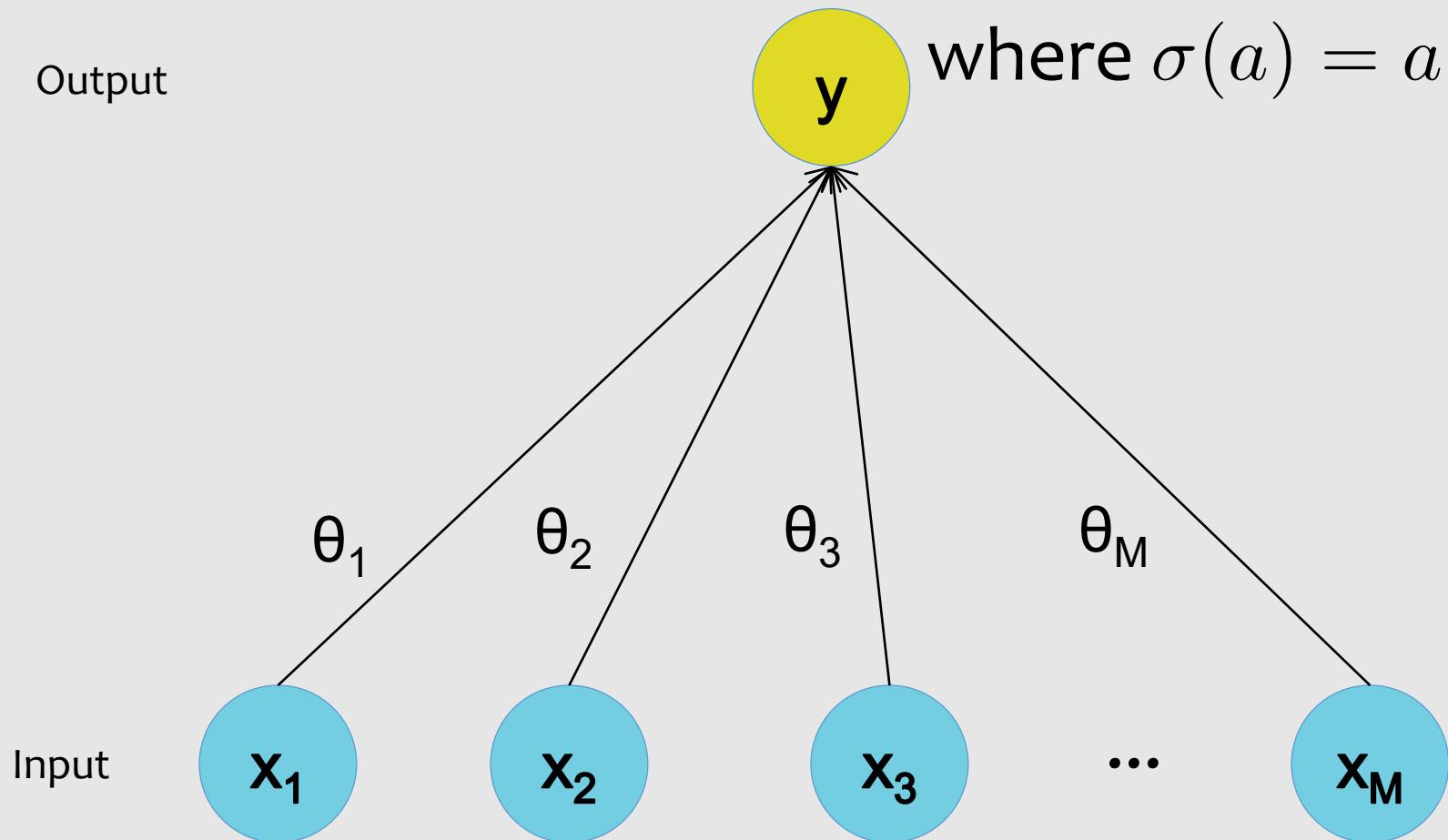
($f_{\boldsymbol{\theta}}(\mathbf{x}_i), \mathbf{y}_i$)
opposite the gradient)

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\mathbf{x}_i), \mathbf{y}_i)$$

Linear Regression Model ($x \rightarrow y$)

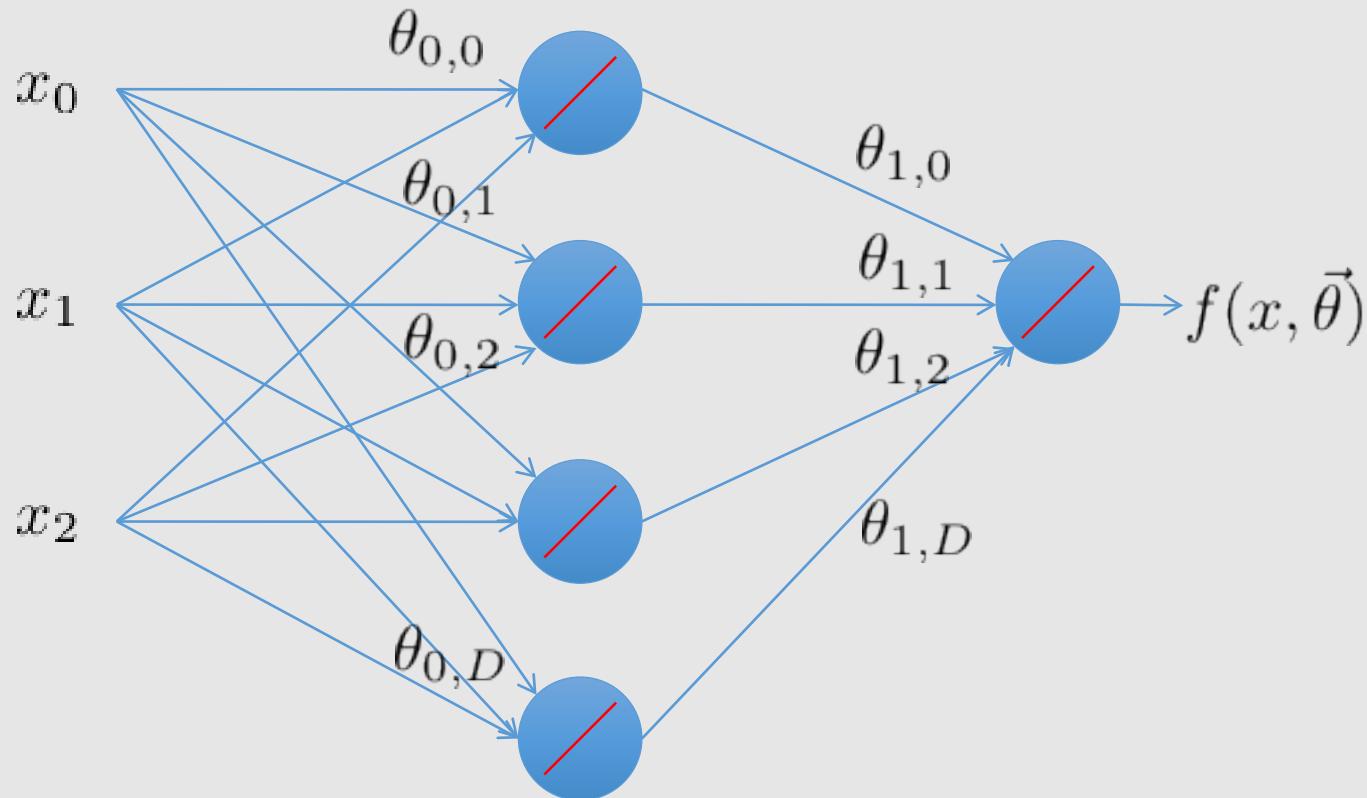
$$y = h_{\theta}(x) = \sigma(\theta^T x)$$

Output



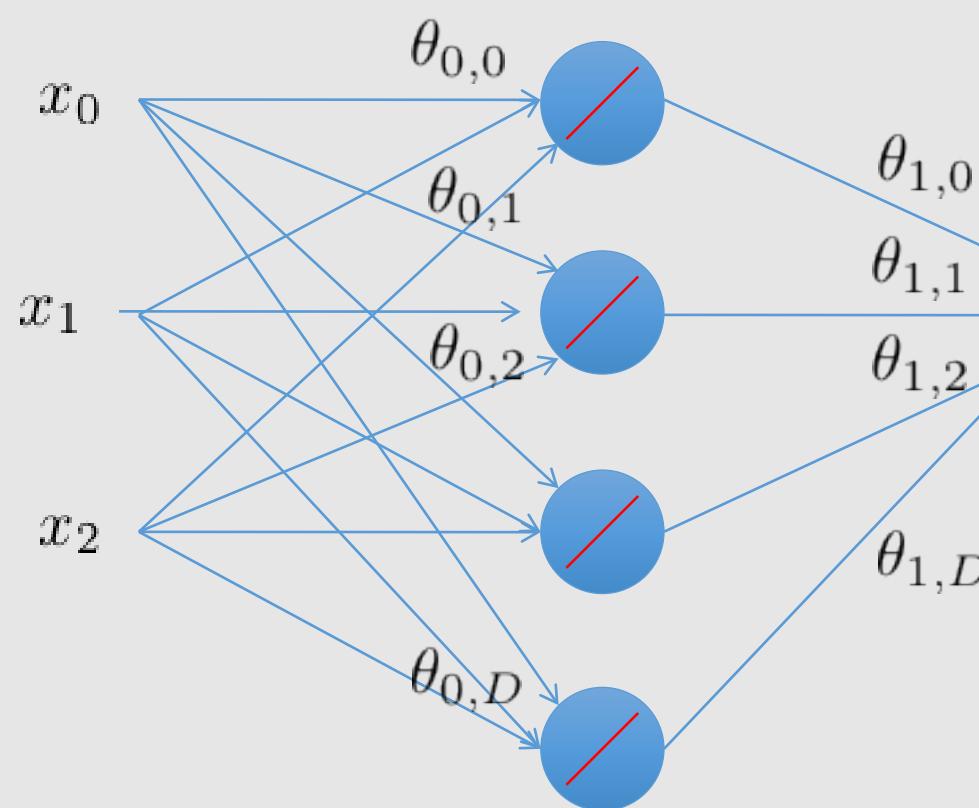
Linear Regression Neural Networks

- **Question:** What happens when we arrange **linear neurons** in a multilayer network?



Linear Regression Neural Networks

- Nothing special happens.
 - The product of two linear transformations is itself a linear transformation.



$$f(x, \vec{\theta}) = \sum_{i=0}^D \theta_{1,i} \sum_{n=0}^{N-1} \theta_{0,i,n} x_n$$

$$f(x, \vec{\theta}) = \sum_{i=0}^D \theta_{1,i} [\theta_{0,i}^T \vec{x}]$$

$$f(x, \vec{\theta}) = \sum_{i=0}^D [\hat{\theta}_i^T \vec{x}]$$

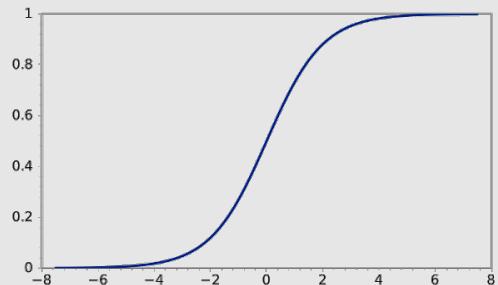
Logistic Regression Model ($x \rightarrow y$)

$$y = h_{\theta}(x) = \sigma(\theta^T x)$$

Output

y

where $\sigma(a) = \frac{1}{1 + \exp(-a)}$



Input

x₁

x₂

x₃

...

x_M

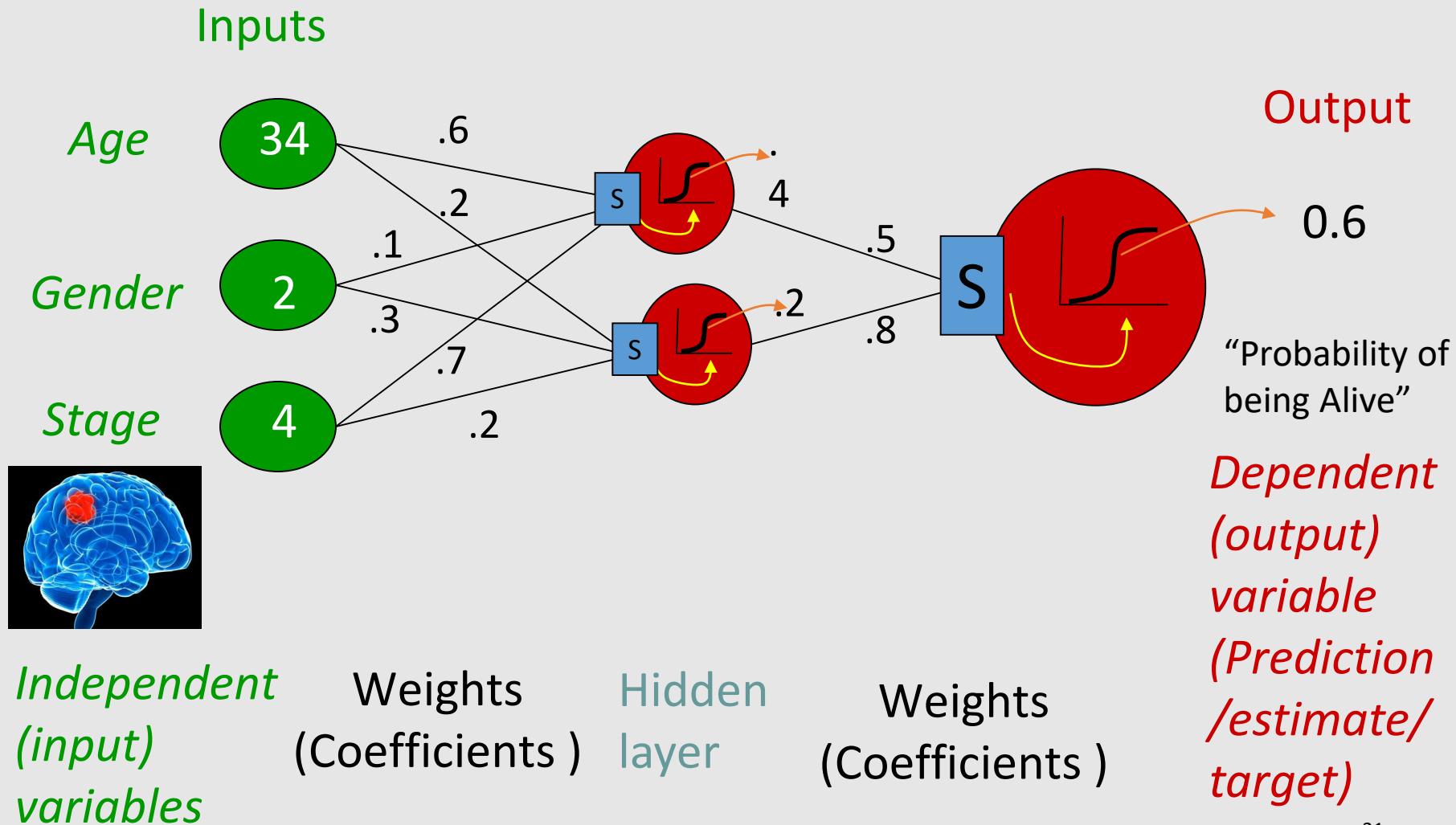
θ_1

θ_2

θ_3

θ_M

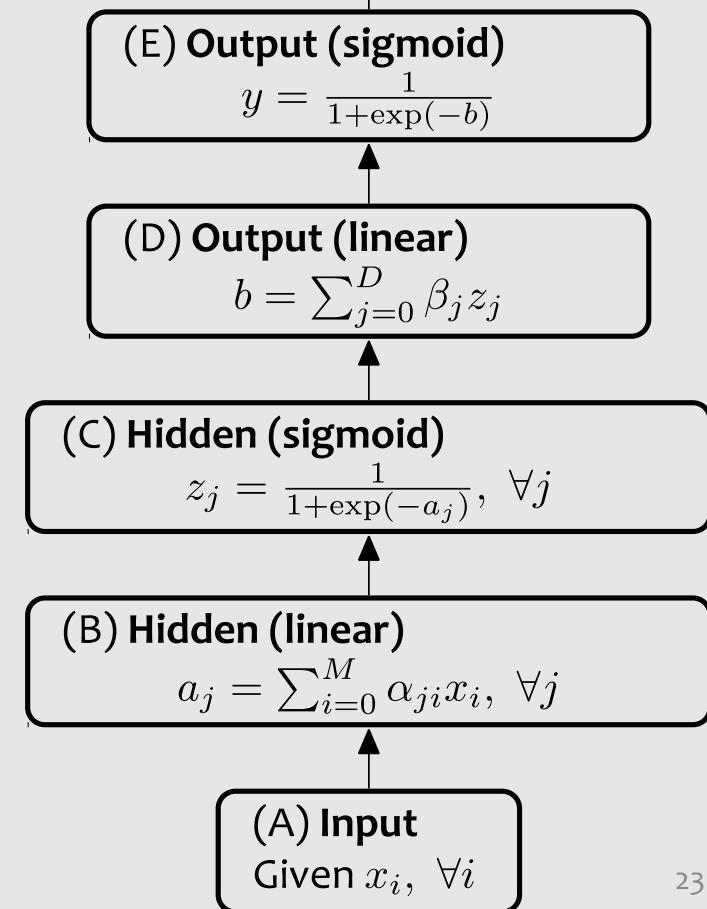
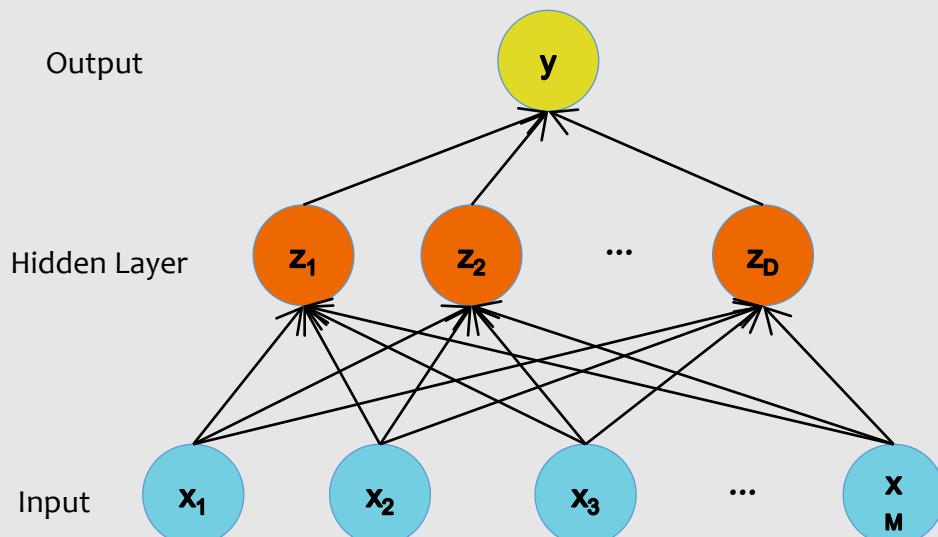
Logistic Regression Neural Networks



Week 7 Contents / Objectives

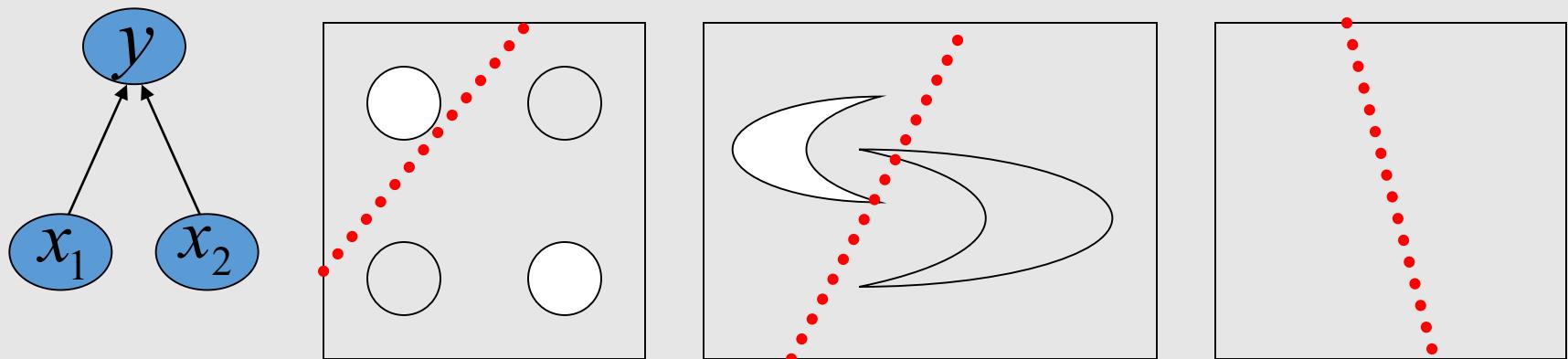
- Learning with Neurons
- Neural Networks (NNs)
- **NN Decision Boundary & Features**
- Convolutional NN Basics
- Convolutional NN Unboxing

Hidden Layers → Decisions



Decision Boundary

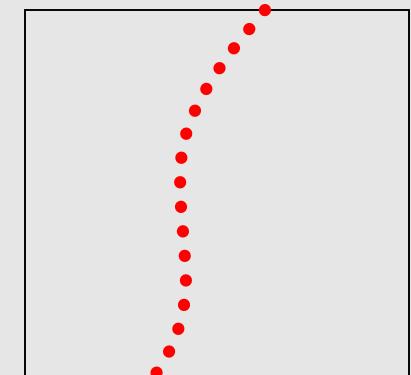
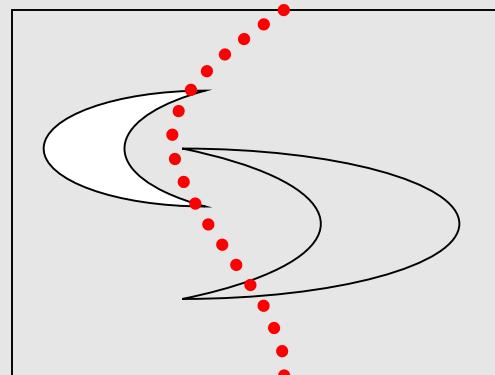
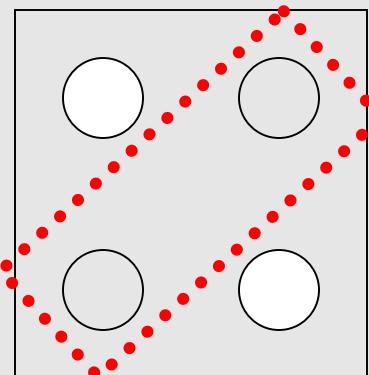
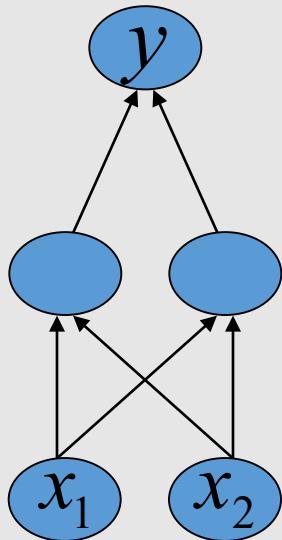
- 0 hidden layers: linear classifier
 - Hyperplanes



Example from to Eric Postma via Jason Eisner

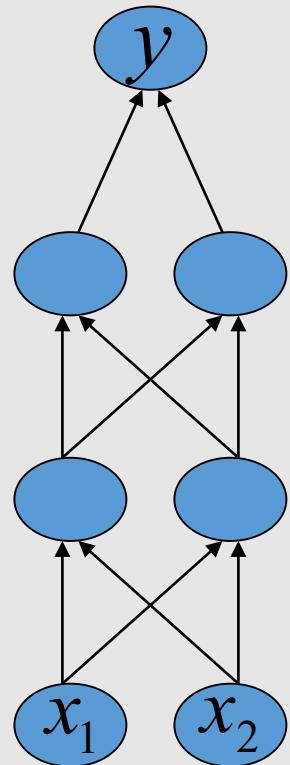
Decision Boundary

- 1 hidden layer
 - Boundary of convex region (open or closed)

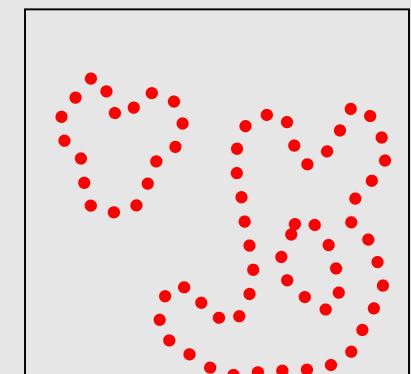
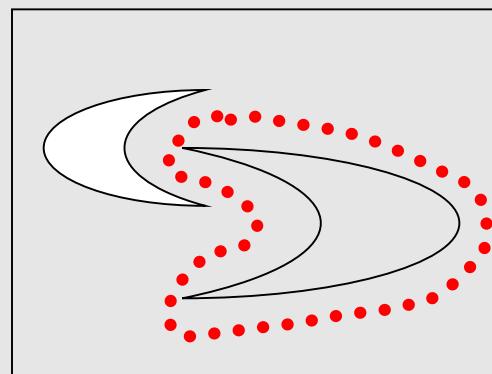
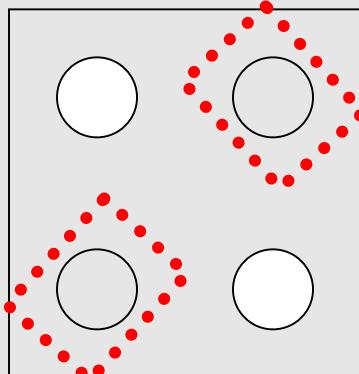


Example from to Eric Postma via Jason Eisner

Decision Boundary

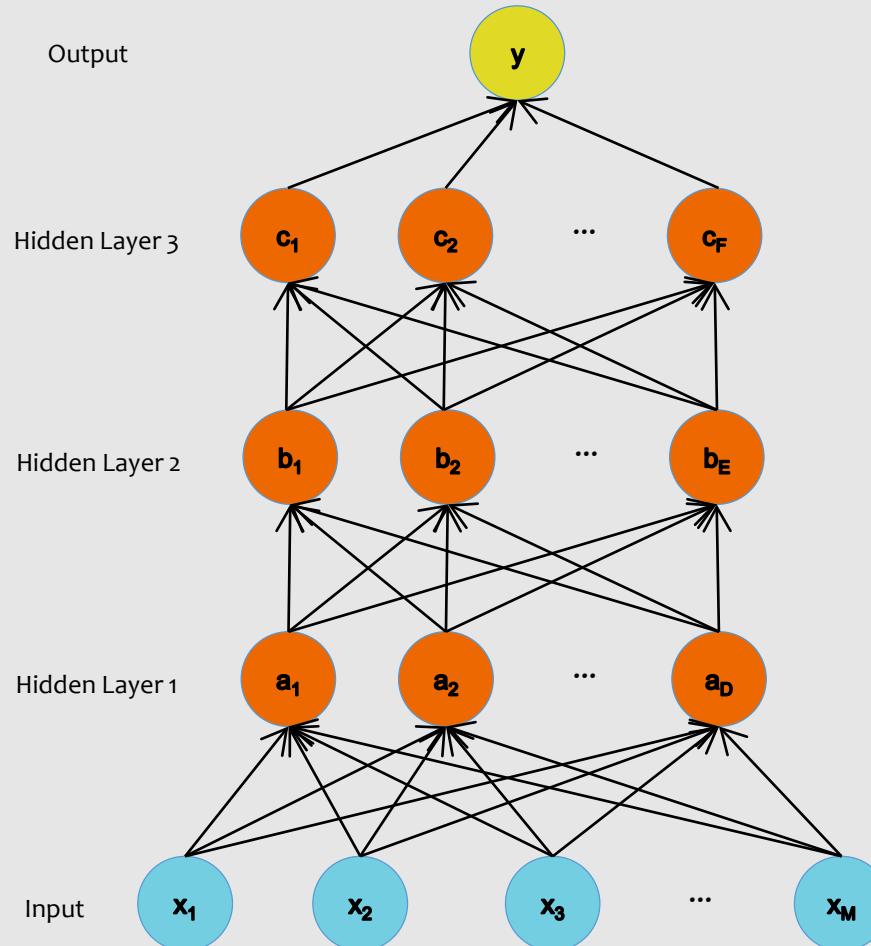


- 2 hidden layers
 - Combinations of convex regions



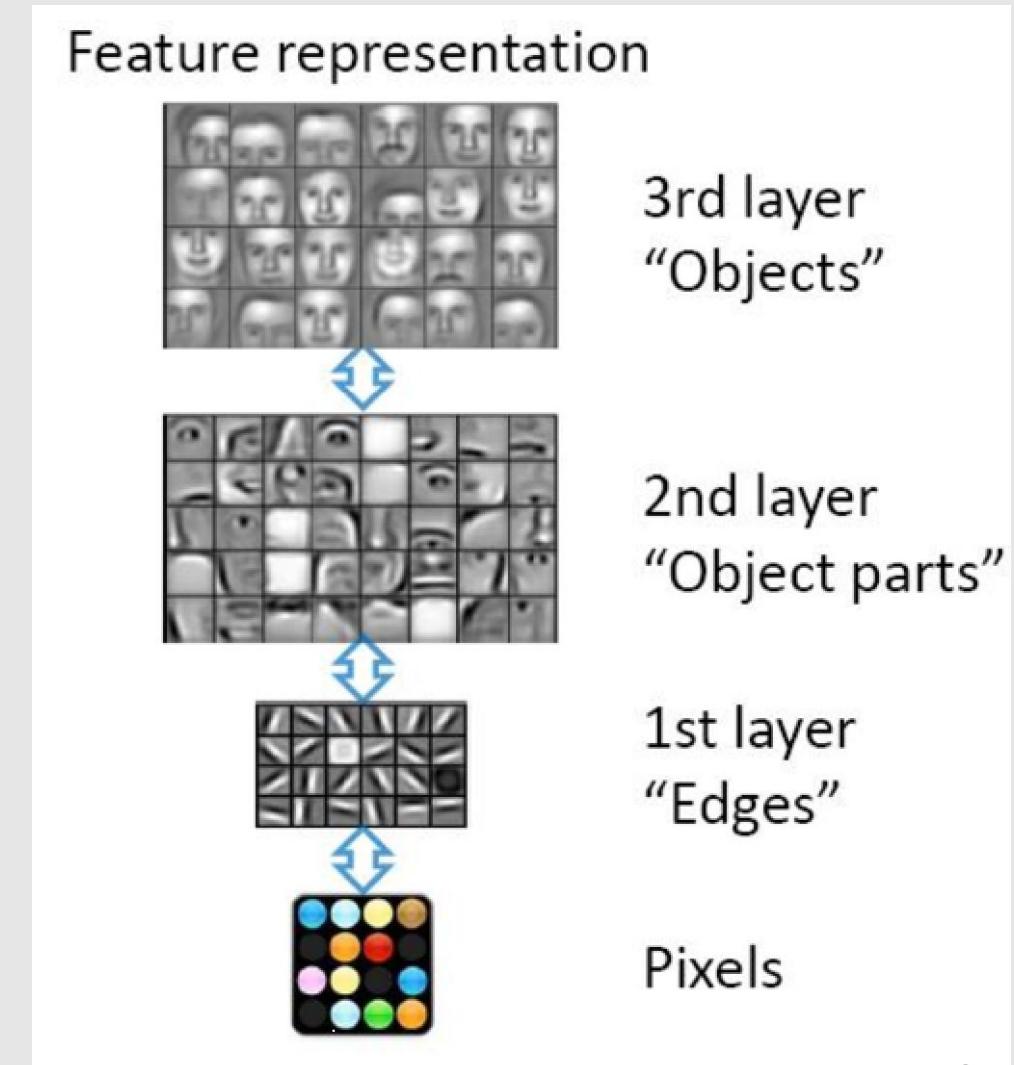
Example from to Eric Postma via Jason Eisner

Deeper Networks



Different Levels of Abstraction

- We don't know the “right” levels of abstraction
- So let the model figure it out!



Different Levels of Abstraction

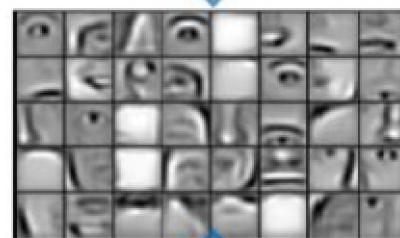
Face Recognition:

- Deep Network can build up increasingly higher levels of abstraction
- Lines, parts, regions

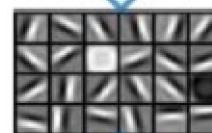
Feature representation



3rd layer
“Objects”



2nd layer
“Object parts”

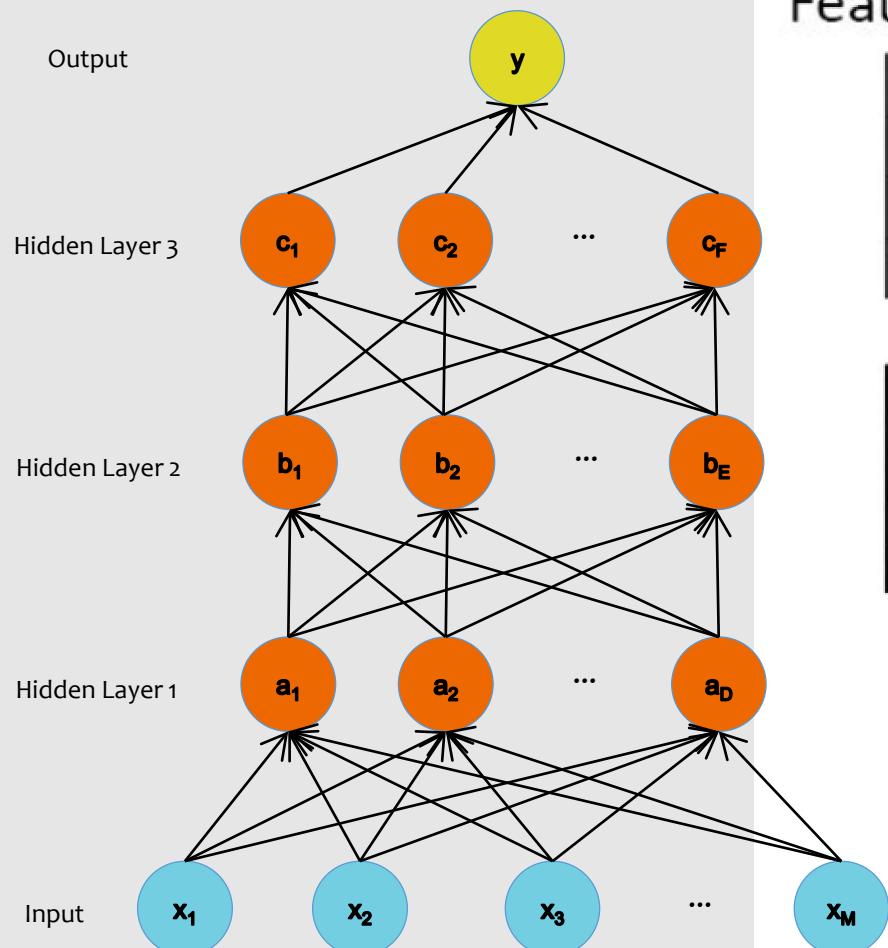


1st layer
“Edges”

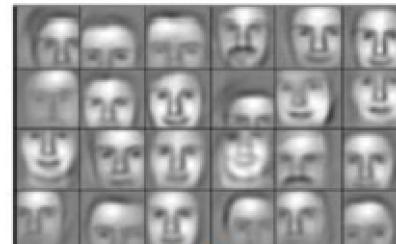


Pixels

Different Levels of Abstraction



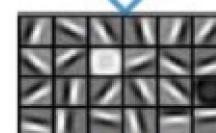
Feature representation



3rd layer
“Objects”



2nd layer
“Object parts”



1st layer
“Edges”

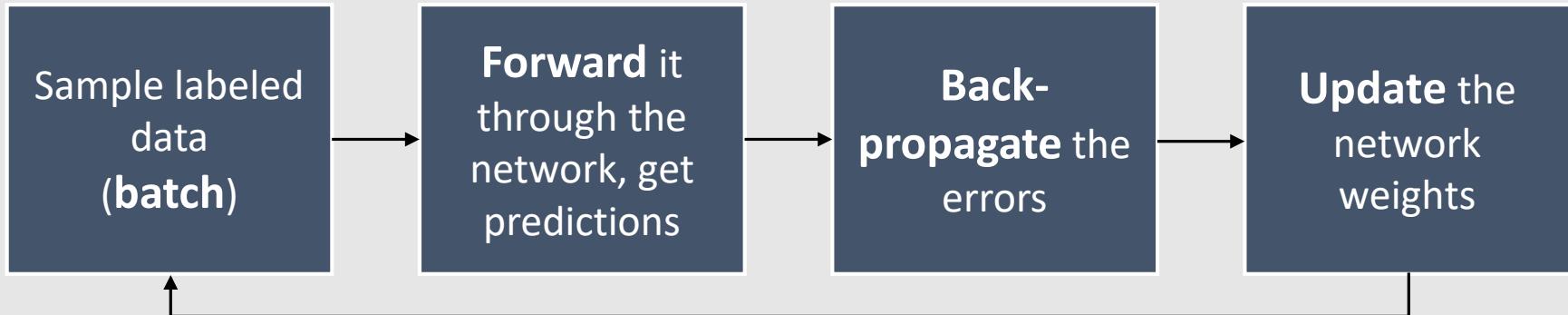


Pixels

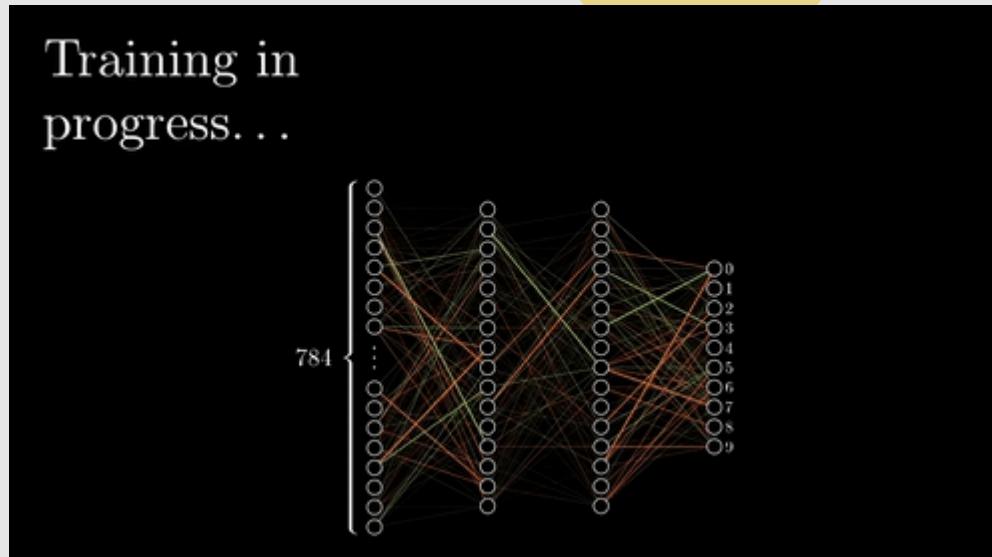
Machine Learning Ingredients

- **Data:** + pre-processing (& visualisation), e.g., $\mathcal{N}(0,1)$
- **Model**
 - Structure ~ Architecture \leftarrow expert knowledge
 - Must **specify** before ML, can optimise via cross validation (CV)
 - **Hyper-parameter**, e.g., prior, #degree, layer \leftarrow knowledge
 - Must **specify** (choices) and can optimise via CV (**tuning**)
 - Parameters (theta)
 - Compute/learn parameter, e.g., **weights**, bias \leftarrow optimisation alg.
- Evaluation **metric** (what's best): loss/error function
- **Optimisation:** (how to find the best) learnable parameters

Neural Network Training



Data → Model → Metric → Optimisation



Neural Network Ingredients

- **Data:** + pre-processing, e.g., $\mathcal{N}(0,1)$
- **Model**
 - Structure/Architecture: layered network
 - **Hyper-parameter:** layer specs, e.g. #layers, #neurons/units, activation function
 - Parameters (theta): layer weights & biases
- Evaluation metric (loss): max likelihood (min NLL), cross-entropy, etc.
- Optimisation: backpropagation (gradient-based)

Week 7 Contents / Objectives

- Learning with Neurons
- Neural Networks (NNs)
- NN Decision Boundary & Features
- **Convolutional NN Basics**
- Convolutional NN Unboxing

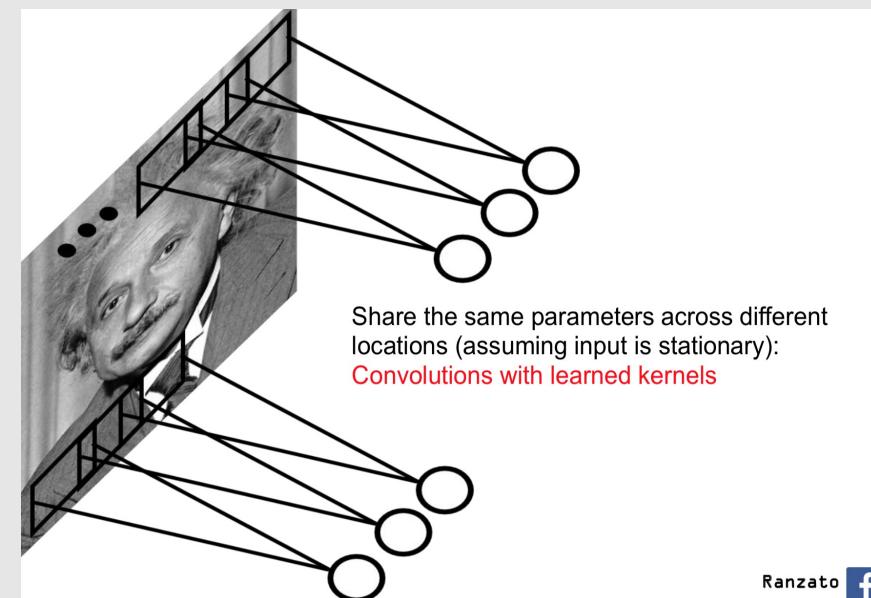
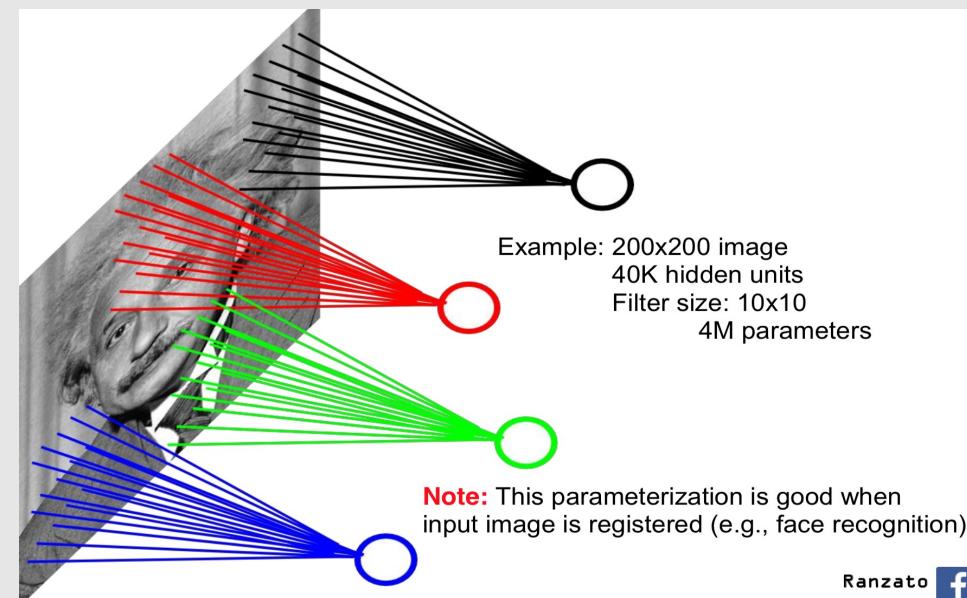
Fully Connected (FC) Layer

- Linear layers such as linear/logistic regression
- What if our network is bigger?
 - Input image: 200×200 pixels, first hidden layer: 500 units
 - **Question:** How many weights for input \rightarrow 1st hidden?
20 million
 - **Q:** Why is using an FC layer problematic for images?
 - Computing predictions (forward pass) will take a long time
 - A large number of weights requires a lot of training data to avoid overfitting
 - Small shift in image can result in large change in prediction
 - **Not** making use of the image geometry

Convolutional Neural Network

- Key ideas:

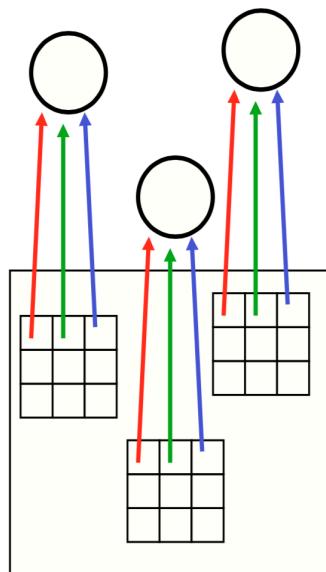
- Locally-connected layers: look for local features in small regions of the image
- Weight-sharing: detect the same local features across the entire image



Weight Sharing

- Each neuron on the higher layer detects the same feature, but in different locations on the lower layer

The red connections all have the same weight.



“Detecting” = the output (activation) is high if the feature is present

“Feature” = something in the image, like an edge, blob or shape

Forward Pass Example (Single Channel)

1 <small>$\times 1$</small>	1 <small>$\times 0$</small>	1 <small>$\times 1$</small>	0	0
0 <small>$\times 0$</small>	1 <small>$\times 1$</small>	1 <small>$\times 0$</small>	1	0
0 <small>$\times 1$</small>	0 <small>$\times 0$</small>	1 <small>$\times 1$</small>	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved Feature

<https://developer.nvidia.com/sites/default/files/pictures/2018/convolution-2.gif>

- The **kernel/filter** (yellow) contains the trainable weights. In the above, the kernel size is 3×3 .
- The “*convolved features*” is another term for “convolution output”

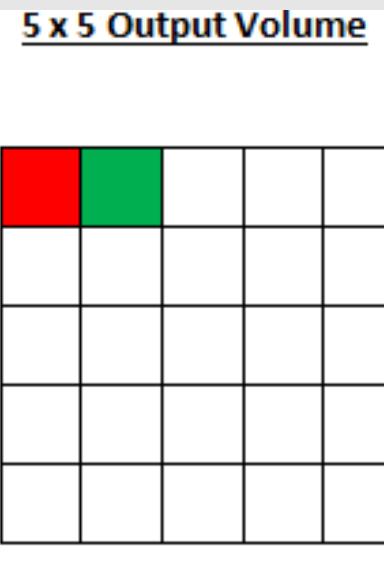
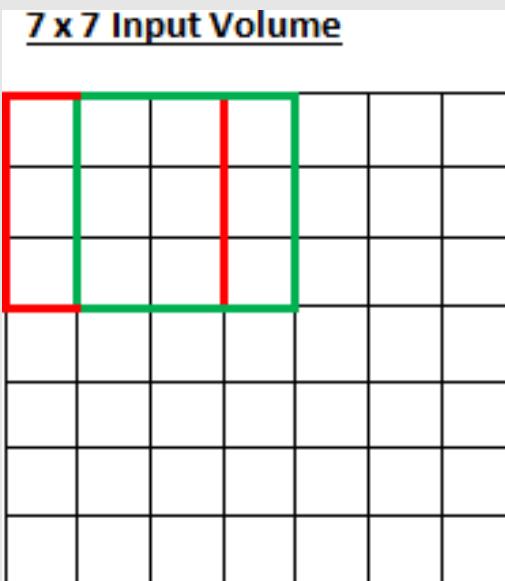
Example of convolution

Greyscale input image: 7×7

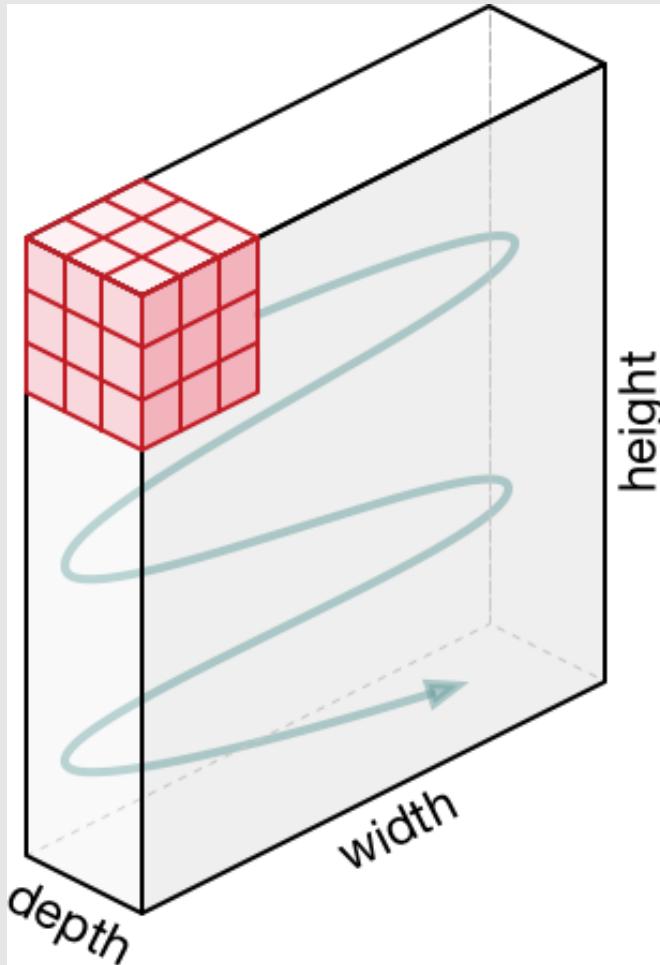
Convolution **kernel**: 3×3

Questions:

- How many units are in the output?
- How many **trainable** weights are there?



Convolution in RGB for colour images

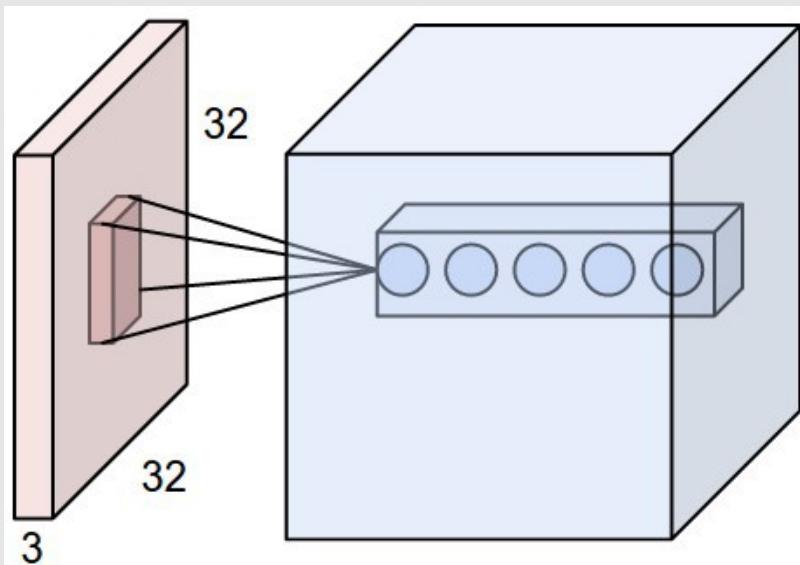


The kernel: a 3-D tensor! In this example, the kernel has size **3** $\times 3 \times 3$.

The first number **3**: the number of **input channels** or **input feature maps**

Detecting Multiple Features

- **Q:** What if we want to detect many features of the input? (e.g. **both** horizontal edges and vertical edges, and maybe even other features?)
- **A:** Have many convolutional filters!

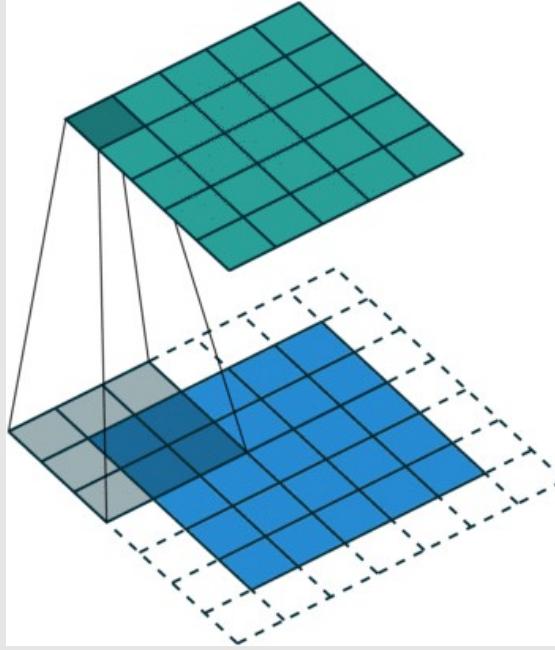


Input image size: $3 \times 32 \times 32$

Convolution kernel (4D): $3 \times 3 \times 3 \times 5$

- The number **3** is the number of **input channels** or **input feature maps**
- The number **5** is the number of **output channels** or **output feature maps**

Zero Padding

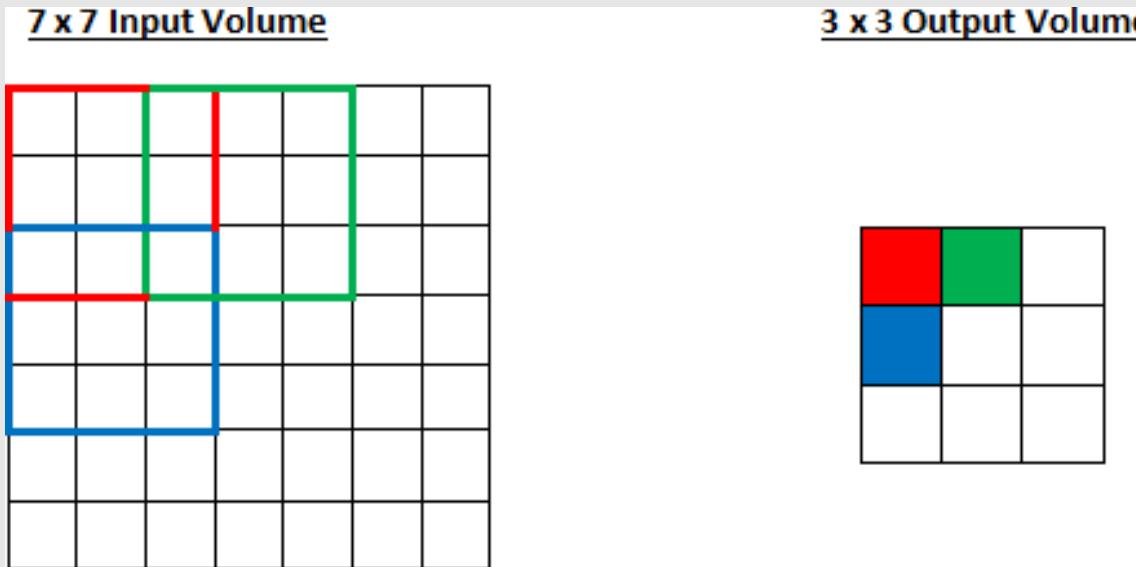


Add zeros around the border of the image (can add more than one pixel of zeros)

Question: Why might we want to add zero padding?

- Keep the next layer's width and height consistent with the previous
- Keep the information around the border of the image

Strided Convolution

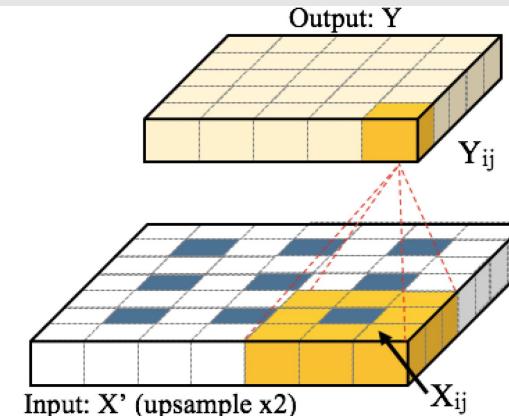
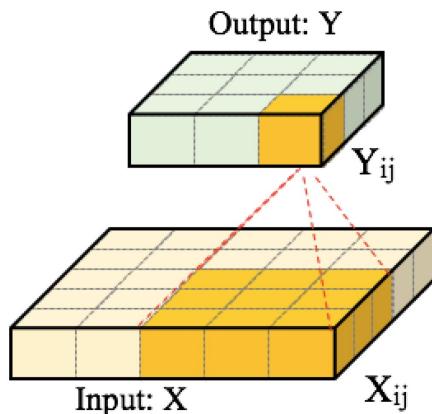


Shift the kernel by **2 (stride=2)** when computing the next output feature.

Objective: to consolidate (summarise) information

输入本来是 3×3 ，但是通过转置卷积，
使得该层变成 7×7
用 3×3 的kernel做卷积，得到 5×5 的输出

Transpose Convolution Layer



(a) Convolutional layer: the input size is $W_1 = H_1 = 5$; the receptive field $F = 3$; the convolution is performed with stride $S = 1$ and no padding ($P = 0$). The output Y is of size $W_2 = H_2 = 3$.

(b) Transposed convolutional layer: input size $W_1 = H_1 = 3$; transposed convolution with stride $S = 2$; padding with $P = 1$; and a receptive field of $F = 3$. The output Y is of size $W_2 = H_2 = 5$.

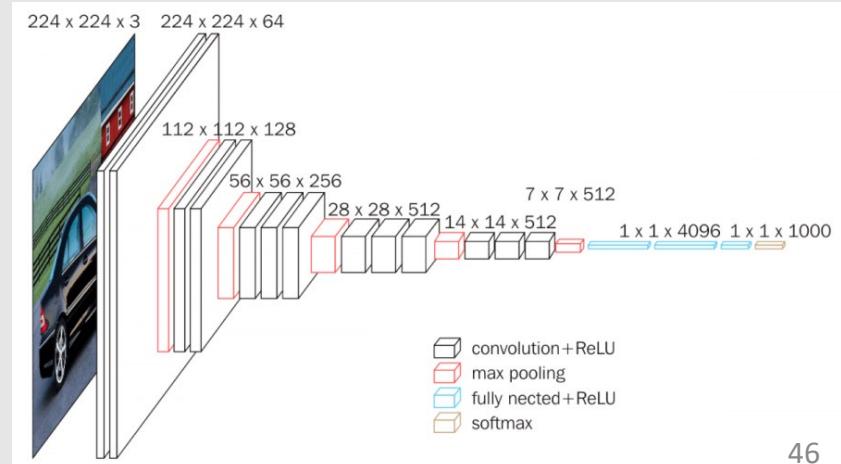
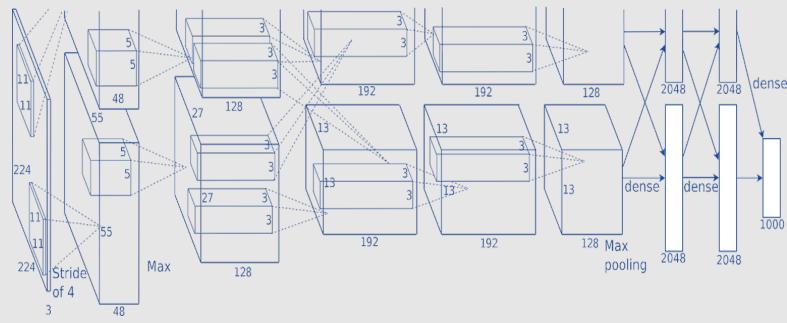
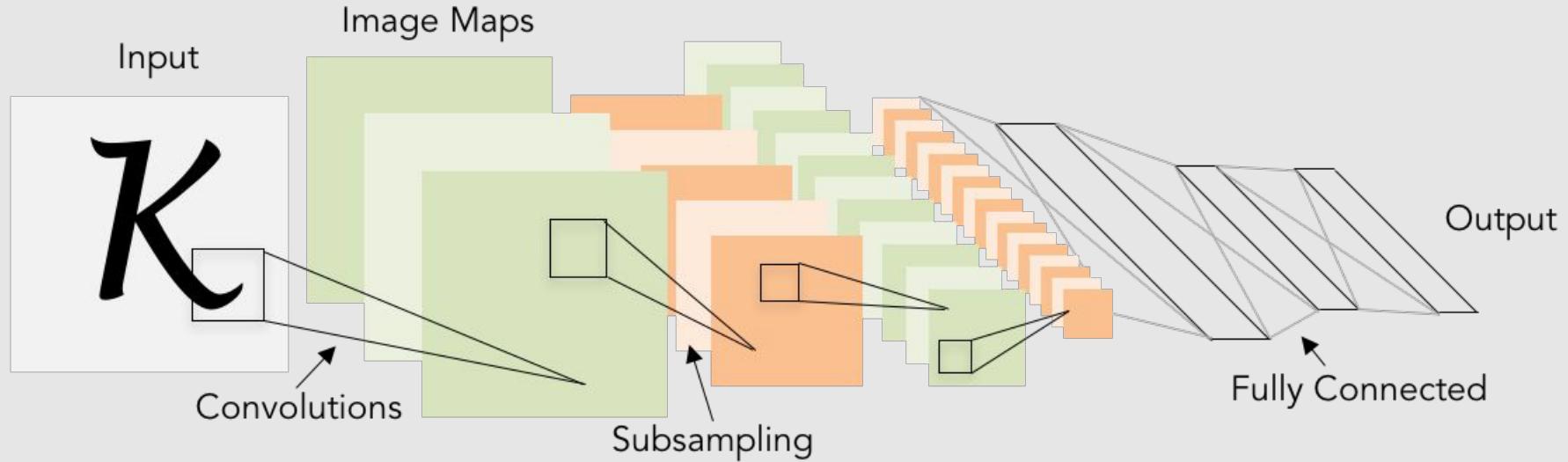
<https://www.mdpi.com/2072-4292/9/6/522/htm>

More at https://github.com/vdumoulin/conv_arithmetic

Week 7 Contents / Objectives

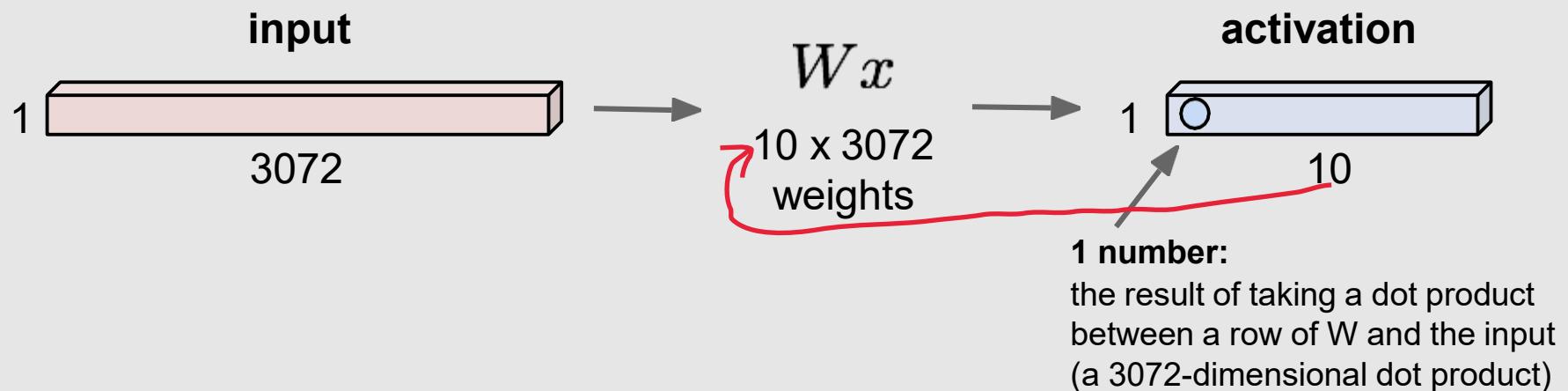
- Learning with Neurons
- Neural Networks (NNs)
- NN Decision Boundary & Features
- Convolutional NN Basics
- **Convolutional NN Unboxing**

Convolutional Neural Networks



Fully Connected Layer

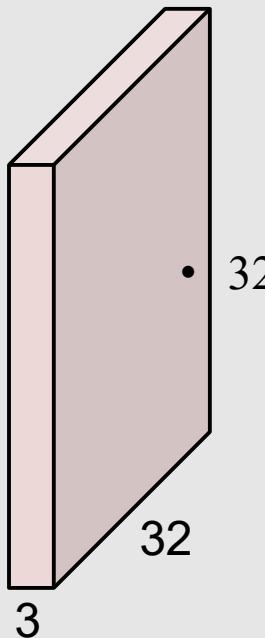
32x32x3 image → stretch to 3072 x 1



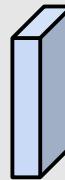
Convolution Layer

Tensor: Preserve spatial structure

- 32x32x3 image



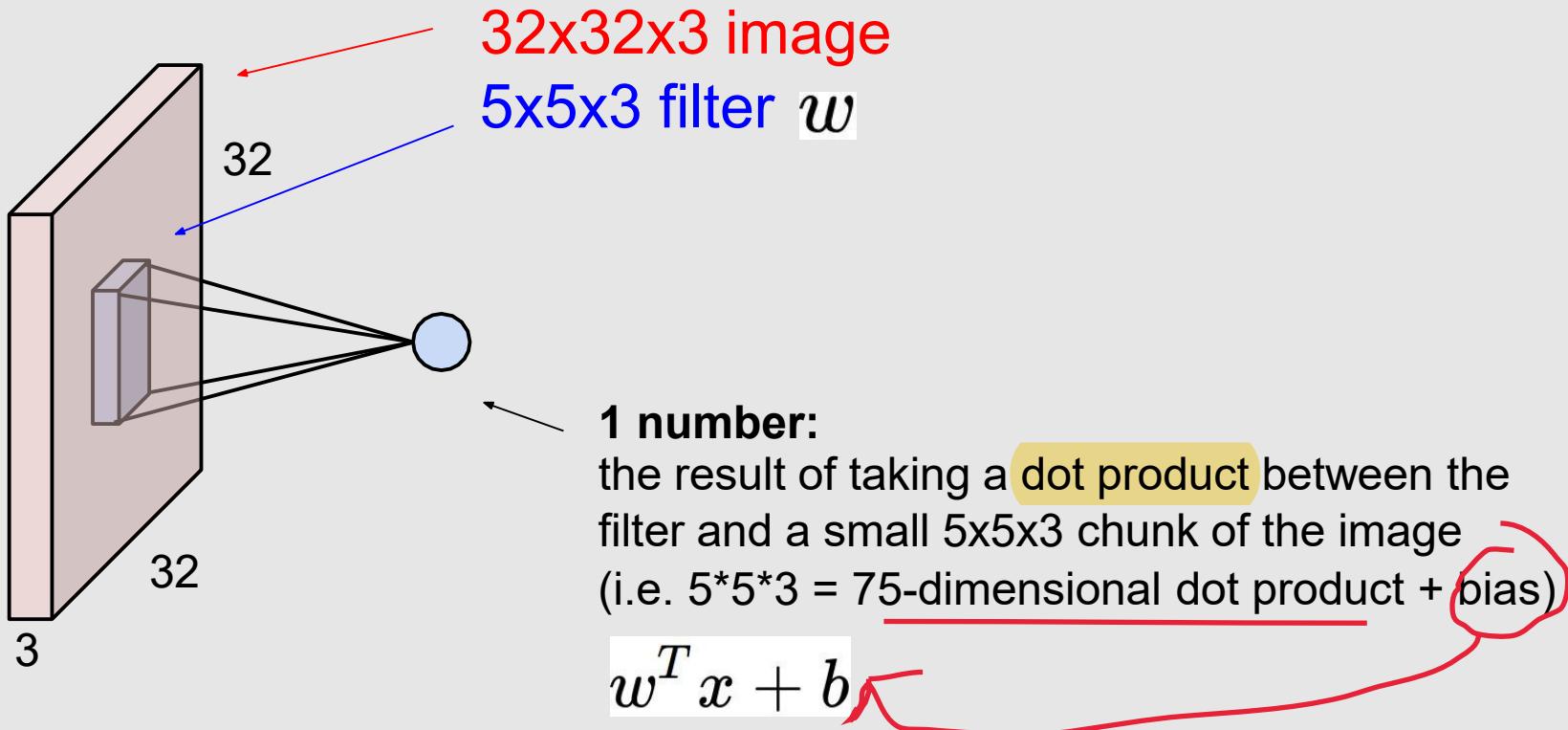
- 5x5x3 filter



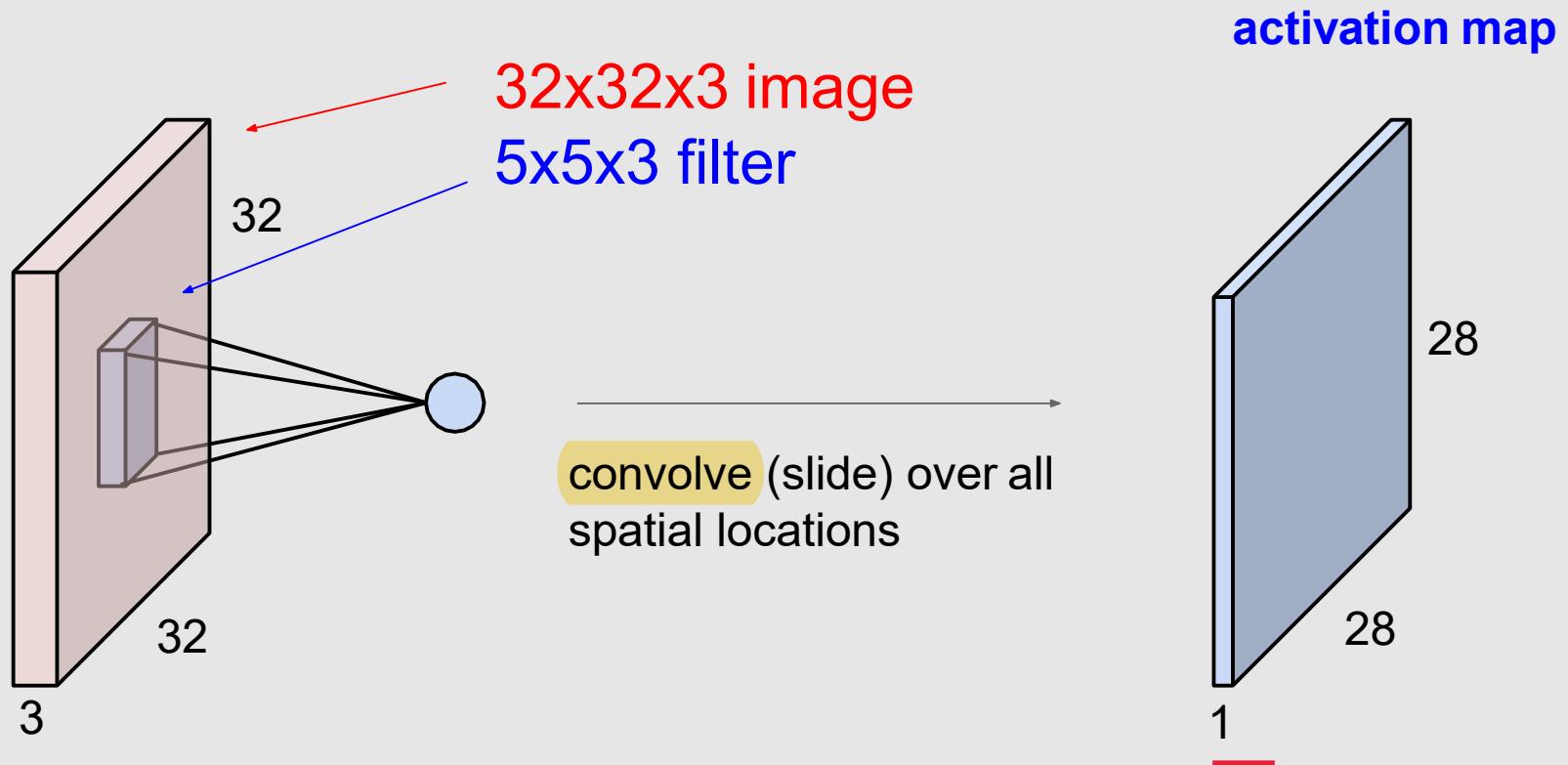
Filters always extend the full depth of the input volume

- **Convolve** the filter with the image
- i.e. “slide over the image spatially, computing dot products”

Convolution Layer

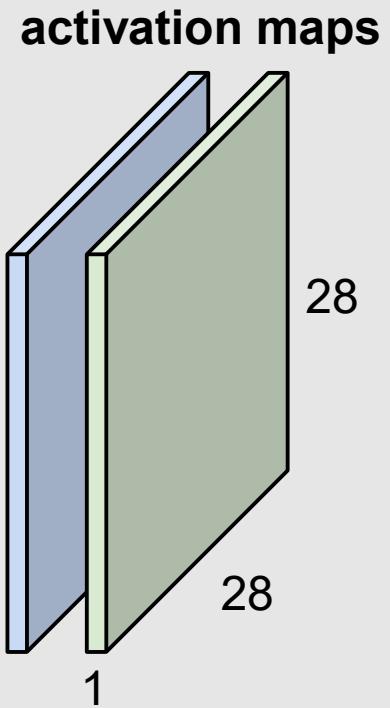
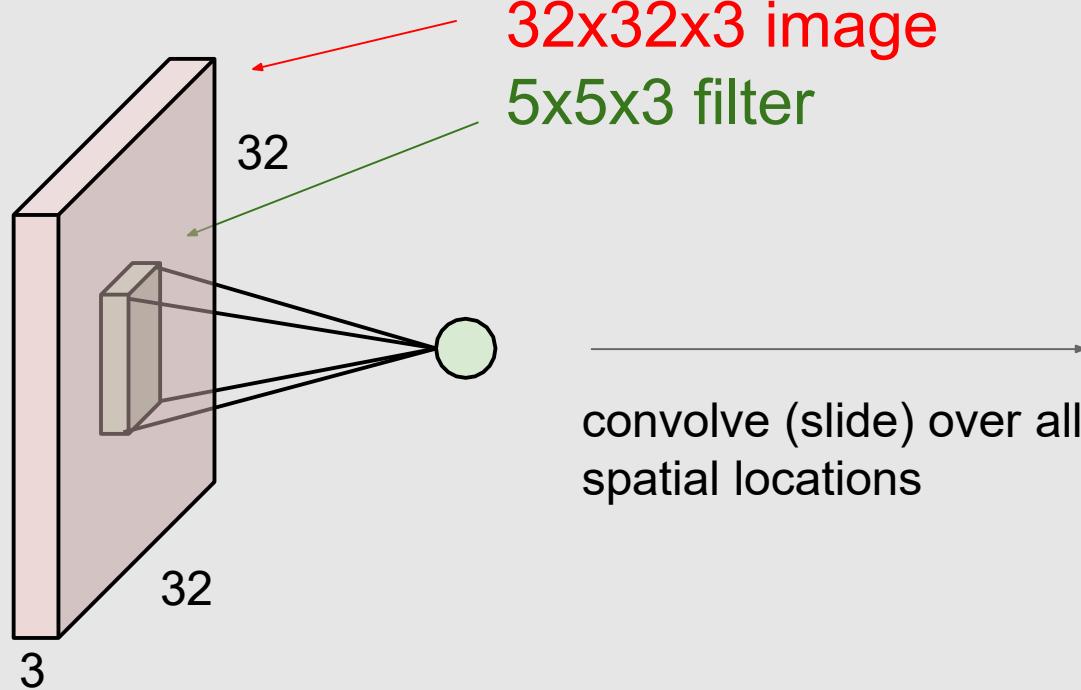


Convolution Layer



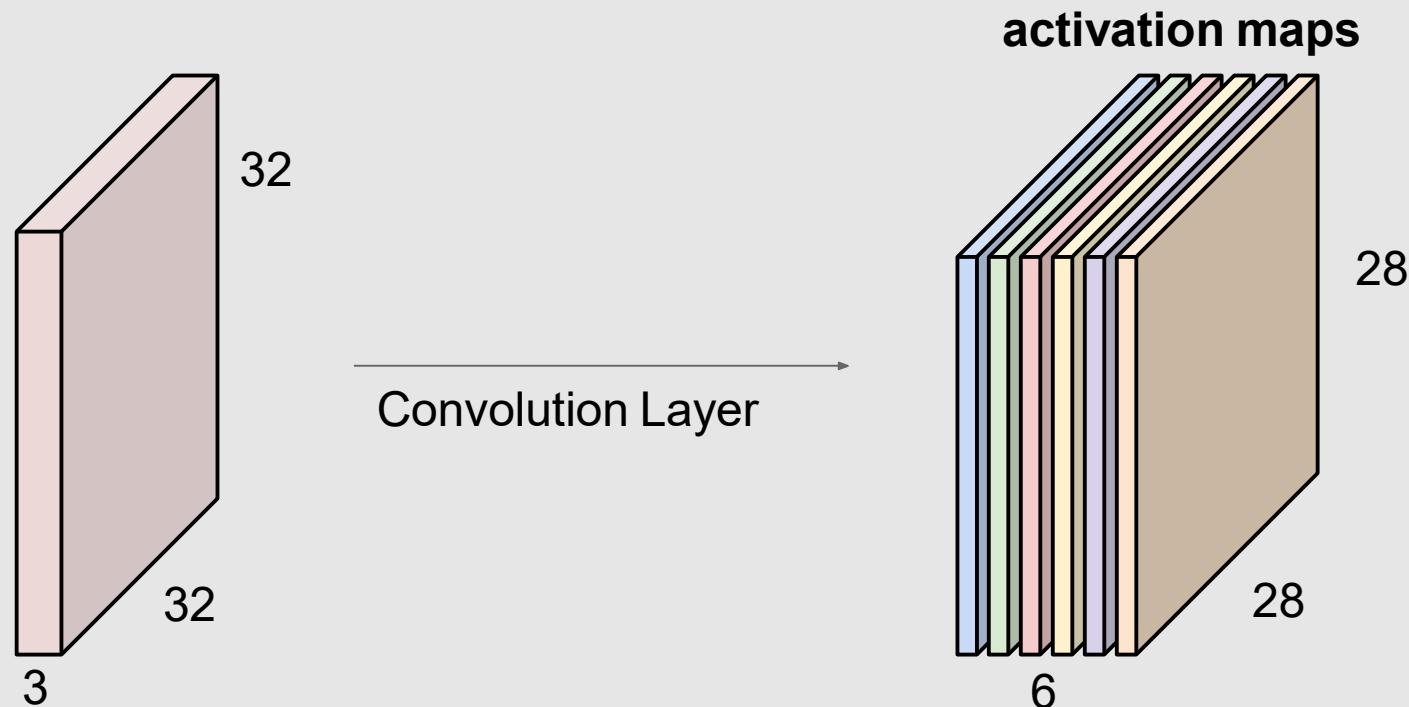
Convolution Layer

consider a second, green filter



Convolution Layer

For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



We stack these up to get a “new image” of size 28x28x6!

Convolution Operation

7

7x7 input (spatially)
assume 3x3 filter

7

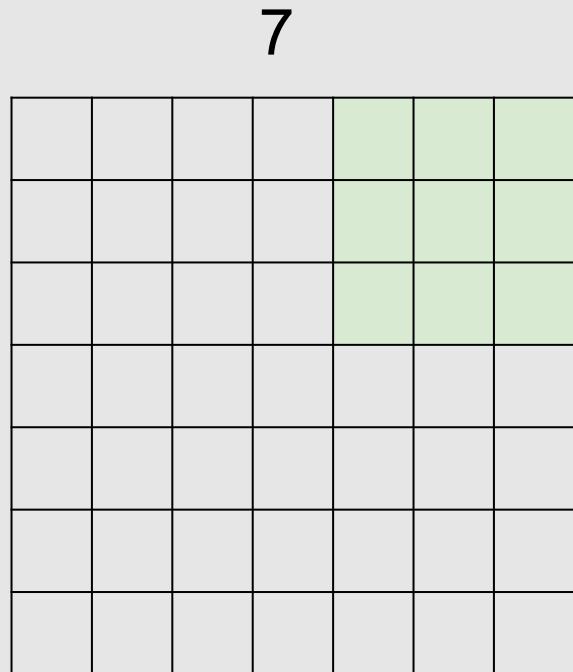
Convolution Operation

7

7x7 input (spatially)
assume 3x3 filter

7

Convolution Operation

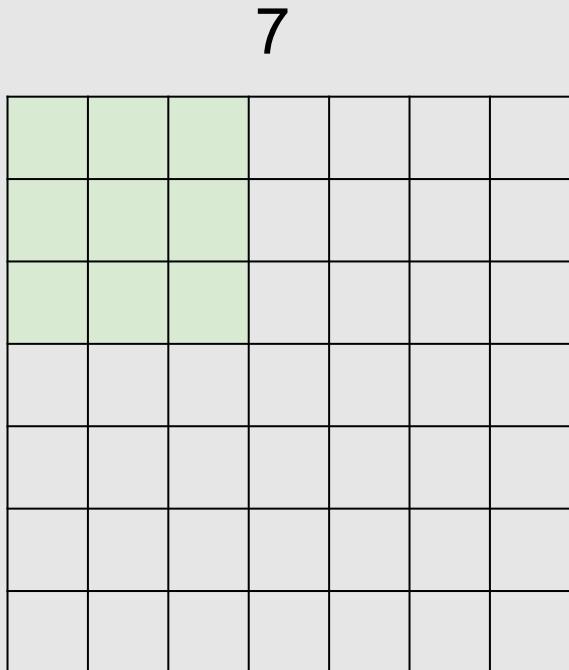


7x7 input (spatially)
assume 3x3 filter

7
→ 5x5
output

After three more sliding and dot products

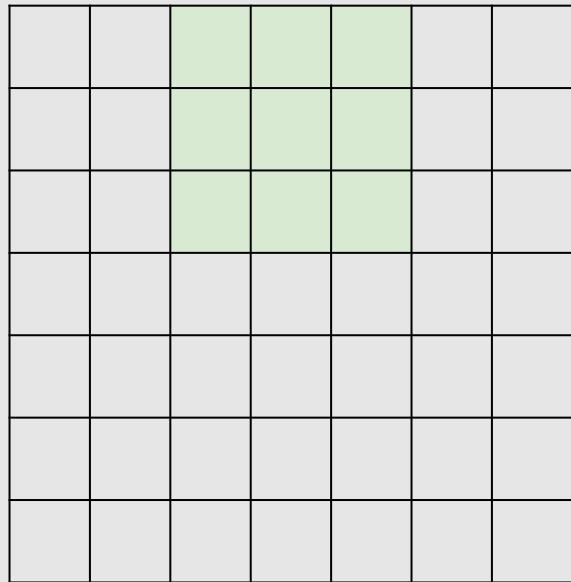
Convolution with Stride



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

Convolution with Stride

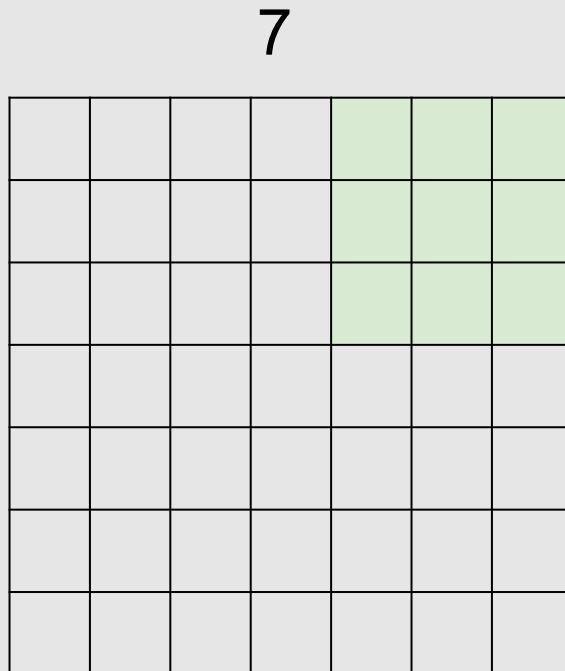
7



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

7

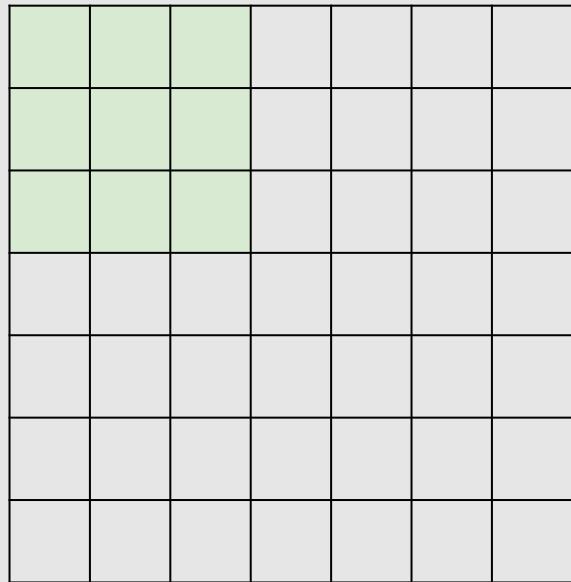
Convolution with Stride



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
→ 3x3 output!

Convolution with Stride

7

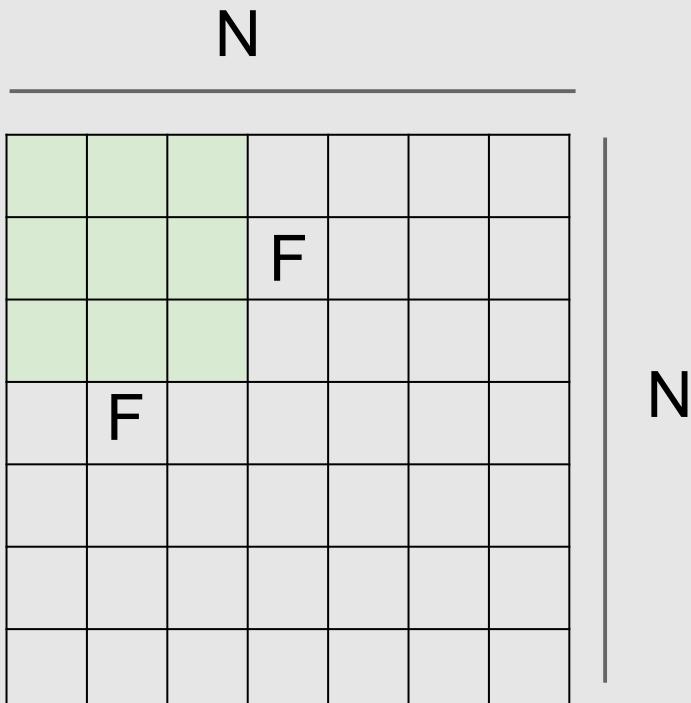


Question: 7x7 input
(spatially) assume 3x3 filter
applied **with stride 3**?

7

doesn't fit!
cannot apply 3x3 filter
on 7x7 input with stride
3 (unless ignoring parts).

Convolution – Size of output



Output size:
 $(N - F) / \text{stride} + 1$

e.g. $N = 7, F = 3$:
stride 1 => $(7 - 3)/1 + 1 = 5$
stride 2 => $(7 - 3)/2 + 1 = 3$
stride 3 => $(7 - 3)/3 + 1 = 2.33$

Zero Padding

0	0	0	0	0	0			
0								
0								
0								
0								

e.g. input 7x7

3x3 filter, applied with stride 1

pad with 1 pixel border => what is the output?

7x7 output!

in general, common to see CONV layers with
stride 1, filters of size FxF, and zero-padding with
(F-1)/2. (will preserve size spatially)

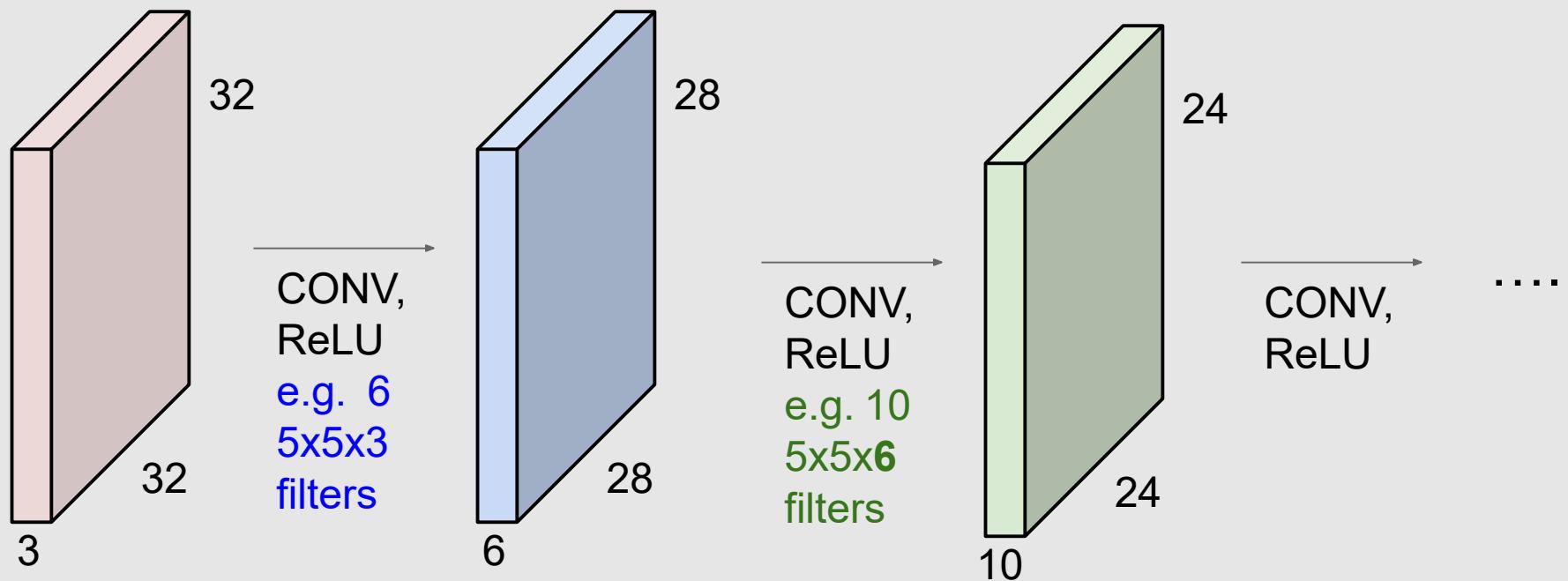
e.g. $F = 3 \Rightarrow$ zero pad with 1

$F = 5 \Rightarrow$ zero pad with 2

$F = 7 \Rightarrow$ zero pad with 3

Convolution Shrink

Example: 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially! (32 → 28 → 24 ...).

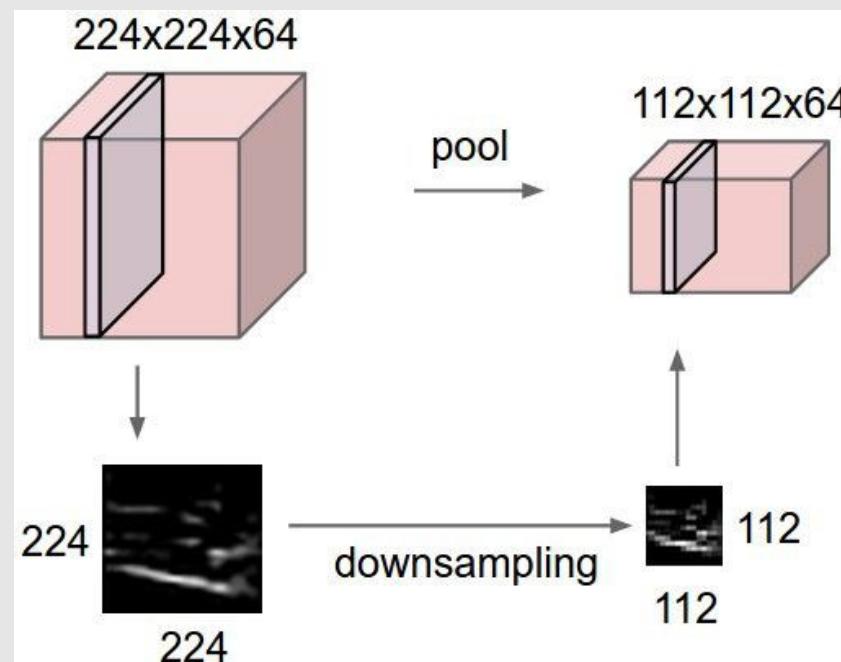


Exercises

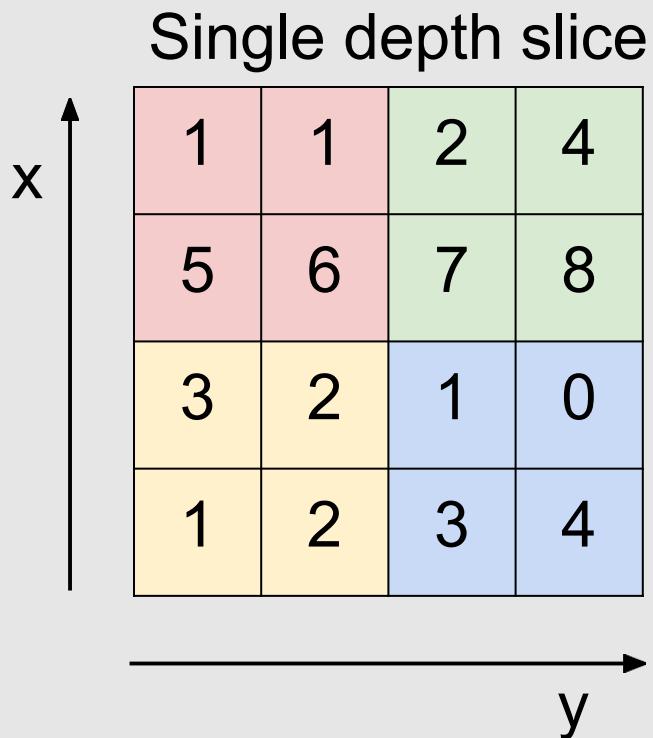
- Input volume: **32x32x3**; **10 5x5** filters with stride **1**, pad **2**
 - Output volume size?
 -
- Number of parameters for this layer?
 -

Pooling Layer: Downsampling

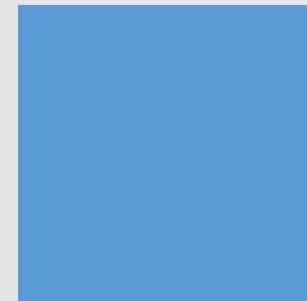
- Make the representations smaller and more manageable → dimensionality reduction
- Operate over each activation map independently:



Max Pooling



max pool with 2x2 filters
and stride 2



Lab 7 CNN (See notebook for details)

```
__init__(self):
super(CNN, self).__init__()
self.conv1 = nn.Conv2d(3, 6, 5) #3: #6: #12: #16: #32: #64: #128: #256: #512: #1024: #2048: #4096: #8192: #16384: #32768: #65536: #131072: #262144: #524288: #1048576: #2097152: #4194304: #8388608: #16777216: #33554432: #67108864: #134217728: #268435456: #536870912: #1073741824: #2147483648: #4294967296: #8589934592: #17179869184: #34359738368: #68719476736: #137438953472: #274877906944: #549755813888: #1099511627776: #2199023255552: #4398046511104: #8796093022208: #17592186044016: #35184372088032: #70368744176064: #140737488352128: #281474976704256: #562949953408512: #112589990681728: #225179981363456: #450359962726912: #900719925453824: #1801439850907648: #3602879701815296: #7205759403630592: #1441151880726184: #2882303761452368: #5764607522904736: #11529215045809472: #23058430091618944: #46116860183237888: #92233720366475776: #18446740713295552: #36893481426591104: #73786962853182208: #14757392570636416: #29514785141272832: #59029570282545664: #118059140565091328: #236118281130182656: #472236562260365312: #944473124520730624: #188894624904146128: #377789249808292256: #755578499616584512: #1511156999232169024: #3022313998464338048: #6044627996928676096: #12089255993857352192: #24178511987714704384: #48357023975429408768: #96714047950858817536: #193428095901717635072: #386856191803435270144: #773712383606870540288: #1547424767213741080576: #3094849534427482161152: #6189699068854964322304: #12379398137709928644608: #24758796275419857289216: #49517592550839714578432: #99035185101679429156864: #198070370203358858313728: #396140740406717716627456: #792281480813435433254912: #158456281626687066649824: #316912563253374133299648: #633825126506748266599296: #126765025311489533199592: #253530050622979066399184: #507060101245958132798368: #1014120202491916265596736: #2028240404983832531193472: #4056480809967665062386848: #8112961619935330124773696: #1622592323987066024954792: #3245184647974132049859584: #6490369295948264099719168: #1298073859189652819943832: #2596147718379305639887664: #5192295436758611279775328: #10384590873517222559550656: #20769181747034445119101312: #41538363494068890238202624: #83076726988137780476405248: #16615345397627556095281096: #33230690795255112190562192: #66461381590510224381124384: #13292276318102044876224768: #26584552636204089752449536: #53169105272408179504898720: #10633821054481635900977640: #21267642108963271801955280: #42535284217926543603910560: #85070568435853087207821120: #17014113687176574401562240: #34028227374353148803124800: #68056454748706297606249600: #13611290949741259521299200: #27222581899482519042598400: #54445163798965038085196800: #10889032759793007617193600: #21778065519586015234387200: #43556131039172030468774400: #87112262078344060937548800: #17422452415668812187509600: #34844904831337624375019200: #69689809662675248750038400: #13937961932535049500076800: #27875923865070099000153600: #55751847730140198000307200: #11150369546028039600614400: #22300739092056079201228800: #44601478184112158402457600: #89202956368224316804915200: #17840591273644863360983200: #35681182547289726721966400: #71362365094579453443932800: #14272473018915890688786400: #28544946037831781377572800: #57089892075663562755145600: #11417978415132732550331200: #22835956830265465100662400: #45671913660530930201324800: #91343827321061860402649600: #18268765464212372080529600: #36537530928424744161059200: #73075061856849488322118400: #14615012371369897664423200: #29230024742739795328846400: #58460049485479590657692800: #116920098909591981315385600: #233840197819183962630771200: #467680395638367925261542400: #935360791276735850523084800: #1870721582553471701046169600: #3741443165106943402092339200: #7482886322213886804184678400: #1496577264442773360836956800: #2993154528885546721673913600: #5986309057771093443347827200: #1197261811554218688669654400: #2394523623108437377339308800: #4789047246216874754678617600: #9578094492433749509357235200: #19156188984867499018714470400: #38312377969734998037428940800: #76624755939469996074857881600: #153249511878939980149755763200: #306499023757879960299511526400: #612998047515759920599023052800: #1225996095031519801198046055600: #2451992190063039602396092111200: #4903984380126079204792184222400: #9807968760252158409584368444800: #19615935520503116819167376889600: #39231871041006233638334753779200: #78463742082012467276669507558400: #15692748416402493455339015116800: #31385496832804986910678030233600: #62770993665609973821356060467200: #12554198733121994764273012093600: #25108397466243989528546024187200: #50216794932487979057092048374400: #100433589864959558114184096748800: #200867179729919116228368193497600: #401734359459838232456736386995200: #803468718919676464913472773990400: #160693743783933292926745554798080: #321387487567866585853491109596160: #642774975135733171706982219192320: #128554995027146634341396443838640: #257109990054293268682792887677280: #514219980108586537365585775354560: #1028439802171773074731171550709120: #2056879604343546149462343101418240: #4113759208687092298924686202836480: #8227518417374184597849372405672960: #1645503683474778119569874801145920: #3291007366949556239139749602291840: #6582014733899112478279499204583680: #1316402946789824955655899408967360: #2632805893579649911311798817934720: #5265611787159299822623597635869440: #1053122357431859964524715327173880: #2106244714863719929049430654347760: #4212489429727439858098861308695520: #8424978859454879716197722617391040: #16849957718909795432395445234782080: #33699915437819590864790890469564160: #67399830875639181729581780939128320: #134799661751278363459163561878256640: #269599323502556726918327123756513280: #538998647005113453836654247513026560: #1077997304010226857673308495026053120: #2155994608020453715346616985052106240: #4311989216040907430693233970104212480: #8623978432081814861386467940208424960: #17247956861636289722732955880416849920: #34495913723272579445465911760833699840: #68991827446545158890931823521667399680: #137983654893090317781863647043346799360: #275967309786180635563727294086693598720: #551934619572361271127454588173387197440: #1103869239144722542254901176346743948880: #2207738478289445084509802352693487897760: #4415476956578890169019604705386955795520: #8830953913157780338039209410773911590400: #17661907826315560676078418821547823180800: #35323815652631121352156837643095646361600: #70647631305262242704313675286191292723200: #14129526261052445408626750457238258446400: #28259052522054890817253500814476516892800: #5651810504410978163450700162895303377600: #11303621008821956326901400325785606753600: #22607242017643912653802800651571213507200: #45214484035287825307605600130357427014400: #90428968070575650615211200260714854028800: #18085793614115130123042400521429708057600: #36171587228230260246084800104289416115200: #72343174456460520492169600208578832230400: #14468634891291054098433600417157666446400: #28937269782582108196867200834315332892800: #57874539565164216393734400166863665785600: #11574907913032843278746800333727331571200: #23149815826065686557493600667454663142400: #46299631652131373114987200133510932684800: #92599263304262746229974400267021865369600: #18519852660852549245994800534043730743200: #37039705321652598491989600106807461486400: #74079410643255196983979200213614922972800: #14815882128651039396795200427229845945600: #29631764257252078793590400854459691891200: #59263528514504157587180800170899383782400: #118527057029008315174361600341798767564800: #237054114058016630348723200683597535129600: #474108228116033260697446400136719517539200: #948216456232066521394892800273439035078400: #1896432912464133042789785600546878070156800: #3792865824928266085579571200109376140313600: #7585731649856532171159142400218752280673200: #15171463297713064342382848004375045612346400: #30342926595426128684765696008750091224692800: #60685853190852257369531392001750082449385600: #121371706381704514738626848035001648897771200: #242743412763409029477253696070032817755542400: #485486825526818058954507392140065635511084800: #970973651053636017909014784280013127102169600: #194194730205727203581803568560026254204339200: #388389460411454407163607137120052508408678400: #776778920822908814327214274240010501681356800: #1553557841645817286554285548480021003362713600: #3107115683291634573108571096960042006725427200: #6214231366583269146217142193920084001345854400: #12428462733166582894434284387840168002685708800: #24856925466333165788868568775680336005371417600: #49713850932666331577737137551360672001074285200: #99427701865332663155474275102721344002148570400: #198855403730665326339491525205442688004287140800: #397710807461330652678983050410885376008574281600: #795421614922661305357966100821770752001714853200: #159084322944532261071593220164354150400342856400: #31816864588906452214318644032870300080068512800: #63633729177812904428637288065740600160013705600: #12726748355562580885727457613140100320027411200: #25453496711125161771454915226280200640054822400: #509069934222503235429098304525604001280109644800: #1018139868445006670858196609051208002560219289600: #2036279736890013341716393218102416005120438579200: #4072559473780026683432786436204832001024087158400: #8145118947560053366865572872409664002048174316800: #16290238891200133413731155744819328004096348633600: #32580477782400266833462311489638656008192697267200: #65160955564800533666924622979277312001638535534400: #130321911129601334133849245558554624003277071068800: #26064382225920266833769849111709848006554142033600: #52128764451840533667339698223419696001310828407200: #104257528903680266834793976446839392002621656814400: #208515057807360533669587952893678784005243313628800: #417030115614721067339175905787357568001048665725600: #834060231229442134678351811574715136002097331451200: #166812046245884226756675762314853027200419466702400: #333624092491768453513351524629705054400838933404800: #667248184983536907026703049259410108800167866809600: #1334496369967073814053406098518202116800335733619200: #266899273993414762810681219703640423200671467038400: #53379854798682952562136243940728084640013493476800: #106759709593365905124272487881456169280026986533600: #213519419186731810248544975762912338400539731067200: #42703883837346362049688995152582467680010794303200: #85407767674692724099377990305164935360021588606400: #170815535353855481997559806010329710720043177212800: #341631070707710963995119601202065942040086354425600: #683262141415421927990239202404131884080017278851200: #1366524282830843859804784048082637768160034557702400: #2733048565661687719609568096165275536320069115404800: #5466097131323375439219136192327551072640013823059200: #1093219426264675079443827384645512245280027646018400: #2186438852529350158887654769291024490560055292036800: #4372877705058700317775309538582048981120011058073600: #8745755410117400635550619077164097962240022116147200: #1749151082023500127110338154328195924480044232294400: #3498302164047000254220676308656391848960088464588800: #6996604328094000508441352617312783697920017692977600: #1399320865618000101688275323463567339584003538595200: #279864173123600020337655064692714467916800707718400: #559728346247200040675310129385428935833600141546800: #111945669249400081350620258771085787167200283093600: #223891338498800162701240517542171574334400566187200: #447782676997600325402481035084343148688001132374400: #895565353995200650804962070168686293776002264748800: #179113070798400130169924144033734587552004529497600: #35822614159680026033984828806747117104009058995200: #71645228319360052067969657613485423408001817990400: #14329045638720010413939315326858846816003635980800: #28658091277440020827878630653717773632007271961600: #57316182554880041655757261307435547264001455923200: #114632365109600833311544126148711094528002911846400: #229264730219201666622788252297422209056005823692800: #458529460438403333245576504594844418112001164737600: #917058920876806666491153008189688836224002329475200: #183411784175361333288236016379377767248004658950400: #366823568350722666576472032758755534496009317900800: #733647136701445333152944065517511068960018635801600: #1467294273402886662314880131035022137920037271603200: #2934588546805773324629760262070044275840074543206400: #5869177093611546649259520524140088451680014908612800: #1173835418722309329451840104828017692320029816825600: #2347670837444618658903680209656035384640059633651200: #4695341674889237317807360419312070769280011926702400: #9390683349778474635614720838624141538560023853404800: #1878136669955694927122940167724828277120047706809600: #375627333991
```

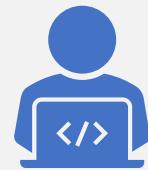
Acknowledgement

- The slides used materials from:
Matt Gormley, Eric Xing, Nina Balcan, Fei-Fei Li & Justin Johnson & Serena Yeung, Lisa Zhang, Michael Guerzhoy, Svetlana Lazebnik, Ismini Lourentzou, Dekai Wu

Recommended Reading

- [CS231n: Convolutional Neural Networks for Visual Recognition from Stanford \(Fei-Fei Li et al.\)](#)
- [The Deep Learning Book with a free official html version provided by the authors \(Ian Goodfellow et al.\)](#)
- [Convolution arithmetic](#)
- PyTorch documentations
- The lab notebook and references

Next



Lab notebooks



Feedback (if any)
@end of the week