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

ESSENTIALLY 2 APPROACHES

PERTURBATION

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

OCCLUSION SENSITIVITY

BACK-PROPAGATION

THE BASIC IDEA IS TO REVERSE THE NETWORK TO

INCEPTIONISM

ESSENTIALLY 2 APPROACHES

PERTURBATION

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

BACK-PROPAGATION

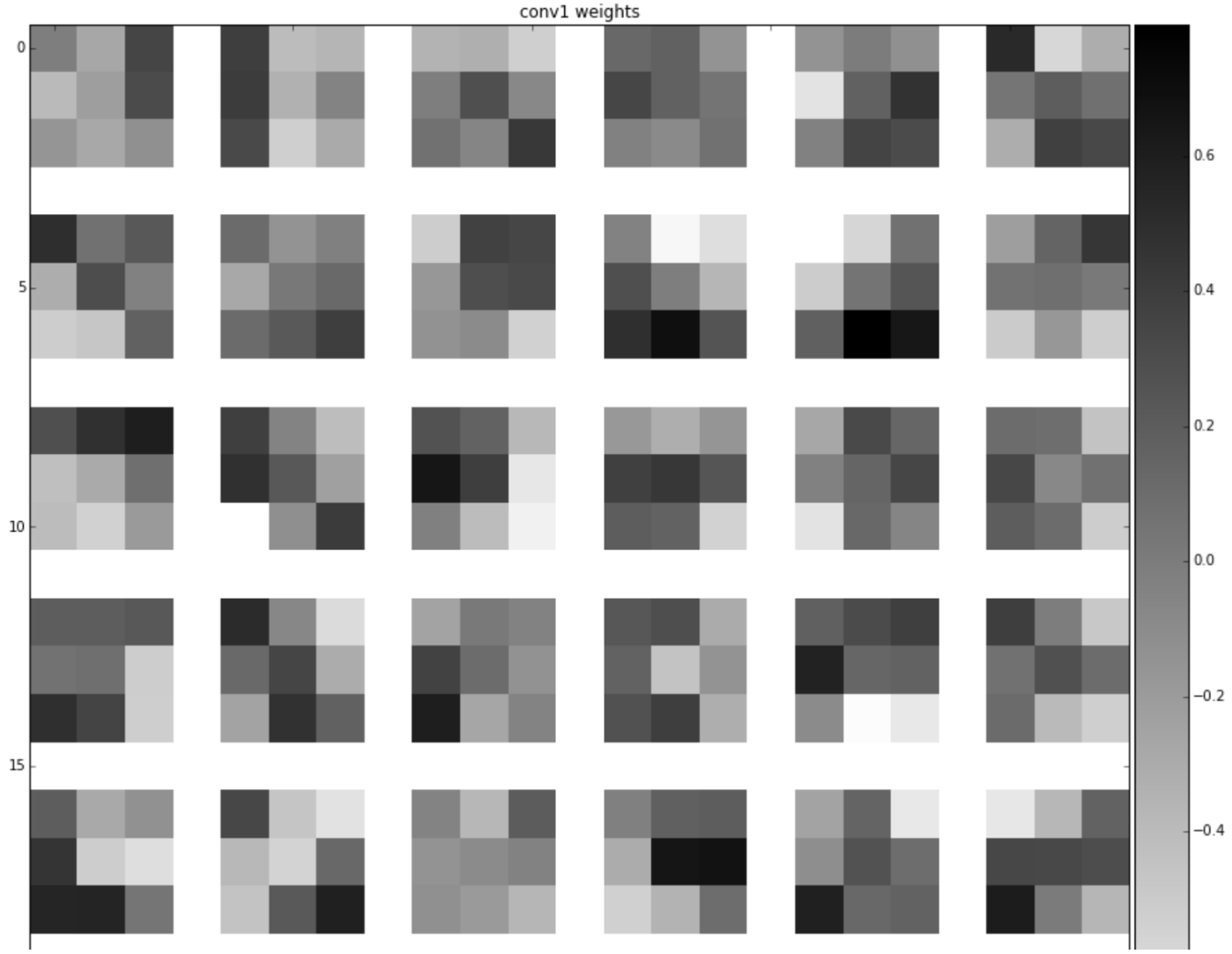
THE BASIC IDEA IS TO REVERSE THE NETWORK TO MAXIMIZE THE RESPONSE

OCCLUSION SENSITIVITY

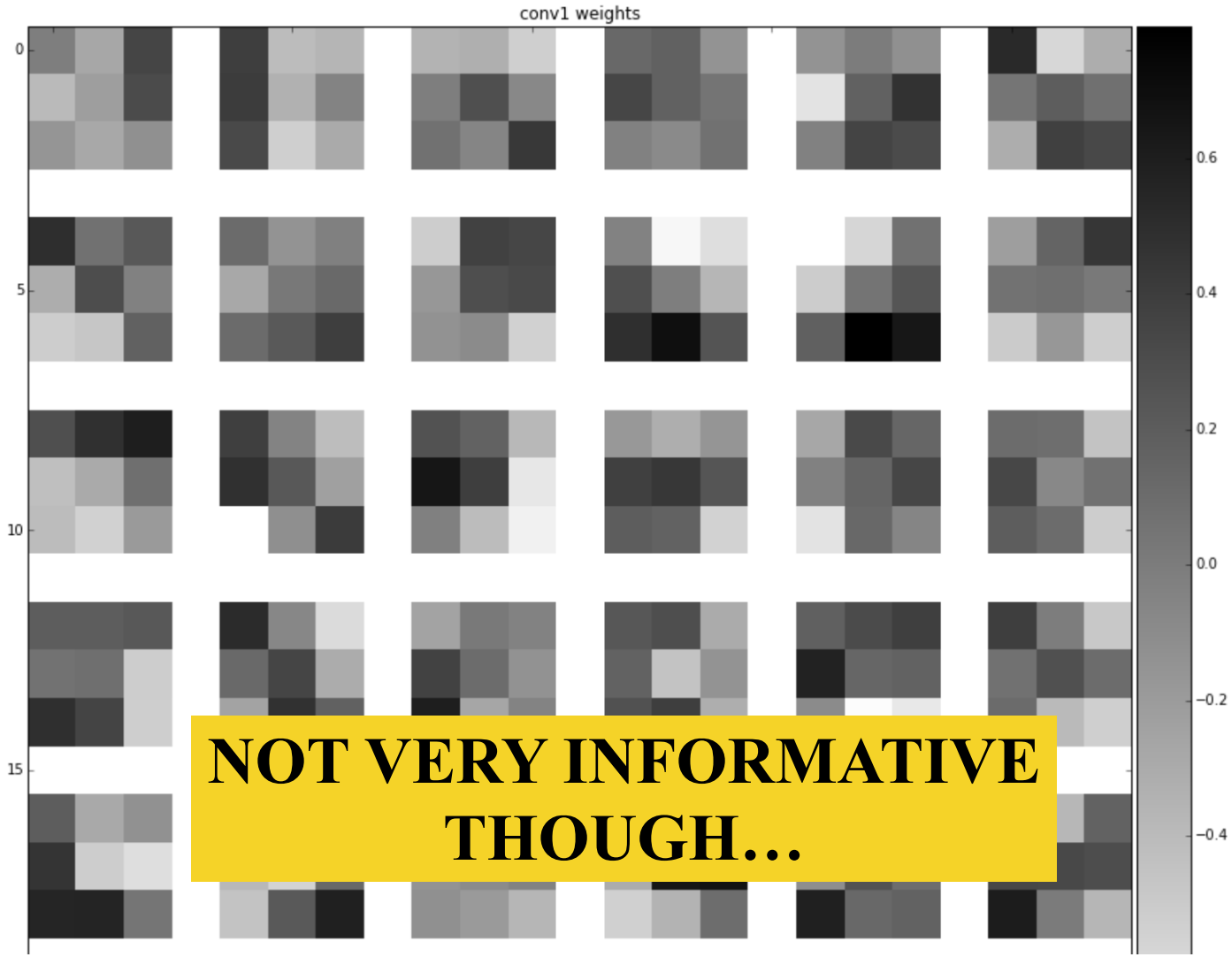
INCEPTIONISM

INTEGRATED GRADIENTS

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

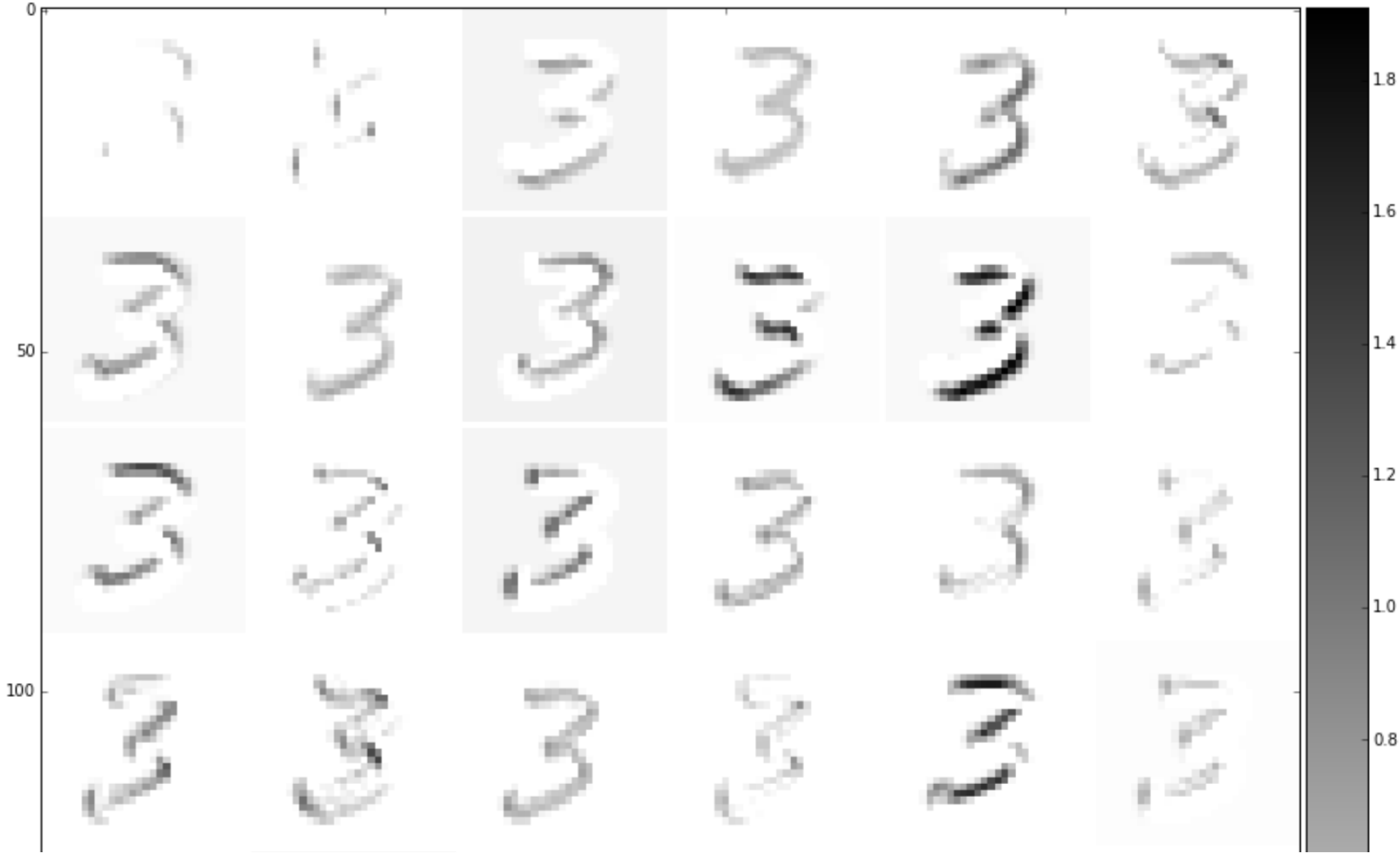
IN KERAS:

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:

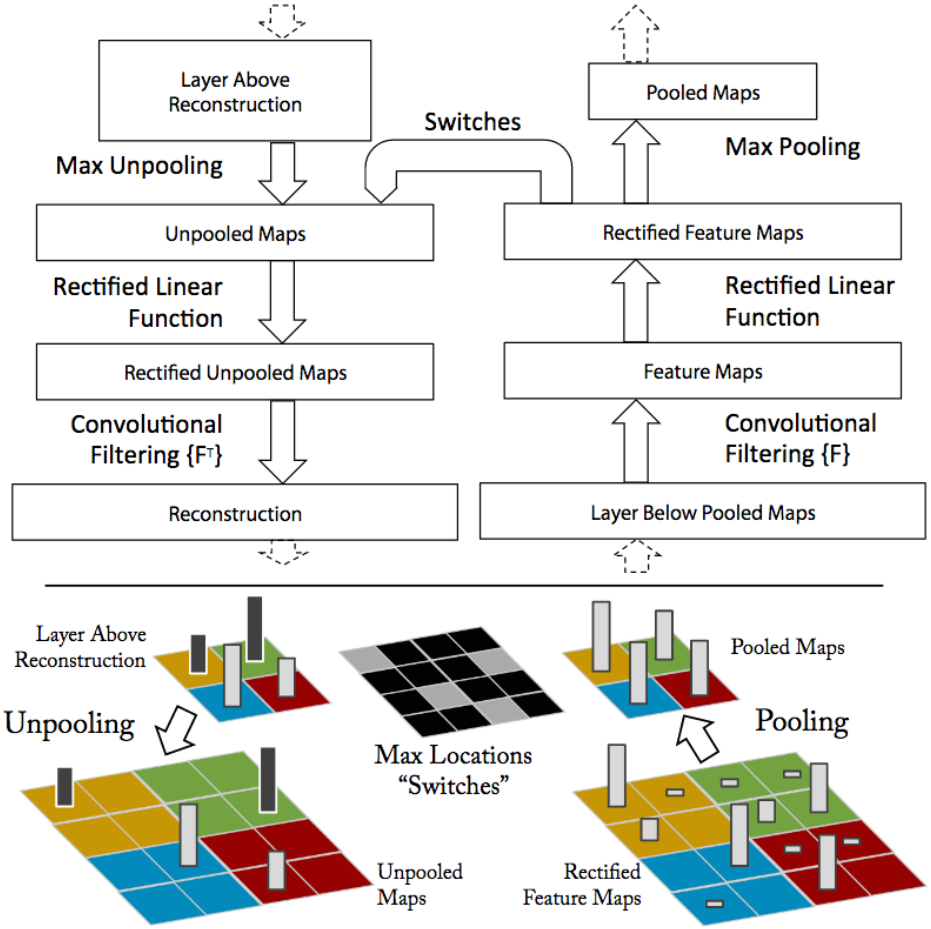
```
def display_all_layer_filters(model, layer_dict, layer_name, input_img, dir_out=None):
    """ """
    if len(input_img.shape) == 2:  # grey level image
        img = input_img.reshape(1, 1, input_img.shape[0], input_img.shape[1])
    elif len(input_img.shape) == 3:
        img = input_img.reshape(1,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

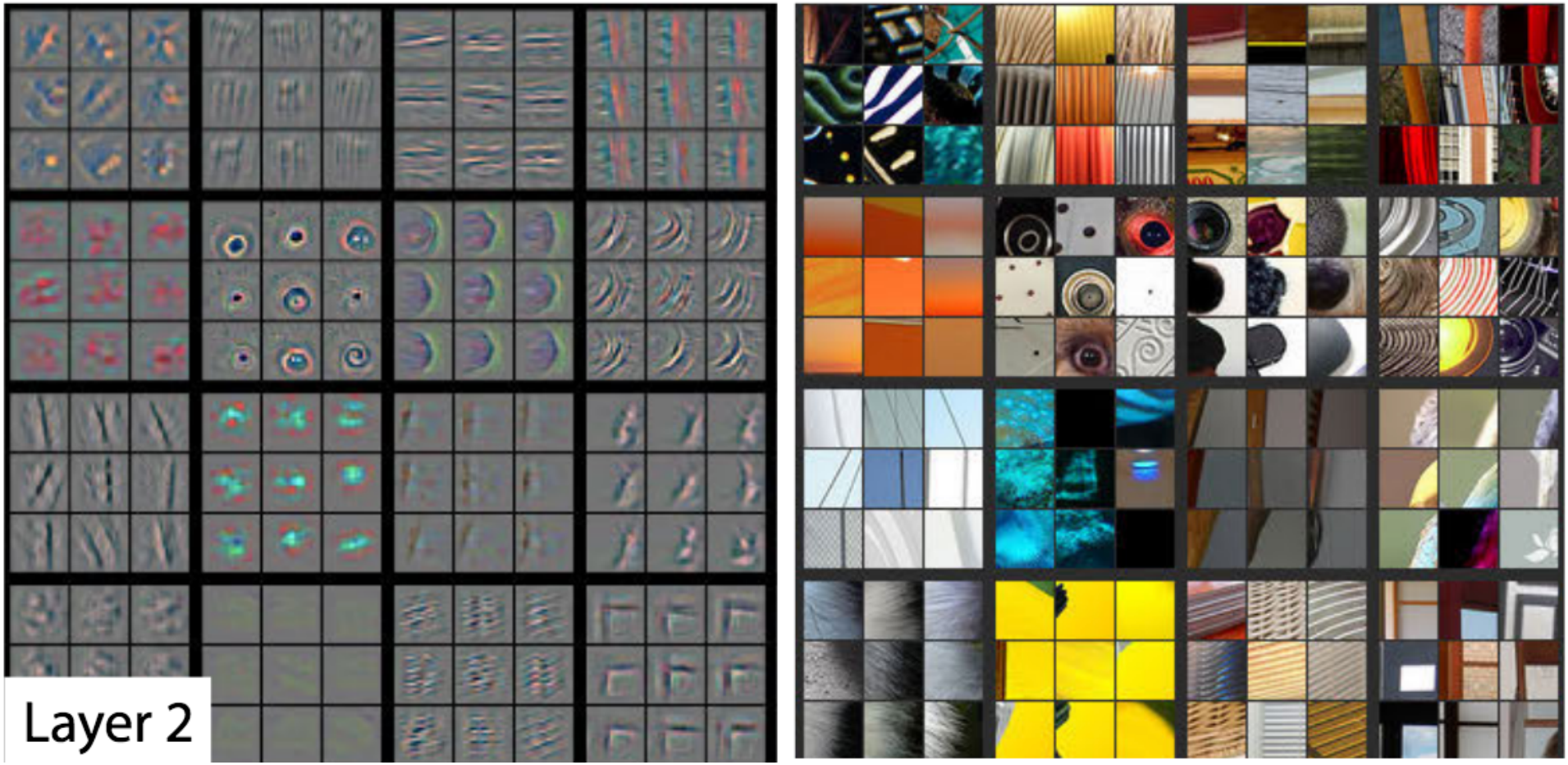
get layer output
for a series of “layer_names”

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



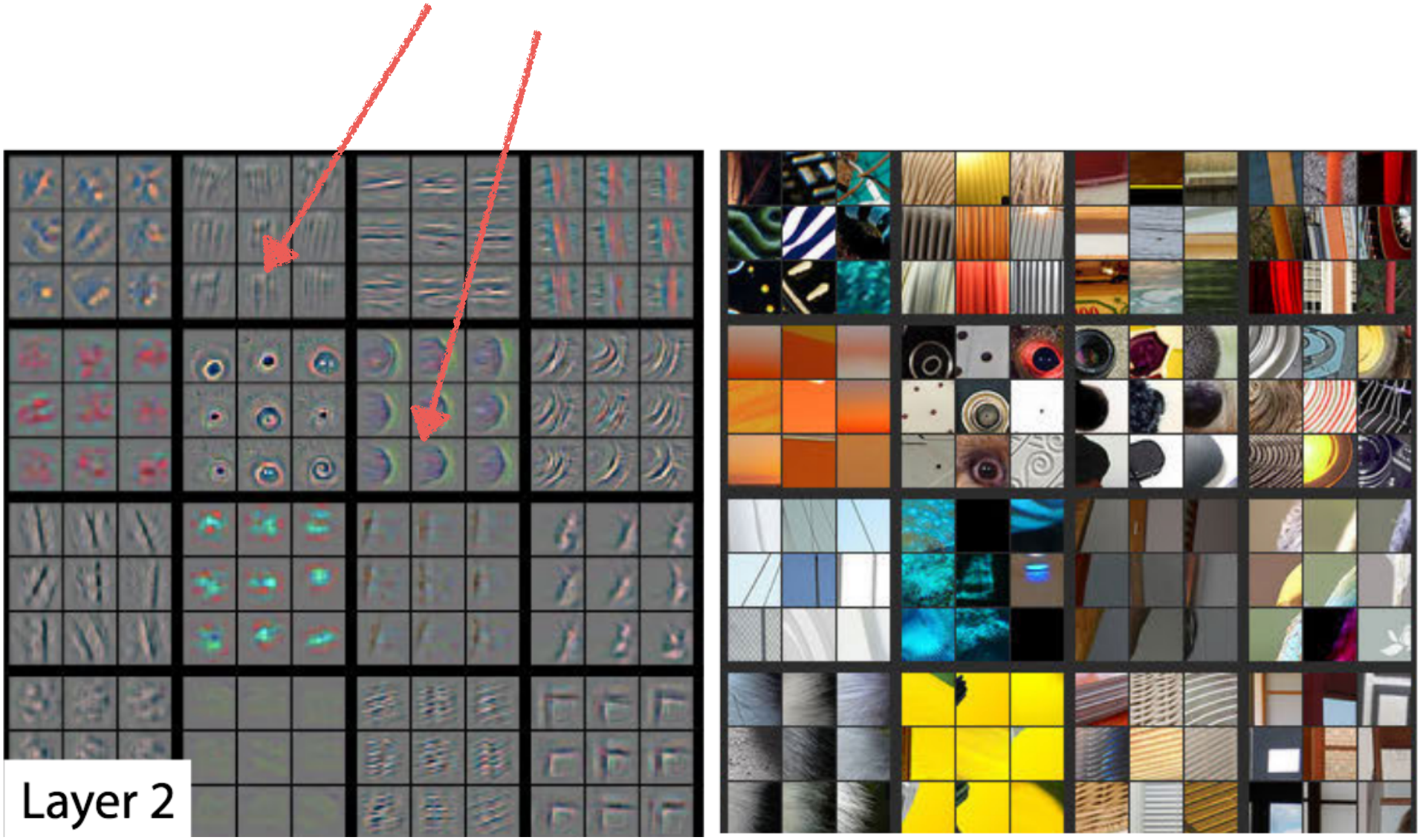
IT ALLOWS TO SEE
WHICH
REGIONS OF THE INPUT
GENERATED
A MAXIMUM RESPONSE
IN A NEURON

Zeiler+14



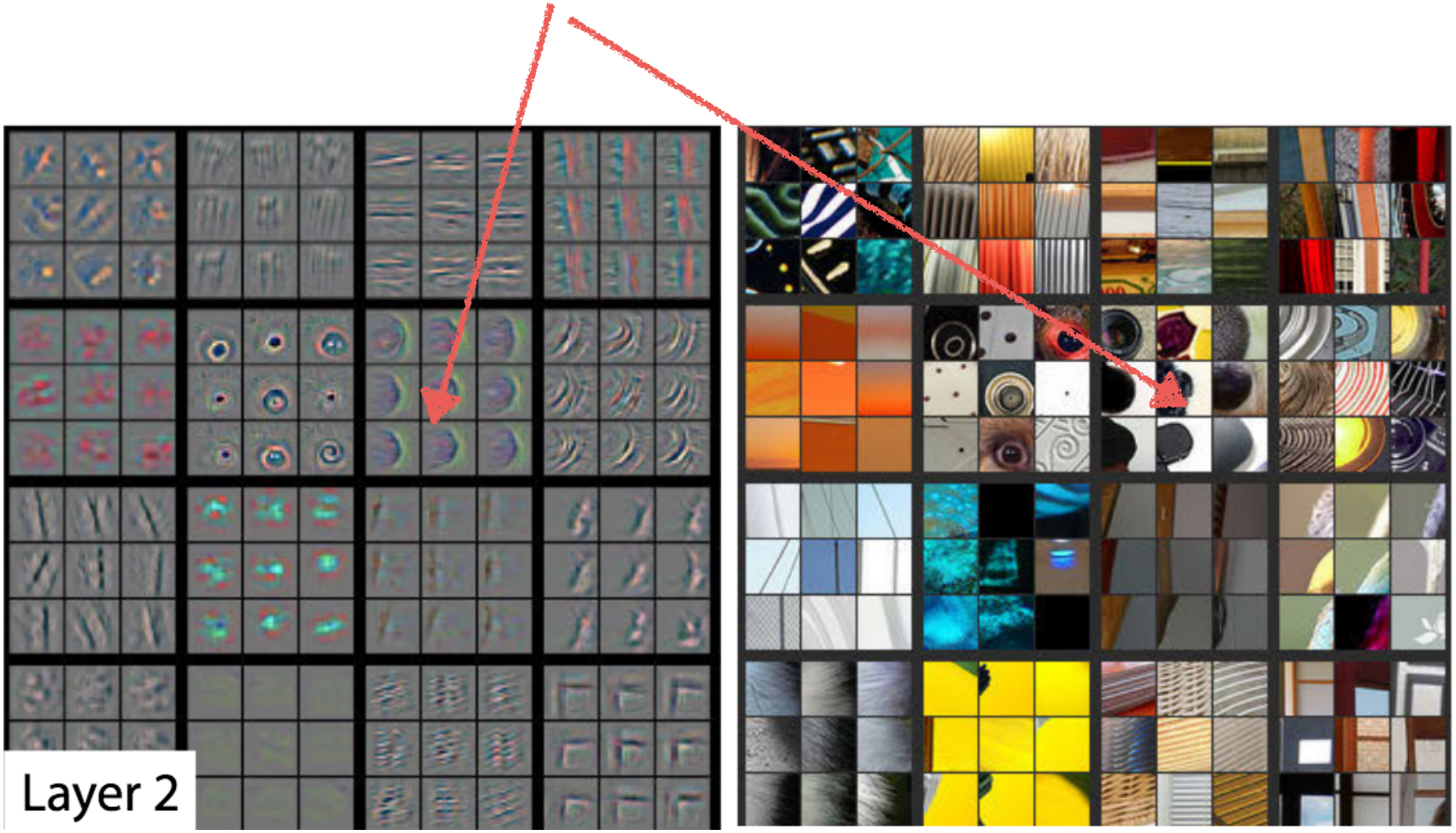
Zeiler+14

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



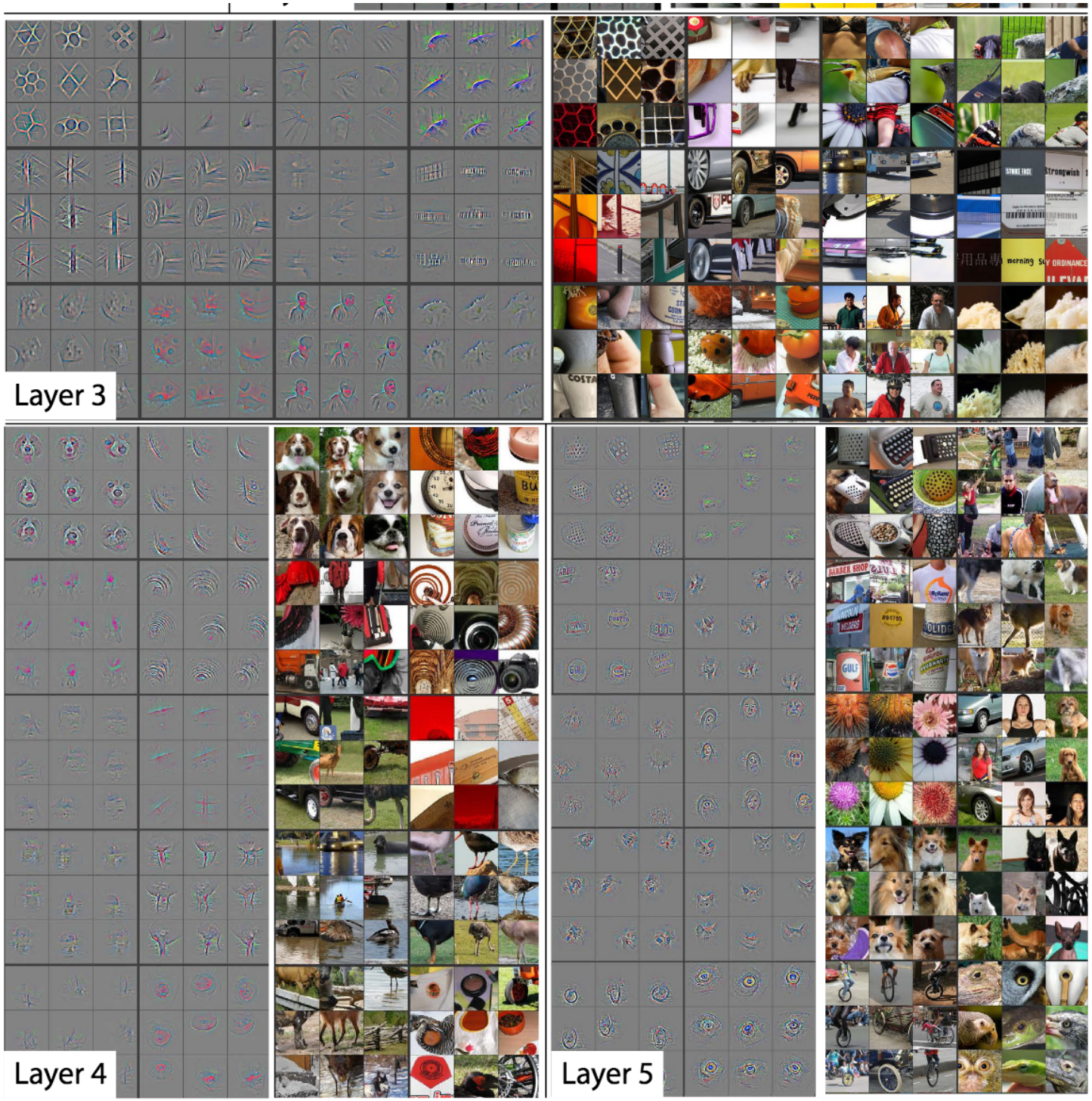
Zeiler+14

THE CORRESPONDING REGIONS OF IMAGES THAT GENERATED THE MAXIMUM RESPONSE



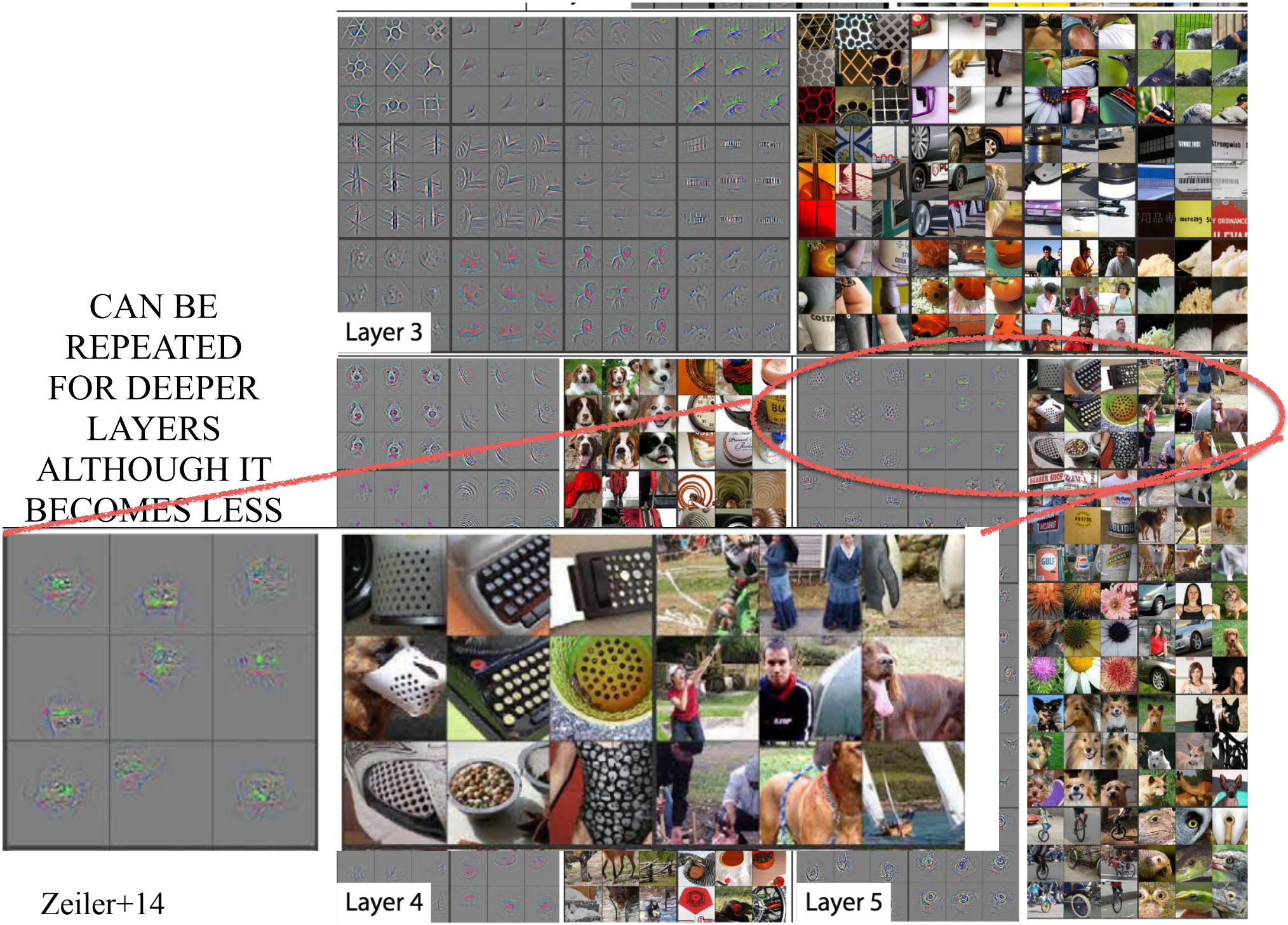
Zeiler+14

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

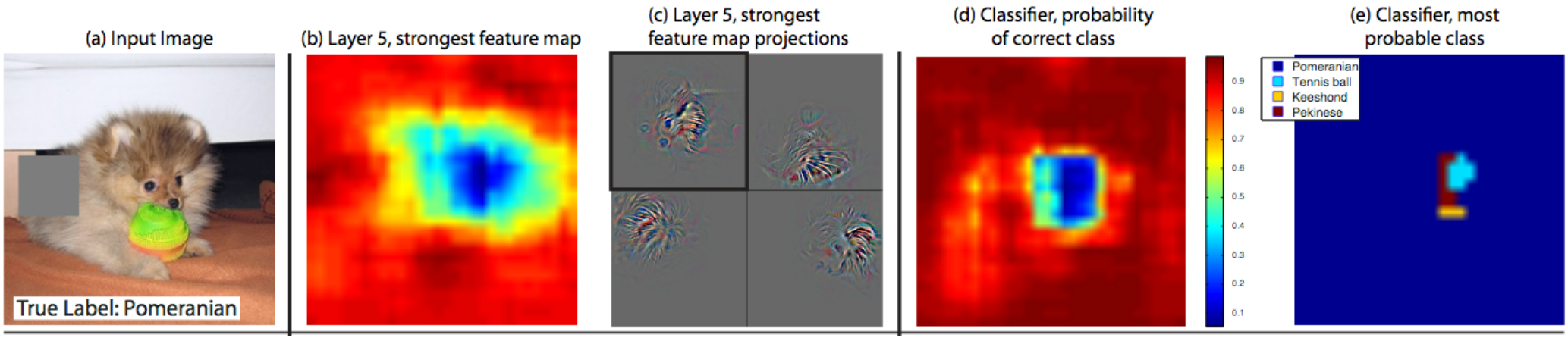
OCCLUSION SENSITIVITY

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>



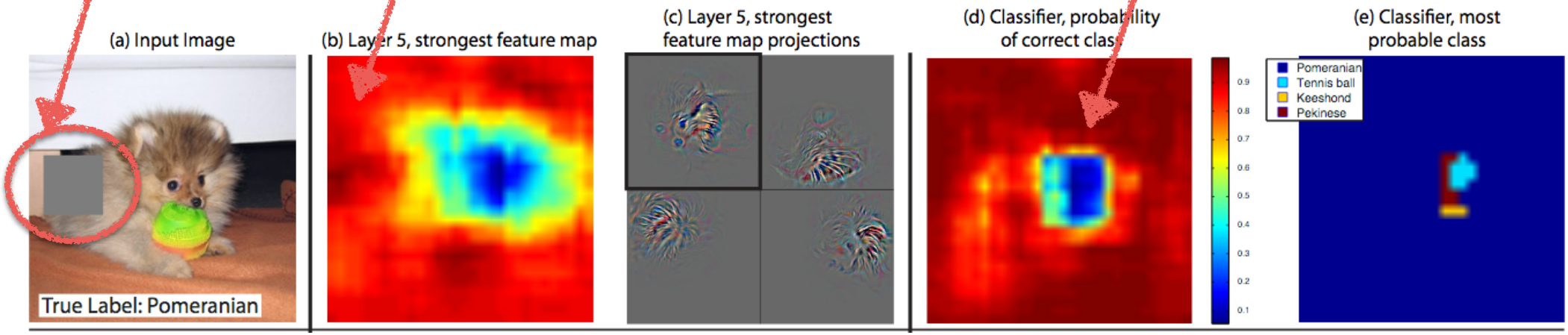
Zeiler+14

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

LIFE VISUALIZATION OF ARBITRARY NEURONS

DEEPVIS TOOLBOX

https://www.youtube.com/watch?v=AgkfIQ4IGaM

“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

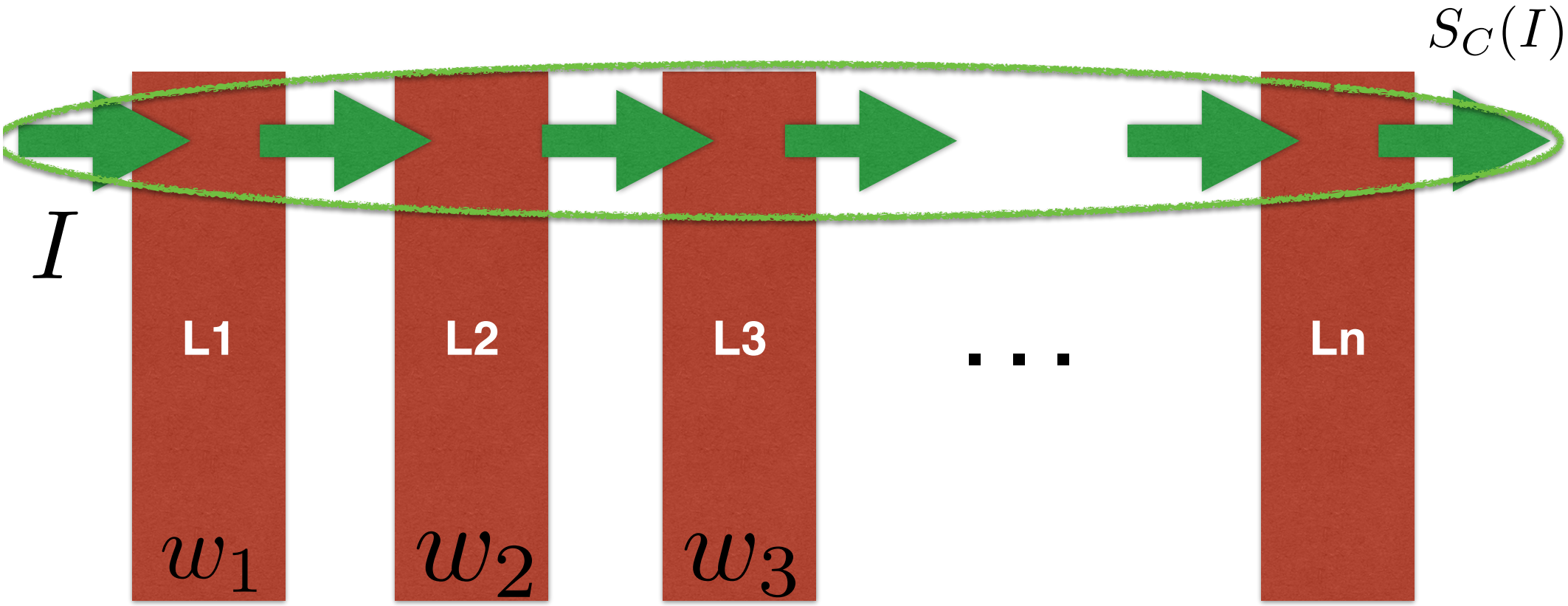
image I

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

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

Simonyan+14

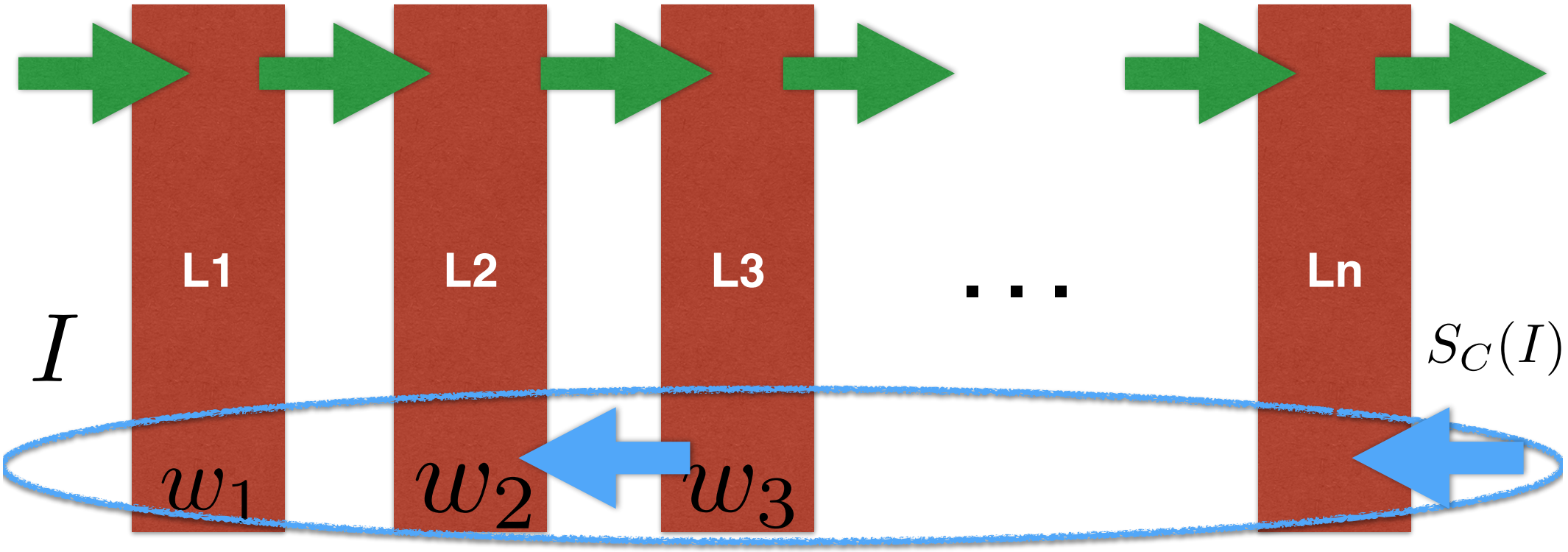
INCEPTIONISM - DEEP DREAM



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?”

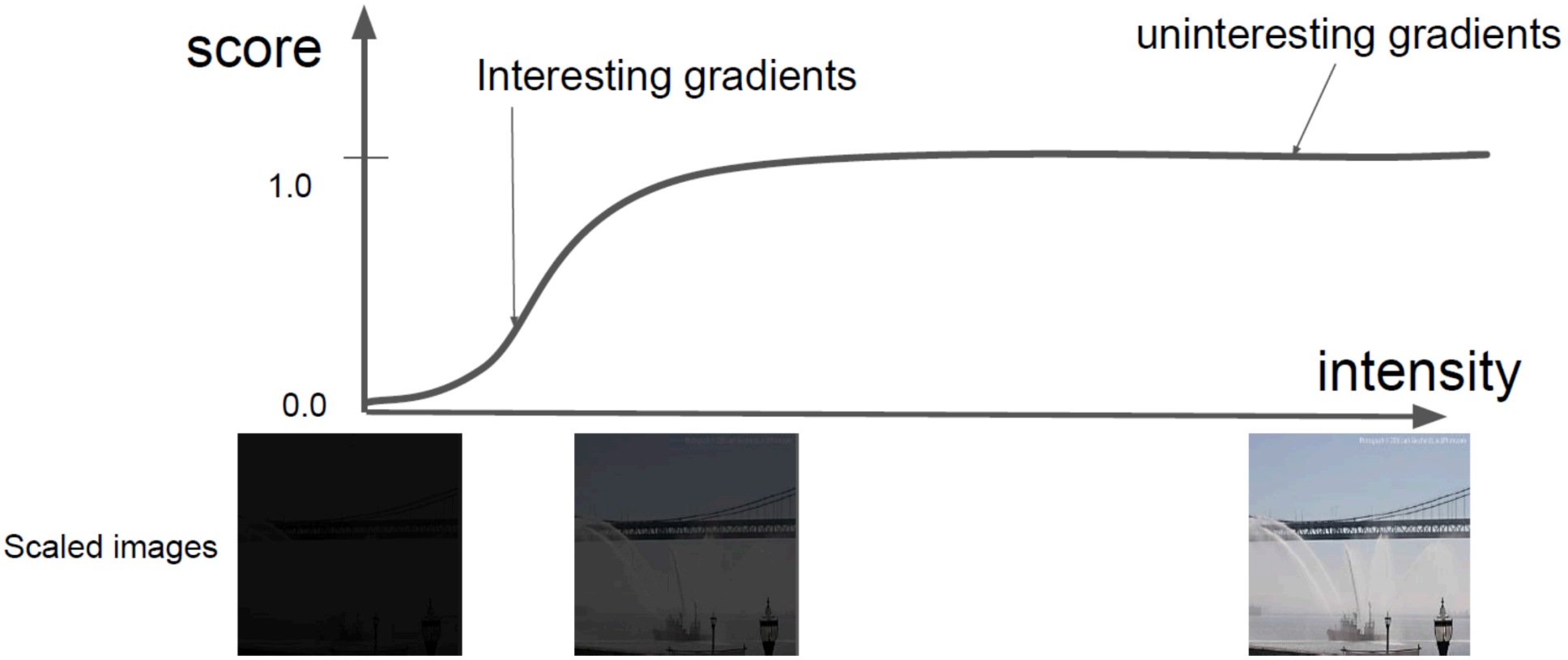
Attribute using gradient of the output w.r.t each input feature

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

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)

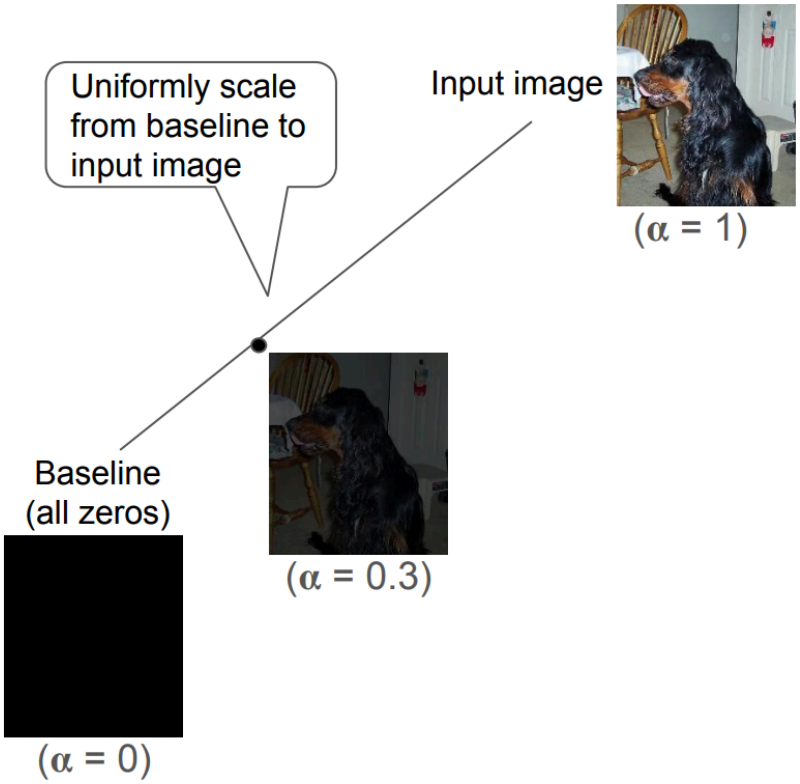
Does not work super well…

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

INTEGRATED GRADIENTS
[Applied to galaxies]