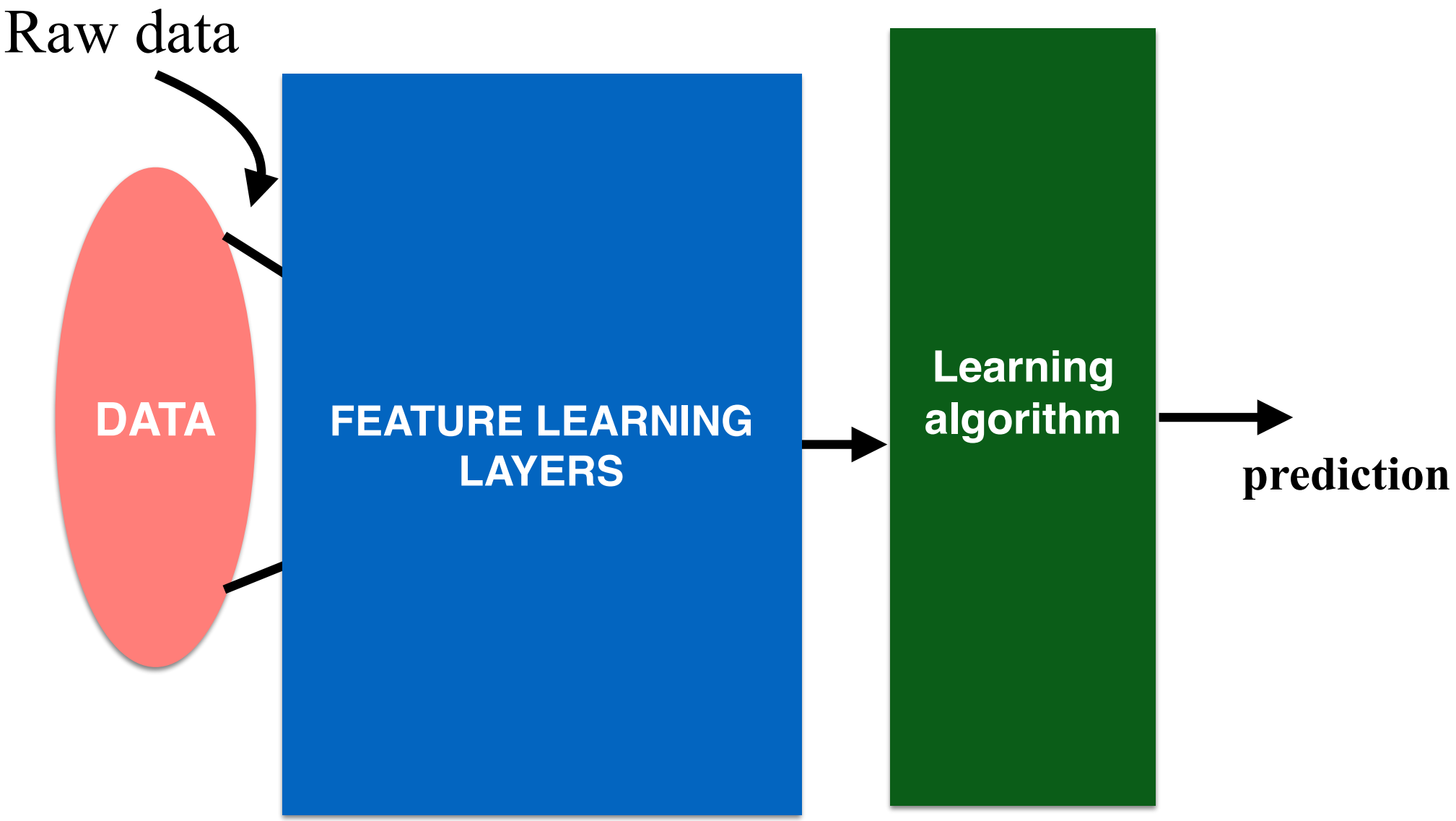
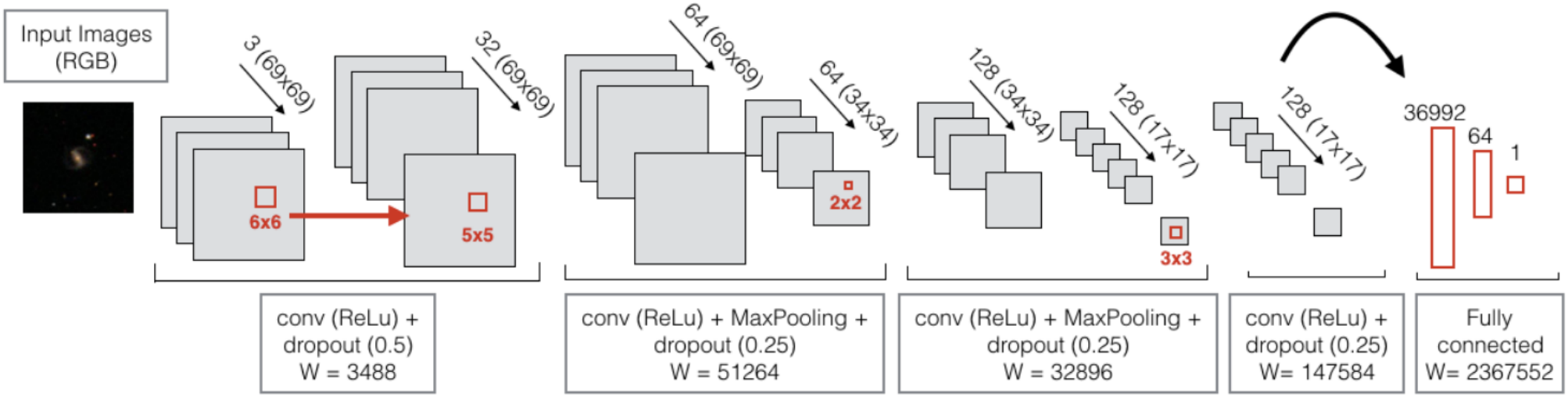

Raw data
DATA
FEATURE LEARNING LAYERS
Learning algorithm
prediction

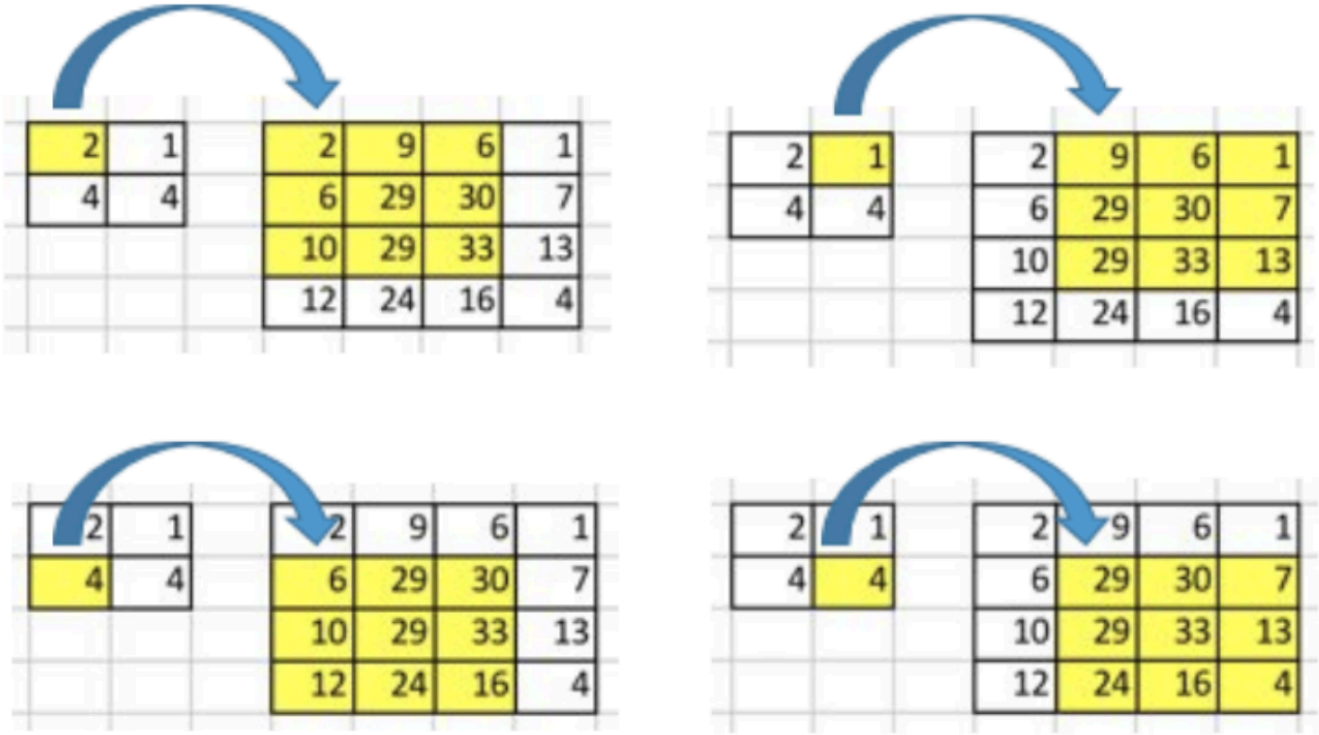
EXAMPLE OF VERY SIMPLE CNN



Dominguez-Sanchez+18

TRANSPOSED CONVOLUTION

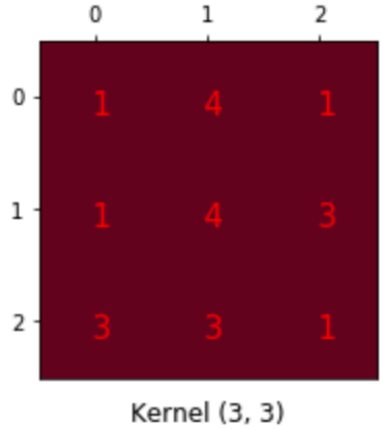
ALLOWS TO INCREASE THE SIZE



Going Backward of Convolution

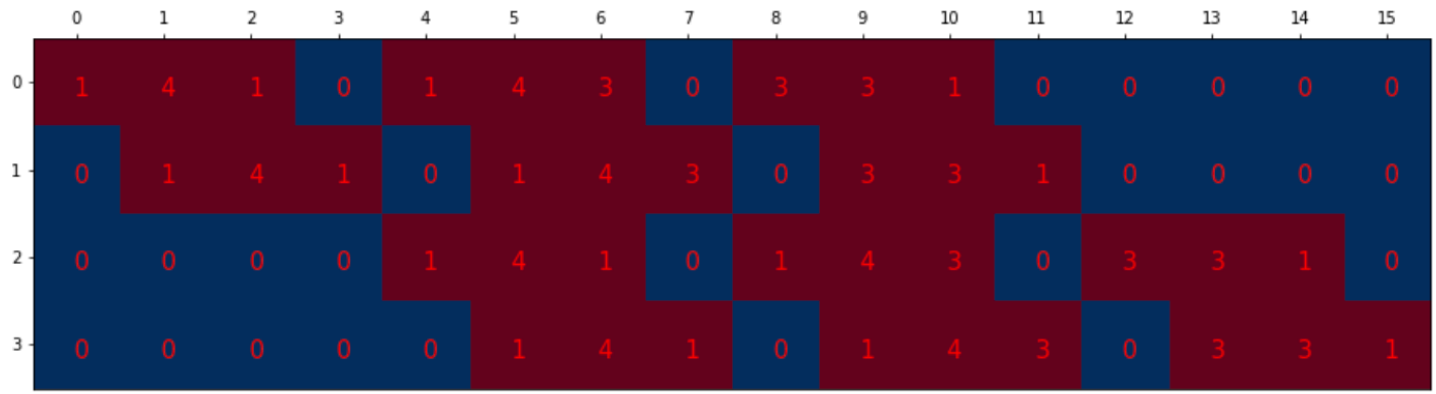
EXAMPLE TAKEN FROM HERE

CONVOLUTION MATRIX



Kernel (3, 3)

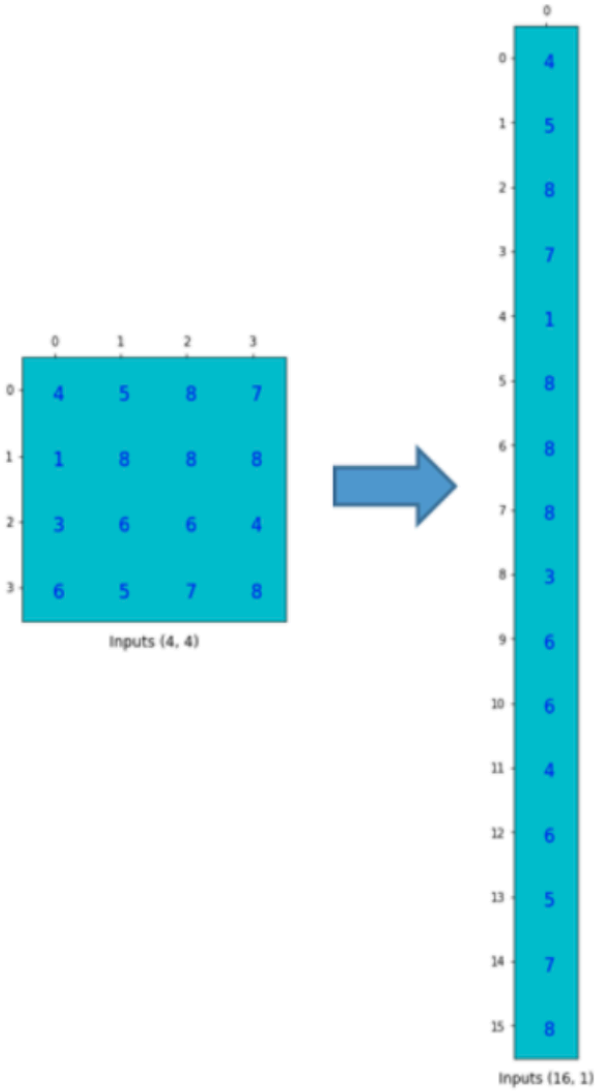
THE KERNEL CAN BE ARRANGED IN FORM OF A MATRIX:



Convolution Matrix (4, 16)

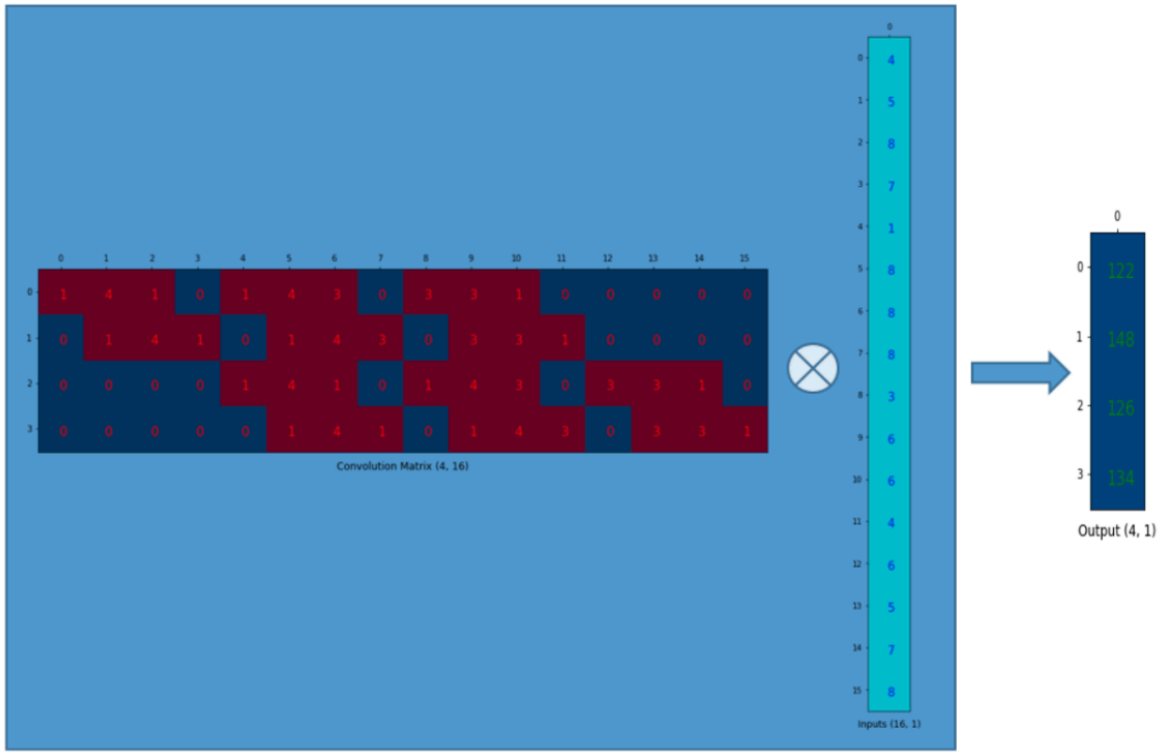
EXAMPLE TAKEN FROM HERE

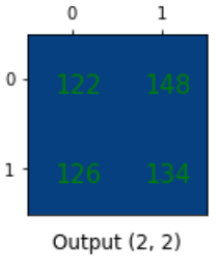
THE INPUT IS FLATTENED INTO A COLUMN VECTOR



EXAMPLE TAKEN FROM HERE

THE CONVOLUTION IS TRANSFORMED INTO A PRODUCT OF MATRICES

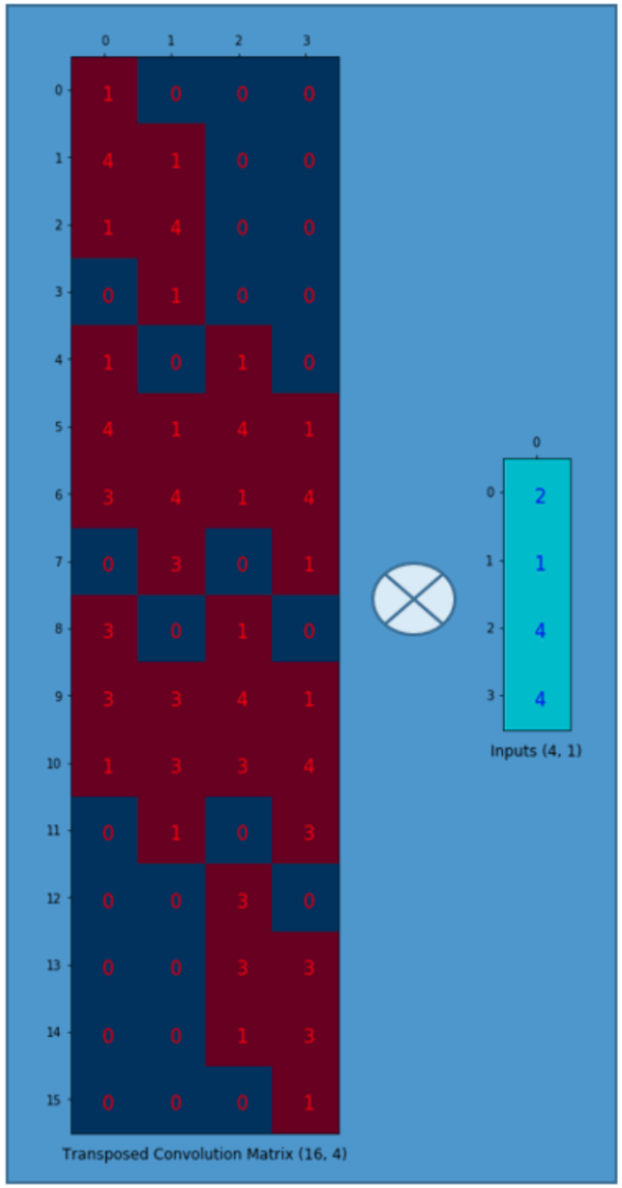


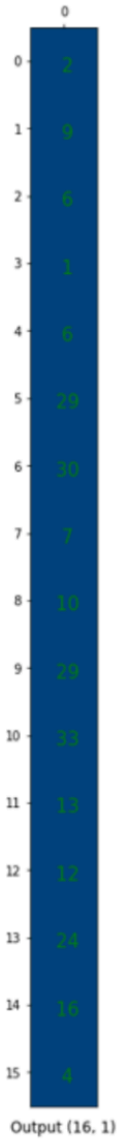


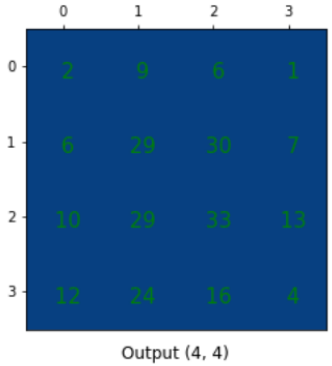
RESHAPED OUTPUT

EXAMPLE TAKEN FROM HERE

THE TRANSPOSED CONVOLUTION IS THE INVERSE OPERATION



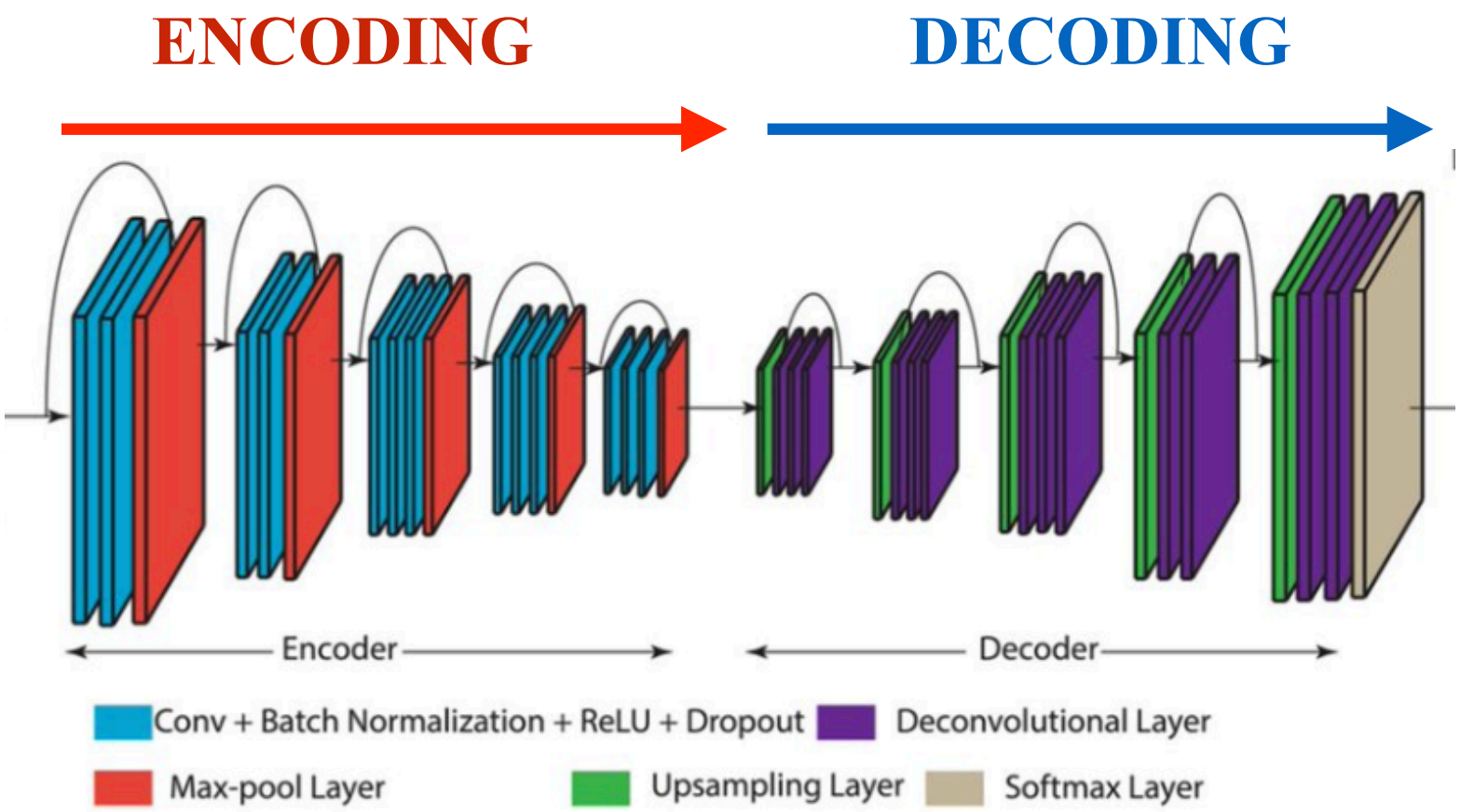




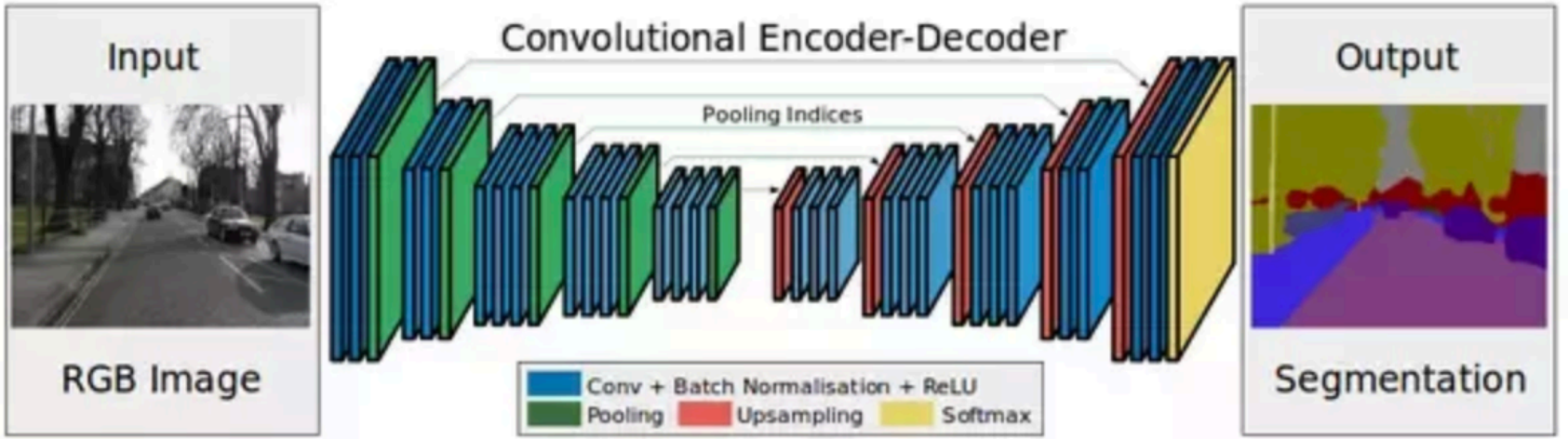
RESHAPED OUTPUT

EXAMPLE TAKEN FROM HERE

DECODER-ENCODER NETWORKS

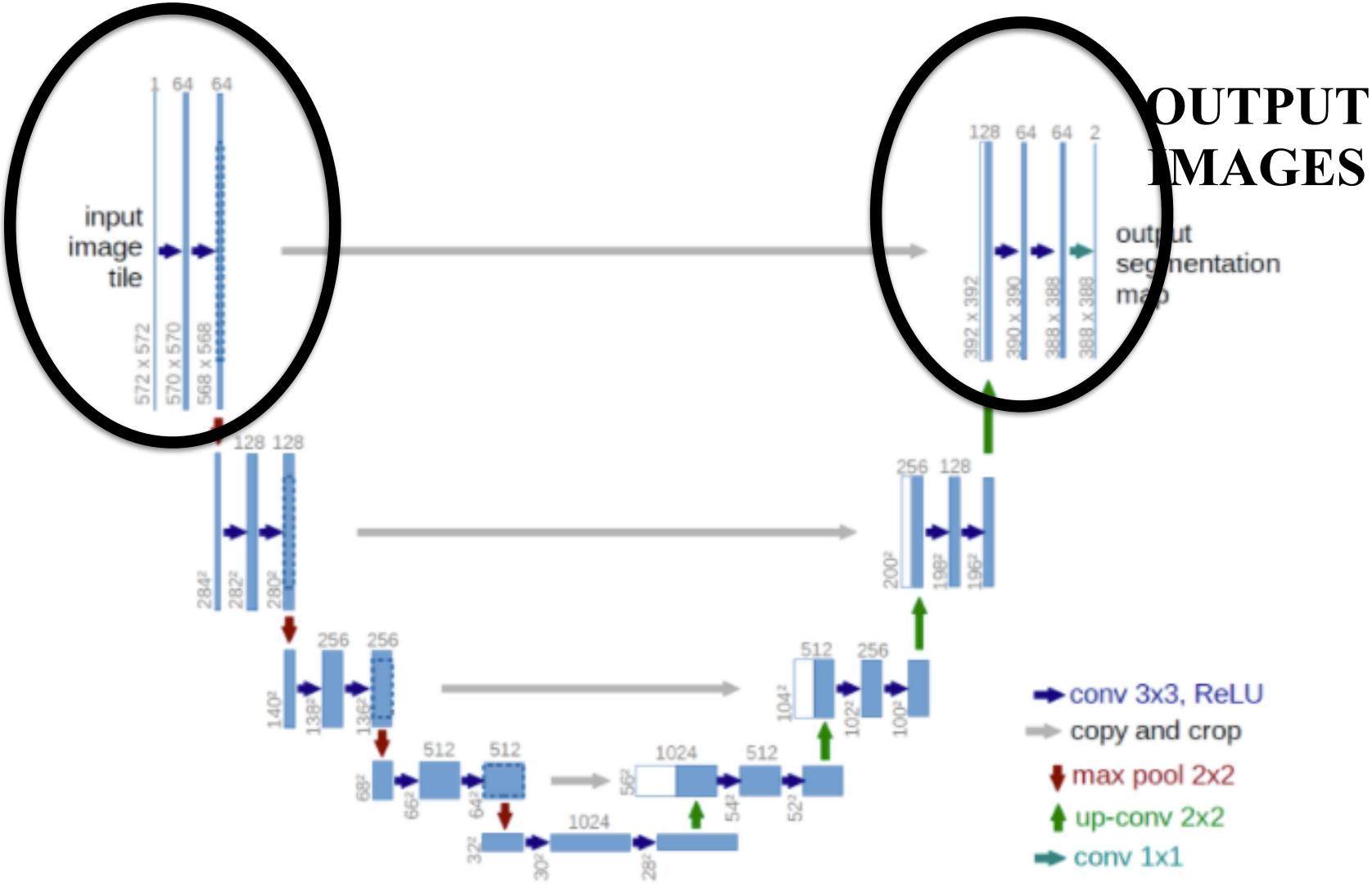


MAIN APPLICATION IS IMAGE SEGMENTATION



ENCODING-DECODING TO EXTRACT IMAGE FEATURES: U-NET

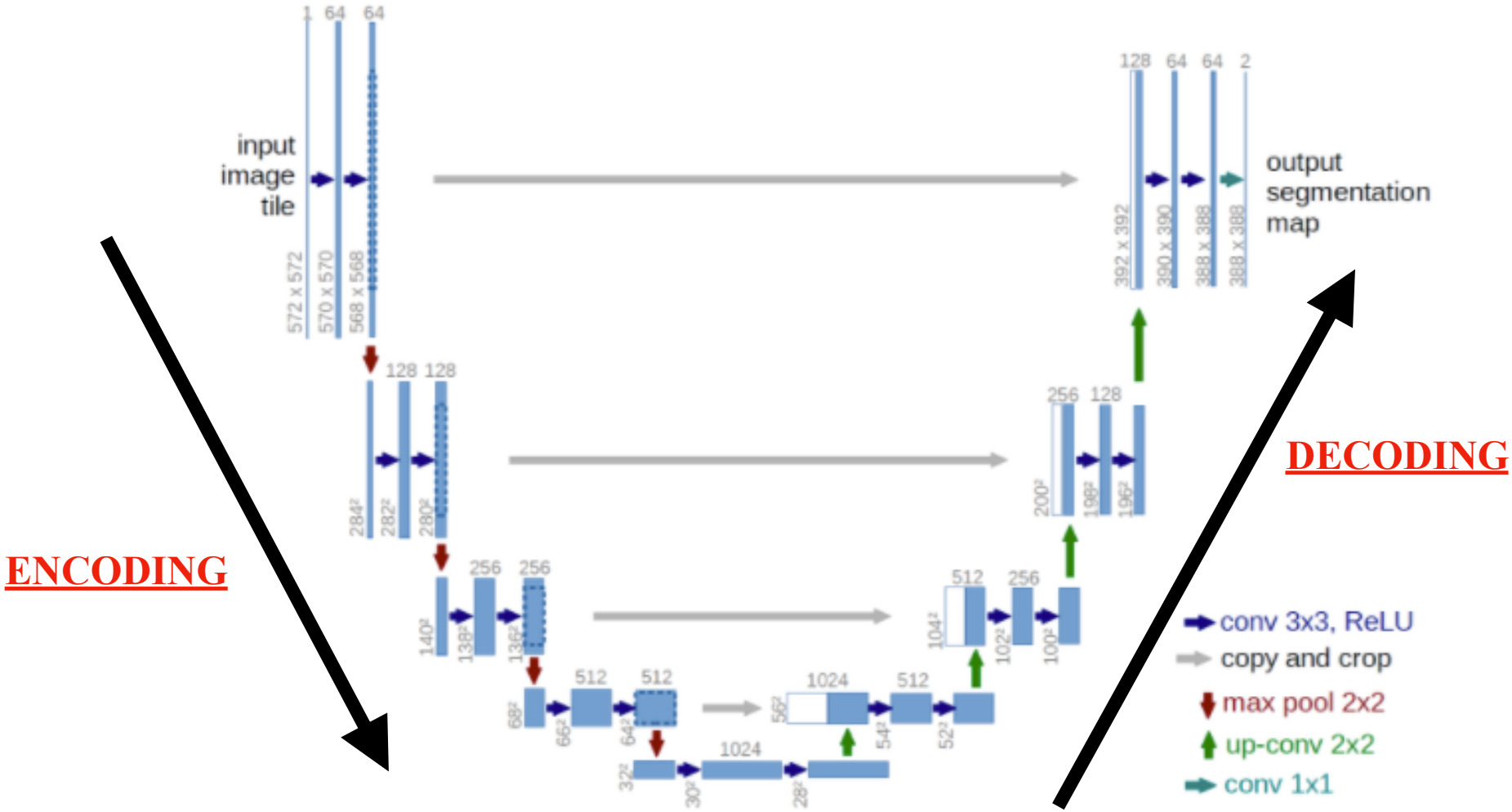
INPUT IMAGES



OUTPUT IMAGES

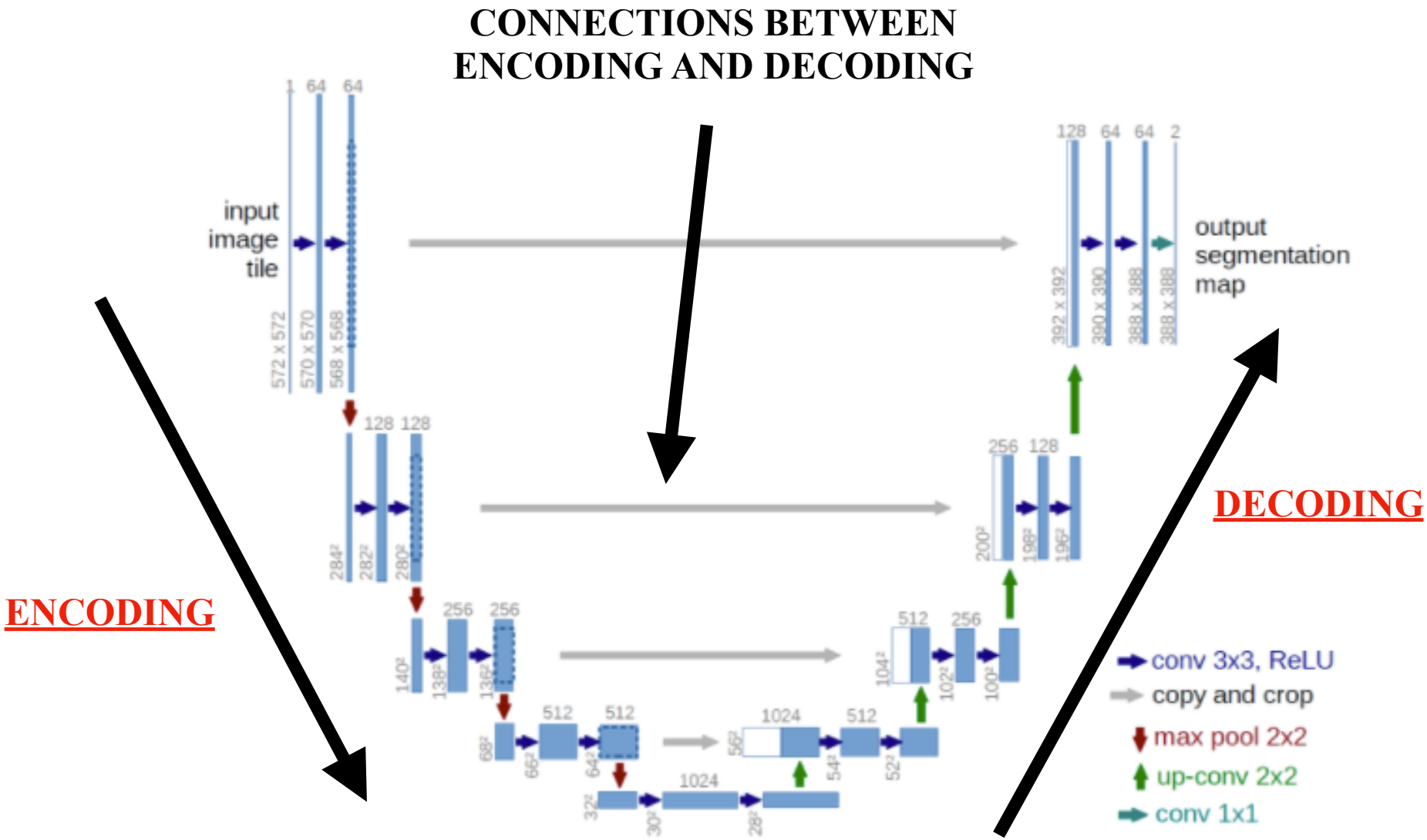
Ronnerberger+15

ENCODING-DECODING TO EXTRACT IMAGE FEATURES: U-NET



Ronnerberger+15

ENCODING-DECODING TO EXTRACT IMAGE FEATURES: U-NET

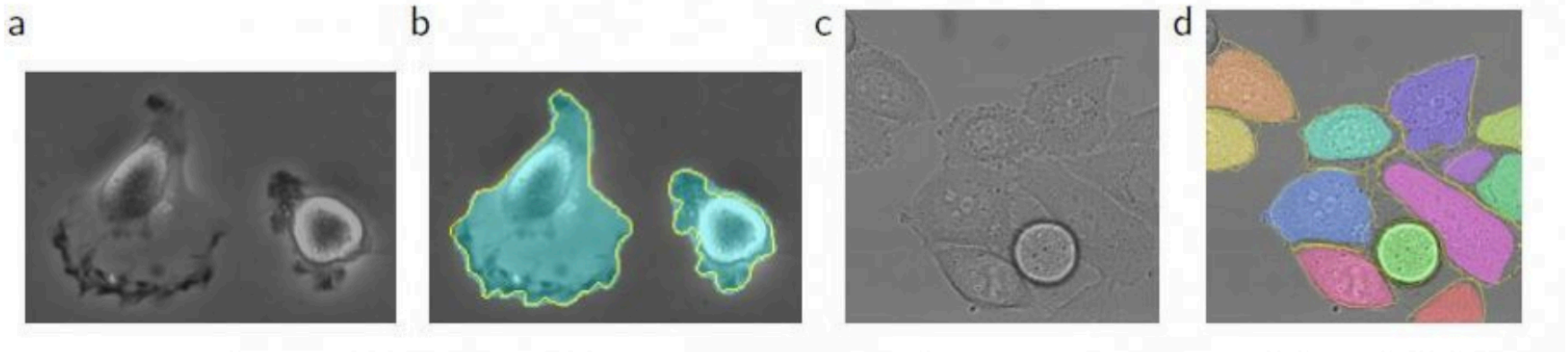


Ronnerberger+15

WHAT IS THIS USEFUL FOR?

WHAT IS THIS USEFUL FOR?

A STANDARD TOOL FOR IMAGE SEGMENTATION

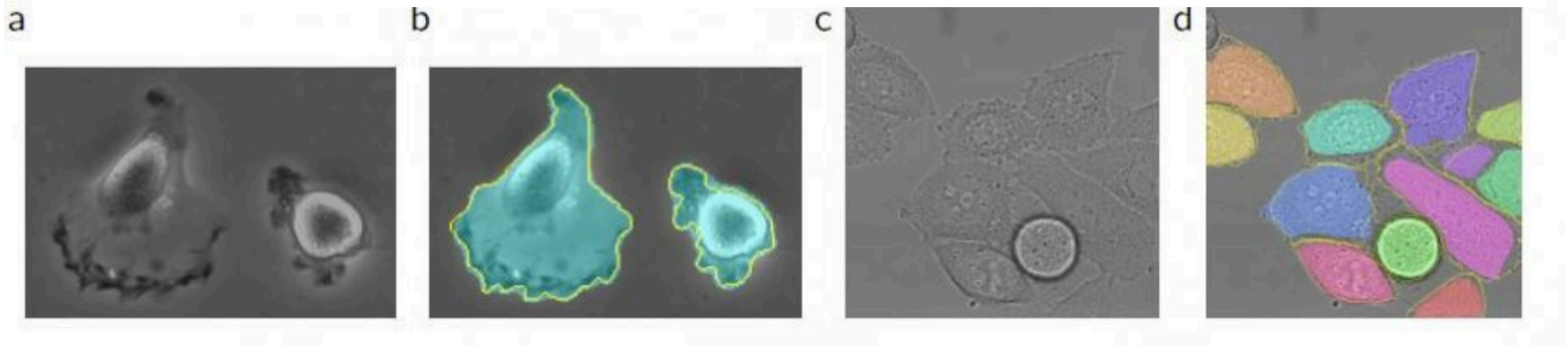


Result on the ISBI cell tracking challenge. (a) part of an input image of the PhC-U373 data set. (b) Segmentation result (cyan mask) with the manual ground truth (yellow border) (c) input image of the DIC-HeLa data set. (d) Segmentation result (random colored masks) with the manual ground truth (yellow border).

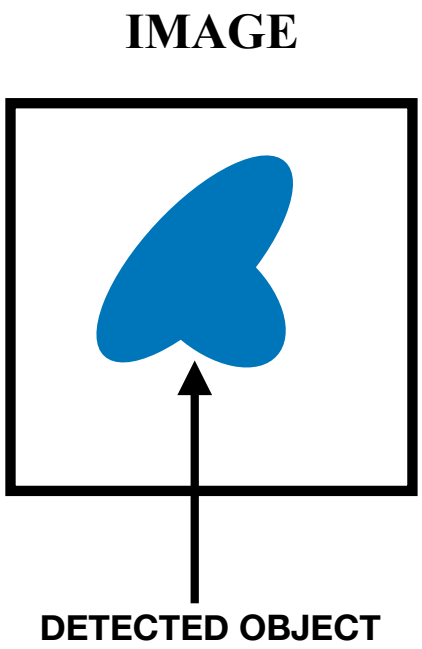
WHAT IS THIS USEFUL FOR?

A STANDARD TOOL FOR IMAGE SEGMENTATION

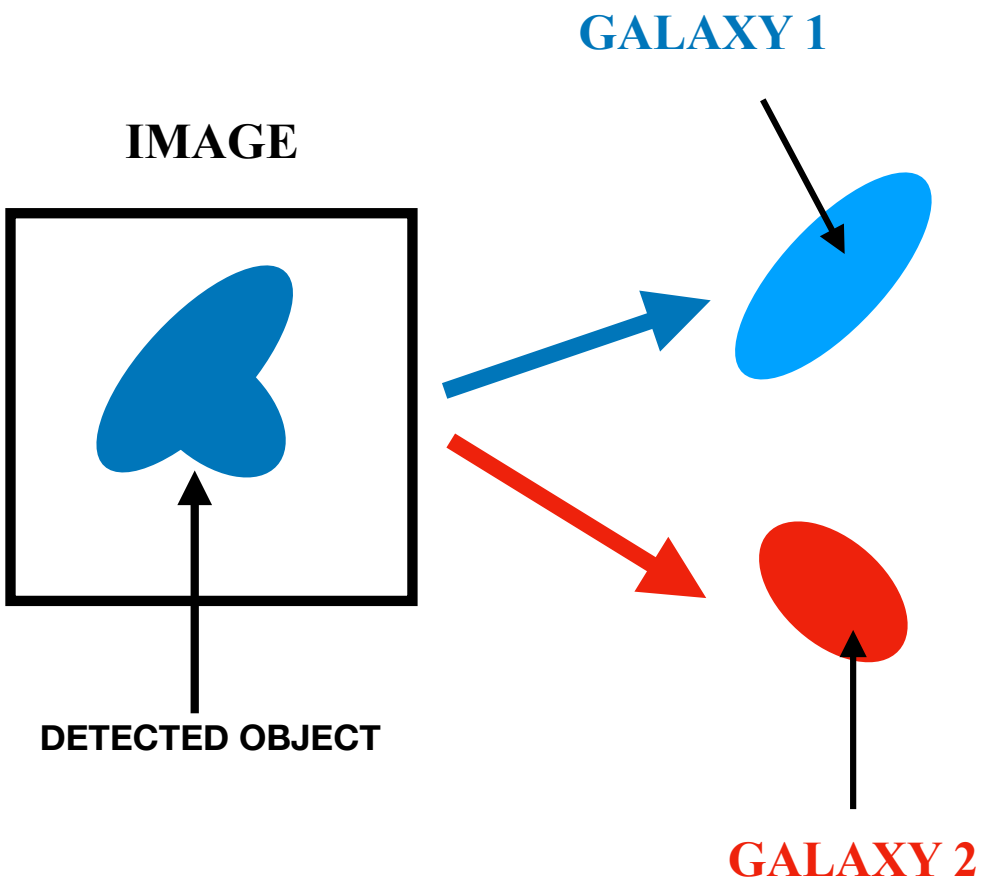
MORE ON THE SEMINAR LATER TODAY



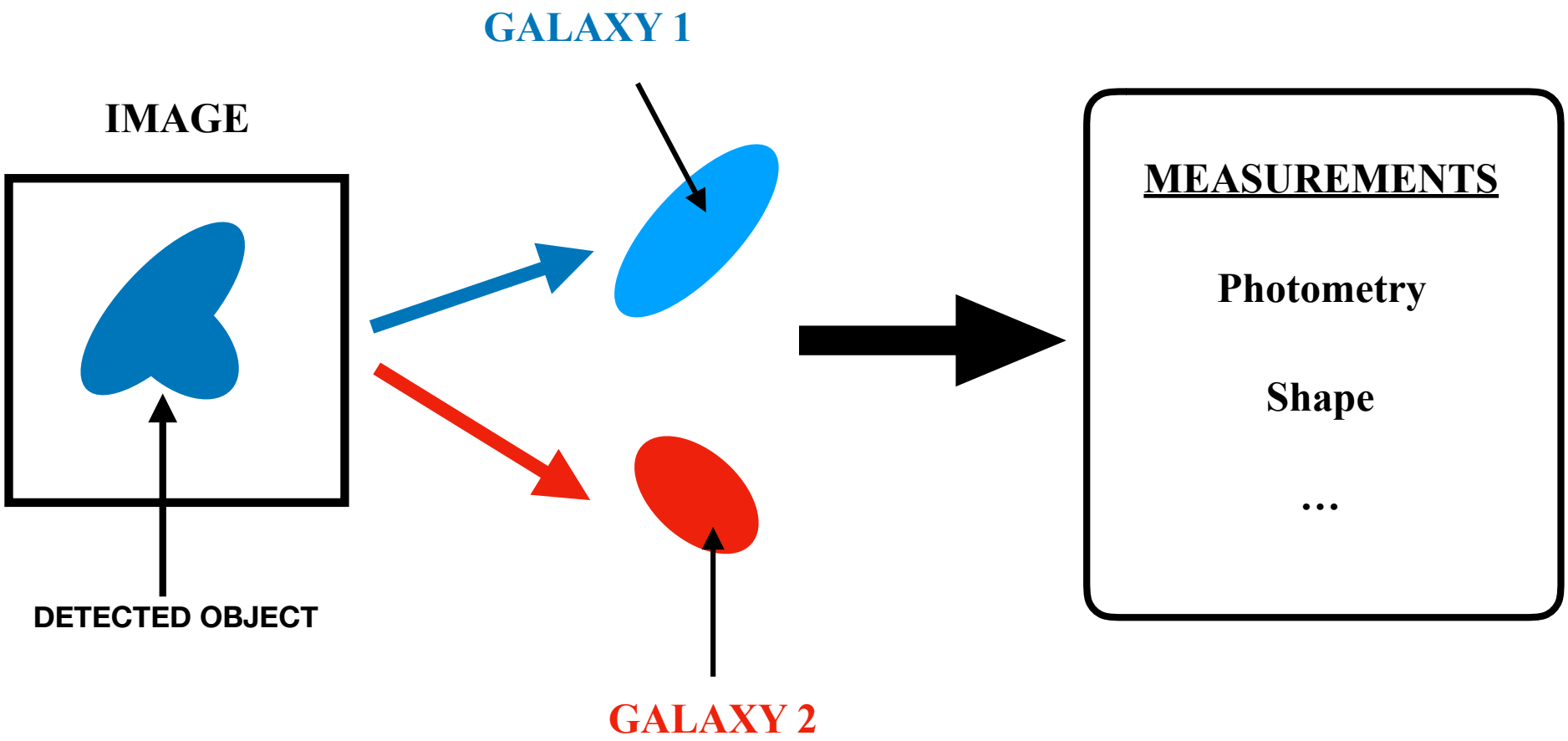
Result on the ISBI cell tracking challenge. (a) part of an input image of the PhC-U373 data set. (b) Segmentation result (cyan mask) with the manual ground truth (yellow border) (c) input image of the DIC-HeLa data set. (d) Segmentation result (random colored masks) with the manual ground truth (yellow border).



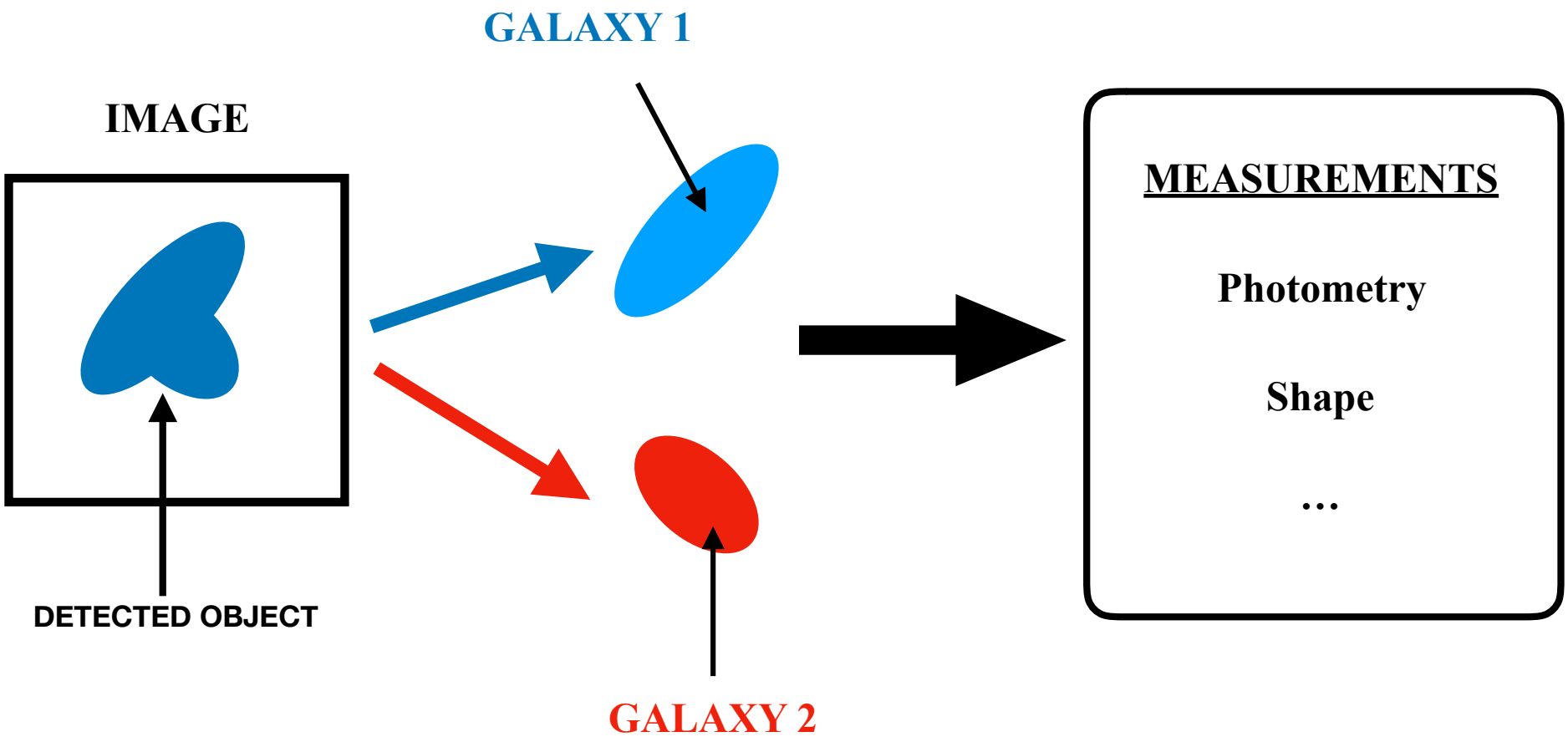
Boucaud+19



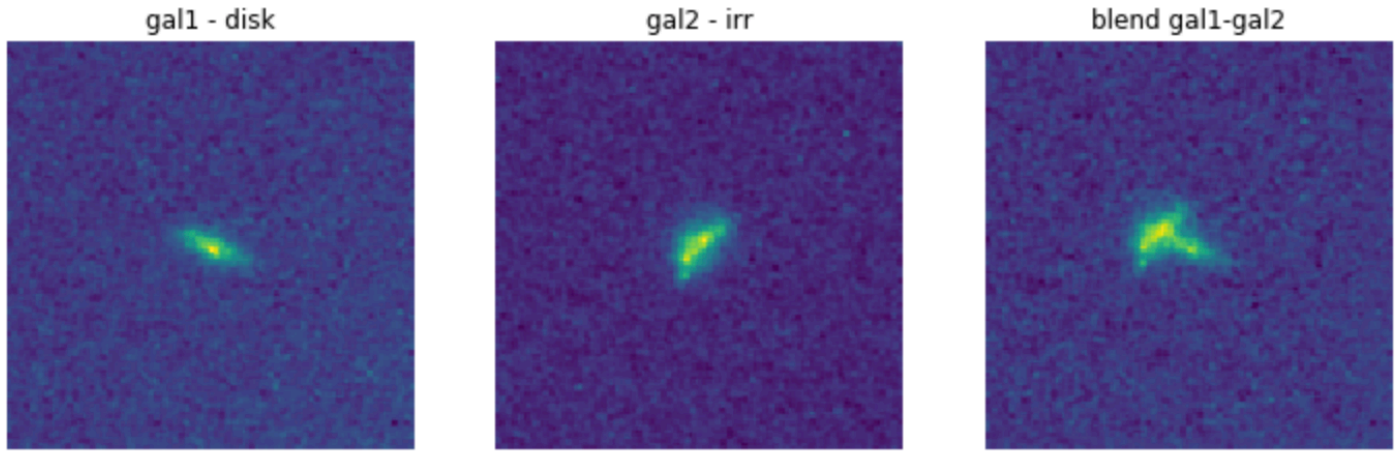
Boucaud+19


GALAXY 1
IMAGE
MEASUREMENTS
Photometry
Shape
…
DETECTED OBJECT
GALAXY 2

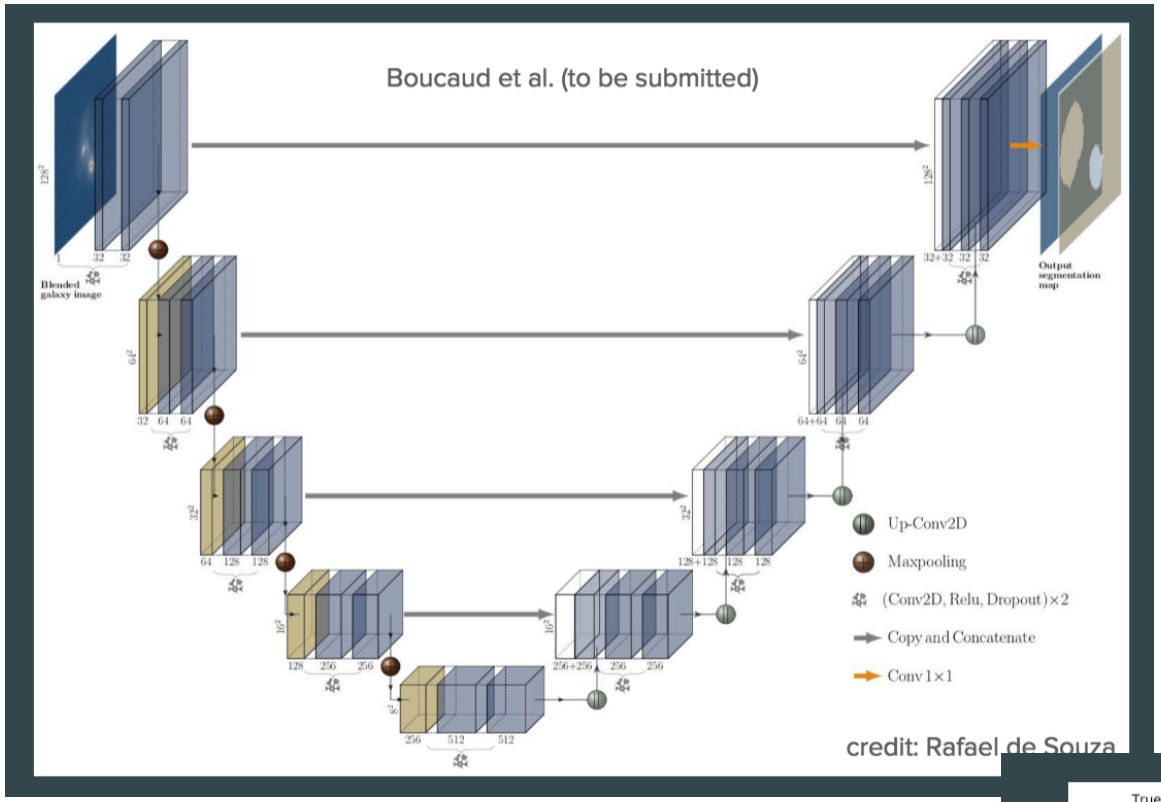
Boucaud+19

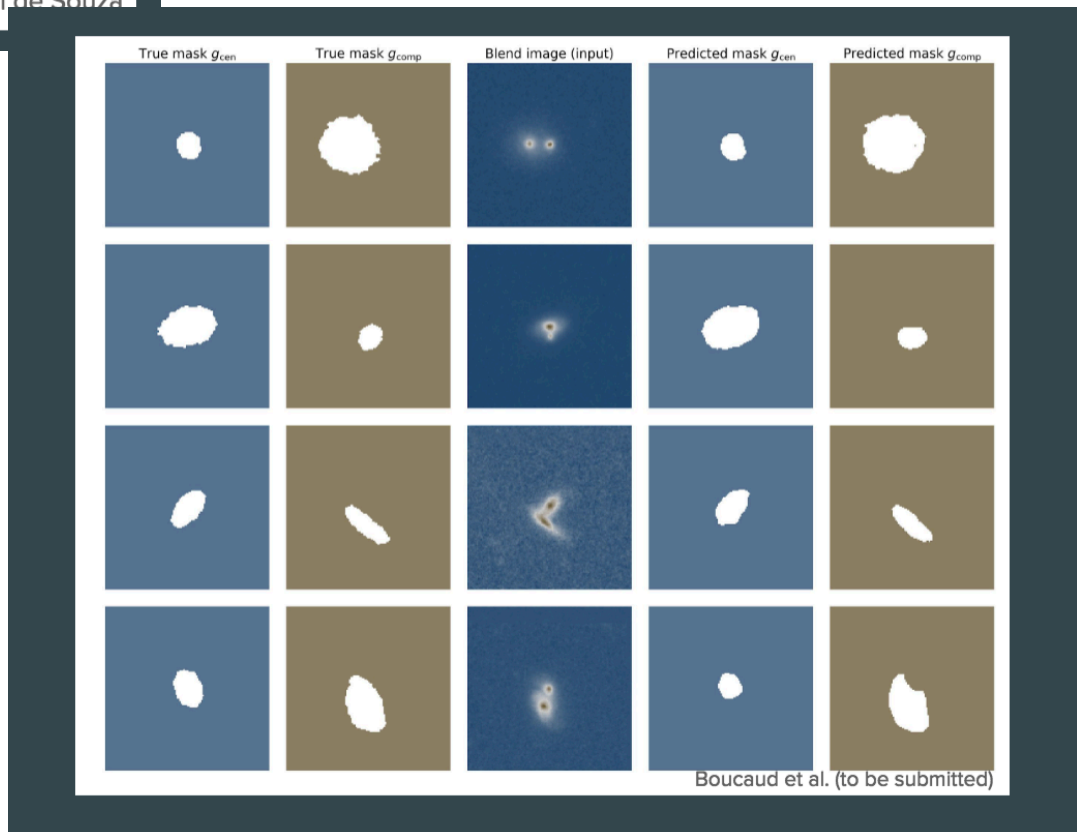


ISOLATED GALAXIES ARTIFICALLY BLENDED



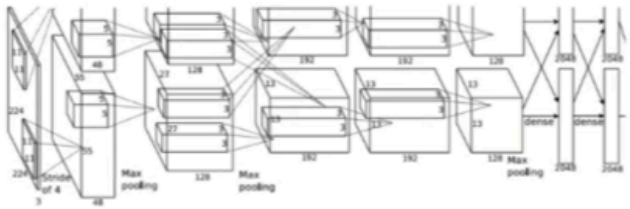
Boucaud+19


Boucaud et al. (to be submitted)
Blended galaxy image
Output segmentation map
Up-Conv2D
Maxpooling
(Conv2D, Relu, Dropout)×2
Copy and Concatenate
Conv 1×1
credit: Rafael de Souza

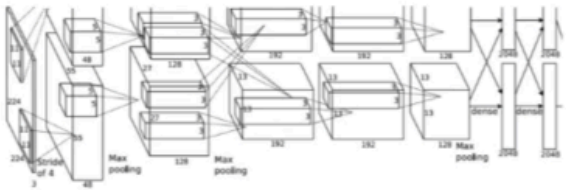

True mask g_{cen}
True mask g_{comp}
Blend image (input)
Predicted mask g_{cen}
Predicted mask g_{comp}
Boucaud et al. (to be submitted)

A WORD ON R-CNNs..

JUST FOR YOUR RECORDS…



CAT: (x, y, w, h)

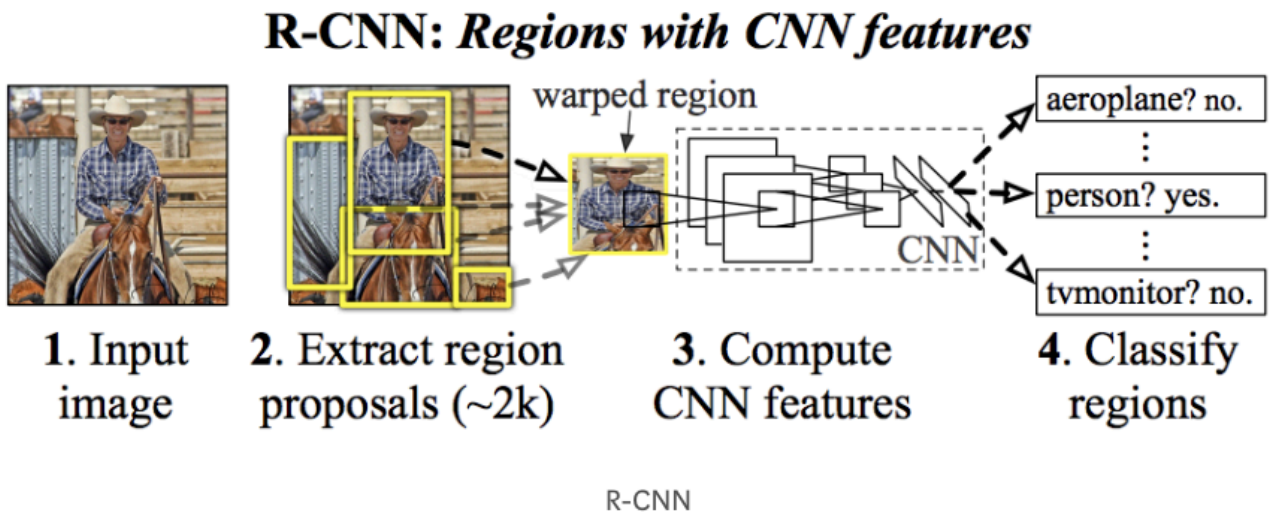


DUCK: (x, y, w, h)
DUCK: (x, y, w, h)
….

Girshick+14

A WORD ON R-CNNs..

JUST FOR YOUR RECORDS…

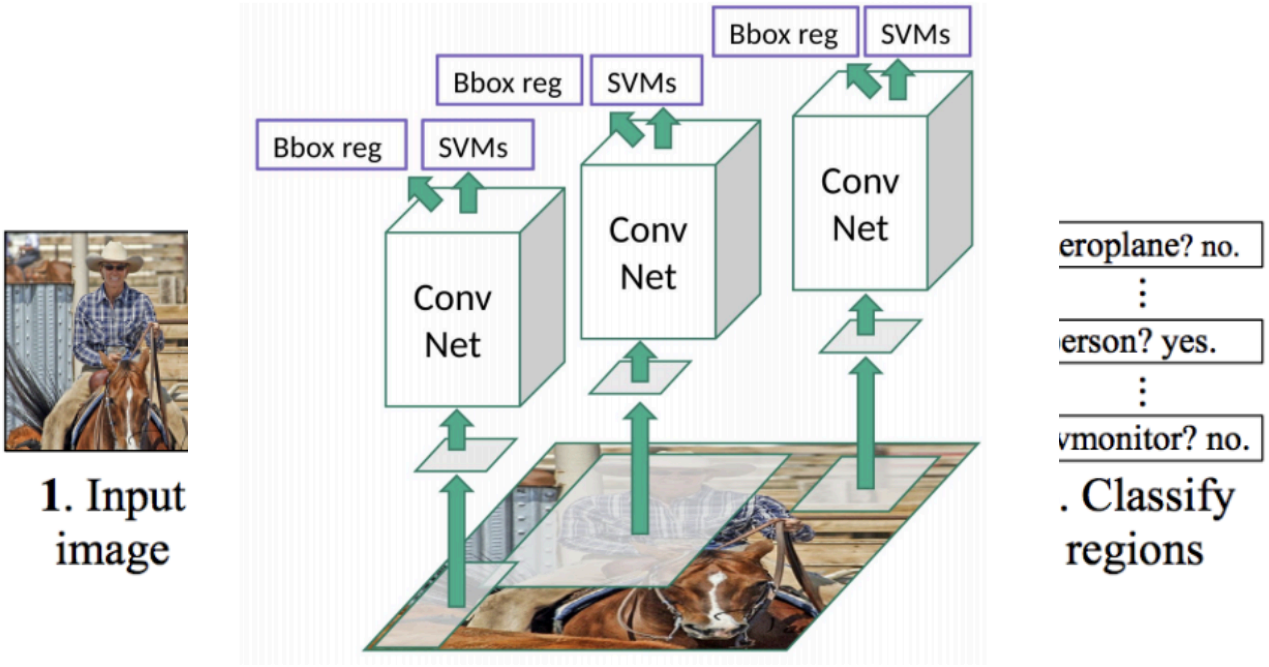


R-CNN

Girshick+14

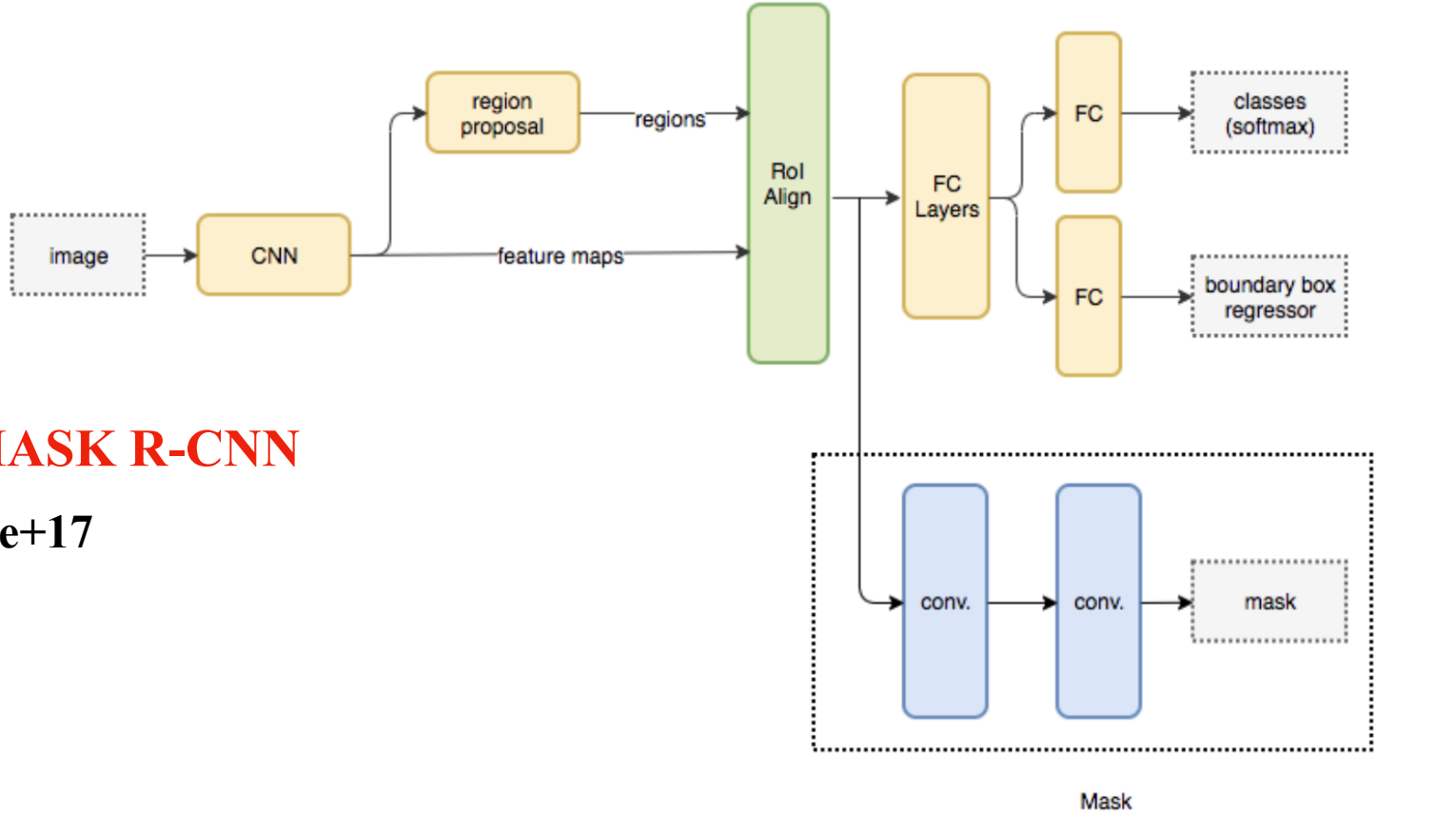
A WORKD ON R-CNNs..

JUST FOR YOUR RECORDS…



Girshick+14

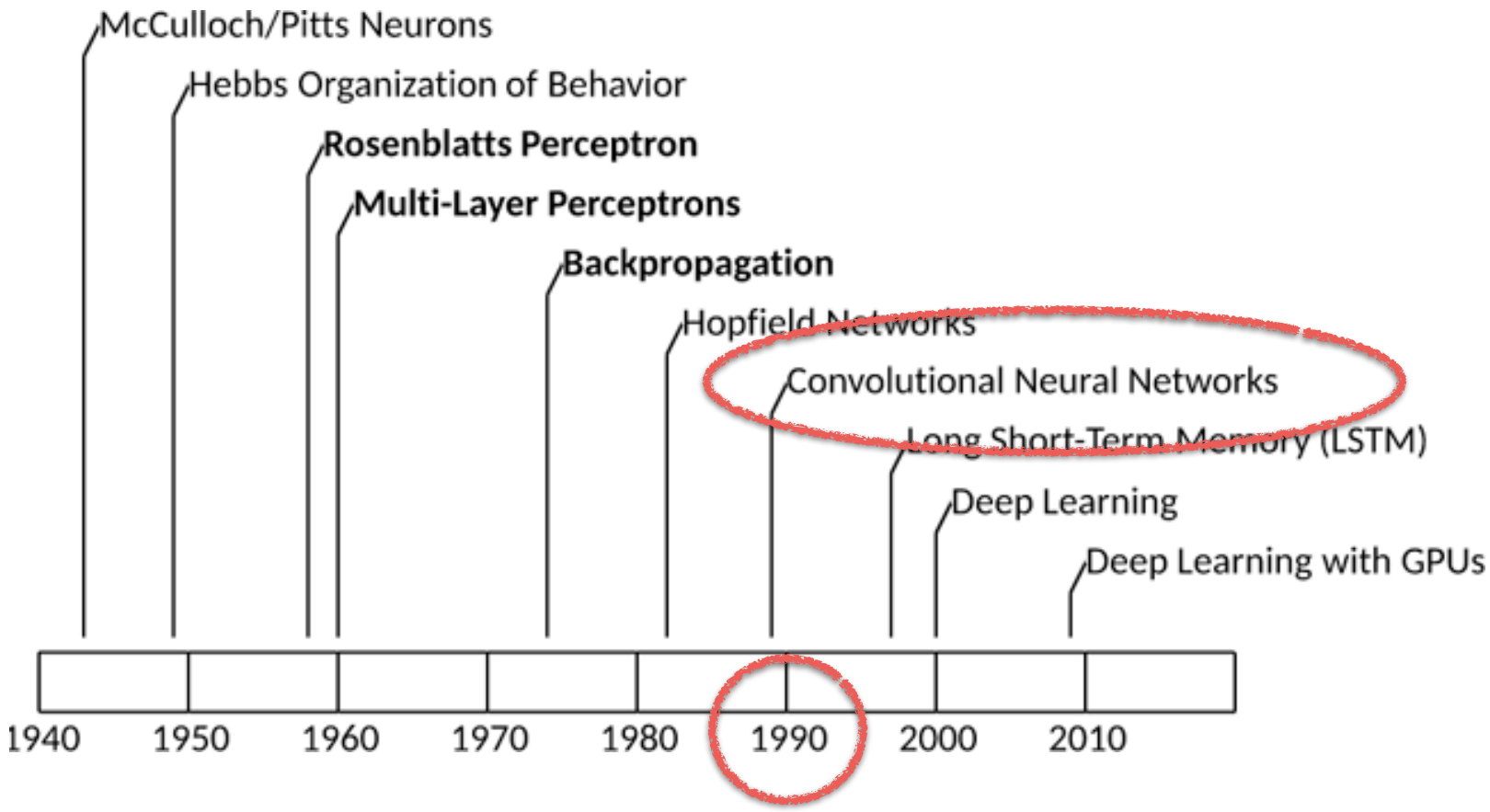
SIMULTANEOUS DETECTION + SEGMENTATION + CLASSIFICATION



MASK R-CNN

He+17

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 vs. CPUs

CPUs

FEWER CORES (~10x)

EACH CORE IS FASTER

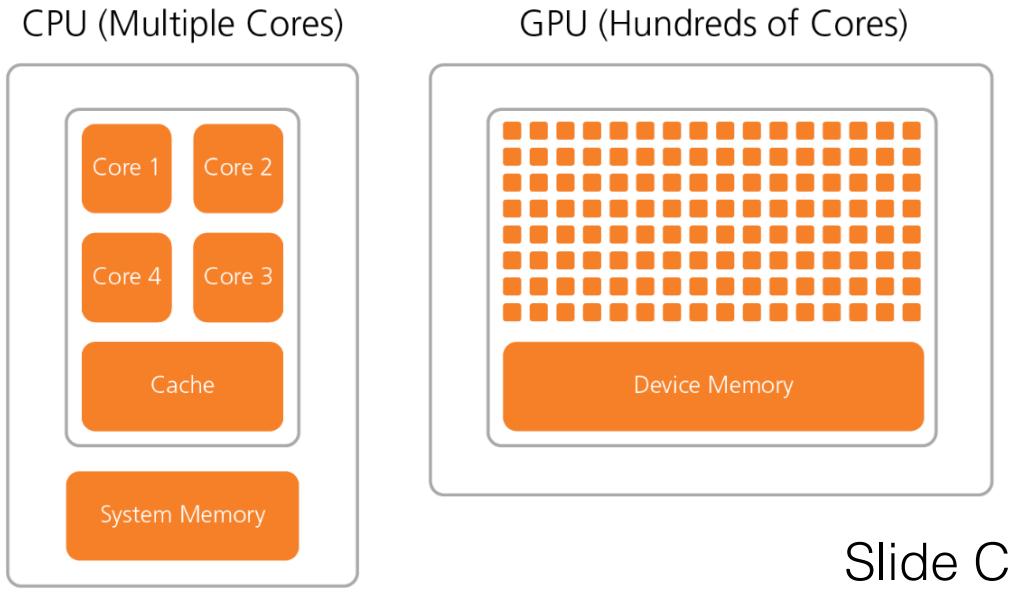
USEFUL FOR SEQUENTIAL TASKS

GPUs

MORE CORES (100x)

EACH CORE IS SLOWER

USEFUL FOR PARALLEL TASKS



Slide Credit:

GPUs vs. CPUs

More benchmarks available here.

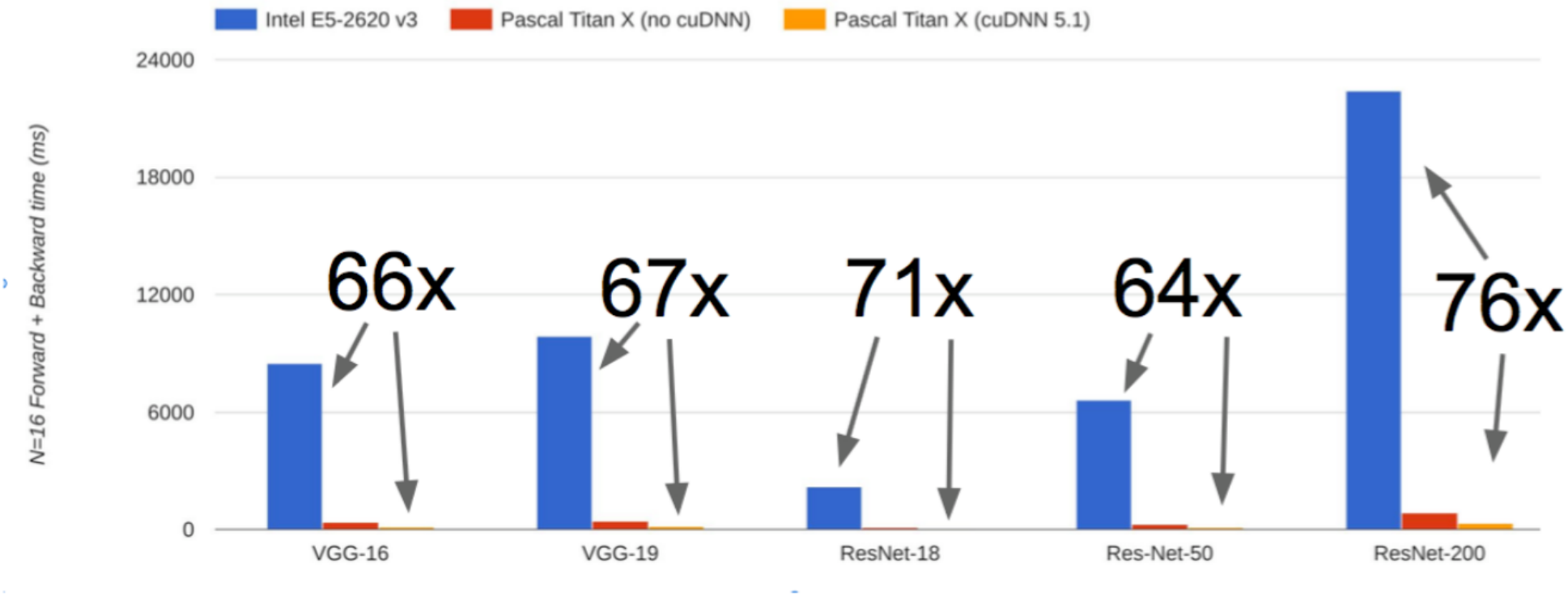


Figure credit: J. Johnson

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

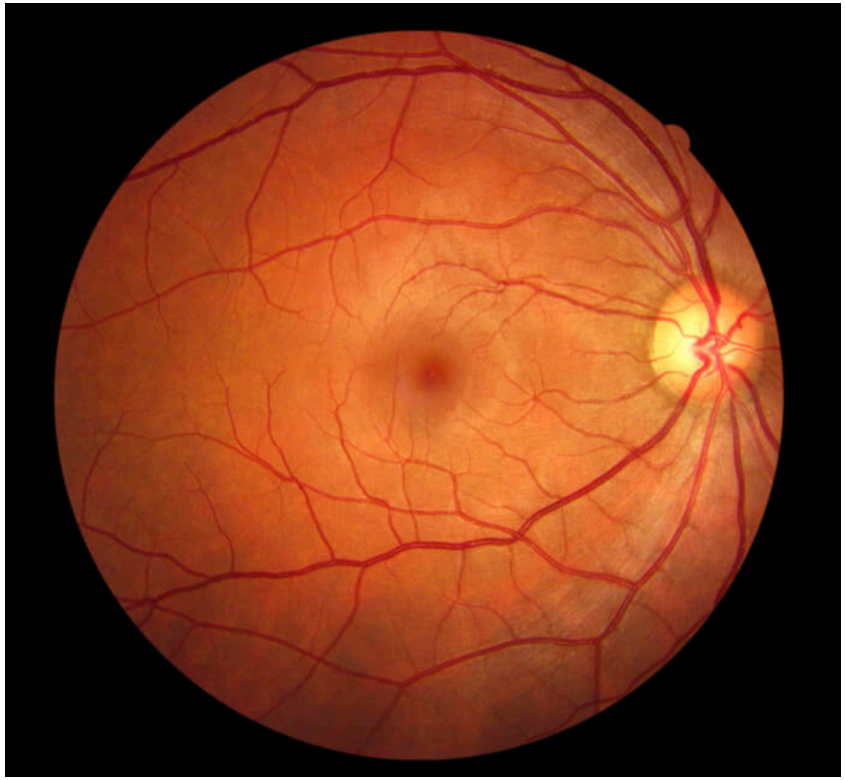


IMAGE OF THE BACK OF THE EYE

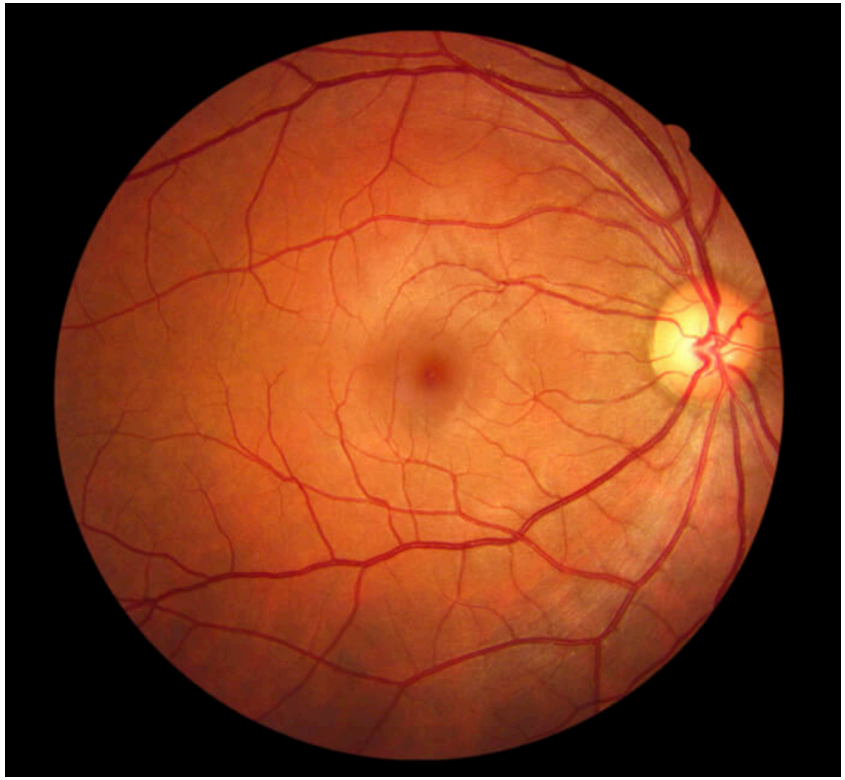


IMAGE OF THE BACK OF THE EYE

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

VISUALIZING CNNs
[understanding CNNs decisions]

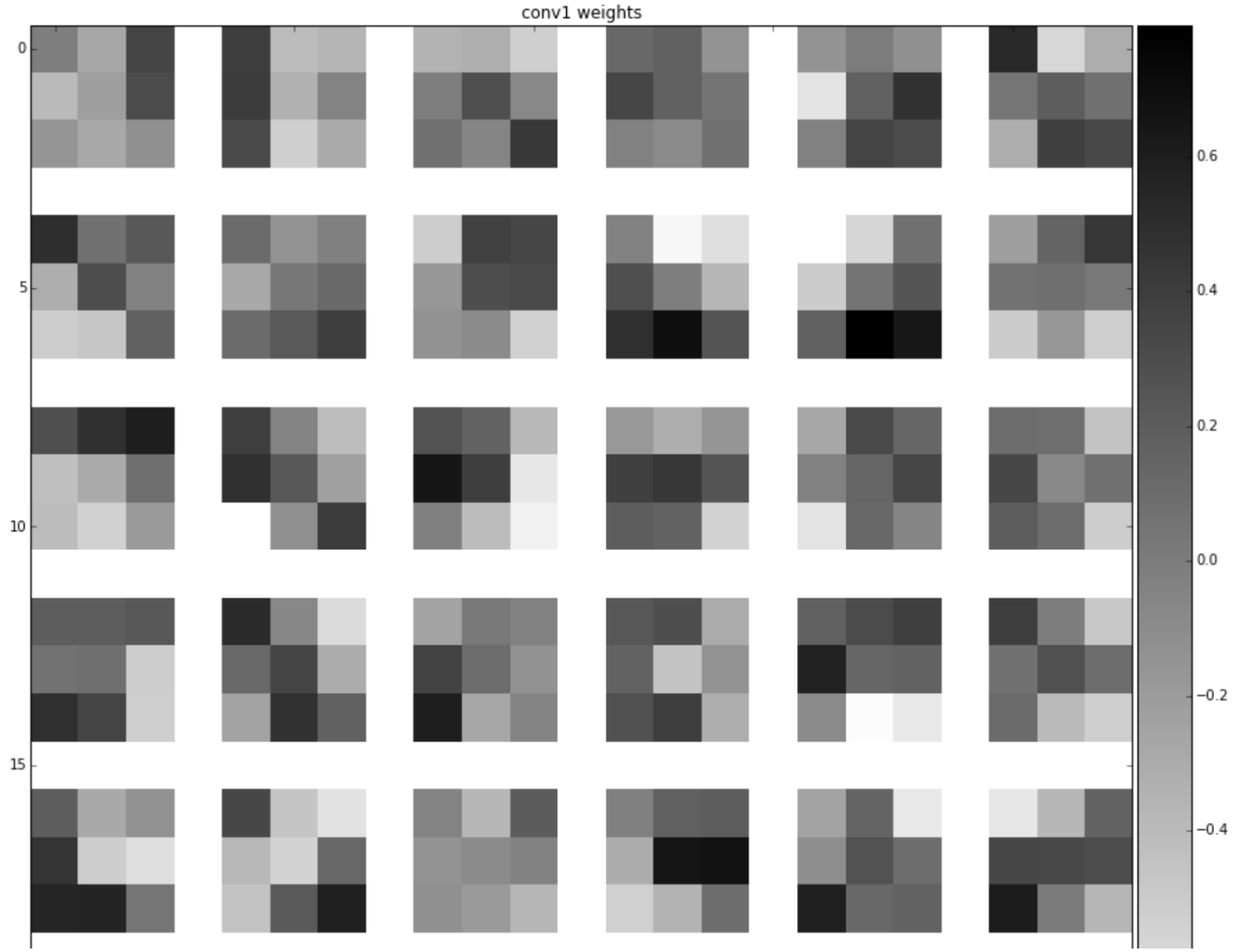
Attribution techniques

DEEP NETWORKS ARE “BLACK BOXES”?

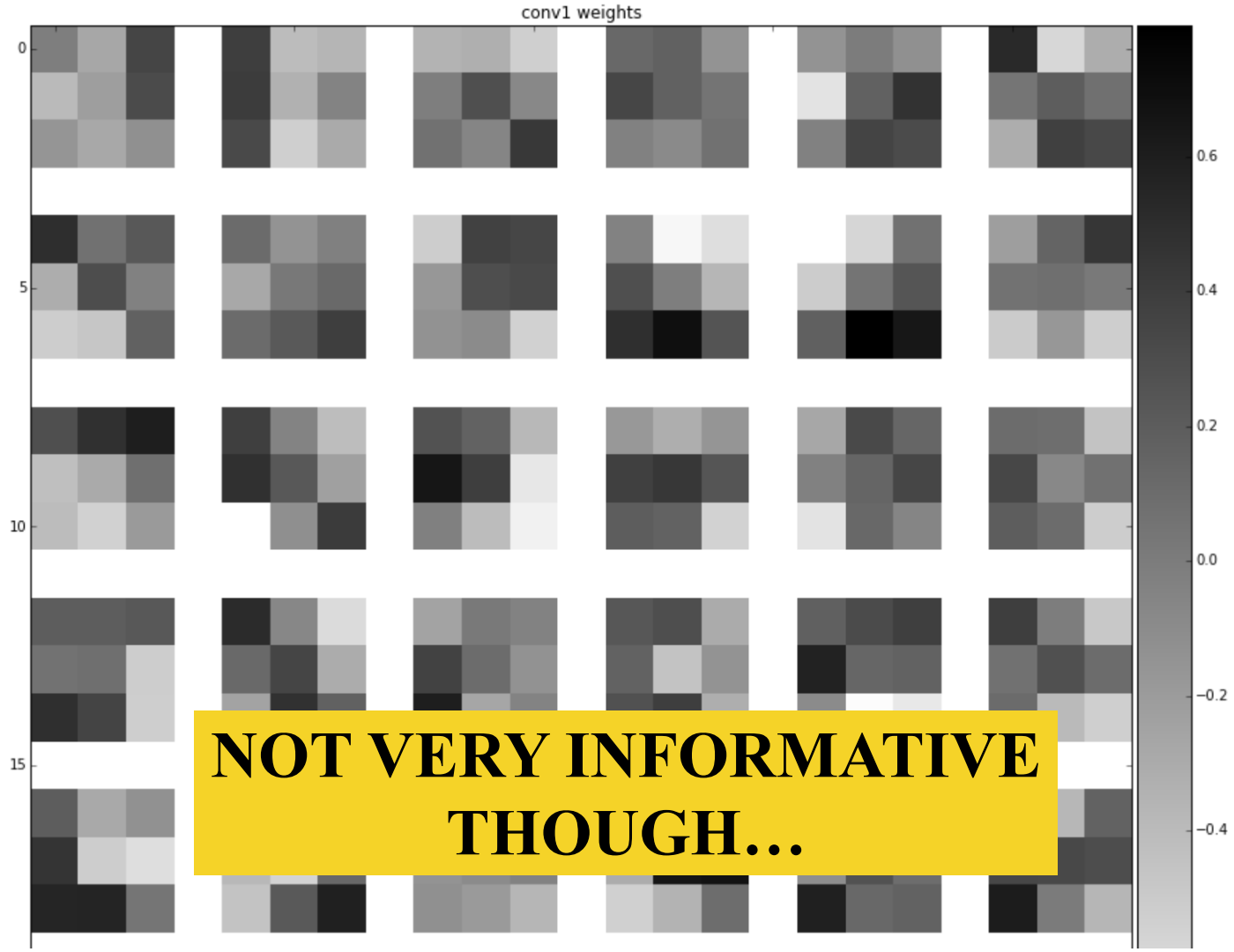
INTERPRETING THE RESULTS IS
EXTREMELY DIFFICULT

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

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



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



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

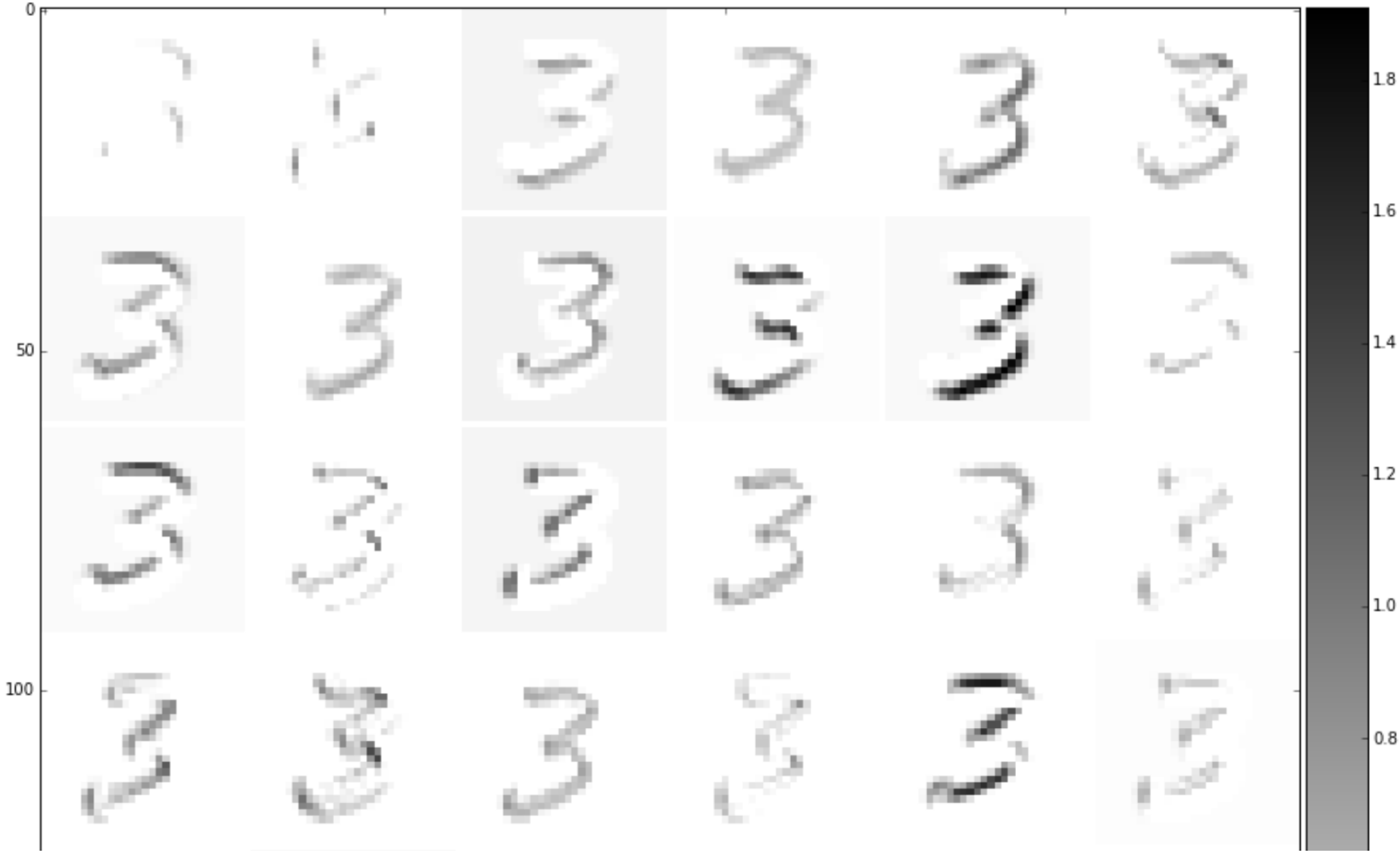
IN KERAS:

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


0
50
100
1.8
1.6
1.4
1.2
1.0
0.8

KERAS:

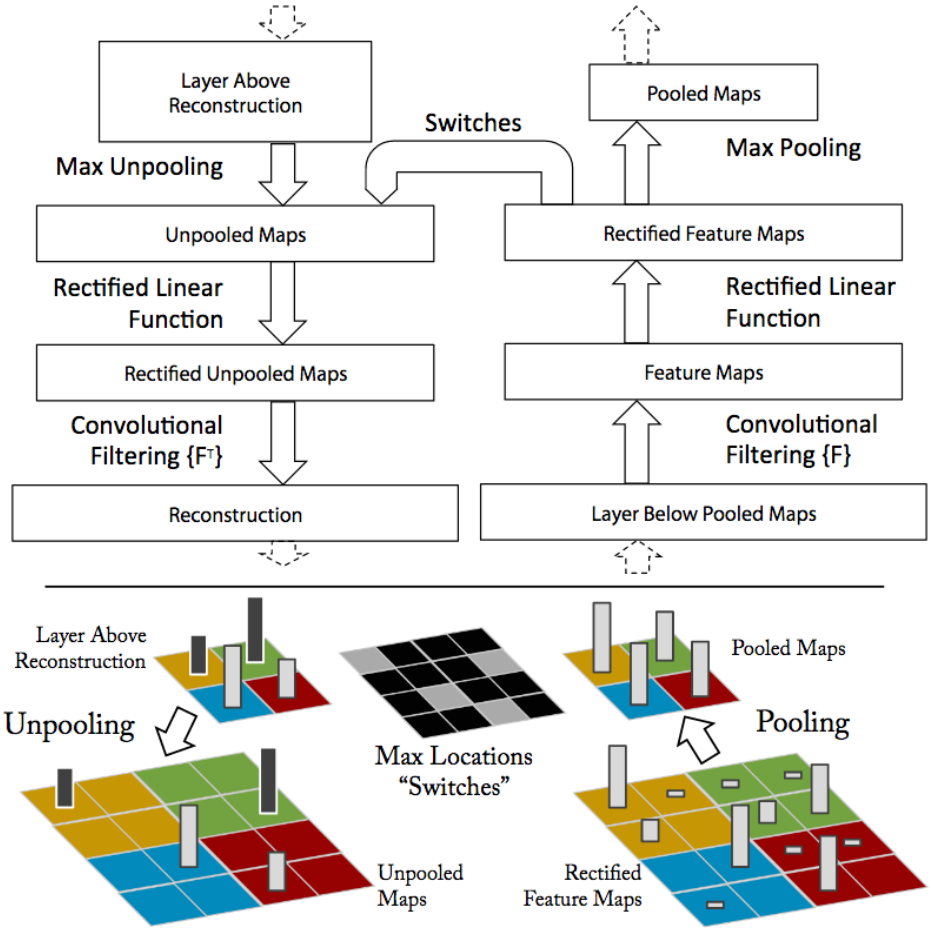
```
def display_all_layer_filters(model, layer_dict, layer_name, input_img, dir_out=None):
    """ """
    if len(input_img.shape) == 2:  # grey level image
        img = input_img.reshape(1, 1, input_img.shape[0], input_img.shape[1])
    elif len(input_img.shape) == 3:
        img = input_img.reshape(1, input_img.shape[0], input_img.shape[1], input_img.shape[2])
    else:
        raise TypeError("Input image should have 2 or 3 dimensions")

    get_layer_output = K.function([model.layers[0].input, K.learning_phase()],
[layer_dict[layer_name].output, K.learning_phase()])
    learning_ph = 0  # we are testing
    layer_output = get_layer_output([img, learning_ph])[0]

    number_of_filters = layer_output.shape[1]
    rows = np.uint8(np.floor(np.sqrt(number_of_filters)))
    cols = number_of_filters // rows
    rest = number_of_filters % rows
    if rest > 0:
        rows += 1
    fig, axarray = plt.subplots(rows, cols, sharex='col', sharey='row', figsize=(18,12))
    fig.suptitle(layer_name)

    for r in range(rows):
        for c in range(cols):
            index = r+rows*c
            if index >= number_of_filters:
                break
            # axarray[r,c].set_title()
            axarray[r,c].imshow(layer_output[0, index, :, :], interpolation="nearest", cmap="gray")
```

KERAS:

```
def display_all_layer_filters(model, layer_dict, layer_name, input_img, dir_out=None):
    """ """
    if len(input_img.shape) == 2:  # grey level image
        img = input_img.reshape(1, 1, input_img.shape[0], input_img.shape[1])
    elif len(input_img.shape) == 3:
        img = input_img.reshape(1, input_img.shape[0], input_img.shape[1], input_img.shape[2])
    else:
        raise TypeError("Input image should have 2 or 3 dimensions")

    get_layer_output = K.function([model.layers[0].input, K.learning_phase()],
[layer_dict[layer_name].output, K.learning_phase()])
    learning_ph = 0  # we are testing
    layer_output = get_layer_output([img, learning_ph])[0]

    number_of_filters = layer_output.shape[1]
    rows = np.uint8(np.floor(np.sqrt(number_of_filters)))
    cols = number_of_filters // rows
    rest = number_of_filters % rows
    if rest > 0:
        rows += 1
    fig, axarray = plt.subplots(rows, cols, sharex='col', sharey='row', figsize=(18,12))
    fig.suptitle(layer_name)

    for r in range(rows):
        for c in range(cols):
            index = r+rows*c
            if index >= number_of_filters:
                break
            # axarray[r,c].set_title()
            axarray[r,c].imshow(layer_output[0, index, :, :], interpolation="nearest", cmap="gray")
```

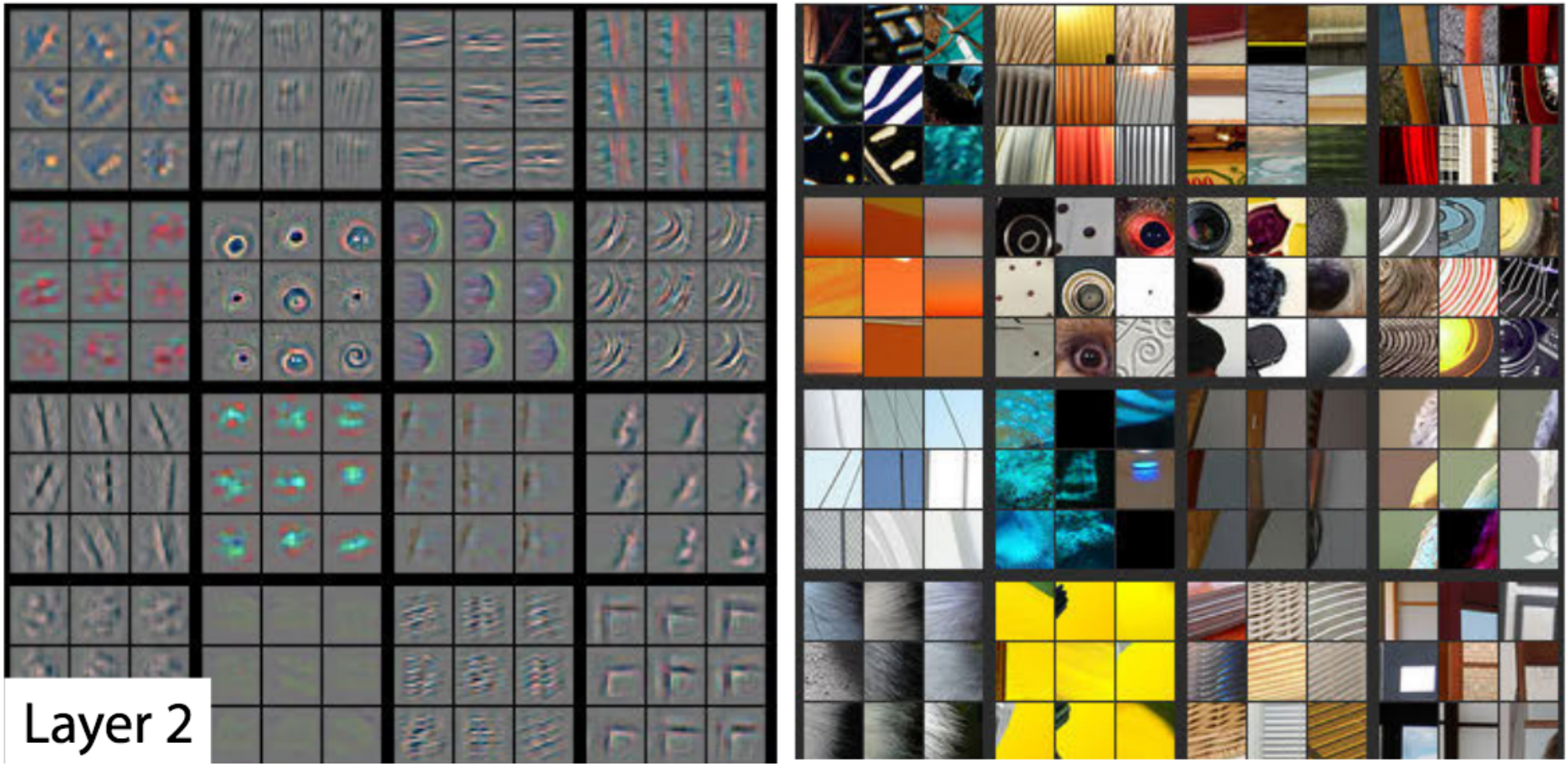
get layer output
for a series of “layer_names”

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



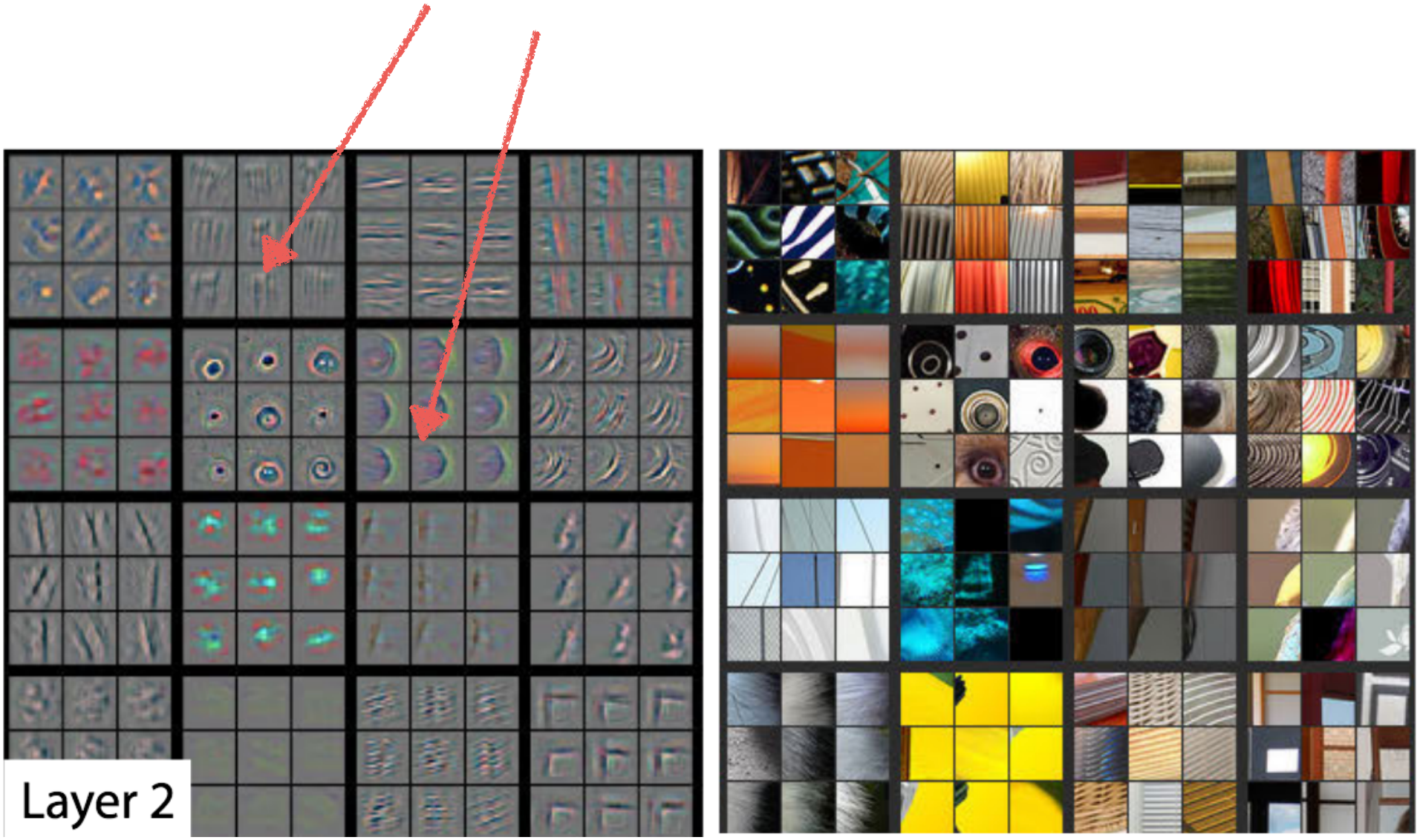
IT ALLOWS TO SEE
WHICH
REGIONS OF THE INPUT
GENERATED
A MAXIMUM RESPONSE
IN A NEURON

Zeiler+14



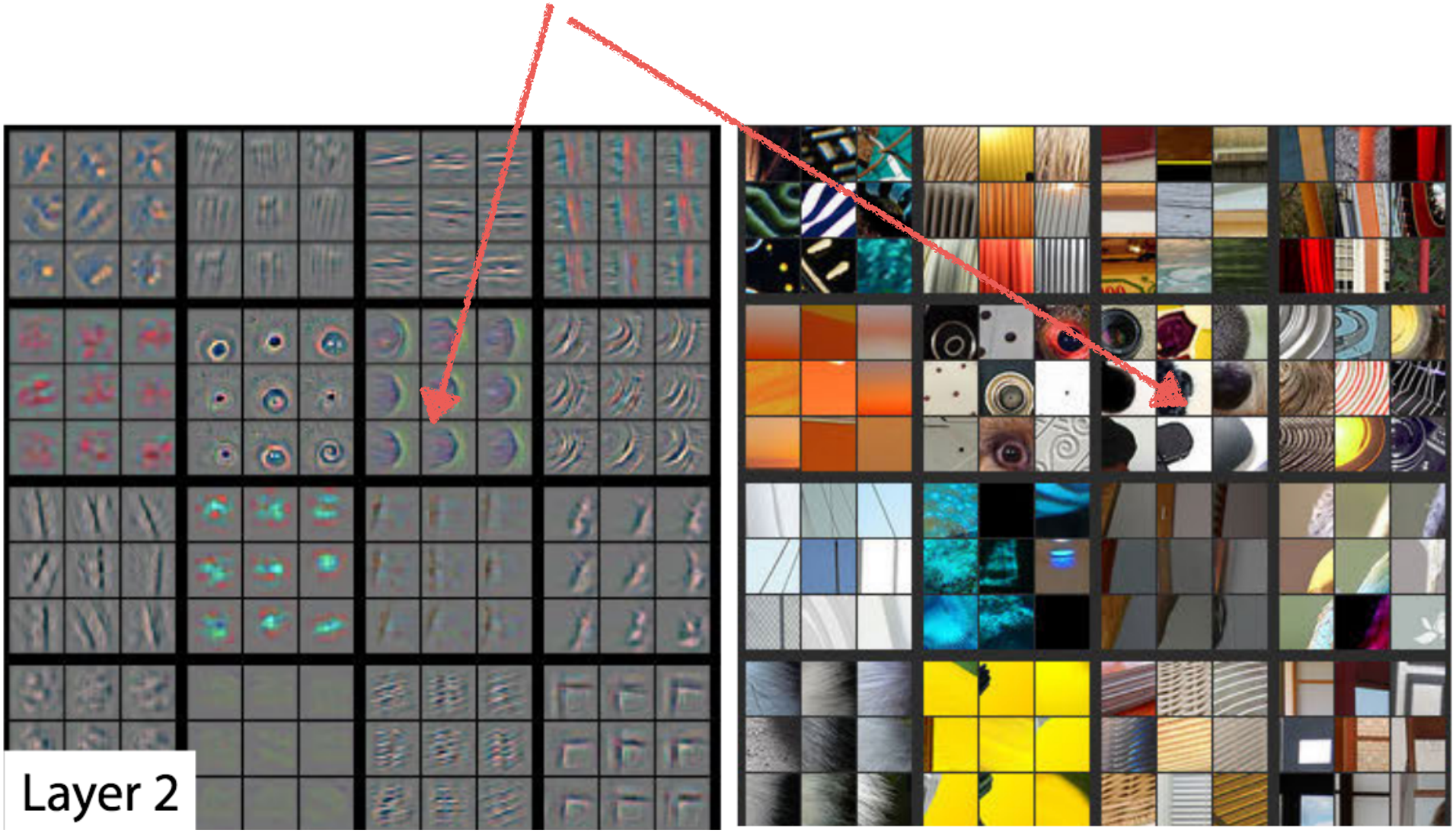
Zeiler+14

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



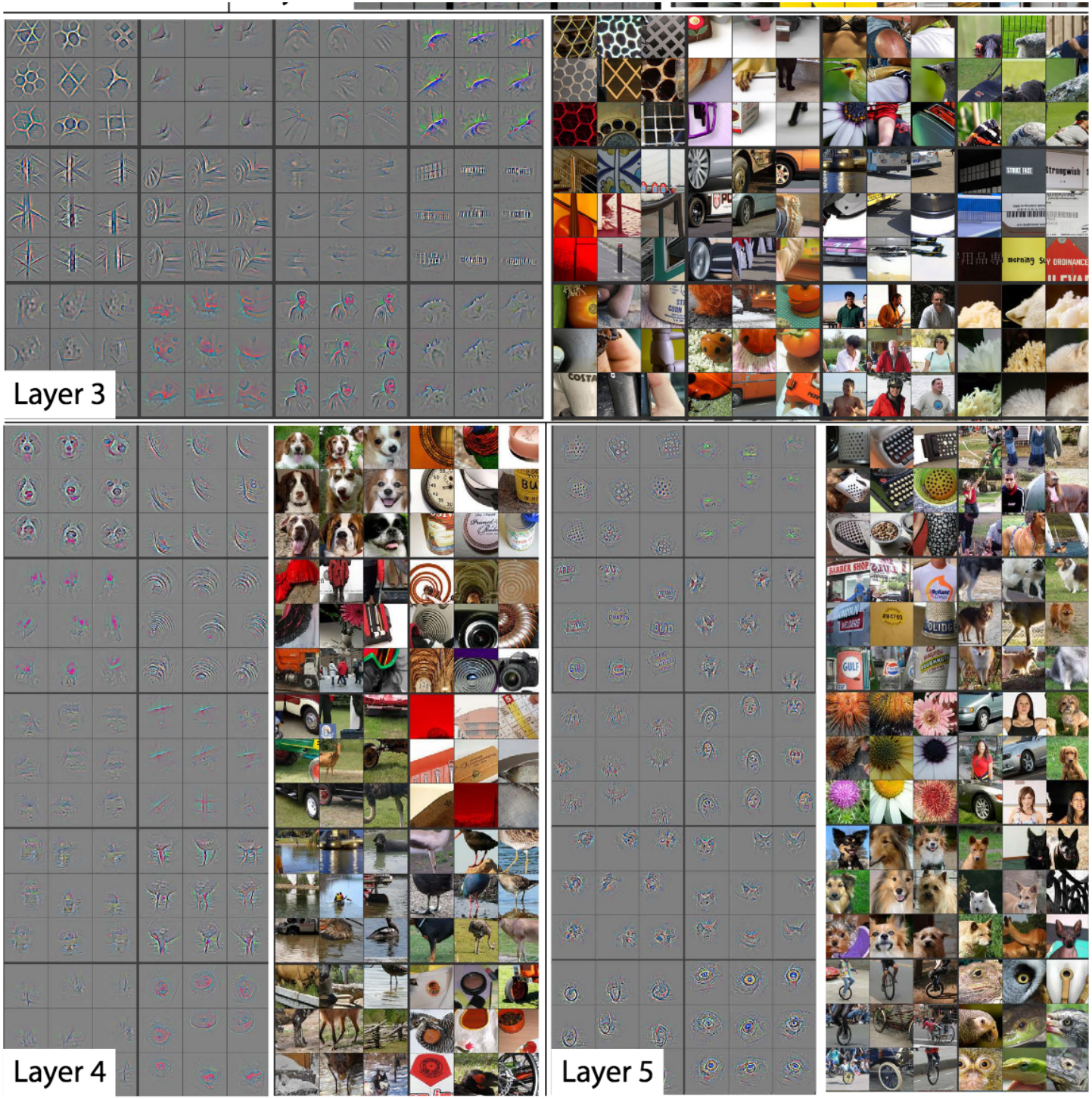
Zeiler+14

THE CORRESPONDING REGIONS OF IMAGES THAT GENERATED THE MAXIMUM RESPONSE



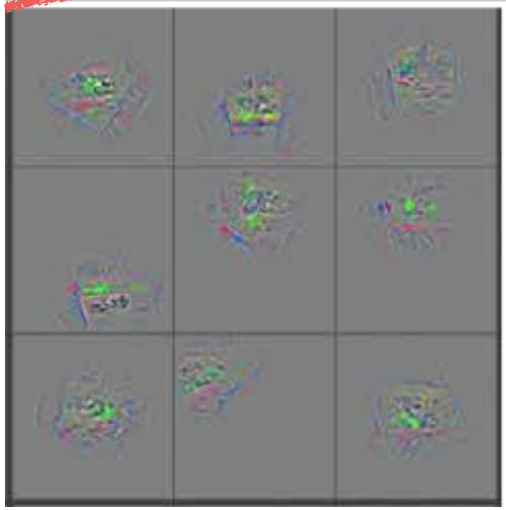
Zeiler+14

CAN BE REPEATED FOR DEEPER LAYERS ALTHOUGH IT BECOMES LESS INTUITIVE

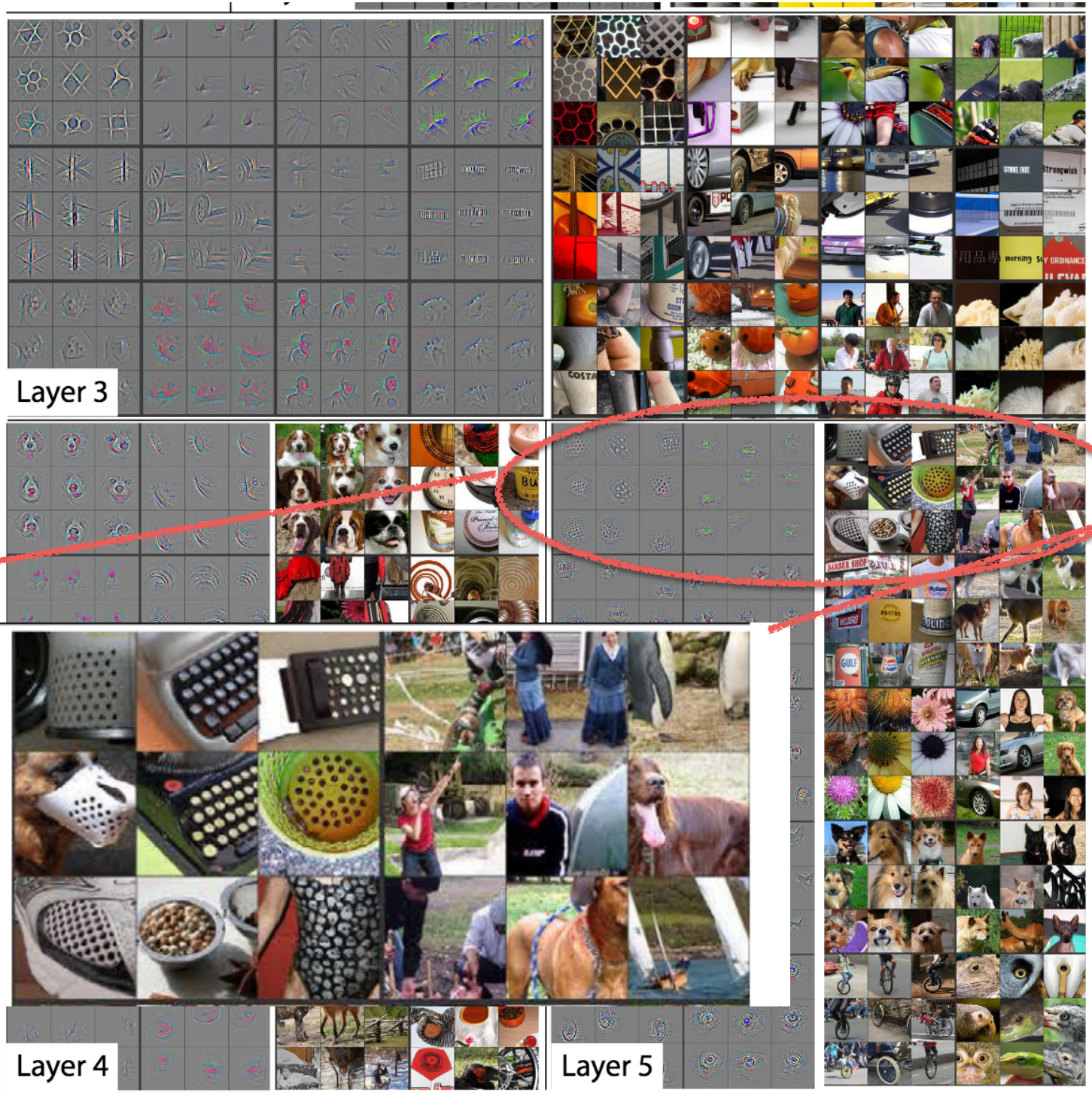


Zeiler+14

CAN BE REPEATED FOR DEEPER LAYERS ALTHOUGH IT BECOMES LESS



Zeiler+14



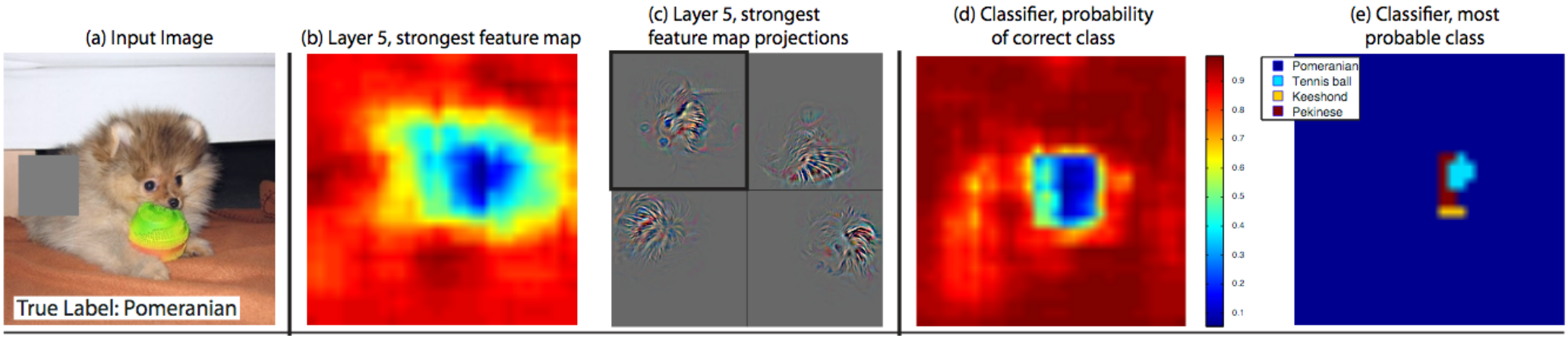
KERAS IMPLEMENTATION OF VISUALIZATIONS THROUGH DECONVNETS

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>



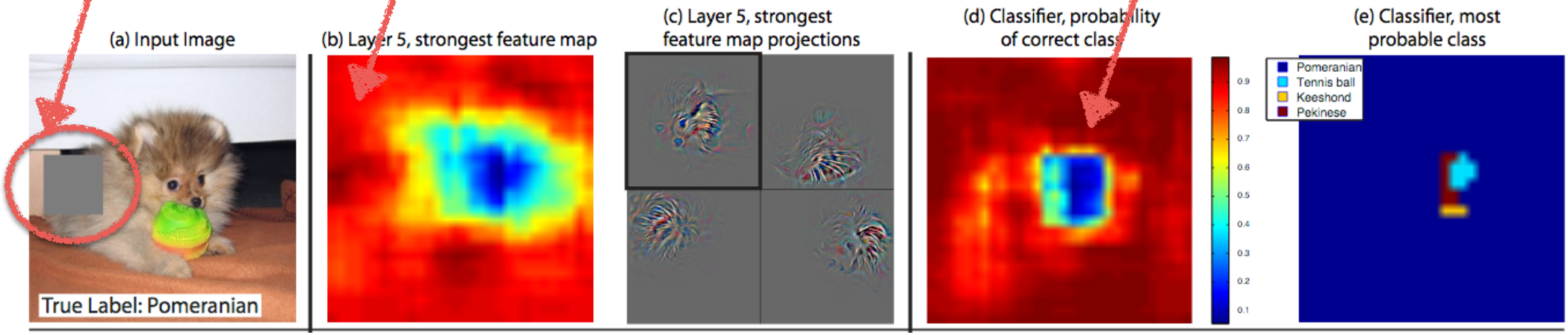
Zeiler+14

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

occluded region

the output probability as a
function of the occluding square
position



Zeiler+14

INCEPTIONISM - DEEP DREAM

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

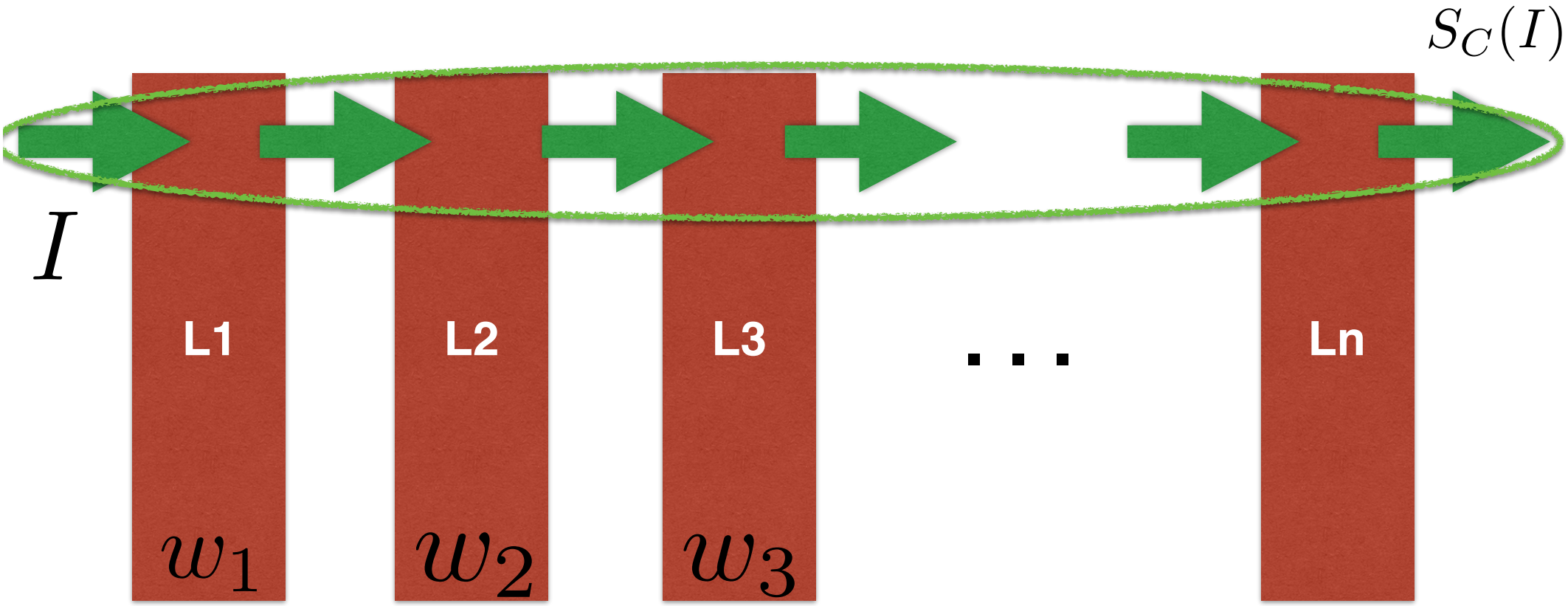
image I

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

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

Simonyan+14

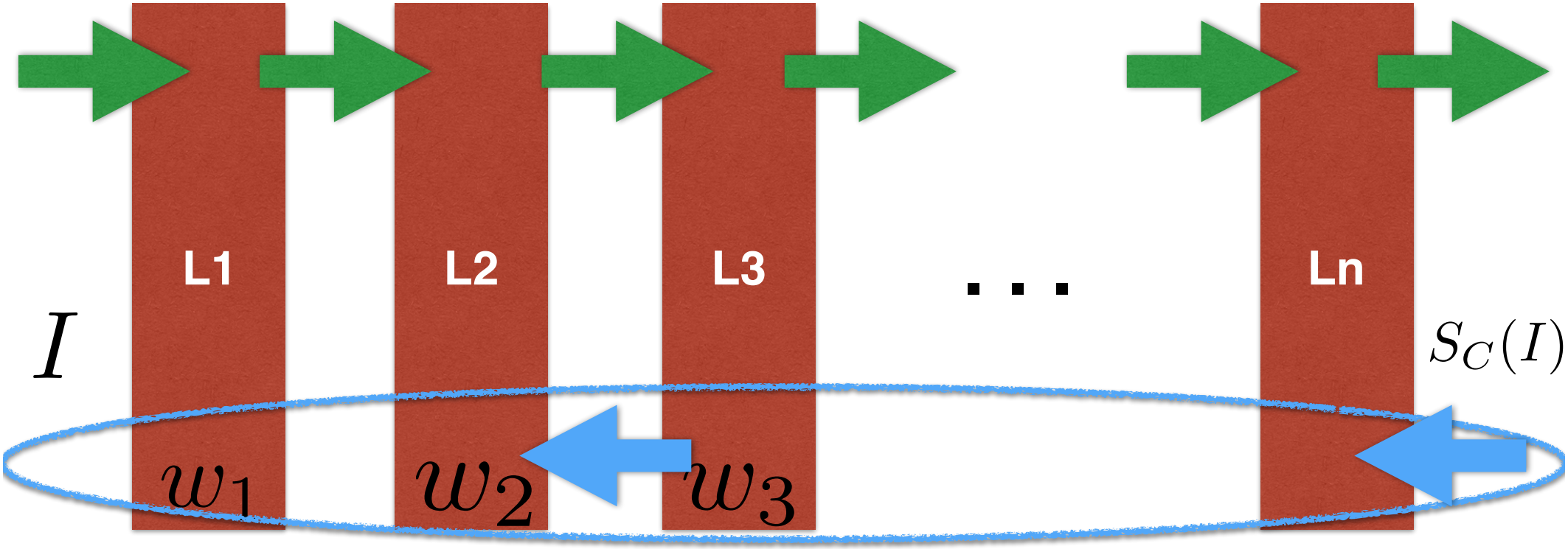
INCEPTIONISM - DEEP DREAM



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

Simonyan+14

INCEPTIONISM - DEEP DREAM

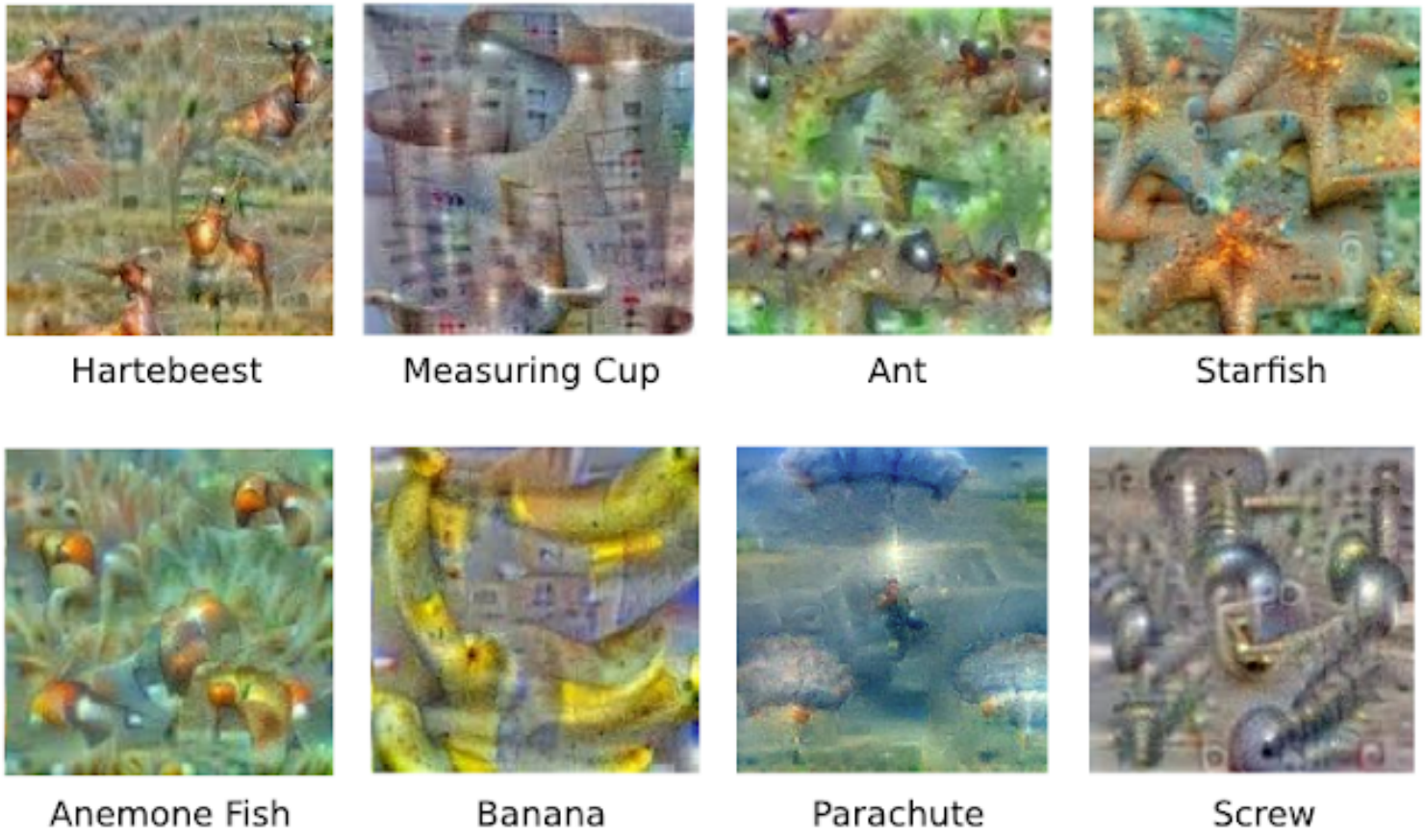


DURING THE RECONSTRUCTION PHASE, I IS LEARNT TRHOUG BACKPROPAGATION KEEPING THE WEIGHTS FIXED

Simonyan+14

INCEPTIONISM - DEEP DREAM

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



INCEPTIONISM - DEEP DREAM

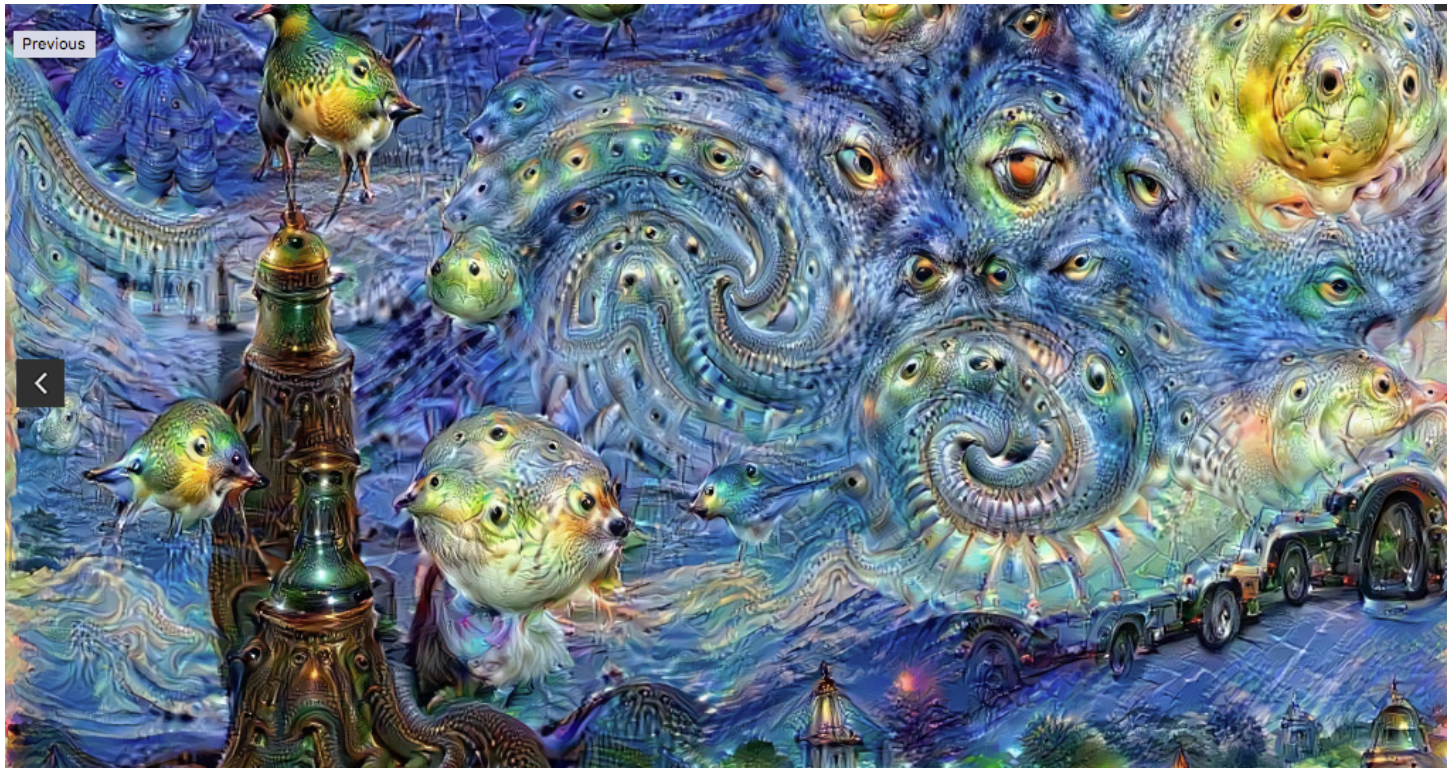
RESULTS REVEAL INTERESTING INFORMATION ON HOW THE NETWORKS BUILD REPRESENTATIONS OF OBJECTS

SOME STRANGE CASES…

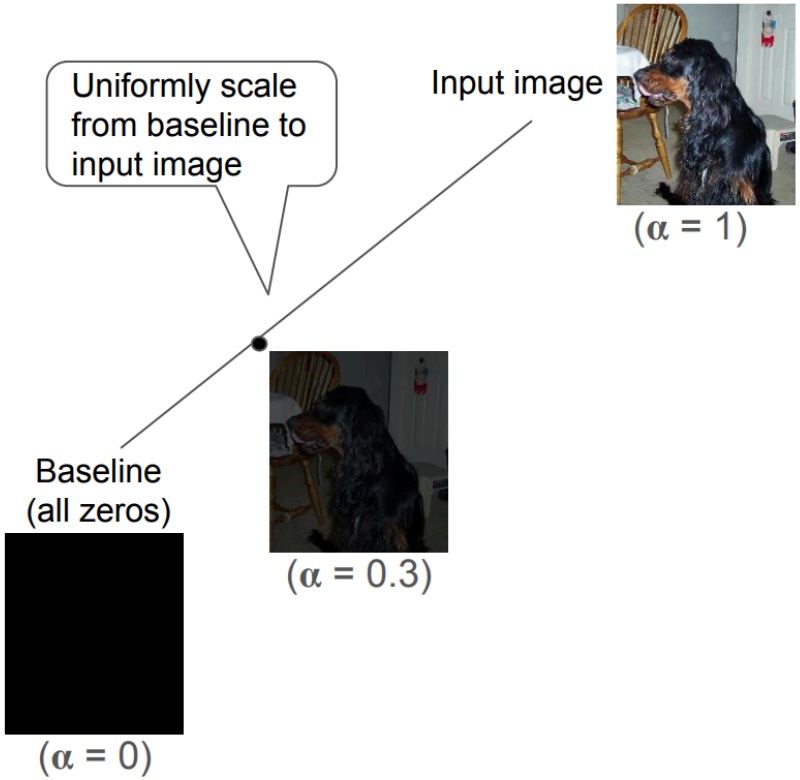
DEEP DREAM

https://deepdreamgenerator.com/

IT HAS NOW BECOME A SORT OF ART?



INTEGRATED GRADIENTS



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

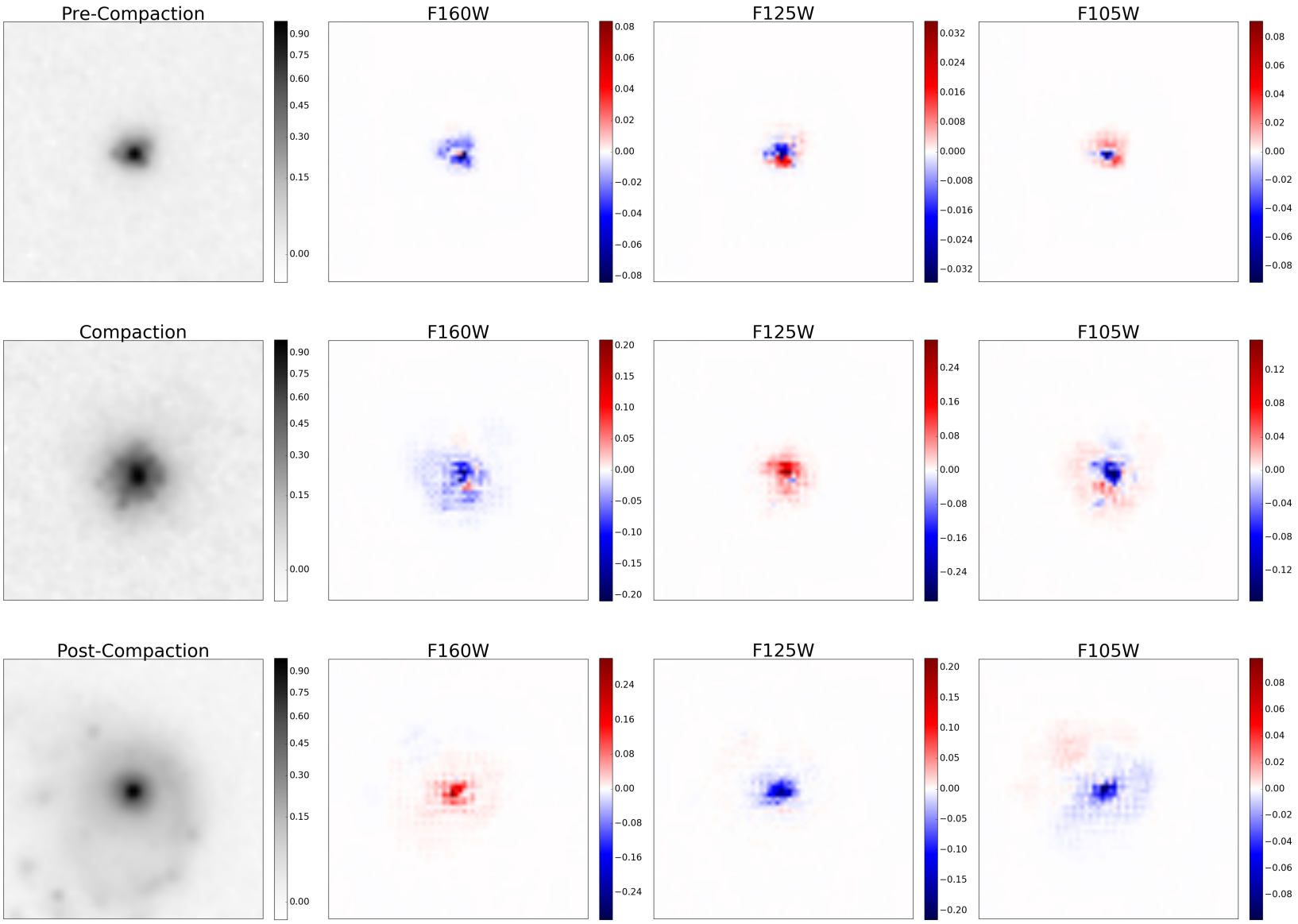
INTEGRATED GRADIENTS

Human label: accordion
Network's top label: toaster

Integrated gradient

Sundararajan+17

INTEGRATED GRADIENTS
[Applied to galaxies]



INTEGRATED GRADIENTS

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