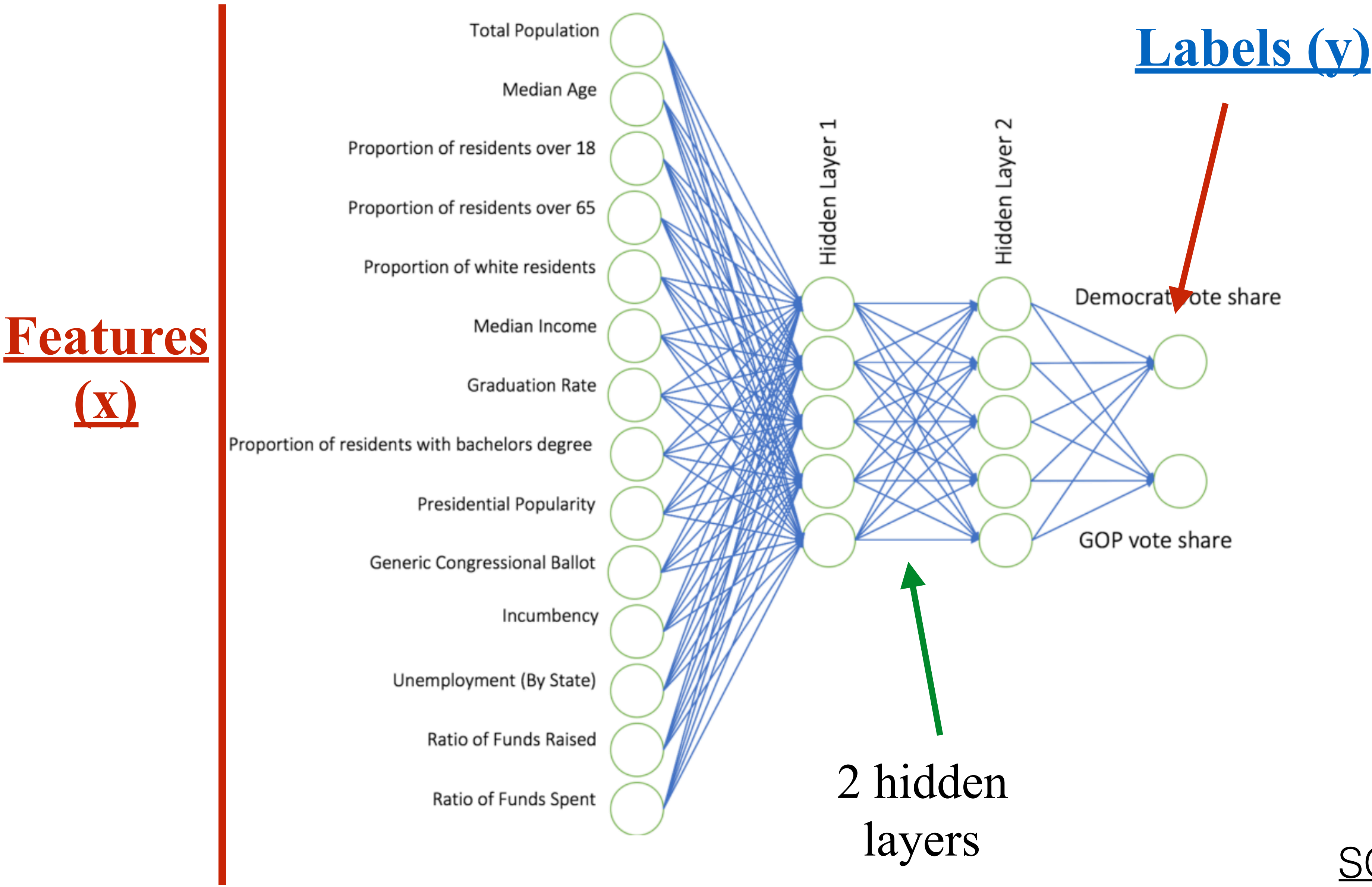
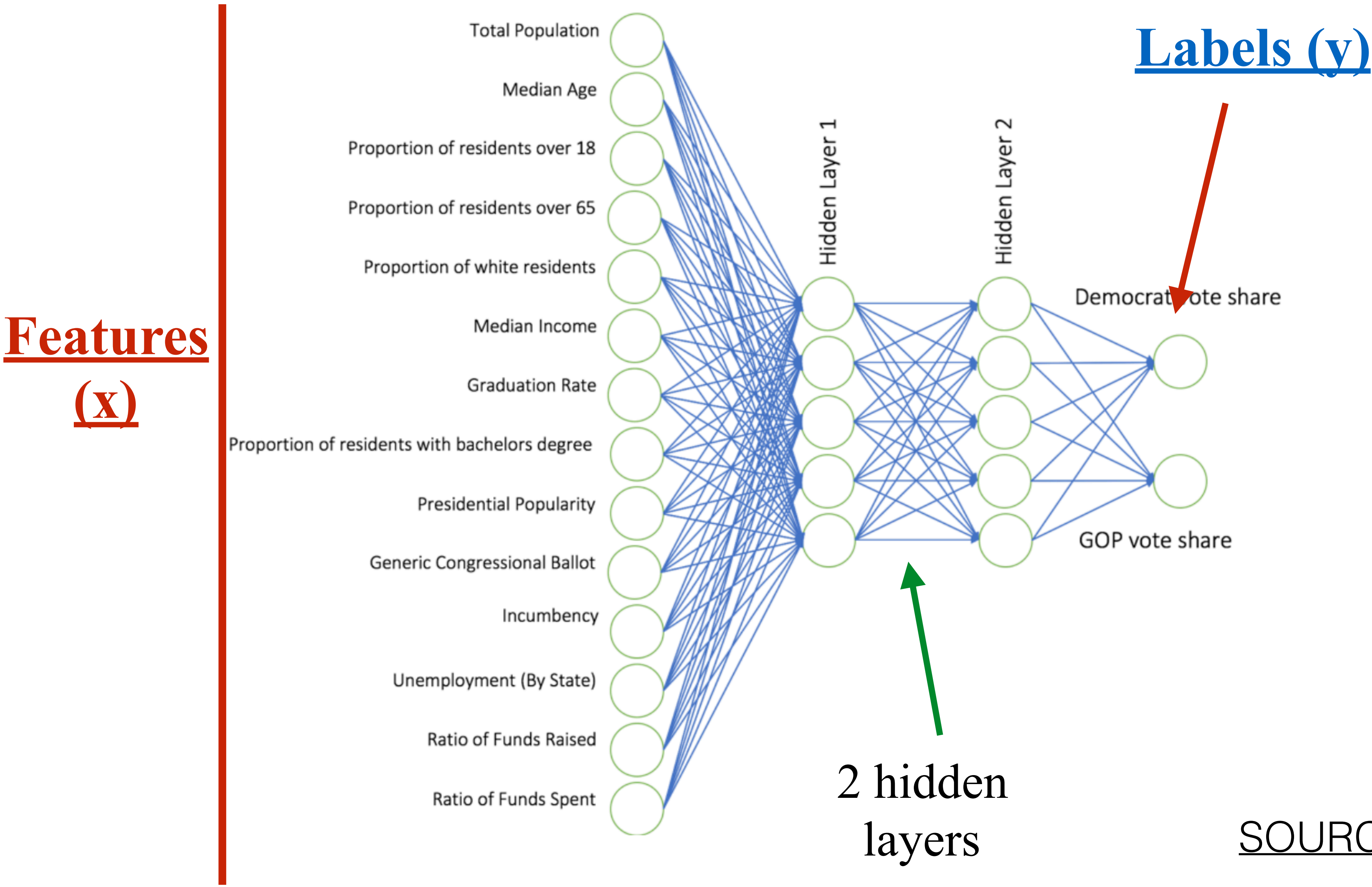
NEURAL NETWORK TO PREDICT RESULTS OF MIDTERM ELECTIONS



SOURCE

NEURAL NETWORK TO PREDICT RESULTS OF MIDTERM ELECTIONS

$$p = g_3(W_3 g_2(W_2 g_1(W_1 \vec{x_0})))$$



SOURCE

NEURAL NETWORK TO PREDICT RESULTS OF MIDTERM ELECTIONS

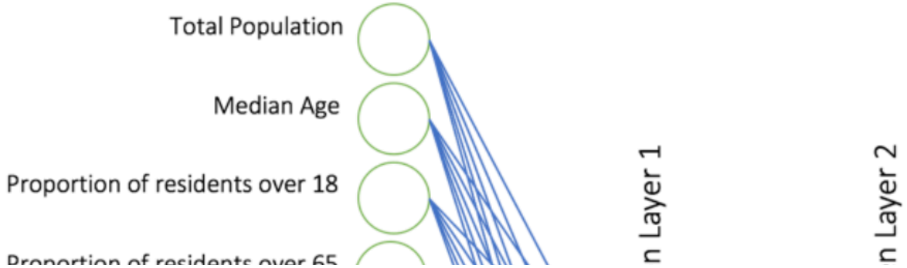
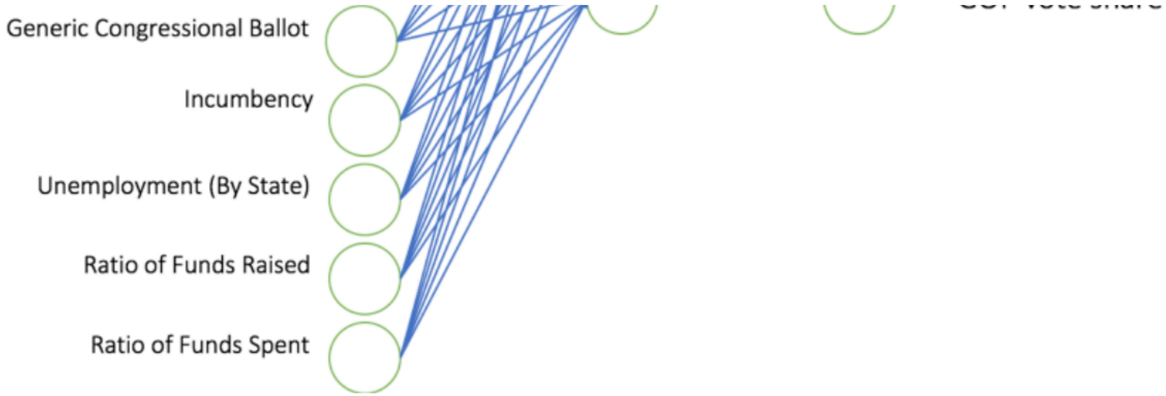


Table 2 – Results of both Models:

Model not including past elections (Model A)	***Model including past elections (Model B)***
D +3	***D +17***
Democrat Seats: 219	***Democrat Seats: 226***
Republican Seats: 216	***Republican Seats: 209***
33% chance of Republicans keeping house*	**0.3% chance of Republicans keeping house***

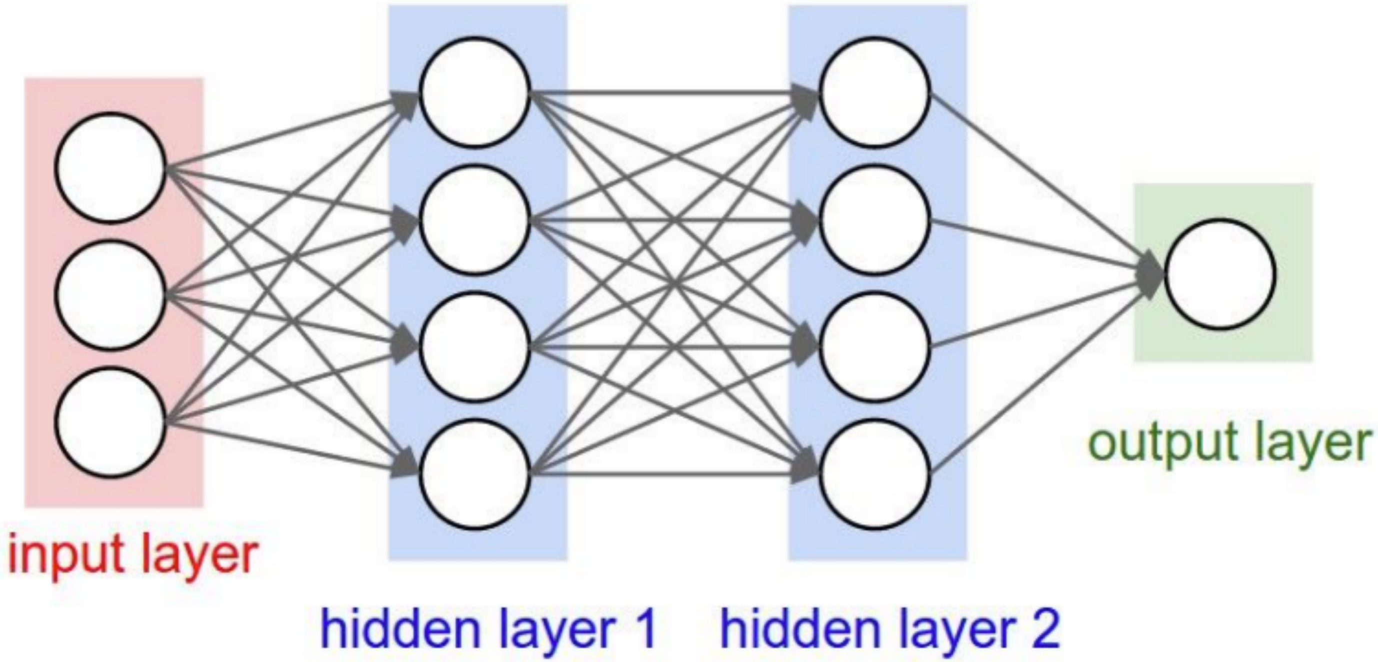


SOURCE

OK, SO NOW LET'S FIND THE WEIGHTS

OPTIMIZATION

[OR HOW TO FIND THE WEIGHTS?]

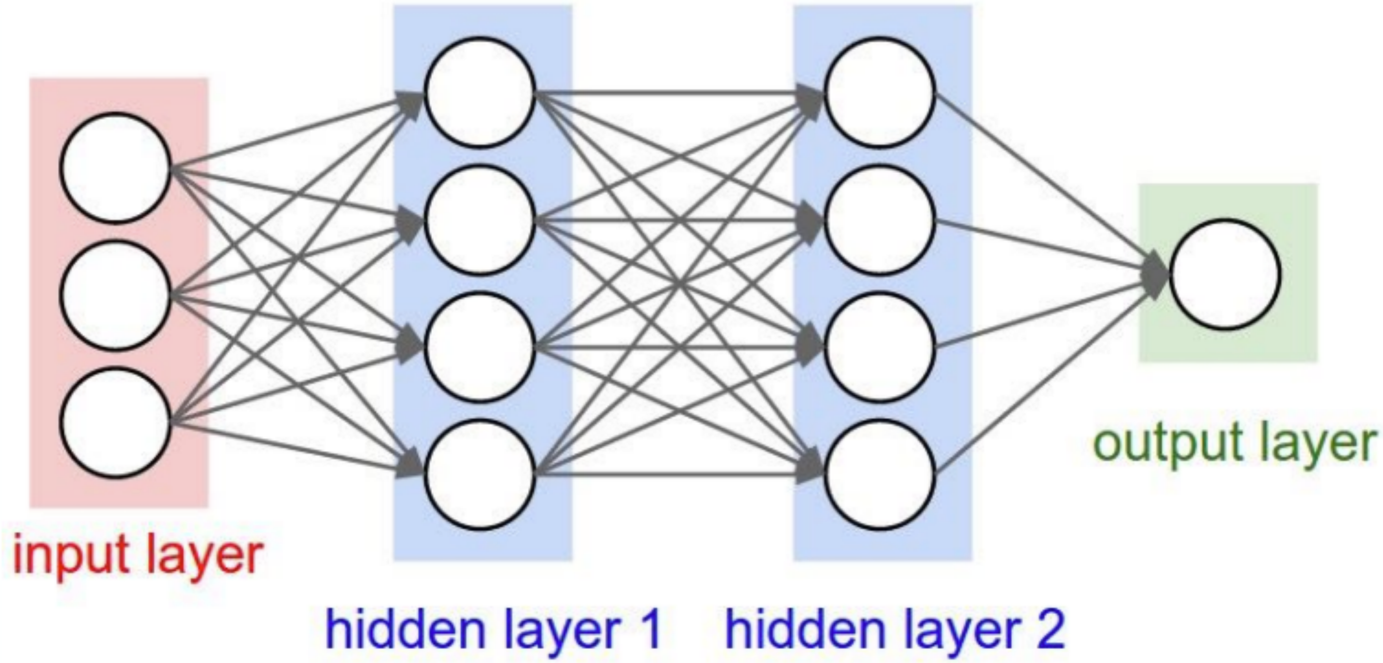


$$p = g_3(W_3 g_2(W_2 g_1(W_1 \vec{x_0})))$$

NETWORK FUNCTION

OPTIMIZATION

[OR HOW TO FIND THE WEIGHTS?]



$$p = g_3(W_3 g_2(W_2 g_1(W_1 \vec{x_0})))$$

$$\frac{1}{N}\sum_{i=1}^{N}(y_i - p_i)^2$$

LOSS FUNCTION

WE SIMPLY WANT TO MINIMIZE THE LOSS FUNCTION WITH RESPECT TO THE WEIGHTS, i.e. FIND THE WEIGHTS THAT GENERATE THE MINIMUM LOSS

WE SIMPLY WANT TO MINIMIZE THE LOSS FUNCTION WITH RESPECT TO THE WEIGHTS, i.e. FIND THE WEIGHTS THAT GENERATE THE MINIMUM LOSS

WE THEN USE STANDARD MINIMIZATION ALGORITHMS THAT YOU ALL KNOW…

FOR EXAMPLE….

Gradient Descent

$$W_{t+1} = W_t - \lambda_t \nabla f(W_t)$$

[gradient]

Newton

$$W_{t+1} = W_t - \lambda [H f(W_t)]^{-1} \nabla f(W_t)$$

[hessian]

NEWTON CONVERGES FASTER…

FOR EXAMPLE....

Gradient Descent

$$W_{t+1} = W_t - \lambda_t \nabla f(W_t)$$

[gradient]

Newton

$$W_{t+1} = W_t - \lambda [H f(W_t)]^{-1} \nabla f(W_t)$$

[hessian]

NEWTON CONVERGES FASTER…

BUT NEEDS THE HESSIAN

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 f}{\partial W_1^2} & \frac{\partial^2 f}{\partial W_1\,\partial W_2} & \cdots & \frac{\partial^2 f}{\partial W_1\,\partial W_n} \\ \frac{\partial^2 f}{\partial W_2\,\partial W_1} & \frac{\partial^2 f}{\partial W_2^2} & \cdots & \frac{\partial^2 f}{\partial W_2\,\partial W_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial W_n\,\partial W_1} & \frac{\partial^2 f}{\partial W_n\,\partial W_2} & \cdots & \frac{\partial^2 f}{\partial W_n^2} \end{bmatrix}$$

FOR EXAMPLE….

Gradient Descent

$$W_{t+1} = W_t - \lambda_t \nabla f(W_t)$$

[gradient]

MOST USED BY FAR….

Newton

$$W_{t+1} = W_t - \lambda [H f(W_t)]^{-1} \nabla f(W_t)$$

[hessian]

NEWTON CONVERGES FASTER…

BUT NEEDS THE HESSIAN

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 f}{\partial W_1^2} & \frac{\partial^2 f}{\partial W_1\,\partial W_2} & \cdots & \frac{\partial^2 f}{\partial W_1\,\partial W_n} \\ \frac{\partial^2 f}{\partial W_2\,\partial W_1} & \frac{\partial^2 f}{\partial W_2^2} & \cdots & \frac{\partial^2 f}{\partial W_2\,\partial W_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial W_n\,\partial W_1} & \frac{\partial^2 f}{\partial W_n\,\partial W_2} & \cdots & \frac{\partial^2 f}{\partial W_n^2} \end{bmatrix}$$

FOR EXAMPLE....

Gradient Descent

$$W_{t+1} = W_t - \lambda_t \nabla f(W_t)$$

[gradient]

EVERYTHING RELIES
ON COMPUTING THE GRADIENT

MOST USED BY FAR....

Newton

$$W_{t+1} = W_t - \lambda [H f(W_t)]^{-1} \nabla f(W_t)$$

[hessian]

NEWTON CONVERGES FASTER…

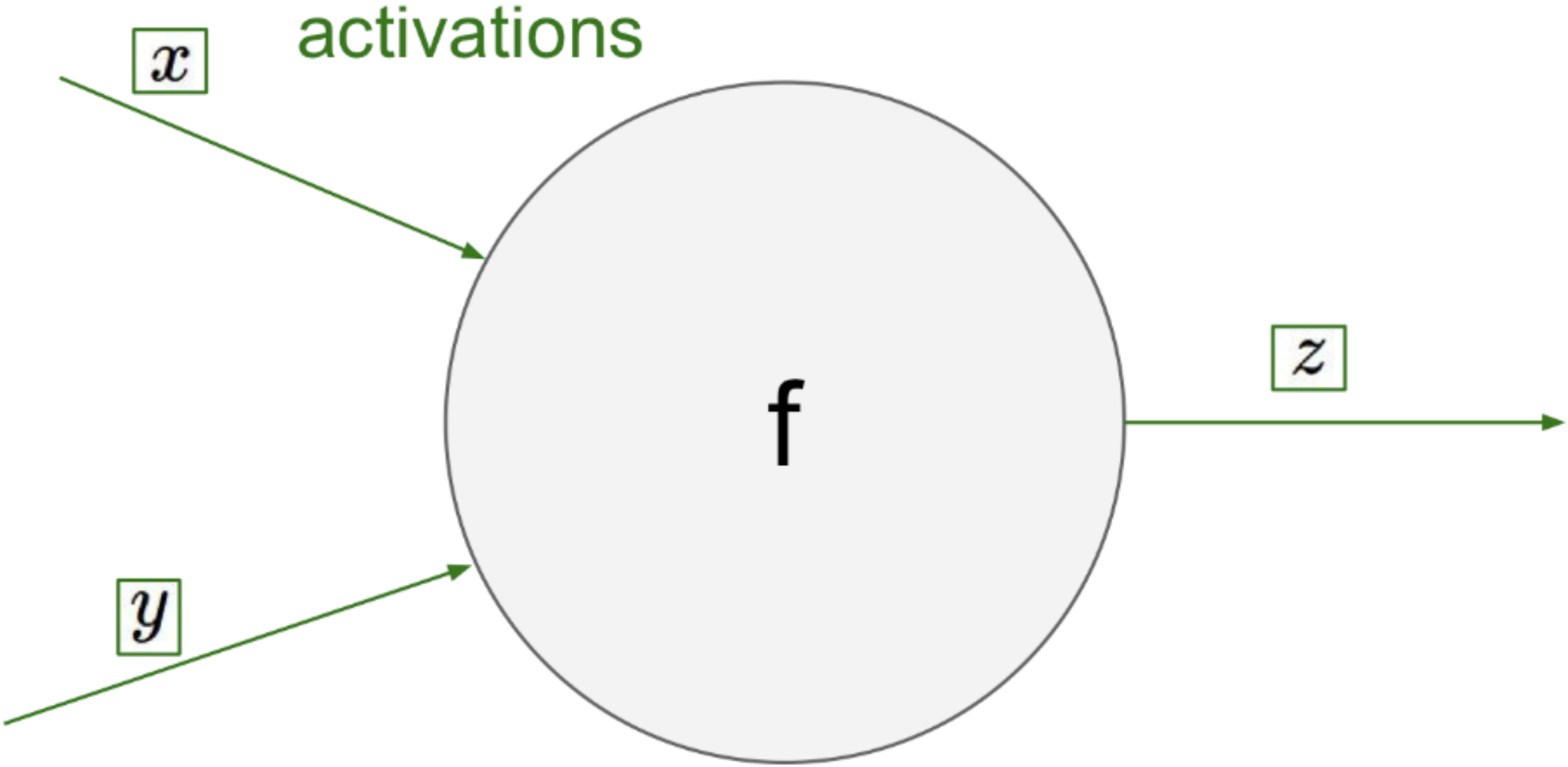
BUT NEEDS THE HESSIAN

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 f}{\partial W_1^2} & \frac{\partial^2 f}{\partial W_1\,\partial W_2} & \cdots & \frac{\partial^2 f}{\partial W_1\,\partial W_n} \\ \frac{\partial^2 f}{\partial W_2\,\partial W_1} & \frac{\partial^2 f}{\partial W_2^2} & \cdots & \frac{\partial^2 f}{\partial W_2\,\partial W_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial W_n\,\partial W_1} & \frac{\partial^2 f}{\partial W_n\,\partial W_2} & \cdots & \frac{\partial^2 f}{\partial W_n^2} \end{bmatrix}$$

NICE, BUT I NEED TO COMPUTE TE GRADIENT AT EVERY ITERATION OF AN ARBITRARY COMPLEX FUNCTION!

BACKPROPAGATION

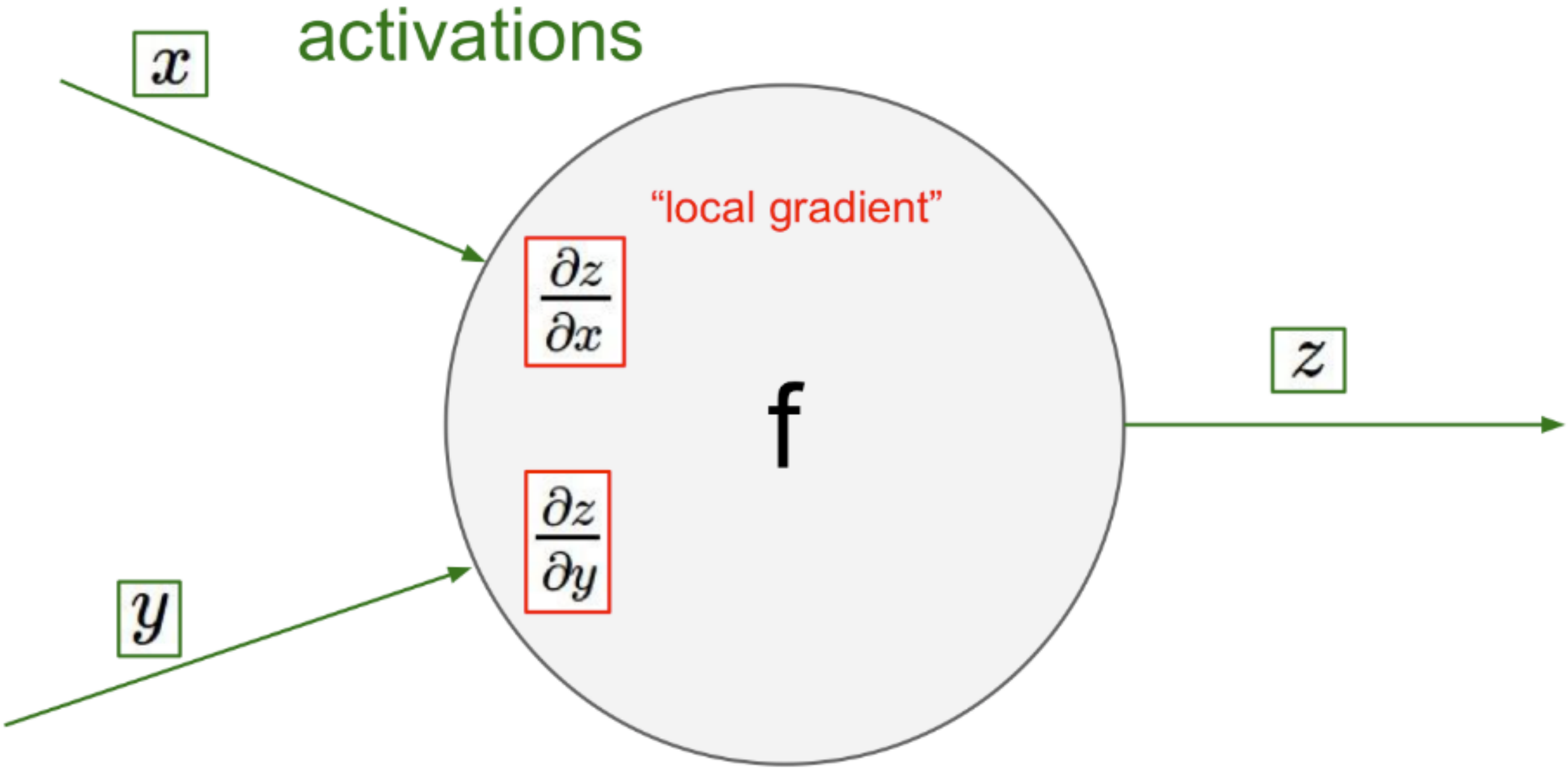
[AT THE NEURON LEVEL]



Credit: A. Karpathy

BACKPROPAGATION

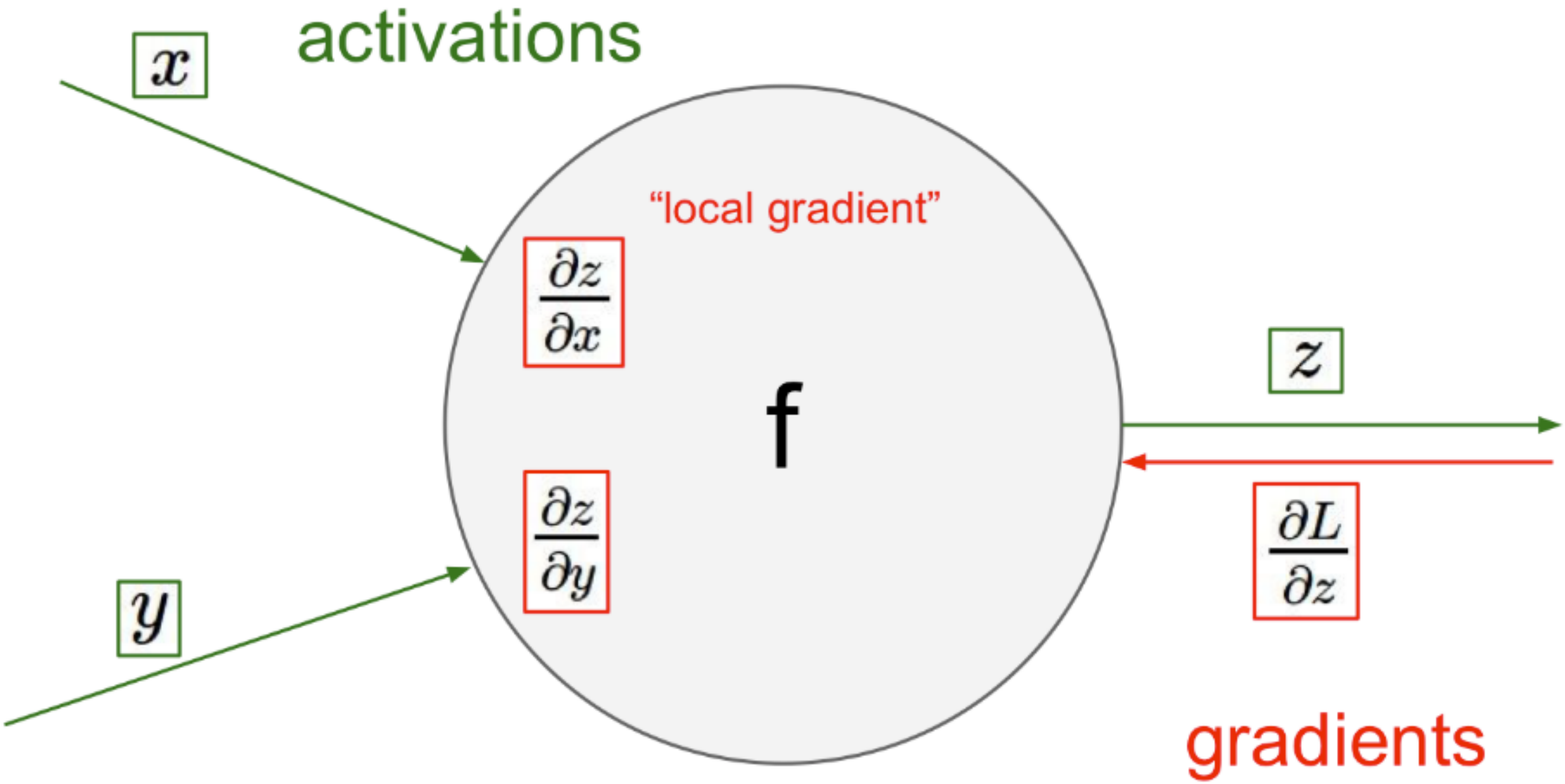
[AT THE NEURON LEVEL]



Credit: A. Karpathy

BACKPROPAGATION

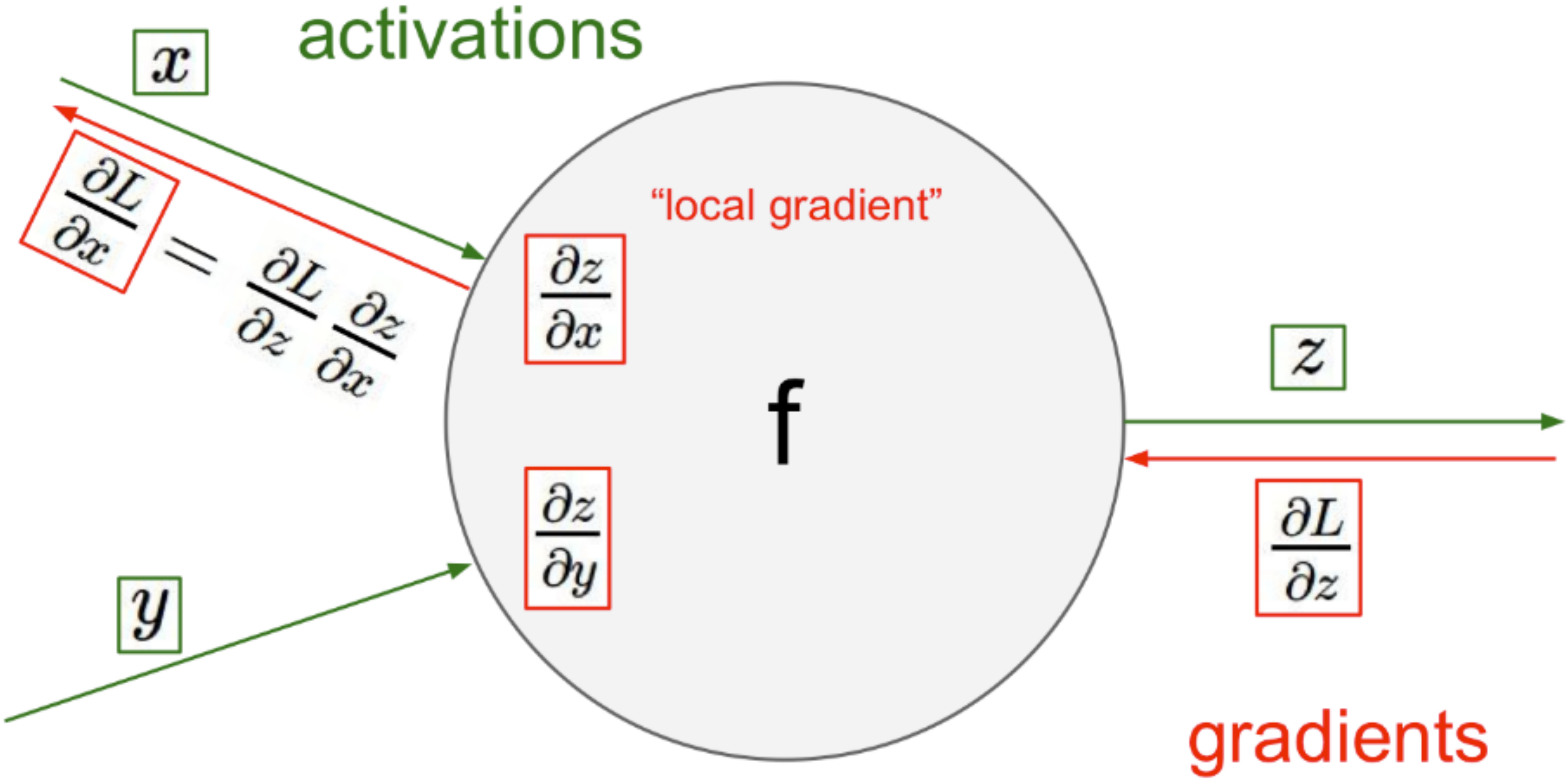
[AT THE NEURON LEVEL]



Credit: A. Karpathy

BACKPROPAGATION

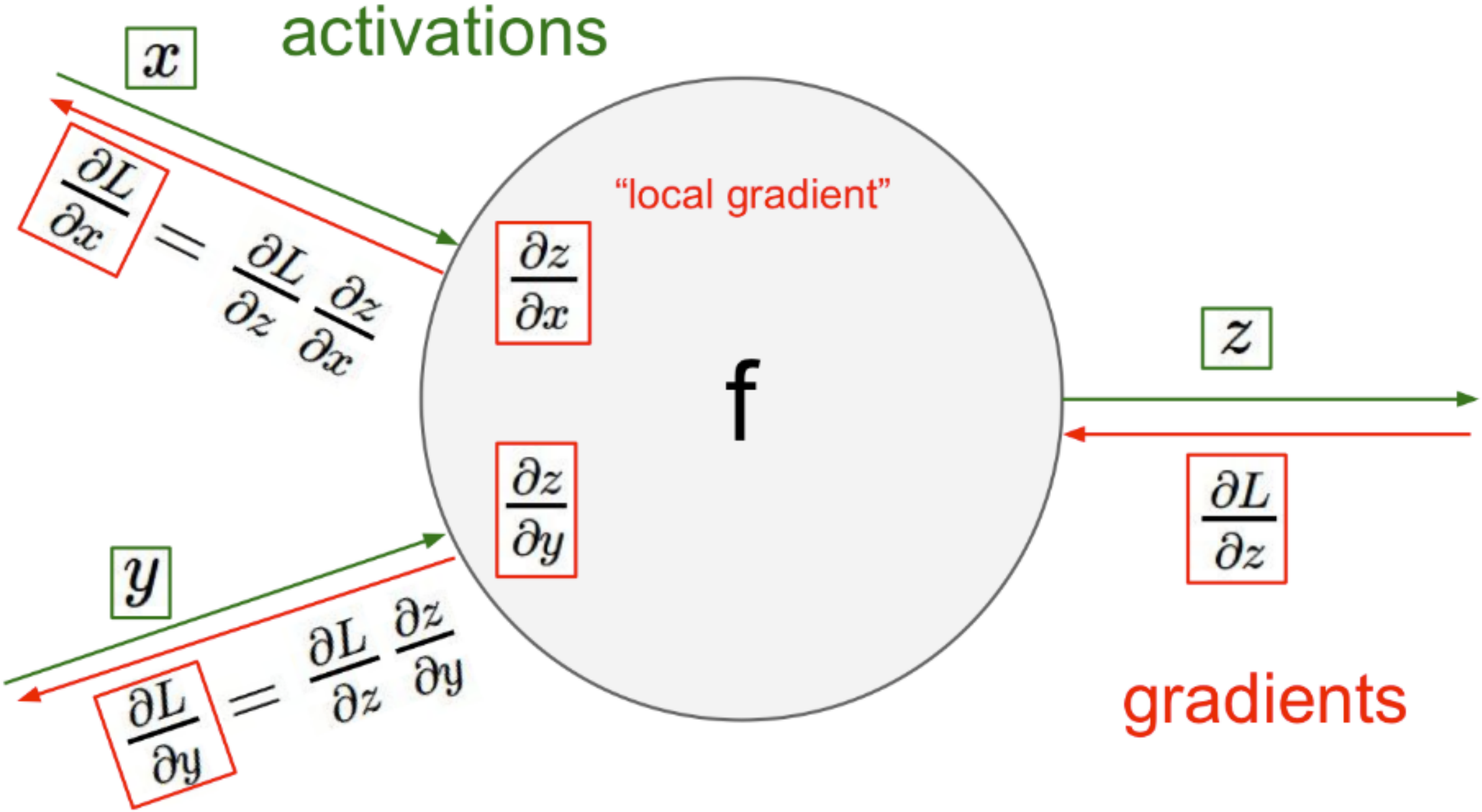
[AT THE NEURON LEVEL]



Credit: A. Karpathy

BACKPROPAGATION

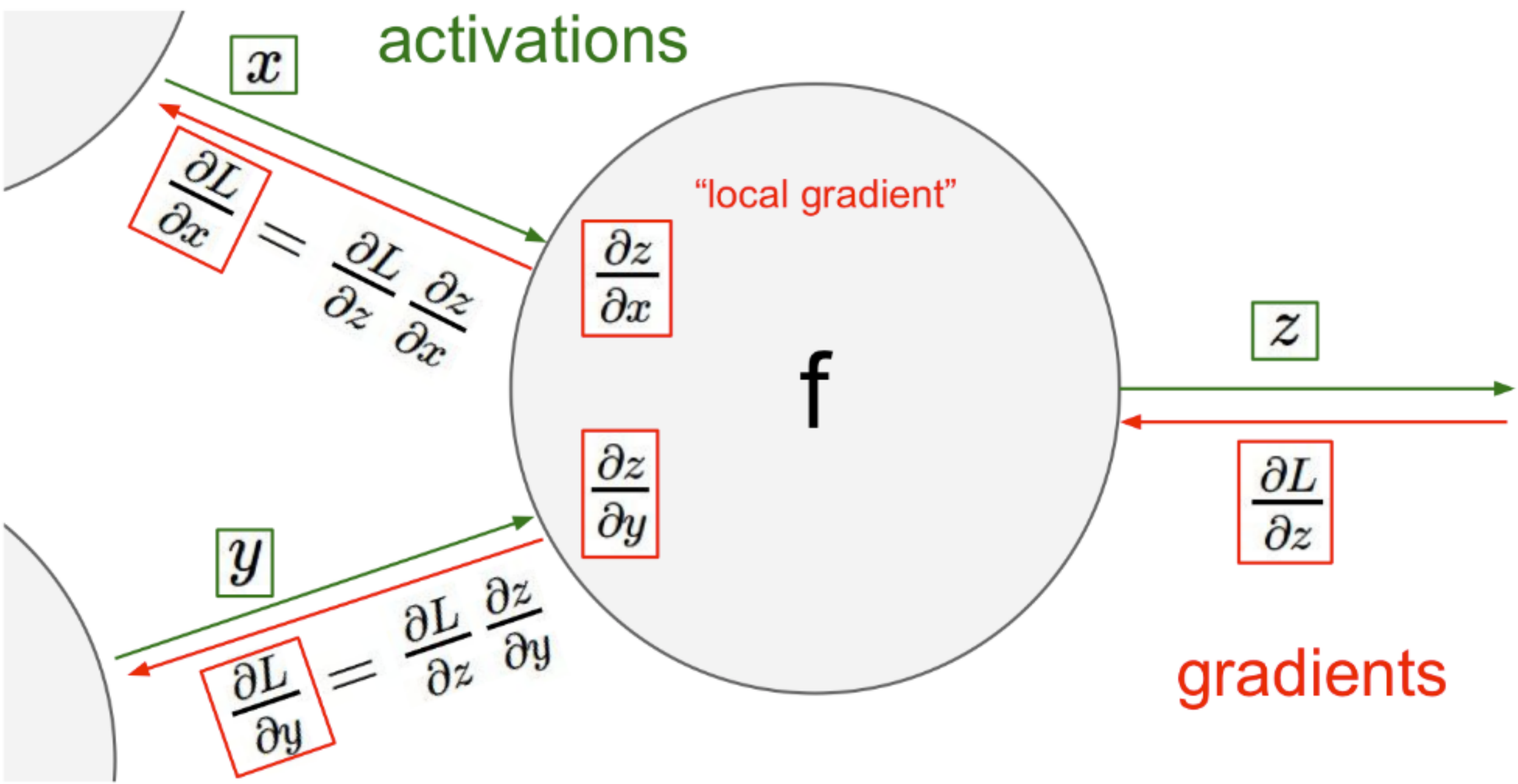
[AT THE NEURON LEVEL]



Credit: A. Karpathy

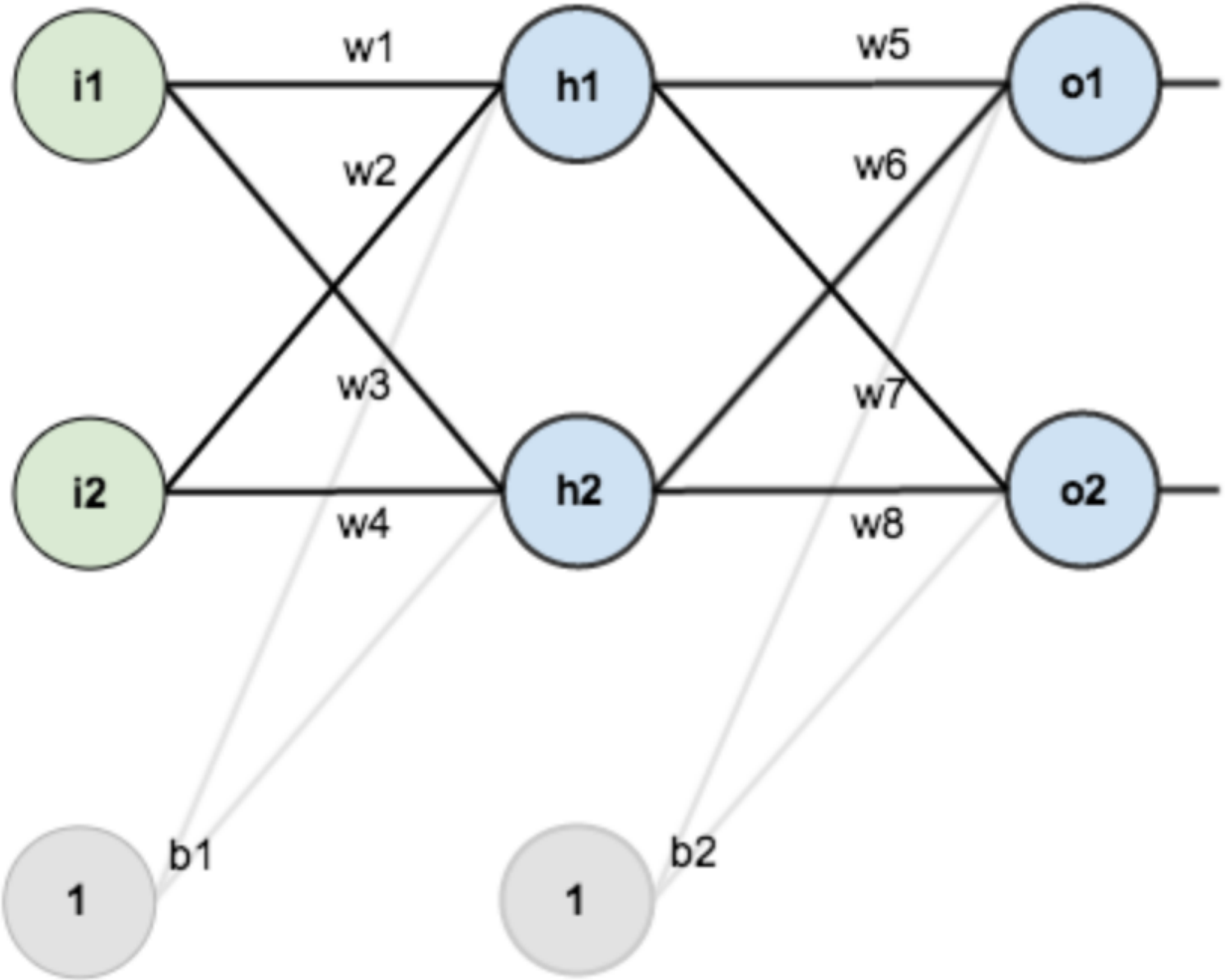
BACKPROPAGATION

[AT THE NEURON LEVEL]

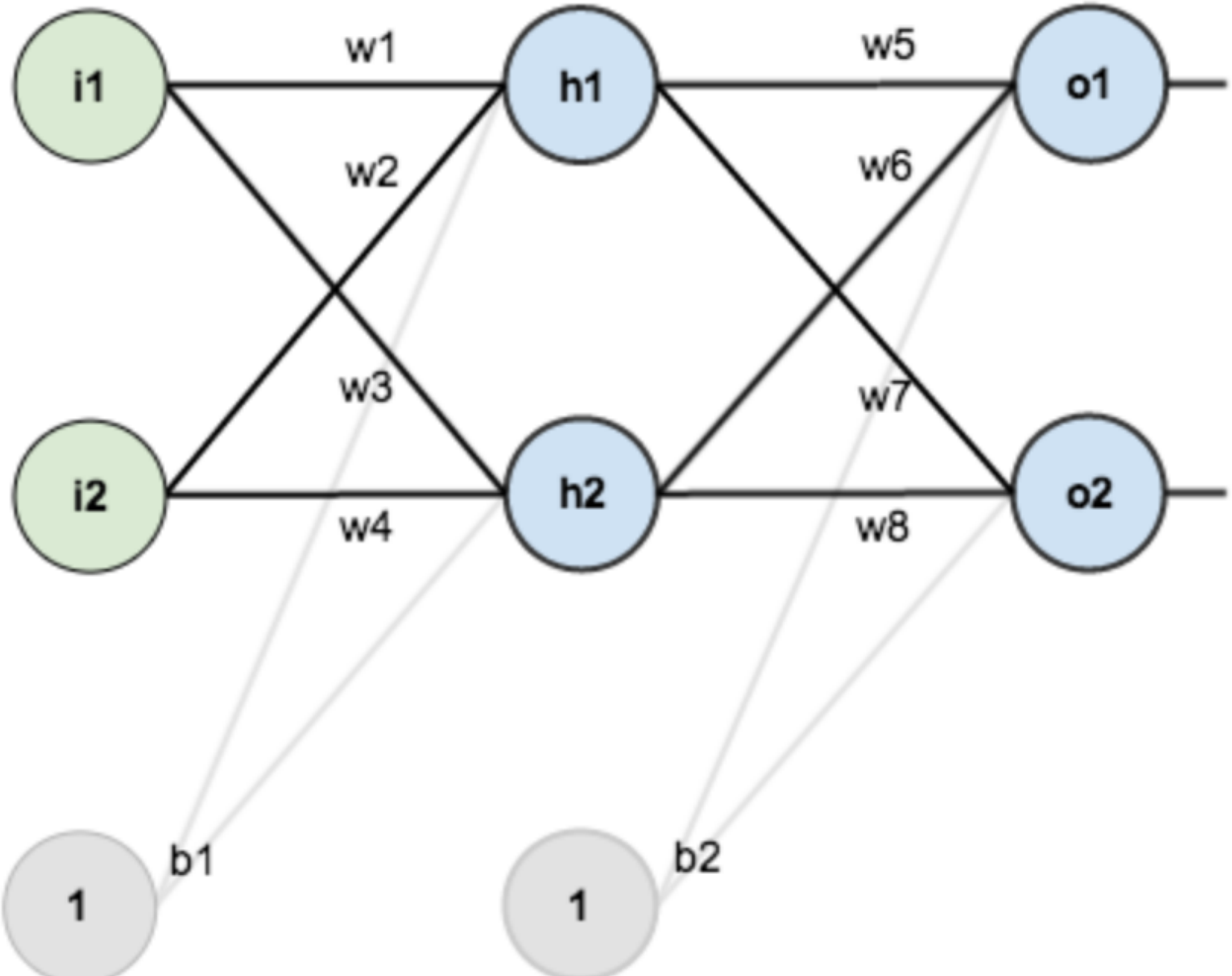


Credit: A. Karpathy

LET’S FOLLOW A NETWORK WHILE IT LEARNS…

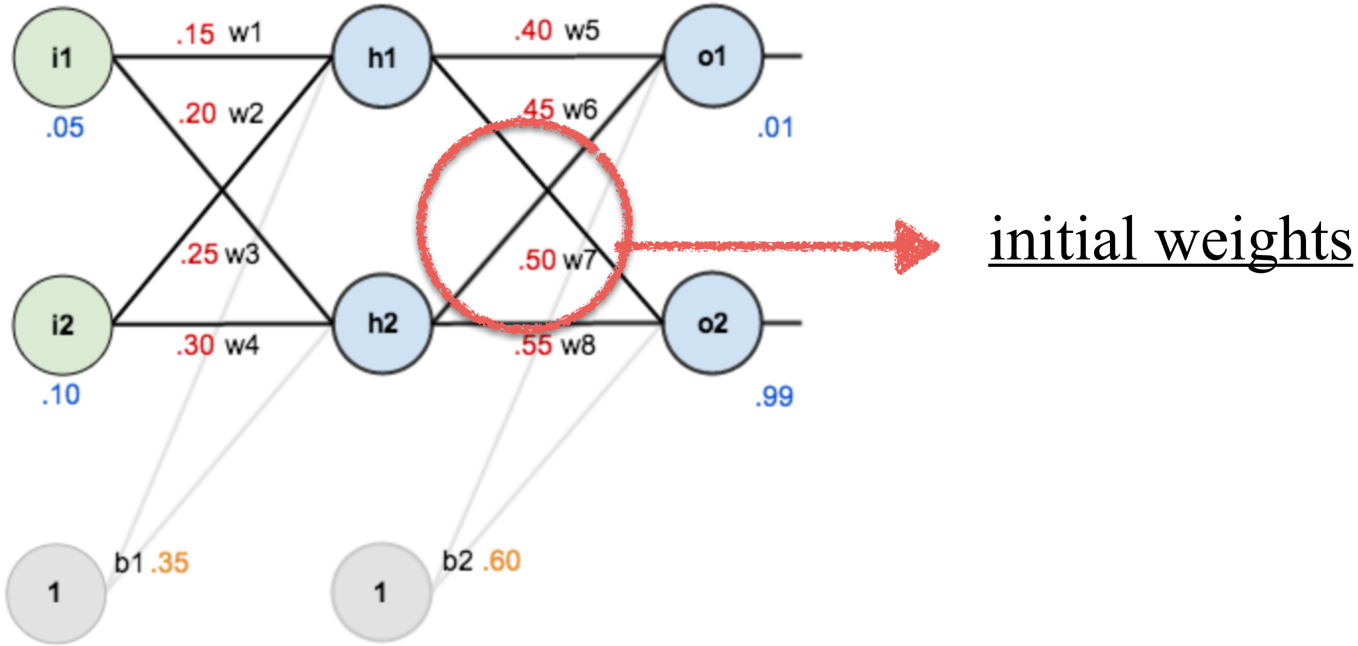


EXAMPLE TAKEN FROM HERE



LET'S ASSUME A VERY SIMPLE TRAINING SET:
X=(0.05, 0.10) —> Y=(0.01,0.99)

EXAMPLE TAKEN FROM HERE

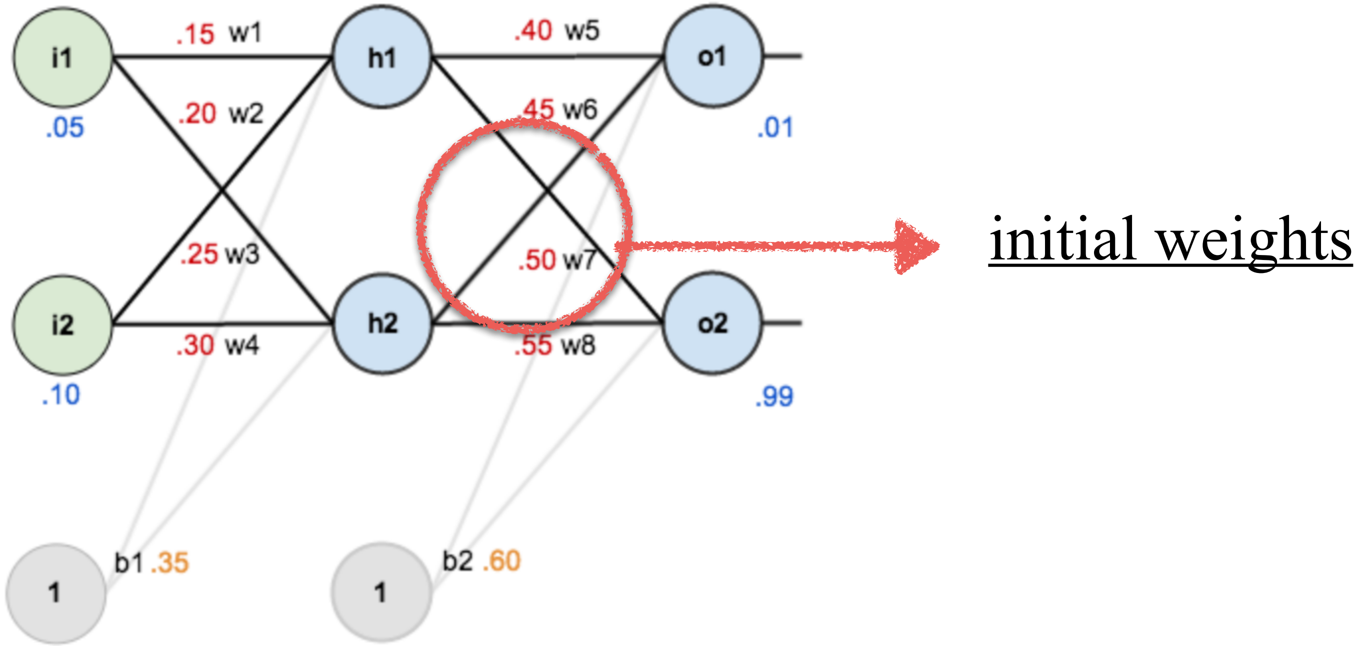


1. THE FORWARD PASS

$$in_{h1} = w_1 i_1 + w_2 i_2 + b_1$$

$$in_{h1} = 0.15 \times 0.05 + 0.2 \times 0.1 + 0.35 = 0.3775$$

[with initial weights]



1. THE FORWARD PASS

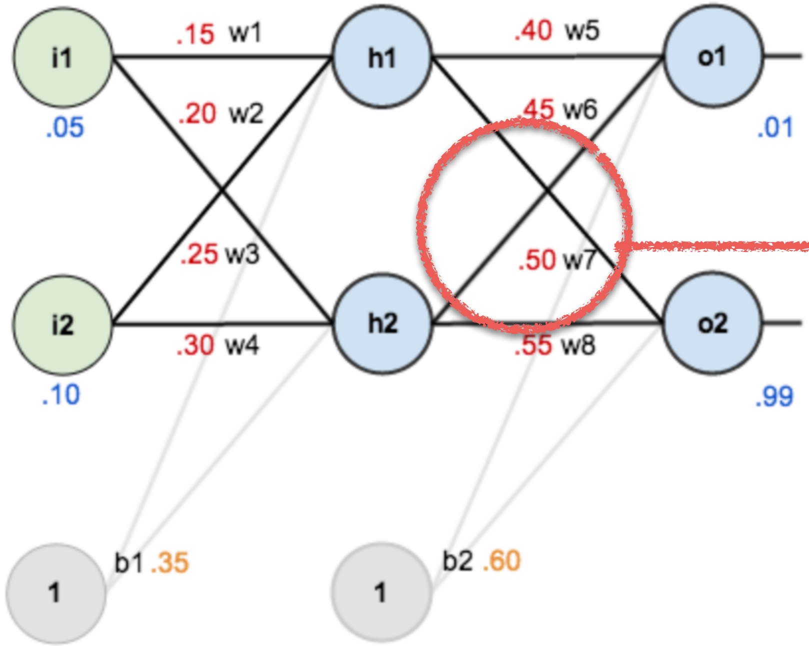
$$in_{h1} = w_1 i_1 + w_2 i_2 + b_1$$

$$in_{h1} = 0.15 \times 0.05 + 0.2 \times 0.1 + 0.35 = 0.3775$$

[with initial weights]

$$out_{h1} = \frac{1}{1 + e^{-in_{h1}}} = 0.5932$$

[after the activation function]



initial weights

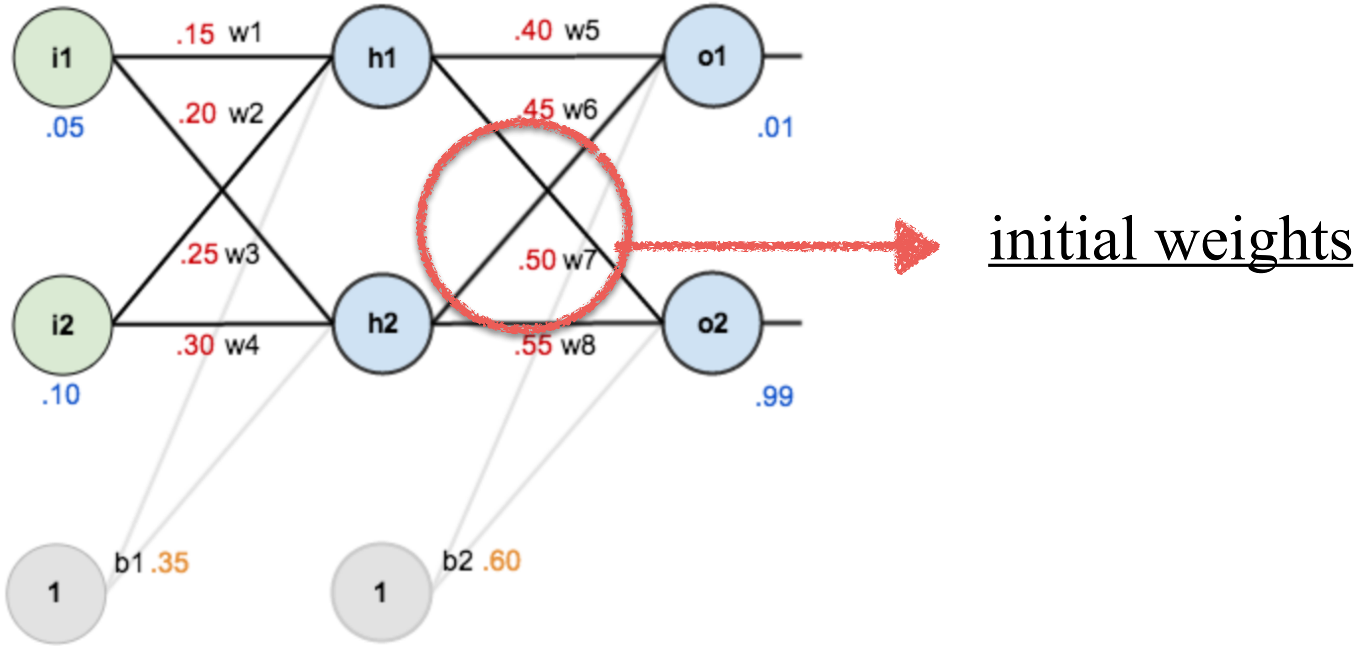
1. THE FORWARD PASS

WE CONTINUE TO o1

$$in_{o1} = w_5 out_{h1} + w_6 out_{h2} + b_2$$

$$in_{o1} = 0.4 \times 0.593 + 0.45 \times 0.596 + 0.6 = 1.105$$

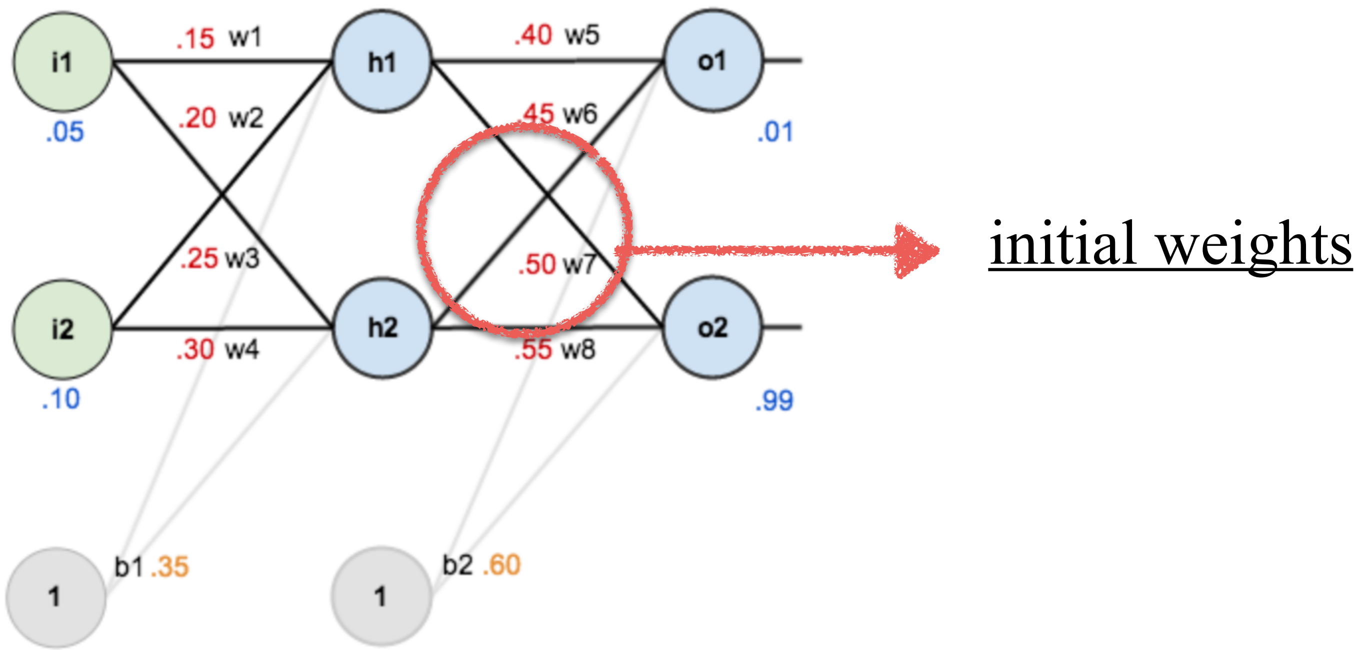
$$out_{o1} = \frac{1}{1 + e^{-1.105}} = 0.751$$



1. THE FORWARD PASS

AND THE SAME FOR o2

$$out_{o2} = 0.7729$$

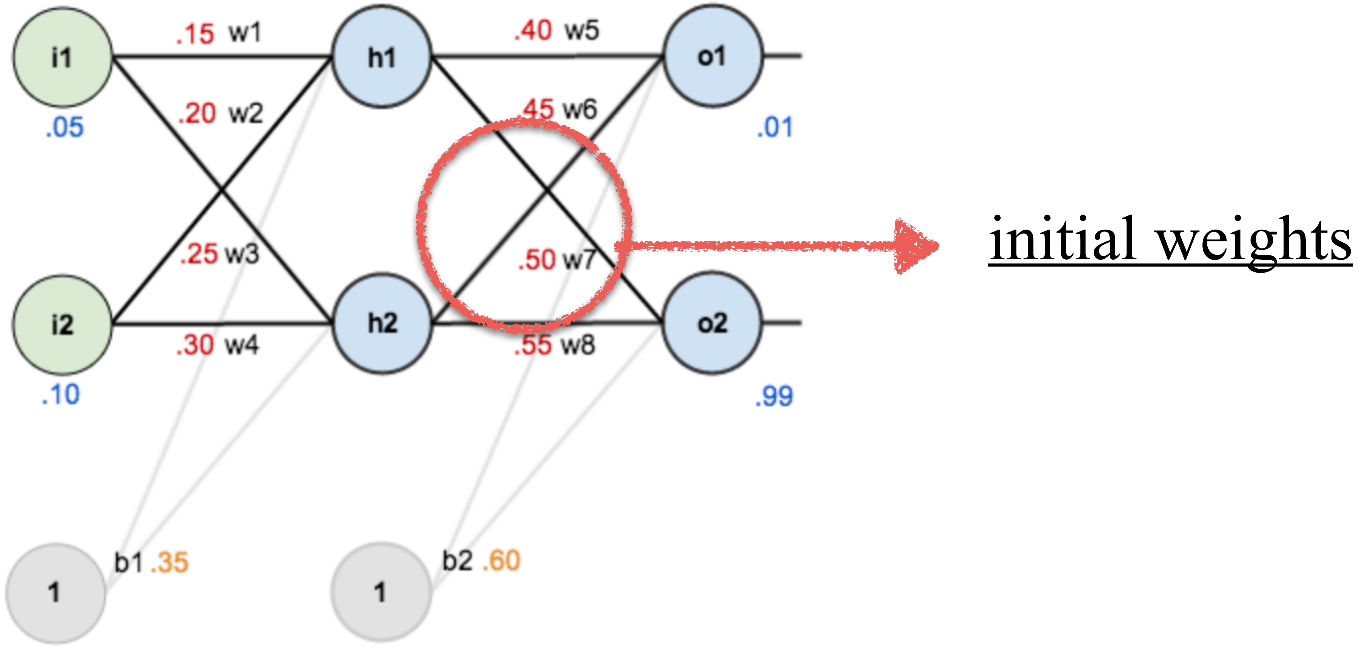


2. THE LOSS FUNCTION

$$L_{total} = \sum 0.5(target - output)^2$$

$$L_{o1} = 0.5(target_{o1} - output_{01})^2 = 0.5 \times (0.01 - 0.751)^2 = 0.274$$

$$L_{o2} = 0.023$$



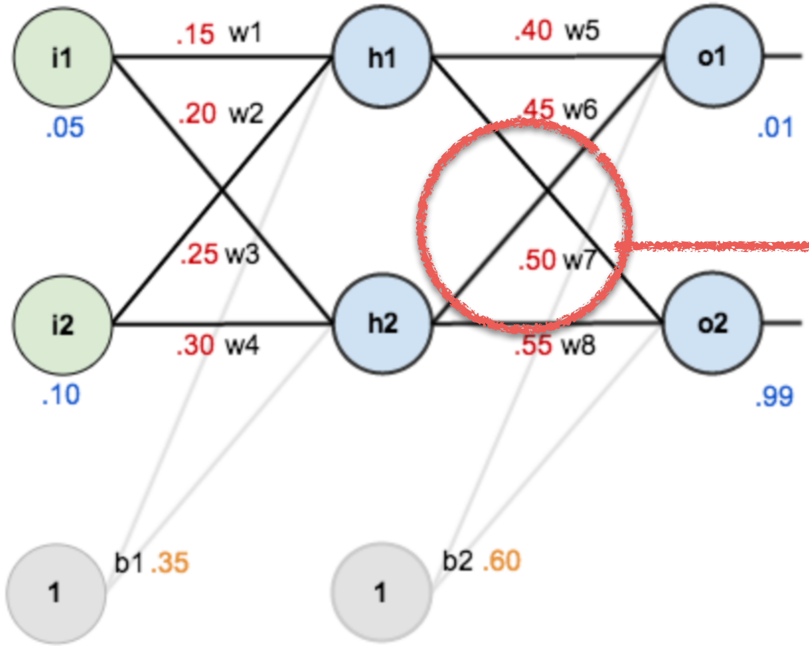
2. THE LOSS FUNCTION

$$L_{total} = \sum 0.5(target - output)^2$$

$$L_{o1} = 0.5(target_{o1} - output_{01})^2 = 0.5 \times (0.01 - 0.751)^2 = 0.274$$

$$L_{o2} = 0.023$$

$$L_{total} = L_{o1} + L_{o2} = 0.298$$



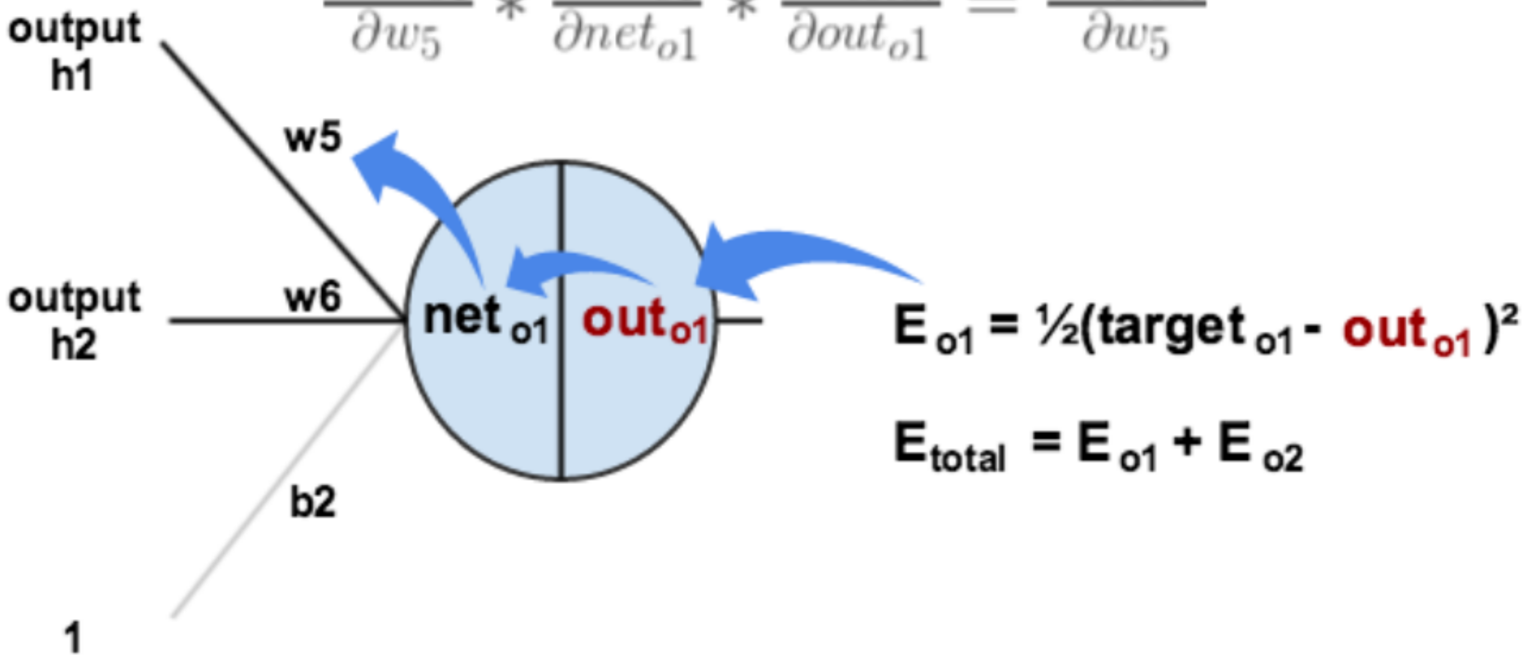
initial weights

3. THE BACKWARD PASS

FOR W5

WE WANT: $$\frac{\partial L_{total}}{\partial w_5}$$ [gradient of loss function]

$$\frac{\partial net_{o1}}{\partial w_5} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial E_{total}}{\partial out_{o1}} = \frac{\partial E_{total}}{\partial w_5}$$



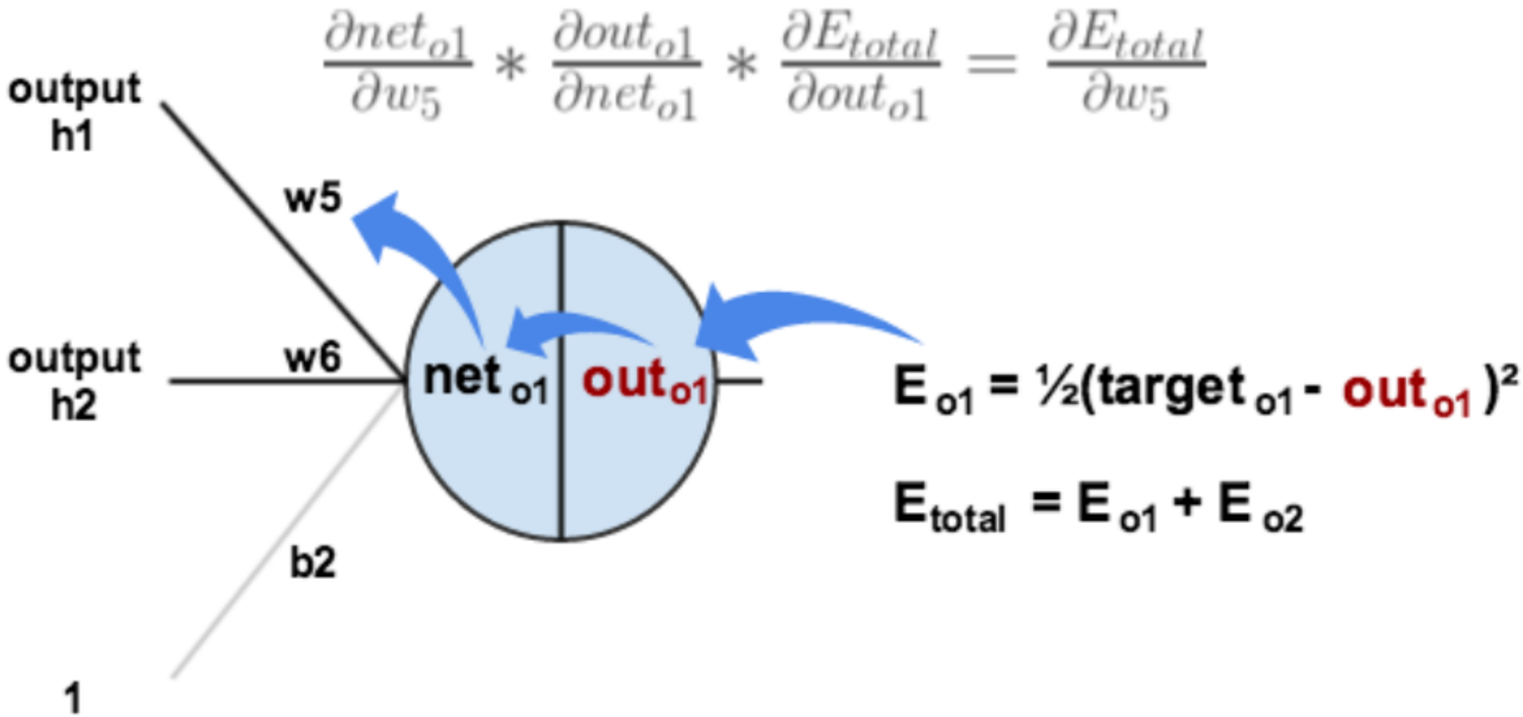
3. THE BACKWARD PASS

FOR W5

WE WANT: $$\frac{\partial L_{total}}{\partial w_5}$$ [gradient of loss function]

WE APPLY THE CHAIN RULE:

$$\frac{\partial L_{total}}{\partial w_5} = \frac{\partial L_{total}}{\partial out_{o1}} \times \frac{\partial out_{o1}}{\partial in_{o1}} \times \frac{\partial in_{o1}}{\partial w_5}$$

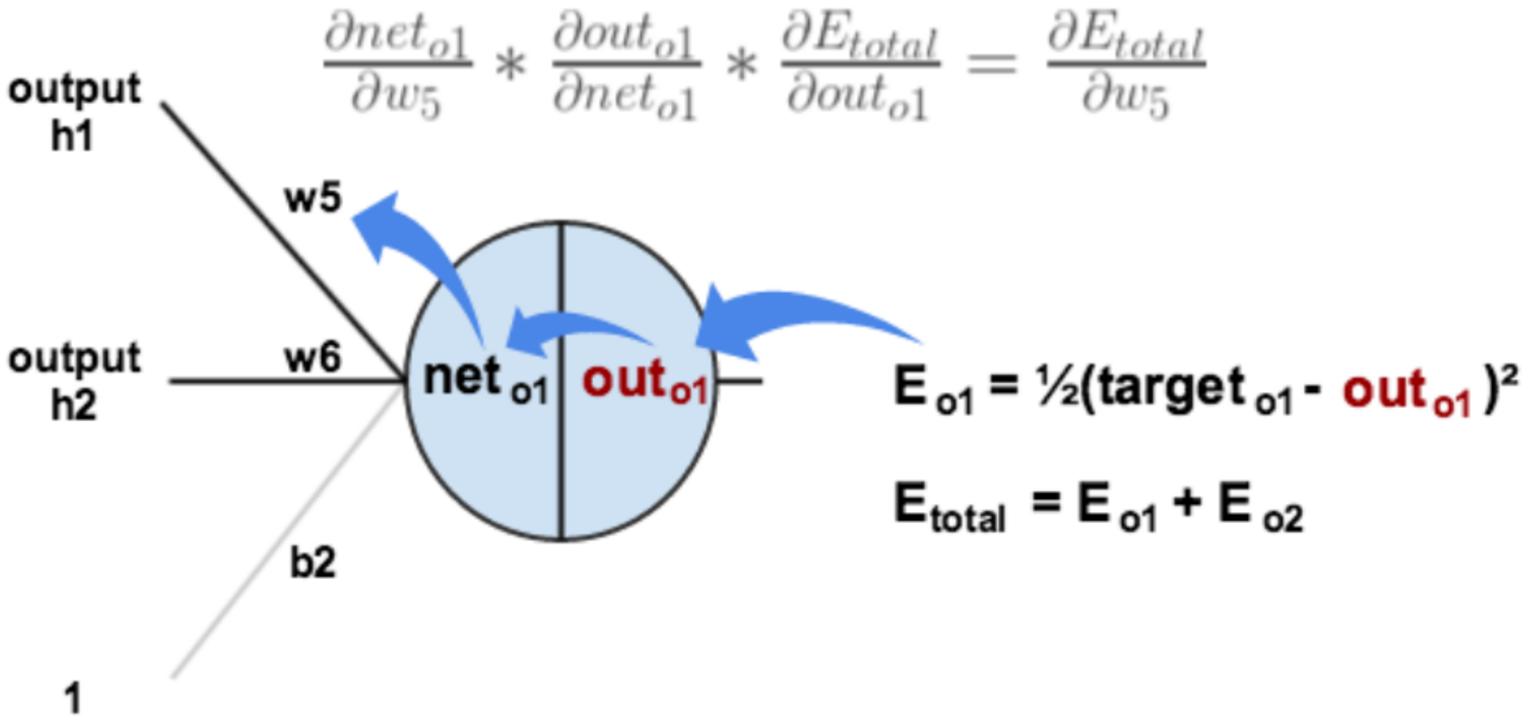


3. THE BACKWARD PASS

$$\frac{\partial L_{total}}{\partial w_5} = \frac{\partial L_{total}}{\partial out_{o1}} \times \frac{\partial out_{o1}}{\partial in_{o1}} \times \frac{\partial in_{o1}}{\partial w_5}$$

$$L_{total} = 0.5(target_{o1} - out_{o1})^2 + 0.5(target_{o2} - out_{o2})^2$$

$$\frac{\partial L_{total}}{\partial out_{o1}} = 2 \times 0.5(target_{o1} - out_{o1}) \times (-1) = 0.741$$

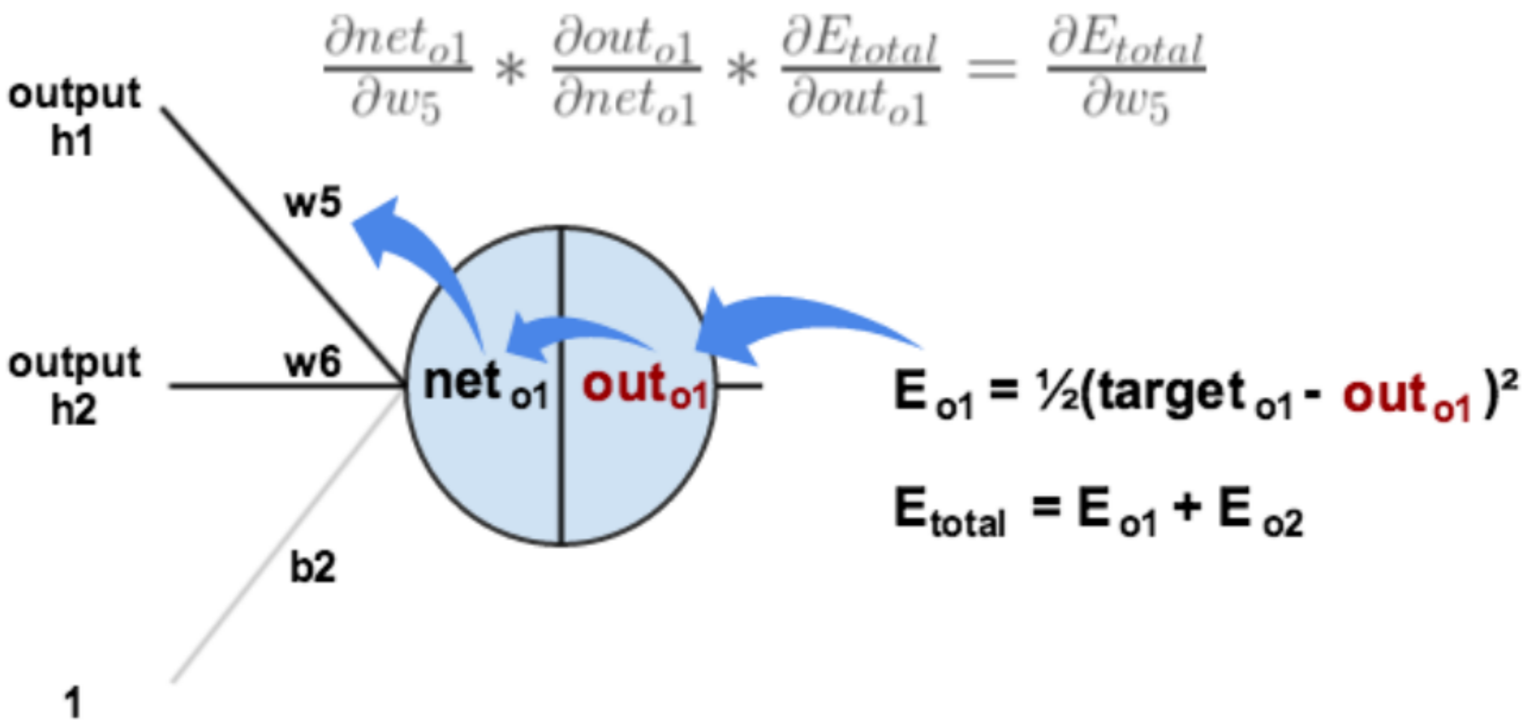


3. THE BACKWARD PASS

$$\frac{\partial L_{total}}{\partial w_5} = \frac{\partial L_{total}}{\partial out_{o1}} \times \frac{\partial out_{o1}}{\partial in_{o1}} \times \frac{\partial in_{o1}}{\partial w_5}$$

$$out_{o1} = \frac{1}{1 + e^{-in_{o1}}}$$

$$\frac{\partial out_{o1}}{\partial in_{o1}} = out_{o1} \times (1 - out_{o1}) = 0.186$$

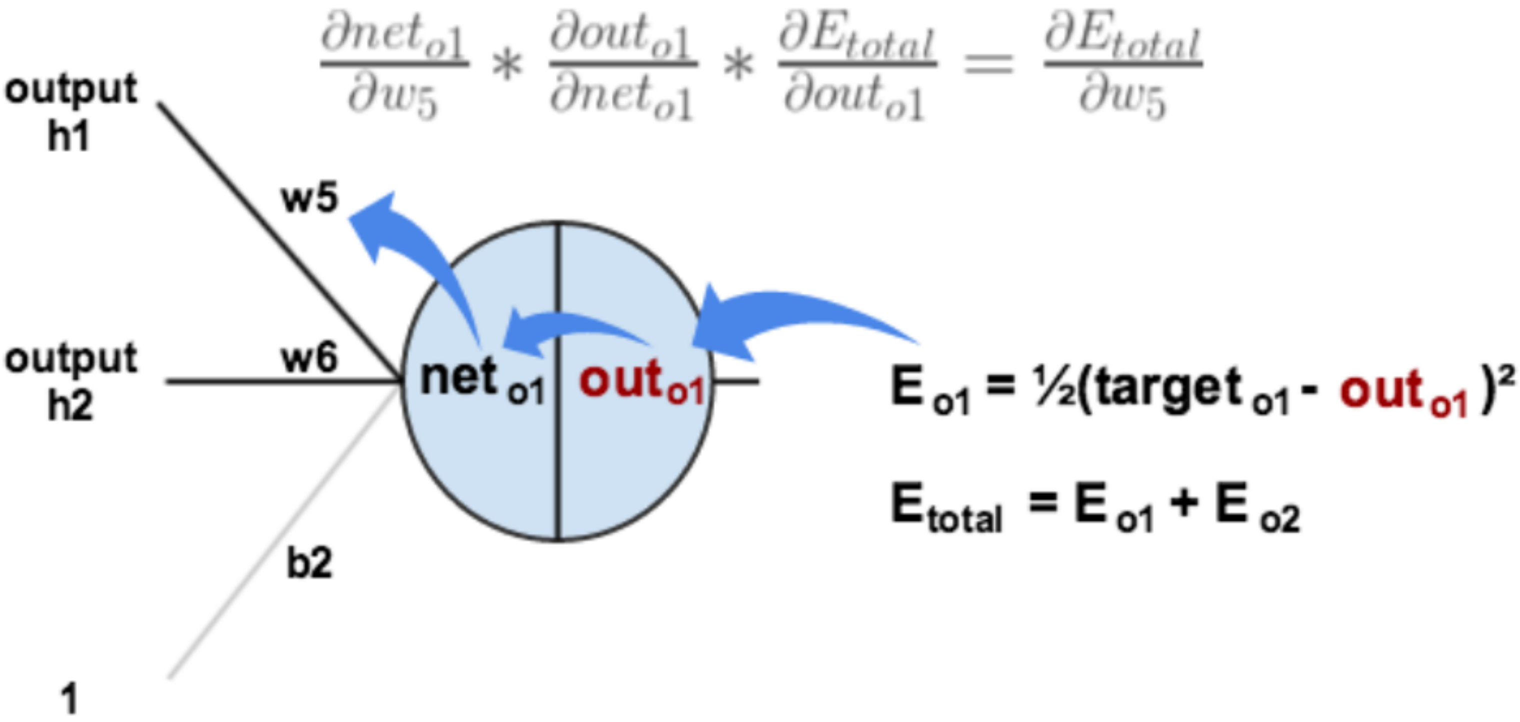


3. THE BACKWARD PASS

$$\frac{\partial L_{total}}{\partial w_5} = \frac{\partial L_{total}}{\partial out_{o1}} \times \frac{\partial out_{o1}}{\partial in_{o1}} \times \frac{\partial in_{o1}}{\partial w_5}$$

$$in_{o1} = w_5 \times out_{h1} + w_6 \times out_{h2} + b_2$$

$$\frac{\partial in_{o1}}{\partial w_5} = out_{h1} \times w_5^{1-1} = out_{h1} = 0.593$$



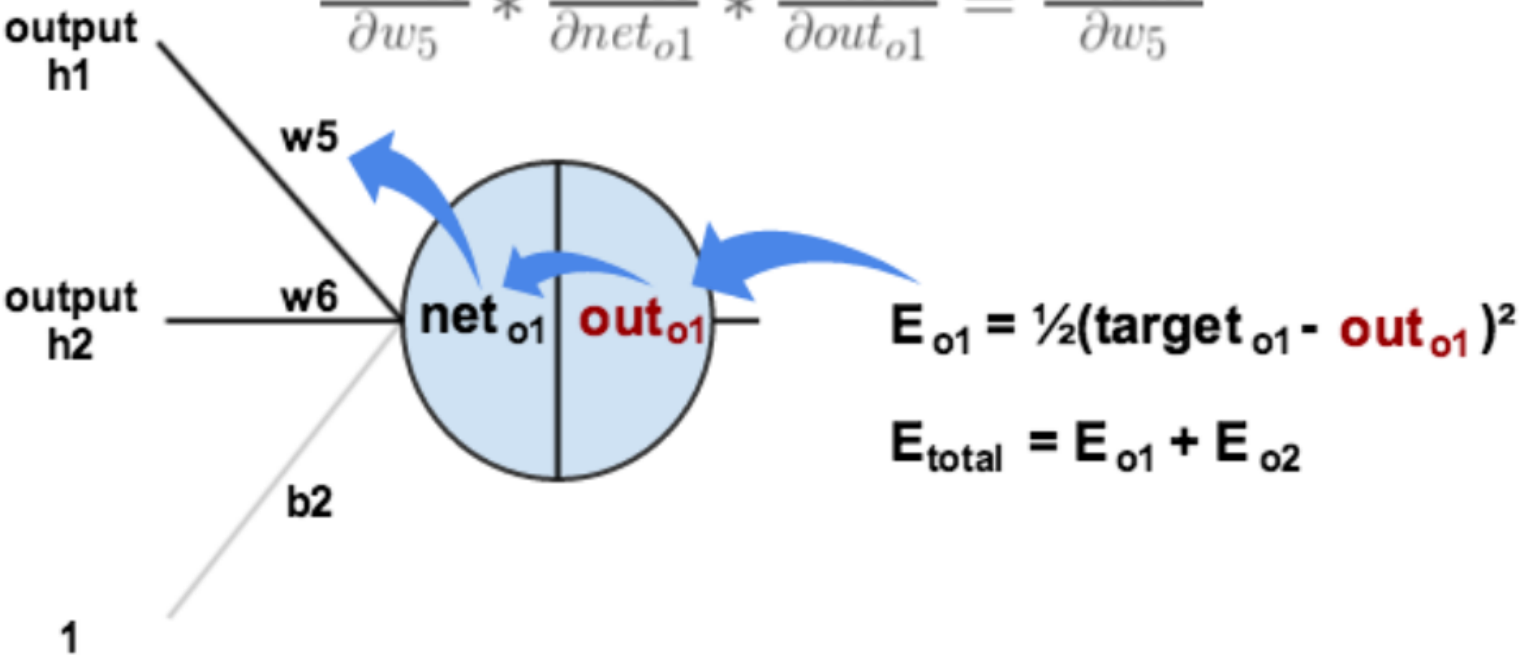
3. THE BACKWARD PASS

ALL TOGETHER:

$$\frac{\partial L_{total}}{\partial w_5} = \frac{\partial L_{total}}{\partial out_{o1}} \times \frac{\partial out_{o1}}{\partial in_{o1}} \times \frac{\partial in_{o1}}{\partial w_5}$$

$$\frac{\partial L_{total}}{\partial w_5} = 0.741 \times 0.186 \times 0.593 = 0.082$$

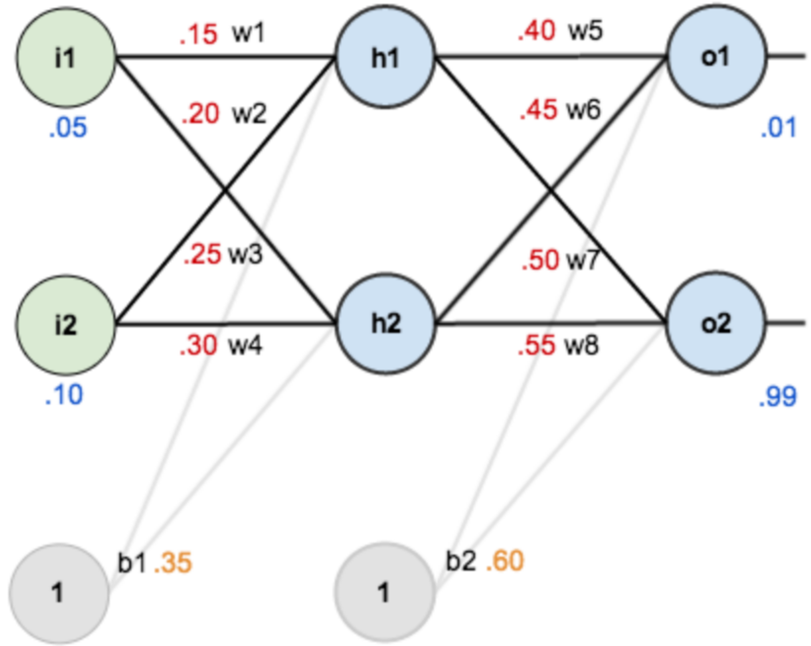
$$\frac{\partial net_{o1}}{\partial w_5} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial E_{total}}{\partial out_{o1}} = \frac{\partial E_{total}}{\partial w_5}$$



4. UPDATE WEIGHTS WITH GRADIENT AND LEARNING RATE

$$w_5^{t+1} = w_5 - \lambda \times \frac{\partial L_{total}}{\partial w_5}$$

$$w_5^{t+1} = 0.4 - 0.5 \times 0.082 = 0.358$$

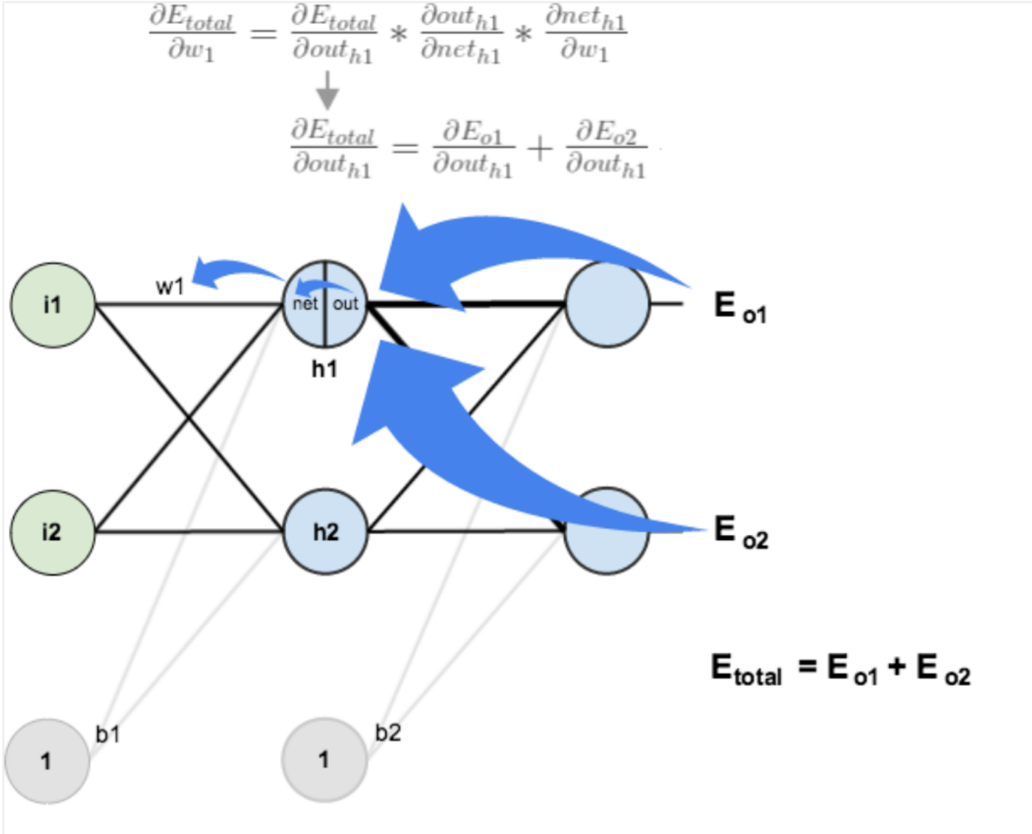


THIS IS REPEATED FOR THE OTHER WEIGHTS OF THE OUTPUT LAYER

$$w_6^{t+1} = 0.408$$

$$w_7^{t+1} = 0.511$$

$$w_8^{t+1} = 0.561$$



AND BACK-PROPAGATED TO THE HIDDEN LAYERS

VISUALIZE SIMPLE NETWORK LEARNING

ONE KEY PROBLEM WITH GRADIENT DESCENT IS THAT IT EASILY CONVERGES TO LOCAL MINIMA BY FOLLOWING THE STEEPEST DESCENT

ONE KEY PROBLEM WITH GRADIENT DESCENT IS THAT IT EASILY CONVERGES TO LOCAL MINIMA BY FOLLOWING THE STEEPEST DESCENT

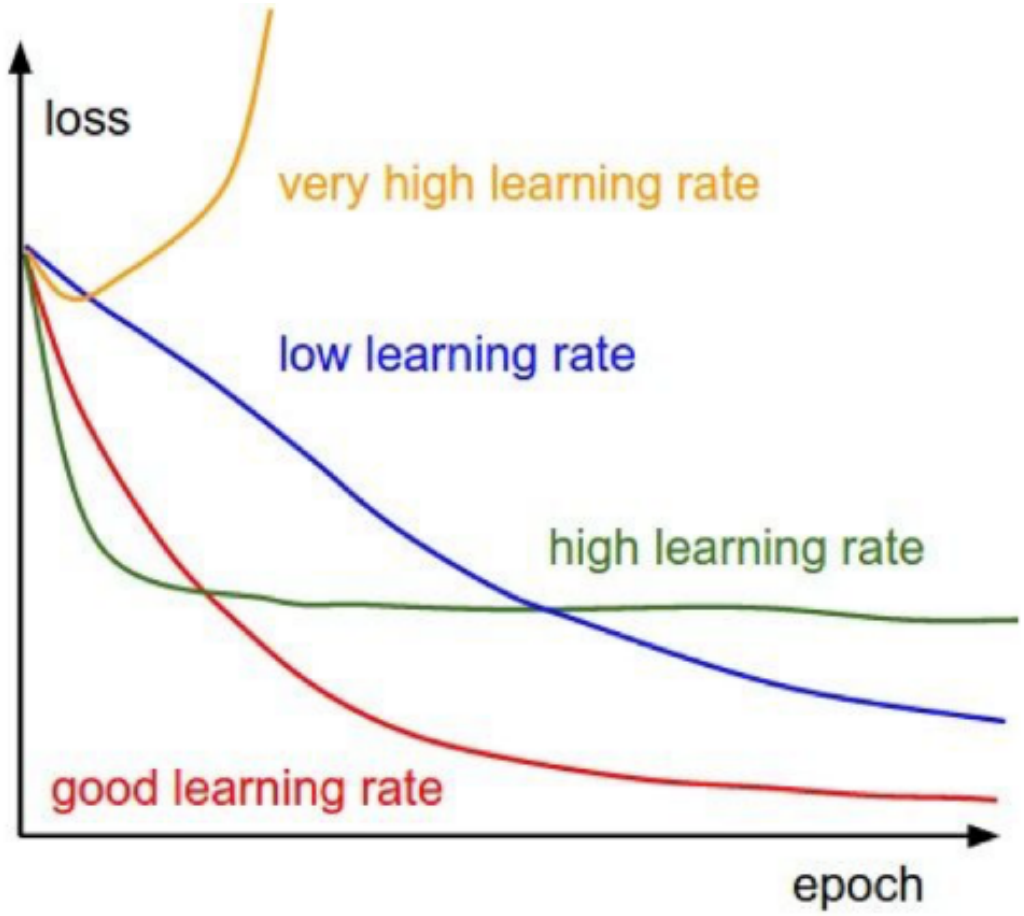
THE CHOICES OF THE INITIAL WEIGHTS AND THE LEARNING RATES ARE IMPORTANT

ONE KEY PROBLEM WITH GRADIENT DESCENT IS THAT IT EASILY CONVERGES TO LOCAL MINIMA BY FOLLOWING THE STEEPEST DESCENT

THE CHOICES OF THE <u>INITIAL WEIGHTS</u> AND THE <u>LEARNING RATES</u> ARE IMPORTANT

WE WILL TALK ABOUT THIS LATER

LEARNING RATES



Credit:

LEARNING RATES

$$W_{t+1} = W_t - \boxed{\lambda} \nabla f(W_t)$$

THERE ARE DIFFERENT WAYS
TO UPDATE THE LEARNING RATE

Credit:

LEARNING RATES

$$W_{t+1} = W_t - \lambda \nabla f(W_t)$$

THERE ARE DIFFERENT WAYS
TO UPDATE THE LEARNING RATE

ADAGRAD:

THE LEARNING RATE IS SCALED DEPENDING ON THE HISTORY OF PREVIOUS GRADIENTS

$$W_{t+1} = W_t - \frac{\lambda}{\sqrt{G_t + \epsilon}} \nabla f(W_t)$$

G IS A MATRIX CONTAINING ALL PREVIOUS GRADIENTS. WHEN THE GRADIENT BECOMES LARGE THE LEARNING RATE IS DECREASED AND VICE VERSA.

$$G_{t+1} = G_t + (\nabla f)^2$$

Credit:

LEARNING RATES

$$W_{t+1} = W_t - \lambda \nabla f(W_t)$$

THERE ARE DIFFERENT WAYS
TO UPDATE THE LEARNING RATE

RMSPROP:

THE LEARNING RATE IS SCALED DEPENDING ON THE HISTORY OF PREVIOUS GRADIENTS

$$W_{t+1} = W_t - \frac{\lambda}{\sqrt{G_t + \epsilon}} \nabla f(W_t)$$

SAME AS **ADAGRAD** BUT G IS CALCULATED BY EXPONENTIALLY DECAYING AVERAGE

$$G_{t+1} = \lambda G_t + (1 - \lambda)(\nabla f)^2$$

Credit:

ADAM [Adaptive moment estimator]:

SAME IDEA, USING FIRST AND SECOND ORDER MOMENTUMS

$$G_{t+1} = \beta_2 G_t + (1-\beta_2)(\nabla f)^2 \quad M_{t+1} = \beta_1 M_t + (1-\beta_1)(\nabla f)$$

$$W_{t+1} = W_t - \frac{\lambda}{\sqrt{\hat{G}_t + \epsilon}} \hat{M}_t$$

with: $\hat{M}_{t+1} = \dfrac{M_t}{1-\beta_1} \qquad \hat{G}_{t+1} = \dfrac{G_t}{1-\beta_2}$

<u>ADAM [Adaptive moment estimator]:</u>

SAME IDEA, USING FIRST AND SECOND ORDER MOMENTUMS

$$G_{t+1} = \beta_2 G_t + (1-\beta_2)(\nabla f)^2 \qquad M_{t+1} = \beta_1 M_t + (1-\beta_1)(\nabla f)$$

$$W \qquad \sqrt{G_t} + \epsilon$$

with: $$\hat{M}_{t+1} = \frac{M_t}{1-\beta_1} \qquad \hat{G}_{t+1} = \frac{G_t}{1-\beta_2}$$

IN KERAS:

RMSprop

[source]

```
keras.optimizers.RMSprop(lr=0.001, rho=0.9, epsilon=None, decay=0.0)
```

RMSProp optimizer.

It is recommended to leave the parameters of this optimizer at their default values (except the learning rate, which can be freely tuned).

This optimizer is usually a good choice for recurrent neural networks.

Arguments

- **lr**: float >= 0. Learning rate.
- **rho**: float >= 0.
- **epsilon**: float >= 0. Fuzz factor. If `None`, defaults to `K.epsilon()`.
- **decay**: float >= 0. Learning rate decay over each update.

References

- rmsprop: Divide the gradient by a running average of its recent magnitude

IN KERAS:

Adam

[source]

```
keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False)
```

Adam optimizer.

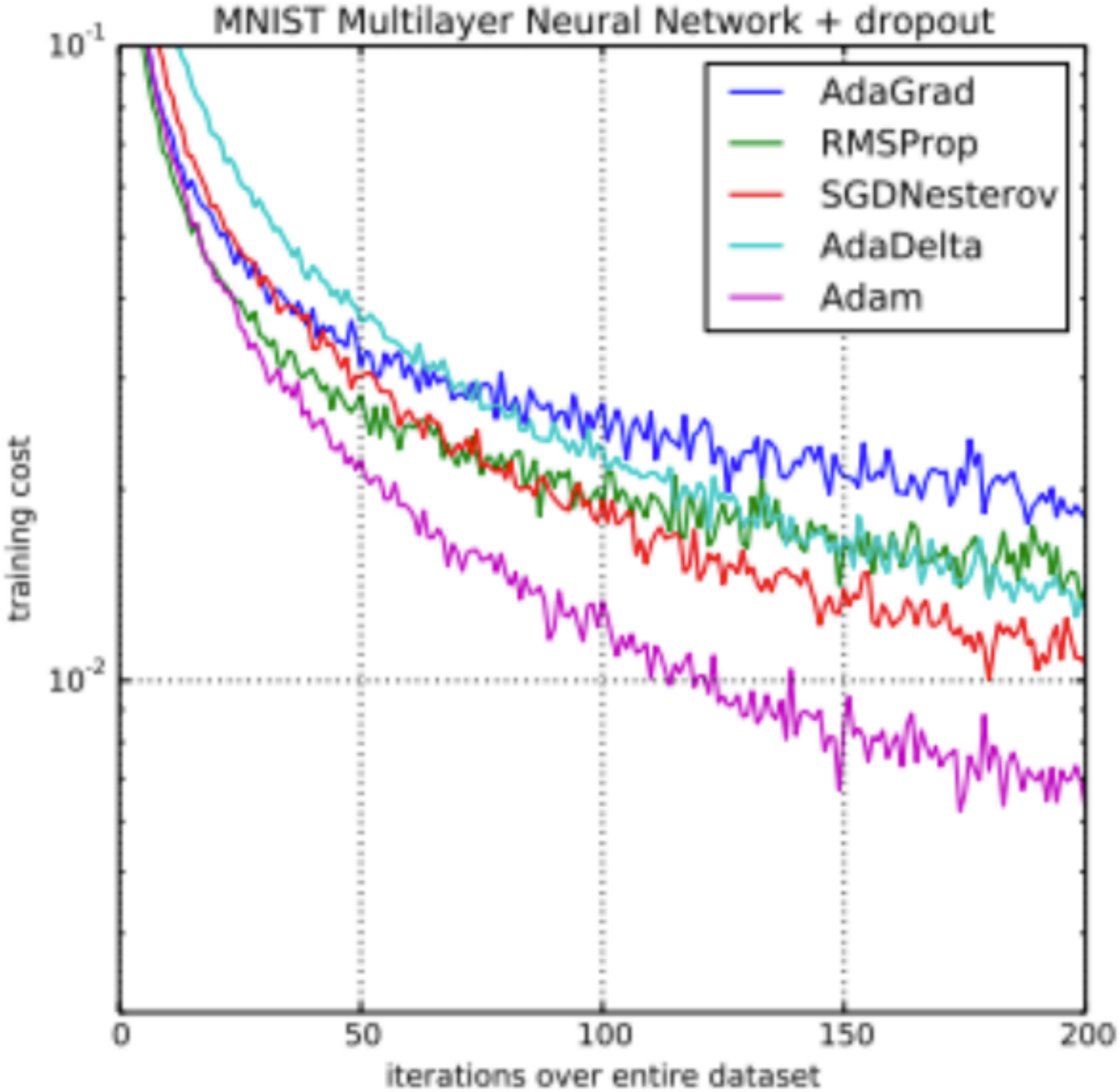
Default parameters follow those provided in the original paper.

Arguments

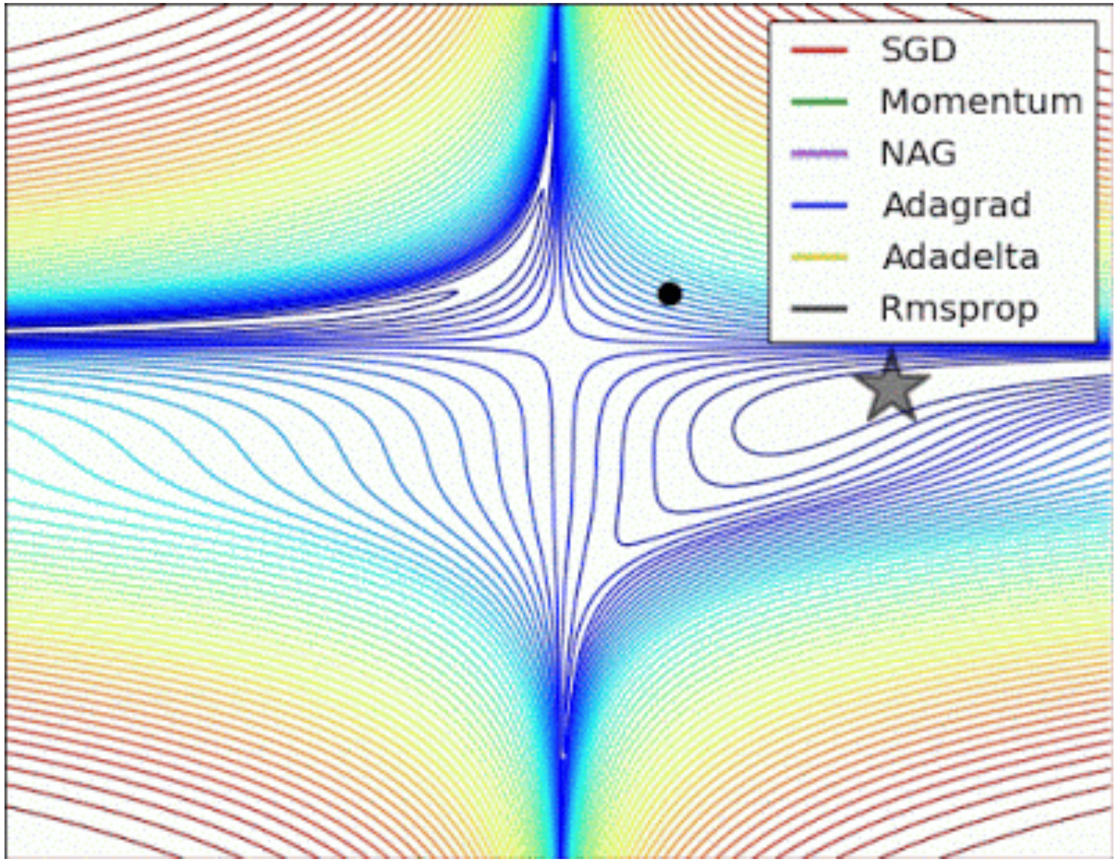
- **lr**: float >= 0. Learning rate.
- **beta_1**: float, 0 < beta < 1. Generally close to 1.
- **beta_2**: float, 0 < beta < 1. Generally close to 1.
- **epsilon**: float >= 0. Fuzz factor. If `None`, defaults to `K.epsilon()`.
- **decay**: float >= 0. Learning rate decay over each update.
- **amsgrad**: boolean. Whether to apply the AMSGrad variant of this algorithm from the paper "On the Convergence of Adam and Beyond".

References

- Adam - A Method for Stochastic Optimization
- On the Convergence of Adam and Beyond


MNIST Multilayer Neural Network + dropout
10^{-1}
10^{-2}
training cost
0
50
100
150
200
iterations over entire dataset
AdaGrad
RMSProp
SGDNesterov
AdaDelta
Adam

Credit



Credit

BATCH GRADIENT DESCENT

LOCAL MINIMA CAN ALSO BE AVOIDED BY COMPUTING THE GRADIENT IN SMALL BATCHES INSTEAD OF OVER THE FULL DATASET

BATCH GRADIENT DESCENT

LOCAL MINIMA CAN ALSO BE AVOIDED BY COMPUTING THE GRADIENT IN SMALL BATCHES INSTEAD OF OVER THE FULL DATASET

MINI-BATCH GRADIENT DESCENT

$$V_{t+1/num} = W_t - \lambda_t \nabla f(W_t; x^{(i,i+b)}, y^{(i,i+b)})$$

THE GRADIENT IS COMPUTED OVER A BATCH OF SIZE B

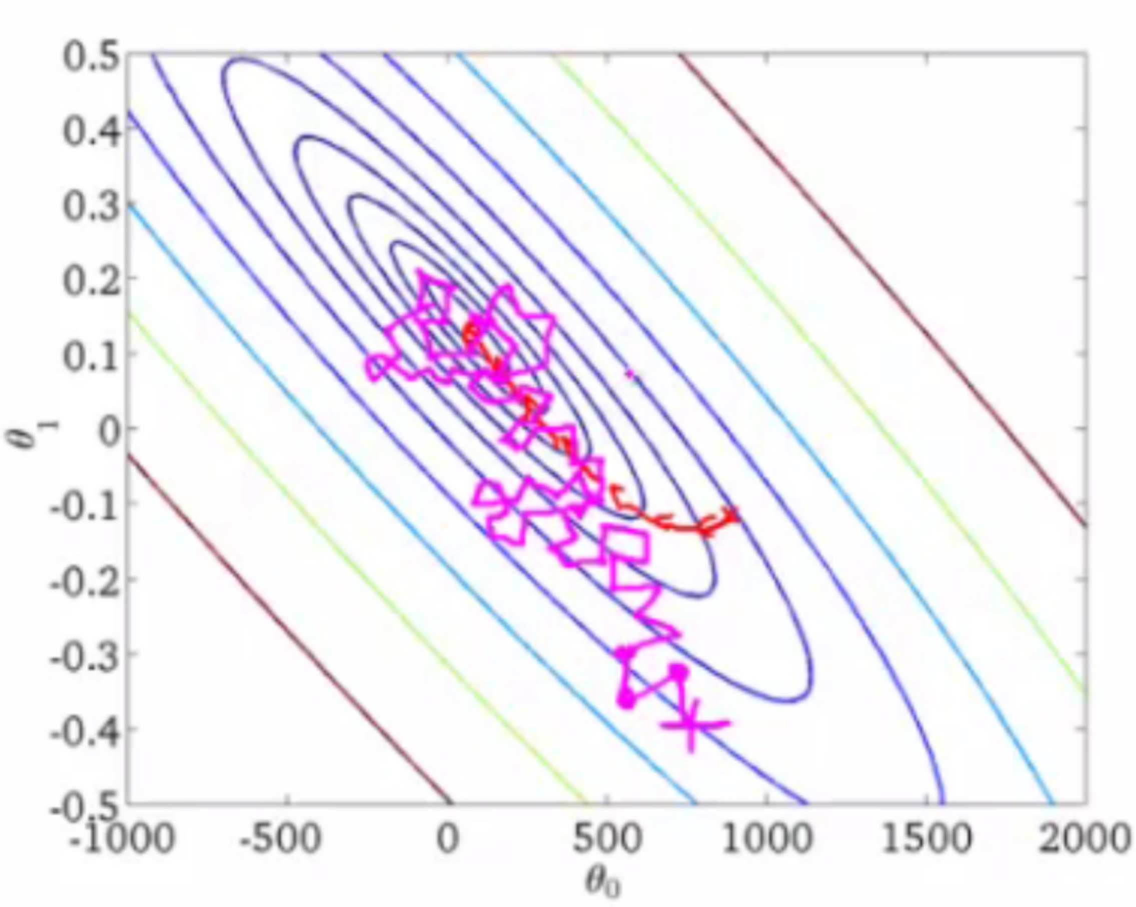
STOCHASTIC GRADIENT DESCENT

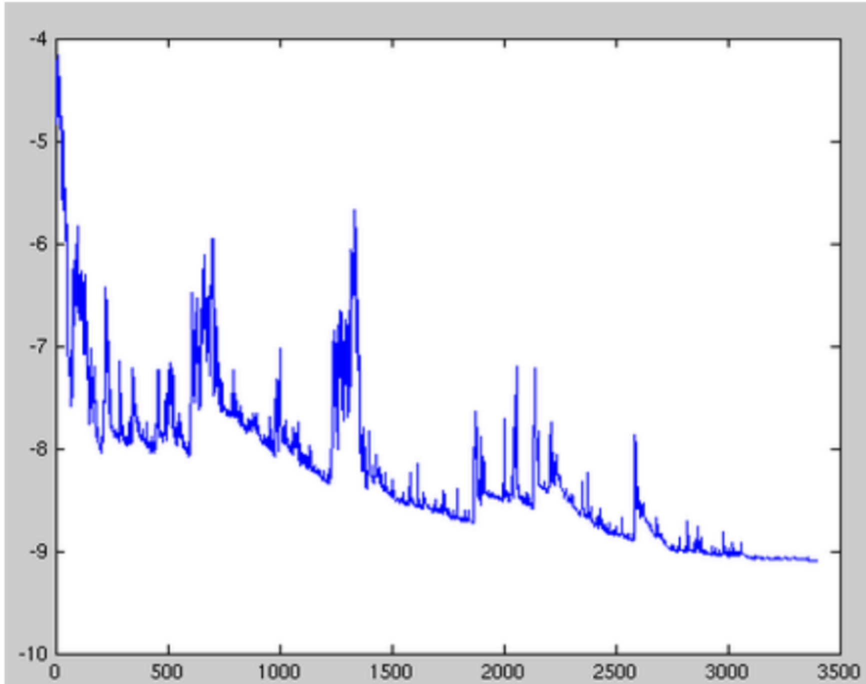
THE EXTREME CASE IS TO COMPUTE THE GRADIENT ON EVERY TRAINING EXAMPLE.

STOCHASTIC GRADIENT DESCENT

$$V_{t+1/num} = W_t - \lambda_t \nabla f(W_t; x^{(i,i+b)}, y^{(i,i+b)})$$

b=1





Fluctuations in the total objective function as gradient steps with respect to mini-batches are taken.

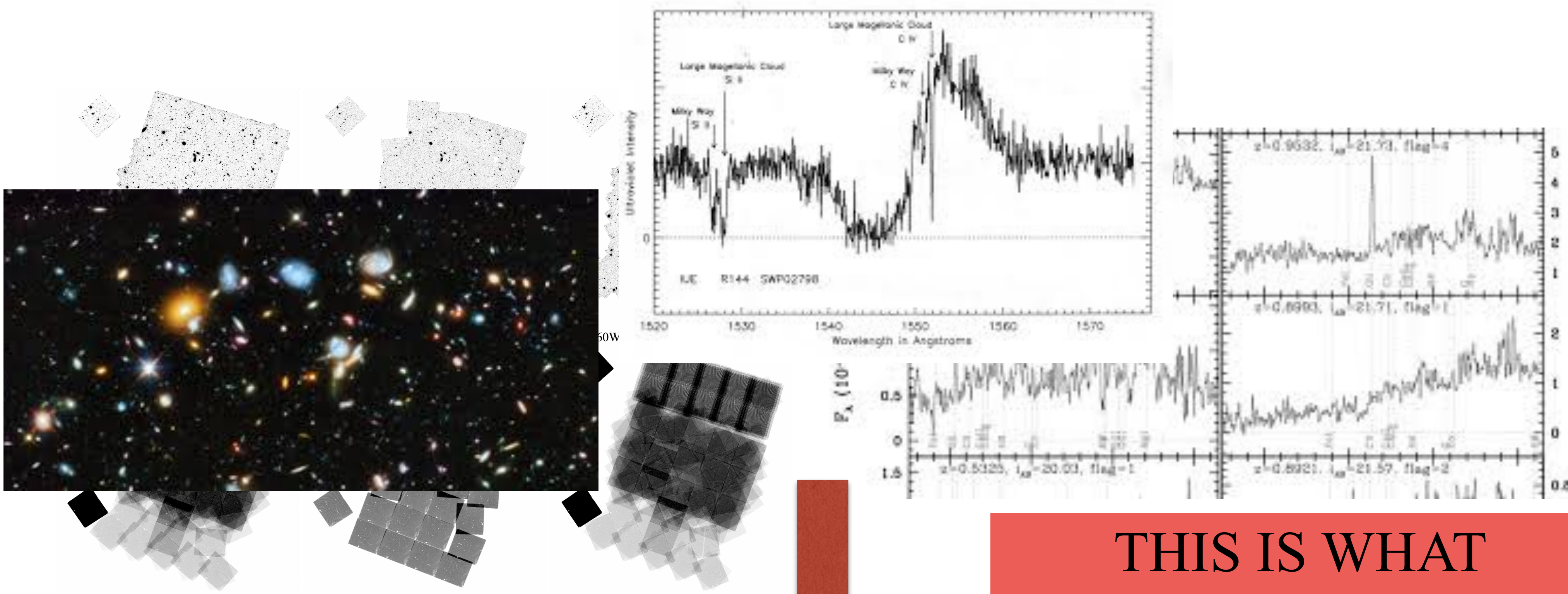
Credit

CAN WE GO DEEP NOW?

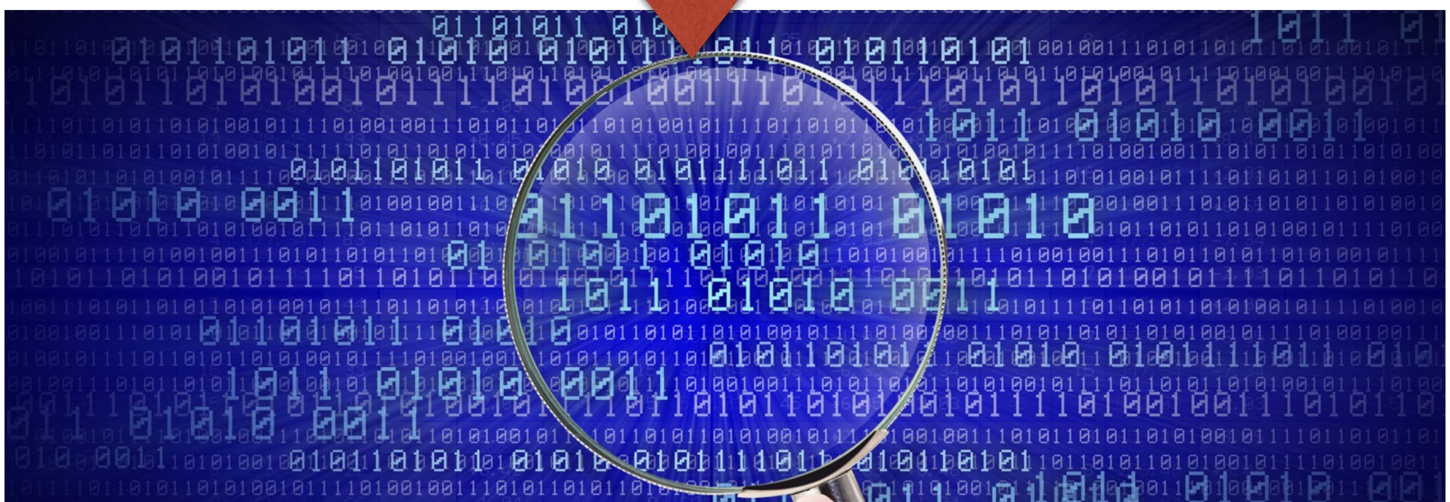
CAN WE GO DEEP NOW?

ALMOST THERE…LET'S THINK FOR A MOMENT ABOUT WHAT WE PUT AS INPUT…

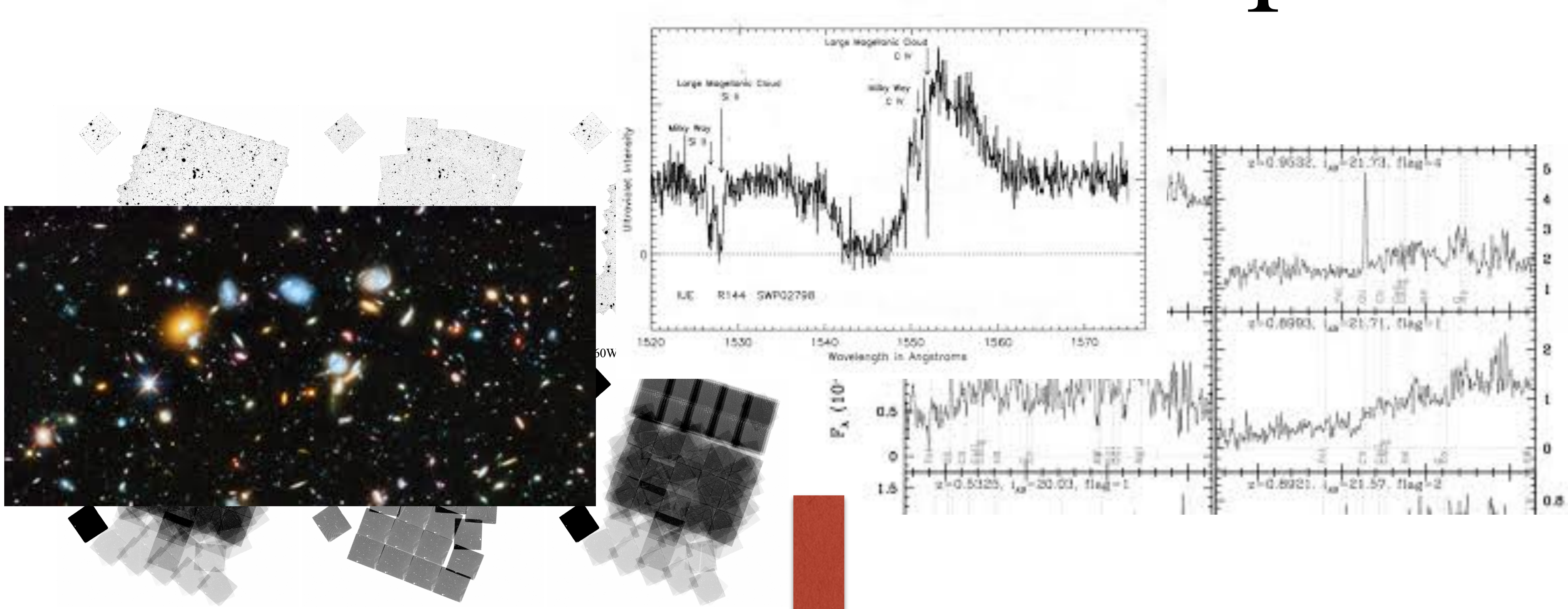
What do we put as input?



THIS IS WHAT MACHINES SEE

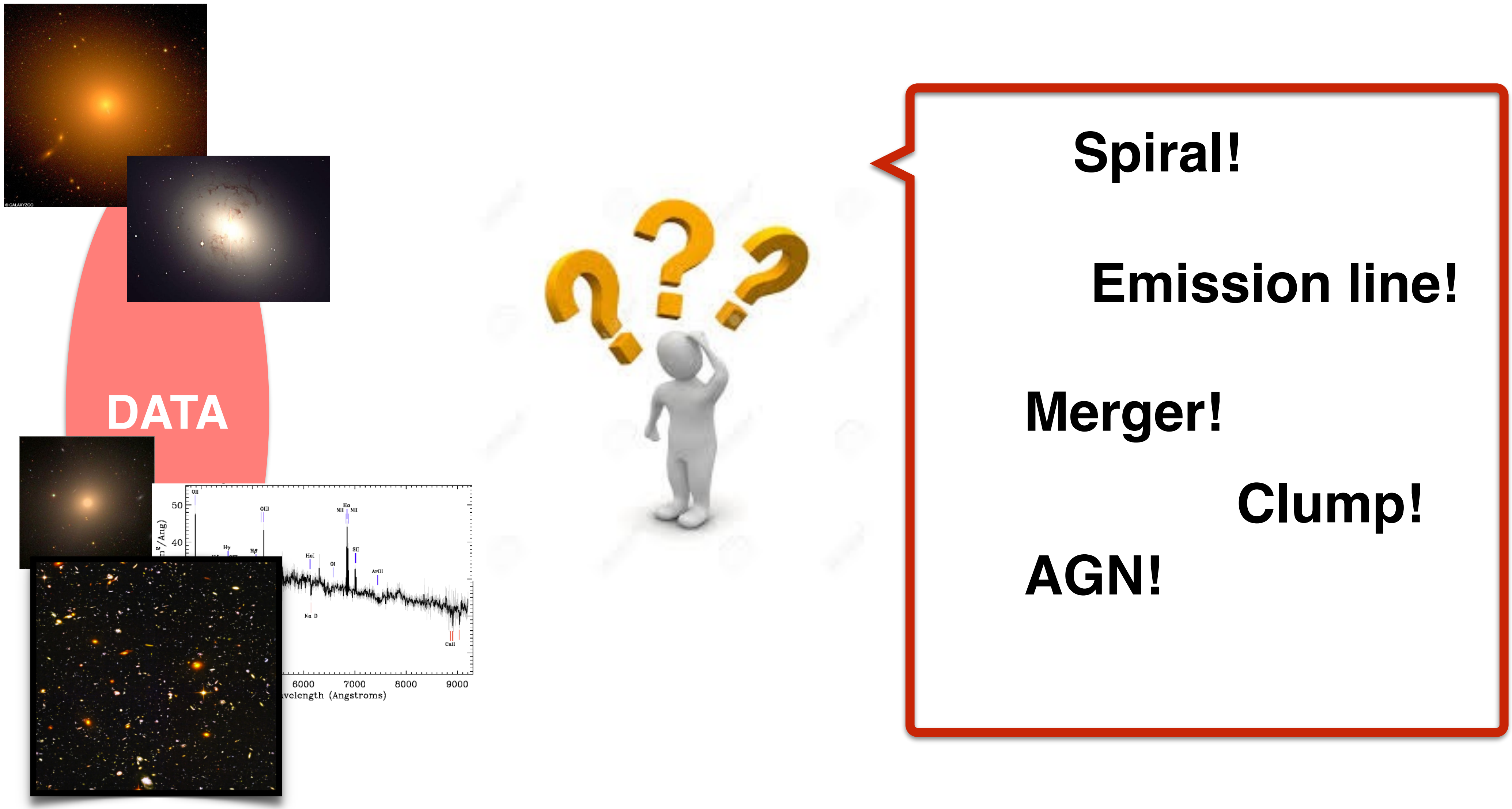


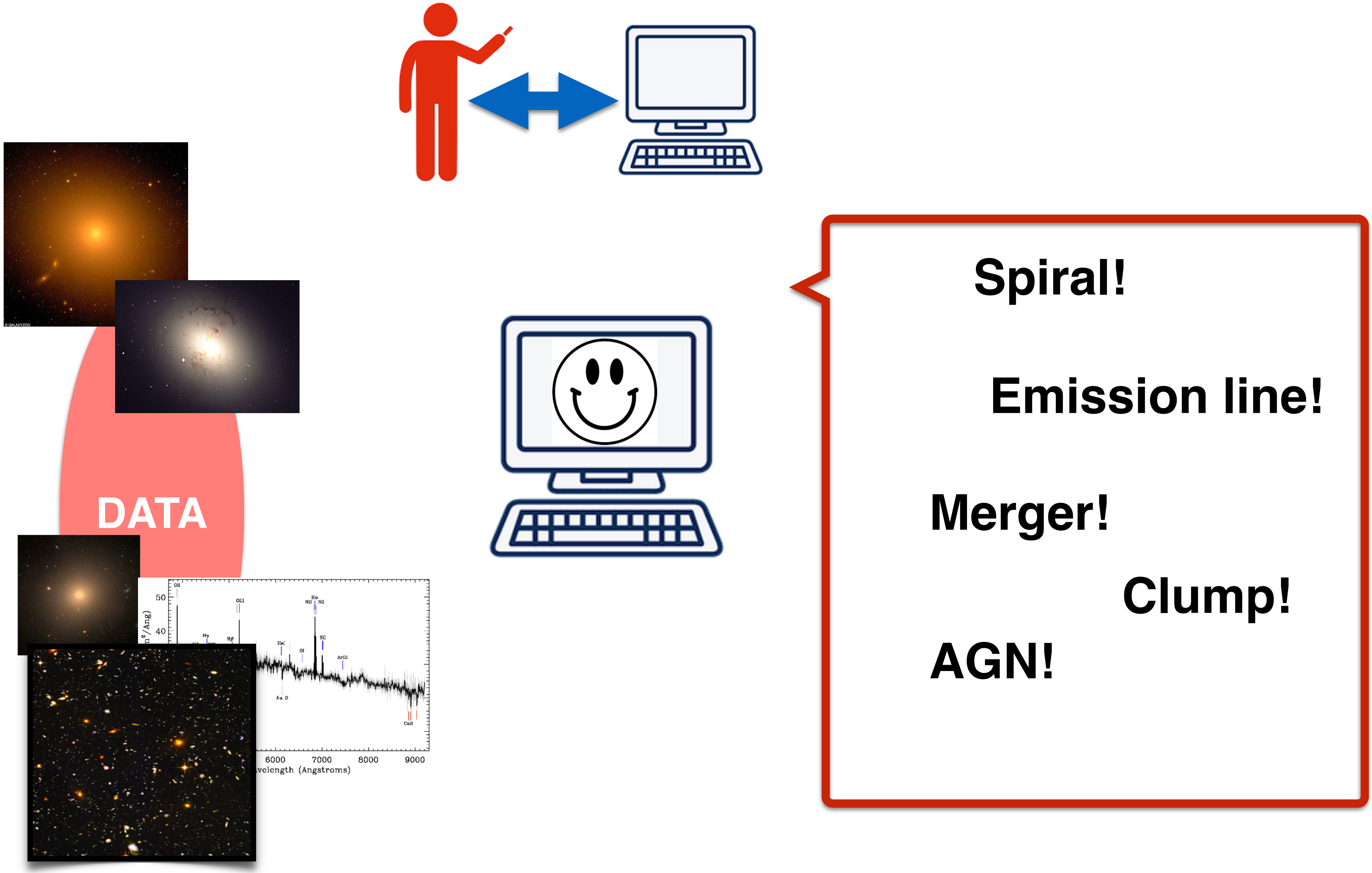
What do we put as input?



PRE-PROCESS DATA TO EXTRACT MEANINGFUL INFORMATION

THIS IS GENERALLY CALLED **FEATURE EXTRACTION**


DATA
50
40
6000
7000
8000
9000
Spiral!
Emission line!
Merger!
Clump!
AGN!



DATA
Spiral!
Emission line!
Merger!
Clump!
AGN!

$$f_{\vec{W}}(\vec{x}) = \vec{y}$$

LABEL

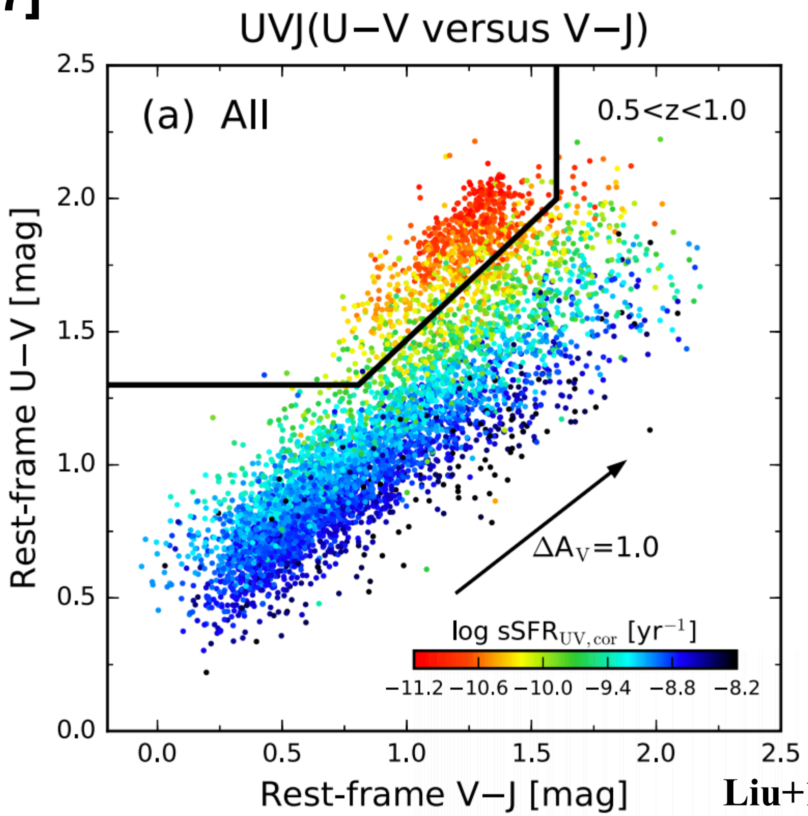
Q(0) , SF(1)

NETWORK FUNCTION

(U-V, V-J) FEATURES

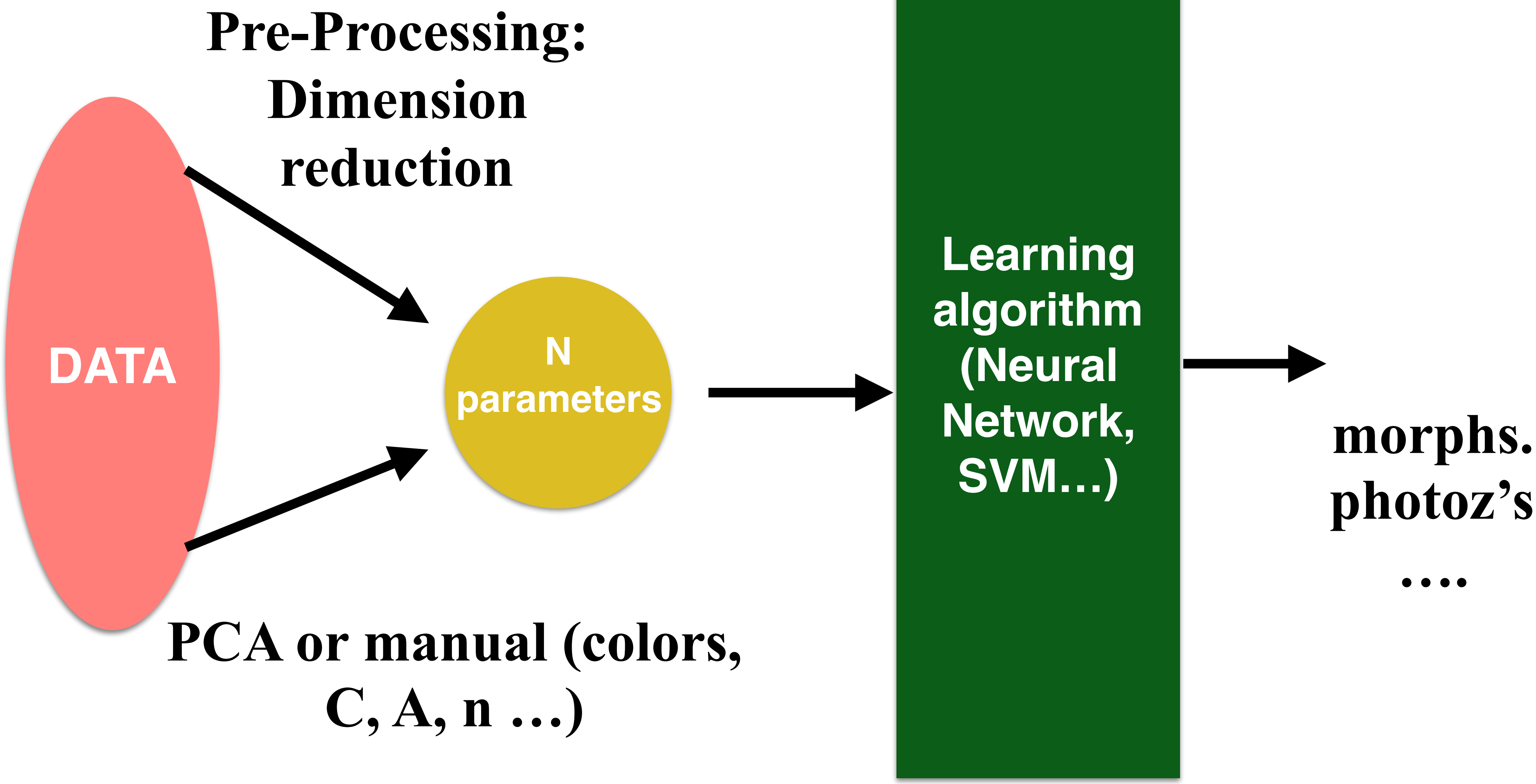
sgn[(u-v)-0.8*(v-j)-0.7]

WEIGHTS

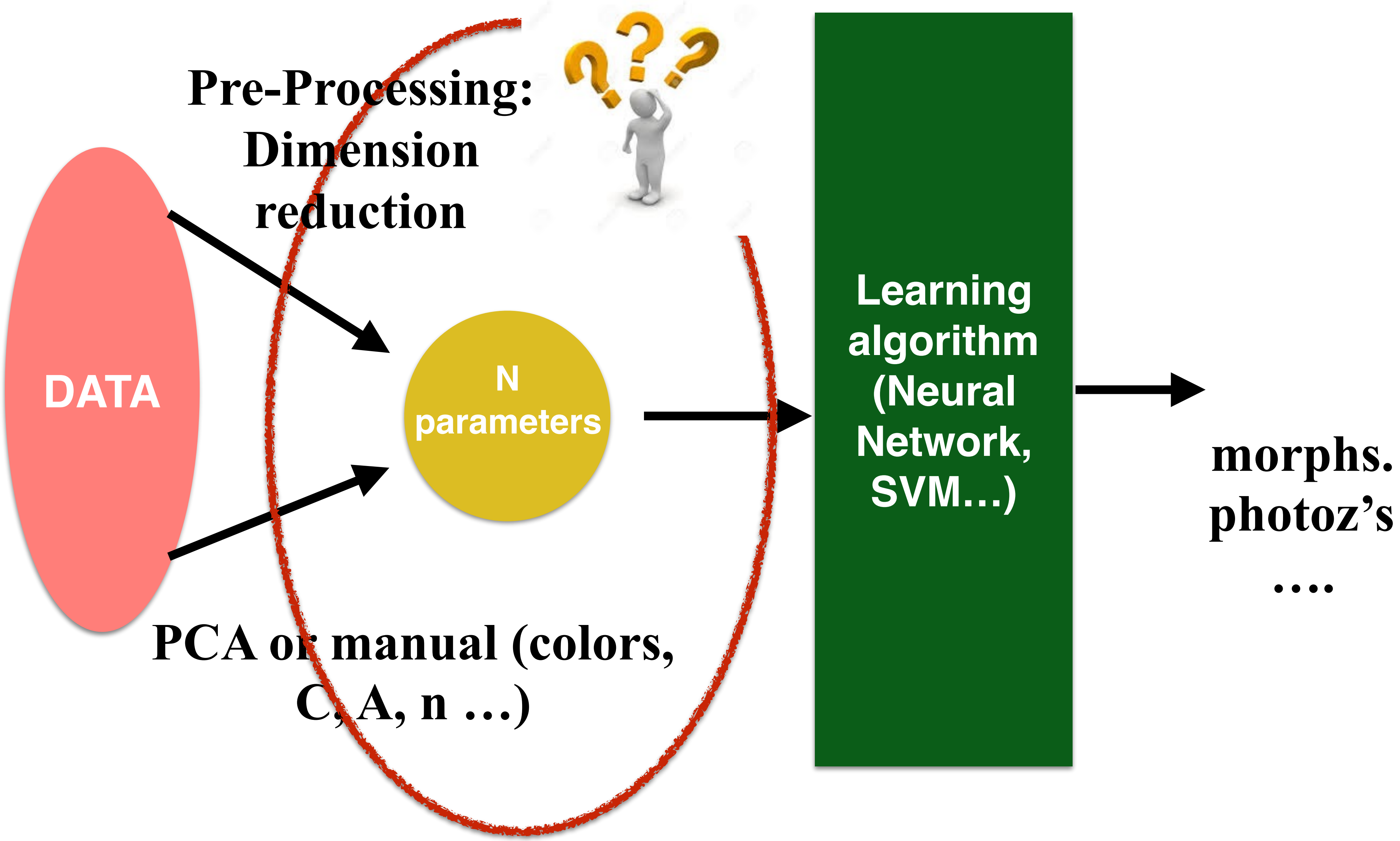


Liu+18

THE “CLASSICAL” APPROACH



“CLASSICAL” MACHINE LEARNING



In Astronomy

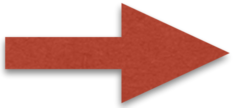
- Colors, Fluxes
- Shape indicators
- Line ratios, spectral features
- Stellar Masses, Velocity Dispersions

→ Requires specialized software before feeding the machine learning algorithm

IT IMPLIES A DIMENSIONALITY REDUCTION!

PHOTOMETRIC REDSHIFTS

SDSS

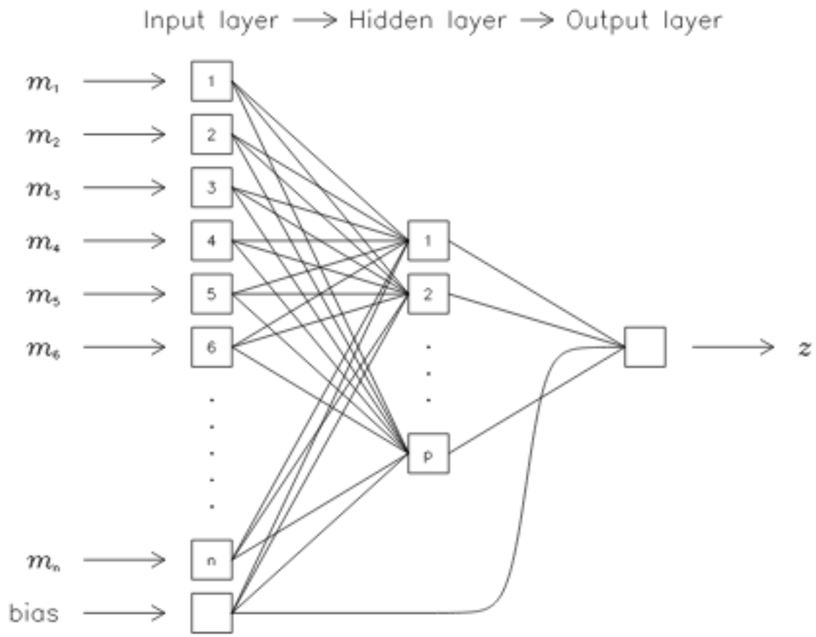


g

r

i

z



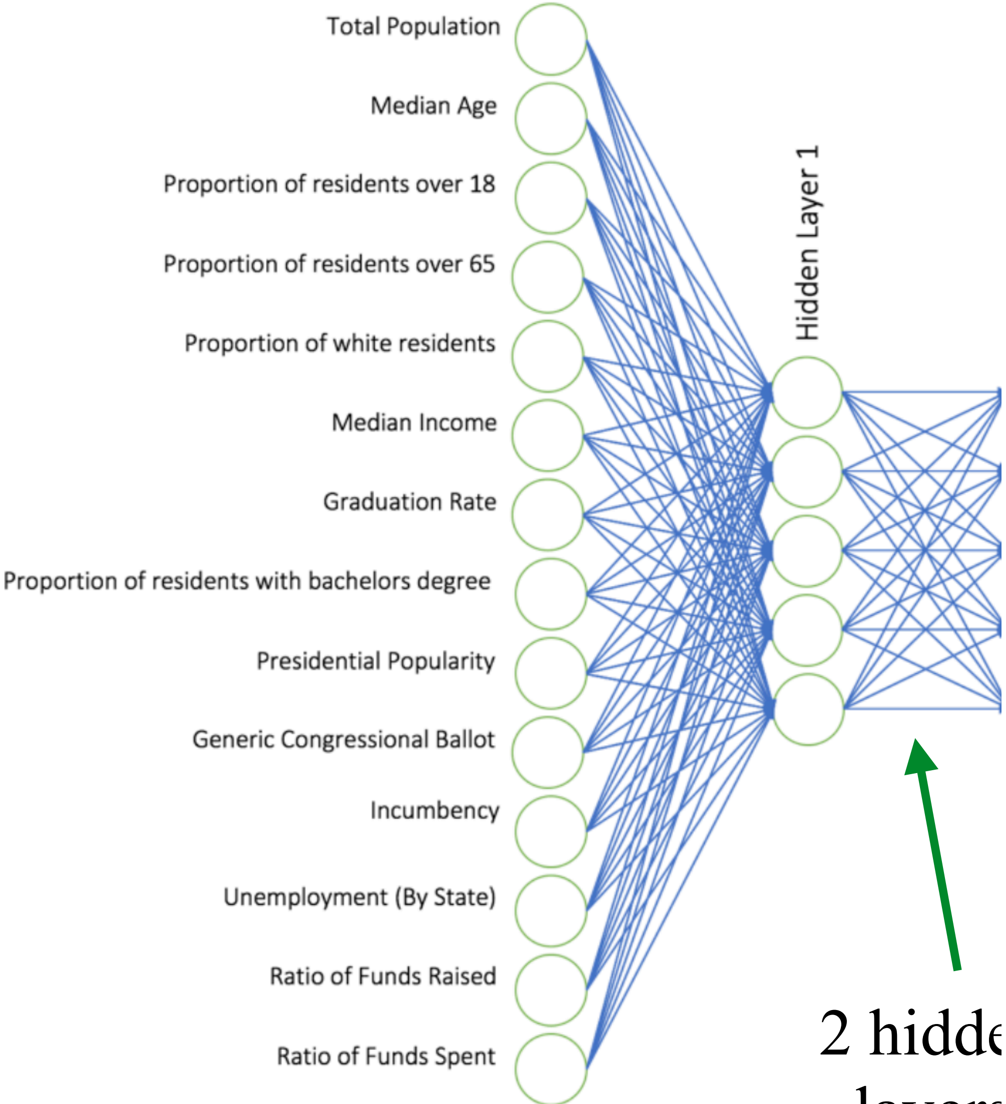
Collister+08

EVERYTHING IS IN THE FEATURES….WHAT IF I IGNORED SOME IMPORTANT FEATURES?

EVERYTHING IS IN THE FEATURES….WHAT IF I IGNORED SOME IMPORTANT FEATURES?

NEURAL NETWORK TO PREDICT RESULTS OF MIDTERM ELECTIONS

Features (x)



2 hidde layers

Bad Weather, Known to Lower Turnout, Will Greet Many Voters

Rain can decrease voter numbers, which studies show tends to help Republicans. "I hope it rains hard tomorrow," one Republican candidate said.

10h ago

Other general computer vision features [for images!]

- Pixel Concatenation
- Color histograms
- Texture Features
- Histogram of Gradients
- SIFT

FOR MANY YEARS COMPUTER VISION
RESEARCHERS HAVE BEEN TRYING TO FIND THE MOST GENERAL FEATURES

Other general computer vision features [for images!]

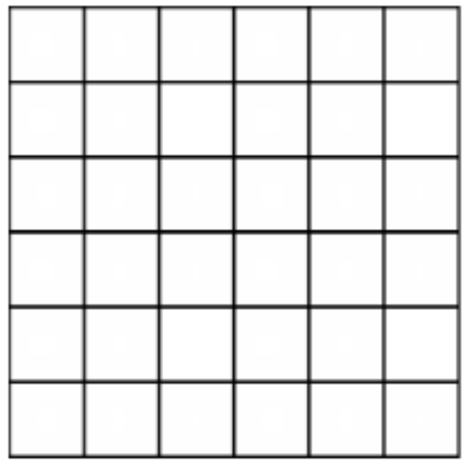
- Pixel Concatenation
- Color histograms
- Texture Features
- Histogram of Gradients
- SIFT

FOR MANY YEARS COMPUTER VISION
RESEARCHERS HAVE BEEN TRYING TO FIND THE MOST GENERAL FEATURES

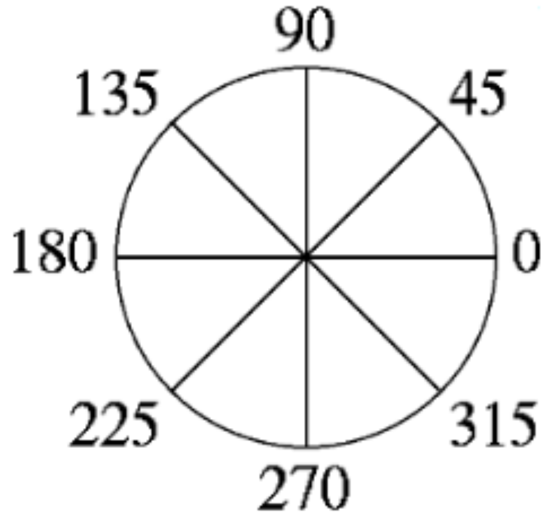
THE BEST CLASSICAL SOLUTION [BEFORE 2012] WHERE BASED **ON LOCAL FEATURES**

HISTOGRAM OF ORIENTED GRADIENTS (HoG)

1. DIVIDE IMAGE INTO SMALL SPATIAL REGIONS CALLED CELLS

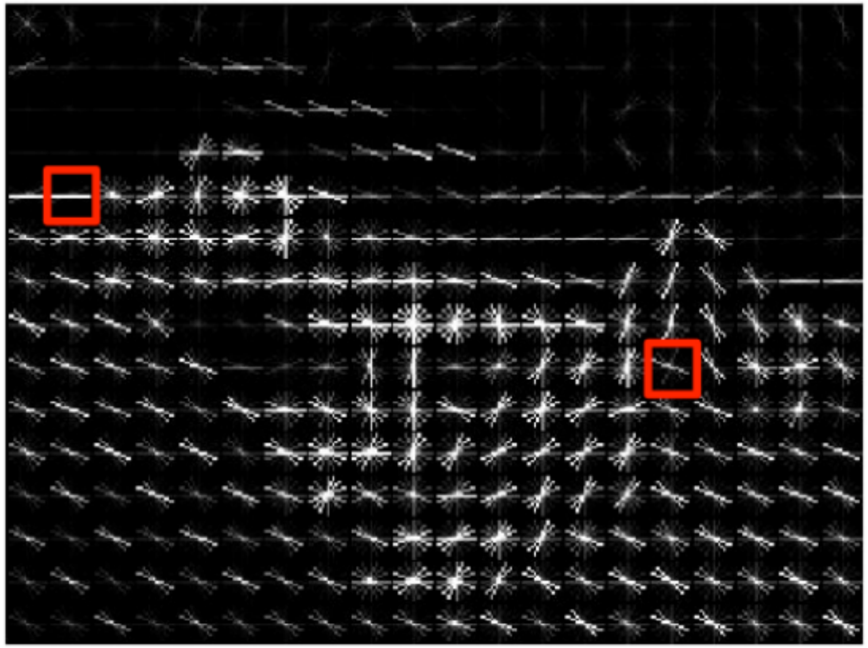


2. COMPUTE INTENSITY GRADIENTS OVER N DIRECTIONS [TYPICALLY 9 FOR IMAGE]

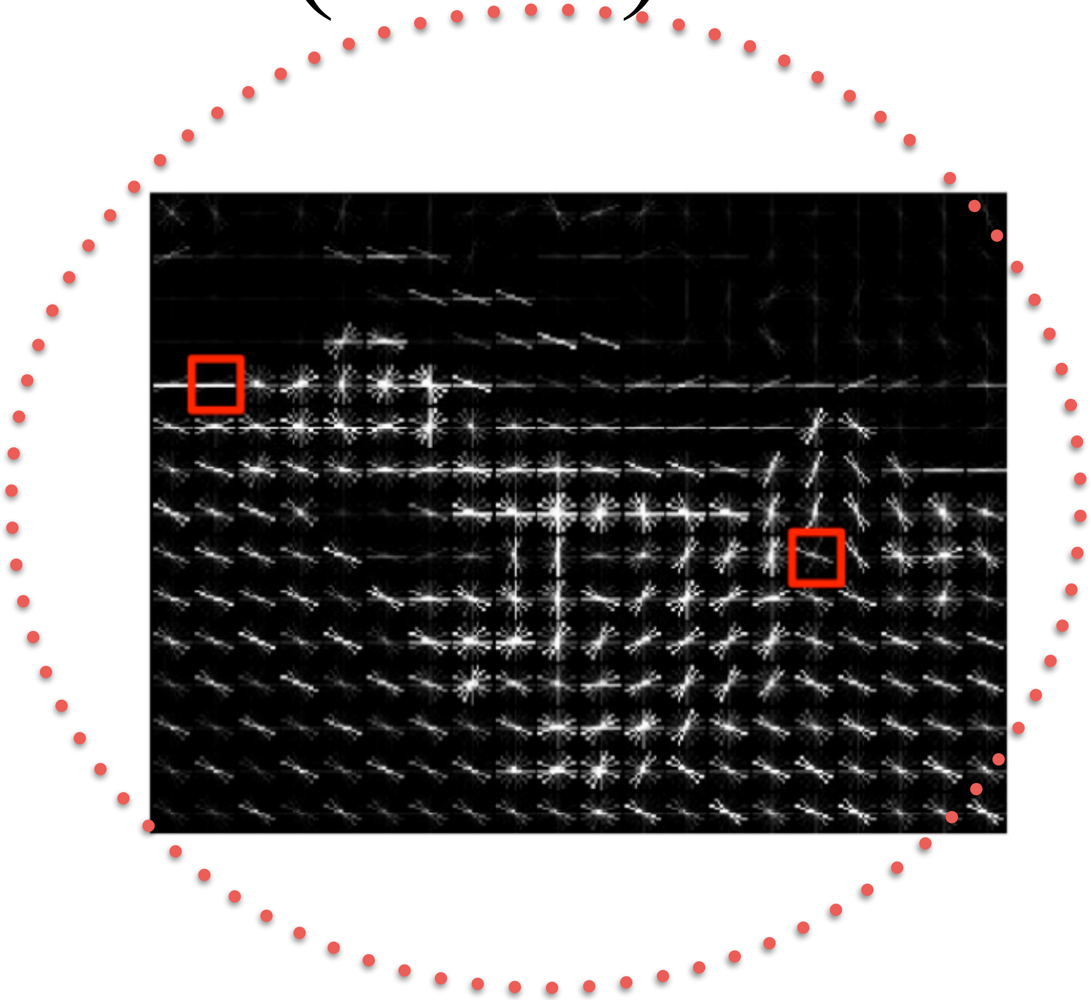


3. COMPUTE WEIGHTED 1-D HISTOGRAM OF ALL DIRECTIONS. A CELL IS REDUCED TO N NUMBERS

HISTOGRAM OF ORIENTED GRADIENTS (HoG)



HISTOGRAM OF ORIENTED GRADIENTS (HoG)



KEEP THIS IMAGE IN MIND FOR LATER…

What about using raw data?

ALL INFORMATION IS IN THE INPUT DATA

WHY REDUCING ?

LET THE NETWORK FIND THE INFO

What about using raw data?

ALL INFORMATION IS IN THE INPUT DATA

WHY REDUCING ?

LET THE NETWORK FIND THE INFO

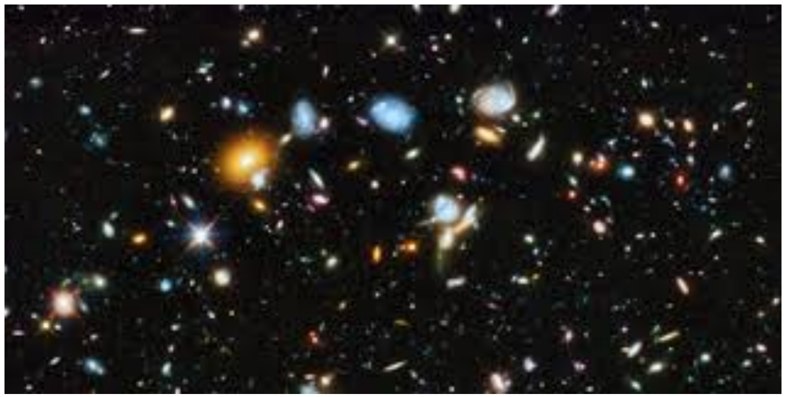
LARGE DIMENSION SIGNALS SUCH AS IMAGES OR SPECTRA WOULD REQUIRE TREMENDOUSLY LARGE MODELS

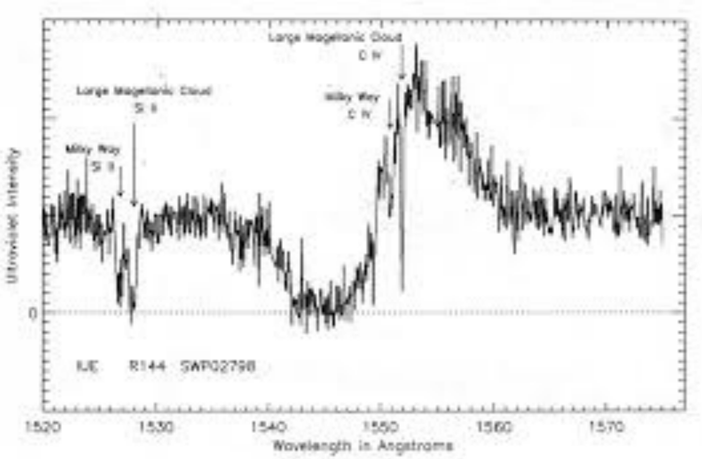
A 512x512 image as input of a fully connected layer producing output of same size:

$$(512 \times 512)^2 = 7e10$$

<u>BUT</u>

FEEDING INDIVIDUAL RESOLUTION ELEMENTS IS NOT VERY EFFICIENT SINCE IT LOOSES ALL INVARIANCE TO TRANSLATION AND IGNORES CORRELATION IN THE DATA





FEEDING INDIVIDUAL RESOLUTION ELEMENTS IS NOT VERY EFFICIENT SINCE IT LOOSES ALL INVARIANCE TO TRANSLATION

SO?

DEEP LEARNING

$$f_{\vec{W}}(\vec{x}) = \vec{y} \longrightarrow$$

LABEL
Q , SF

LET THE MACHINE FIGURE THIS OUT ("unsupervised feature extraction")

LET'S GO A STEP FORWARD INTO LOOSING CONTROL....

PART III: CONVOLUTIONAL NEURAL NETWORKS

Discrete Convolution

1D:
[Spectra]

$$f(x) * g(x) = \sum_{k=-\infty}^{k=+\infty} f(k).g(k-x)$$

2D:
[Images]

$$f(x,y) * g(x,y) = \sum_{k=-\infty}^{k=+\infty} \sum_{l=-\infty}^{l=+\infty} f(k,l).g(x-k, y-l)$$

DISCRETE CONVOLUTION

1D:
[Spectra]

$$f(x) * g(x) = \sum_{k=-\infty}^{k=+\infty} f(k).g(k-x)$$

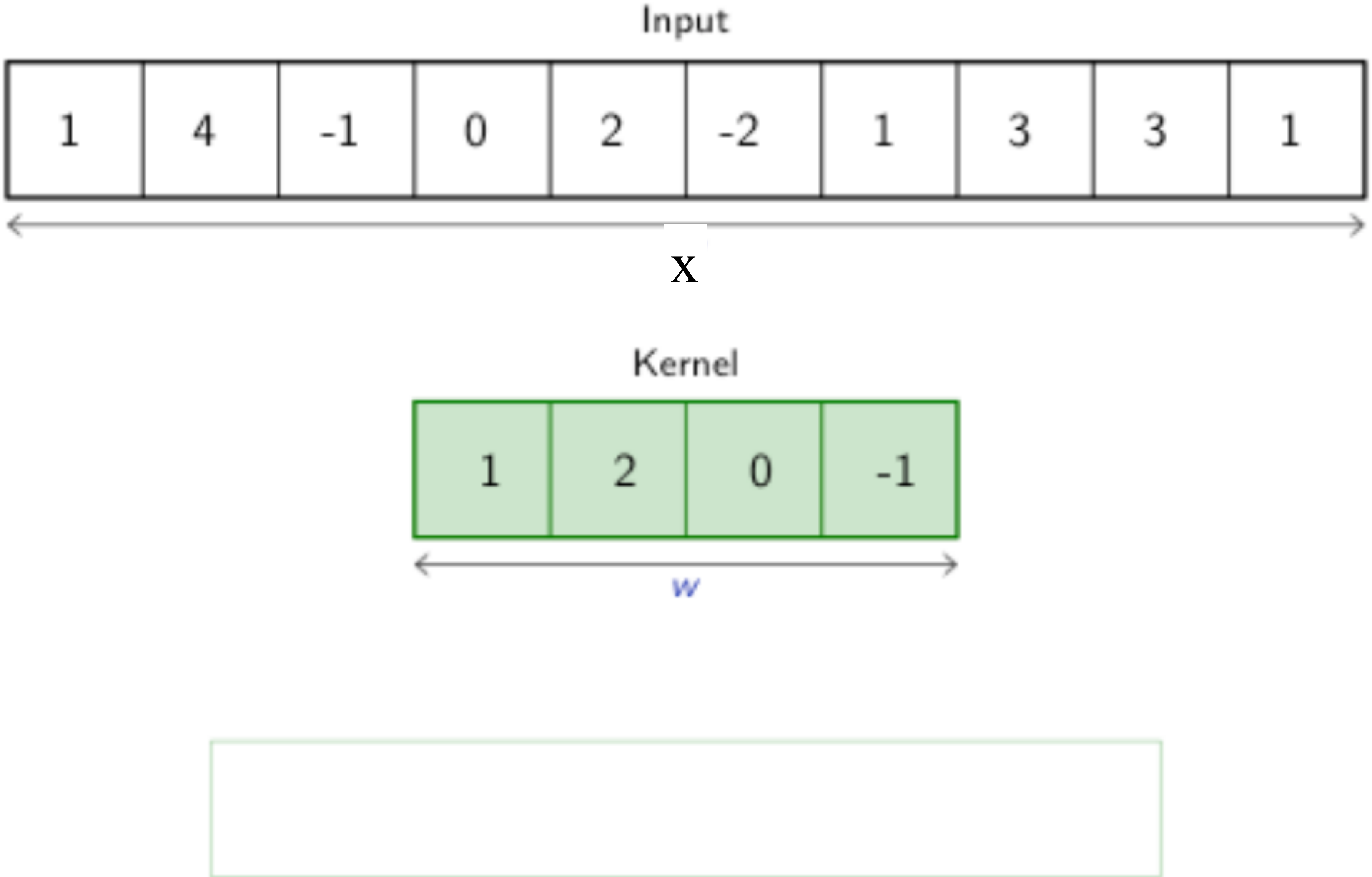
2D:
[Images]

$$f(x,y) * g(x,y) = \sum_{k=-\infty}^{k=+\infty} \sum_{l=-\infty}^{l=+\infty} f(k,l).g(x-k,y-l)$$

<u>CONVOLUTION KERNEL</u>

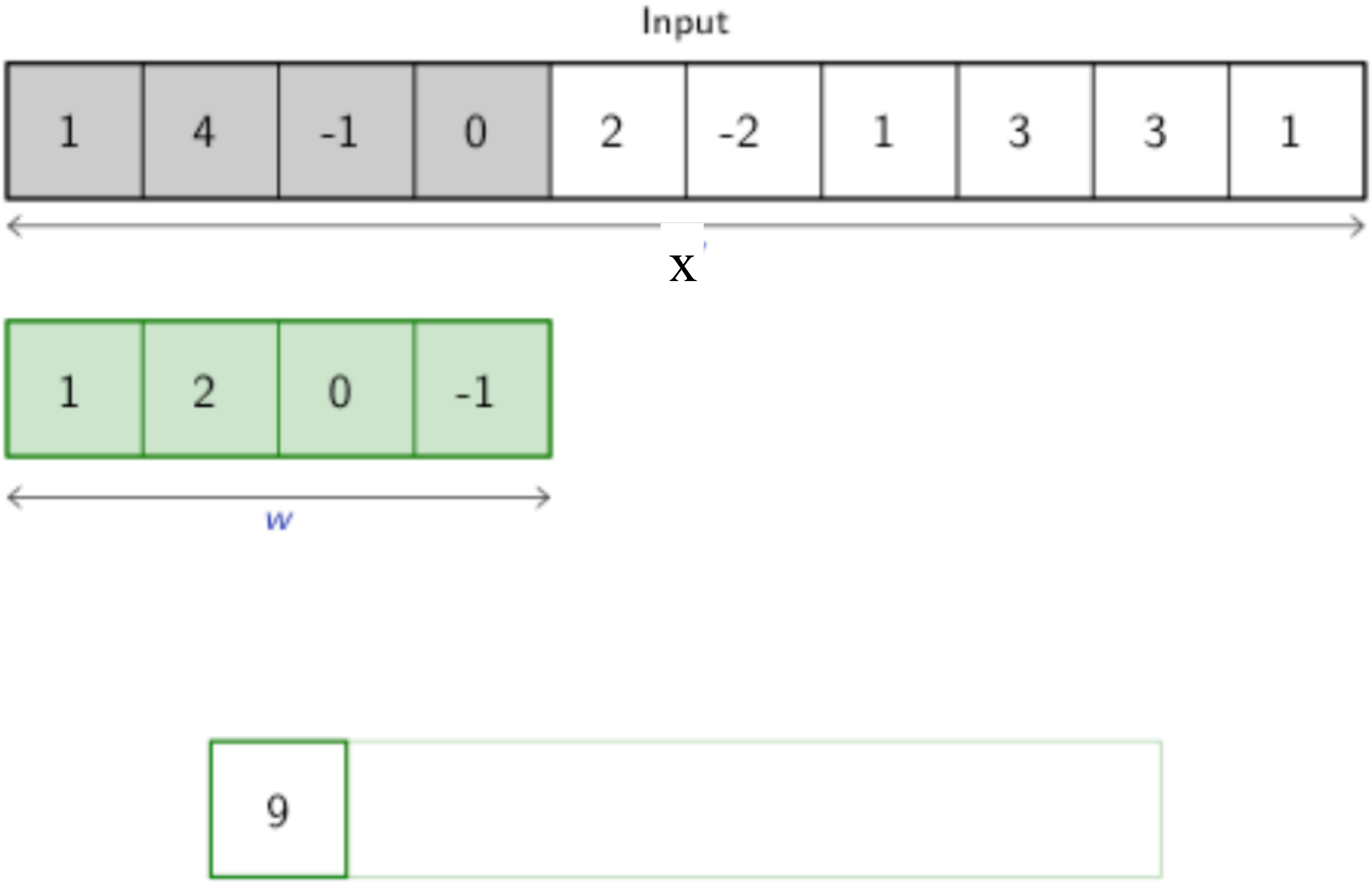
<u>INPUT DATA</u>

1-D CONVOLUTION



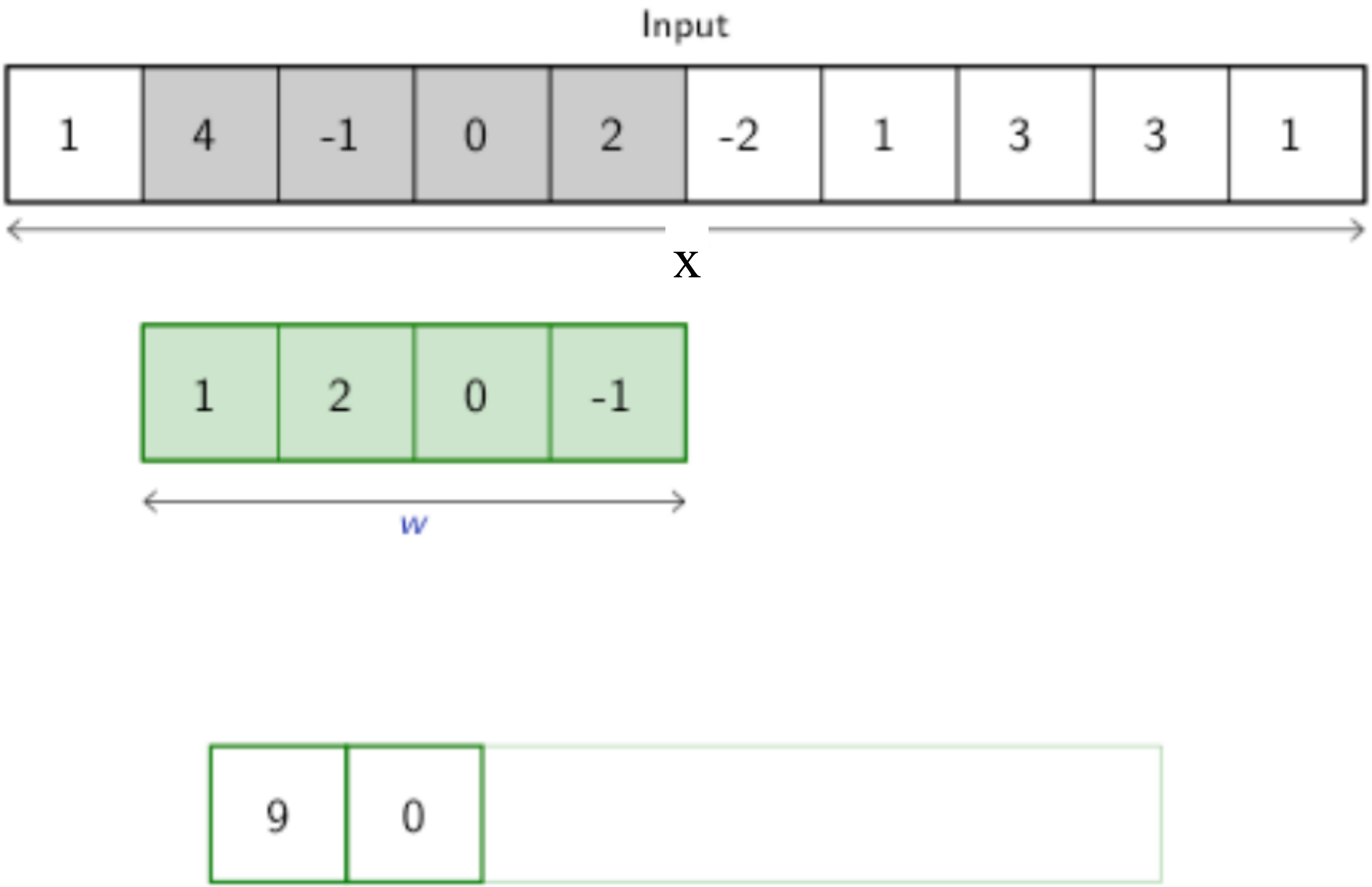
credit

1-D CONVOLUTION



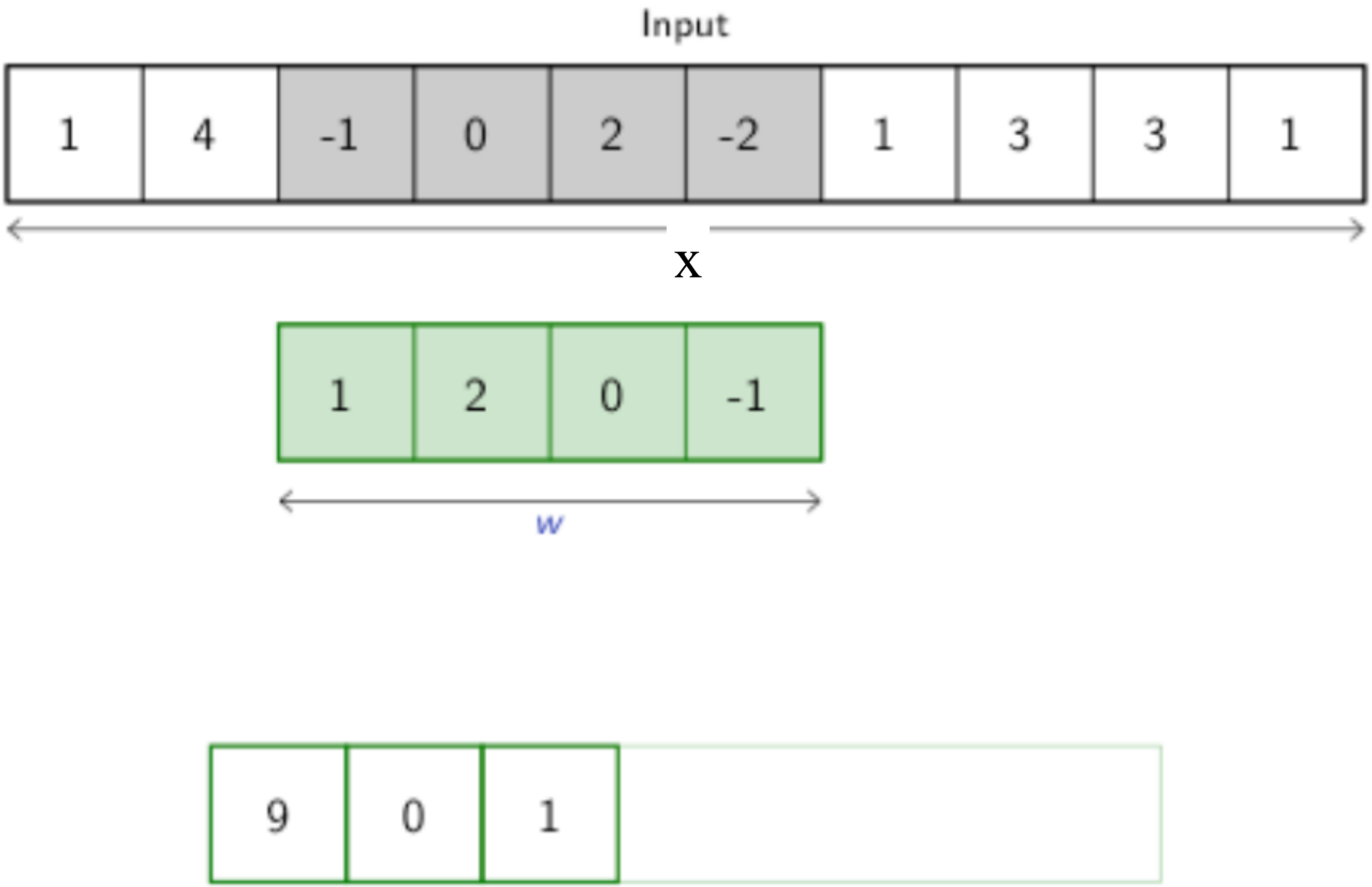
credit

1-D CONVOLUTION



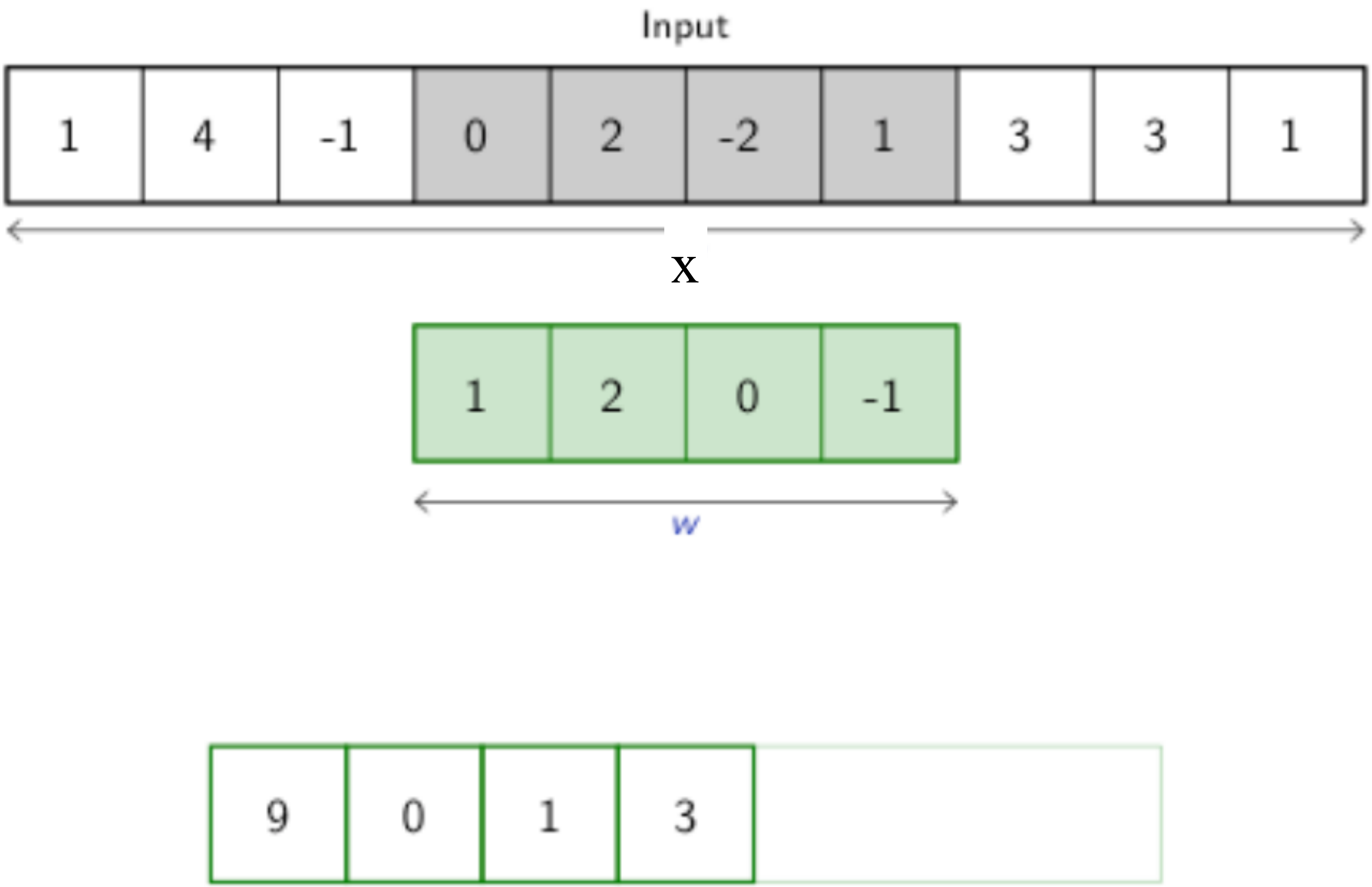
credit

1-D CONVOLUTION



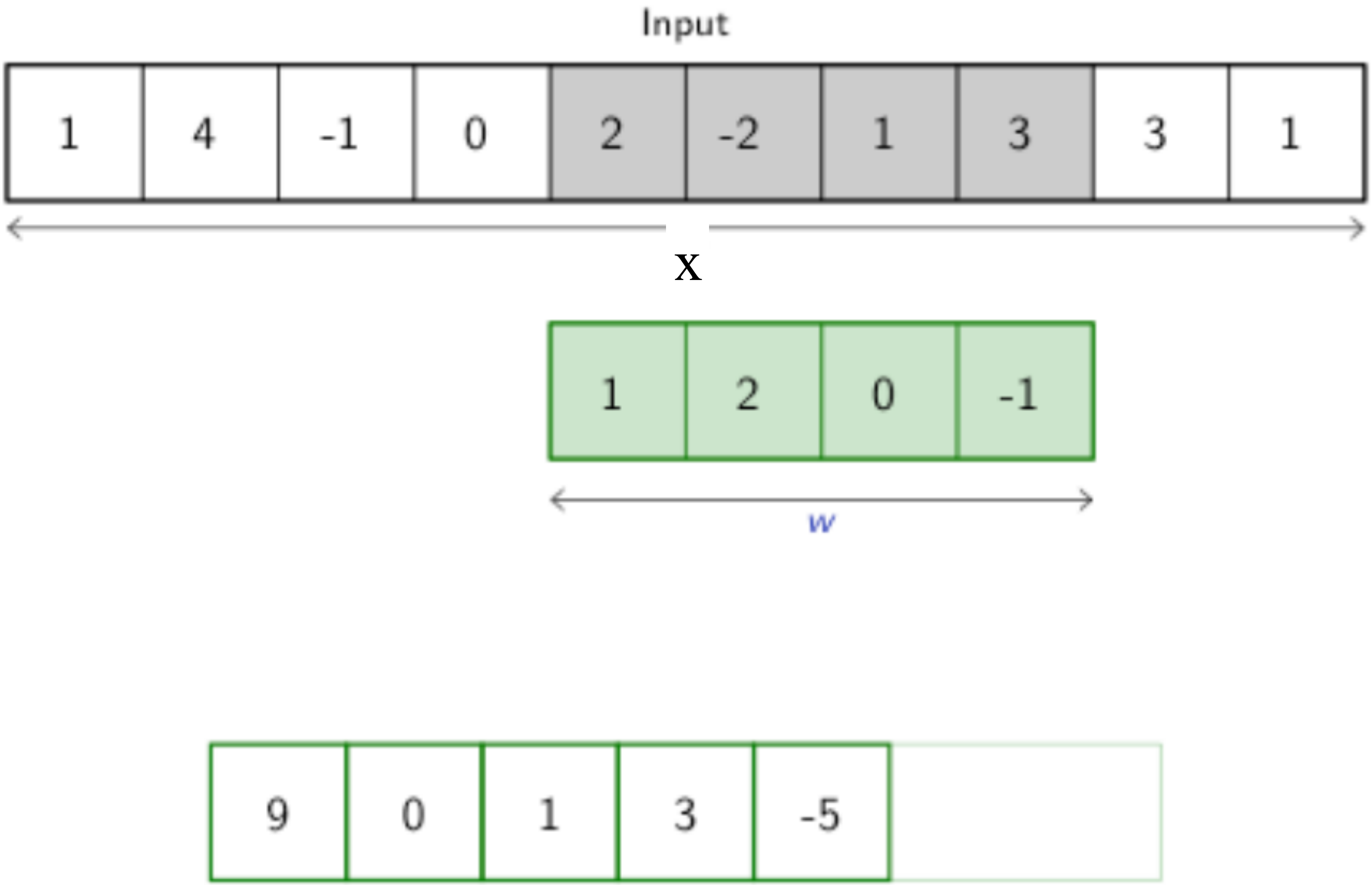
credit

1-D CONVOLUTION



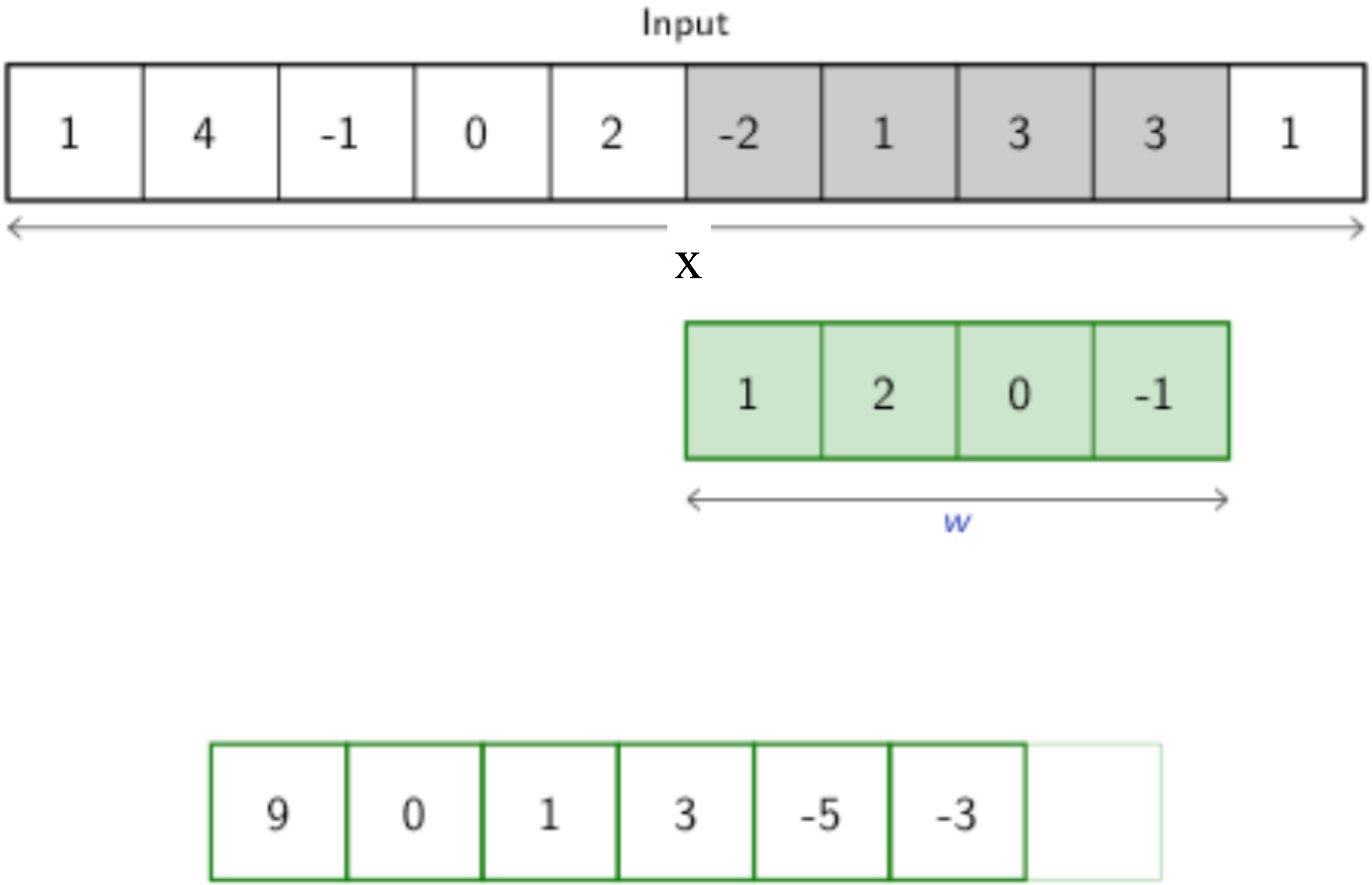
credit

1-D CONVOLUTION



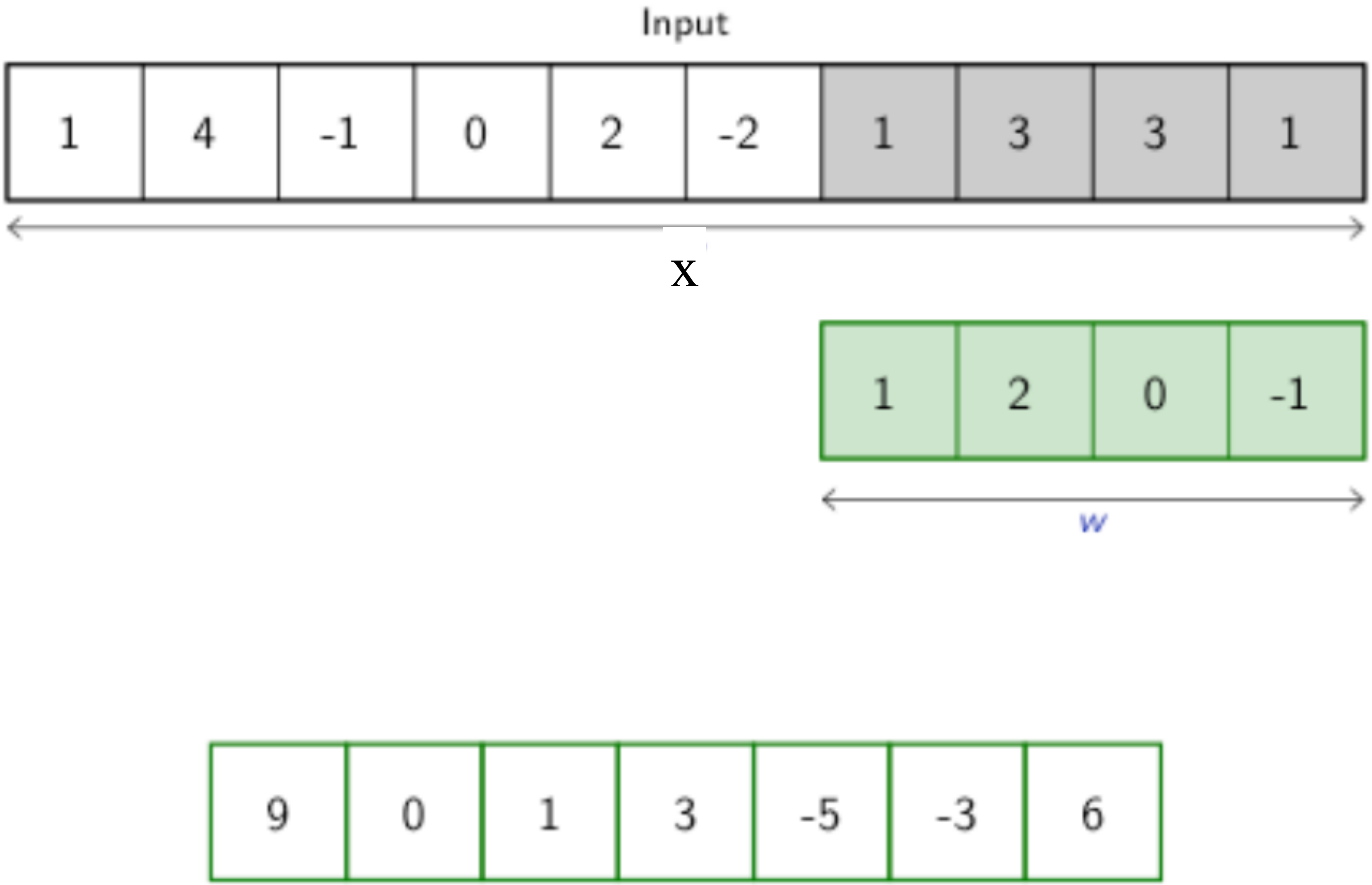
credit

1-D CONVOLUTION



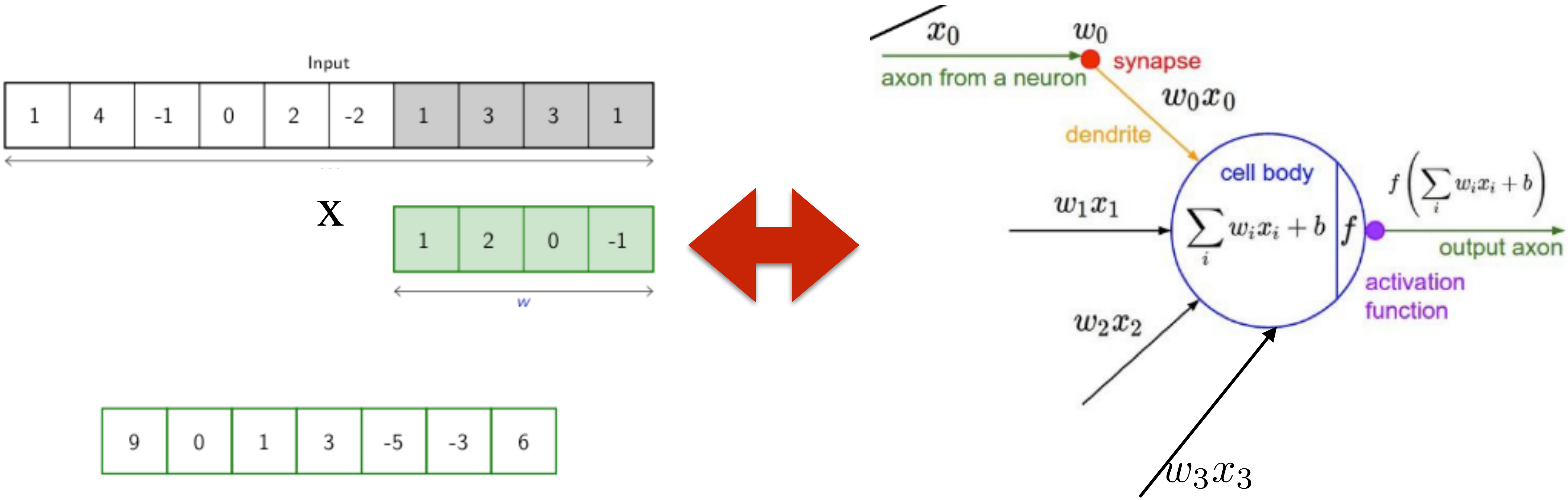
credit

1-D CONVOLUTION

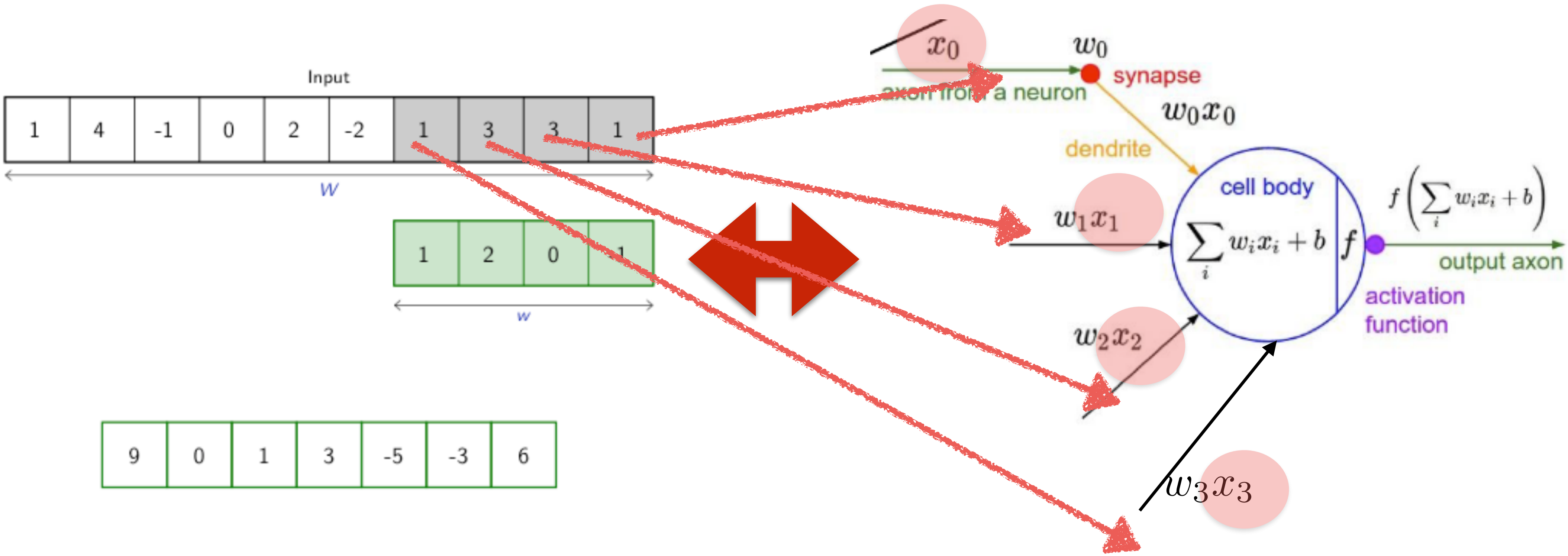


credit

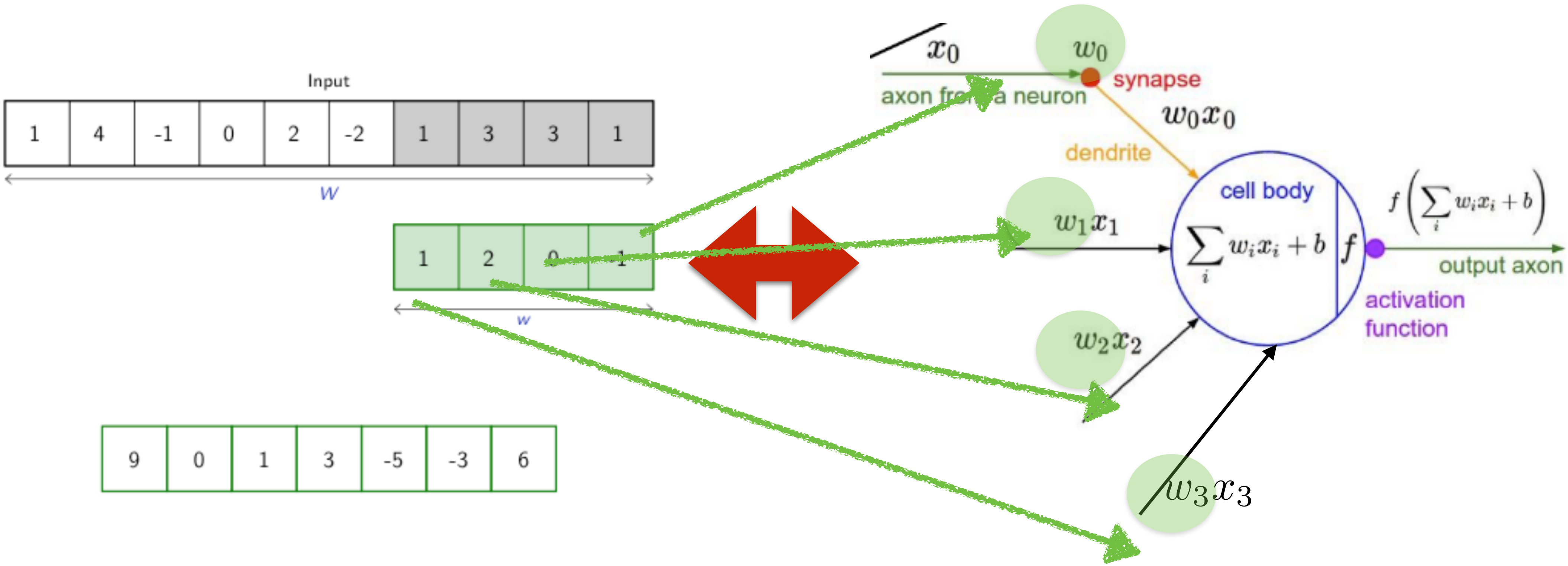
THE CONVOLUTION BUILDING BLOCK OPERATION IS EQUIVALENT TO A NEURON WITH AS MANY INPUTS AS KERNEL ELEMENTS AND WEIGHTS EQUAL TO THE KERNEL



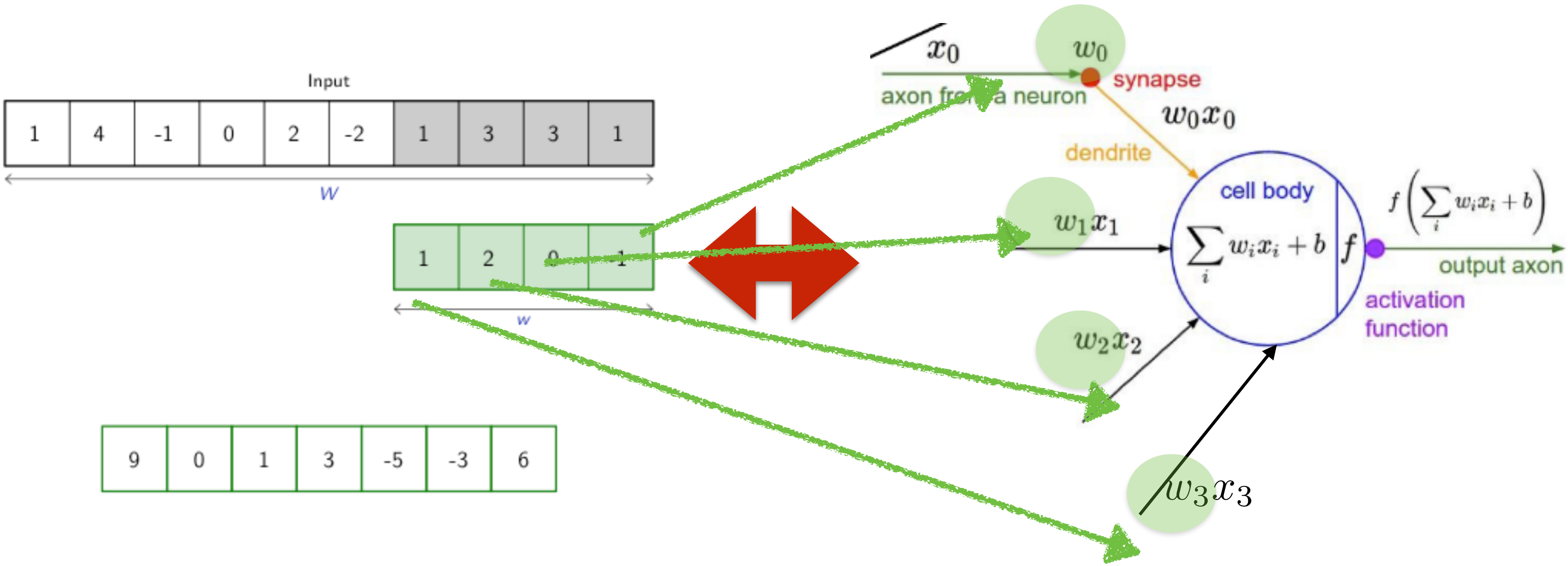
THE CONVOLUTION BUILDING BLOCK OPERATION IS EQUIVALENT TO A NEURON WITH AS MANY INPUTS AS KERNEL ELEMENTS AND WEIGHTS EQUAL TO THE KERNEL



THE CONVOLUTION BUILDING BLOCK OPERATION IS EQUIVALENT TO A NEURON WITH AS MANY INPUTS AS KERNEL ELEMENTS AND WEIGHTS EQUAL TO THE KERNEL



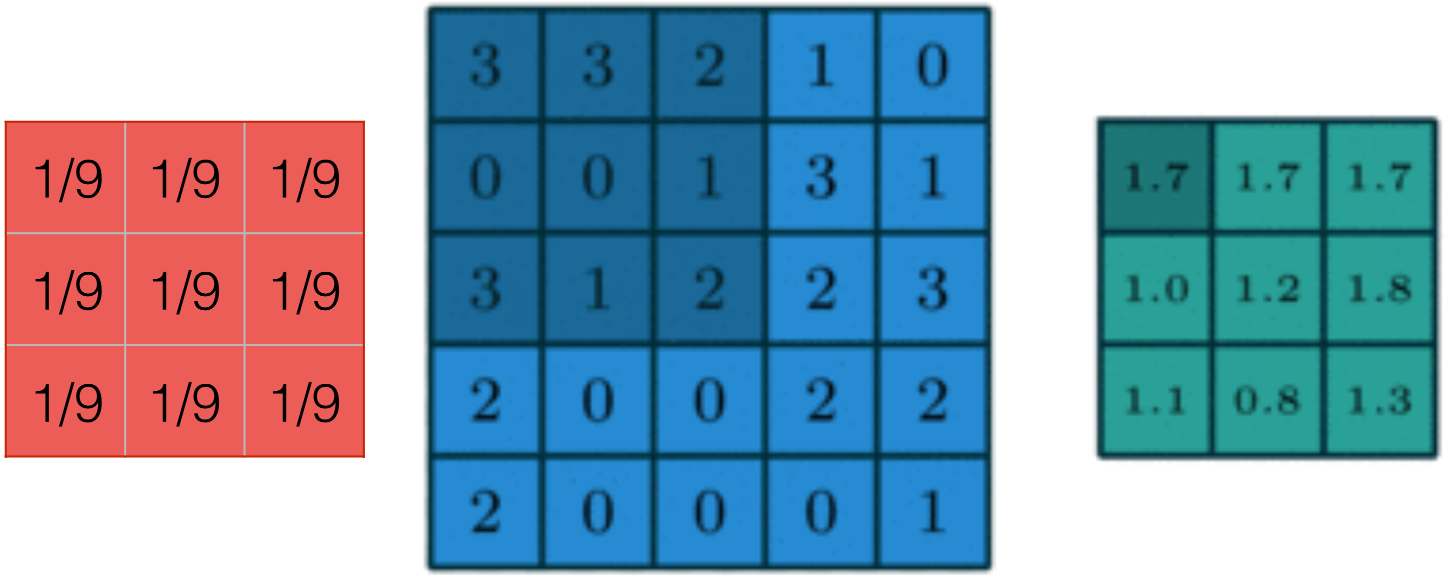
THE CONVOLUTION BUILDING BLOCK OPERATION IS EQUIVALENT TO A NEURON WITH AS MANY INPUTS AS KERNEL ELEMENTS AND WEIGHTS EQUAL TO THE KERNEL



WITH THE ADVANTAGE THAT THE SAME WEIGHTS ARE APPLIED TO ALL THE SIGNAL: <u>TRANSLATION INVARIANCE</u>

2-D CONVOLUTION

SAME IDEA, BUT THE KERNEL IS NOW 2D

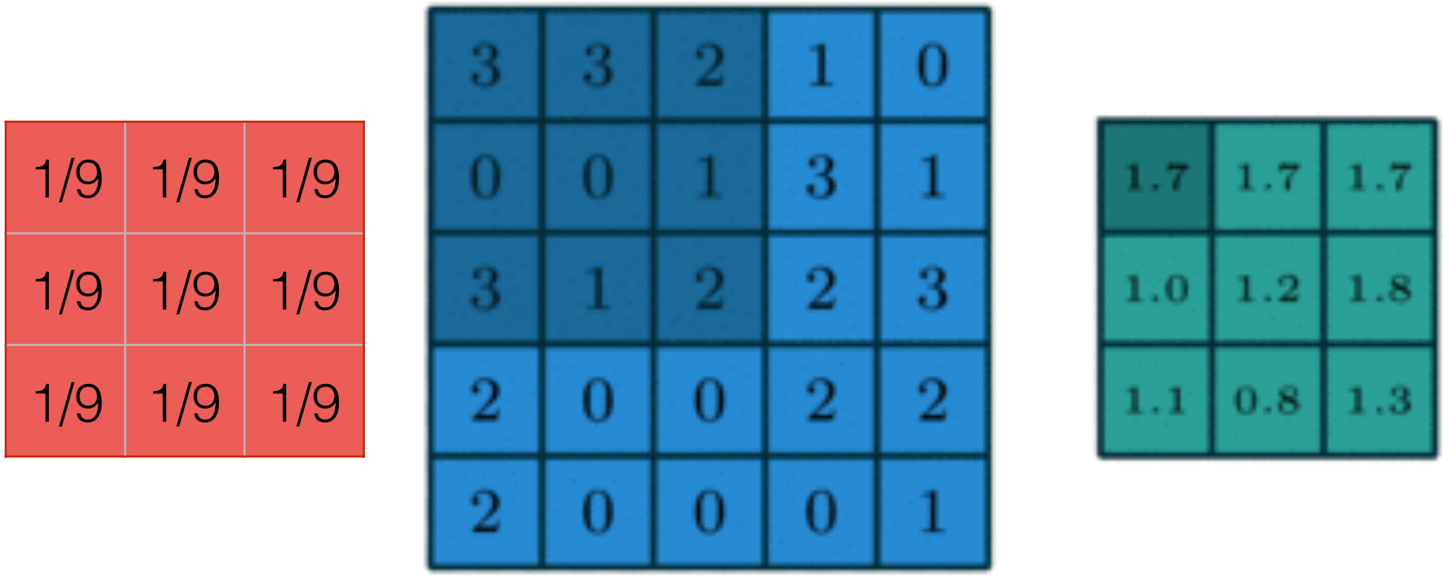


KERNEL **INPUT (IMAGE)** **OUTPUT**

Credit: animations from https://github.com/vdumoulin/conv_arithmetic

2-D CONVOLUTION

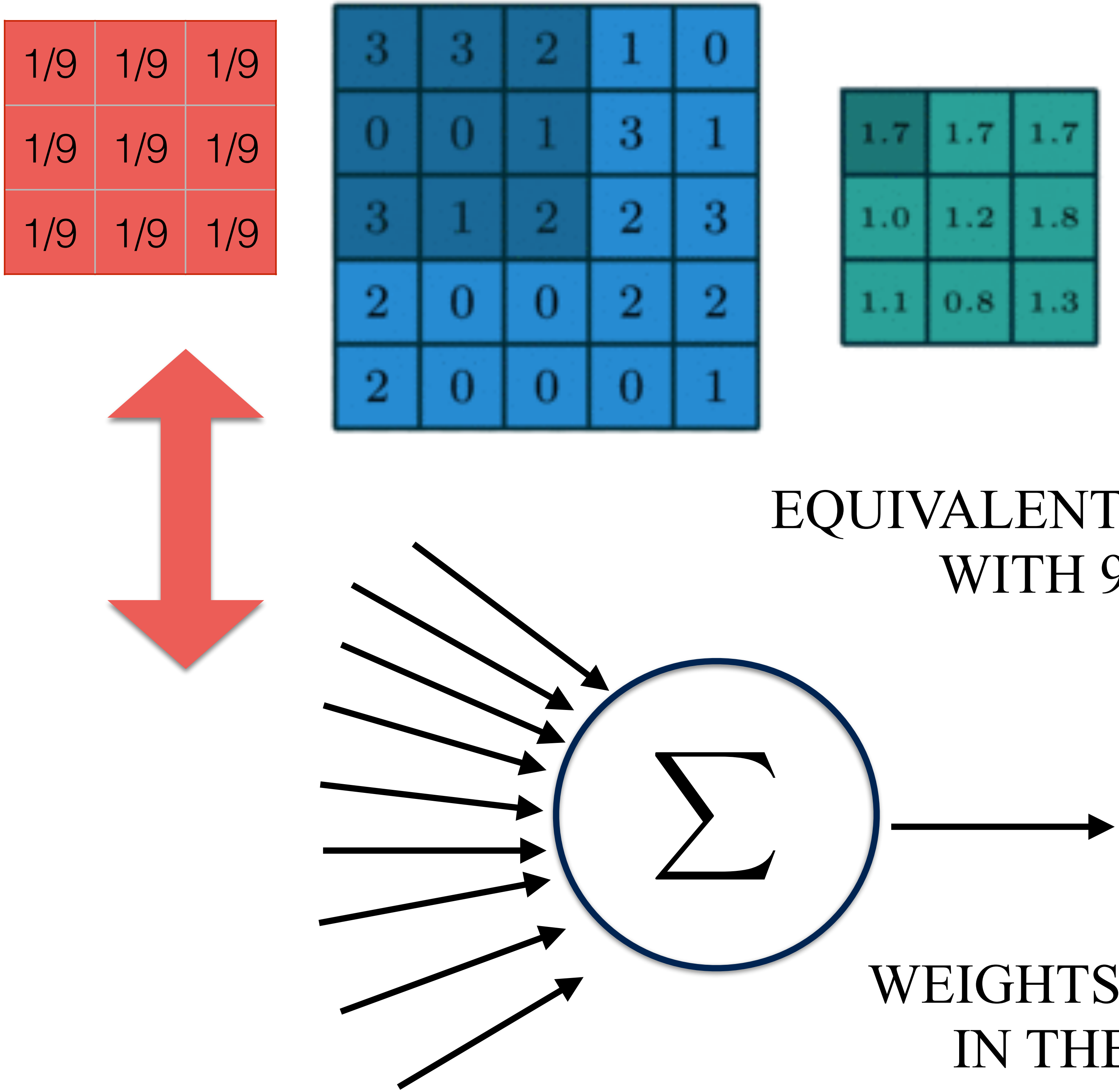
<u>SAME IDEA, BUT THE KERNEL IS NOW 2D</u>



IN THE EXAMPLE: EACH 3x3 REGION GENERATES AN OUTPUT

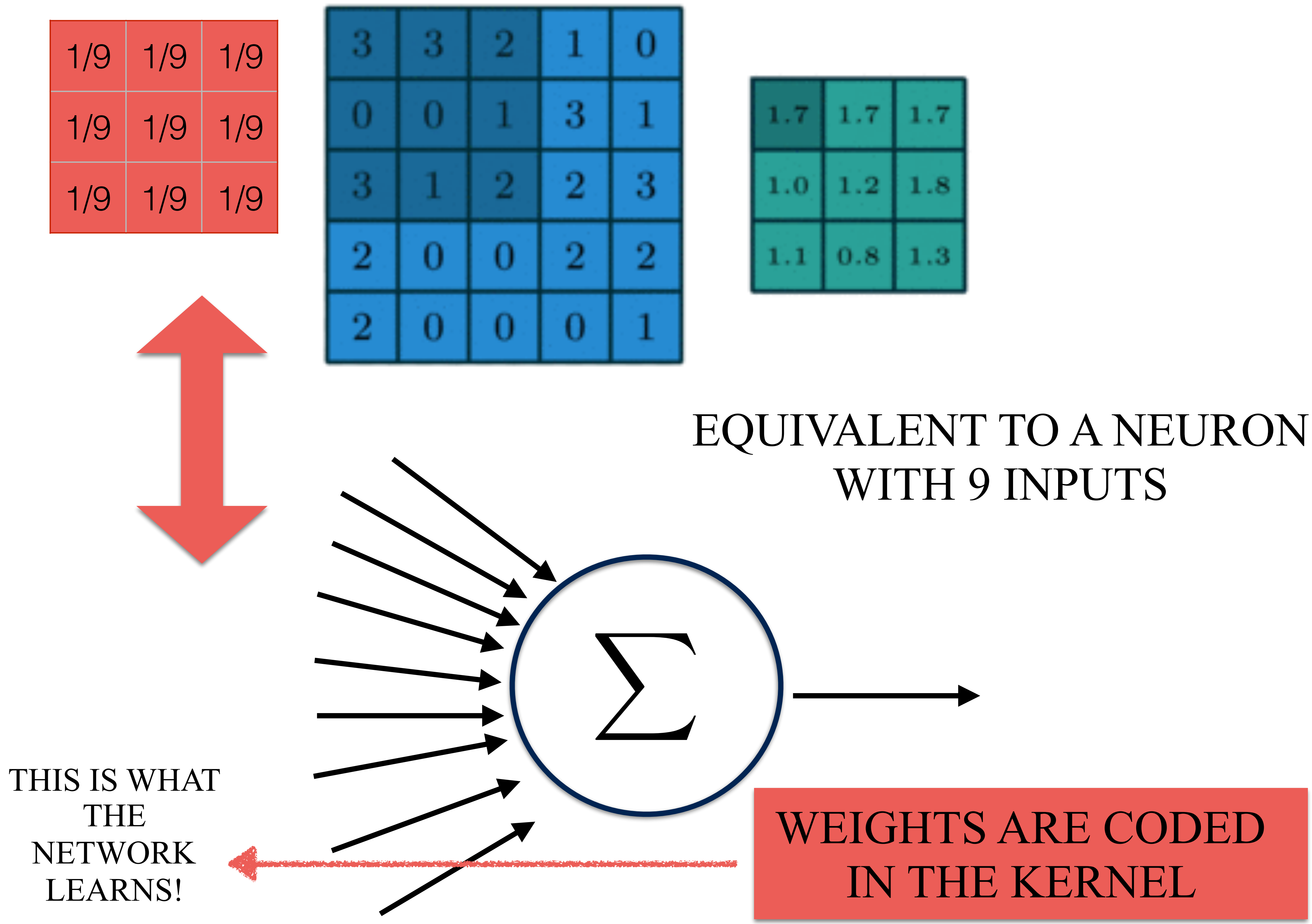
$$\boxed{Size_{output} = Size_{input} - Size_{kernel} + 1}$$

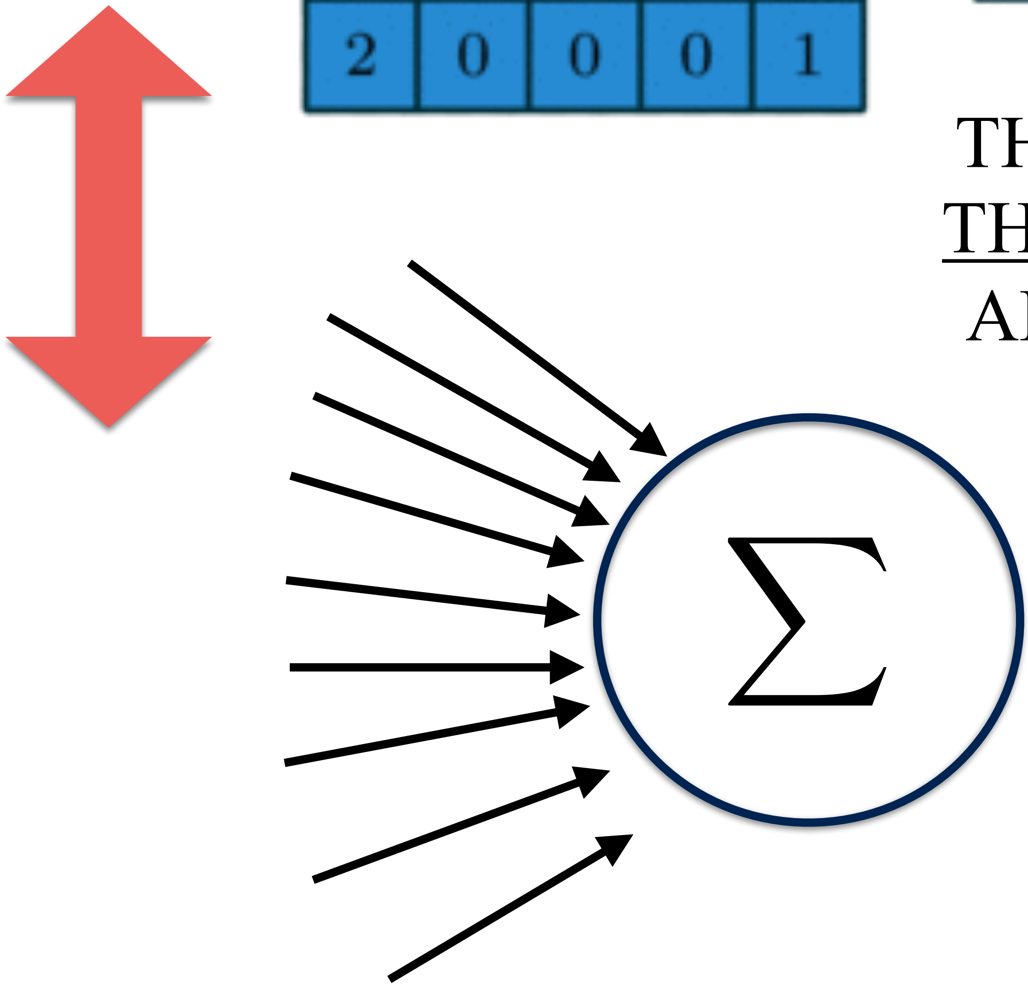
Credit: animations from https://github.com/vdumoulin/conv_arithmetic



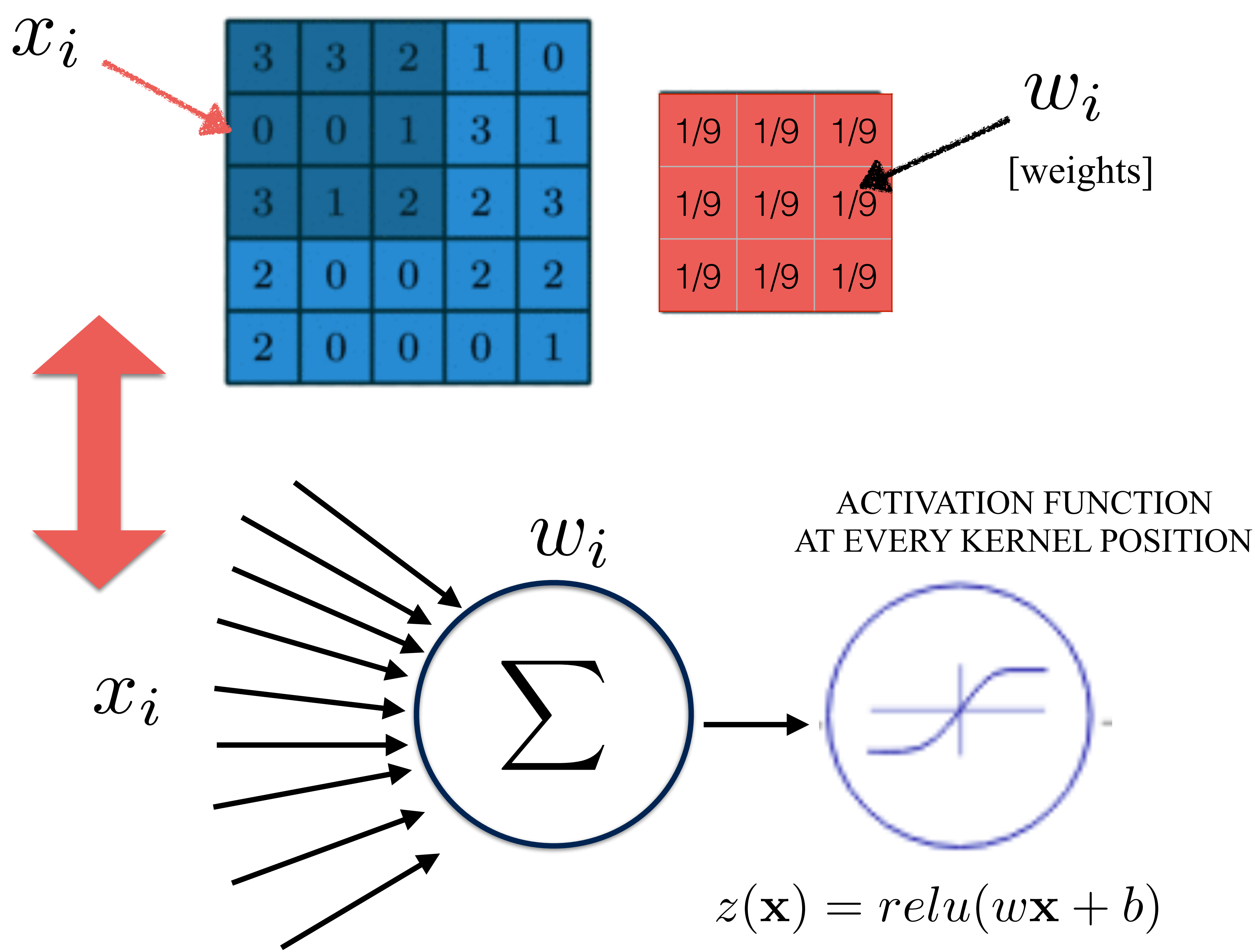
EQUIVALENT TO A NEURON
WITH 9 INPUTS

WEIGHTS ARE CODED
IN THE KERNEL

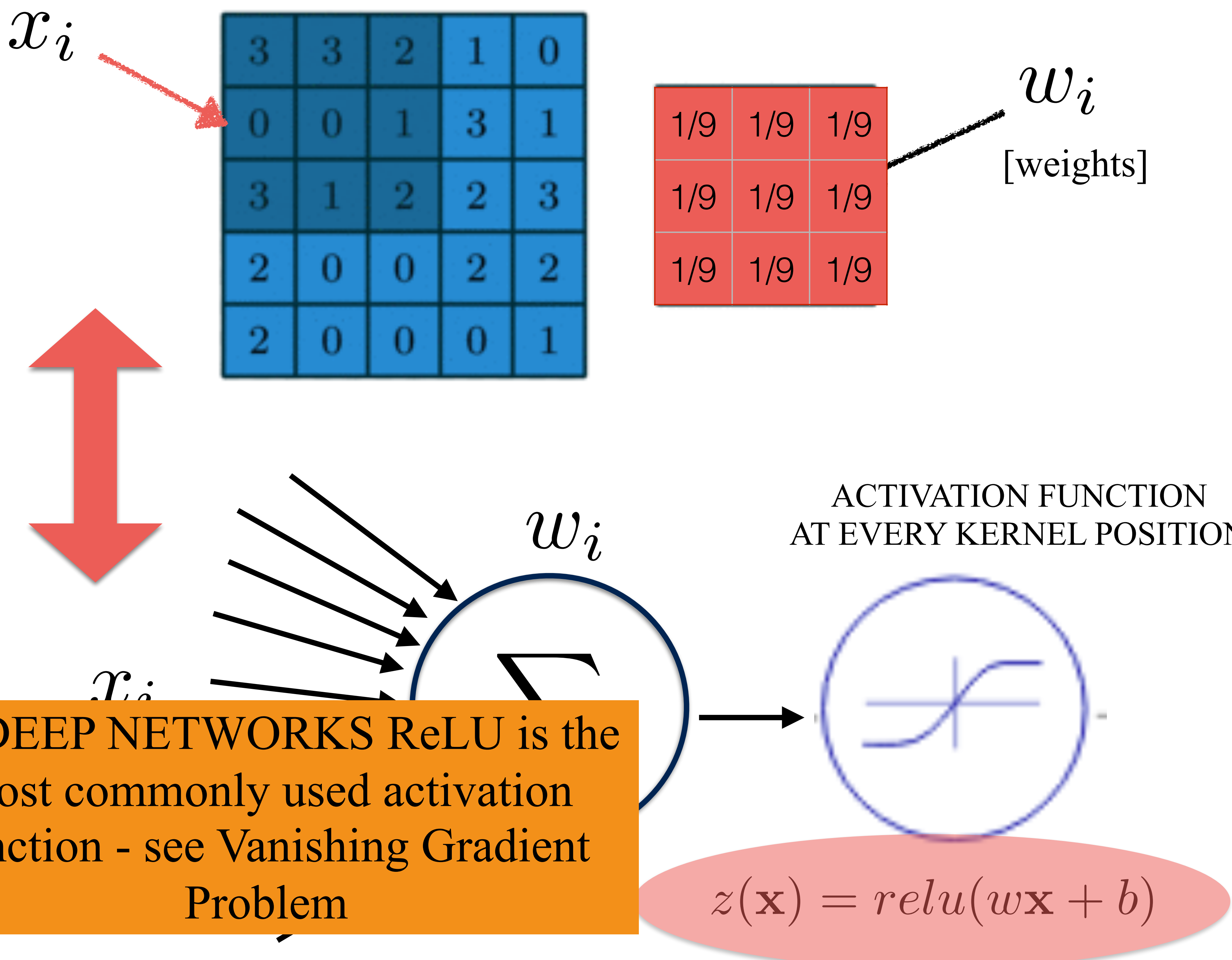

1/9 1/9 1/9
1/9 1/9 1/9
1/9 1/9 1/9
3 3 2 1 0
0 0 1 3 1
3 1 2 2 3
2 0 0 2 2
2 0 0 0 1
1.7 1.7 1.7
1.0 1.2 1.8
1.1 0.8 1.3
EQUIVALENT TO A NEURON
WITH 9 INPUTS
Σ
THIS IS WHAT
THE
NETWORK
LEARNS!
WEIGHTS ARE CODED
IN THE KERNEL



THE KEY IS AGAIN THAT
THE SAME WEIGHTS ARE
APPLIED TO ALL IMAGE
REGIONS

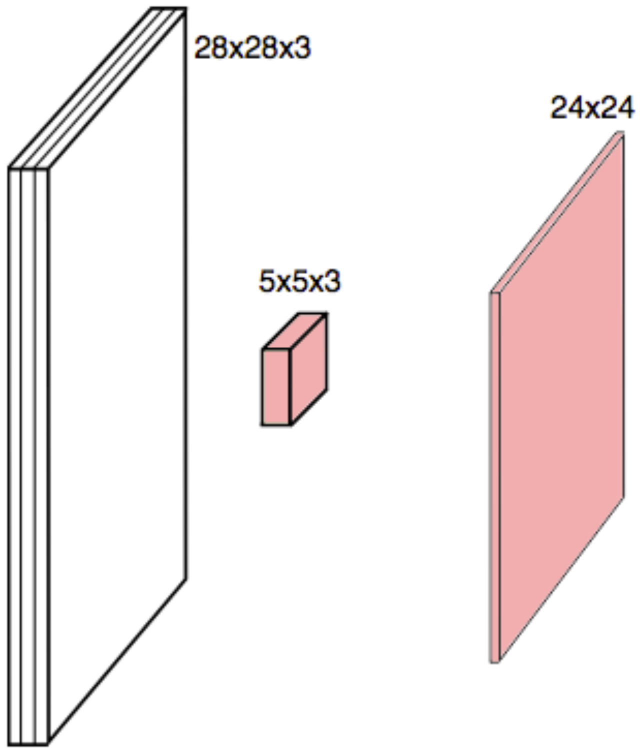


x_i
3 3 2 1 0
0 0 1 3 1
3 1 2 2 3
2 0 0 2 2
2 0 0 0 1
1/9 1/9 1/9
1/9 1/9 1/9
1/9 1/9 1/9
w_i
[weights]
w_i
x_i
Σ
ACTIVATION FUNCTION
AT EVERY KERNEL POSITION
$z(\mathbf{x}) = relu(w\mathbf{x} + b)$

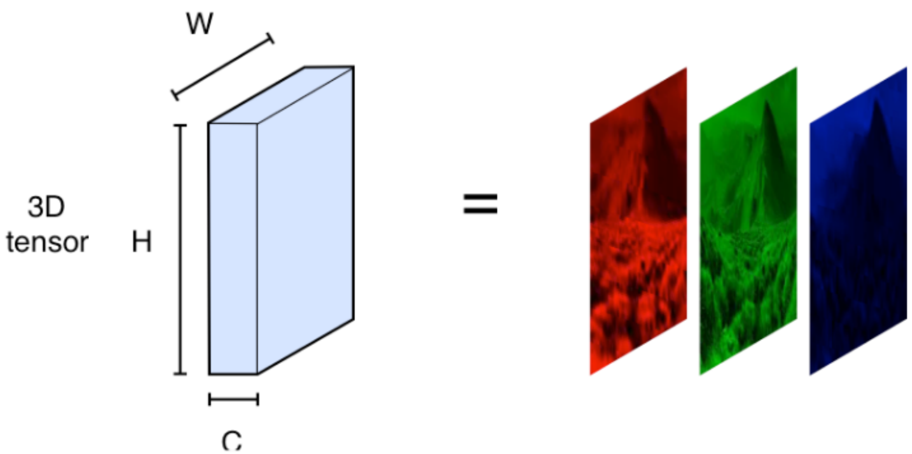


IN DEEP NETWORKS ReLU is the most commonly used activation function - see Vanishing Gradient Problem

CONVOLUTIONS CAN ALSO BE COMPUTED ACROSS CHANNELS (OR COLORS)

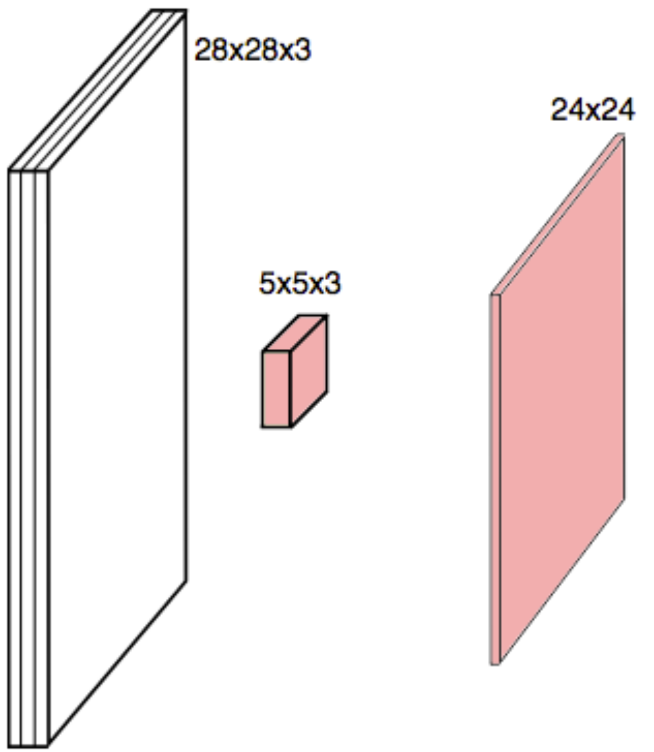


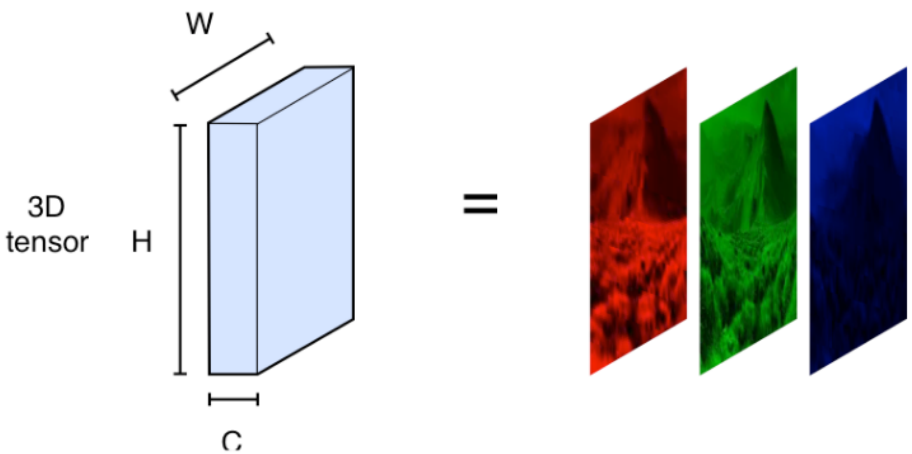
A COLOR IMAGE IS A TENSOR
OF SIZE height x width x channels



CONVOLUTIONS CAN ALSO BE COMPUTED ACROSS CHANNELS (OR COLORS)

A COLOR IMAGE IS A TENSOR
OF SIZE height x width x channels



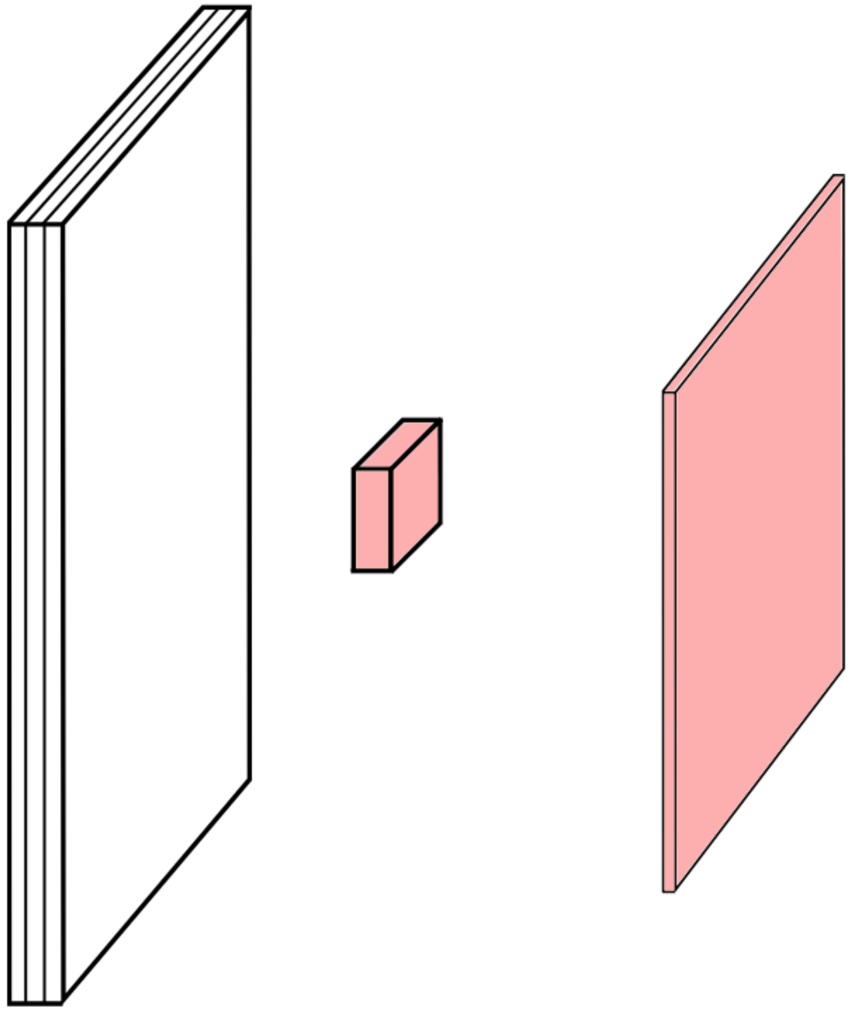


<u>THEN THE KERNEL HAS ALSO 3 CHANNELS</u>

IN ASTRONOMY …

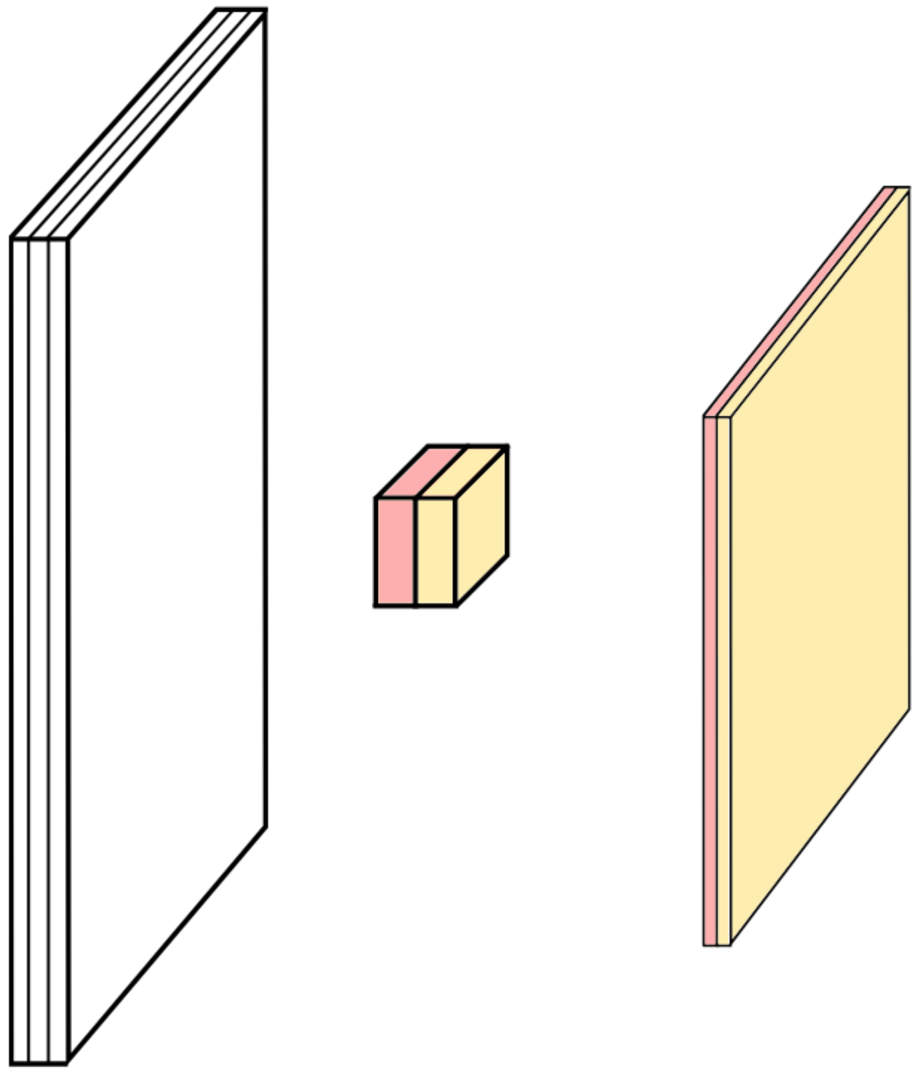
IT OPENS THE DOOR TO ANALYZE MULTIPLE FILTERS () SIMULTANEOUSLY

MULTIPLE CONVOLUTIONS WITH DIFFERENT KERNELS CAN BE PERFORMED



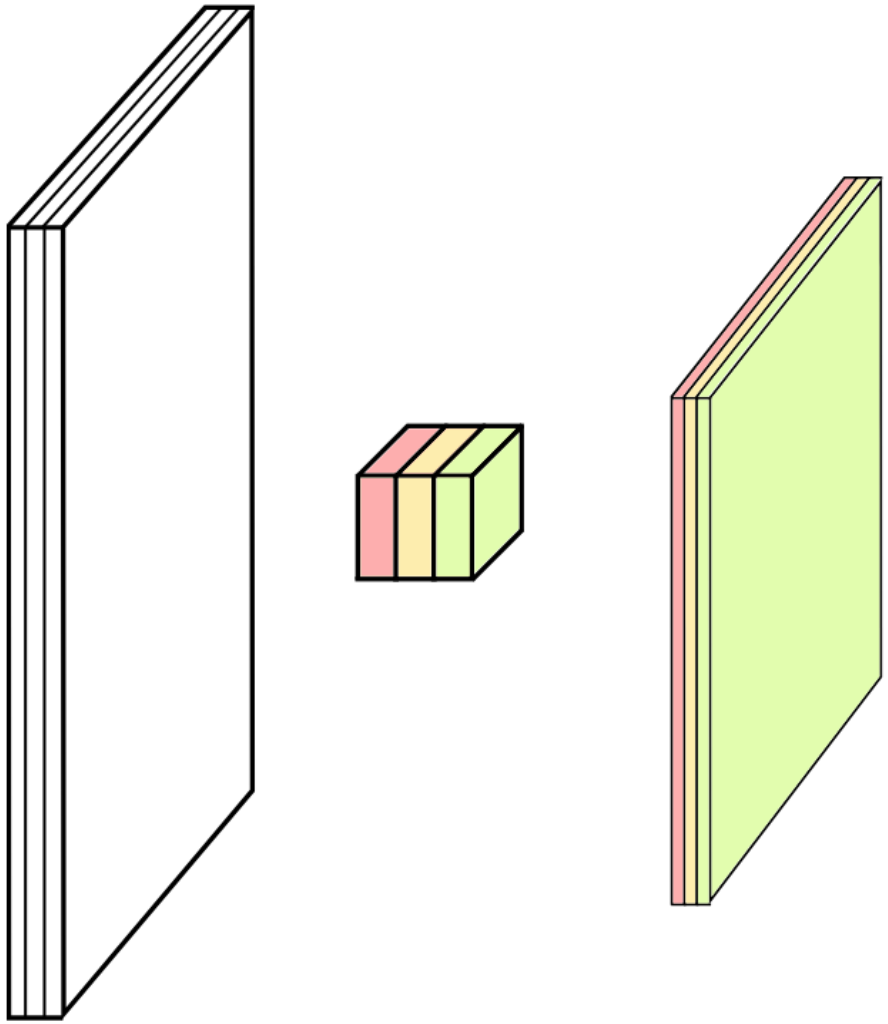
credit

MULTIPLE CONVOLUTIONS WITH DIFFERENT KERNELS CAN BE PERFORMED



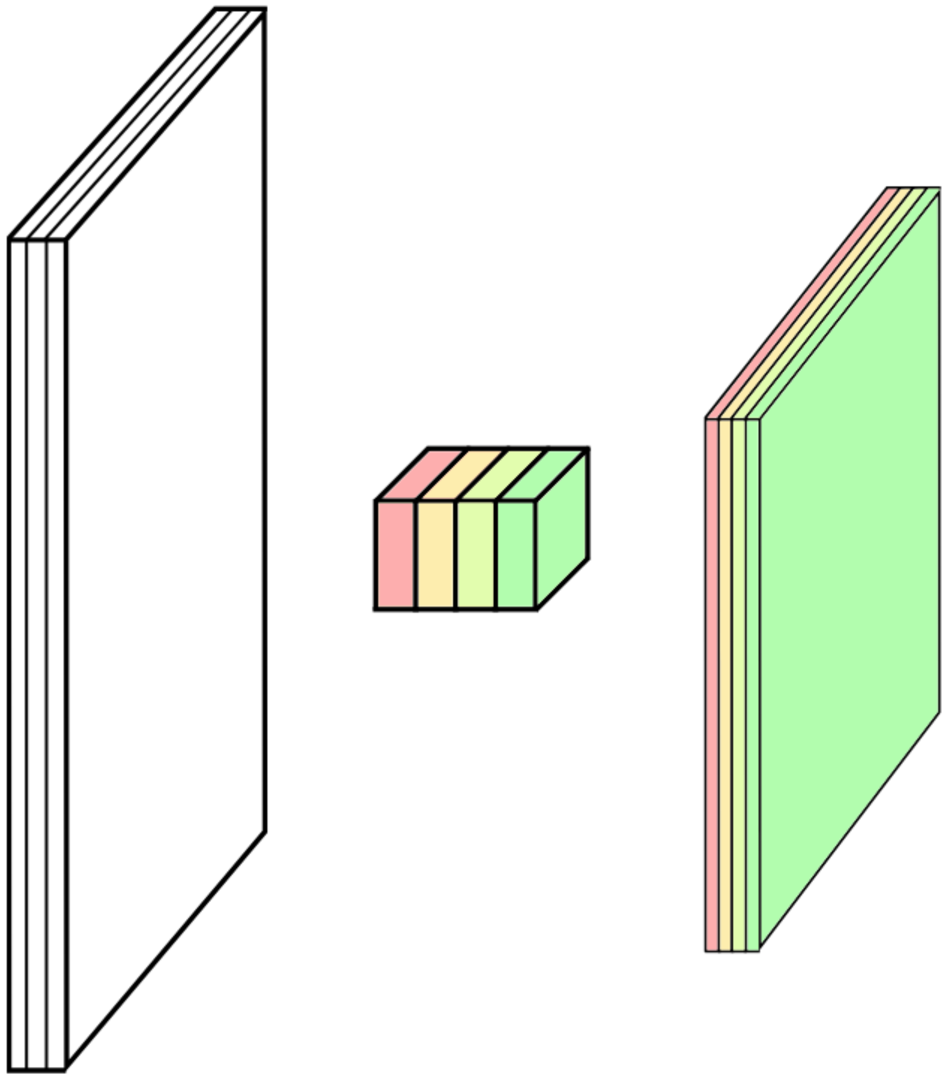
credit

MULTIPLE CONVOLUTIONS WITH DIFFERENT KERNELS CAN BE PERFORMED



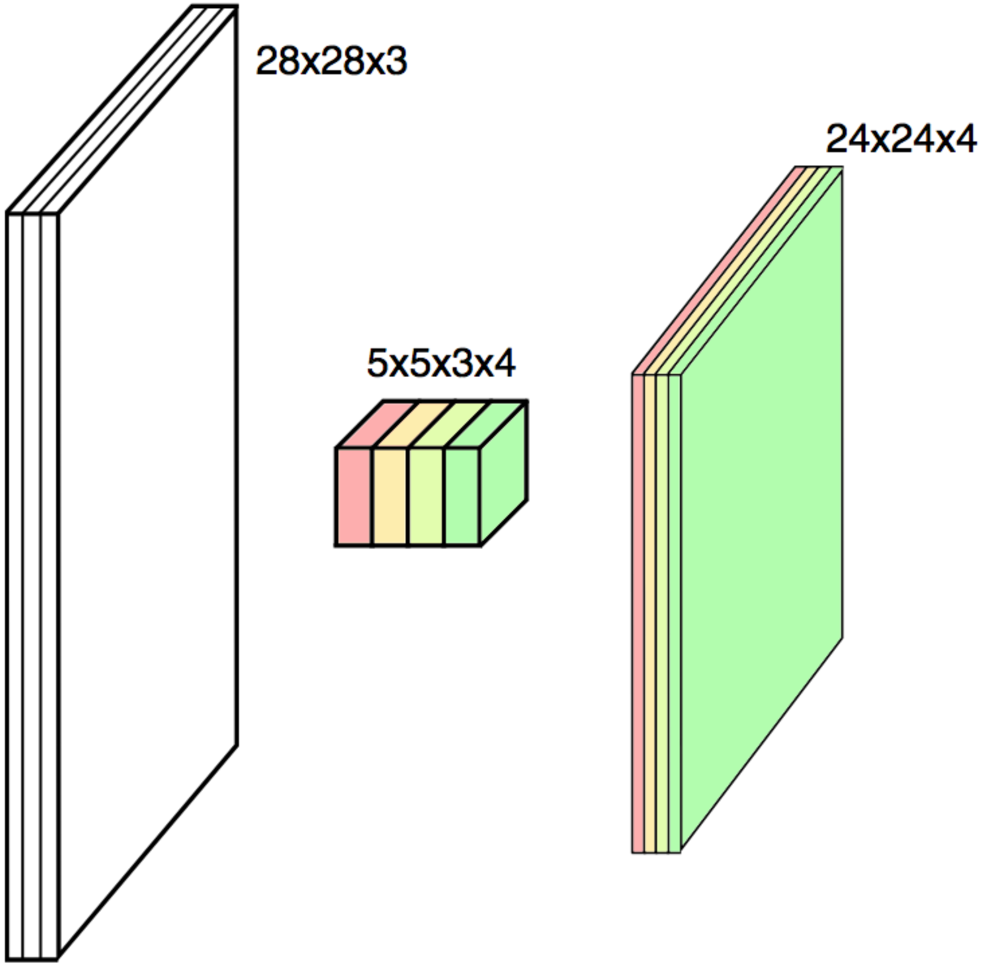
credit

MULTIPLE CONVOLUTIONS WITH DIFFERENT KERNELS CAN BE PERFORMED



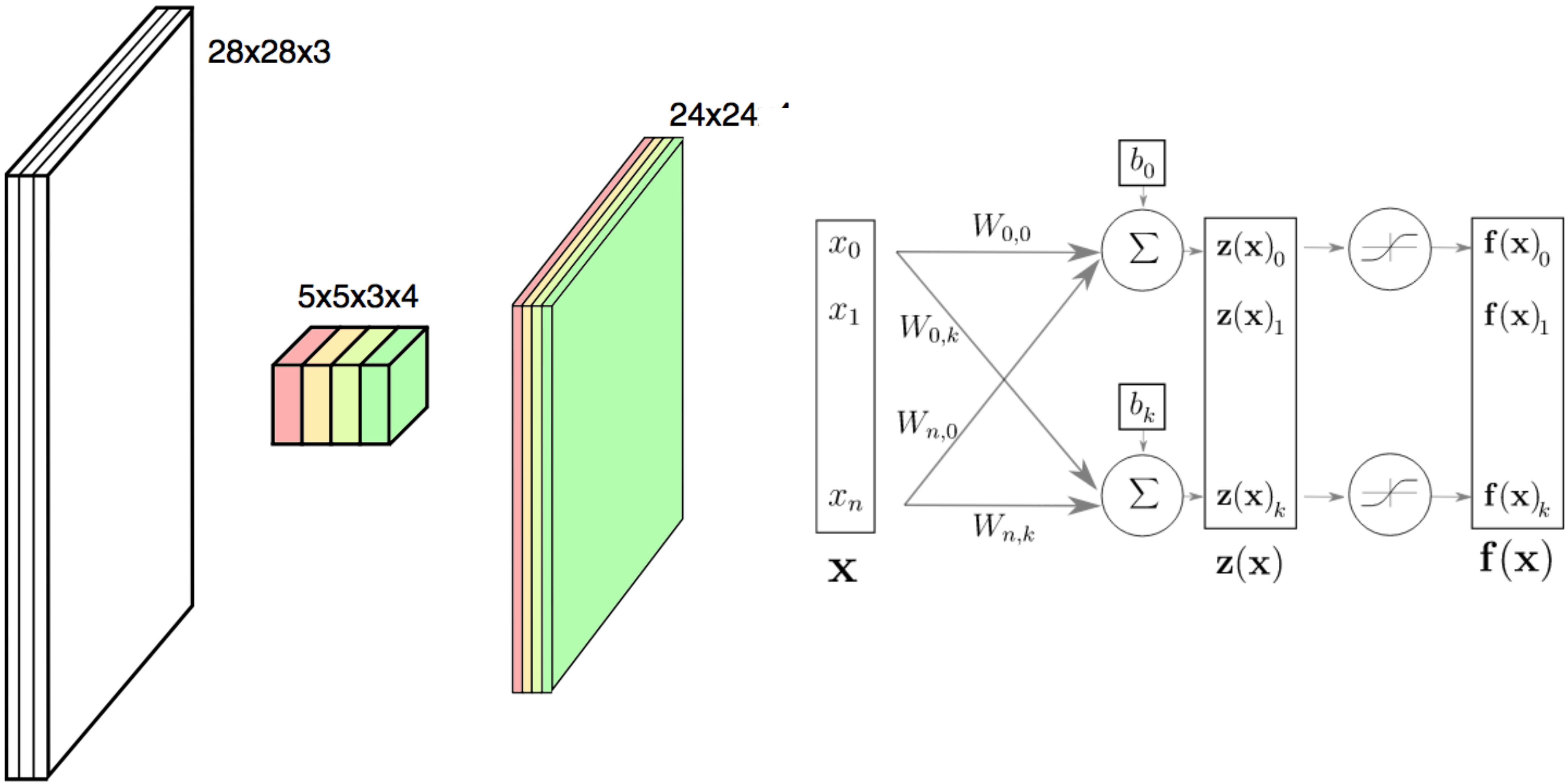
credit

MULTIPLE CONVOLUTIONS WITH DIFFERENT KERNELS CAN BE PERFORMED



credit

MULTIPLE CONVOLUTIONS WITH DIFFERENT KERNELS CAN BE PERFORMED



credit

IN KERAS…

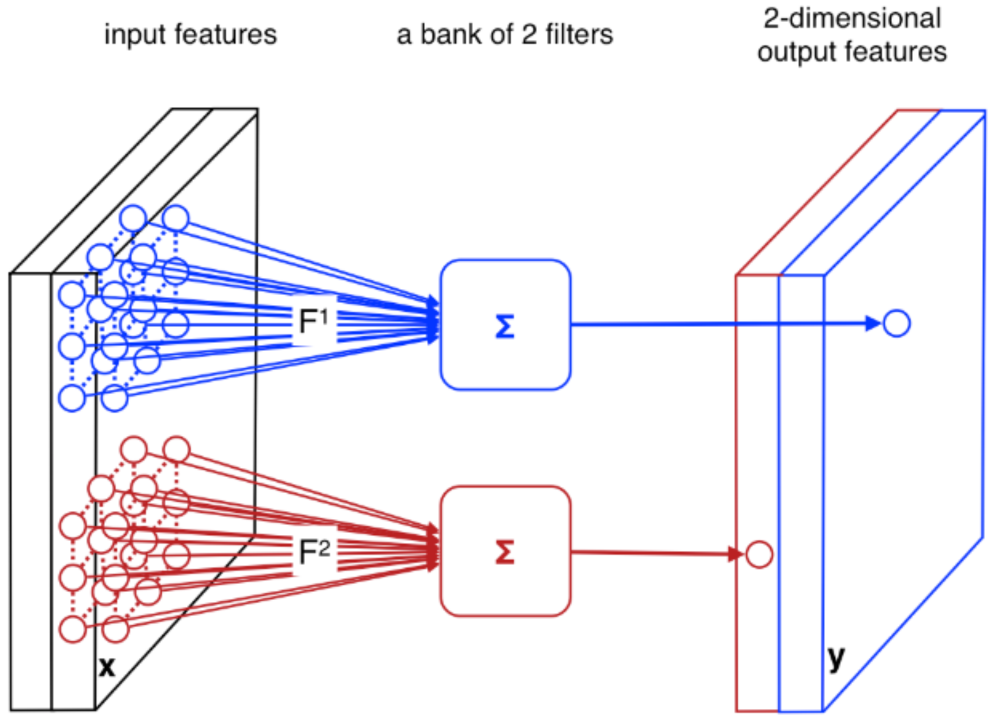
model = Sequential()

model.add(Convolution2D(4,5,5, activation="relu"))

sizeX, sizeY

depth

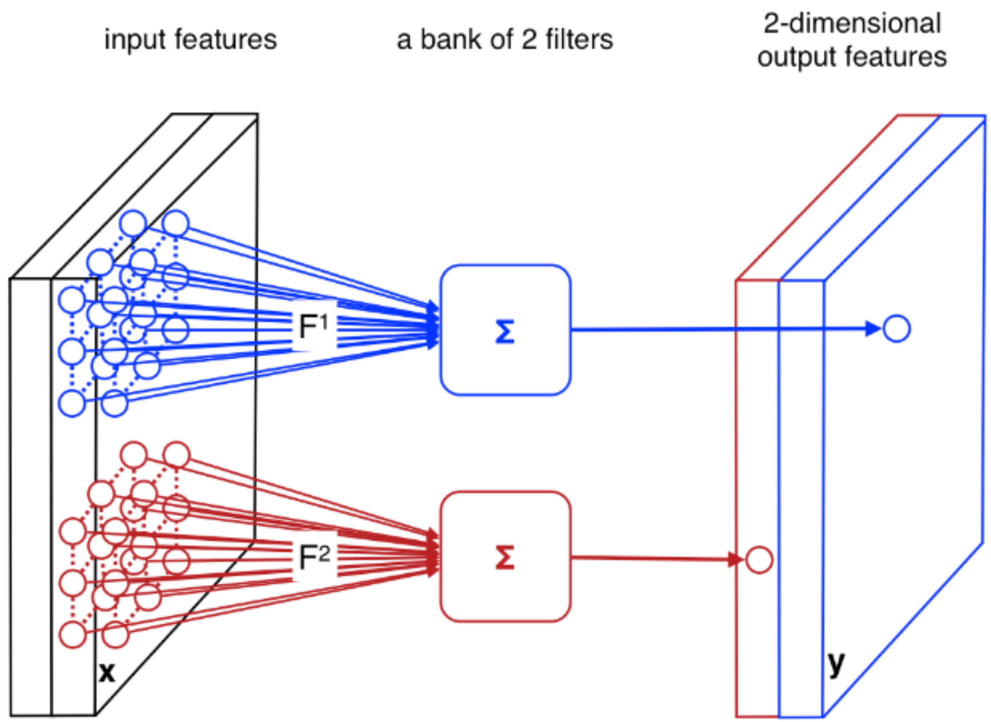
SINCE CONVOLUTIONS OUTPUT ONE SCALAR, THEY CAN BE SEEN AS AN INDIVIDUAL NEURON WITH A <u>RECEPTIVE FIELD LIMITED TO THE KERNEL DIMENSIONS</u>



Credit

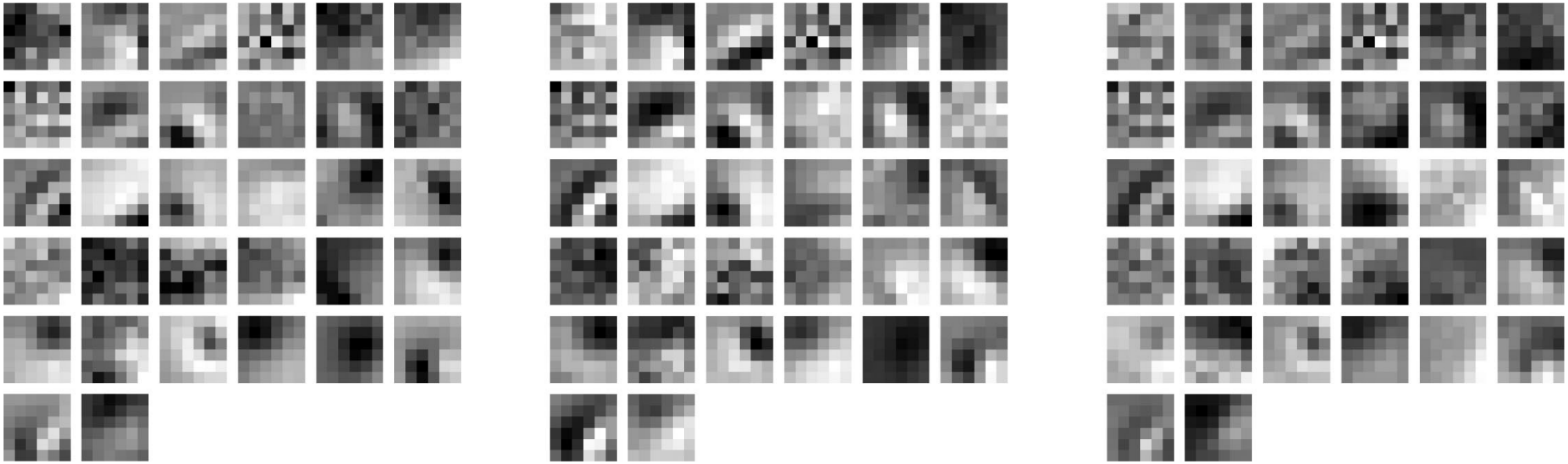
SINCE CONVOLUTIONS OUTPUT ONE SCALAR< THEY CAN BE SEEN AS AN INDIVIDUAL NEURON WITH A RECEPTIVE FIELD LIMITED TO THE KERNEL DIMENSIONS

THE SAME NEURON IS FIRED WITH DIFFERENT AREAS FROM THE INPUT

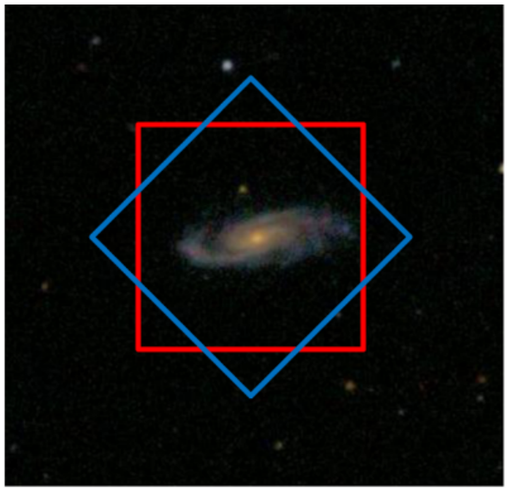


Credit

EXAMPLE OF 32 FILTERS LEARNED IN A CONVOLUTIONAL LAYER

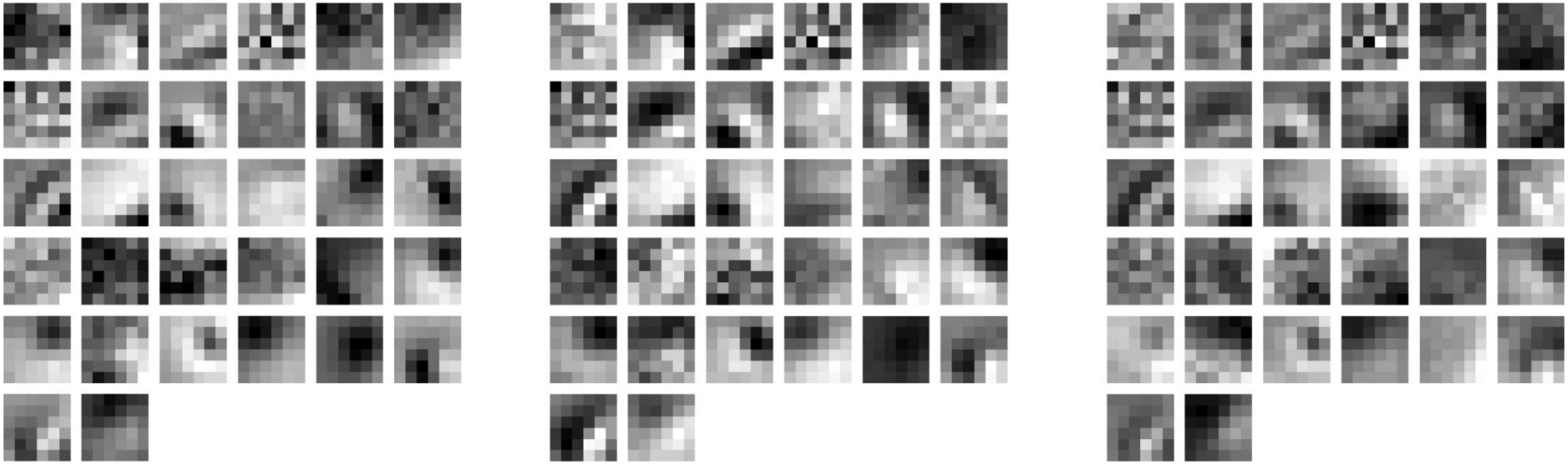


(a) red channel (b) green channel (c) blue channel

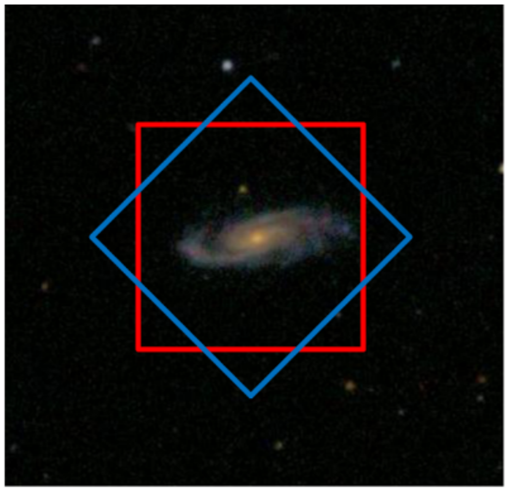


Dieleman+16

EXAMPLE OF 32 FILTERS LEARNED IN A CONVOLUTIONAL LAYER



(a) red channel (b) green channel (c) blue channel



Dieleman+16

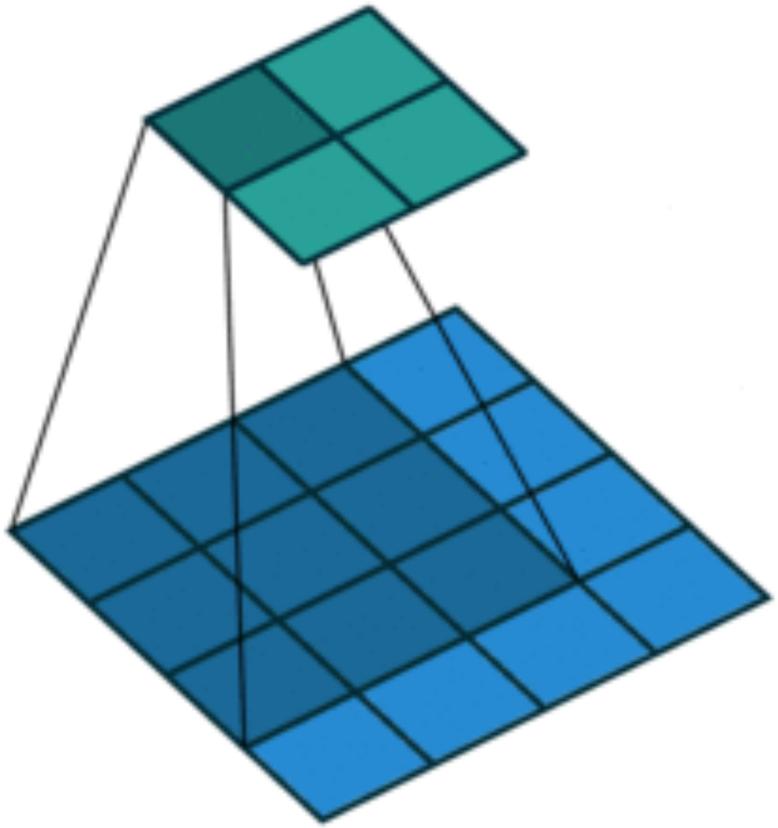
THESE ARE CALLED **FEATURE MAPS**

ESTIMATING SHAPES AND NUMBER OF PARAMETERS

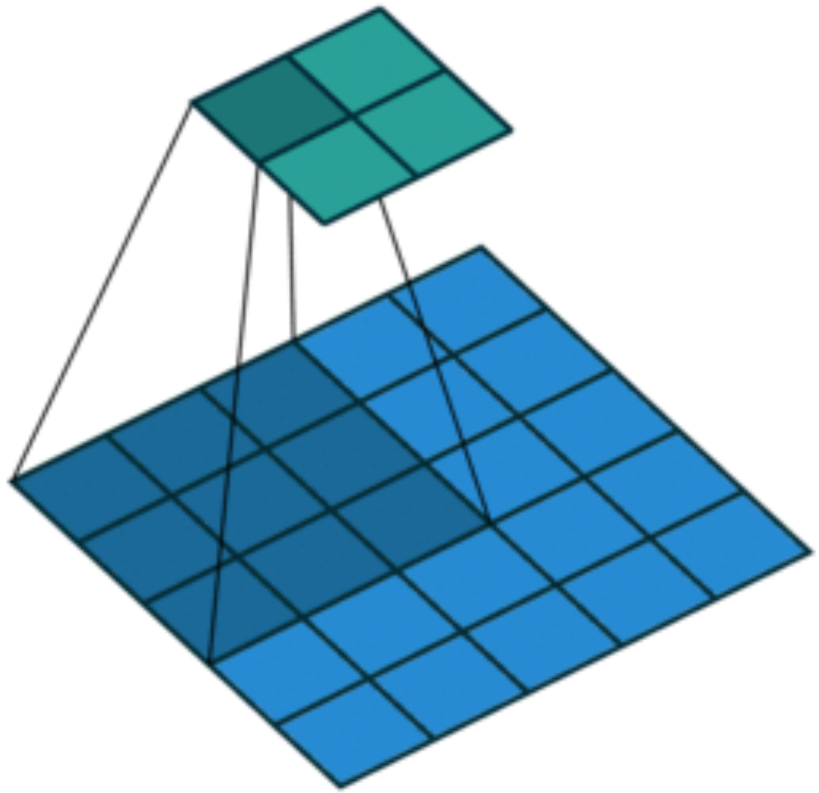
KERNEL SHAPE:	PADDING:	STRIDES:
(F, F, C^i, C^o)	P	S

OUTPUT SIZE: $W_0 = (W^i - F + 2P)/S + 1$

OPTIONS: **STRIDES**

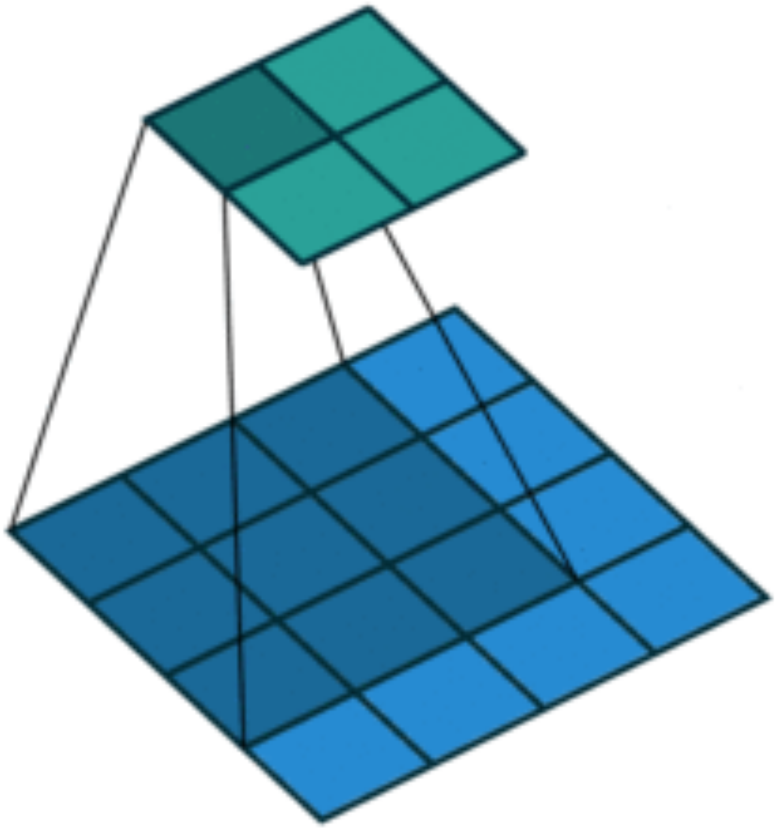


NO STRIDES

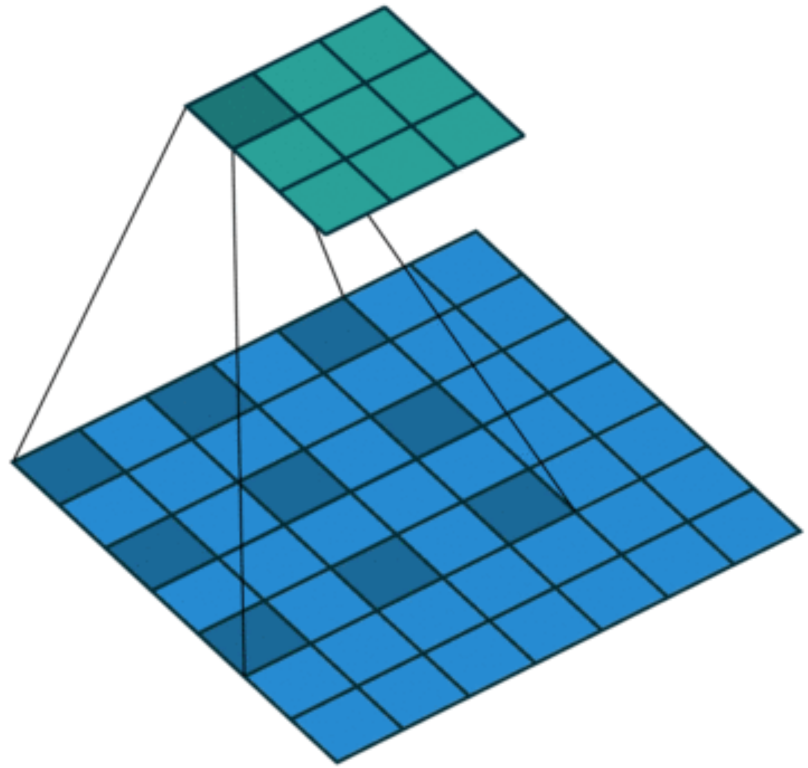


STRIDES

OPTIONS: **DILATION**

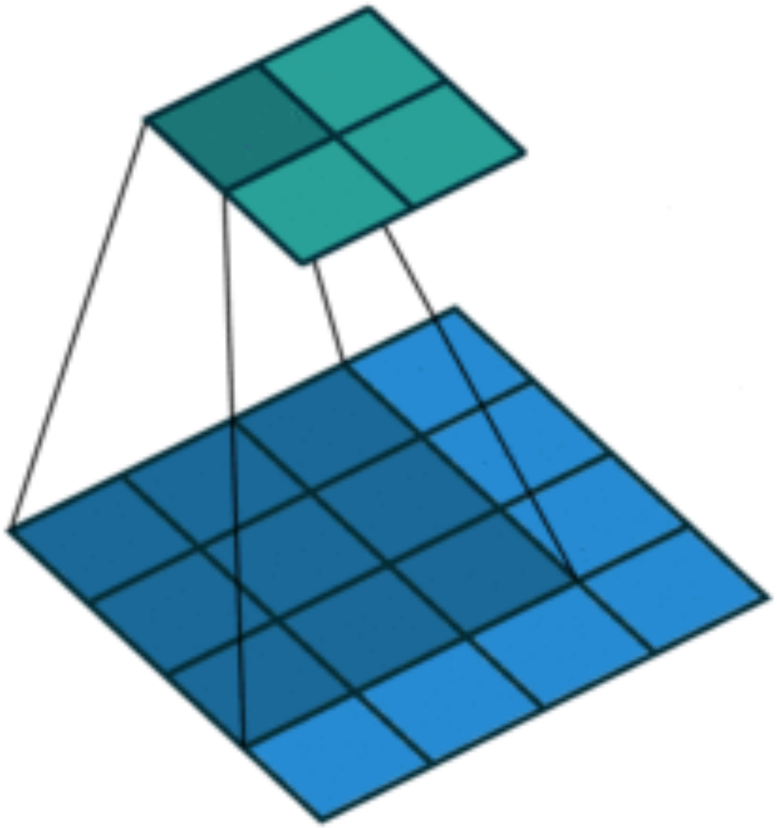


NO STRIDES

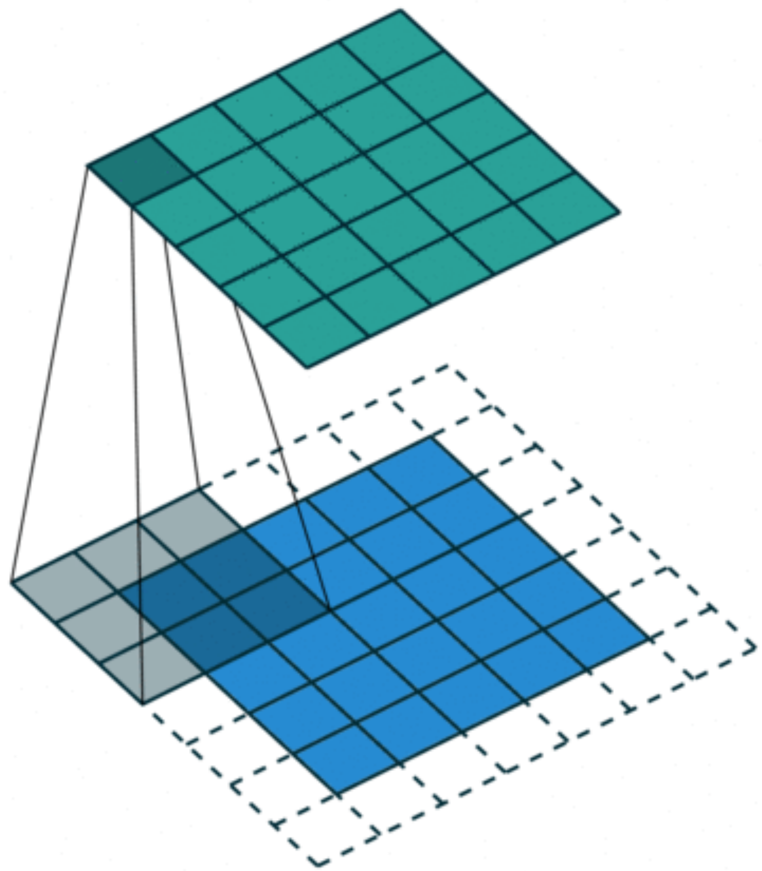


DILATION

<u>OPTIONS:</u> **PADDING**



NO STRIDES



PADDING

ESTIMATING SHAPES AND NUMBER OF PARAMETERS

KERNEL SHAPE:	PADDING:	STRIDES:
(F, F, C^i, C^o)	P	S

OUTPUT SIZE: $W_0 = (W^i - F + 2P)/S + 1$

NUMBER OF PARAMETERS: $(F \times F \times C^i + 1) \times C^o$

ESTIMATING SHAPES AND NUMBER OF PARAMETERS

KERNEL SHAPE:	PADDING:	STRIDES:
(F, F, C^i, C^o)	P	S

OUTPUT SIZE: $W_0 = (W^i - F + 2P)/S + 1$

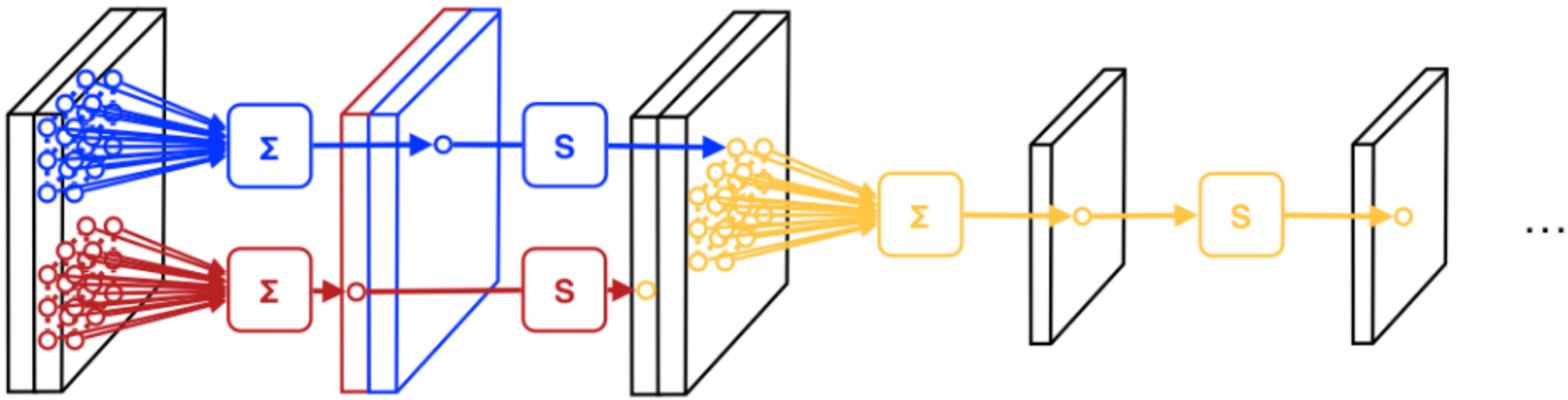
NUMBER OF PARAMETERS: $(F \times F \times C^i + 1) \times C^o$

→ the number of parameters increases fast!

32 filters of 5*5 on a color image —> 2432 parameters to learn

DOWNSAMPLING

DOWNSAMPLING IS APPLIED TO REDUCE THE OVERALL SIZE OF TENSORS



POOLING

CONVOLUTIONS ARE OFTEN FOLLOWED BY AN OPERATION OF DOWNSAMPLING [POOLING]

VERY SIMPLE OPERATION - ONLY ONE OUT OF EVERY N PIXELS ARE KEPT

OFTEN MATCHED WITH AN INCREASE OF THE FEATURE CHANNELS

TYPES OF POOLING

<u>SUM POOLING</u>

$$y = \sum x_{uv}$$

<u>SQUARE SUM POOLING</u>

$$y = \sqrt{\sum x_{uv}^2}$$

<u>MAX POOLING</u>

$$y = max(x_{uv})$$

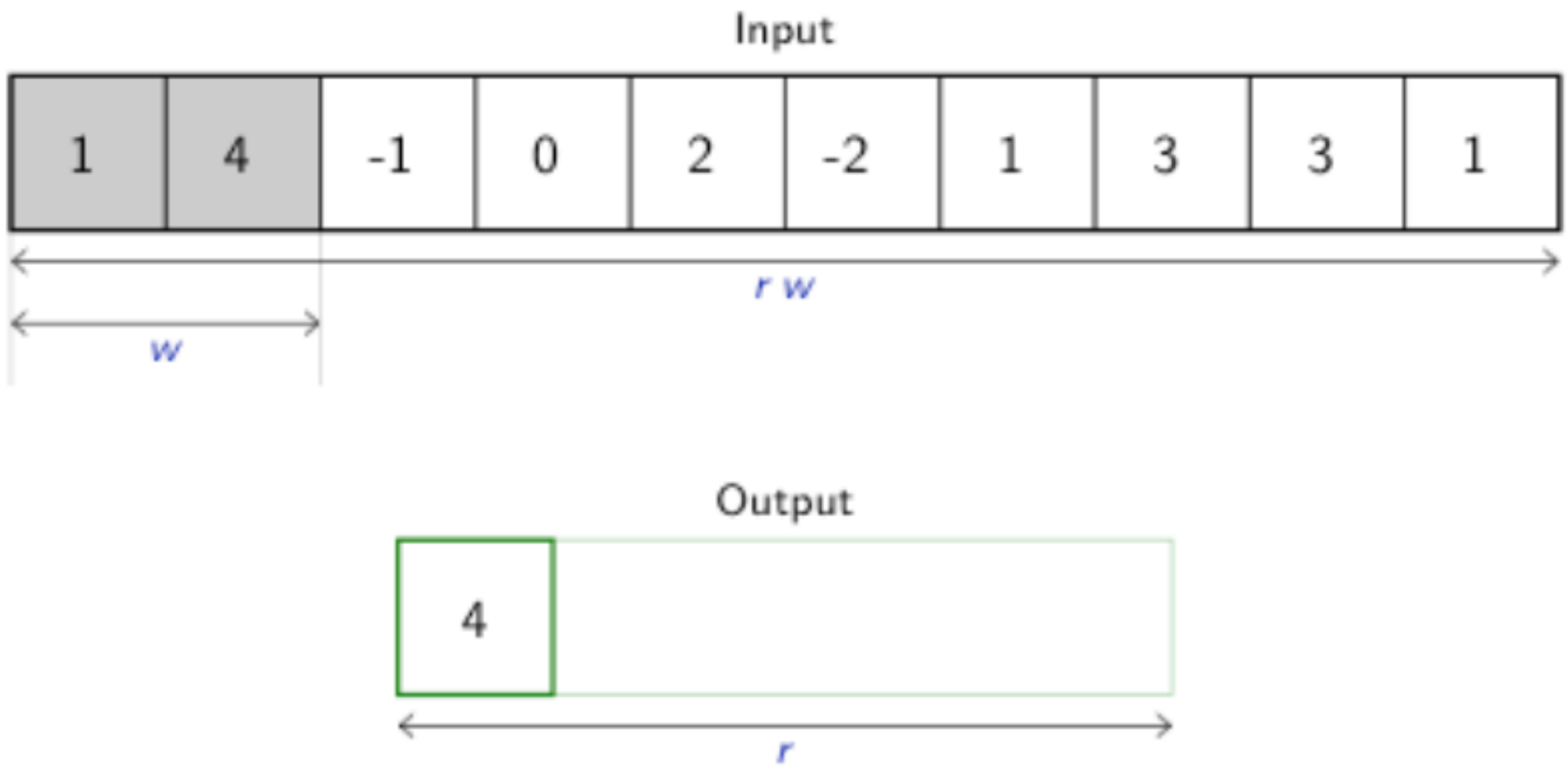
TYPES OF POOLING

<u>SUM POOLING</u> $$y = \sum x_{uv}$$

<u>SQUARE SUM POOLING</u> $$y = \sqrt{\sum x_{uv}^2}$$

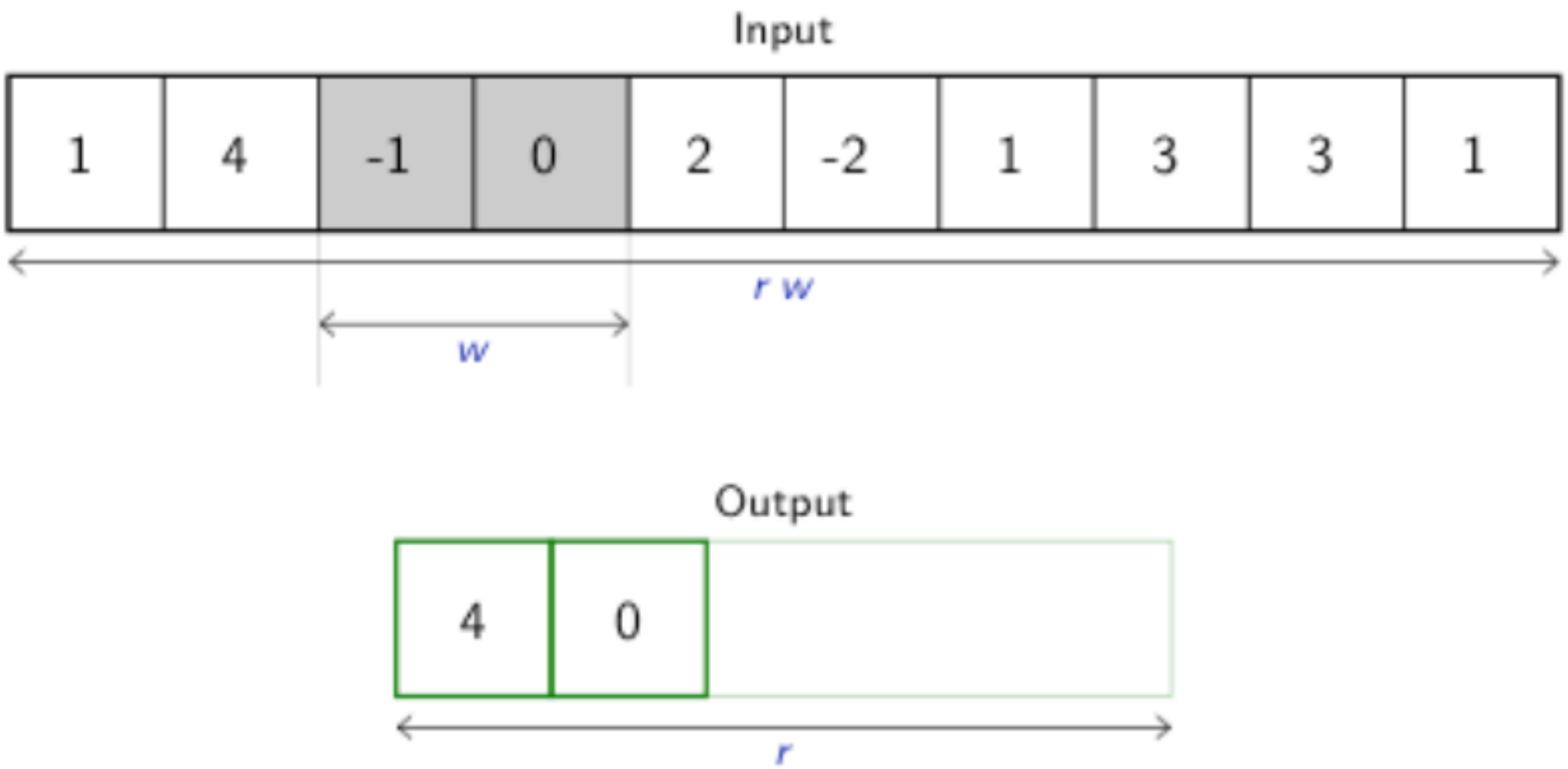
<u>MAX POOLING</u> $$y = max(x_{uv})$$

MAX POOLING 1D



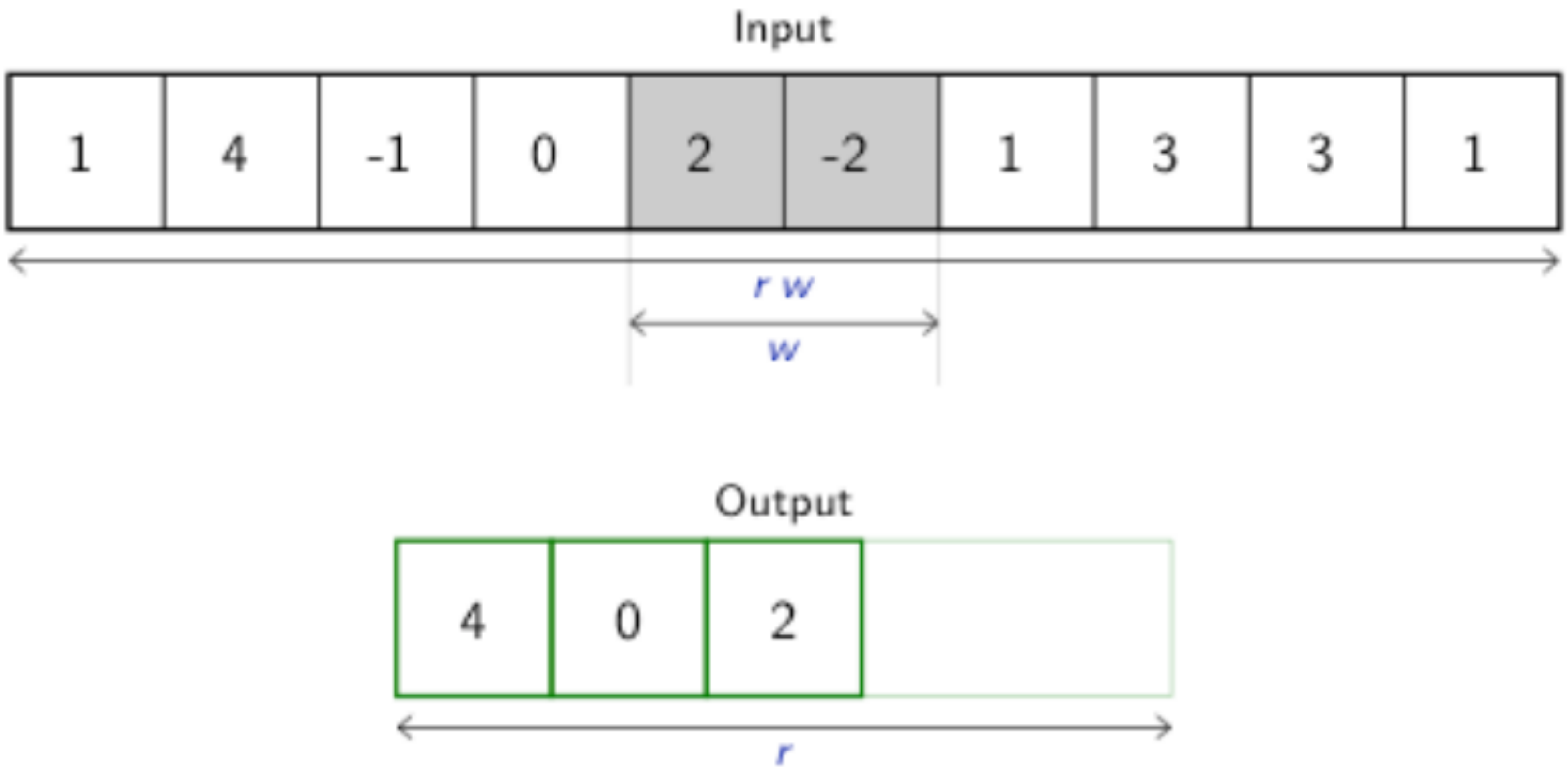
Credit: F. Fleuret

MAX POOLING 1D



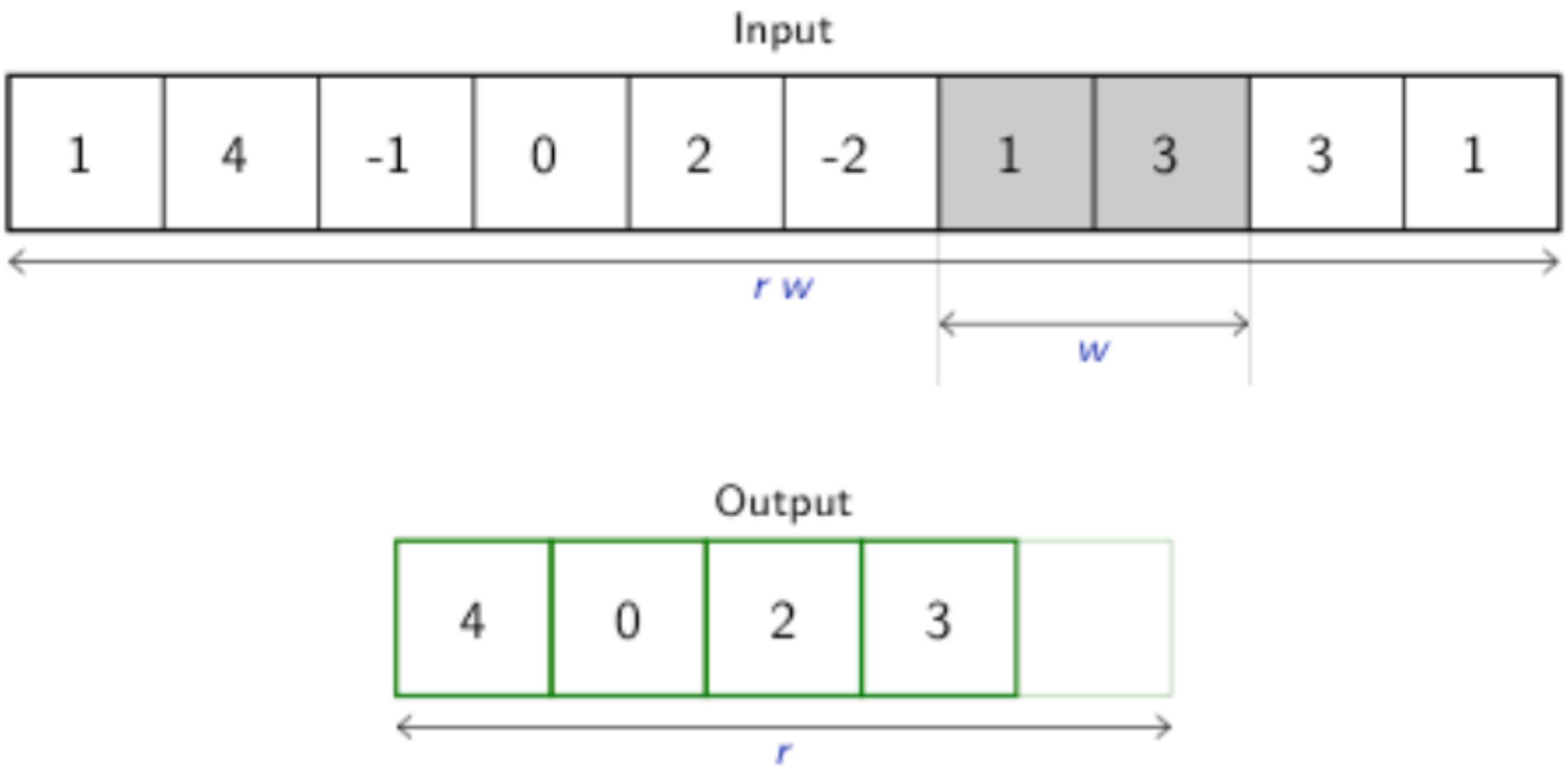
Credit: F. Fleuret

MAX POOLING 1D



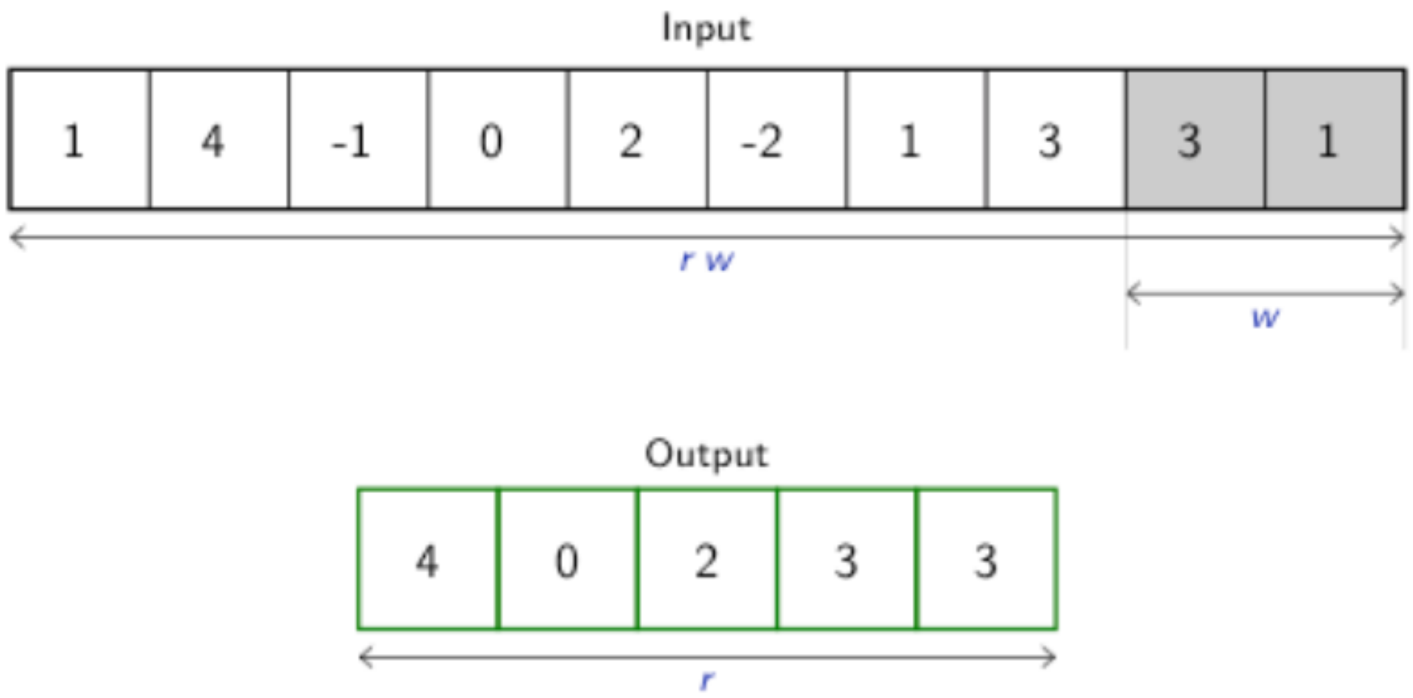
Credit: F. Fleuret

MAX POOLING 1D



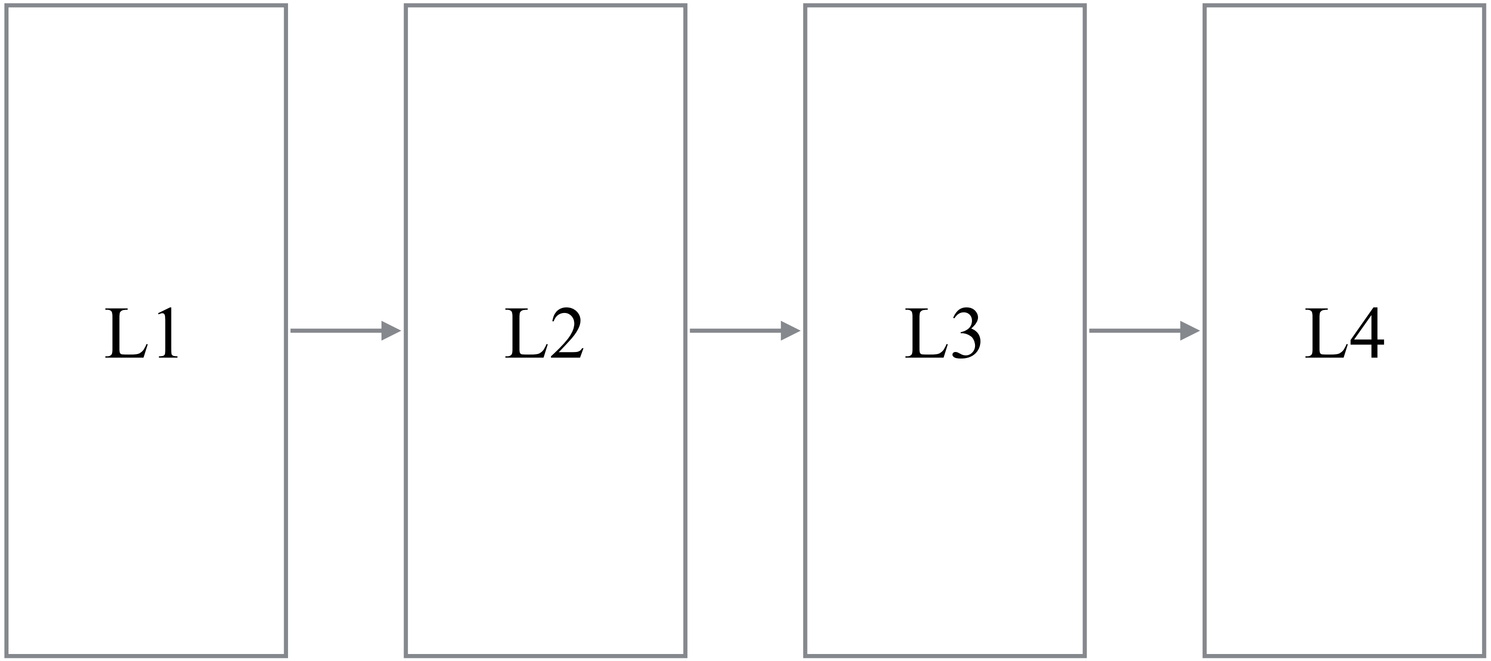
Credit: F. Fleuret

MAX POOLING 1D



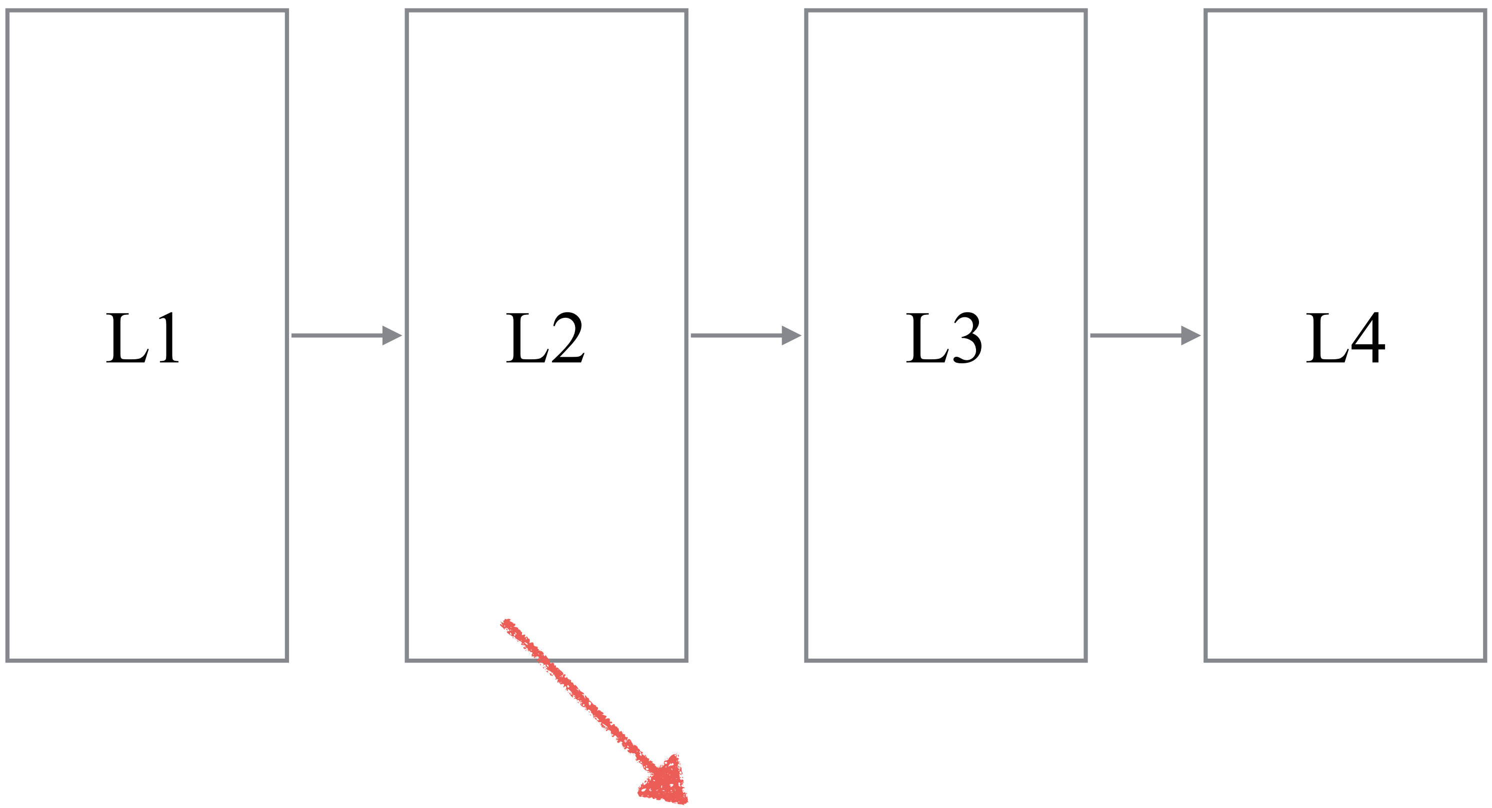
Credit: F. Fleuret

CONVNET OR CNN



A CONCATENATION OF MULTIPLE
CONVOLUTIONAL BLOCKS

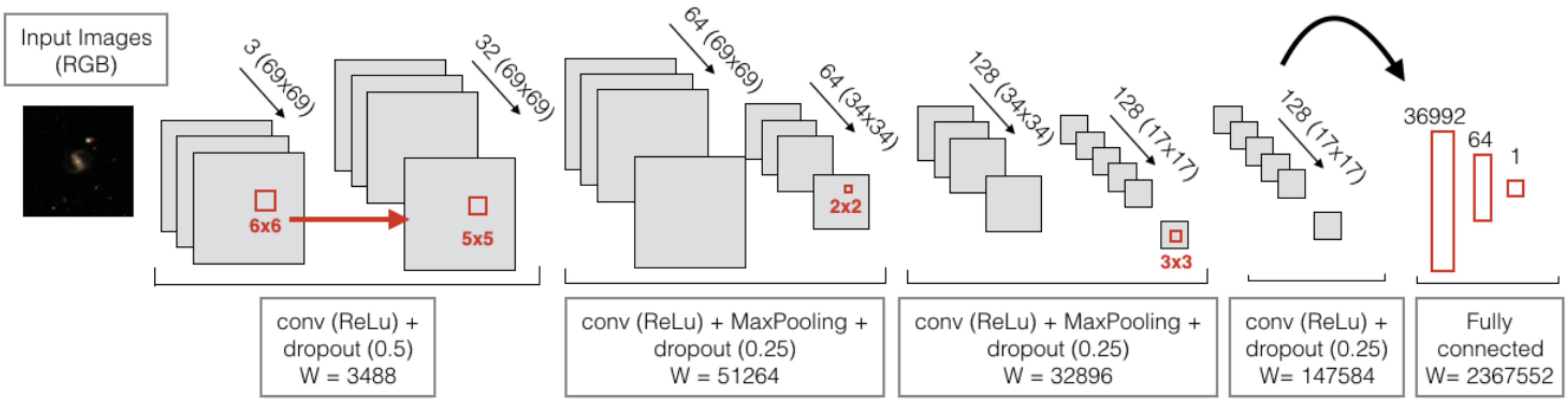
CONVNET OR CNN



EACH BLOCK TYPICALLY MADE OF:

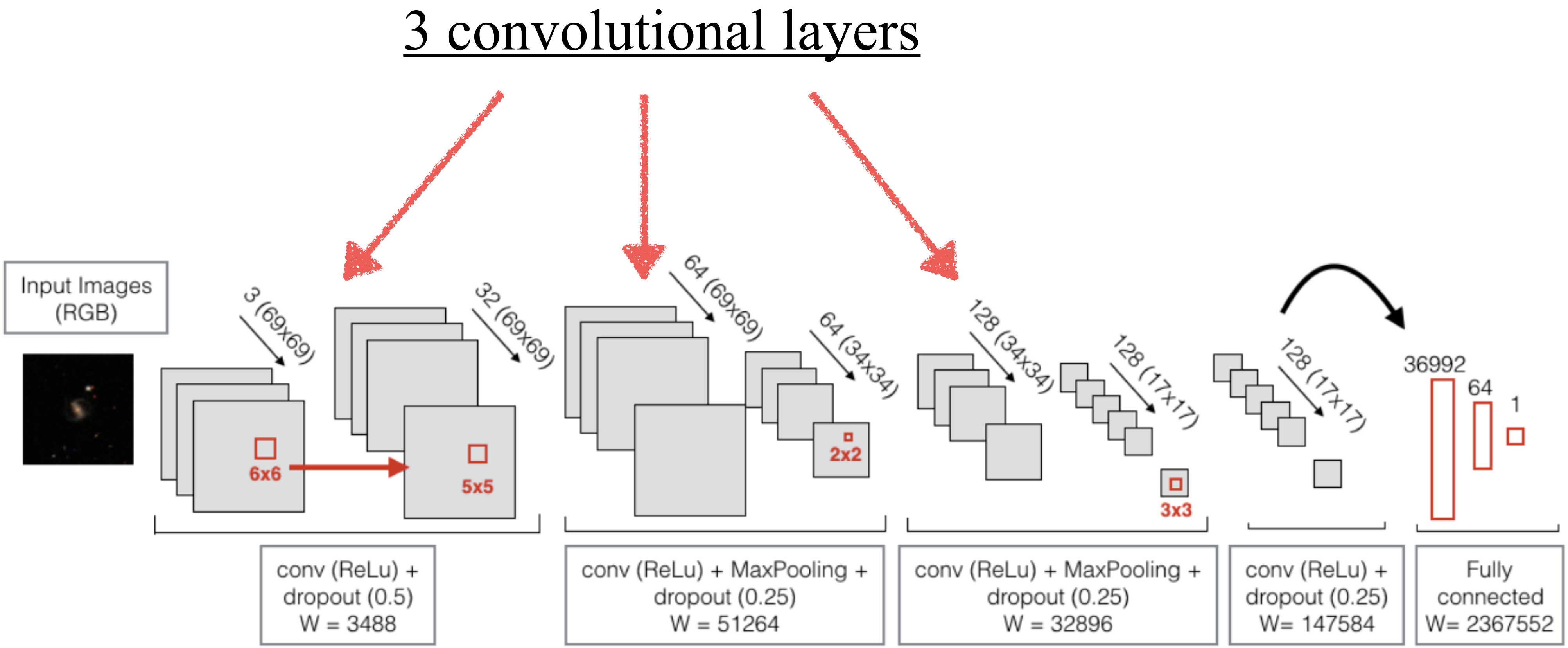
CONV ACTIVATION POOLING (+dropout for training)

EXAMPLE OF VERY SIMPLE CNN



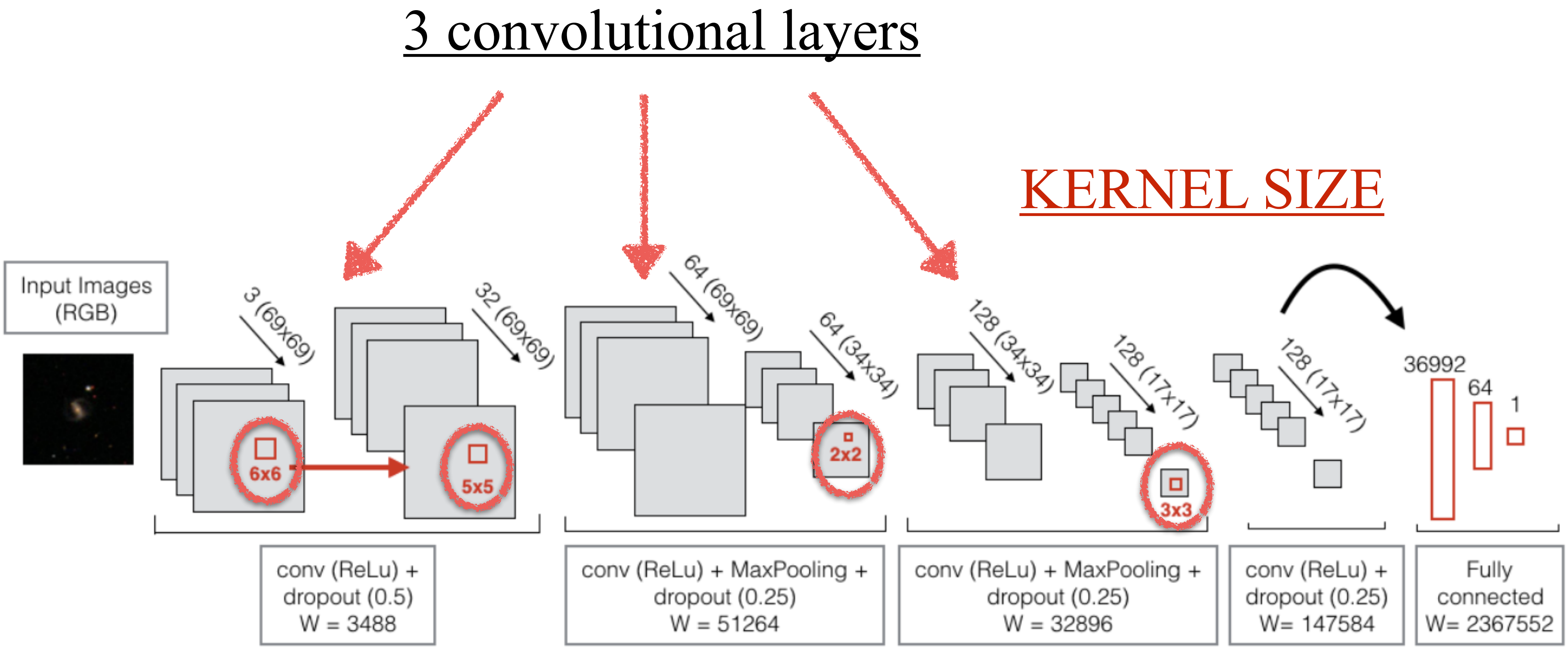
Dominguez-Sanchez+18

EXAMPLE OF VERY SIMPLE CNN



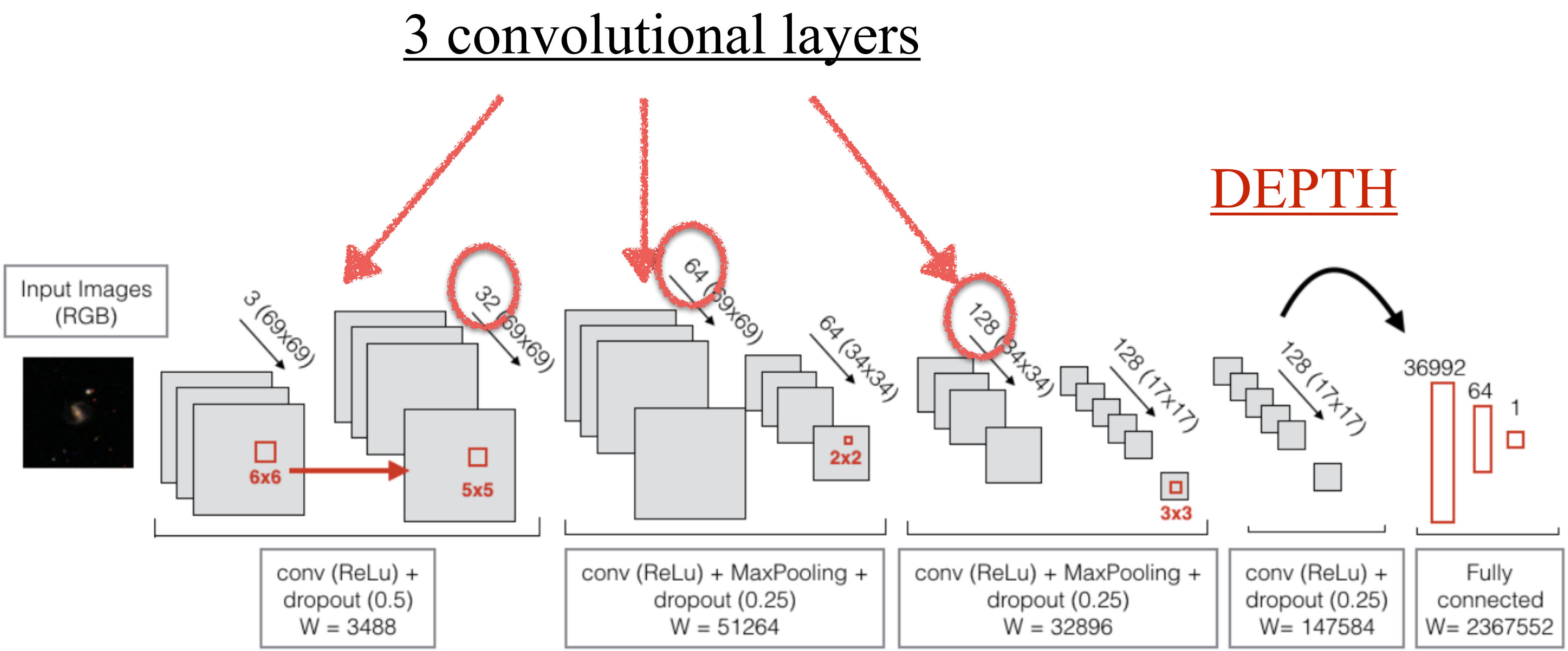
Dominguez-Sanchez+18

EXAMPLE OF VERY SIMPLE CNN



Dominguez-Sanchez+18

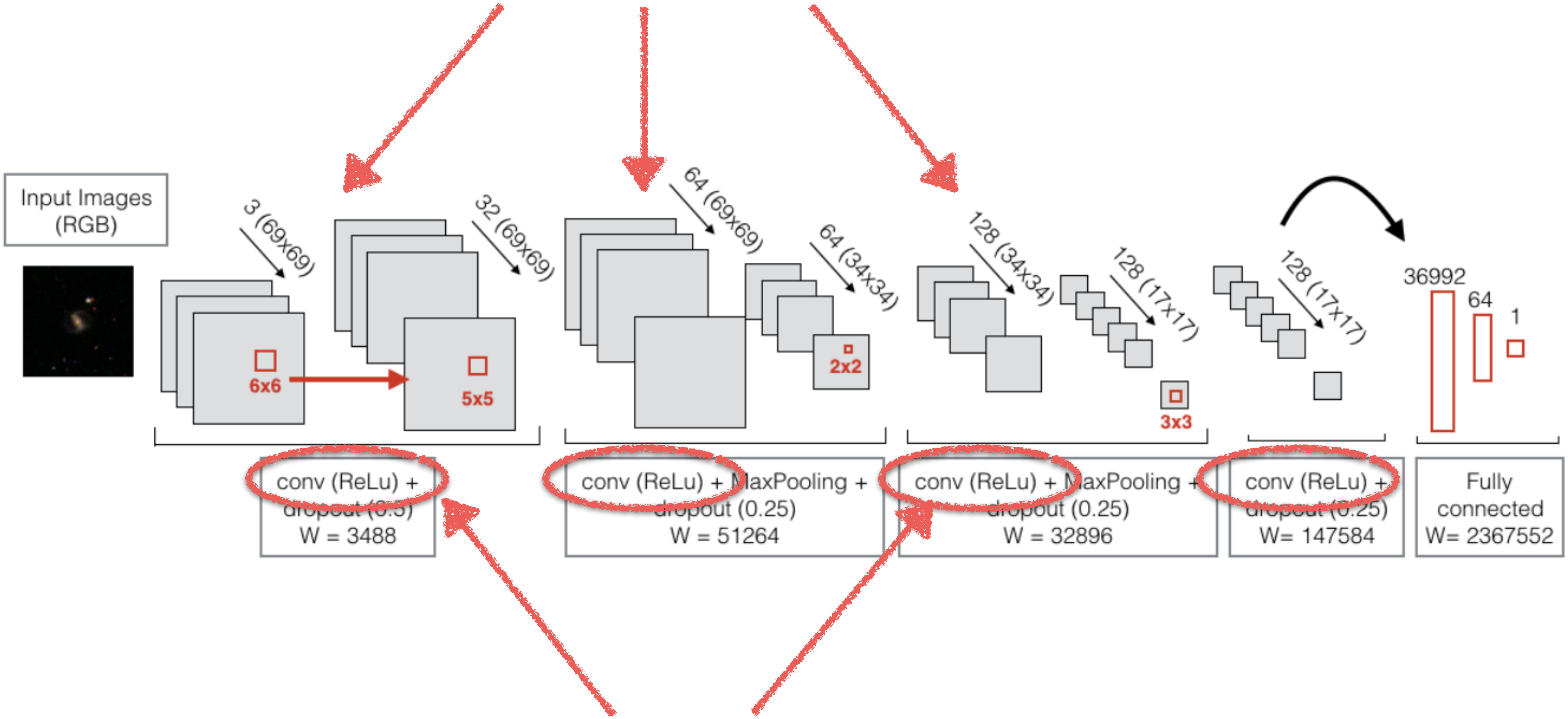
EXAMPLE OF VERY SIMPLE CNN



Dominguez-Sanchez+18

EXAMPLE OF VERY SIMPLE CNN

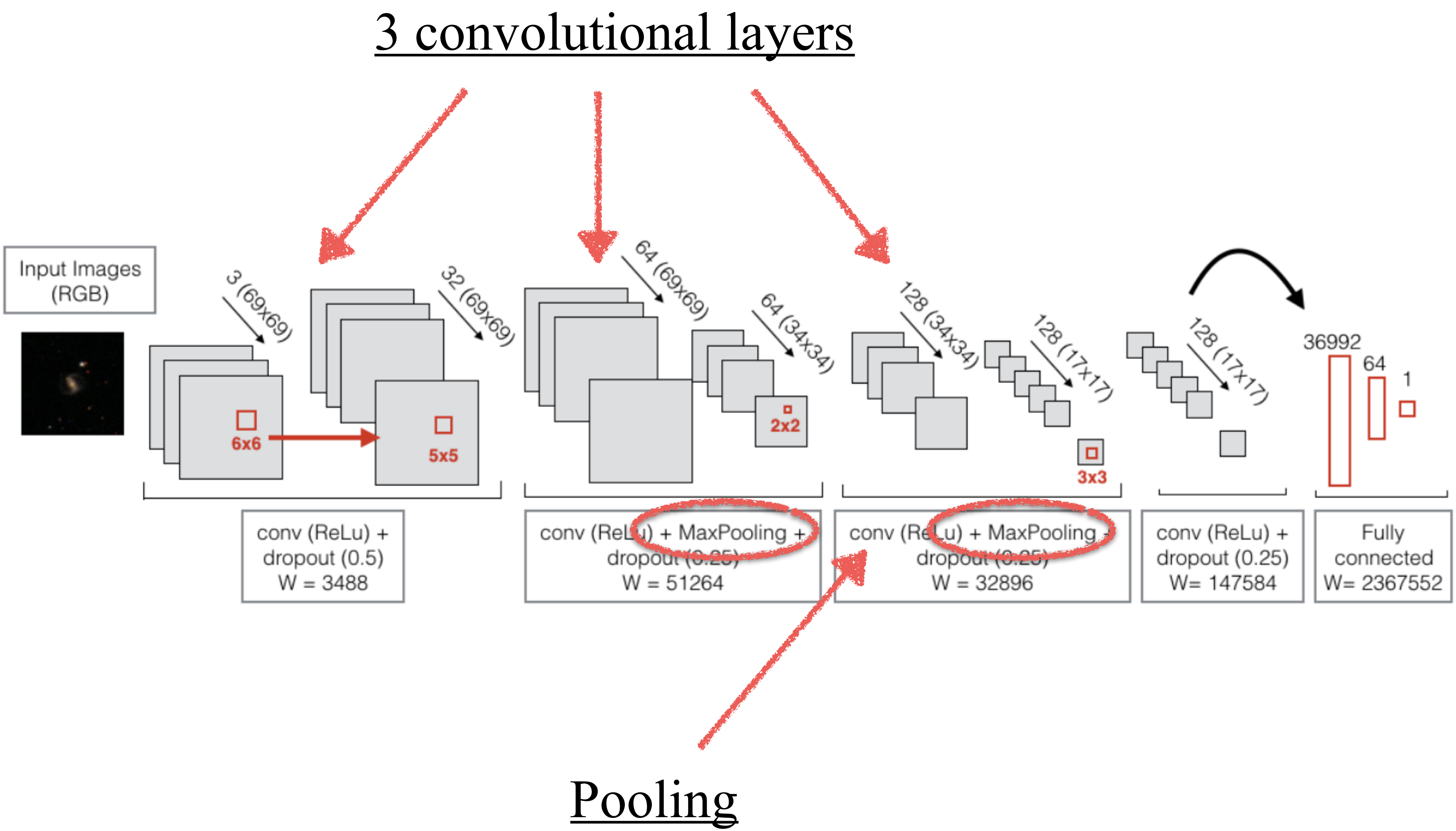
3 convolutional layers



ReLu activation

Dominguez-Sanchez+18

EXAMPLE OF VERY SIMPLE CNN

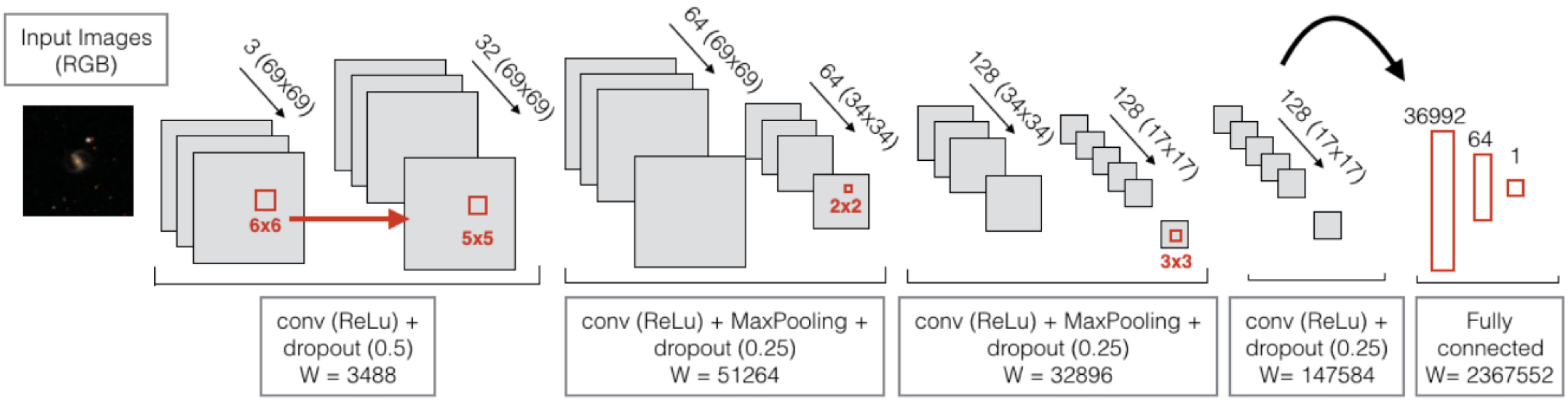


Dominguez-Sanchez+18

EXAMPLE OF VERY SIMPLE CNN

OVERALL:

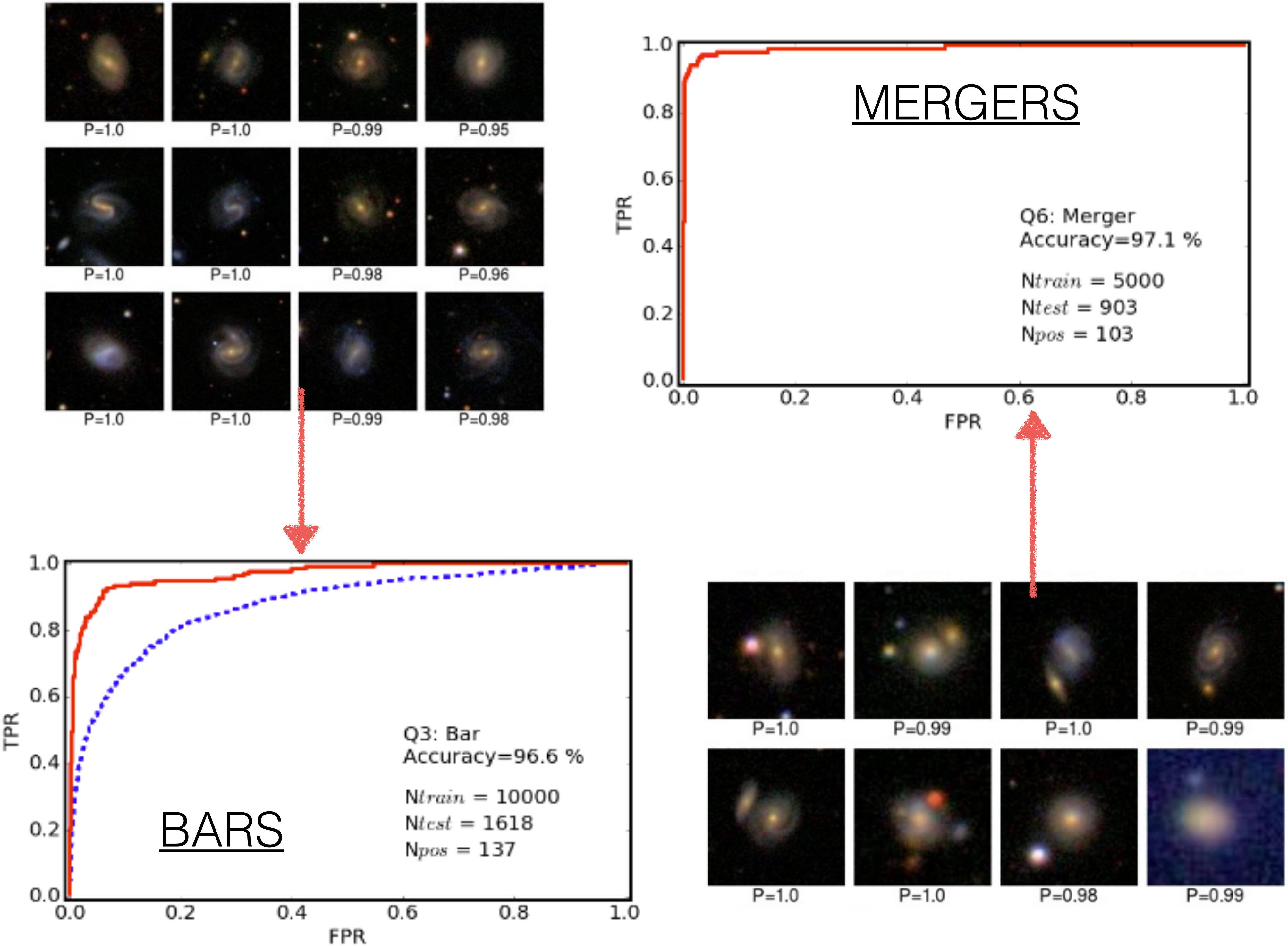
- decrease of tensor size
- increase of depth



Dominguez-Sanchez+18

IMPLEMENTATION IN KERAS

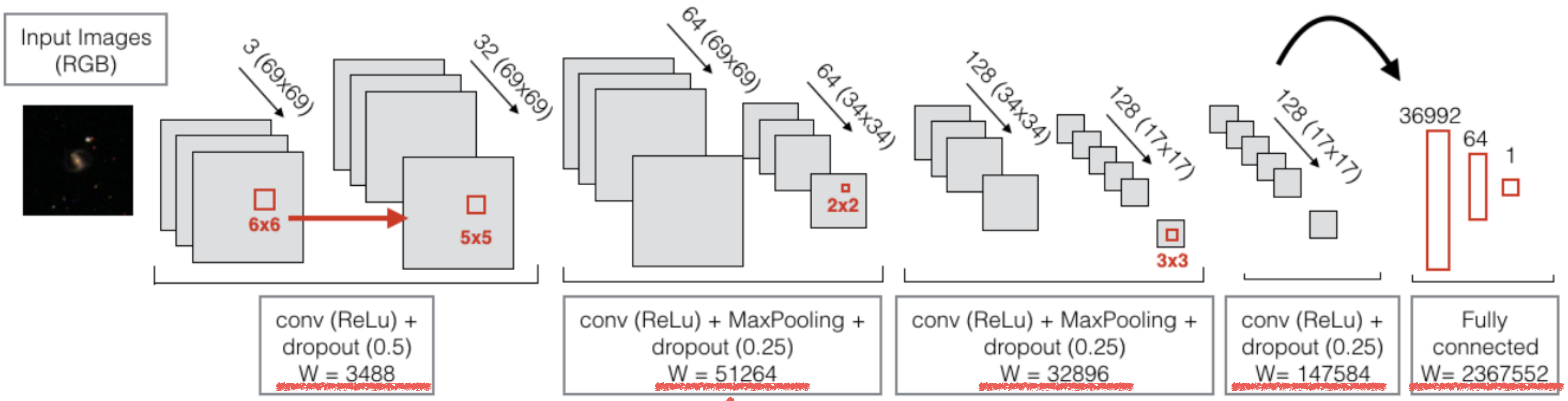
```
#========= Model definition=======

#Convolutional Layers

model = Sequential()
model.add(Convolution2D(32, 6,6, border_mode='same',
                    input_shape=(img_channels, img_rows, img_cols)))
model.add(Activation('relu'))
model.add(Dropout(0.5))

model.add(Convolution2D(64, 5, 5, border_mode='same'))
model.add(Activation('relu'))

model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Convolution2D(128, 2, 2, border_mode='same'))
model.add(Activation('relu'))

model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Convolution2D(128, 3, 3, border_mode='same'))
model.add(Activation('relu'))

model.add(Dropout(0.25))

#Fully Connected start here
#-------------------------------#

model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dropout(.5))
model.add(Dense(1, init='uniform', activation='sigmoid'))

print("Compilation...")

model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
```


P=1.0
P=1.0
P=0.99
P=0.95
P=1.0
P=1.0
P=0.98
P=0.96
P=1.0
P=1.0
P=0.99
P=0.98
MERGERS
Q6: Merger
Accuracy=97.1 %
Ntrain = 5000
Ntest = 903
Npos = 103
TPR
FPR
0.0
0.2
0.4
0.6
0.8
1.0
BARS
Q3: Bar
Accuracy=96.6 %
Ntrain = 10000
Ntest = 1618
Npos = 137
P=1.0
P=0.99
P=1.0
P=0.99
P=1.0
P=1.0
P=0.98
P=0.99

EXAMPLE OF VERY SIMPLE CNN

OVERALL:

- decrease of tensor size
- increase of depth

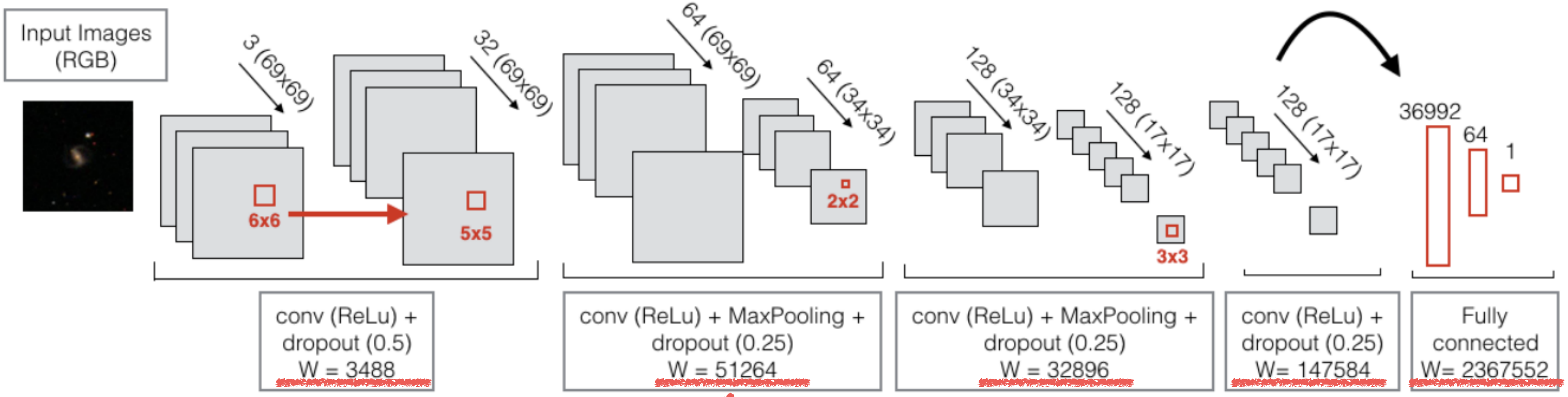


Dominguez-Sanchez+18

Number of parameters

EXAMPLE OF VERY SIMPLE CNN

OVERALL:
- decrease of tensor size
- increase of depth



Dominguez-Sanchez+18

2 million of parameters for this very simple network!

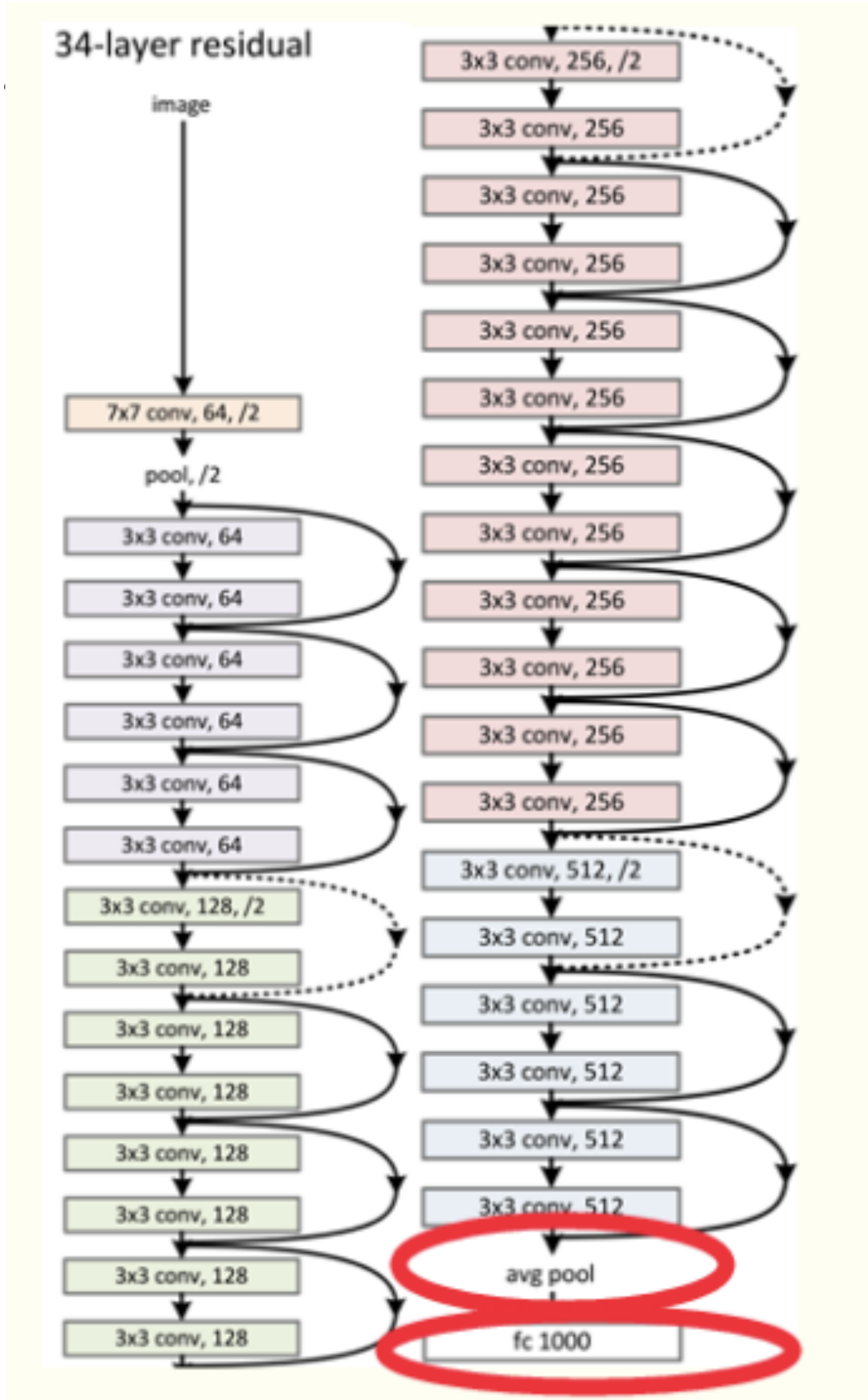
CHECKING THE NUMBER OF PARAMETERS / LAYERS WITH KERAS

model.summary()

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 1, 16, 112, 112)   0
_________________________________________________________________
conv3d_1 (Conv3D)            (None, 16, 16, 112, 112)  448
_________________________________________________________________
batch_normalization_1 (Batch (None, 16, 16, 112, 112)  448
_________________________________________________________________
activation_1 (Activation)    (None, 16, 16, 112, 112)  0
_________________________________________________________________
max_pooling3d_1 (MaxPooling3 (None, 16, 8, 56, 56)     0
_________________________________________________________________
conv3d_2 (Conv3D)            (None, 32, 8, 56, 56)     13856
_________________________________________________________________
batch_normalization_2 (Batch (None, 32, 8, 56, 56)     224
_________________________________________________________________
activation_2 (Activation)    (None, 32, 8, 56, 56)     0
_________________________________________________________________
max_pooling3d_2 (MaxPooling3 (None, 32, 4, 28, 28)     0
_________________________________________________________________
conv3d_3 (Conv3D)            (None, 64, 4, 28, 28)     55360
_________________________________________________________________
batch_normalization_3 (Batch (None, 64, 4, 28, 28)     112
_________________________________________________________________
activation_3 (Activation)    (None, 64, 4, 28, 28)     0
_________________________________________________________________
max_pooling3d_3 (MaxPooling3 (None, 64, 2, 14, 14)     0
_________________________________________________________________
activation_12 (Activation)   (None, 64, 2, 14, 14)     0
=================================================================
Total params: 70,448
Trainable params: 70,056
Non-trainable params: 392
_________________________________________________________________
```

IN THE REAL LIFE…

RESNET

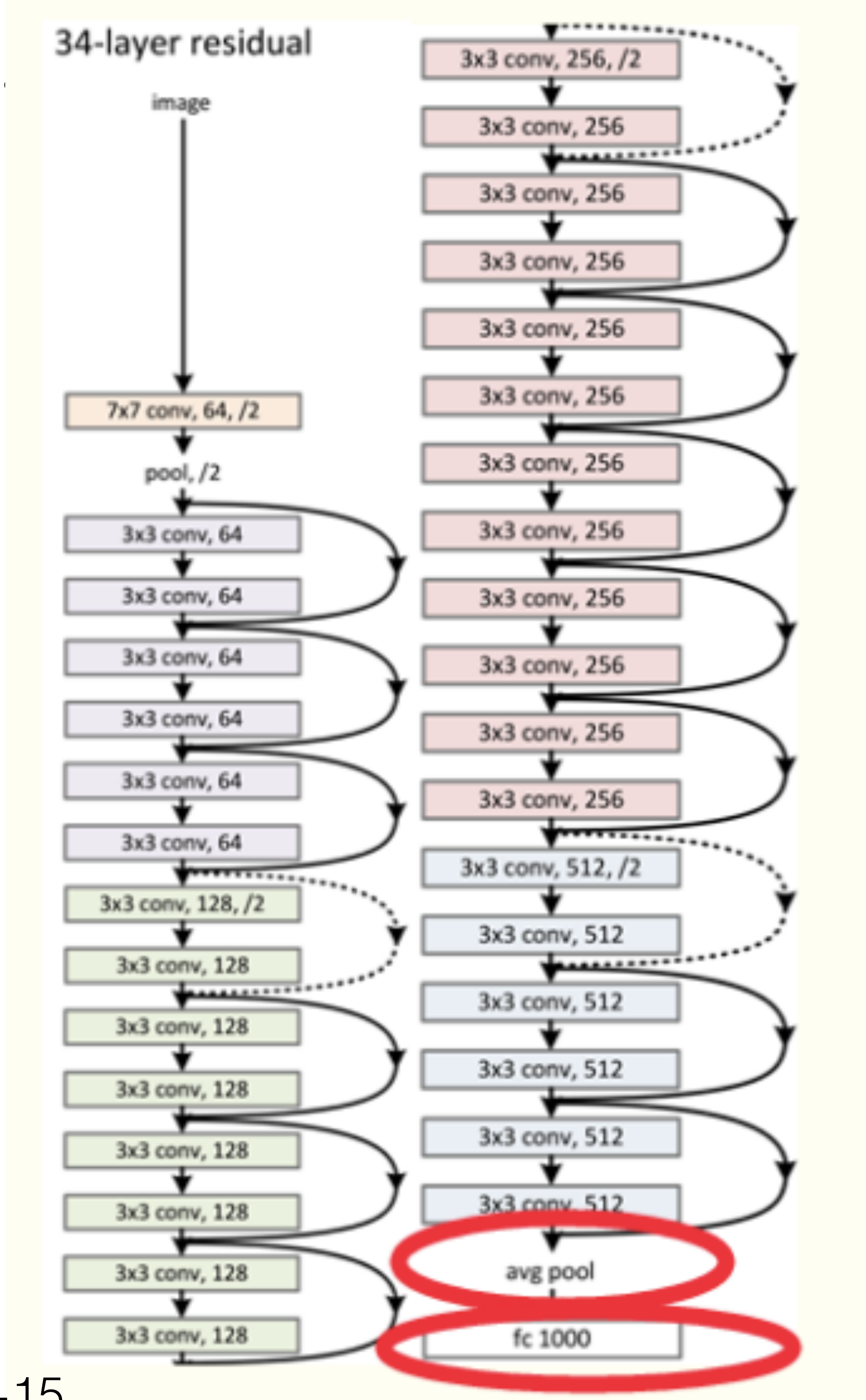


IN THE REAL LIFE…

RESNET

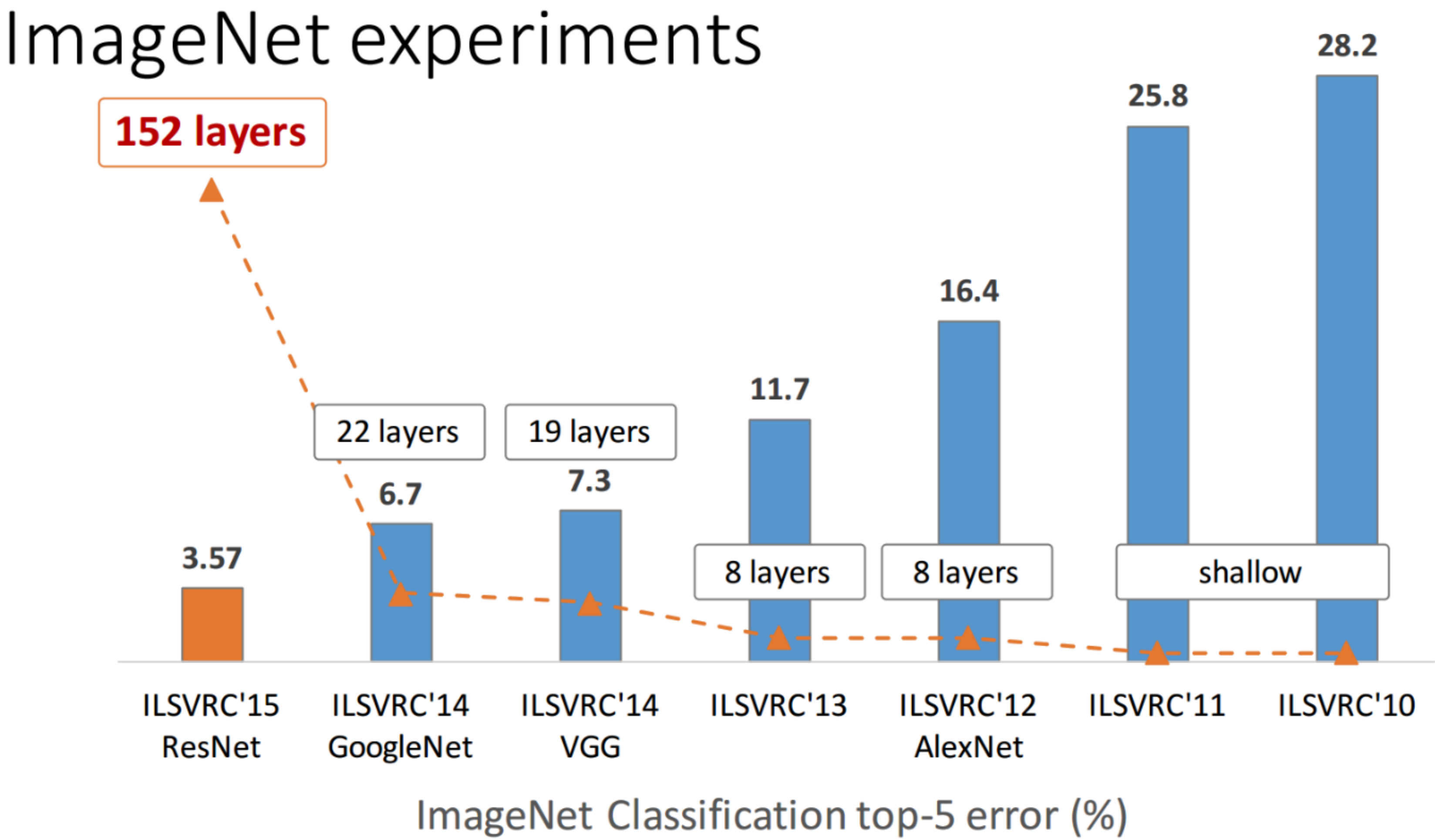
DO WE NEED TO GO THIS DEEP FOR ASTRONOMY APPLICATIONS?

[34 layers - authors explored up to 1202!]



He+15

DEEPER TENDS TO BE BETTER…

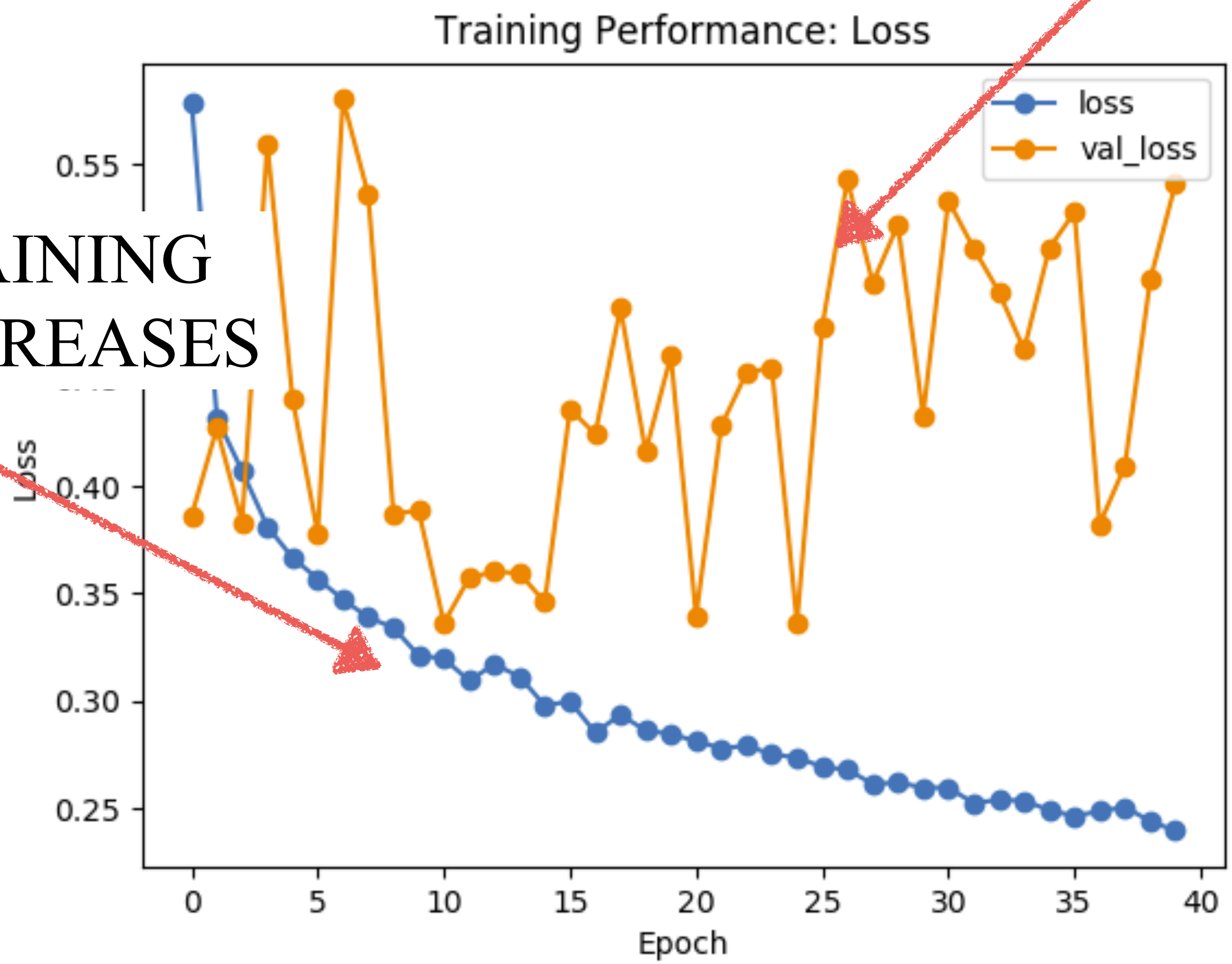


THE PROBLEMS OF GOING "TOO DEEP"

- DEEP NETWORKS ARE MORE DIFFICULT TO OPTIMIZE
- NEED MORE DATA - MORE SUBJECT TO OVER-FITTING
- AND ALSO NEED MORE TIME …

OVER-FITTING

THE TEST STAYS CONSTANT
OR INCREASES



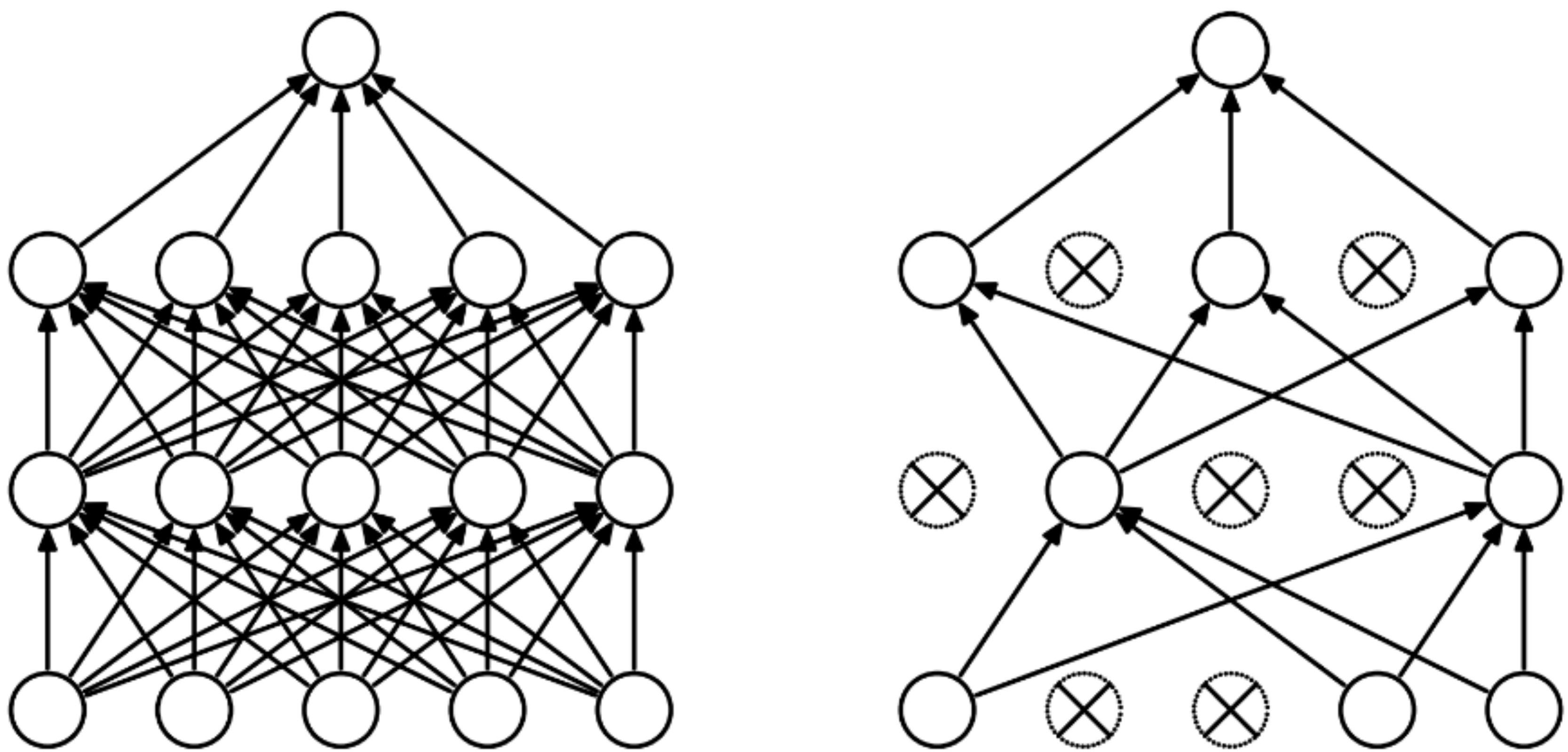
THE TRAINING
LOSS DECREASES

DROPOUT

[Hinton+12]

- THE IDEA IS TO REMOVE NEURONS RANDOMLY DURING THE TRAINING

- ALL NEURONS ARE PUT BACK DURING THE TEST PHASE



(a) Standard Neural Net (b) After applying dropout.

DROPOUT

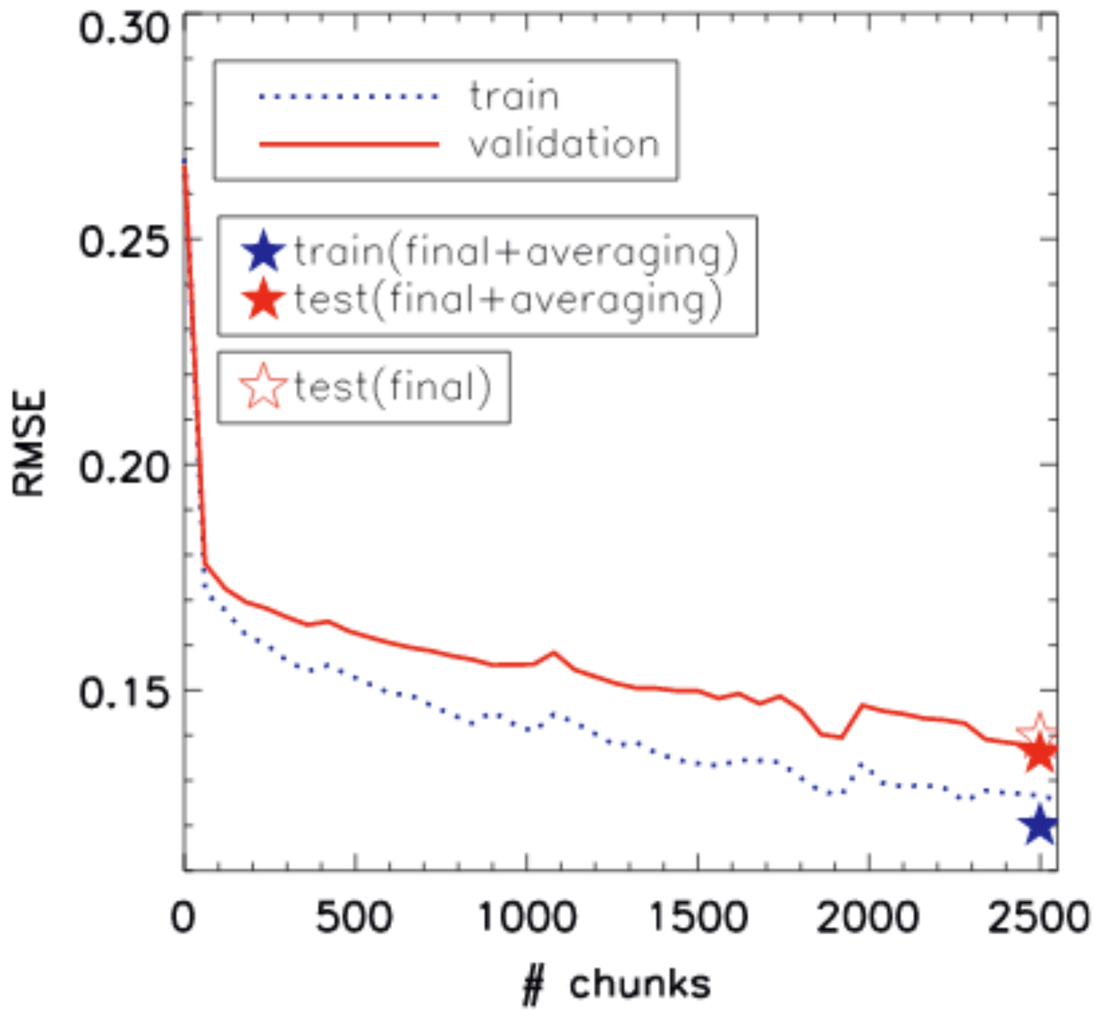
WHY DOES IT WORK?

1. SINCE NEURONS ARE REMOVED RANDOMLY, IT AVOIDS CO-ADAPTATION AMONG THEMSELVES

2. DIFFERENT SETS OF NEURONS WHICH ARE SWITCHED OFF, REPRESENT A DIFFERENT ARCHITECTURE AND ALL THESE DIFFERENT ARCHITECTURES ARE TRAINED IN PARALLEL. FOR N NEURONS ATTACHED TO DROPOUT, THE NUMBER OF SUBSET ARCHITECTURES FORMED IS 2^N. SO IT AMOUNTS TO PREDICTION BEING AVERAGED OVER THESE ENSEMBLES OF MODELS.

DROPOUT

WITH A LITTLE BIT OF DROPOUT

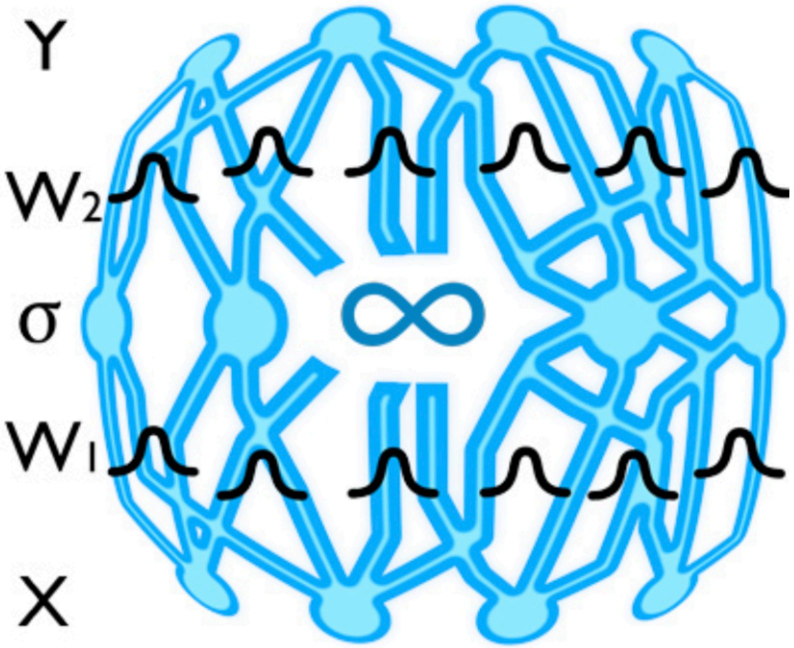


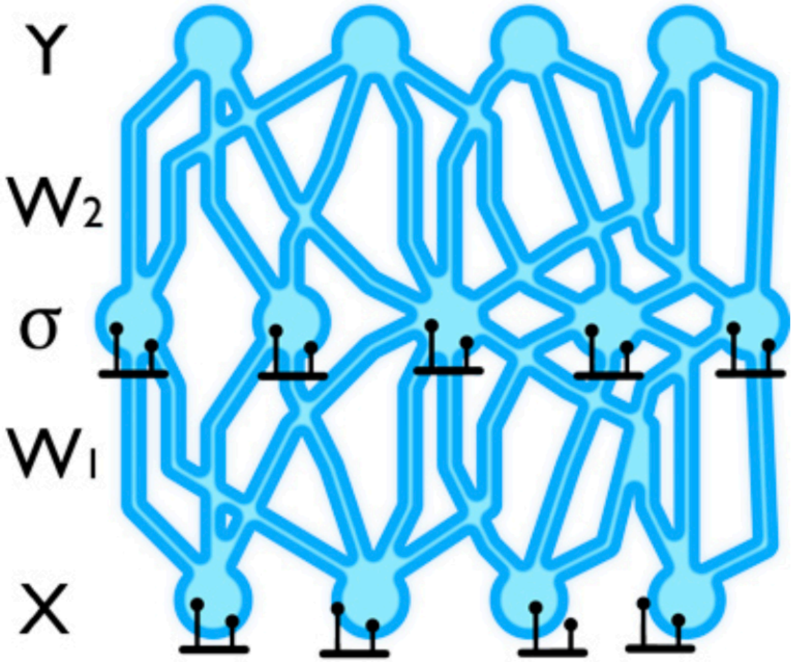
Huertas-Company+15

CAPTURING THE MODEL UNCERTAINTY

NEURAL NETWORKS AS BAYESIAN MODELS

Denker&LEcun91, Neal+95, Graves+11, Kingma+15, **Gal+15…**





BNNs ADD A PRIOR DISTRIBUTION TO EACH WEIGHT - HARD TO TRAIN

GAL+15 SHOW THAT DROPOUT CAN BE USED TO ESTIMATE UNCERTAINTY

IMPLEMENTATION IN KERAS / TENSORFLOW

```
#========= Model definition=======

#Convolutional Layers

model = Sequential()
model.add(Convolution2D(32, 6,6, border_mode='same',
                        input_shape=(img_channels, img_rows, img_cols)))
model.add(Activation('relu'))
model.add(Dropout(0.5))

model.add(Convolution2D(64, 5, 5, border_mode='same'))
model.add(Activation('relu'))

model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Convolution2D(128, 2, 2, border_mode='same'))
model.add(Activation('relu'))

model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Convolution2D(128, 3, 3, border_mode='same'))
model.add(Activation('relu'))

model.add(Dropout(0.25))

#Fully Connected start here
#------------------------------#

model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dropout(.5))
model.add(Dense(1, init='uniform', activation='sigmoid'))

print("Compilation...")

model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
```

VANISHING / EXPLODING GRADIENT PROBLEM

REMEMBER THAT:

$$y_{i+1} = \sigma\left[\sum w_i y_i\right]$$

output layer i+1

activation function

weights

output layer i

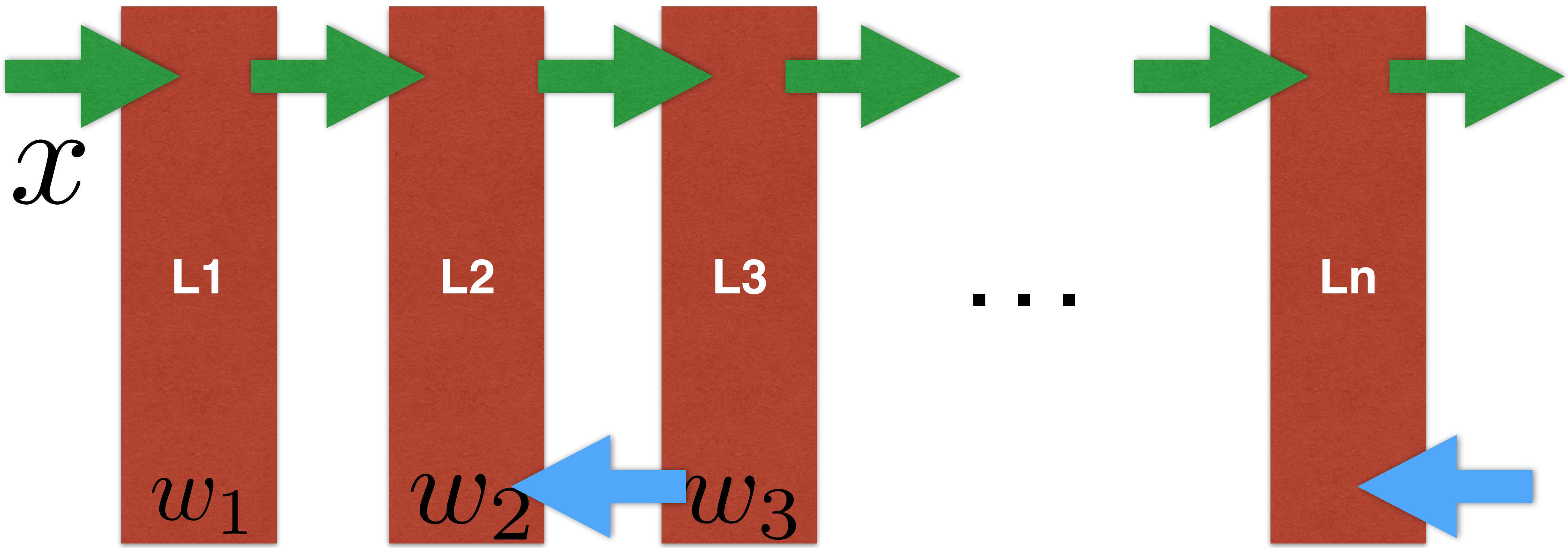
VANISHING / EXPLODING GRADIENT PROBLEM

WITH MANY LAYERS:

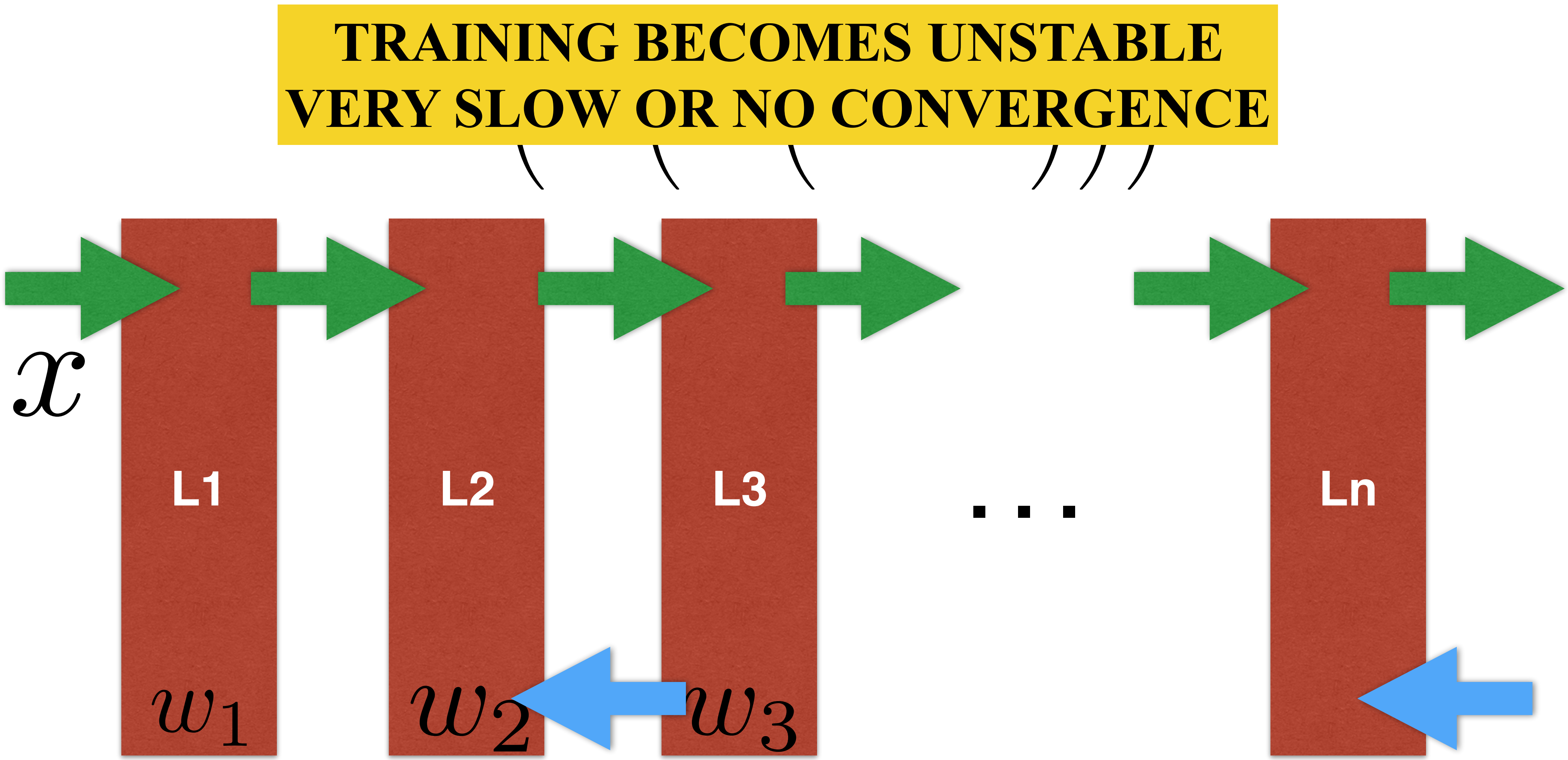
$$y_n = \sigma\left(...\sigma\left(...\sigma\left(\sum w_0 x\right)\right)\right)$$

VANISHING/EXPLODING GRADIENT PROBLEM

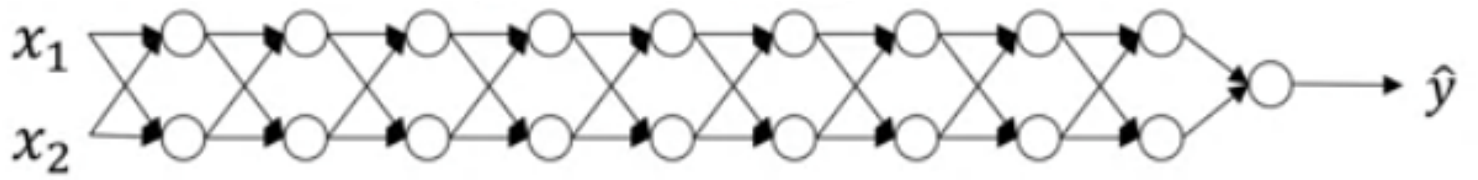
$$y_n = \sigma\left(\ldots\sigma\left(\ldots\sigma\left(\sum w_0 x\right)\right)\right)$$



VANISHING/EXPLODING GRADIENT PROBLEM



VANISHING/EXPLODING GRADIENT PROBLEM



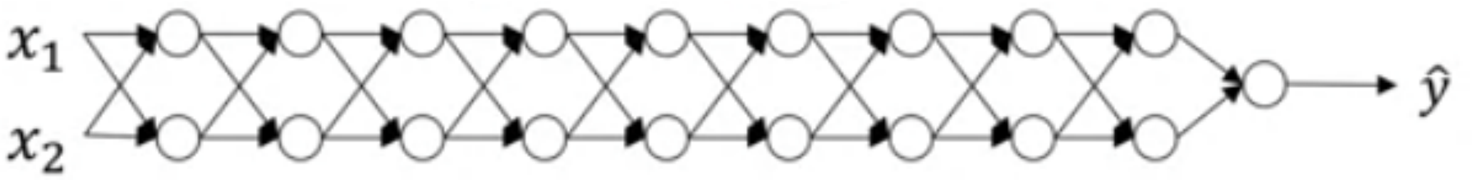
IF WE ASSUME AN IDENTITY ACTIVATION FUNCTION:

$$\hat{y} = x \prod_{n} w_i$$

with:

$$w_i = \begin{pmatrix} w_i^0 & 0 \\ 0 & w_i^1 \end{pmatrix} \qquad x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

VANISHING/EXPLODING GRADIENT PROBLEM



$$w_i = \begin{pmatrix} w_i^0 & 0 \\ 0 & w_i^1 \end{pmatrix} \qquad \hat{y} = x \prod_n w_i$$

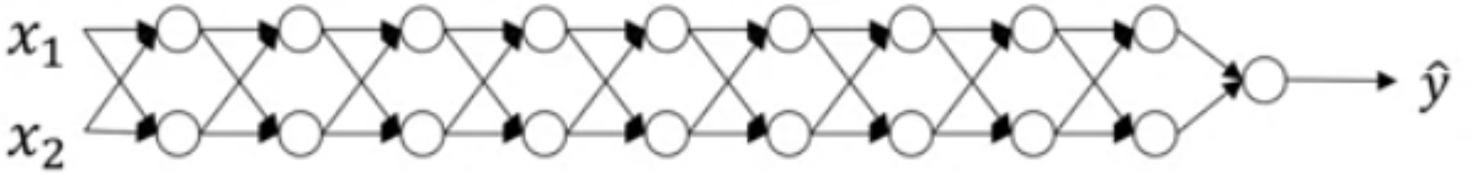
$$x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

IF WEIGHTS ARE ALL INITIALIZED TO VALUES <<1:

$$w_i^L \to 0$$

VANISHING GRADIENT

VANISHING/EXPLODING GRADIENT PROBLEM



$$w_i = \begin{pmatrix} w_i^0 & 0 \\ 0 & w_i^1 \end{pmatrix}$$

$$\hat{y} = x \prod_n w_i$$

$$x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

IF WEIGHTS ARE ALL INITIALIZED TO VALUES >1:

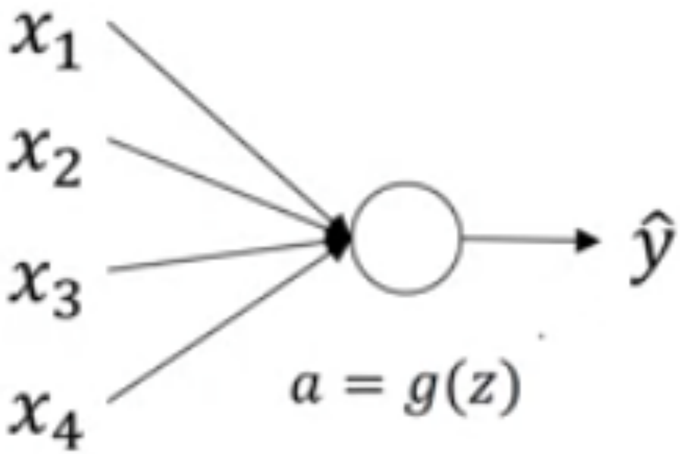
$$w_i^L \rightarrow \infty$$

EXPLODING GRADIENT

VANISHING/EXPLODING GRADIENT PROBLEM

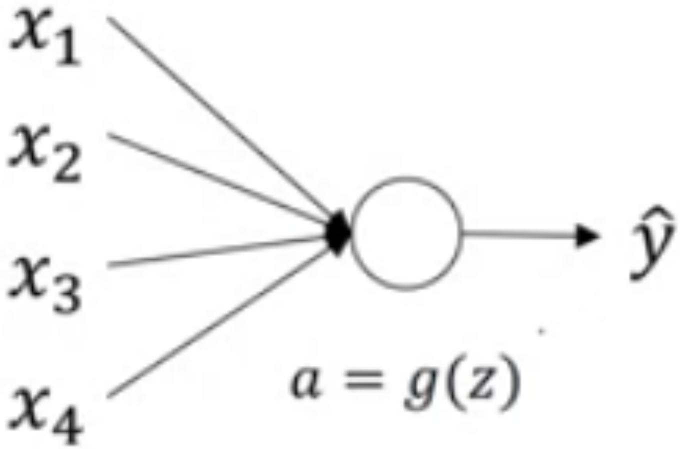
TRAINING BECOMES UNSTABLE
VERY SLOW OR NO CONVERGENCE

WEIGHT INITIALIZATION IS A KEY POINT…



$$z = x_1 w_1 + x_2 w_2 + ... + x_n w_n$$

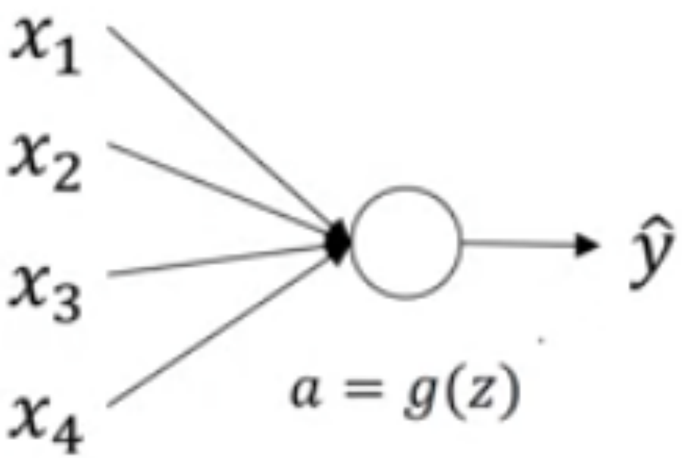
WEIGHT INITIALIZATION IS A KEY POINT…



$$z = x_1 w_1 + x_2 w_2 + \ldots + x_n w_n$$

THE LARGER n, THE SMALLER WEIGHTS SHOULD BE…

WEIGHT INITIALIZATION IS A KEY POINT…



$$z = x_1 w_1 + x_2 w_2 + \ldots + x_n w_n$$

THE LARGER n, THE SMALLER WEIGHTS SHOULD BE…

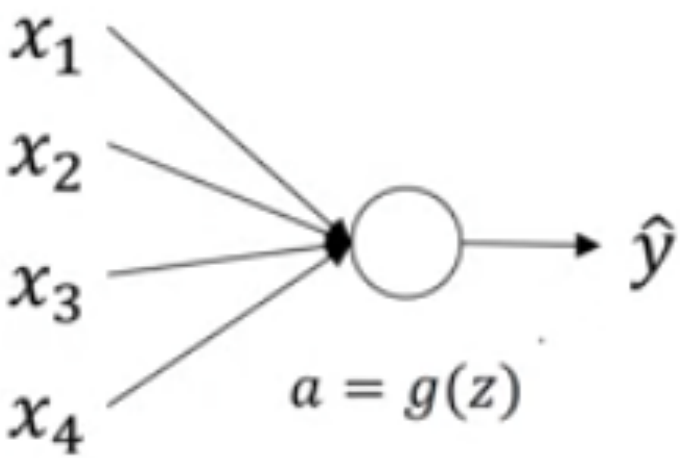
variance

<u>ONE SIMPLE SOLUTION:</u>

$$\sigma^2(w_i) = \frac{1}{n}$$

number of inputs

WEIGHT INITIALIZATION IS A KEY POINT…



$$z = x_1 w_1 + x_2 w_2 + \ldots + x_n w_n$$

THE LARGER n, THE SMALLER WEIGHTS SHOULD BE…

variance

<u>ONE SIMPLE SOLUTION:</u>

$$\sigma^2(w_i) = \frac{1}{n}$$

number of inputs

WEIGHT INITIALIZATION IS A KEY POINT…

For ReLU activation functions we typically use:

$$\sigma^2(w_i) = \frac{2}{n}$$

[He initialization, He+15]

WEIGHT INITIALIZATION IS A KEY POINT…

IMPLEMENTATION IN KERAS:

```
initialization = 'he_normal'
act = 'relu'

model = Sequential()
model.add(Convolution2D(depth, conv_size, conv_size, activation=act, border_mode='same',
name = "conv%i"%(layer_n), init=initialization, W_constraint=constraint))
```

WEIGHT INITIALIZATION IS A KEY POINT…

IMPLEMENTATION IN KERAS:

```
initialization = 'he_normal'
act = 'relu'

model = Sequential()
model.add(Convolution2D(depth, conv_size, conv_size, activation=act, border_mode='same',
name = "conv%i"%(layer_n), init=initialization, W_constraint=constraint))
```

MANY OTHER INITIALIZATIONS AVAILABLE:

`keras.initializers` → https://keras.io/initializers/

BATCH NORMALIZATION

[SZEGEDY+15]

ANOTHER SOLUTION TO KEEP REASONABLE VALUES OF THE ACTIVATIONS IN DEEP NETWORKS

BATCH NORMALIZATION PREVENTS LOW OR LARGE VALUES BY RE-NORMALIZING THE VALUES BEFORE ACTIVATION FOR EVERY BATCH

INPUT

$$\hat{y}_i = \gamma \frac{y_i - E(y_i)}{\sigma(y_i)} + \beta$$

NORMALIZED INPUT

SCATTER

BATCH NORMALIZATION

[SZEGEDY+15]

BATCH NORMALIZATION SPEEDS UP AND STABILIZES TRAINING

AS FOR THE DROPOUT, THERE IS A DIFFERENT BEHAVIOR BETWEEN TRAINING AND TESTING

INPUT

$$\hat{y}_i = \gamma \frac{y_i - E(y_i)}{\sigma(y_i)} + \beta$$

NORMALIZED INPUT

SCATTER

BATCH NORMALIZATION

[SZEGEDY+15]

IN KERAS, IT IS IMPLEMENTED AS AN ADDITIONAL LAYER

BatchNormalization [source]

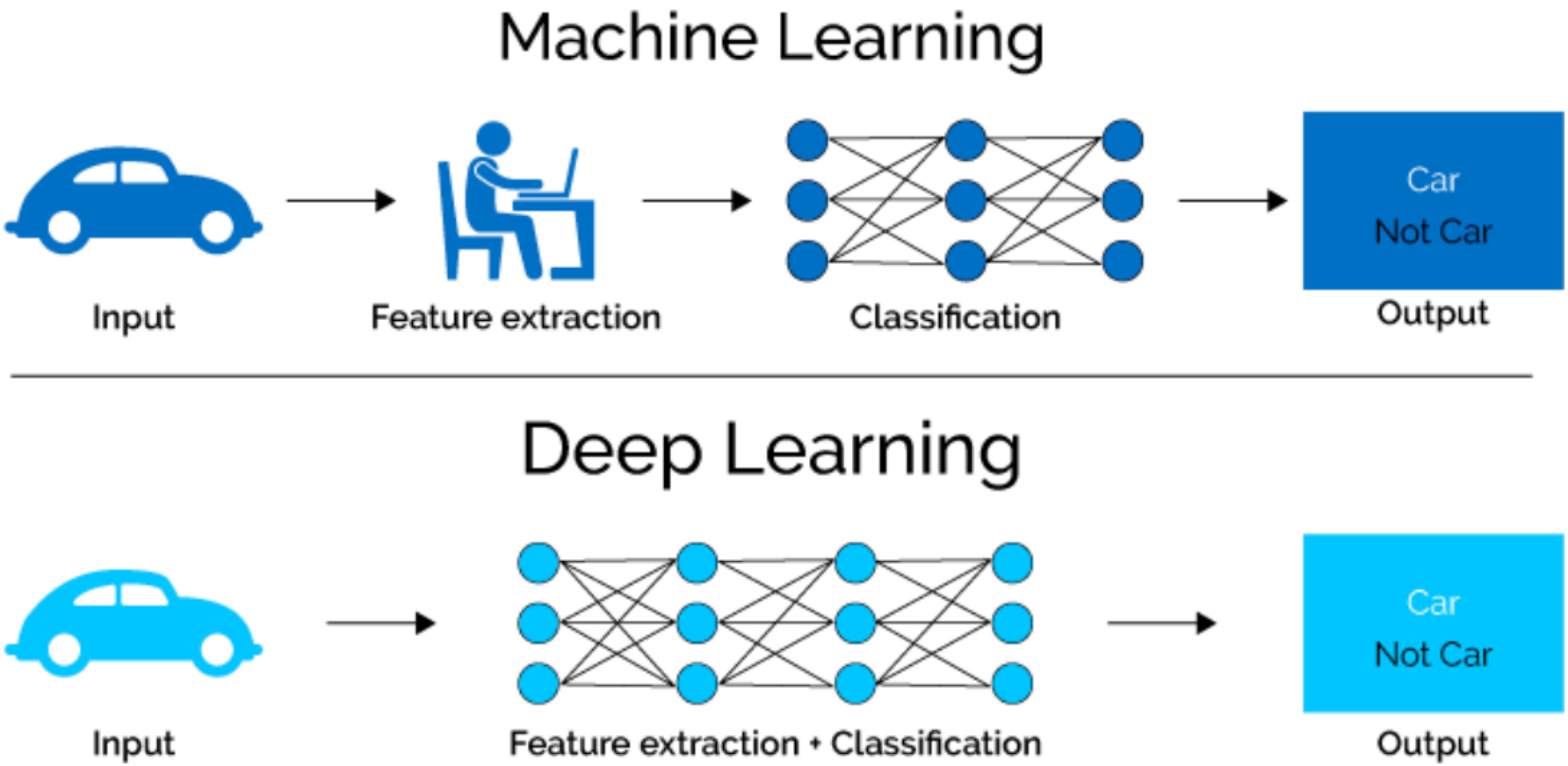
```
keras.layers.BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001, center=True, scale=True, beta_initializer
```

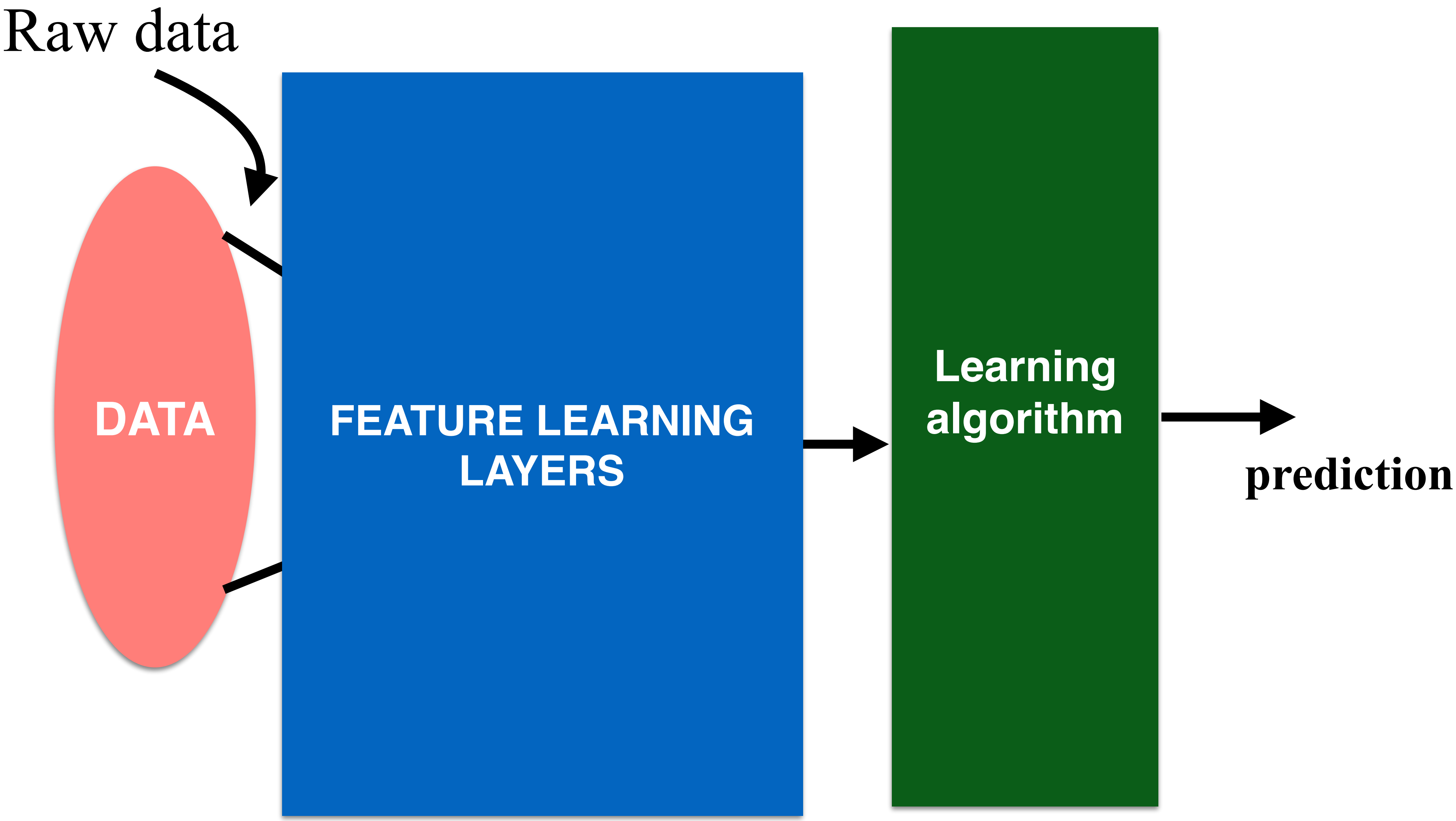
Batch normalization layer (Ioffe and Szegedy, 2014).

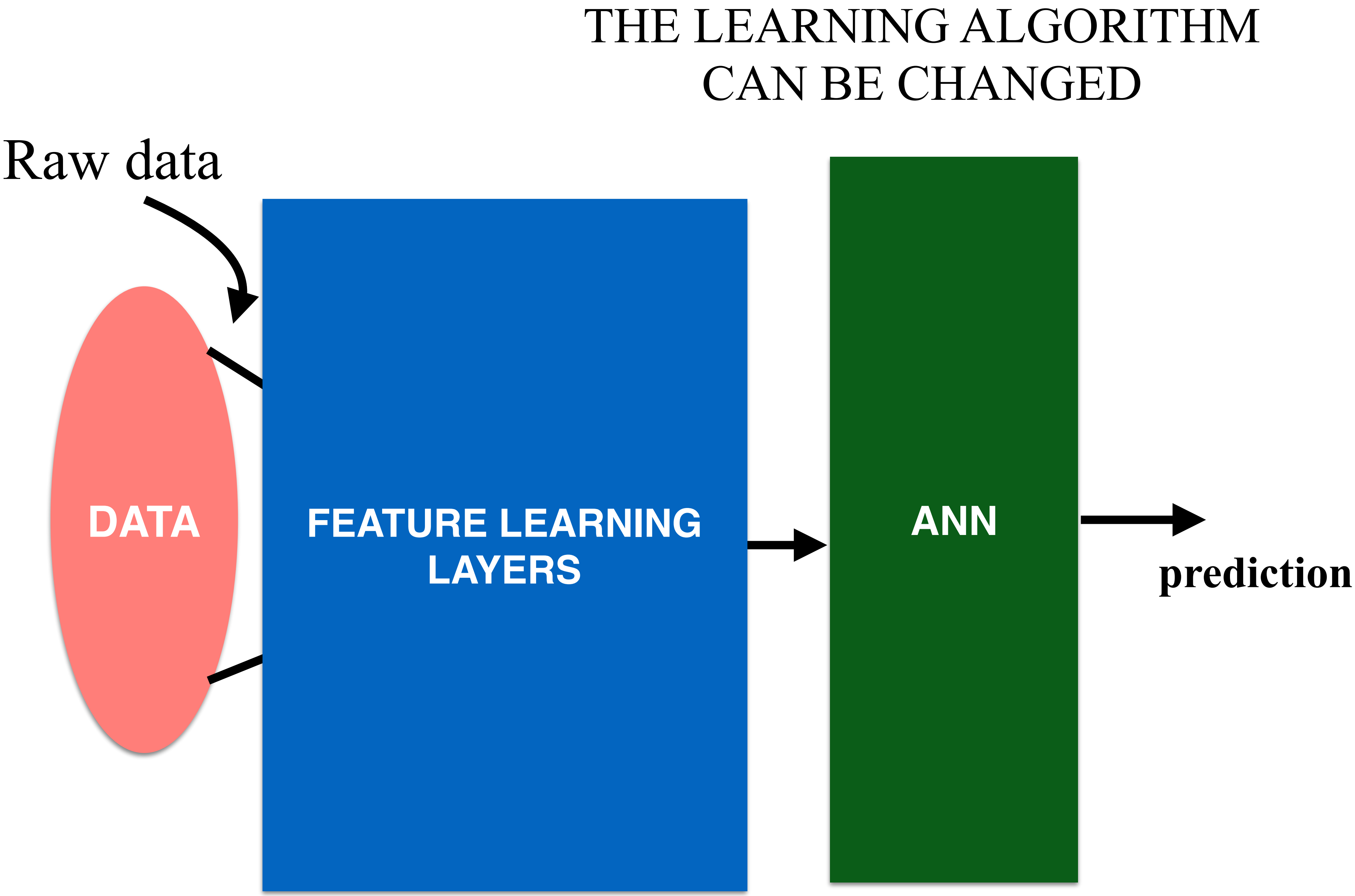
Normalize the activations of the previous layer at each batch, i.e. applies a transformation that maintains the mean activation close to 0 and the activation standard deviation close to 1.

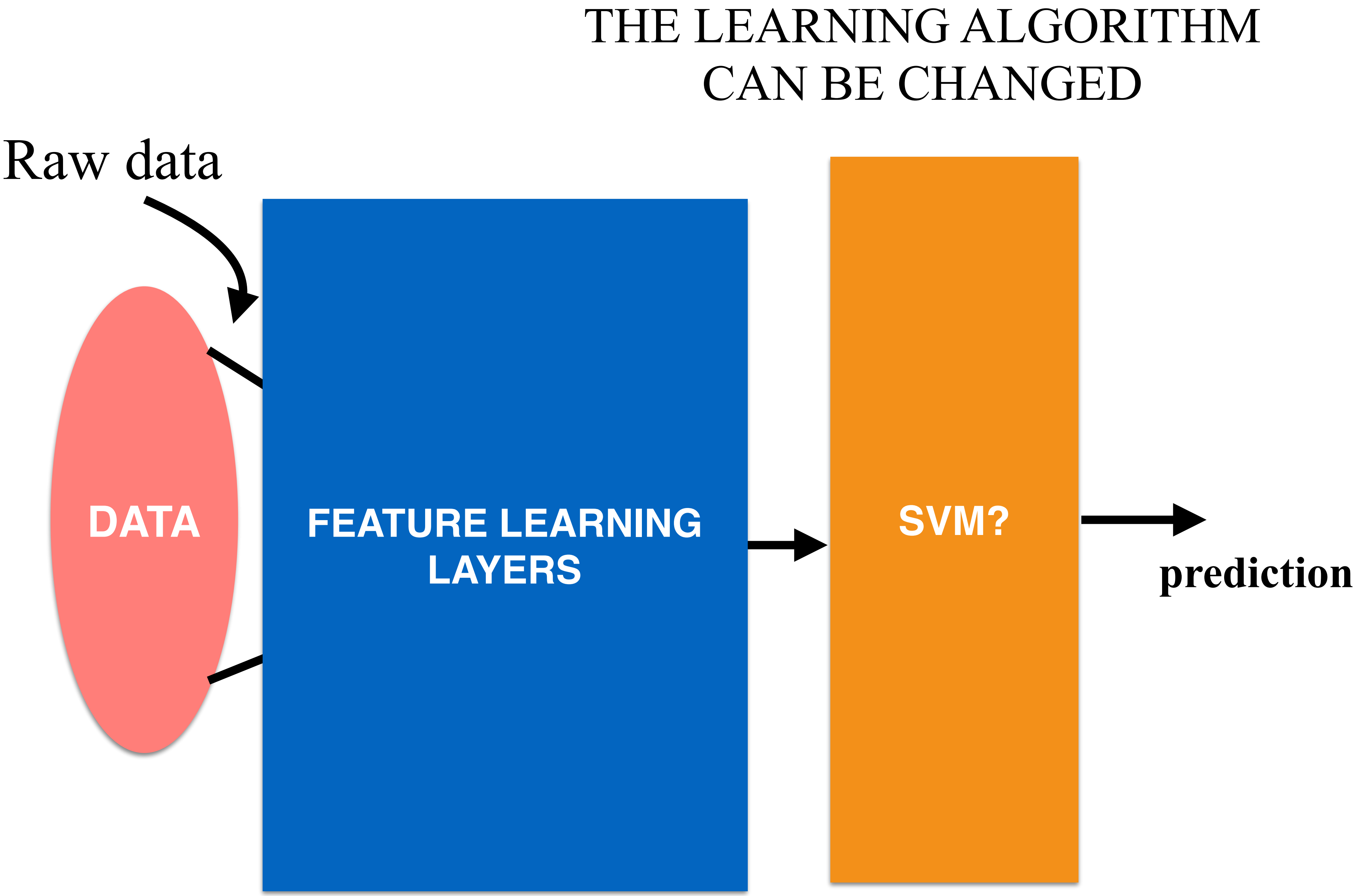
Arguments

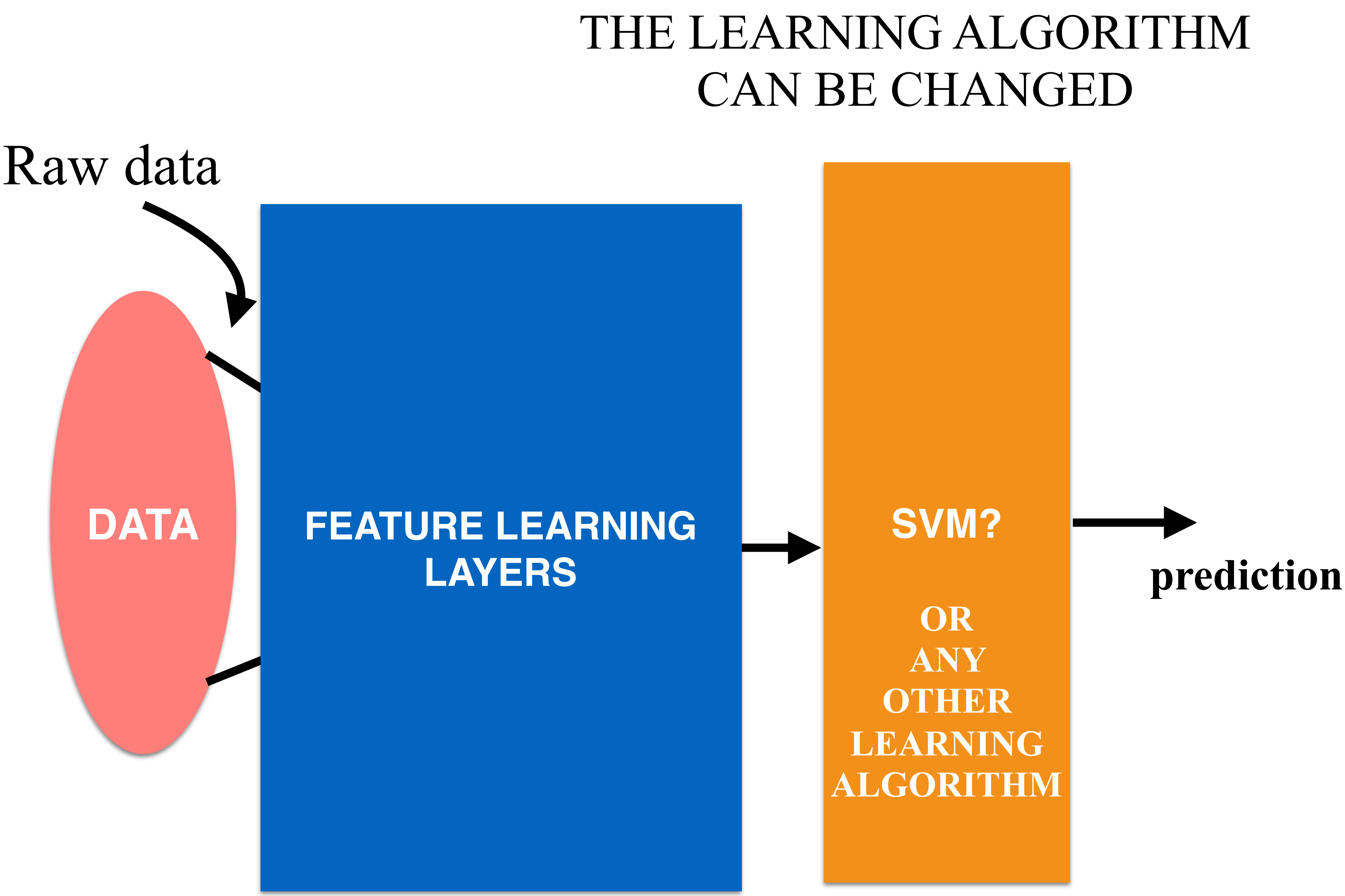
THIS IS A CHANGE OF PARADIGM!



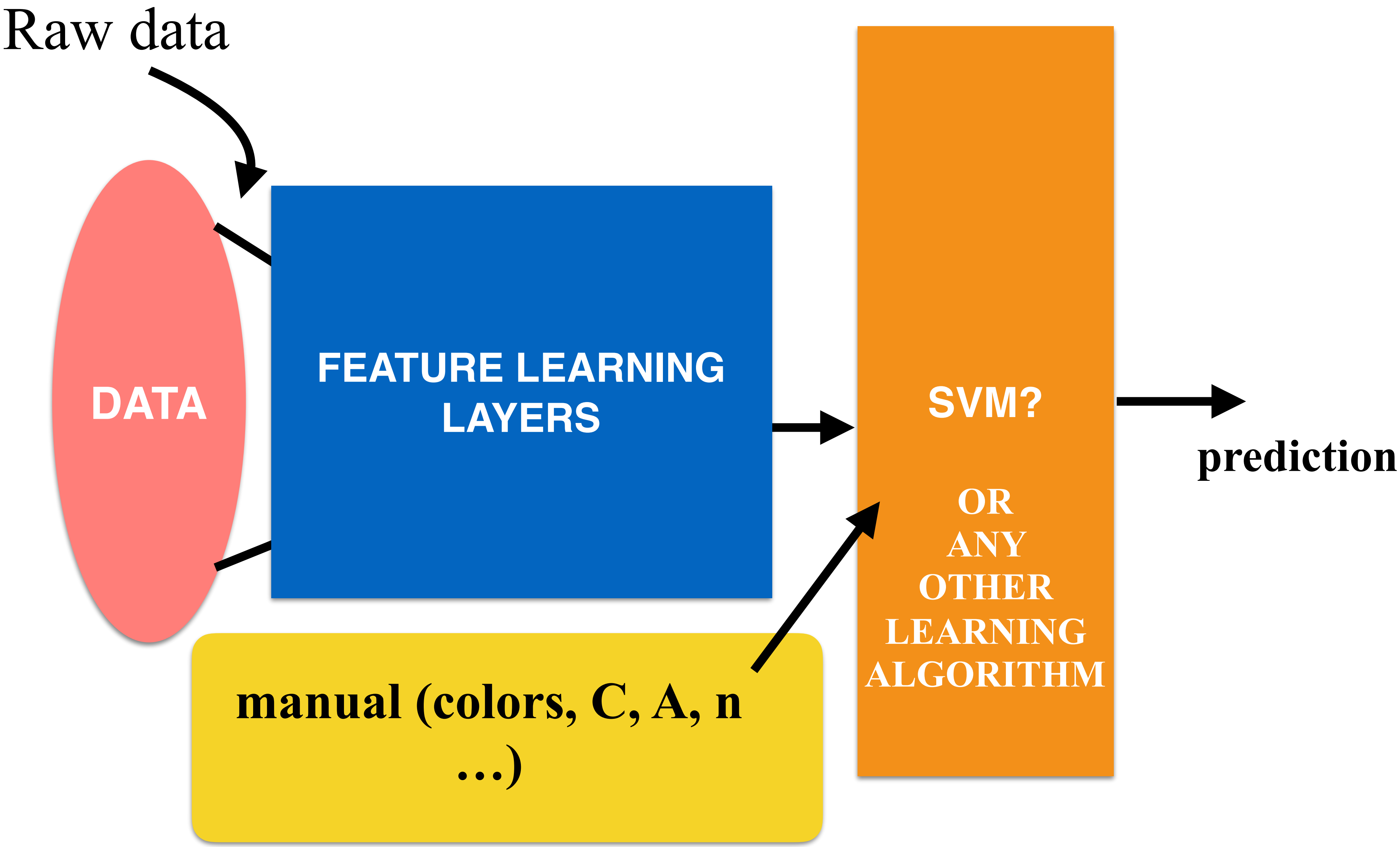

Raw data
DATA
FEATURE LEARNING LAYERS
Learning algorithm
prediction

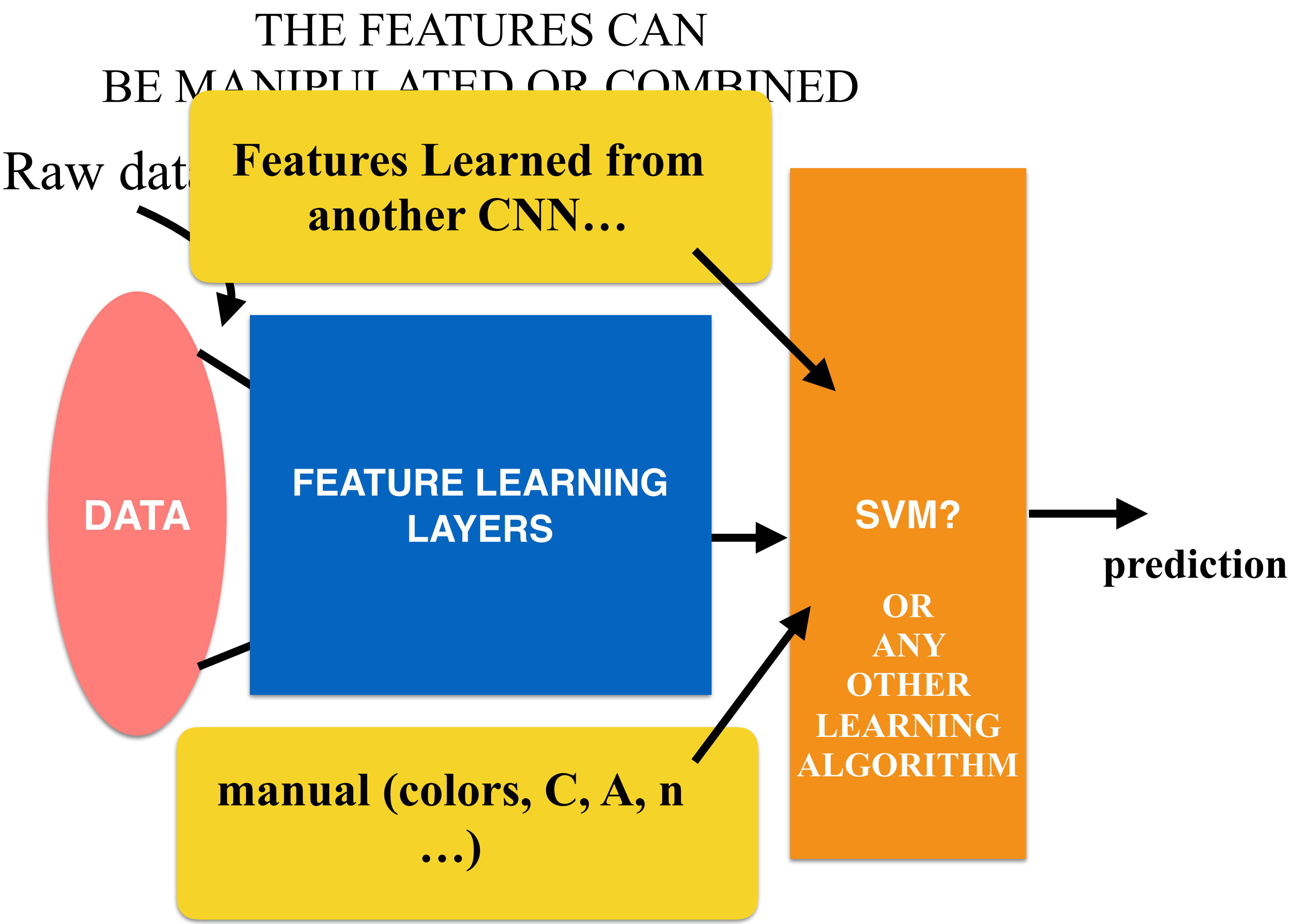

THE LEARNING ALGORITHM
CAN BE CHANGED
Raw data
DATA
FEATURE LEARNING
LAYERS
ANN
prediction


THE LEARNING ALGORITHM
CAN BE CHANGED
Raw data
DATA
FEATURE LEARNING
LAYERS
SVM?
prediction


THE LEARNING ALGORITHM
CAN BE CHANGED
Raw data
DATA
FEATURE LEARNING
LAYERS
SVM?
OR
ANY
OTHER
LEARNING
ALGORITHM
prediction

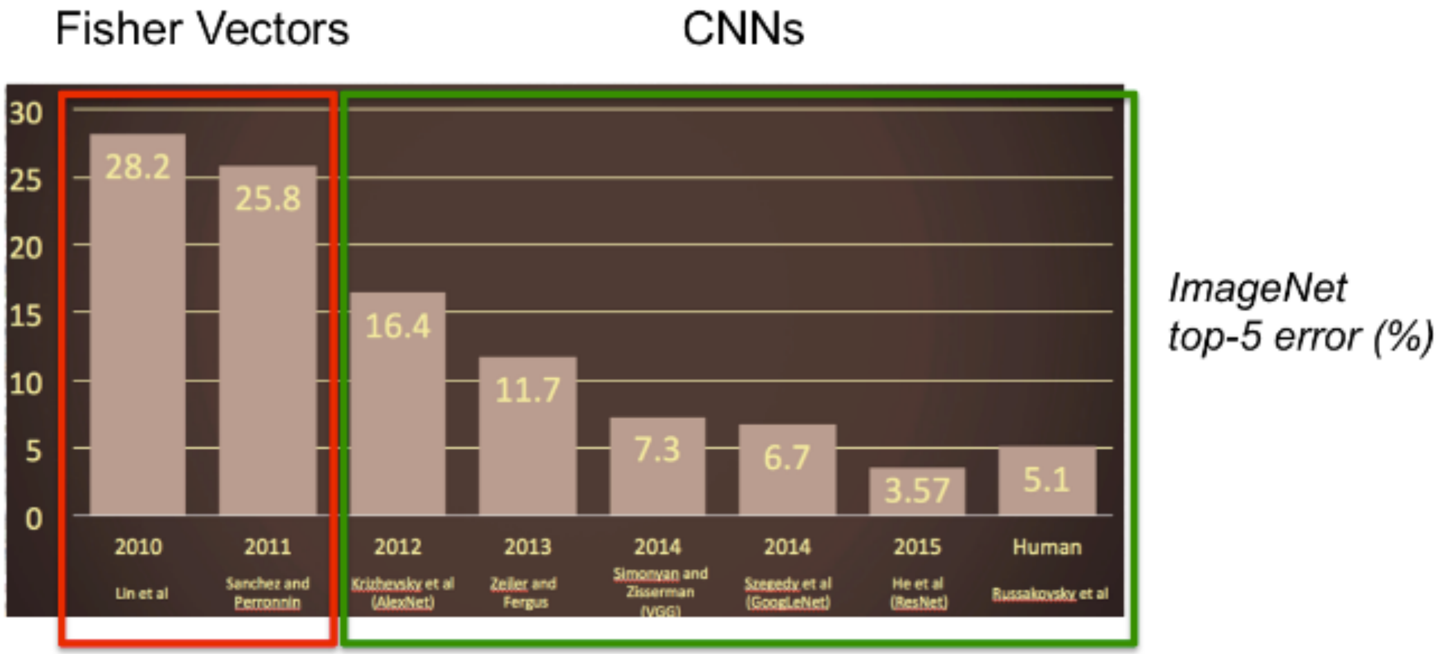
THE FEATURES CAN
BE MANIPULATED OR COMBINED





THE FEATURES CAN
BE MANIPULATED OR COMBINED
Raw dat
Features Learned from another CNN…
DATA
FEATURE LEARNING LAYERS
SVM?
OR ANY OTHER LEARNING ALGORITHM
prediction
manual (colors, C, A, n …)

THIS IS A CHANGE OF PARADIGM!



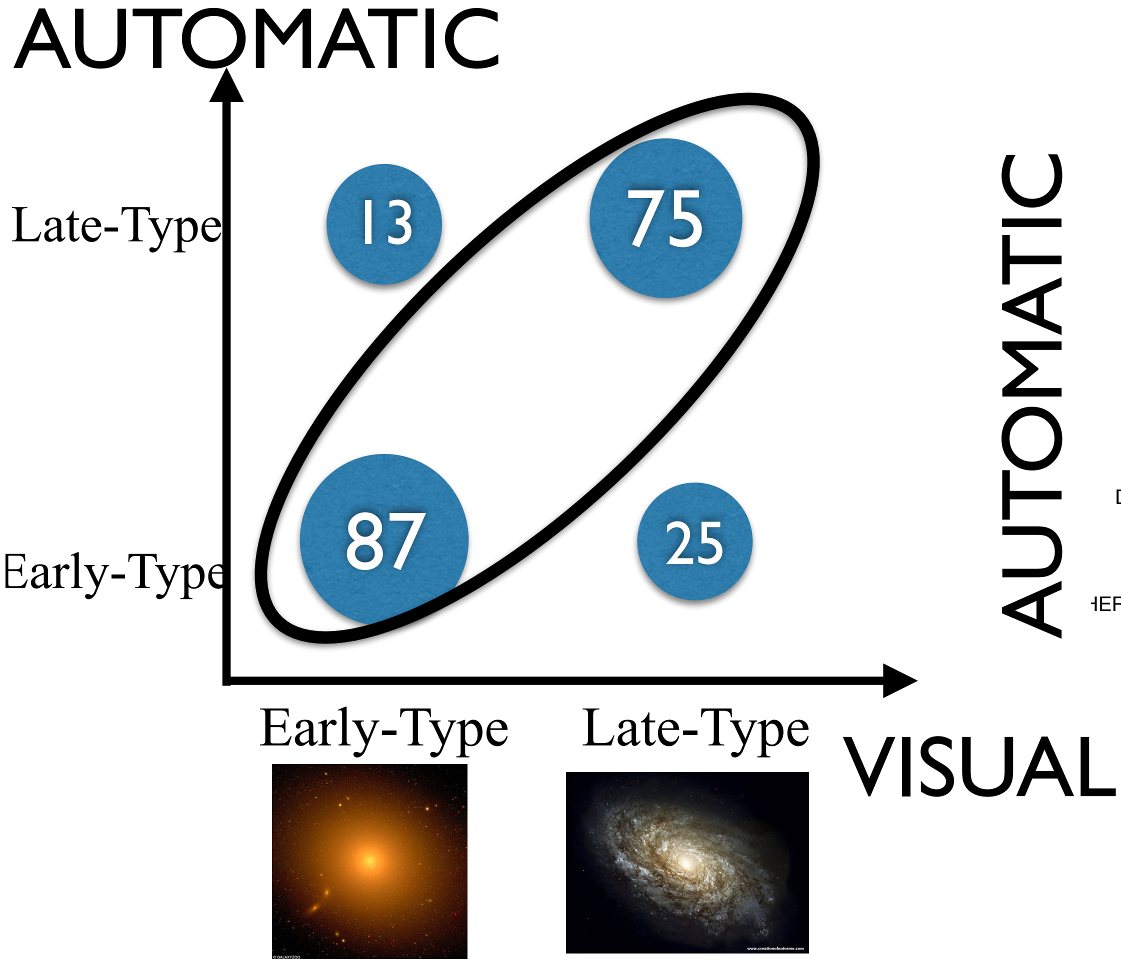
ALSO FOR GALAXY MORPHOLOGY

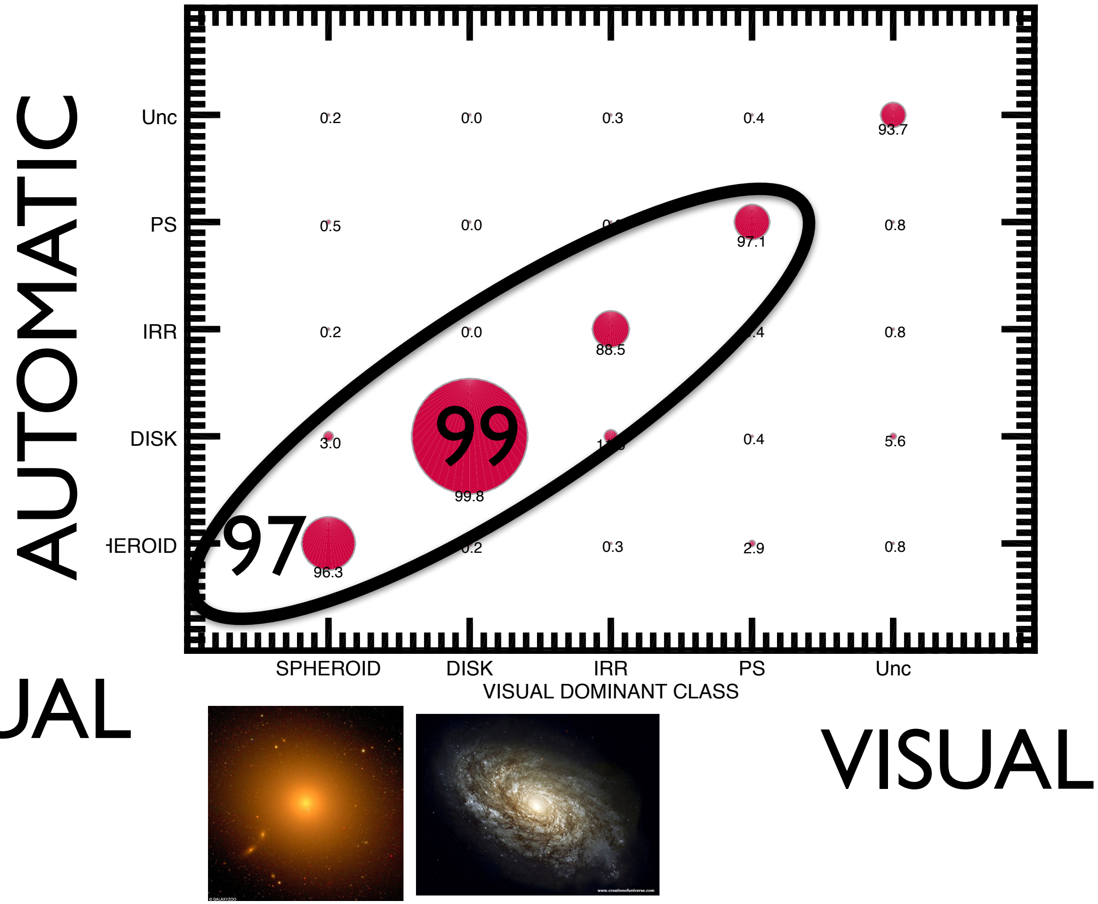
SVMs

CNNs

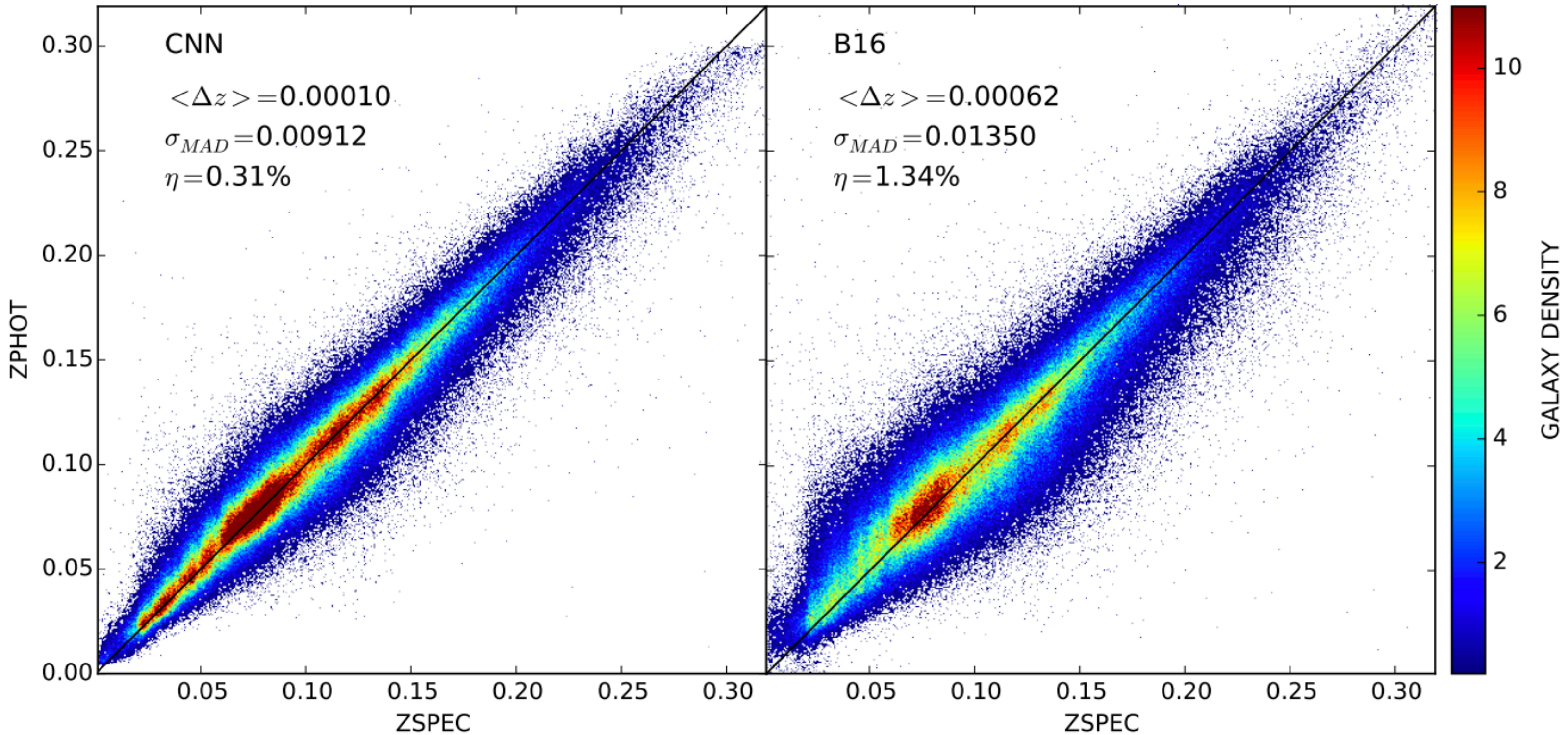
[HUERTAS-COMPANY+14]

[HUERTAS-COMPANY+15b]





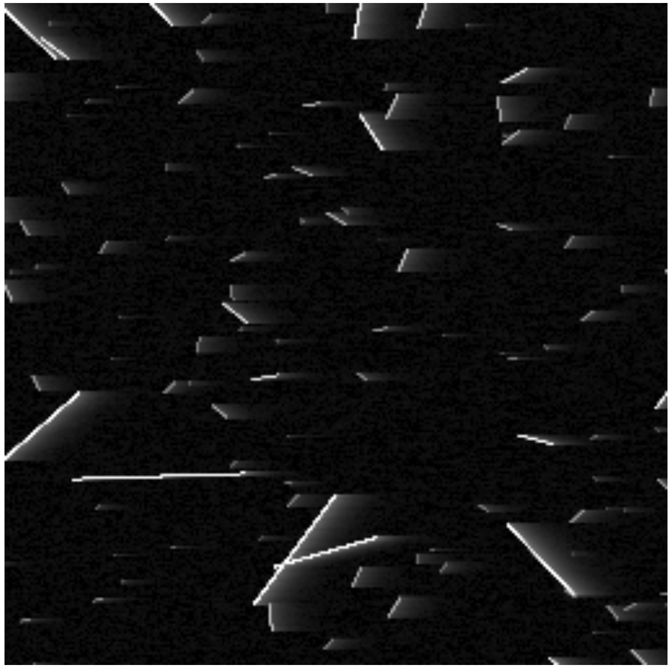
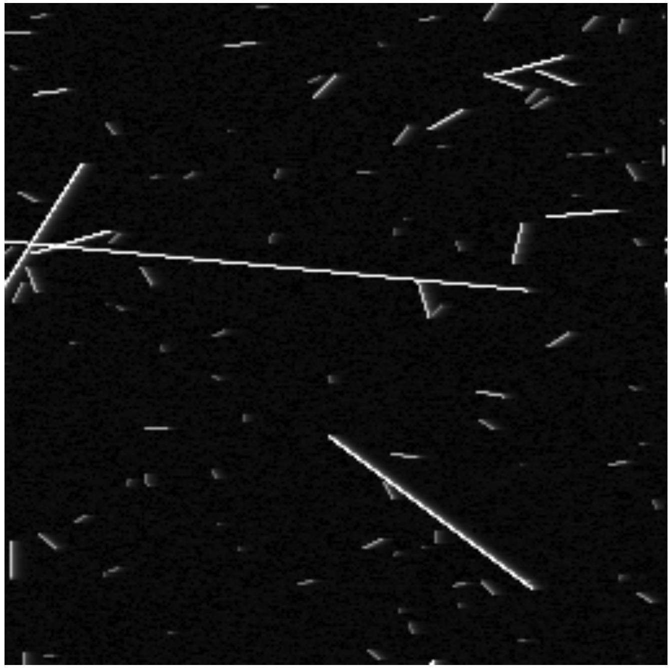
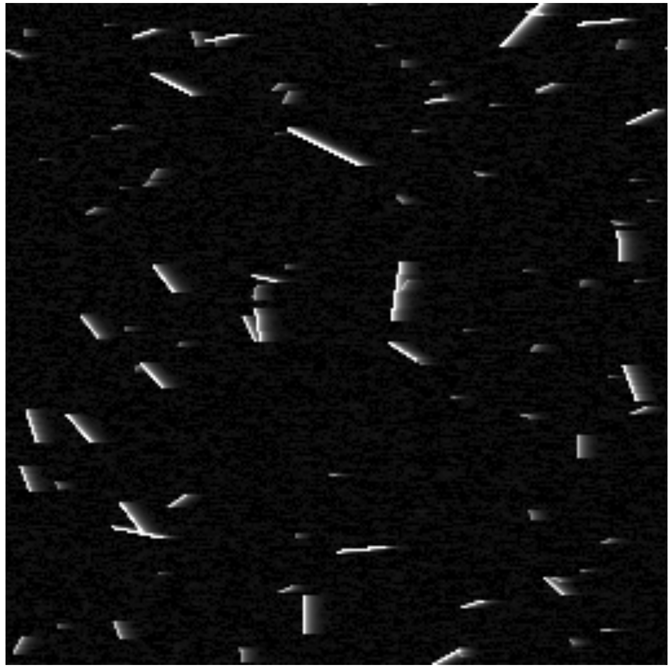
PHOTOMETRIC REDSHIFTS



Pasquet+18

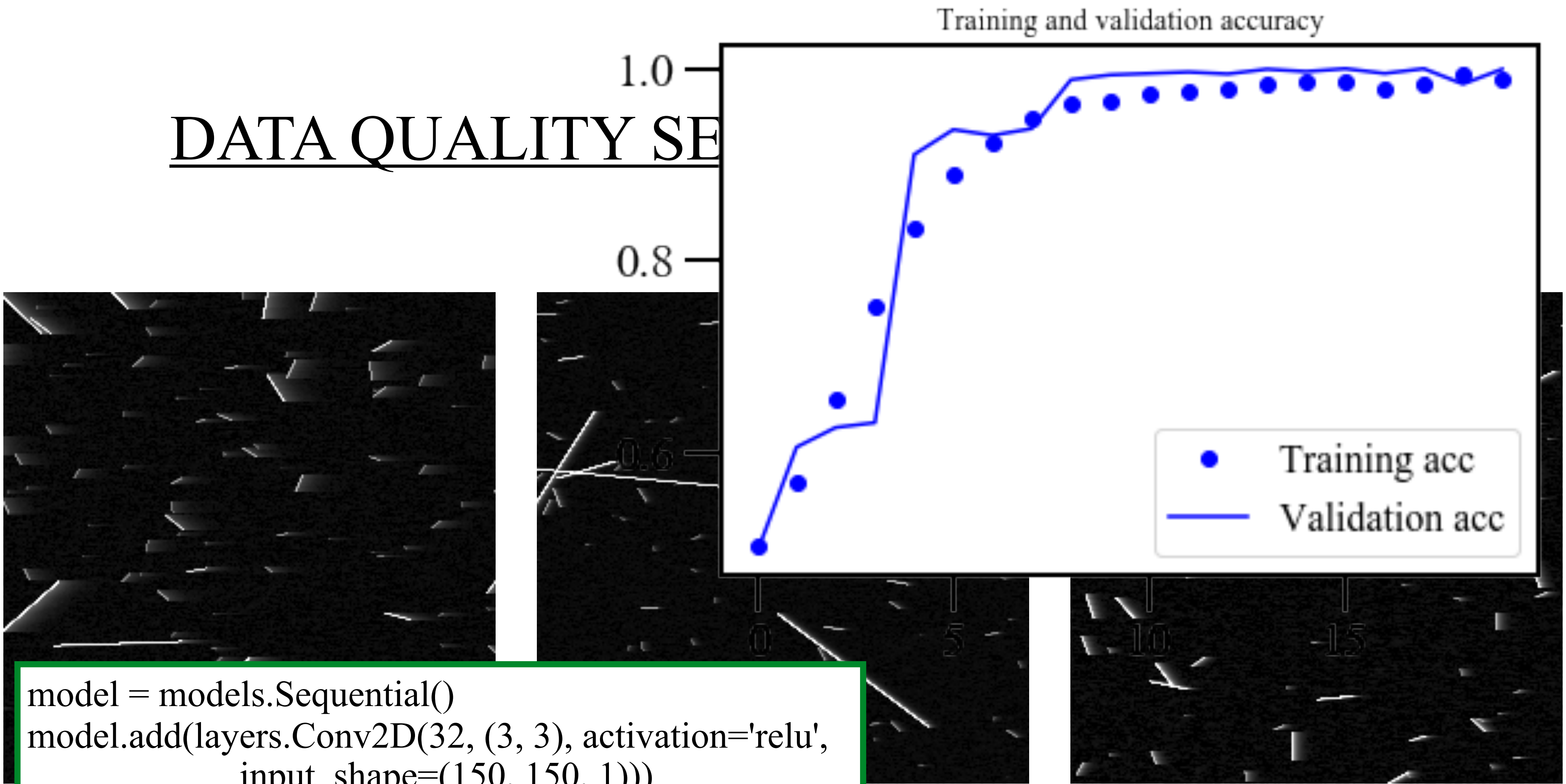
AUTOMATICALLY COMBINING MORPHOLOGY AND COLOR FOR PHOTOZ ESTIMATION

DATA QUALITY SELECTION FOR EUCLID

Thanks to H. McCracken

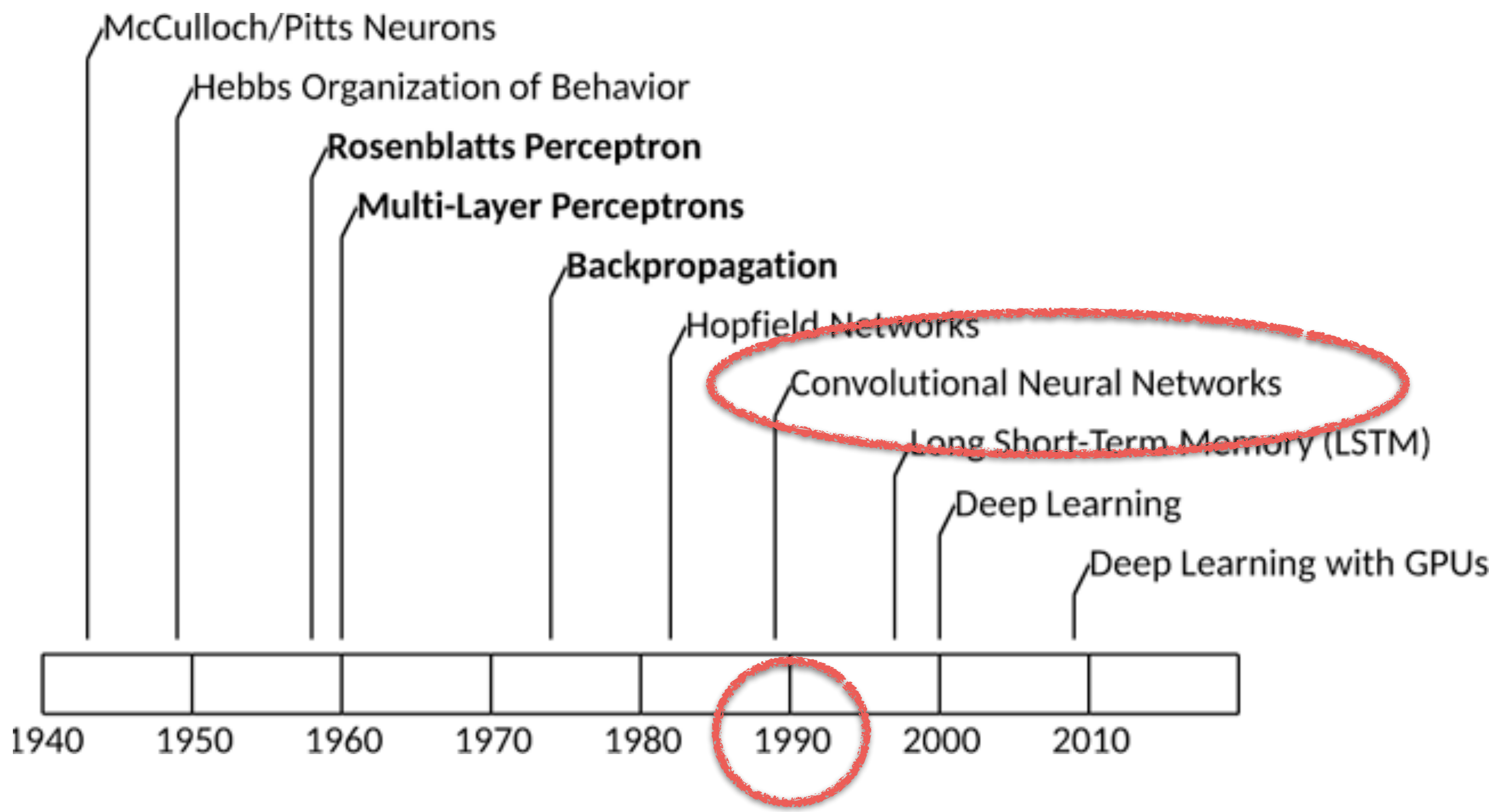
DATA QUALITY SE




```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(150, 150, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

Thanks to H. McCracken

WELL, BUT THIS IS AN "OLD" IDEA - WHY NOW?



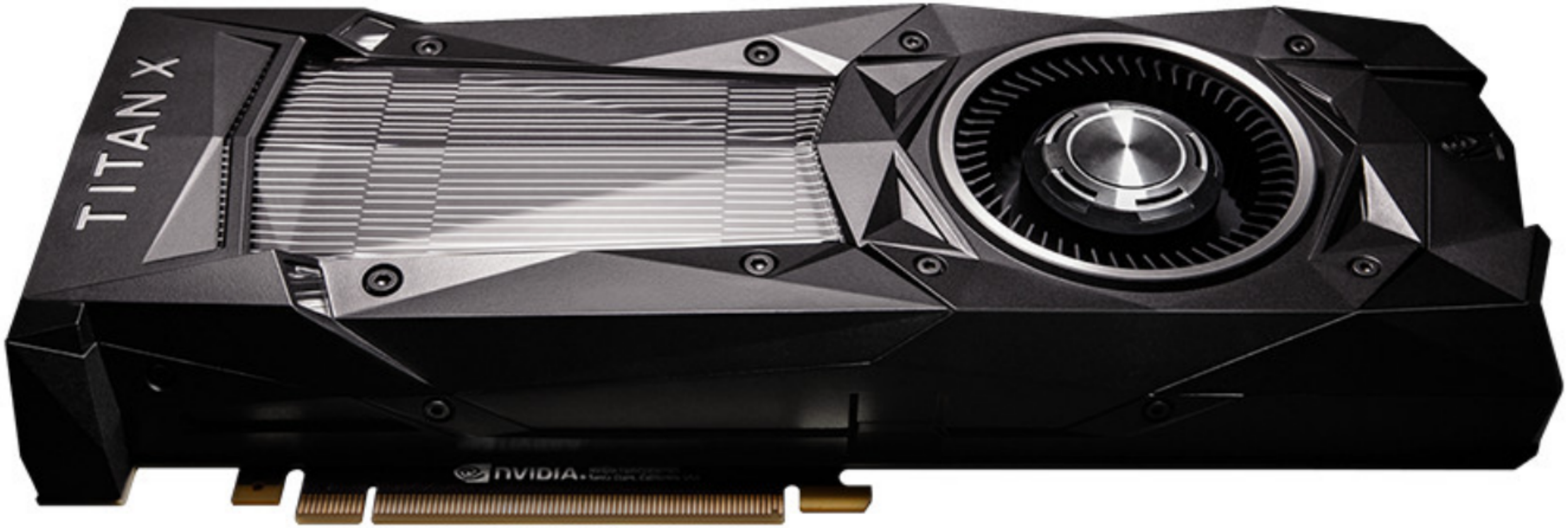
WELL, BUT THIS IS AN “OLD” IDEA - WHY NOW?

1 - MORE DATA TO TRAIN! DEEP NETWORKS HAVE A LARGE NUMBER OF PARAMETERS - THX TO SOCIAL MEDIA …

WELL, BUT THIS IS AN “OLD” IDEA - WHY NOW?

2 - GPUs - TRAINING OF THESE DEEP NETWORKS HAS REMAINED PROHIBITIVELY TIME CONSUMING WITH CPUs - THX TO VIDEO GAMES…

GPUs



NVIDA TITANX GPU

GPUs vs. CPUs

CPUs

FEWER CORES (~10x)

EACH CORE IS FASTER

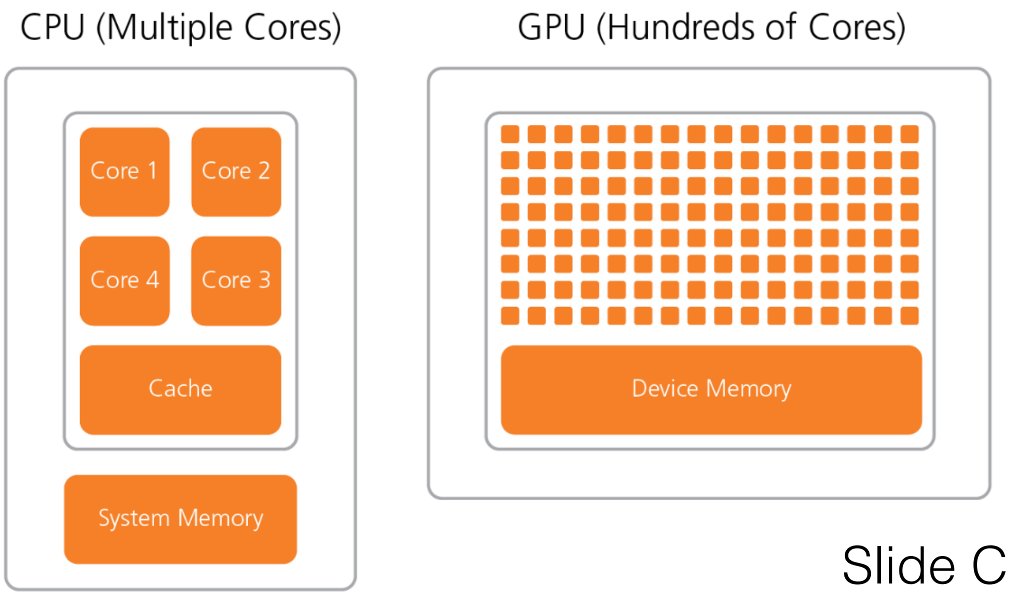
USEFUL FOR SEQUENTIAL TASKS

GPUs

MORE CORES (100x)

EACH CORE IS SLOWER

USEFUL FOR PARALLEL TASKS



Slide Credit:

GPUs vs. CPUs

More benchmarks available here.

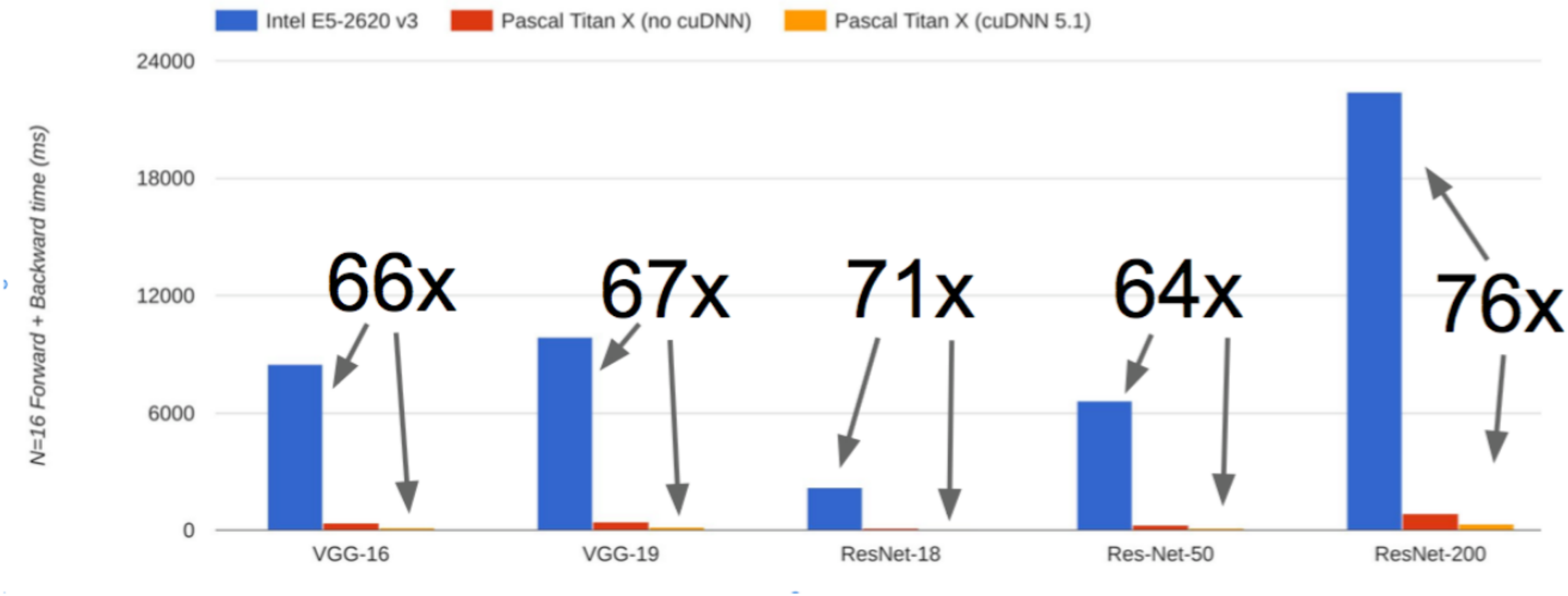


Figure credit: J. Johnson

GPUs for deep learning

NVIDIA GPUs ARE PROGRAMMED THROUGH CUDA [Compute Unified Device Architecture]

ANOTHER ALTERNATIVE IS OPENCL, SUPPORTED BY SEVERAL MANUFACTURES, LESS INVESTMENT [Way less used]

CuDNN IS A LIBRARY FOR SPECIFIC DEEP LEARNING COMPUTATIONS ON NVIDIA GPUs

THE PRICE TO PAY?

1. LARGE NUMBER OF PARAMETERS IMPLIES LARGE DATASETS TO TRAIN

2. LOOSE EVEN MORE DEGREE OF CONTROL OF WHAT THE ALGORITHM IS DOING SINCE THE FEATURE EXTRACTION PROCESS BECOMES UNSUPERVISED

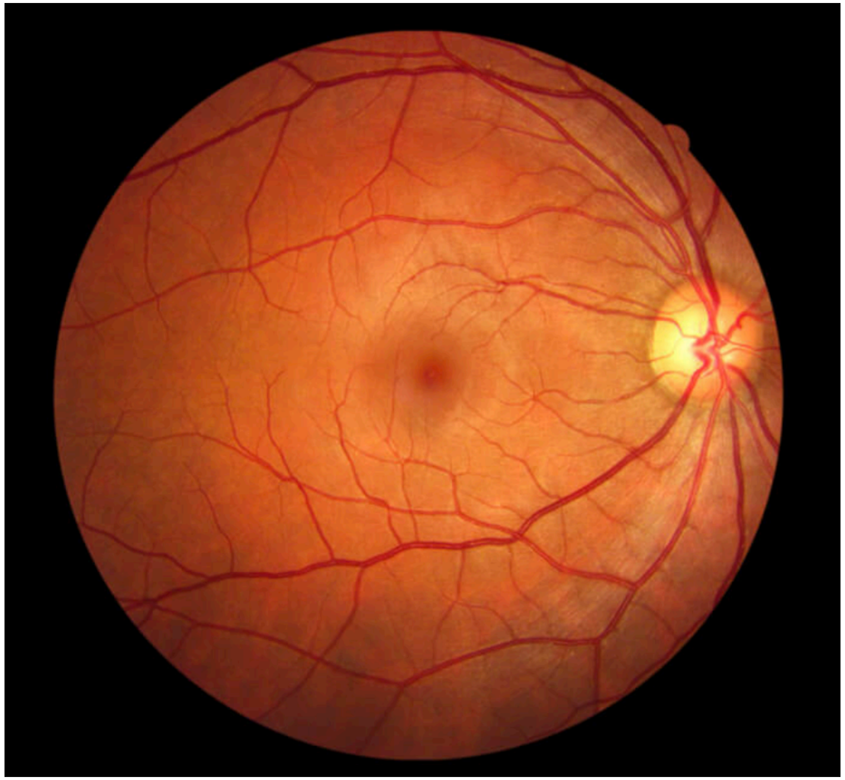


IMAGE OF THE BACK OF THE EYE

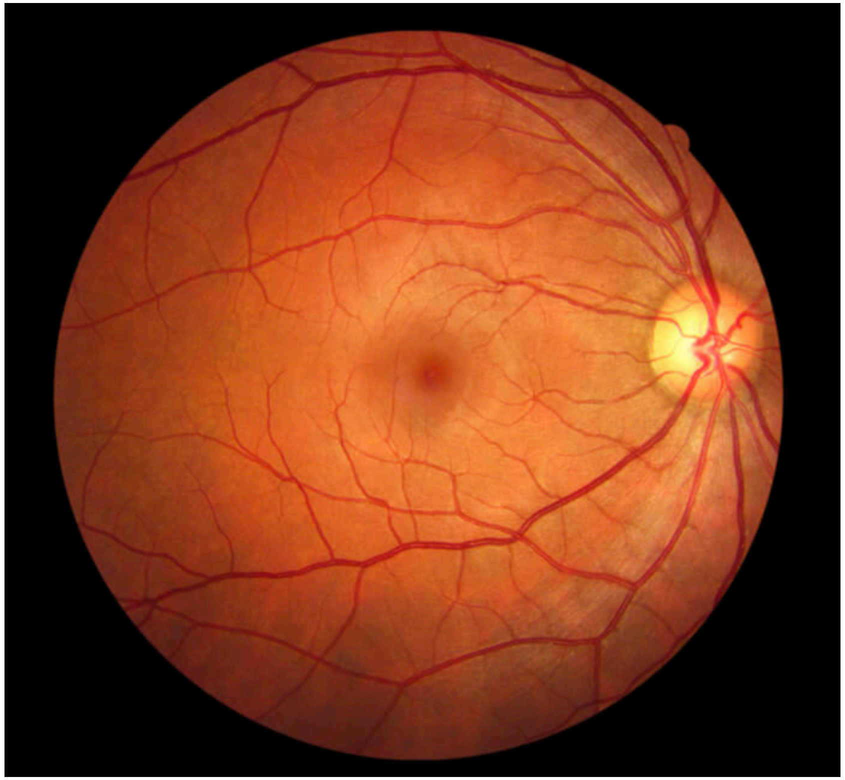


IMAGE OF THE BACK OF THE EYE

DEEP LEARNING CAN IDENTIFY THE PATIENT'S GENDER WITH 95% ACCURACY

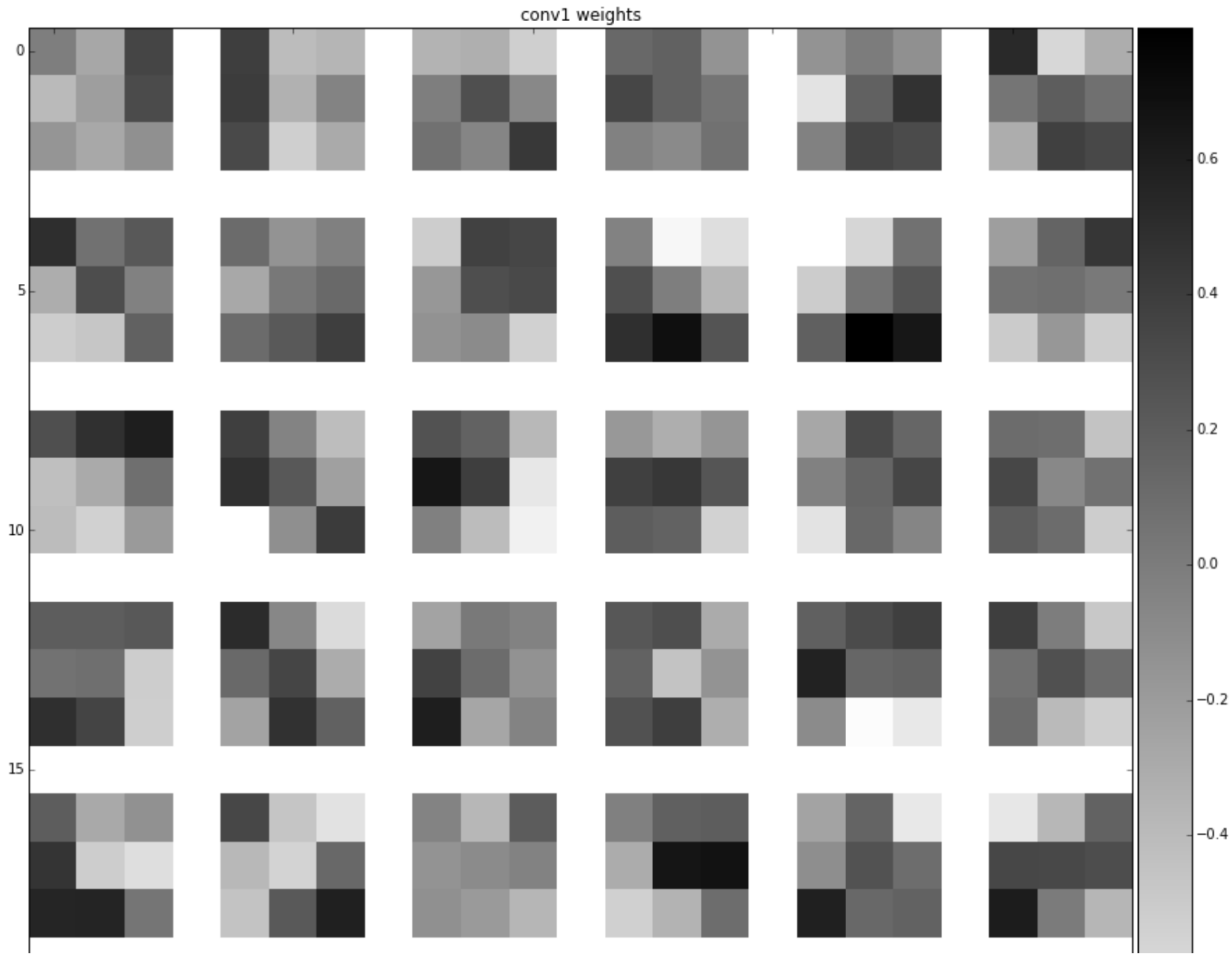
VISUALIZING CNNs
[what happens inside a CNN?]

DEEP NETWORKS ARE “BLACK BOXES”?

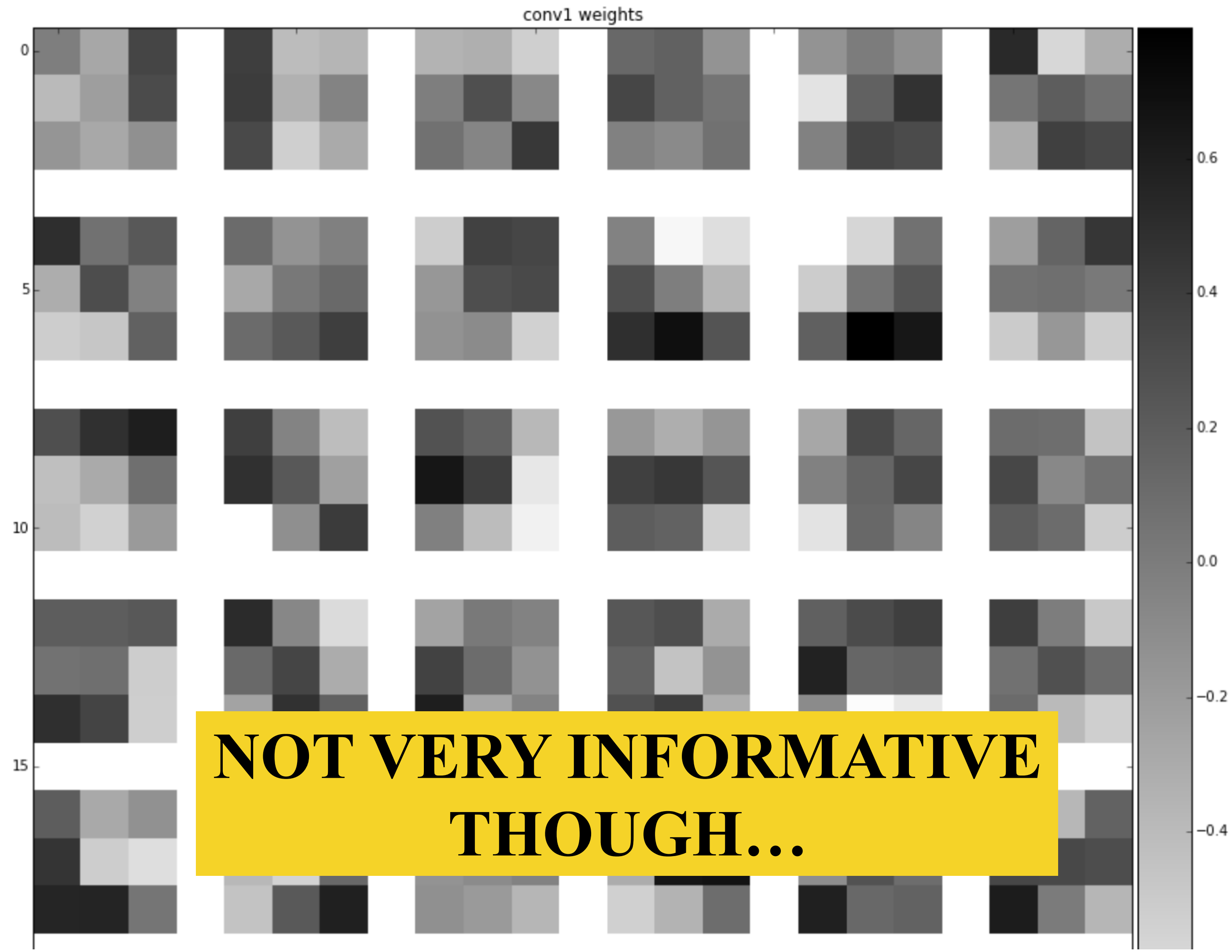
INTERPRETING THE RESULTS IS EXTREMELY DIFFICULT

THIS IS TRUE BUT A LOT OF WORK IS DONE TO UNVEIL THEIR BEHAVIOR

THE SIMPLEST APPROACH IS TO VISUALIZE THE LEARNED WEIGHTS AT INTERMEDIATE LAYERS



THE SIMPLEST APPROACH IS TO VISUALIZE THE LEARNED WEIGHTS AT INTERMEDIATE LAYERS



NOT VERY INFORMATIVE THOUGH…

THE SIMPLEST APPROACH IS TO VISUALIZE THE LEARNED WEIGHTS AT INTERMEDIATE LAYERS

IN KERAS:

```
# build model
model = Sequential()
model.add(Convolution2D(depth, conv_size0, conv_size0, activation=act,
border_mode='same', name = "conv0",
                        input_shape=(img_channels, img_rows, img_cols),
                        init=initialization, W_constraint=constraint))
model.add(Dropout(dropout_rate_conv))

# get the symbolic outputs of each "key" layer (we gave them unique names).
layer_dict = dict([(layer.name, layer) for layer in model.layers])

layer_dict[layer_name].W.get_value(borrow=True)
W = np.squeeze(W)
print("W shape : ", W.shape)

# plot weights
pl.figure(figsize=(15, 15))
pl.title('conv1 weights')
nice_imshow(pl.gca(), make_mosaic(W, 6, 6), cmap=cm.binary)
```

THE SIMPLEST APPROACH IS TO VISUALIZE THE LEARNED WEIGHTS AT INTERMEDIATE LAYERS

IN KERAS:

give names to layers

```
# build model
model = Sequential()
model.add(Convolution2D(depth, conv_size0, conv_size0, activation=act,
border_mode='same', name = "conv0",
                        input_shape=(img_channels, img_rows, img_cols),
                        init=initialization, W_constraint=constraint))
model.add(Dropout(dropout_rate_conv))

# get the symbolic outputs of each "key" layer (we gave them unique names).
layer_dict = dict([(layer.name, layer) for layer in model.layers])

layer_dict[layer_name].W.get_value(borrow=True)
W = np.squeeze(W)
print("W shape : ", W.shape)

# plot weights
pl.figure(figsize=(15, 15))
pl.title('conv1 weights')
nice_imshow(pl.gca(), make_mosaic(W, 6, 6), cmap=cm.binary)
```

THE SIMPLEST APPROACH IS TO VISUALIZE THE LEARNED WEIGHTS AT INTERMEDIATE LAYERS

IN KERAS:

create dictionary to link layers to names

```
# build model
model = Sequential()
model.add(Convolution2D(depth, conv_size0, conv_size0, activation=act,
border_mode='same', name = "conv0",
                        input_shape=(img_channels, img_rows, img_cols),
                        init=initialization, W_constraint=constraint))
model.add(Dropout(dropout_rate_conv))

# get the symbolic outputs of each "key" layer (we gave them unique names).
layer_dict = dict([(layer.name, layer) for layer in model.layers])

layer_dict[layer_name].W.get_value(borrow=True)
W = np.squeeze(W)
print("W shape : ", W.shape)

# plot weights
pl.figure(figsize=(15, 15))
pl.title('conv1 weights')
nice_imshow(pl.gca(), make_mosaic(W, 6, 6), cmap=cm.binary)
```

THE SIMPLEST APPROACH IS TO VISUALIZE THE LEARNED WEIGHTS AT INTERMEDIATE LAYERS

IN KERAS:

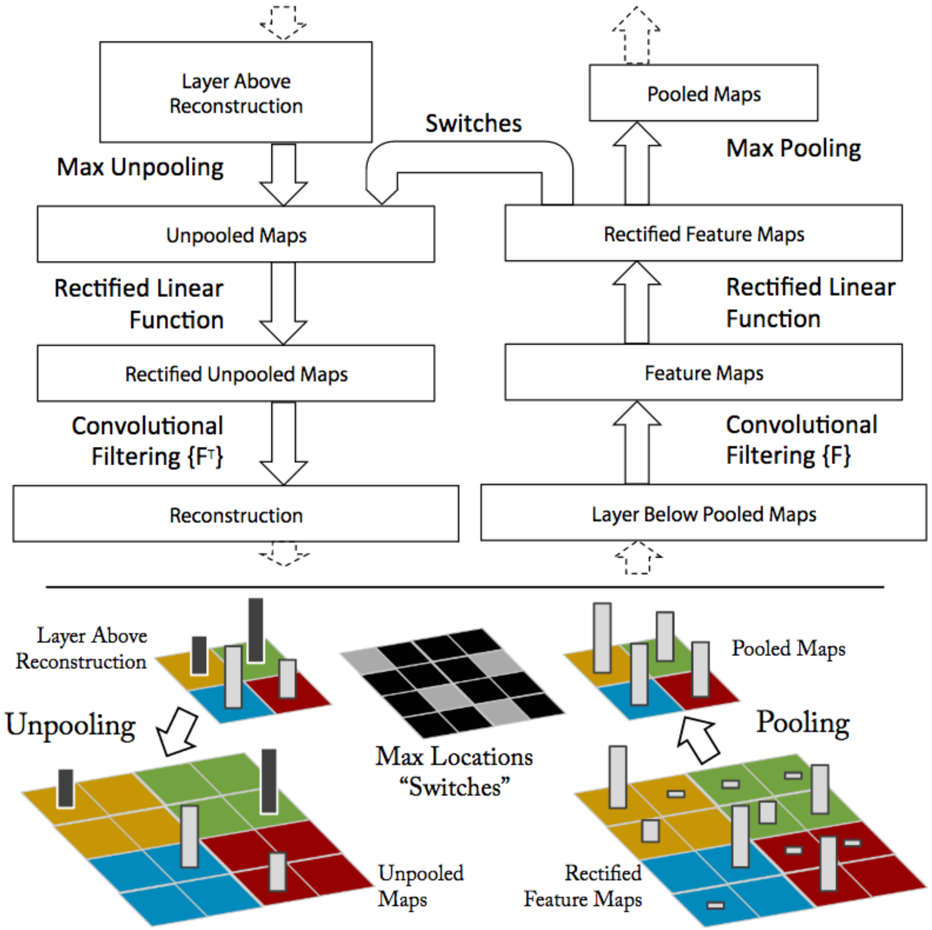
for a given name, get the weights

```
# build model
model = Sequential()
model.add(Convolution2D(depth, conv_size0, conv_size0, activation=act,
border_mode='same', name = "conv0",
                        input_shape=(img_channels, img_rows, img_cols),
                        init=initialization, W_constraint=constraint))
model.add(Dropout(dropout_rate_conv))

# get the symbolic outputs of each "key" layer (we gave them unique names).
layer_dict = dict([(layer.name, layer) for layer in model.layers])

layer_dict[layer_name].W.get_value(borrow=True)
W = np.squeeze(W)
print("W shape : ", W.shape)

# plot weights
pl.figure(figsize=(15, 15))
pl.title('conv1 weights')
nice_imshow(pl.gca(), make_mosaic(W, 6, 6), cmap=cm.binary)
```

USING THE SAME IDEA, ONE CAN ALSO VISUALIZE THE FEATURE MAPS AT INTERMEDIATE LAYERS

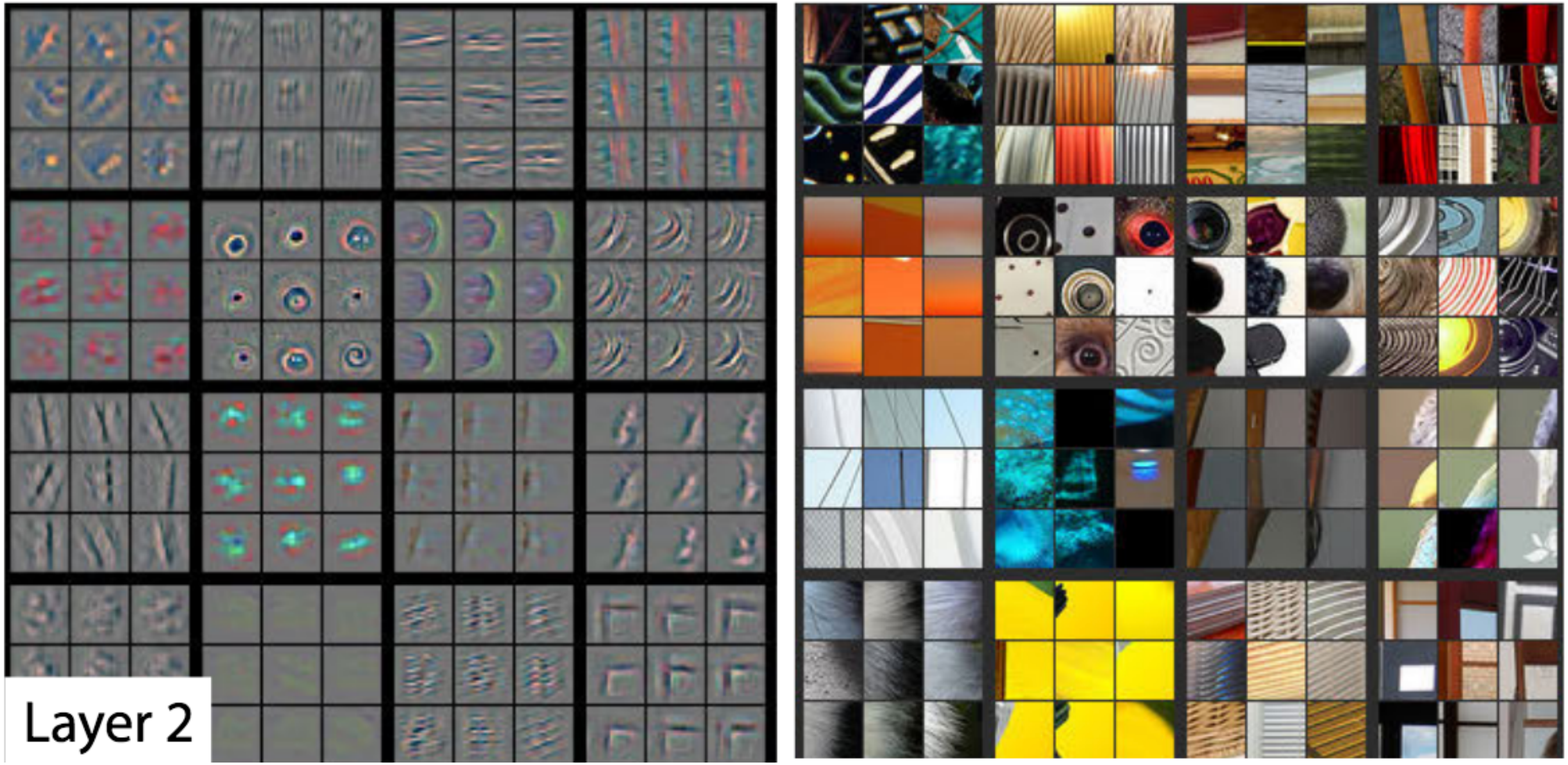
THIS HELPS TRACING THE FEATURES LEARNED BY THE NETWORK

USE "DECONVNETS" TO MAP BACK THE FEATURE MAP INTO THE PIXEL SPACE



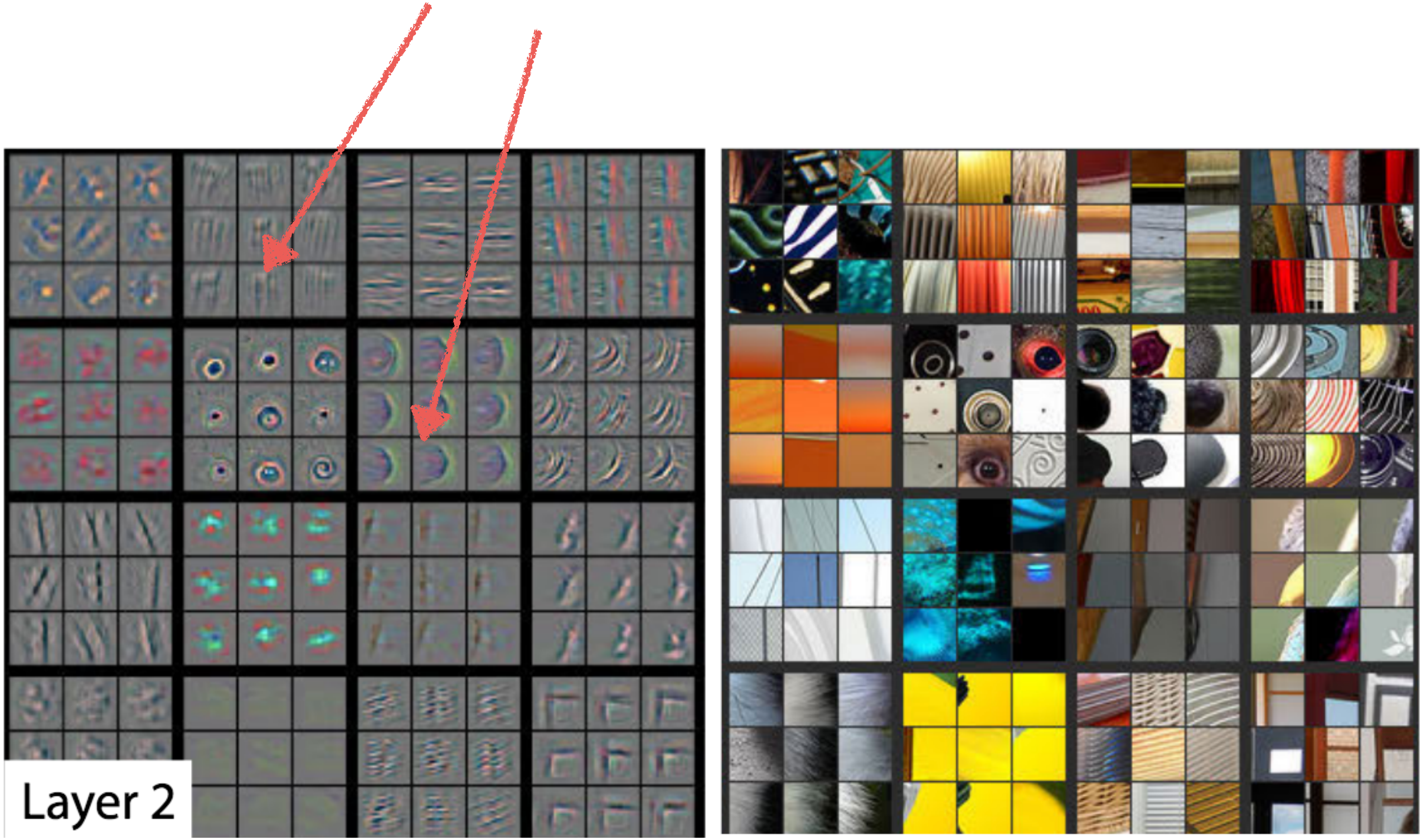
IT ALLOWS TO SEE WHICH
REGIONS OF THE INPUT GENERATED
A MAXIMUM RESPONSE
IN A NEURON

Zeiler+14



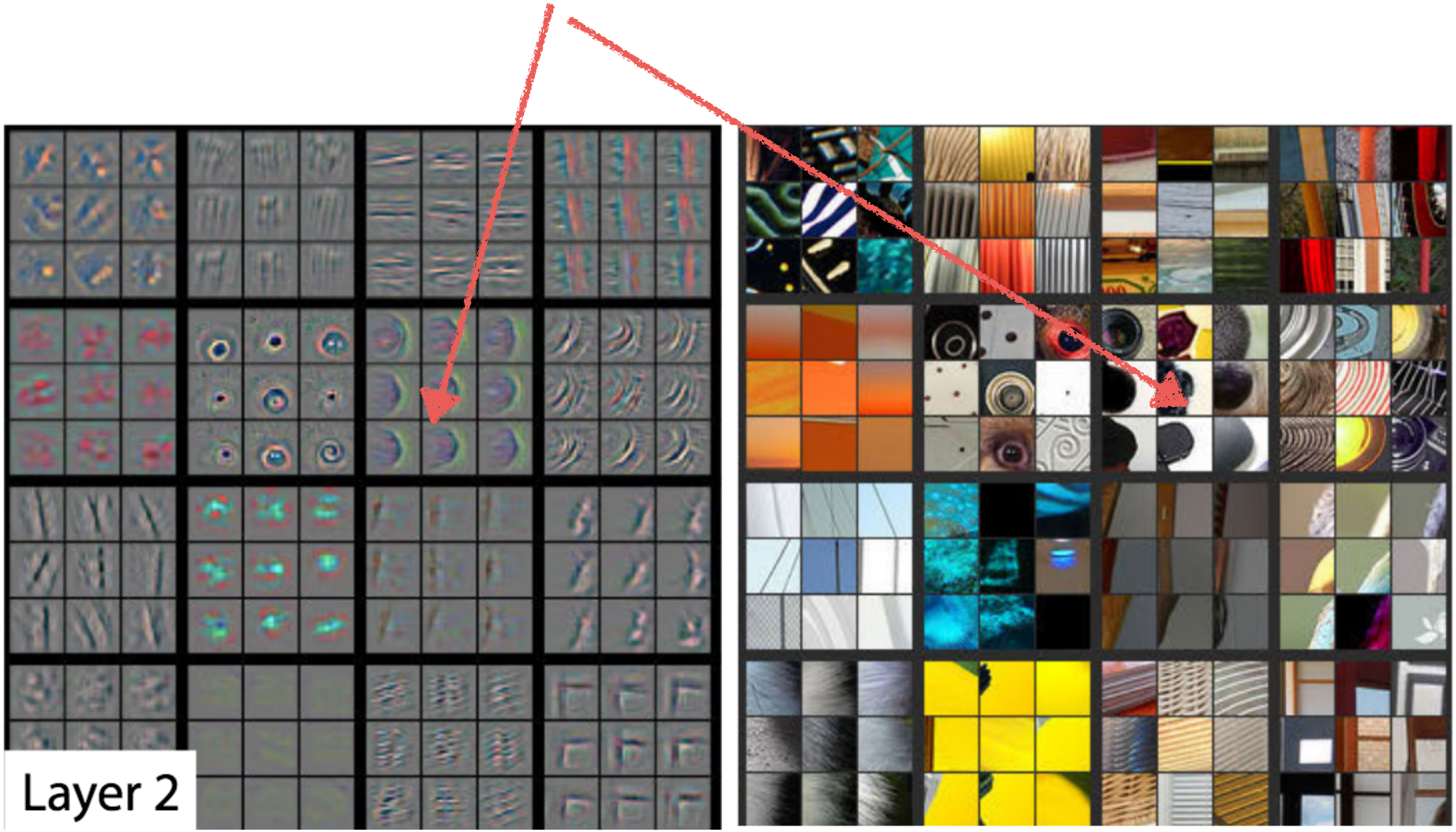
Zeiler+14

EVERY BLOCK OF 9 SHOWS
THE 9 STRONGEST RESPONSES TO A GIVEN FILTER OF LAYER2



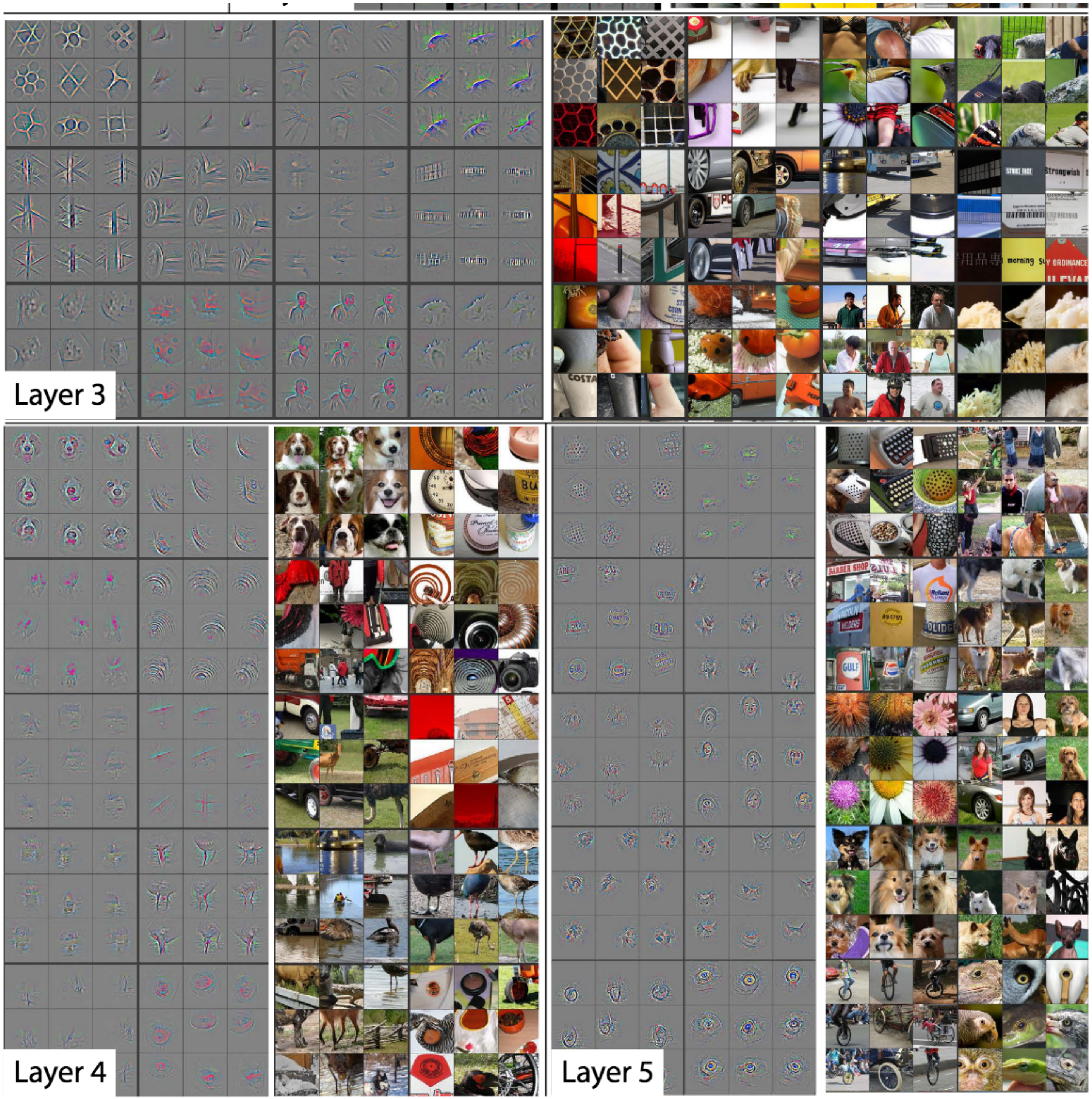
Zeiler+14

THE CORRESPONDING REGIONS OF IMAGES THAT GENERATED THE MAXIMUM RESPONSE



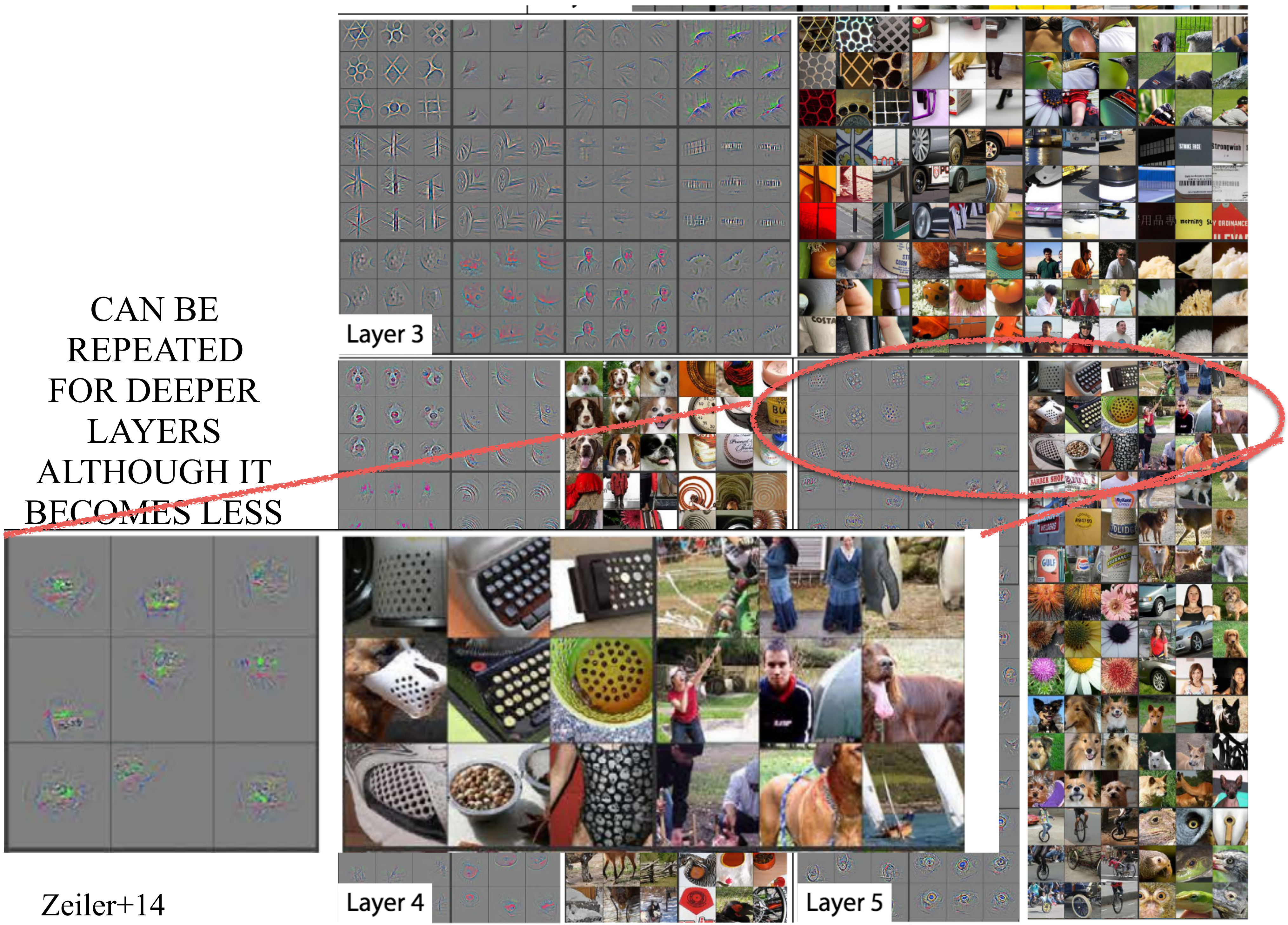
Zeiler+14

CAN BE REPEATED FOR DEEPER LAYERS ALTHOUGH IT BECOMES LESS INTUITIVE



Zeiler+14

CAN BE REPEATED FOR DEEPER LAYERS ALTHOUGH IT BECOMES LESS



Zeiler+14

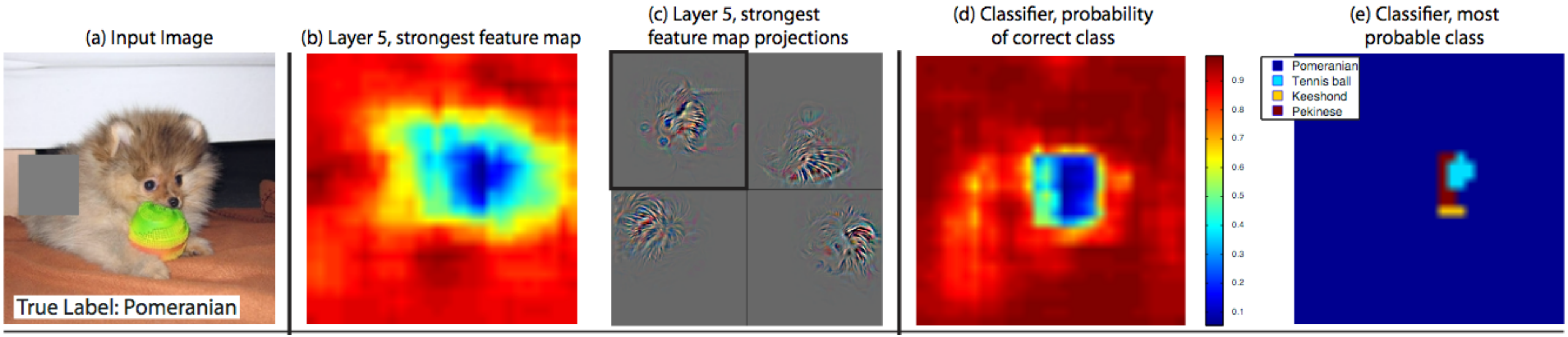
KERAS IMPLEMENTATION OF VISUALIZATIONS THROUGH DECONVNETS

https://github.com/jalused/Deconvnet-keras

OCCLUSION SENSITIVITY TRIES ALSO TO FIND THE REGION OF THE IMAGE THAT TRIGGERED THE NETWORK DECISION BY MASKING DIFFERENT REGIONS OF THE INPUT IMAGE AND ANALYZING THE NETWORK OUPUT

IT ALLOWS TO IF THE NETWORK IS TAKING THE DECISIONS BASED ON THE EXPECTED FEATURES

<u>VERY TIME CONSUMING!</u>



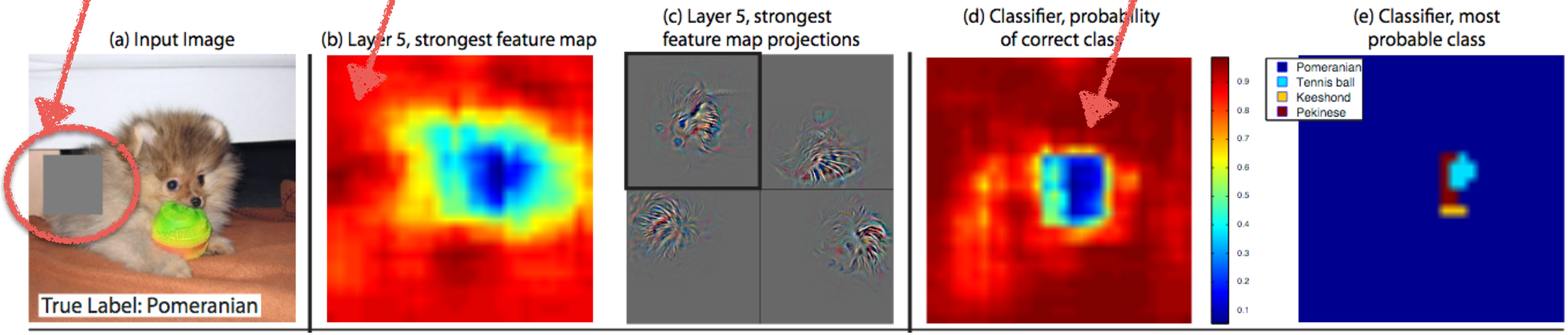
Zeiler+14

OCCLUSION SENSITIVITY TRIES ALSO TO FIND THE REGION OF THE IMAGE THAT TRIGGERED THE NETWORK DECISION BY MASKING DIFFERENT REGIONS OF THE INPUT IMAGE AND ANALYZING THE NETWORK OUPUT

for every position
of the square the maximum response of a given layer
is averaged

the output probability as a
function of the occluding square
position

occluded region



Zeiler+14

INCEPTIONISM - DEEP DREAM

THE IDEA BEHIND INCEPTIONISM TECHNIQUES IS TO INVERT THE NETWORK TO GENERATE AN IMAGE THAT MAXIMIZES THE OUTPUT SCORE

image I

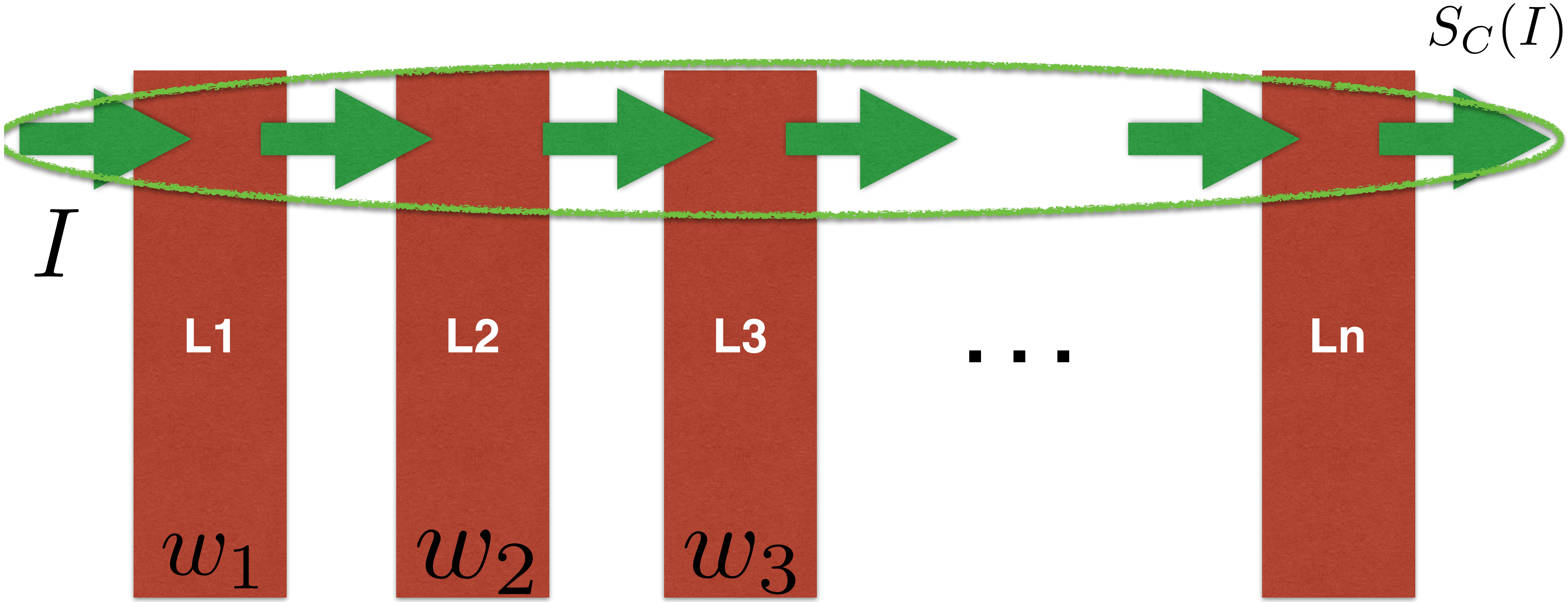
Score of class c for image I

$$\underset{I}{arg\,max}\; S_c(I) - \lambda ||I||_2^2$$

TRY TO FIND AN IMAGE THAT GENERATES A HIGH SCORE FOR A GIVEN CLASS

Simonyan+14

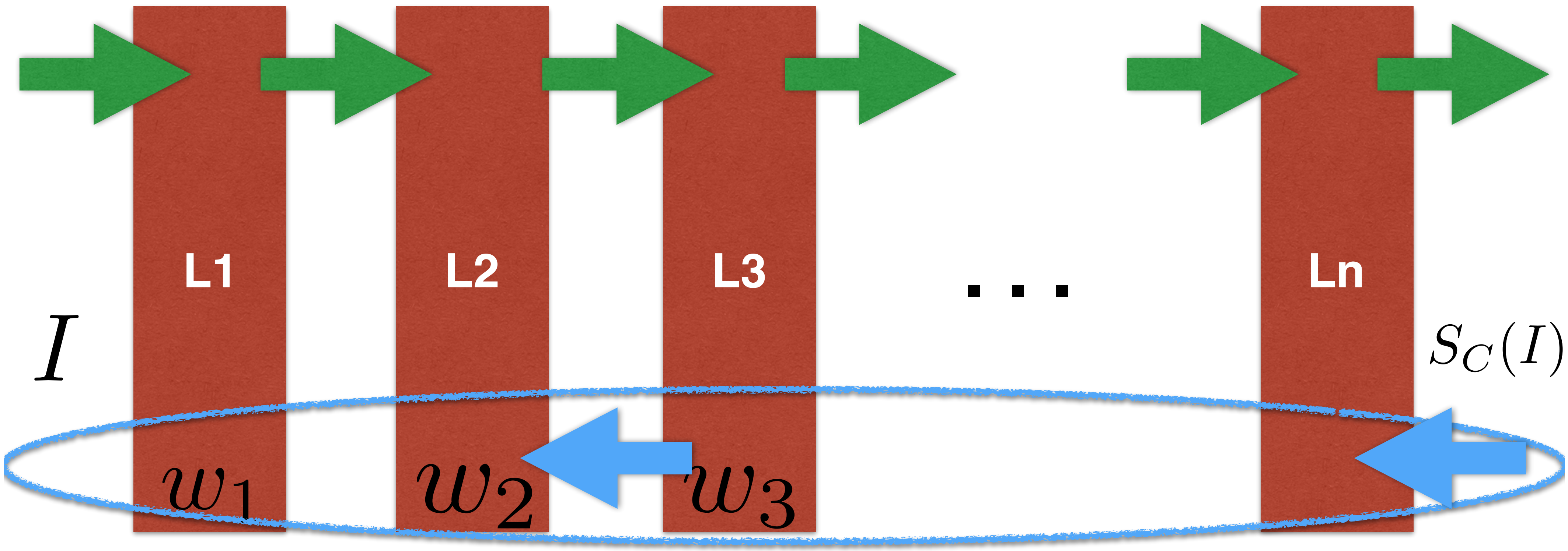
INCEPTIONISM - DEEP DREAM



DURING THE TRAINING PHASE THE WEIGHTS ARE LEARNED TO MAP I INTO Sc

Simonyan+14

INCEPTIONISM - DEEP DREAM

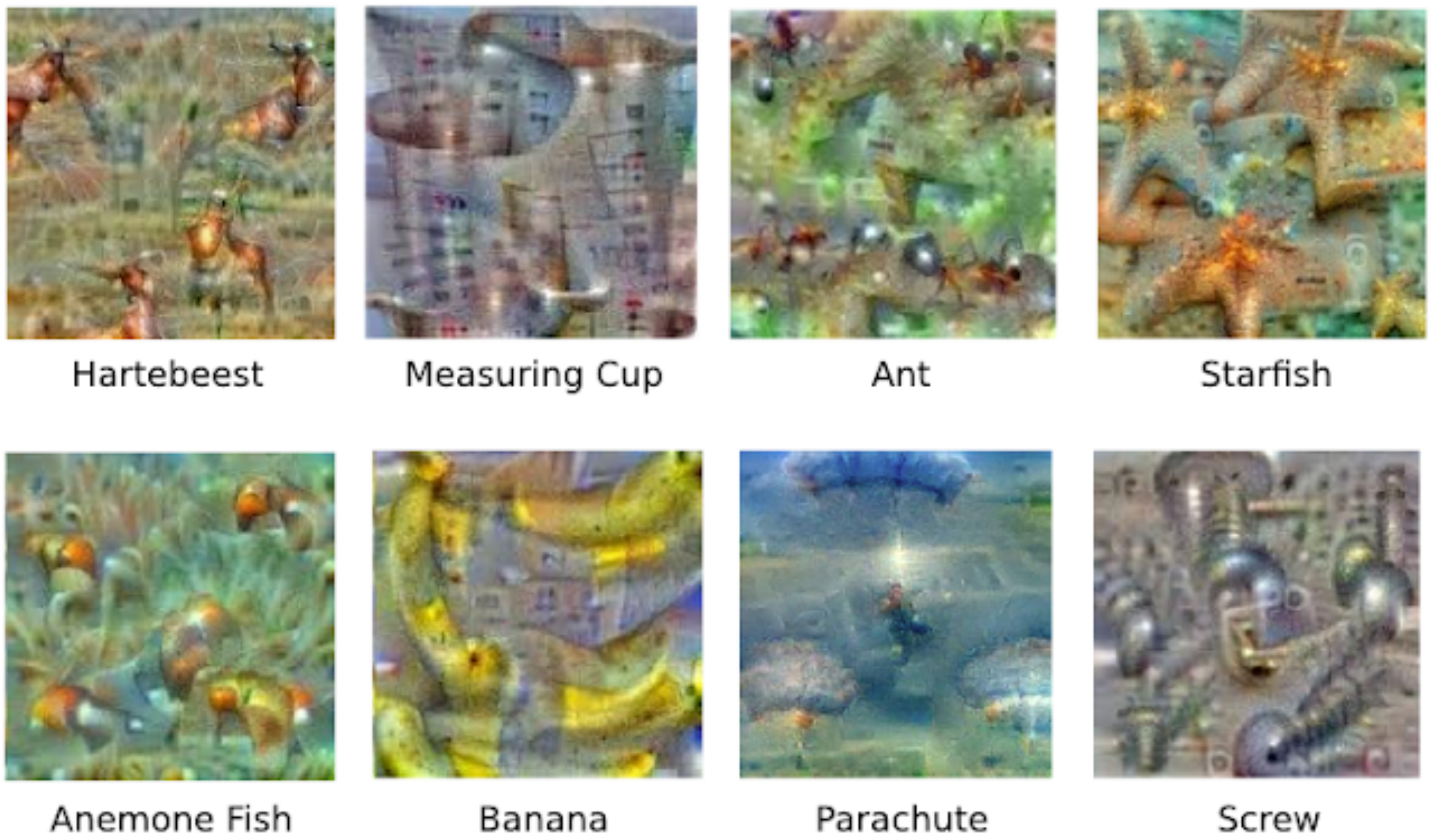


DURING THE RECONSTRUCTION PHASE, I IS LEARNT TRHOUG BACKPROPAGATION KEEPING THE WEIGHTS FIXED

Simonyan+14

INCEPTIONISM - DEEP DREAM

RESULTS REVEAL INTERESTING INFORMATION ON HOW THE NETWORKS BUILD REPRESENTATIONS OF OBJECTS



INCEPTIONISM - DEEP DREAM

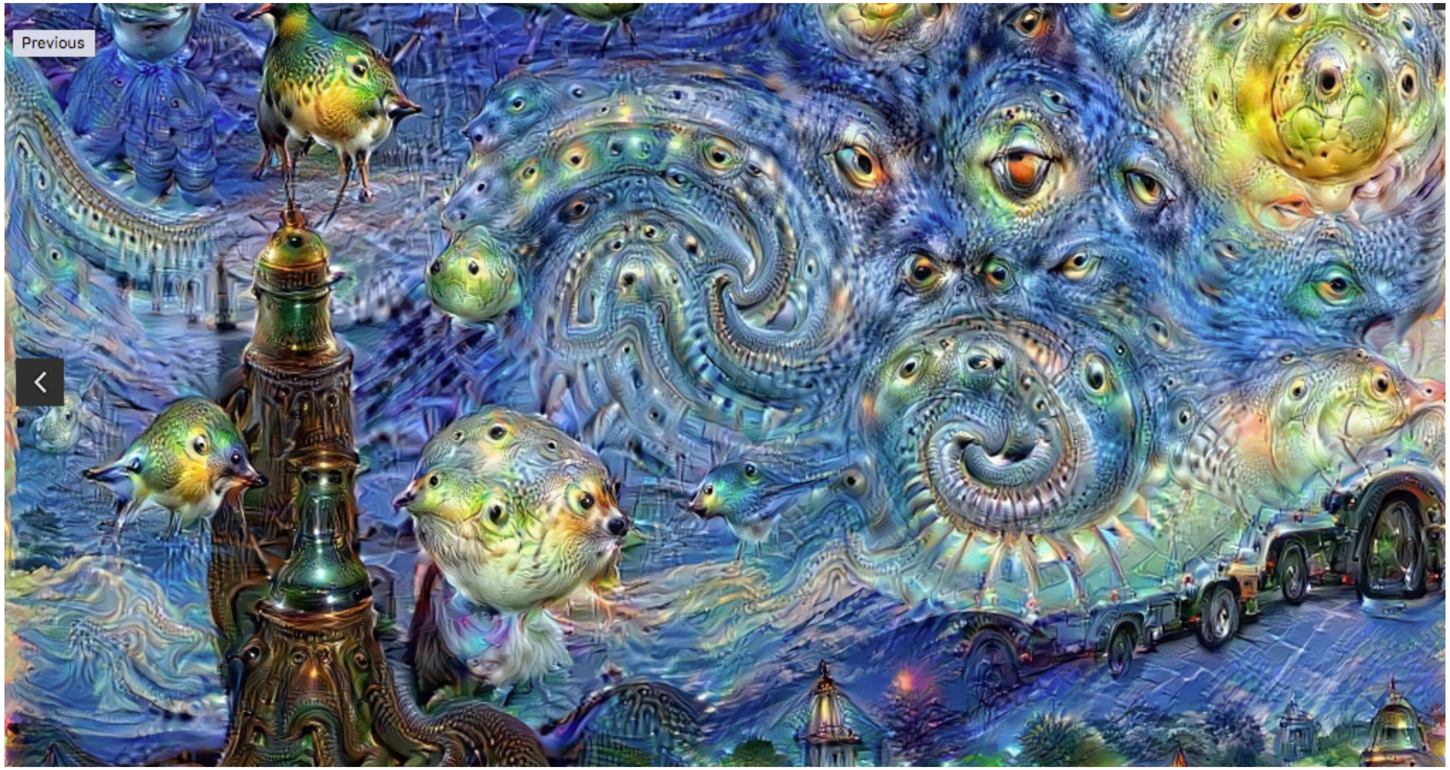
RESULTS REVEAL INTERESTING INFORMATION ON HOW THE NETWORKS BUILD REPRESENTATIONS OF OBJECTS

SOME STRANGE CASES…

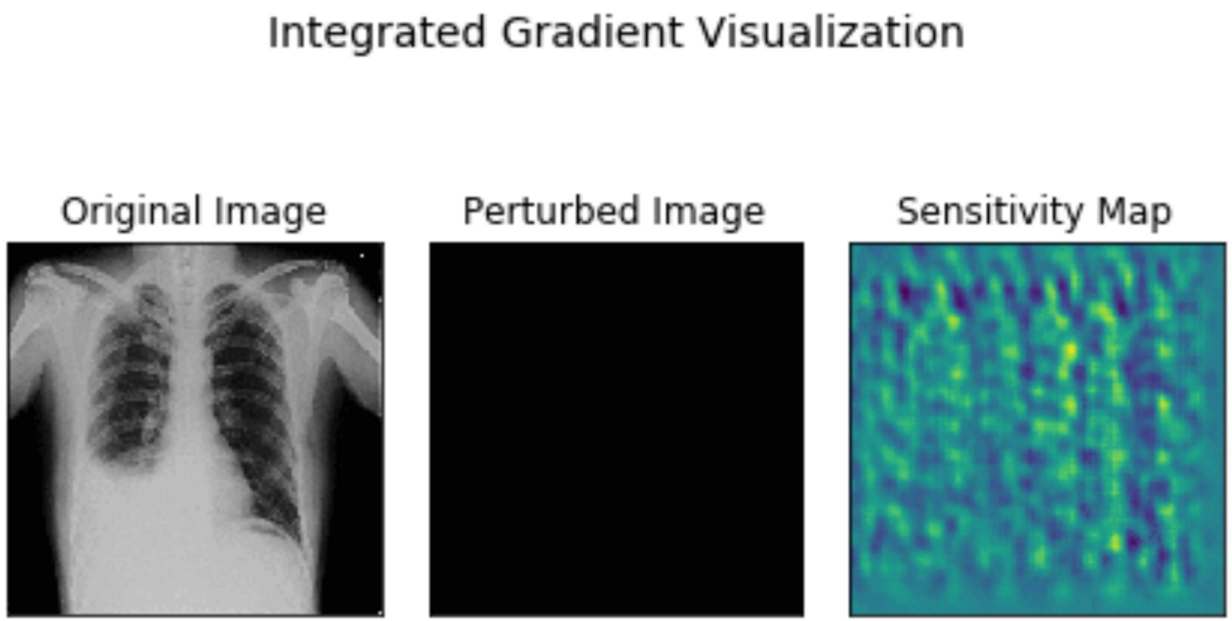
DEEP DREAM

https://deepdreamgenerator.com/

IT HAS NOW BECOME A SORT OF ART?

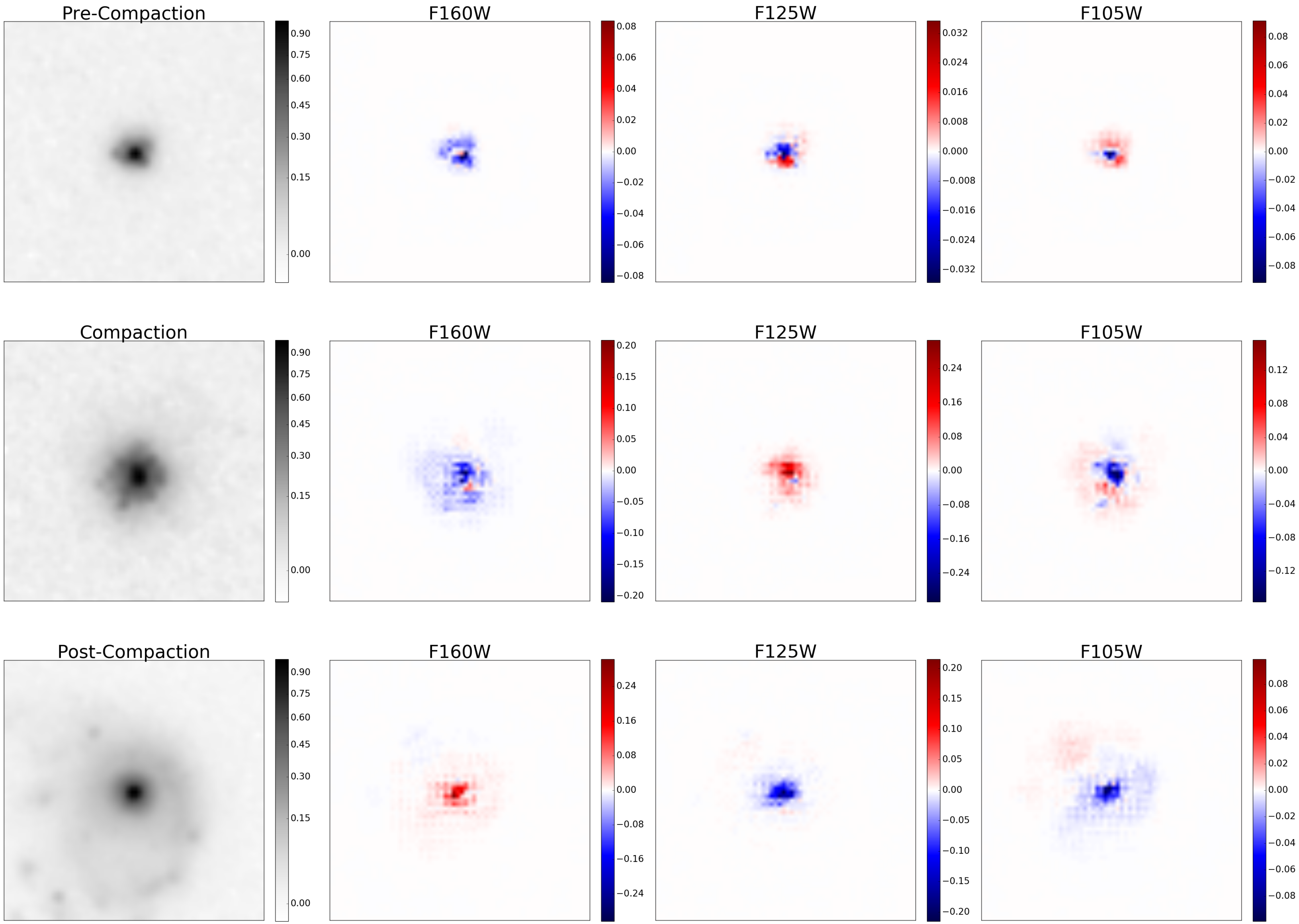


INTEGRATED GRADIENTS



Sundararajan+17

INTEGRATED GRADIENTS



INTEGRATED GRADIENTS

KERAS IMPLEMENTATION:
https://github.com/hiranumn/IntegratedGradients

PART IV: IMAGE 2 IMAGE NETWORKS + INTRODUCTION TO GENERATIVE MODELS