VISUALIZING CNNs
[understanding CNNs decisions]

Attribution techniques

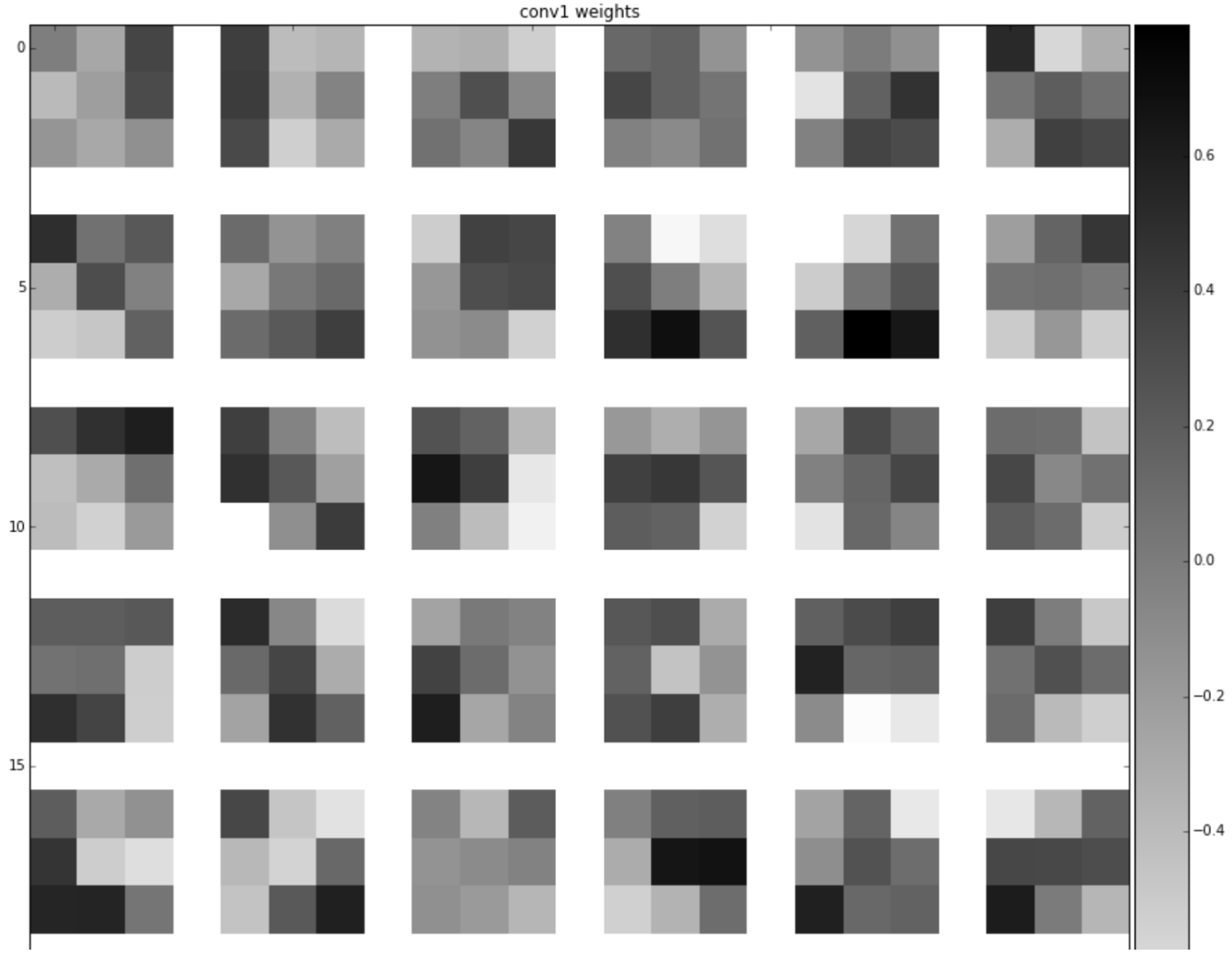
DEEP NETWORKS ARE “BLACK BOXES”?

INTERPRETING THE RESULTS IS
EXTREMELY DIFFICULT

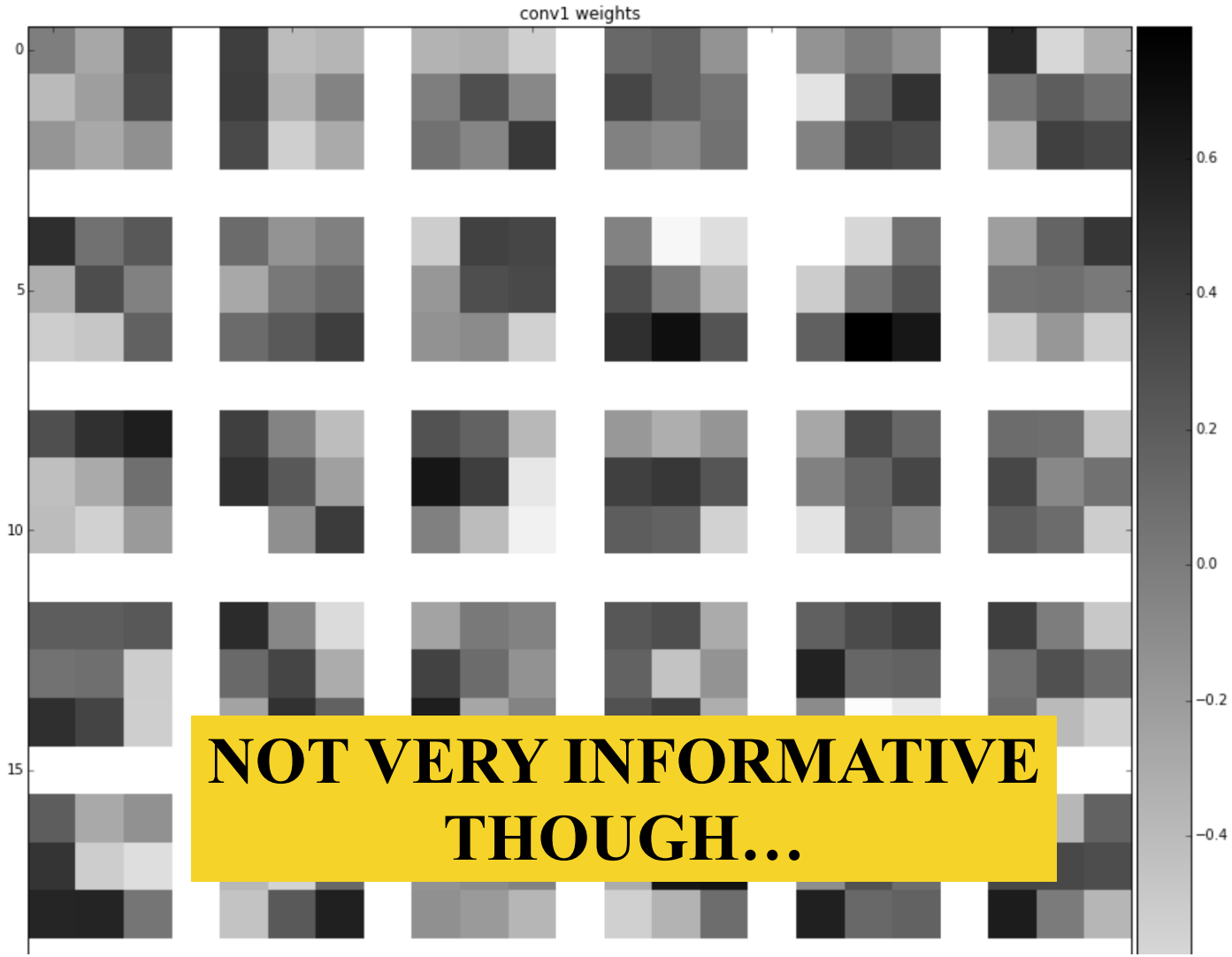
THIS IS TRUE BUT A LOT OF WORK
IS DONE TO UNVEIL THEIR BEHAVIOR

EXPLORING THE FEATURE MAPS

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

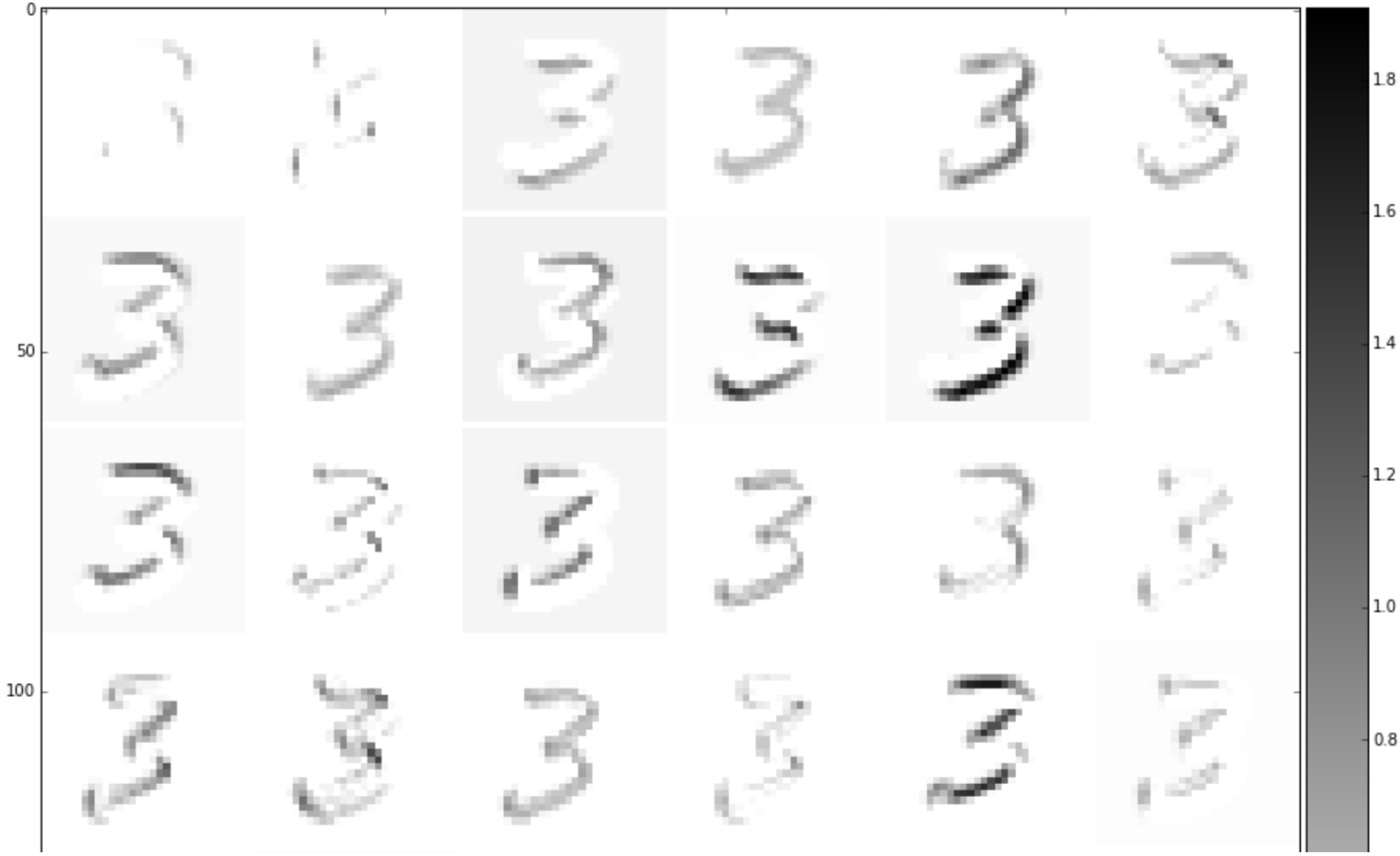


THE SIMPLEST APPROACH IS TO VISUALIZE THE LEARNED WEIGHTS AT INTERMEDIATE LAYERS

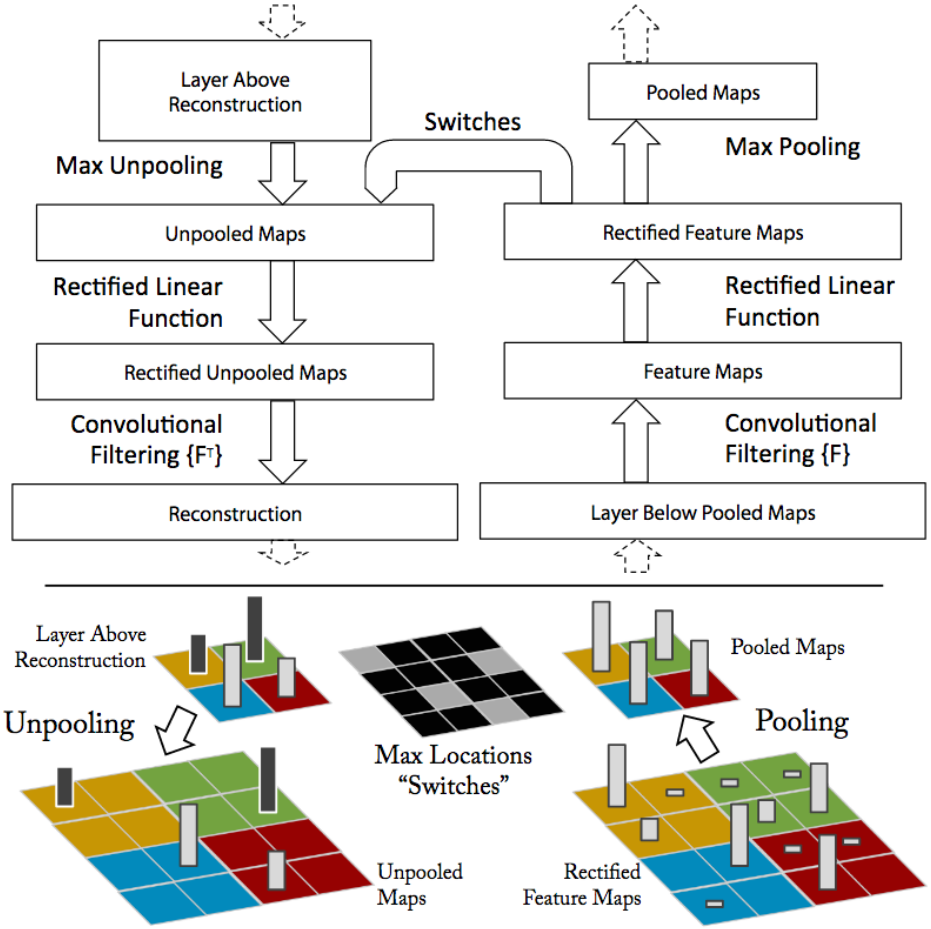


USING THE SAME IDEA, ONE CAN ALSO VISUALIZE THE FEATURE MAPS AT INTERMEDIATE LAYERS

THIS HELPS TRACING THE FEATURES LEARNED BY THE NETWORK

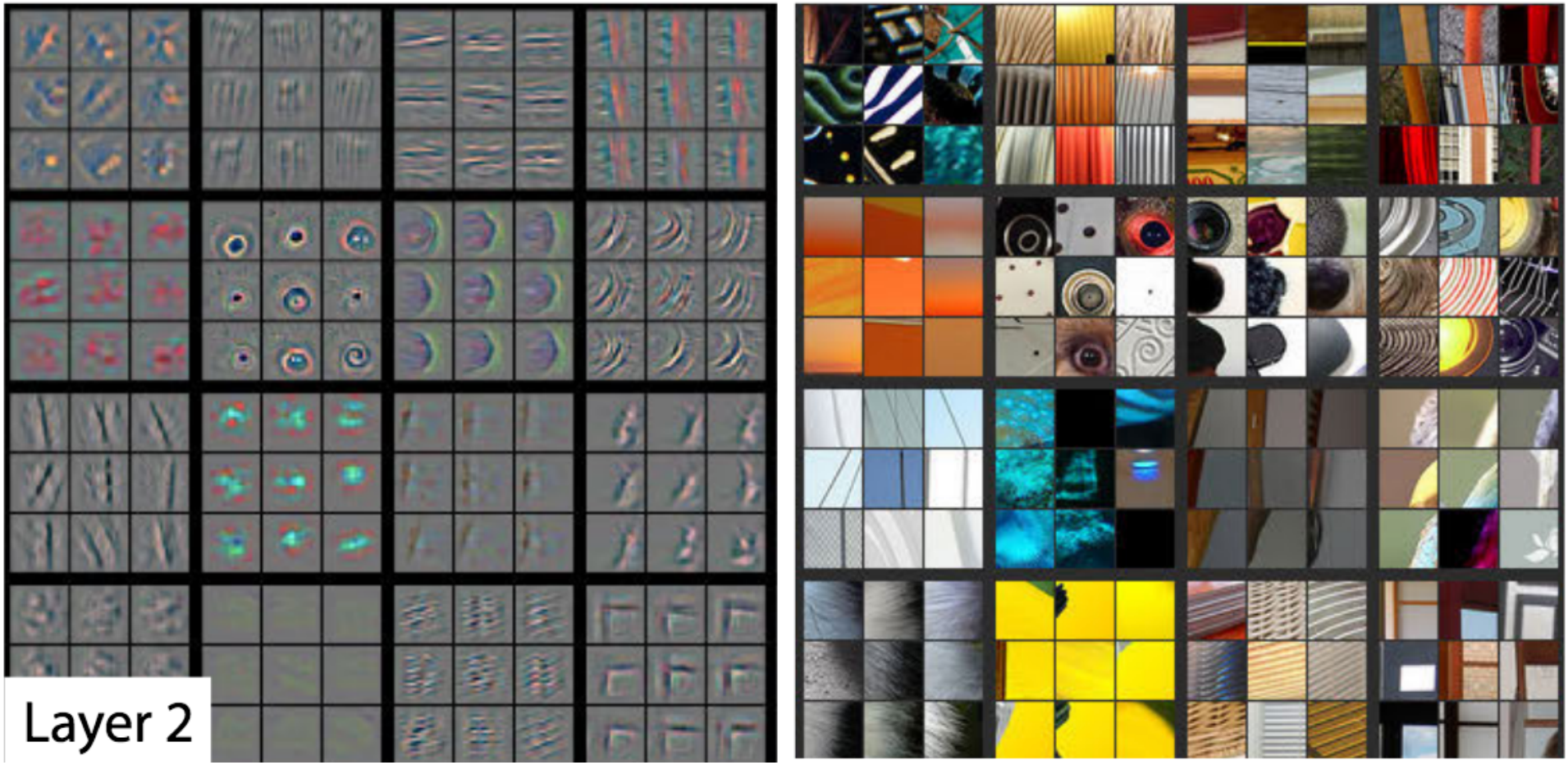

0
50
100
1.8
1.6
1.4
1.2
1.0
0.8

USE “DECONVNETS” TO MAP BACK THE FEATURE MAP INTO THE PIXEL SPACE



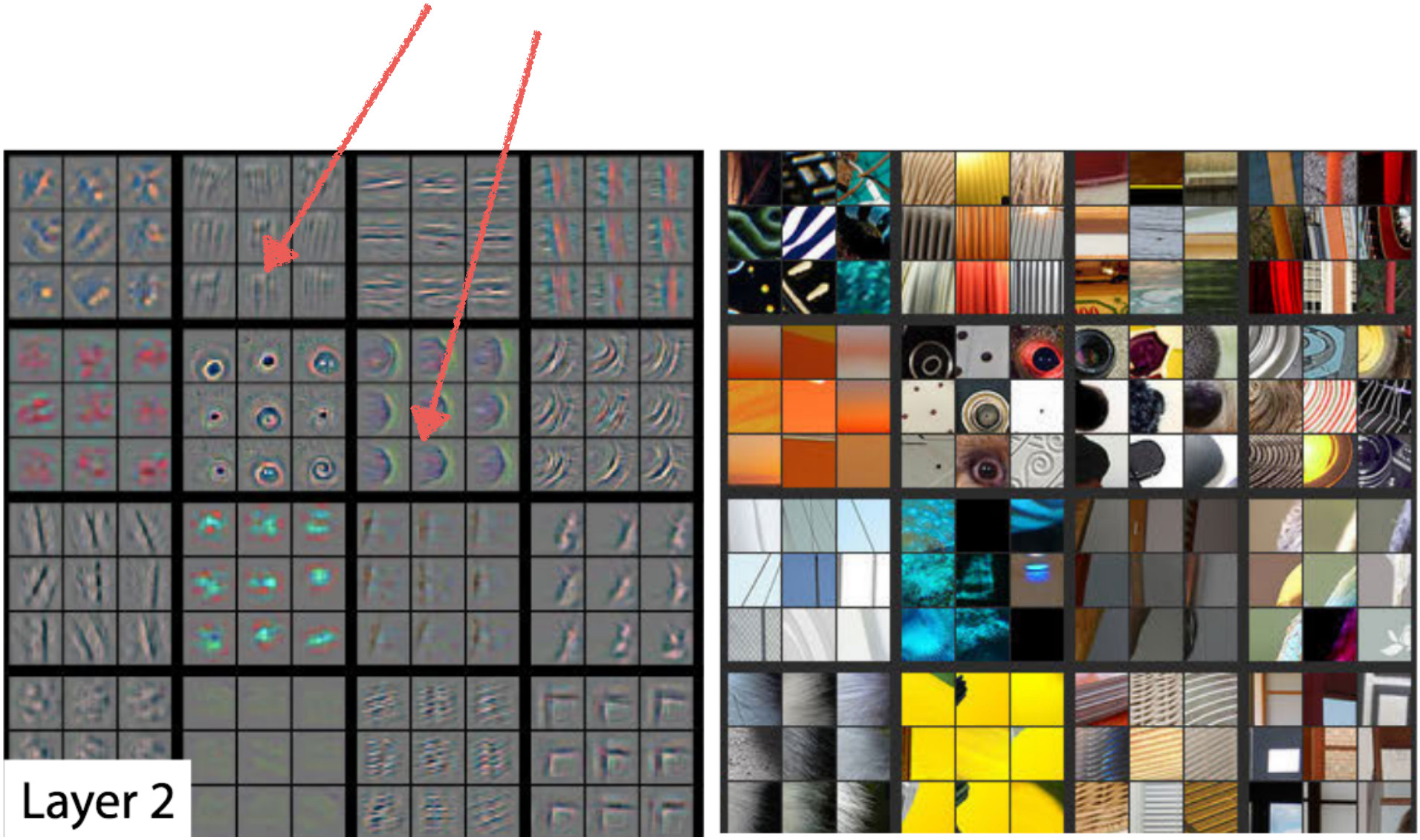
IT ALLOWS TO SEE
WHICH
REGIONS OF THE INPUT
GENERATED
A MAXIMUM RESPONSE
IN A NEURON

Zeiler+14



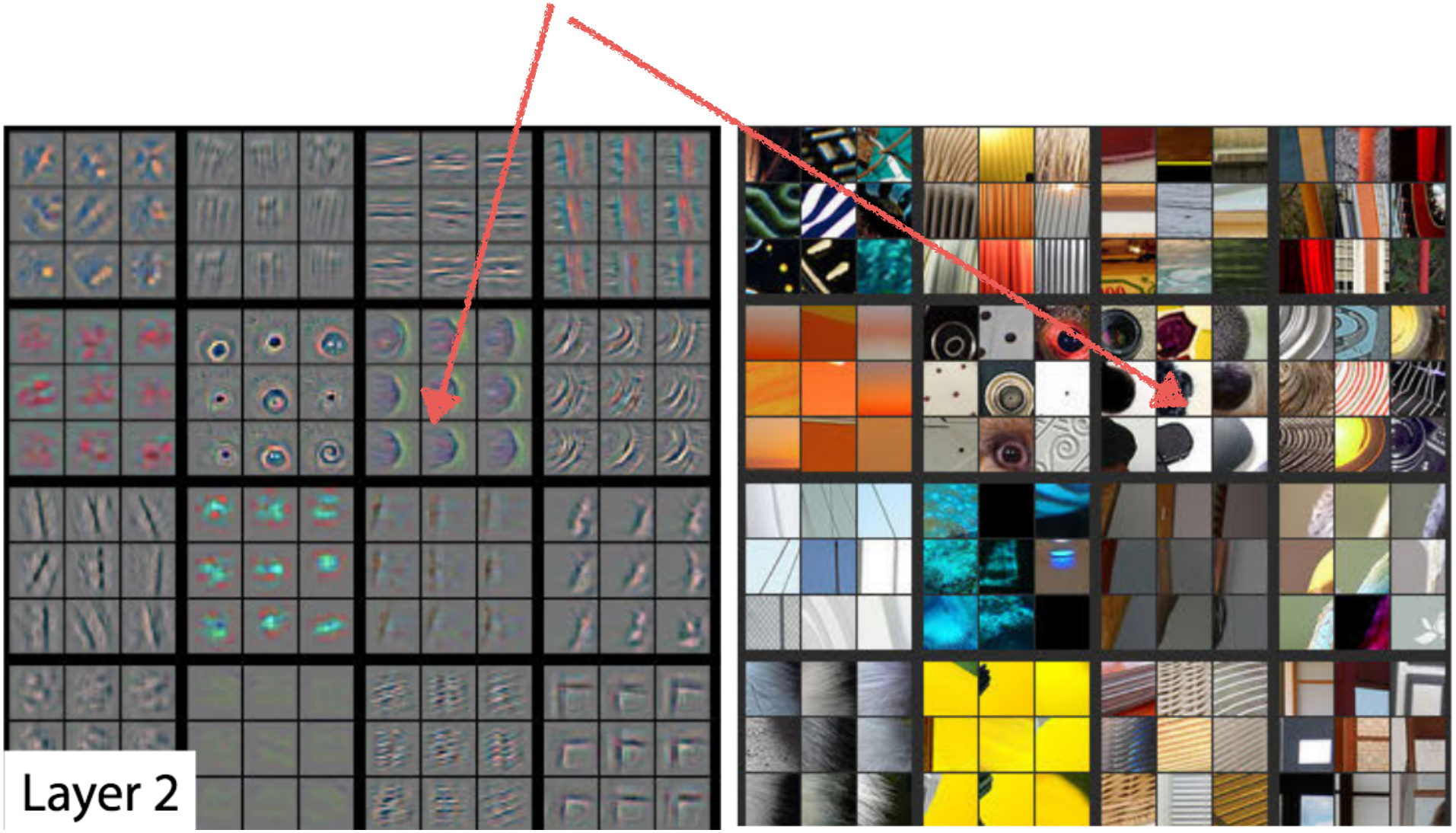
Zeiler+14

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



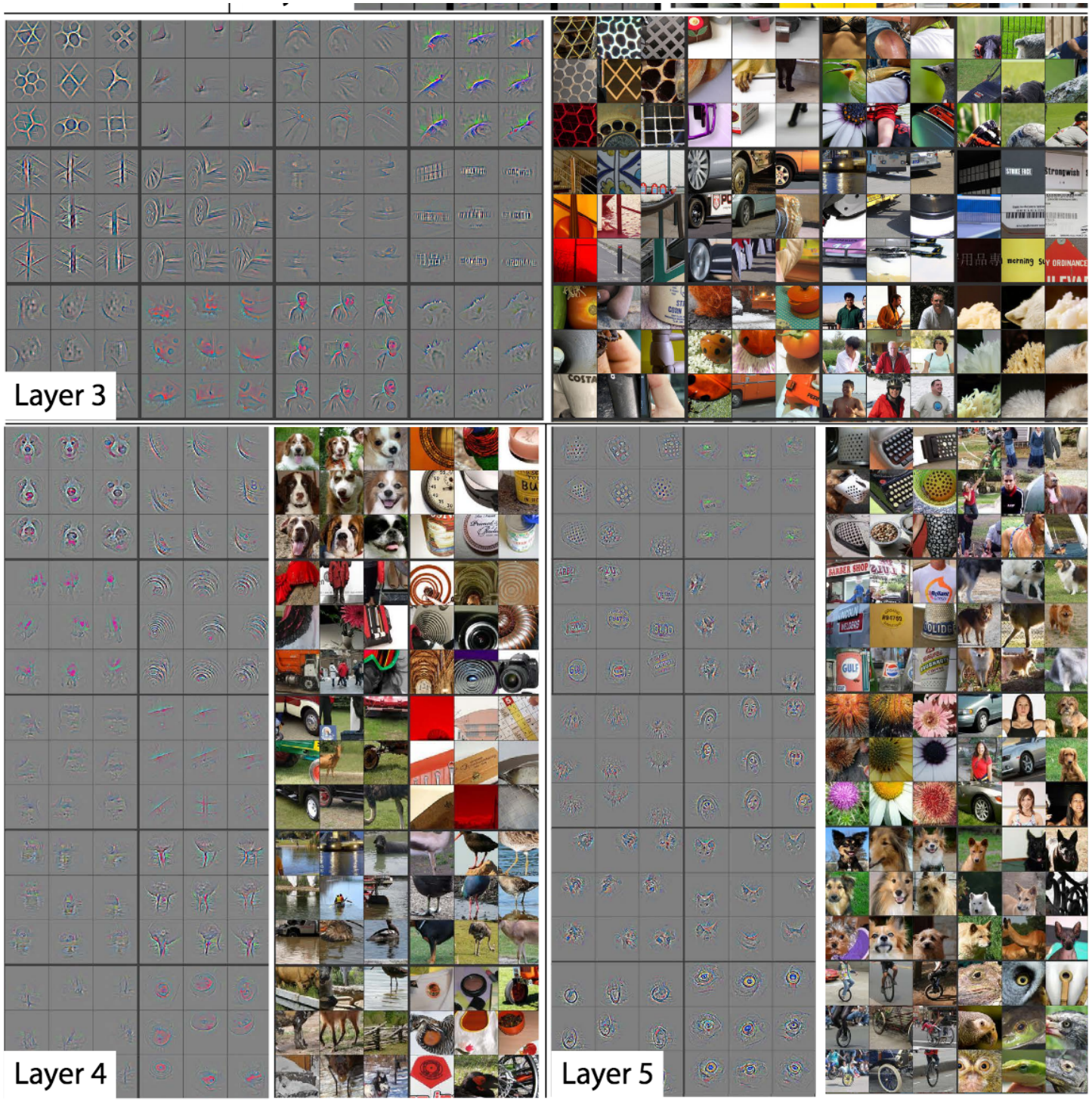
Zeiler+14

THE CORRESPONDING REGIONS OF IMAGES THAT GENERATED THE MAXIMUM RESPONSE



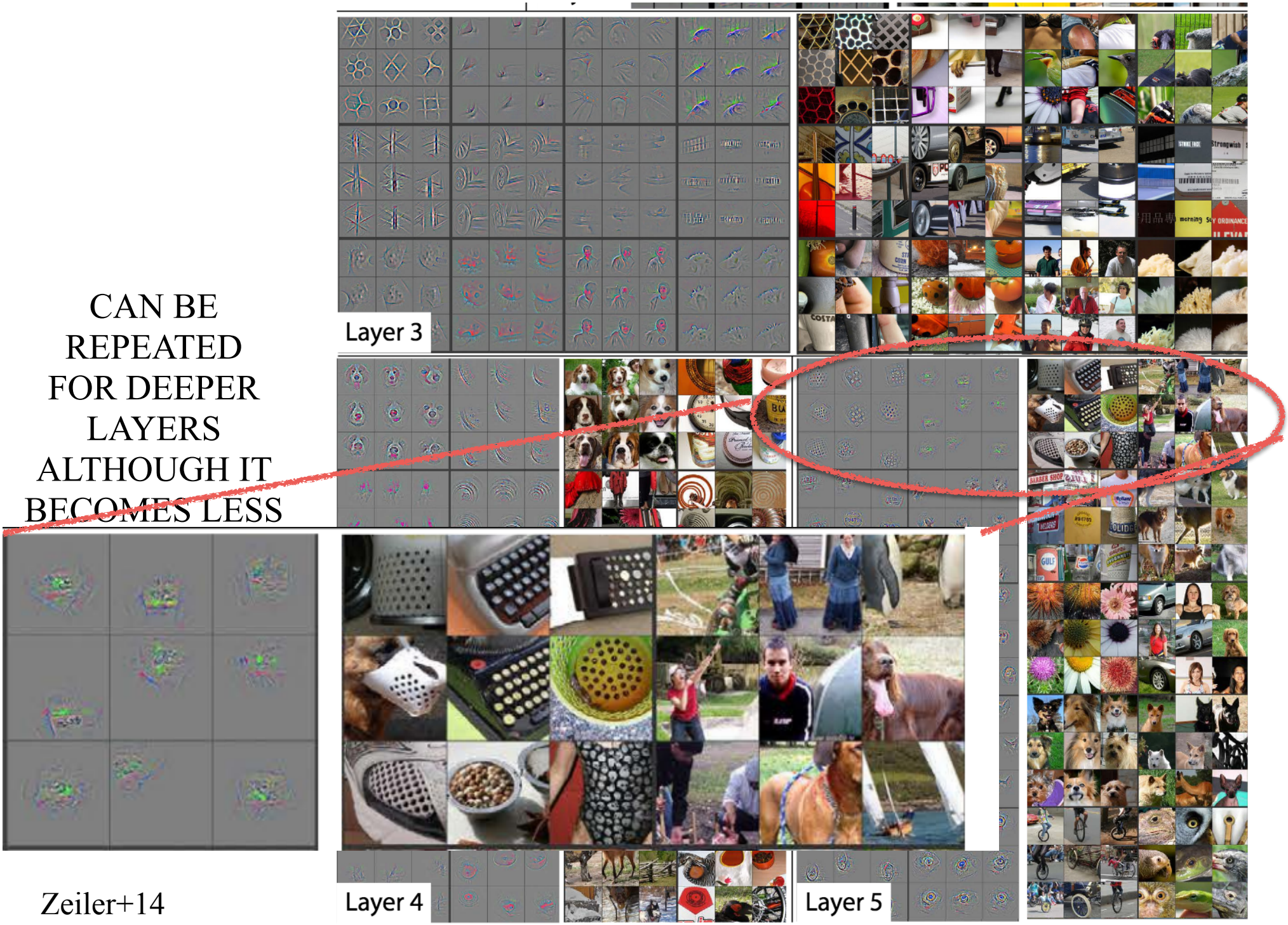
Zeiler+14

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

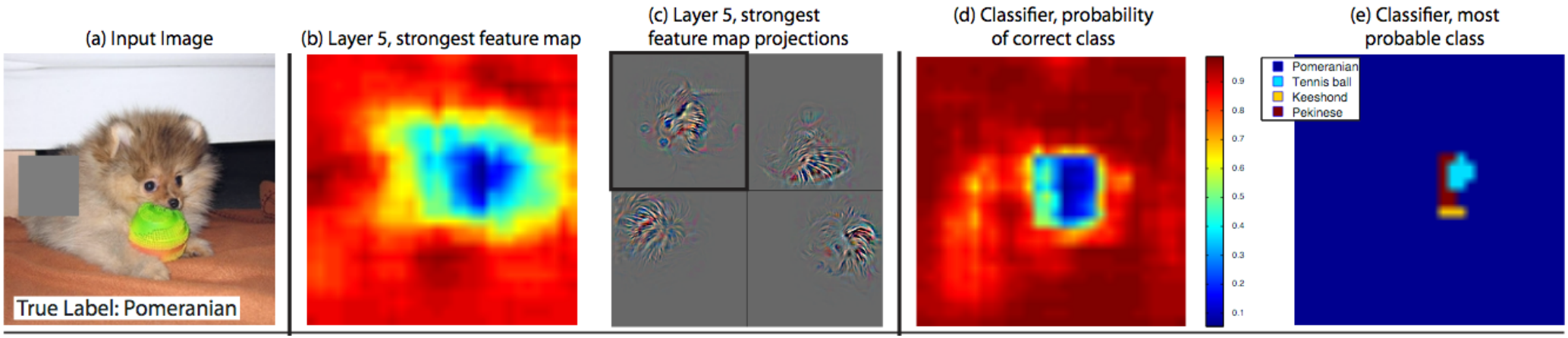
OCCLUSION SENSITIVITY

THE BASIC IDEA IS TO
PERTURB / MODIFY AN INPUT
IMAGE AND SEE THE EFFECT ON
THE PREDICTIONS

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>



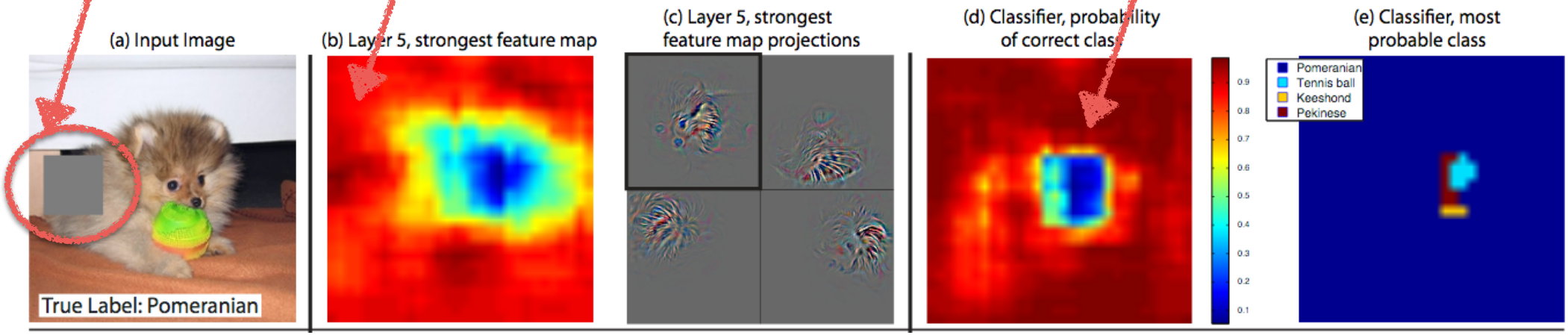
Zeiler+14

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

occluded region

the output probability as a
function of the occluding square
position



Zeiler+14

“INCEPTIONISM” TECHINQUES

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

Score of class c for image I

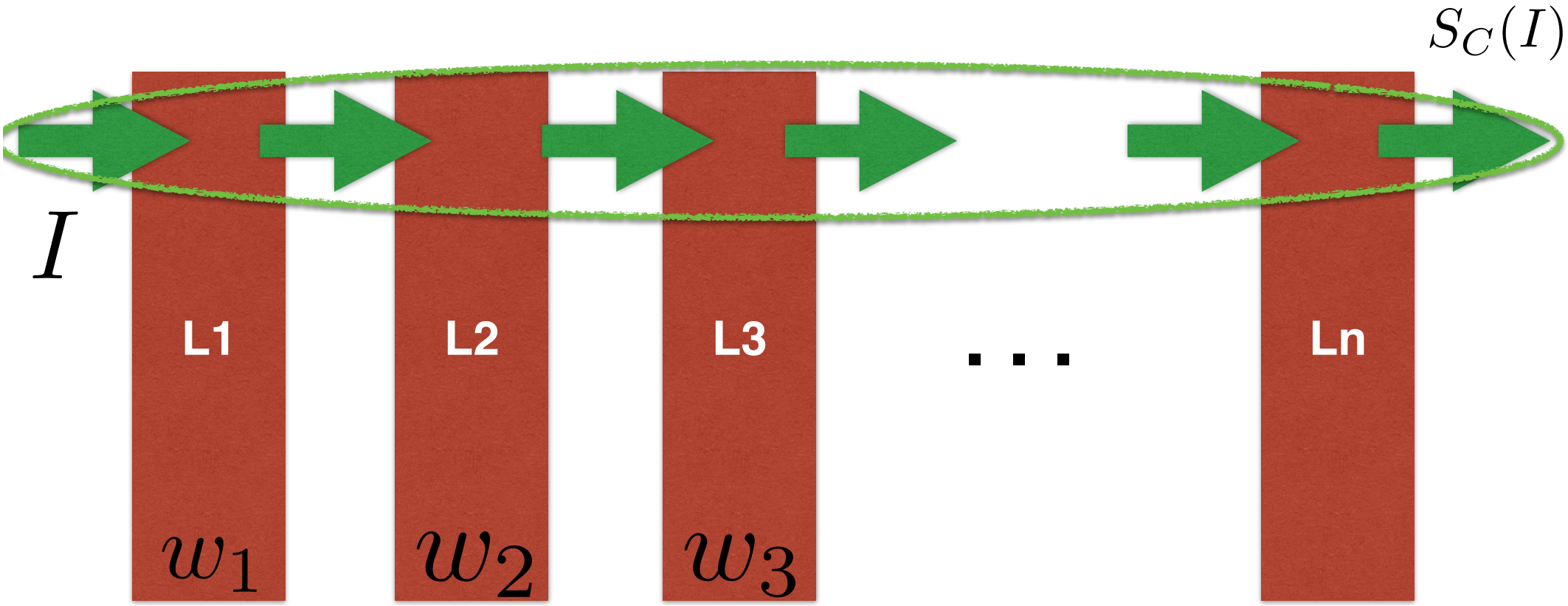
image I

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

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

Simonyan+14

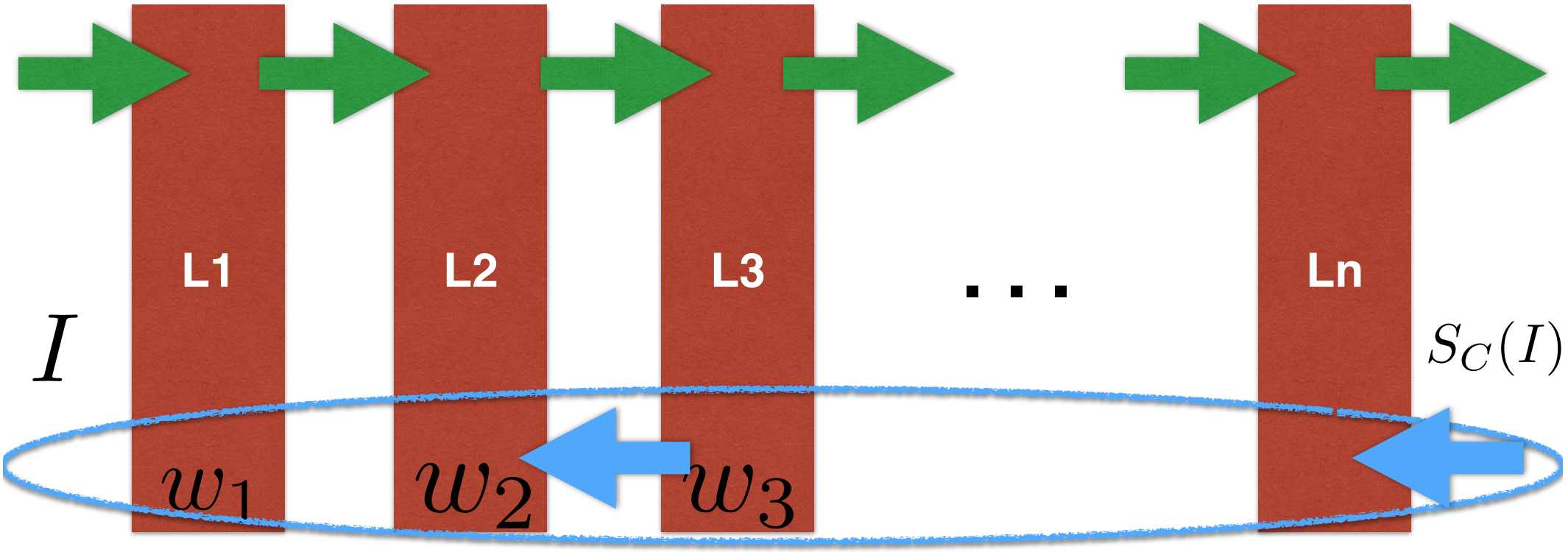
INCEPTIONISM - DEEP DREAM



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

SEE SLIDES ON ACTIVATION ATLAS FOR AN EXAMPLE

GRADIENT BASED ATTRIBUTION

“what pixels in the image are responsible of this classificaiton?”

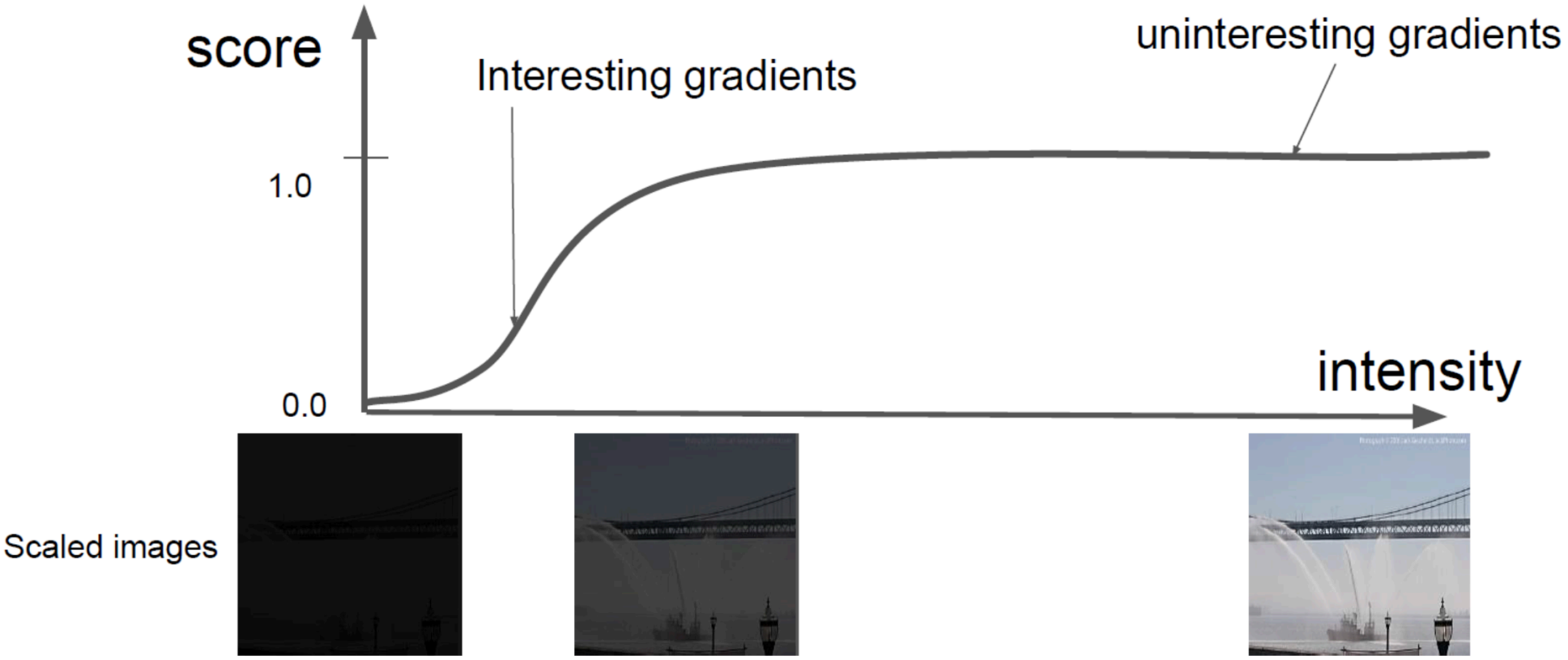
Attribute using gradient of the output w.r.t each input feature

$$x_i \frac{\delta F_w(x)}{\delta x_i}$$

We try to find the importance of each feature (pixel) by computing the gradient w.r.t to that feature (Taylor approximation of the network function)

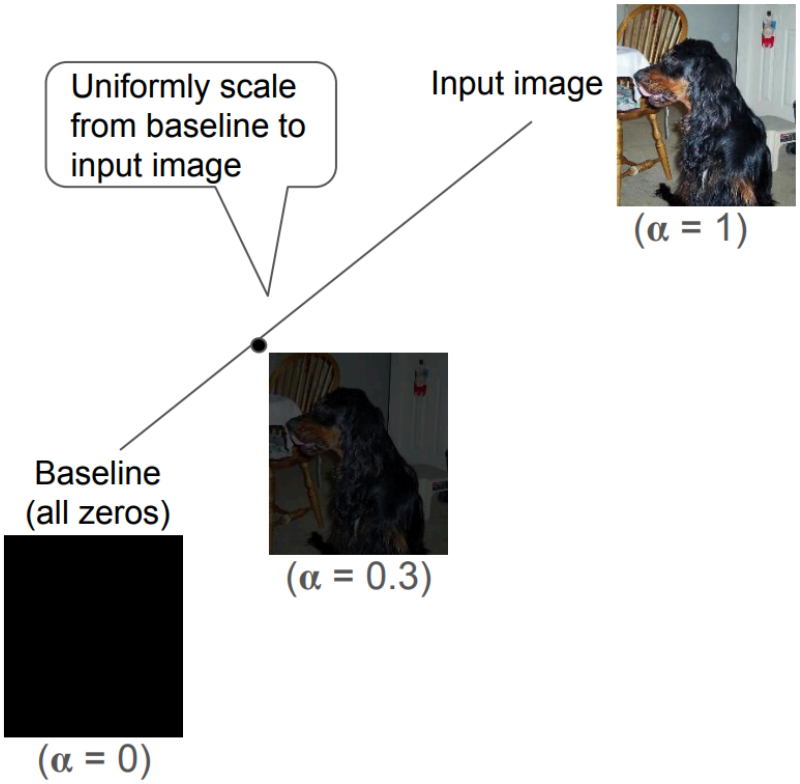
Does not work super well…

http://theory.stanford.edu/~ataly/Talks/sri_attribution_talk_jun_2017.pdf



https://medium.com/@kartikeyabhardwaj98/integrated-gradients-for-deep-neural-networks-c114e3968eae

INTEGRATED GRADIENTS



Construct a sequence of images interpolating from a baseline (black) to the actual image

Average the gradients across these images

Sundararajan+17

INTEGRATED GRADIENTS

Original image (**Drilling platform**)

Gradient at image

Integrated gradient

Sundararajan+17

INTEGRATED GRADIENTS

Human label: accordion
Network's top label: toaster

Integrated gradient

Sundararajan+17

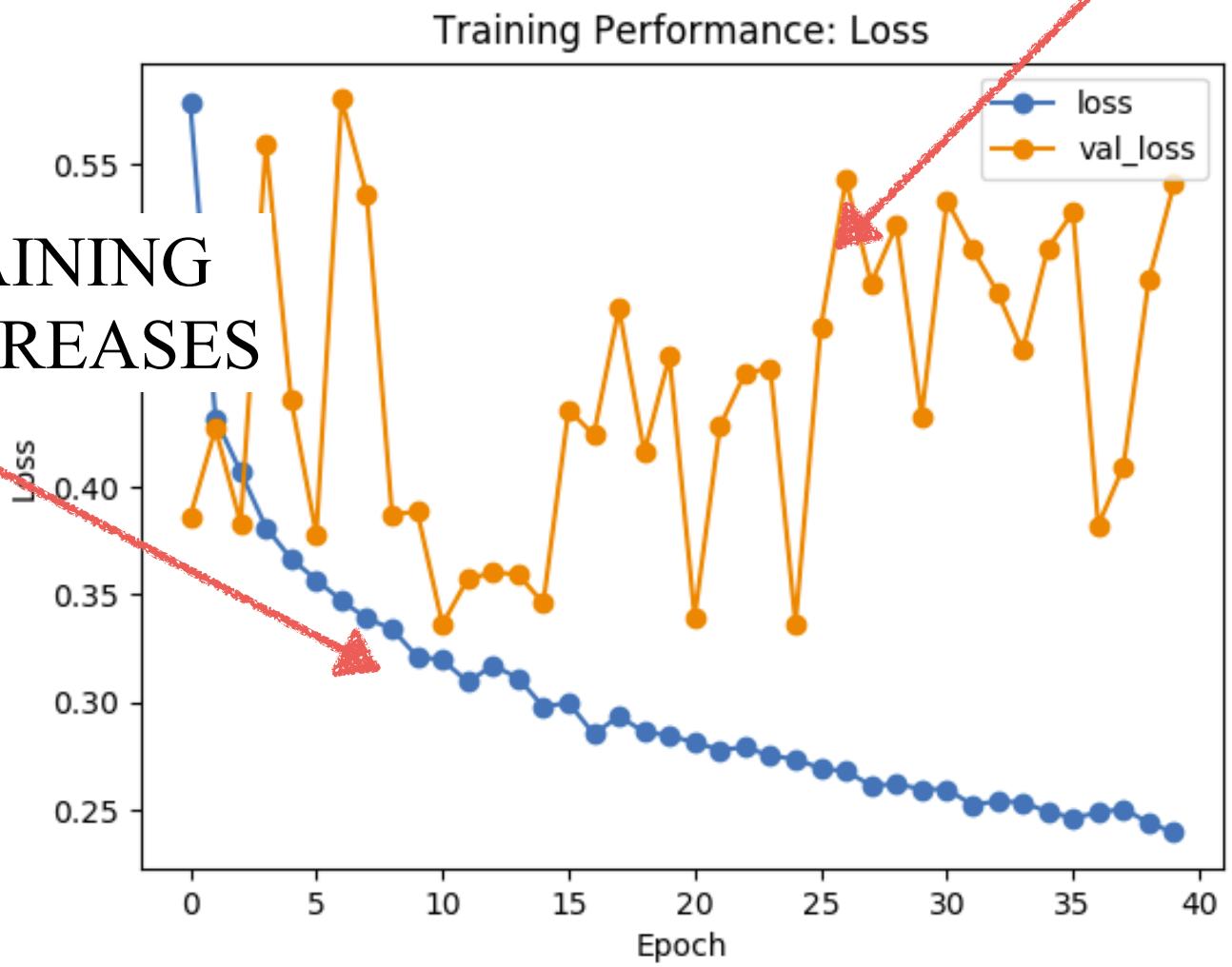
THE PROBLEMS OF GOING “TOO DEEP”

- 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

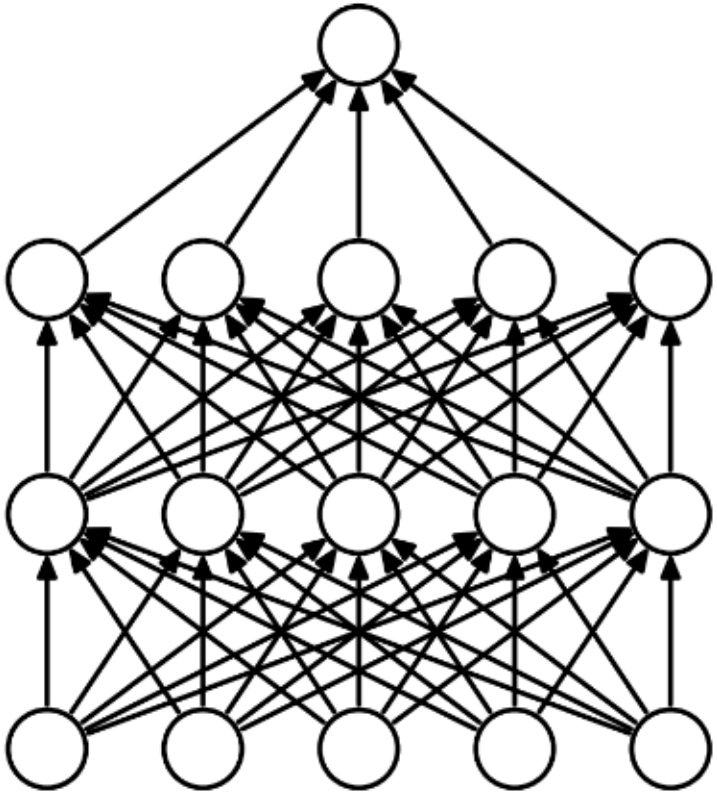


DROPOUT

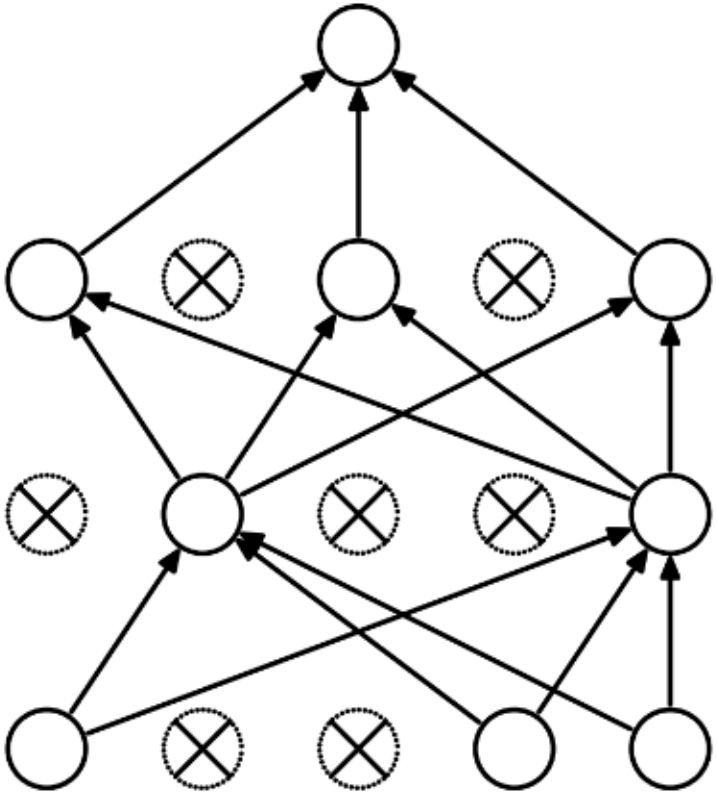
[Hinton+12]

- 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

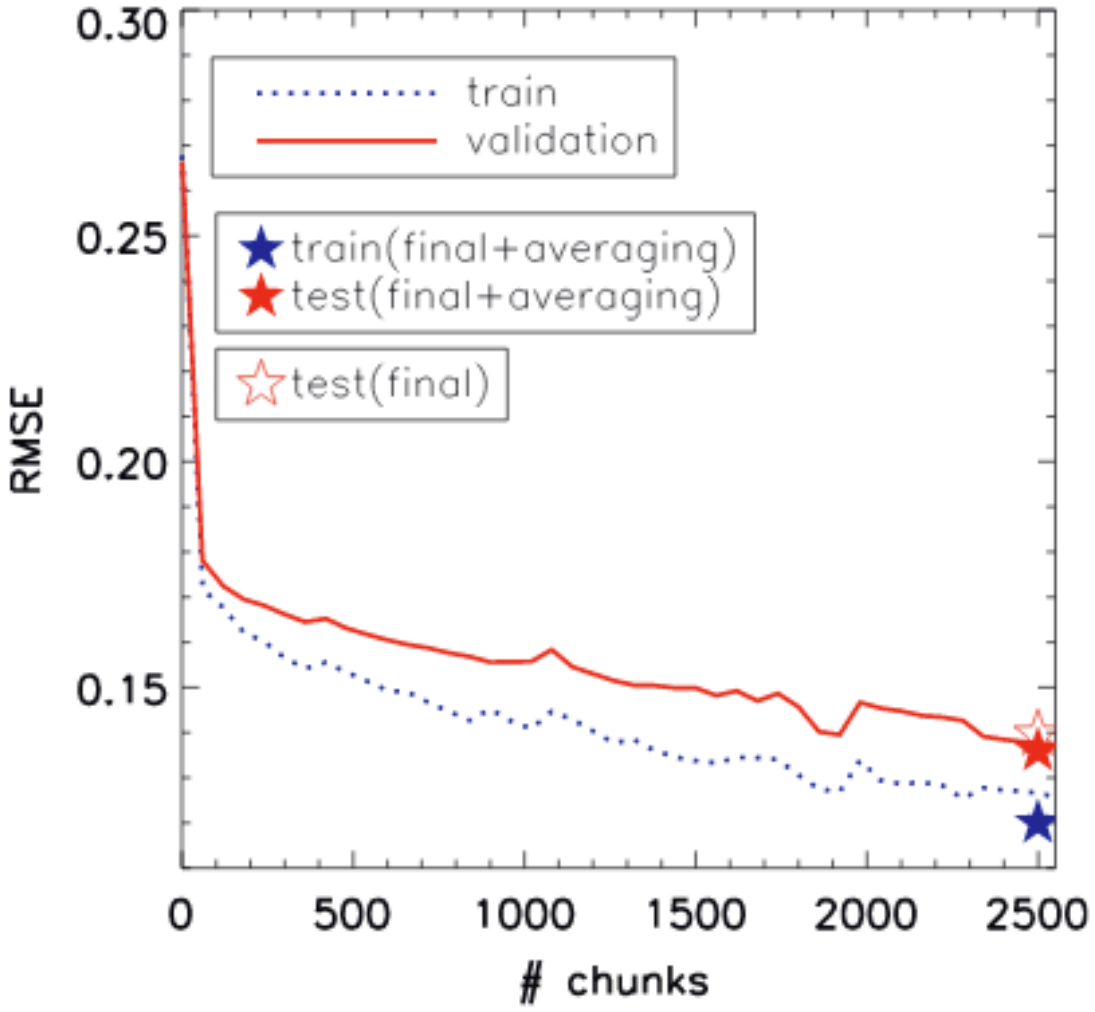
WHY DOES IT WORK?

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.

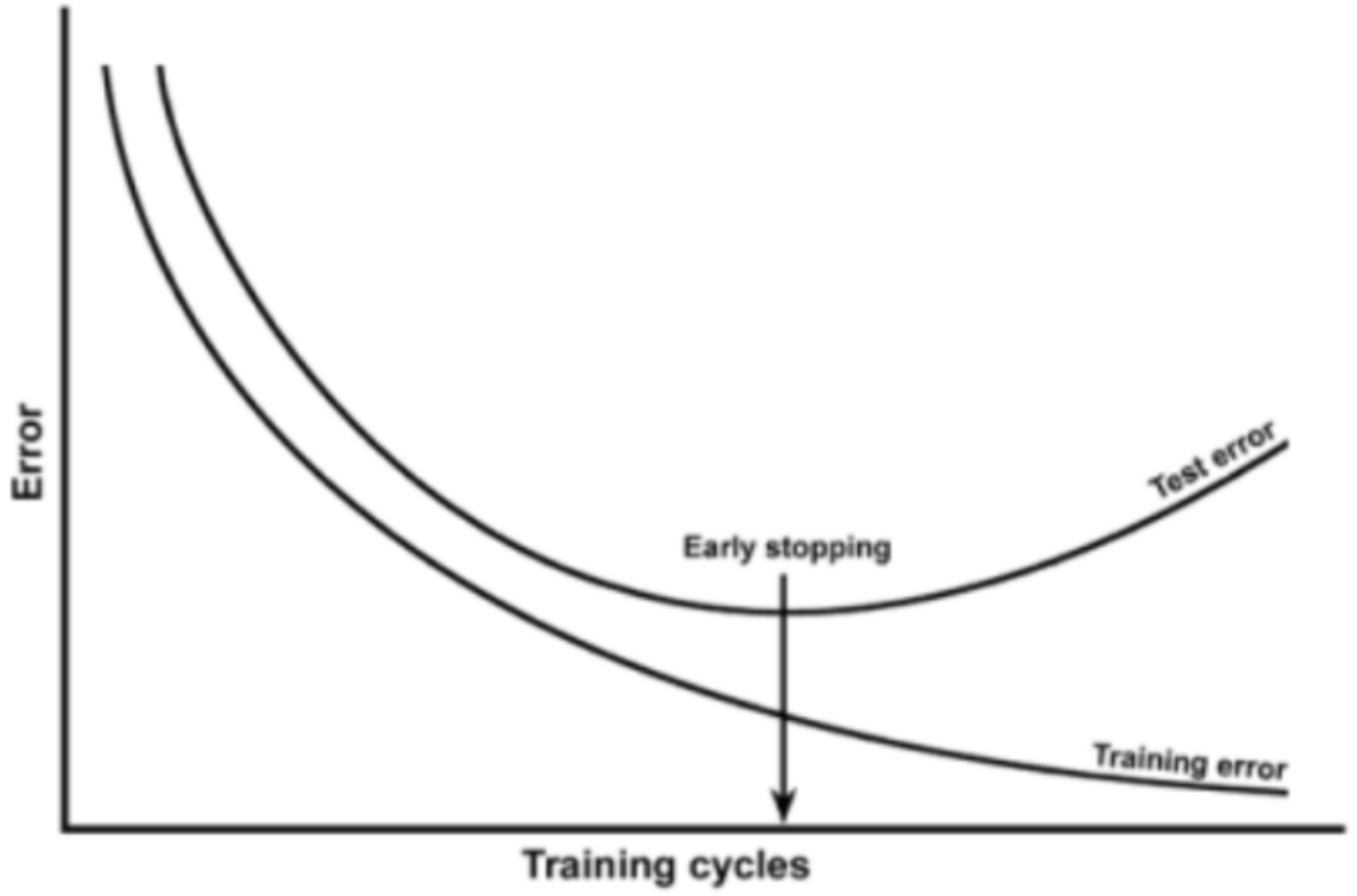
DROPOUT

WITH A LITTLE BIT
OF DROPOUT



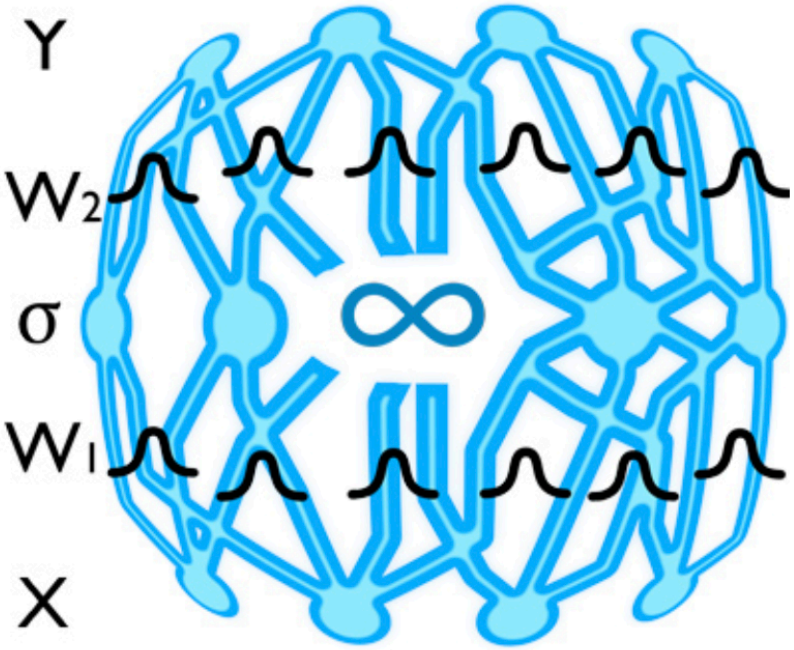
Huertas-Company+15

EARLY STOPPING HELPS ALSO PREVENTING OVERFITTING…

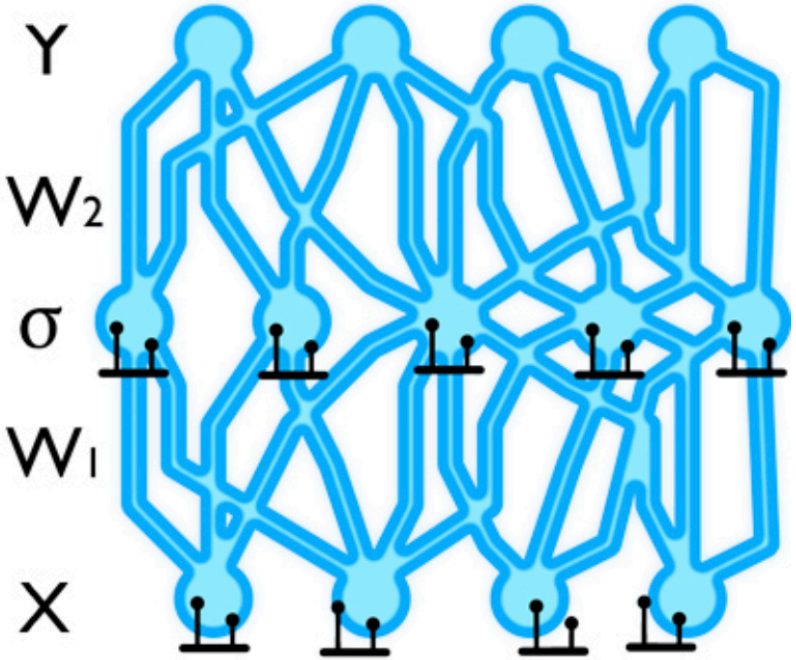


DROPOUT AS EPISTEMIC UNCERTAINTY ESTIMATION

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

DROPOUT AS EPISTEMIC UNCERTAINTY ESTIMATION

We can treat the many different networks (with different neurons dropped out) as Monte Carlo samples from the space of all available models. This provides mathematical grounds to reason about the model's uncertainty and, as it turns out, often improves its performance.

We simply apply dropout at test time, that's all! Then, instead of one prediction, we get many, one by each model. We can then average them or analyze their distributions.

DATA AUGMENTATION

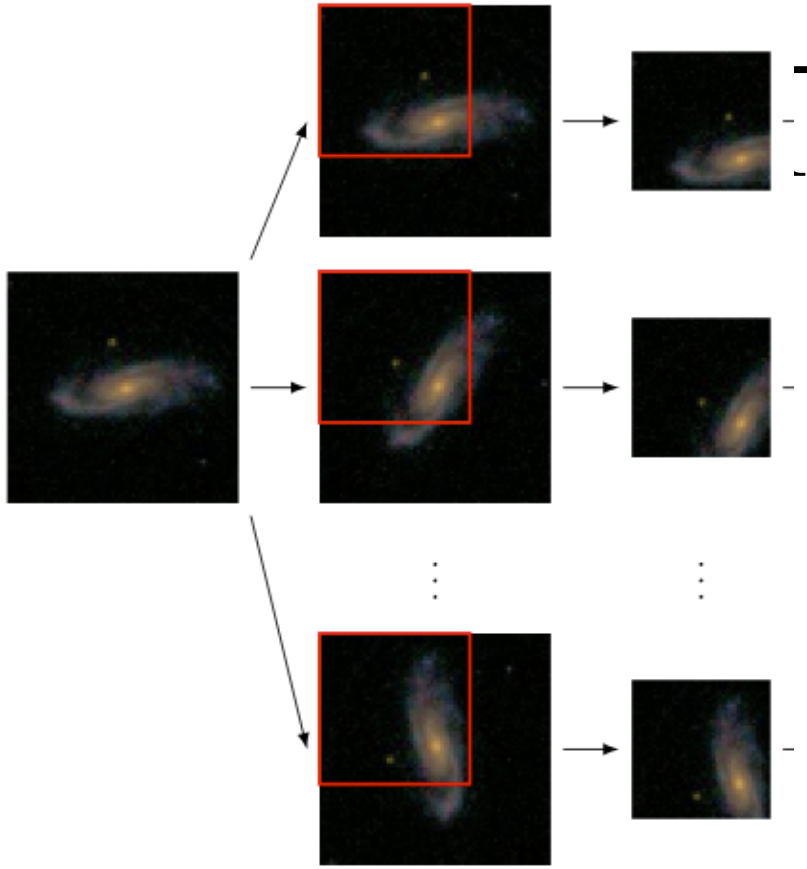
ANOTHER WAY TO REDUCE OVER-FITTING IS TO "AUGMENT" THE SIZE OF THE DATASET AVAILABLE FOR TRAINING

FOR MANY APPLICATIONS THE CLASSIFICATION SHOULD BE INDEPENDENT TO:

- TRANSALTIONS
- ROTATIONS
- SCALINGS

ETC…

DATA [illegible]TATION



Dieleman+15

FOR MANY APPLICATIONS THE CLASSIFICATION SHOULD BE INDEPENDENT TO:

- TRANSALTIONS
- ROTATIONS
- SCALINGS

ETC…

DATA AUGMENTATION

FOR MANY APPLICATIONS THE CLASSIFICATION SHOULD BE INDEPENDENT TO:

- TRANSALTIONS
- ROTATIONS
- SCALINGS

ETC…

Transfer learning

THE CONVOLUTIONAL PART OF A CNN IS
A FEATURE EXTRACTOR ….

Transfer learning

THE CONVOLUTIONAL PART OF A CNN IS
A FEATURE EXTRACTOR

IN THAT RESPECT, THEY ARE VERY FLEXIBLE …

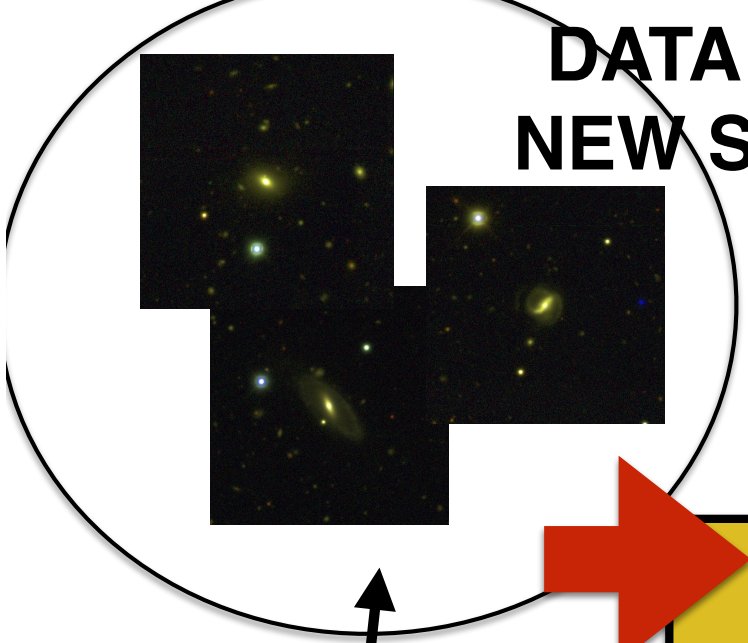
Transfer Learning)

EVEN IF OUR TRAINING SET IS NOT SO LARGE …

WE CAN USE A CNN PRE-TRAINED ON A LARGER SAMPLE

DEPENDING ON HOW SIMILAR BOTH DATASETS ARE, WE CAN:

- RECYCLE THE SAME FEATURES
- FINE-TUNING THE WEIGHTS



DATA FROM NEW SURVEY

How robust to different datasets? Do we always need a big training set?

DEEP-LEARNING BASED MACHINE

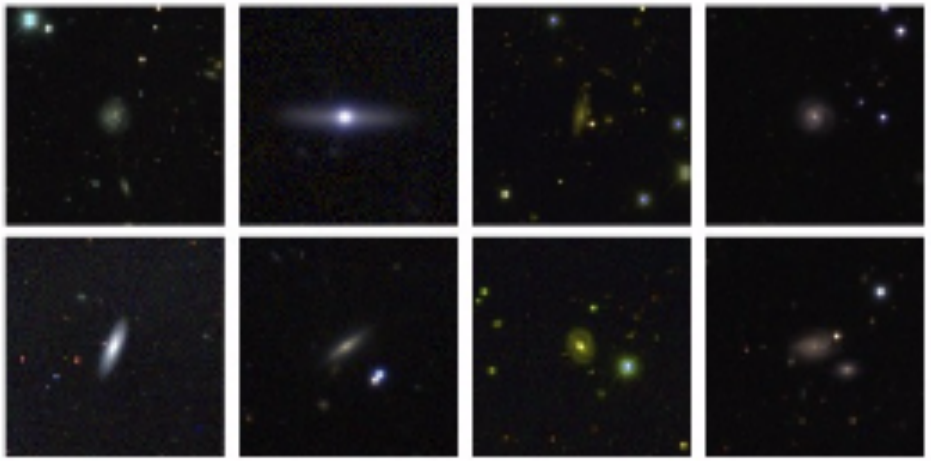
"Improved" Galaxy ZOO like classifications for for the entire sample

Transfer knowledge?

Human classifications from existing survey

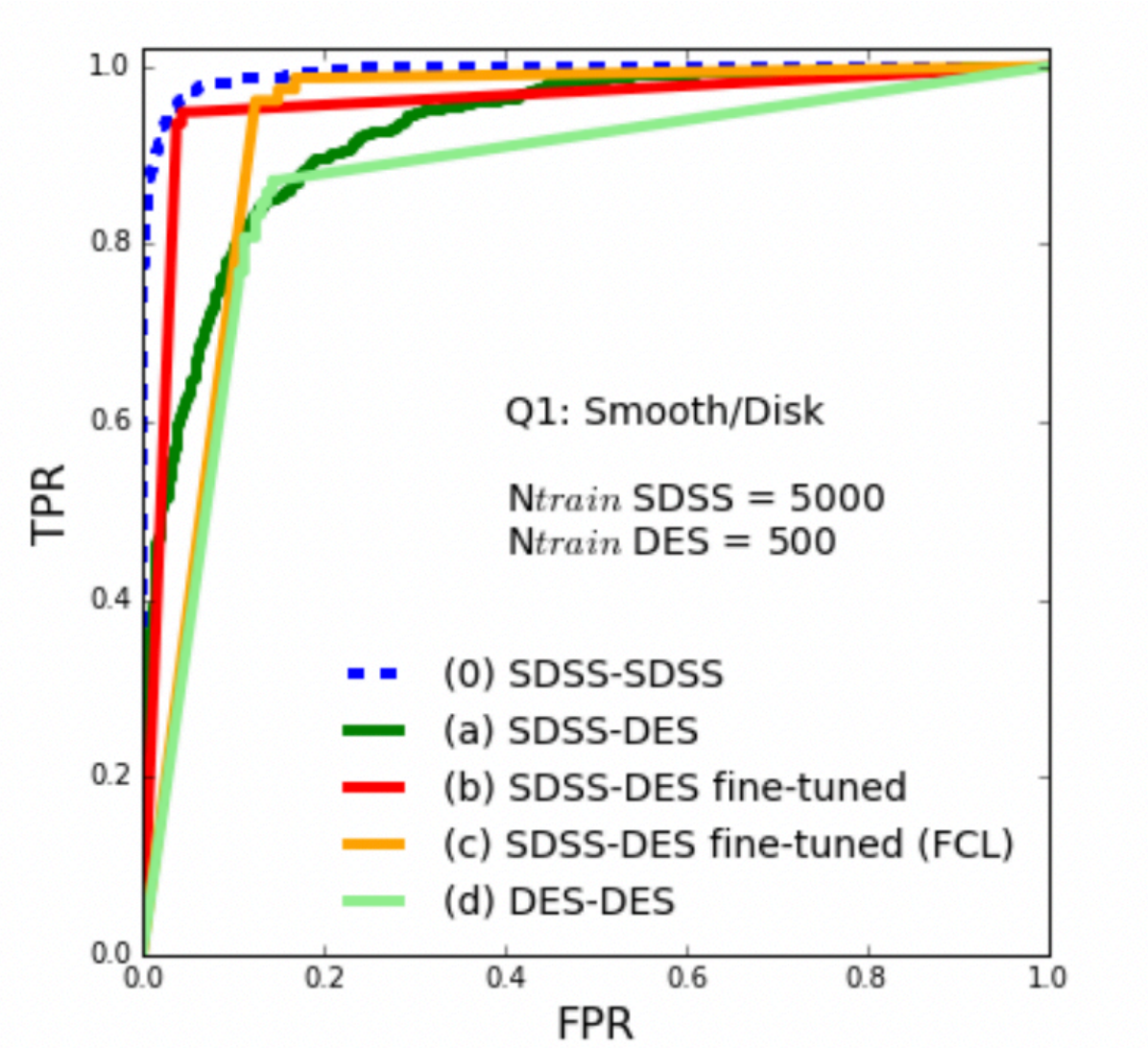
Dominguez-Sanchez+18b

SDSS



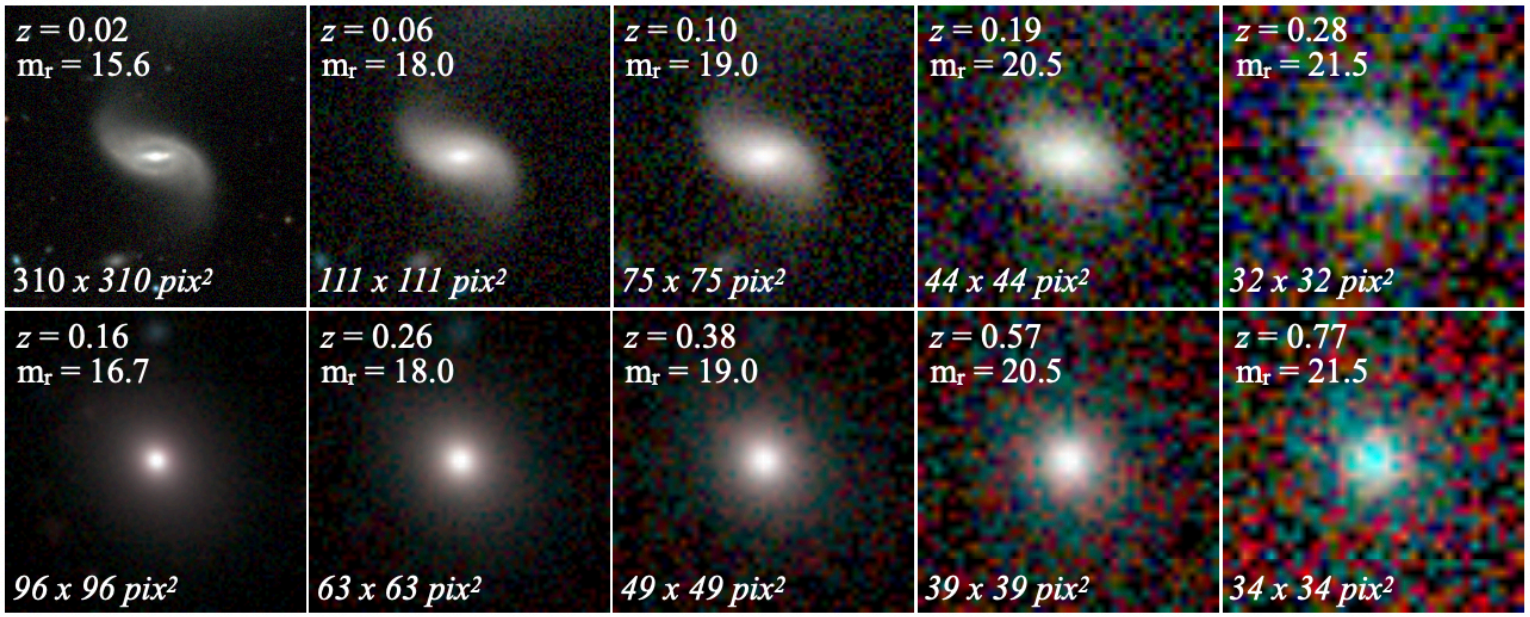
DES

Dominguez-Sanchez+18b



Dominguez-Sanchez+18b

OR YOU CAN ALSO TRAIN ON SIMULATIONS

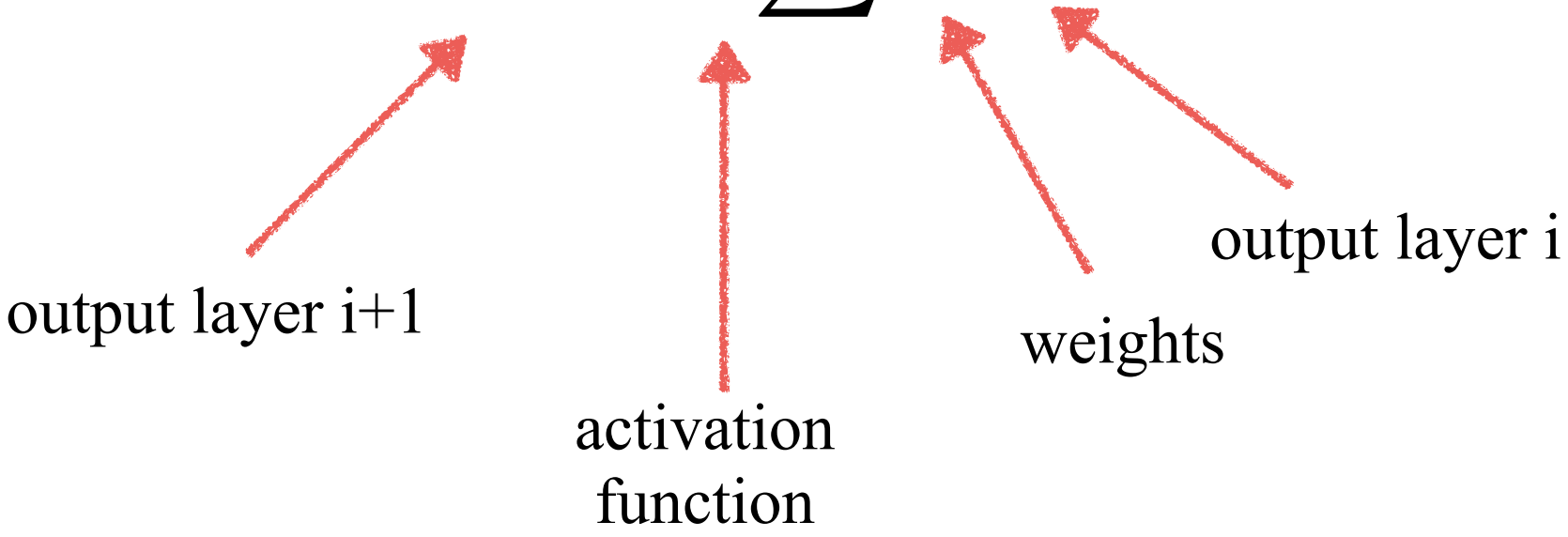


THIS IS WHERE DOMAIN KNOWLEDGE COMES INTO PLAY

VANISHING / EXPLODING GRADIENT PROBLEM

REMEMBER THAT:

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

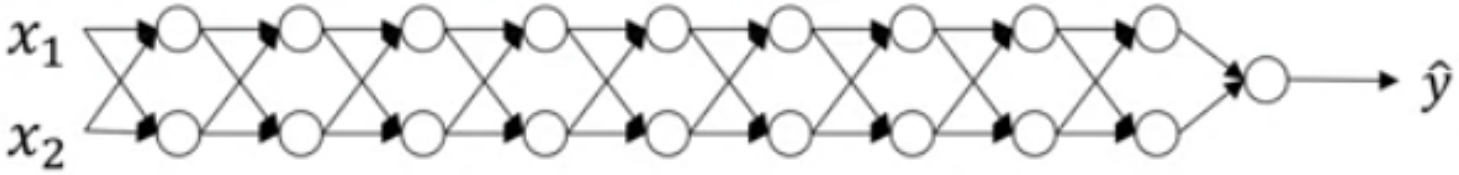


VANISHING / EXPLODING GRADIENT PROBLEM

WITH MANY LAYERS:

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

VANISHING/EXPLODING GRADIENT PROBLEM



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

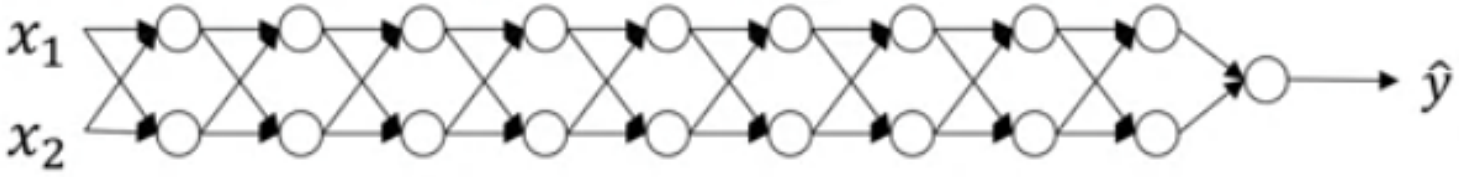
$$x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

IF WEIGHTS ARE ALL INITIALIZED TO VALUES <<1:

$$\hat{y} \to 0$$

VANISHING GRADIENT

VANISHING/EXPLODING GRADIENT PROBLEM



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

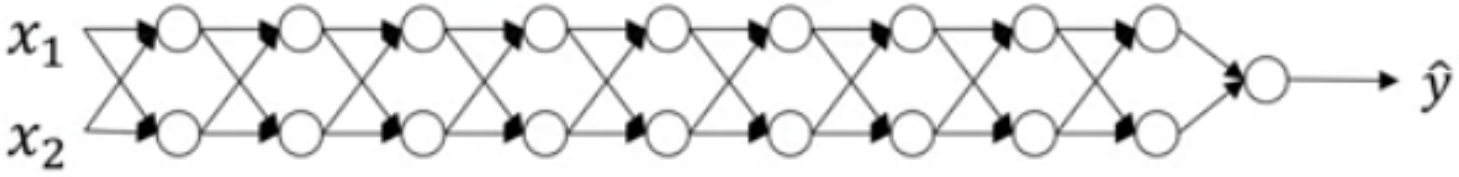
$$x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

IF WEIGHTS ARE ALL INITIALIZED TO VALUES >1:

$$\hat{y} \to \infty$$

EXPLODING GRADIENT

VANISHING/EXPLODING GRADIENT PROBLEM



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

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

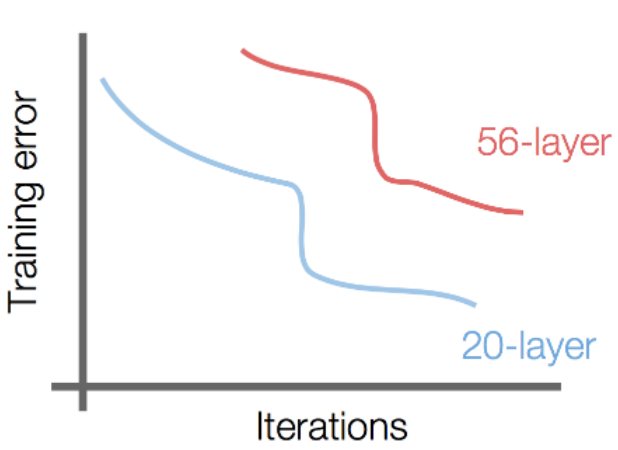
IF WEIGHTS ARE ALL INITIALIZED TO VALUES >1:

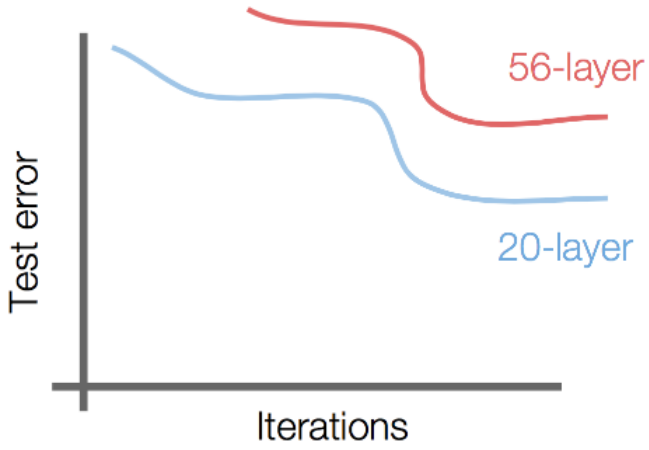
$$w_i^L \to \infty$$

EXPLODING GRADIENT

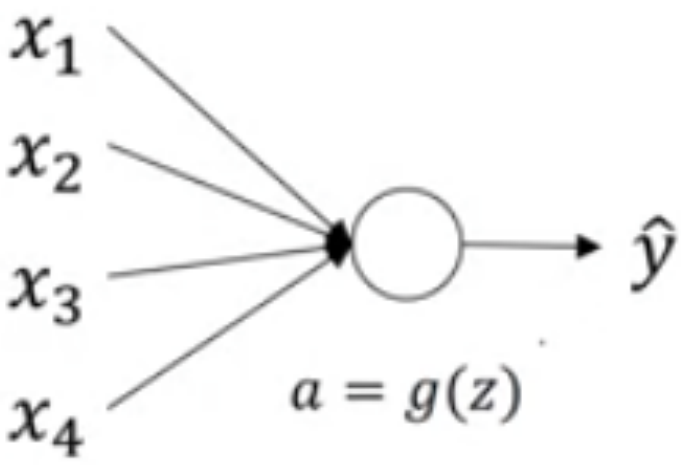
VANISHING/EXPLODING GRADIENT PROBLEM

TRAINING BECOMES UNSTABLE
VERY SLOW OR NO CONVERGENCE



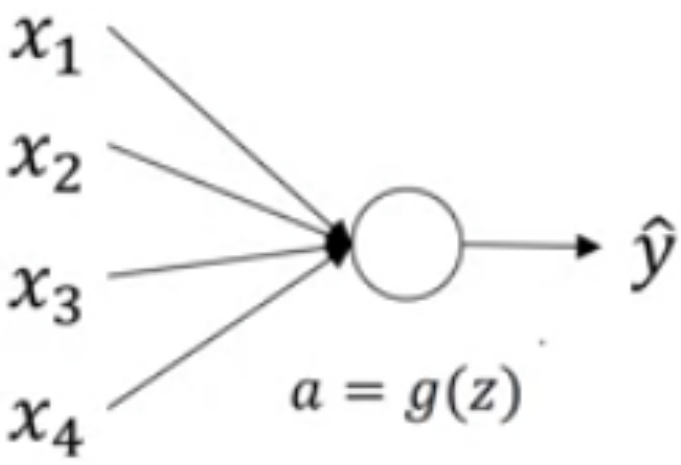


WEIGHT INITIALIZATION IS A KEY POINT…



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

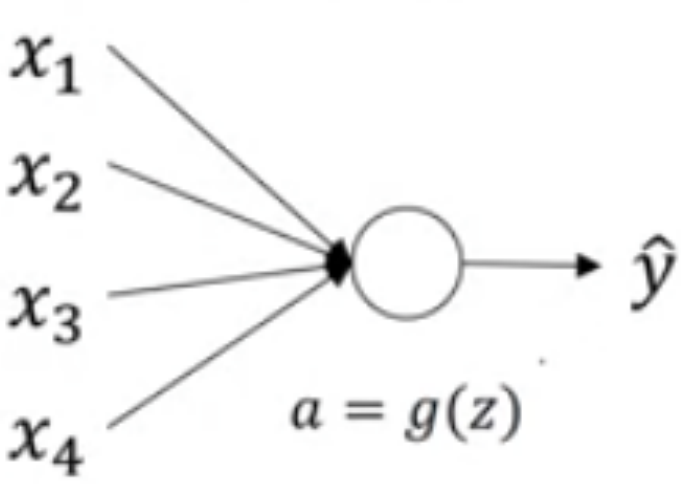
WEIGHT INITIALIZATION IS A KEY POINT…



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

THE LARGER n, THE SMALLER WEIGHTS SHOULD BE…

WEIGHT INITIALIZATION IS A KEY POINT…



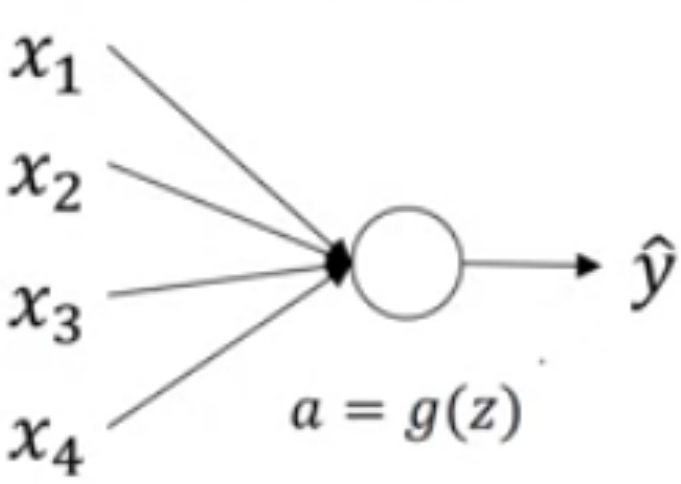
$$z = x_1 w_1 + x_2 w_2 + \ldots + x_n w_n$$

THE LARGER n, THE SMALLER WEIGHTS SHOULD BE…

variance

ONE SIMPLE SOLUTION: $\sigma^2(w_i) = \frac{1}{n}$ ← 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

ONE SIMPLE SOLUTION: $\sigma^2(w_i) = \frac{1}{n}$ ← number of inputs

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:

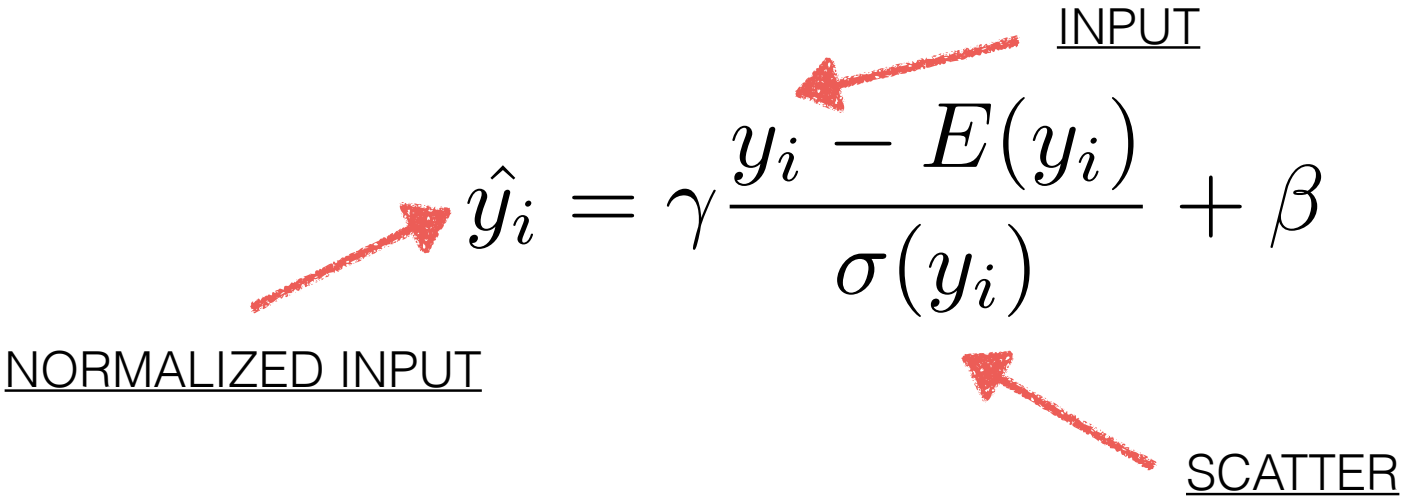
`keras.initializers` → https://keras.io/initializers/

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



BATCH NORMALIZATION

[SZEGEDY+15]

BATCH NORMALIZATION SPEEDS UP AND STABILIZES TRAINING

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

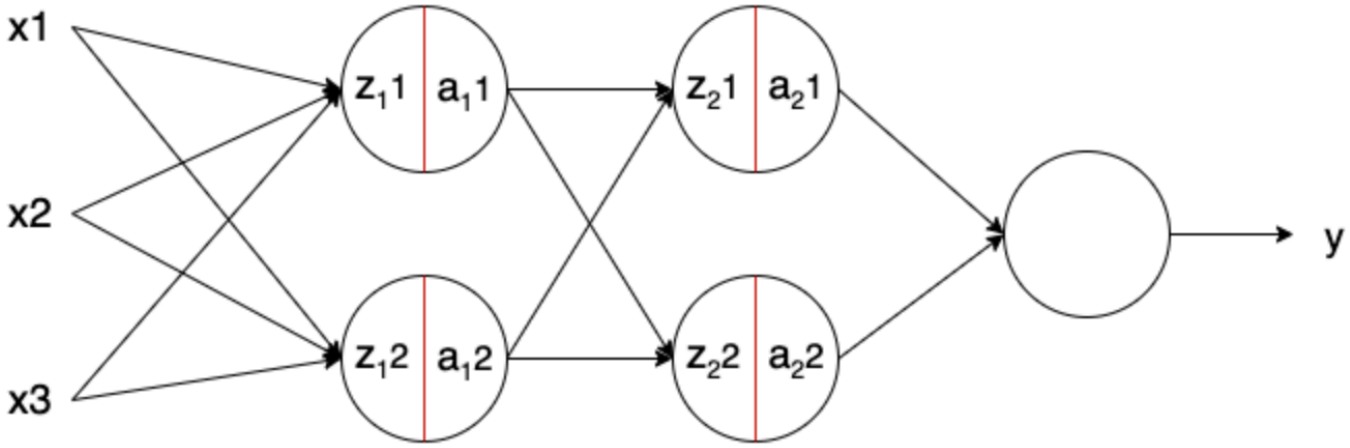
$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

BATCH NORMALIZATION

[SZEGEDY+15]



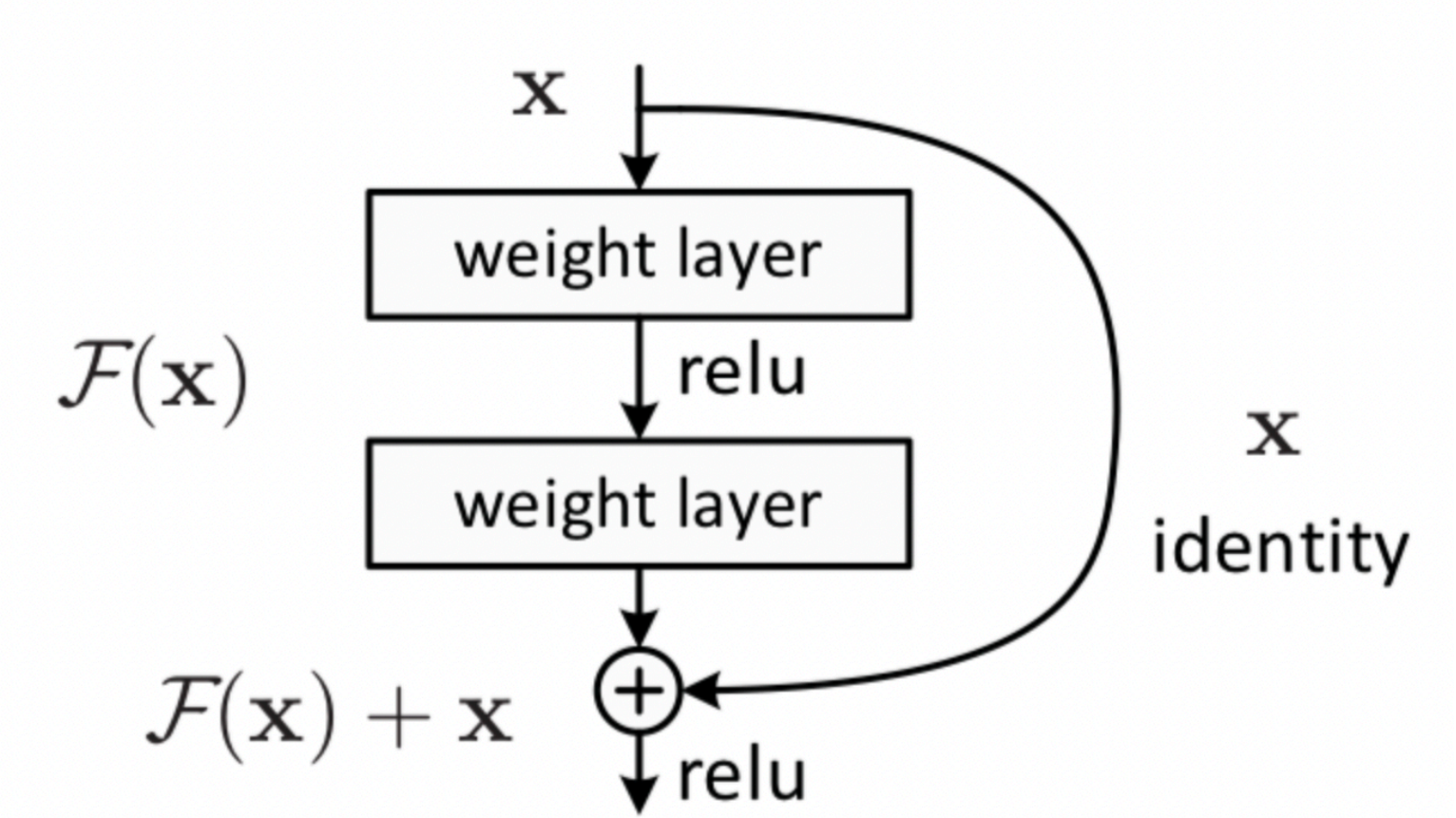
$$z = g(w, x); \qquad z^N = \left(\frac{z - m_z}{s_z}\right) \cdot \gamma + \beta; \qquad a = f(z^N)$$

BATCH NORMALIZATION

[SZEGEDY+15]

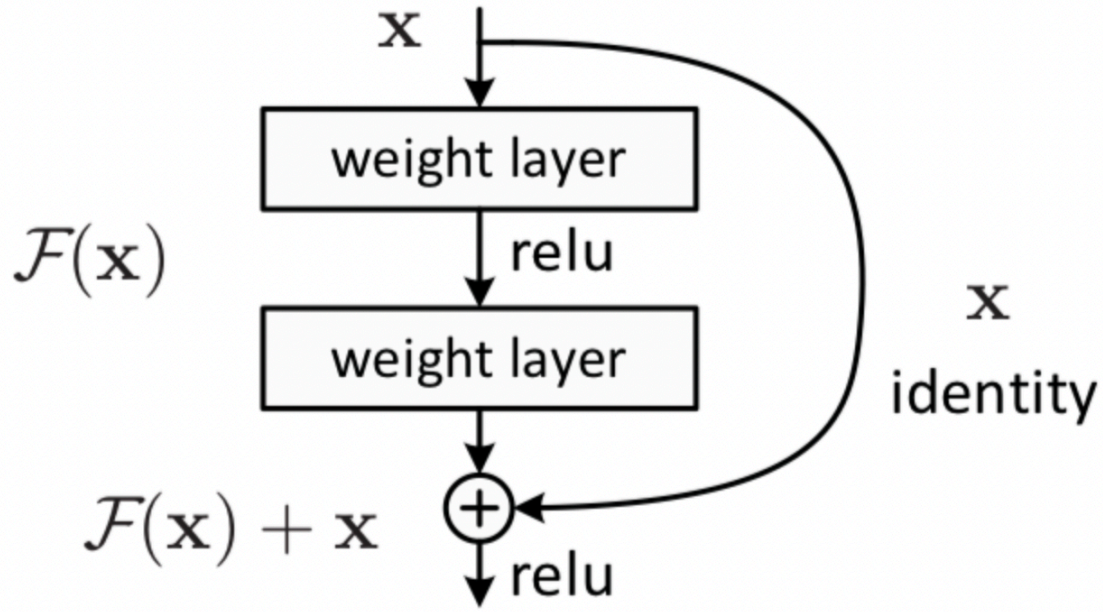
- Speeds up training
- Reduces internal covariate shift of the network
- Regularizes the network, prevents over fitting

RESIDUAL NETWORKS



Kaiming et al. 2015

RESIDUAL NETWORKS

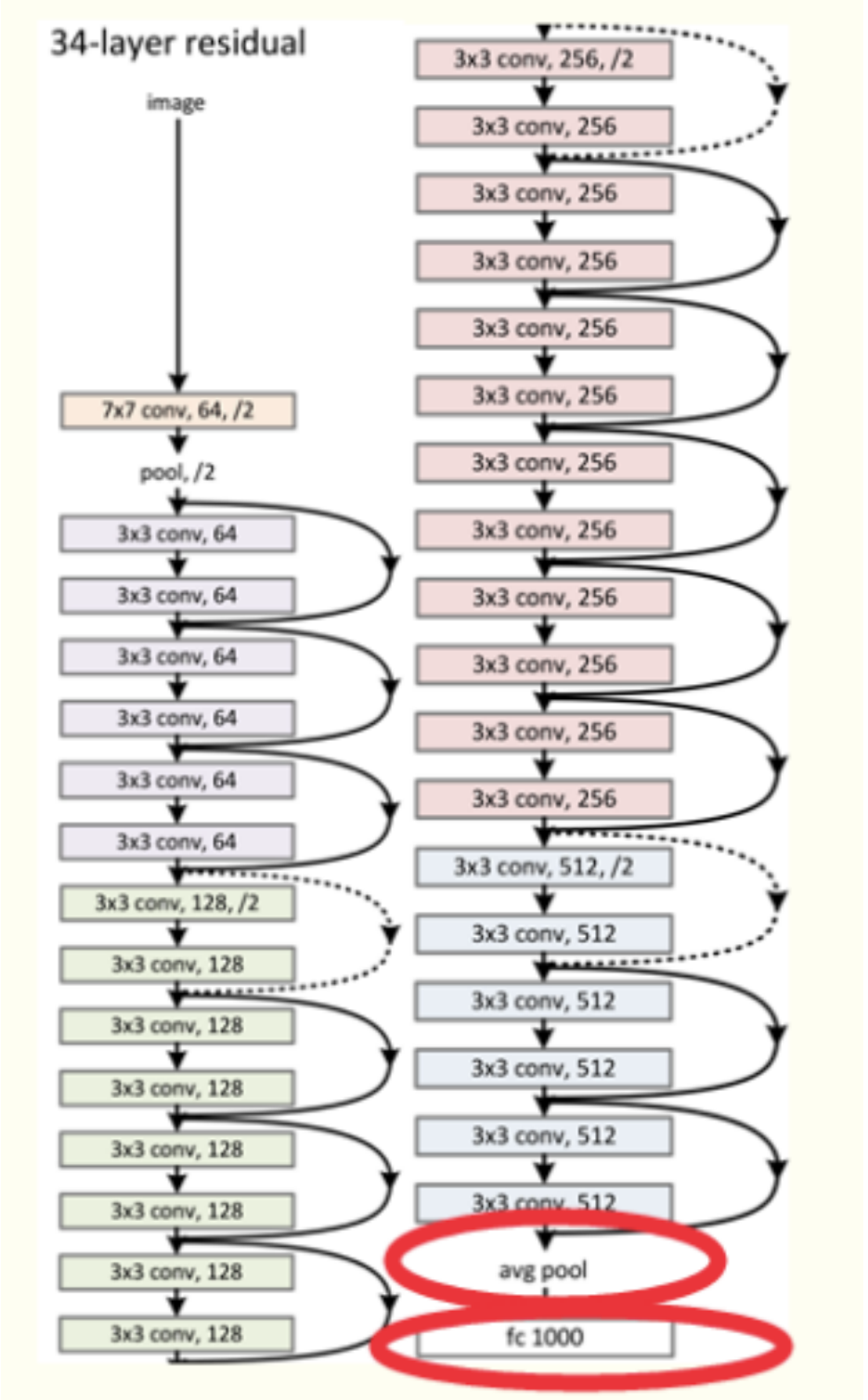


- Adding additional / new layers would not hurt the model's performance as regularisation will skip over them if those layers were not useful.

- If the additional / new layers were useful, even with the presence of regularisation, the weights or kernels of the layers will be non-zero and model performance could increase slightly.

Kaiming et al. 2015

EXAMPLE OF DEEP RESNET



DEEPER TENDS TO BE BETTER…