CNN for
Localization and
Weakly Supervised Localization

Giacomo Boracchi

giacomo.Boracchi@polimi.it

Artificial Neural Networks and Deep Learning

Politecnico di Milano Milano

https://boracchi.faculty.polimi.it/

Data Pre-processing and Batch Normalization

Preprocessing

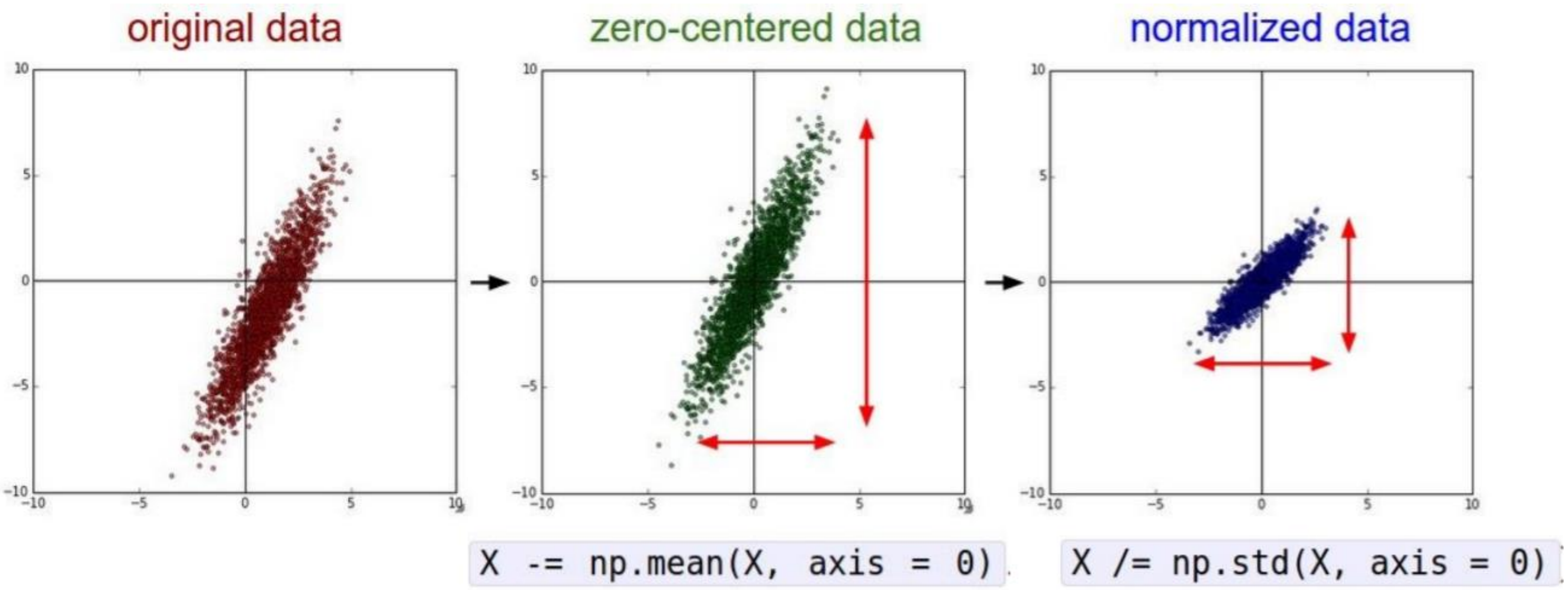
In general, normalization is useful in gradient-based optimizers.

Normalization is meant to **bring training data "around the origin"** and possibly further rescale the data

In practice, **optimization on pre-processed data is made easier** and results are less sensitive to perturbations in the parameters

There are several options

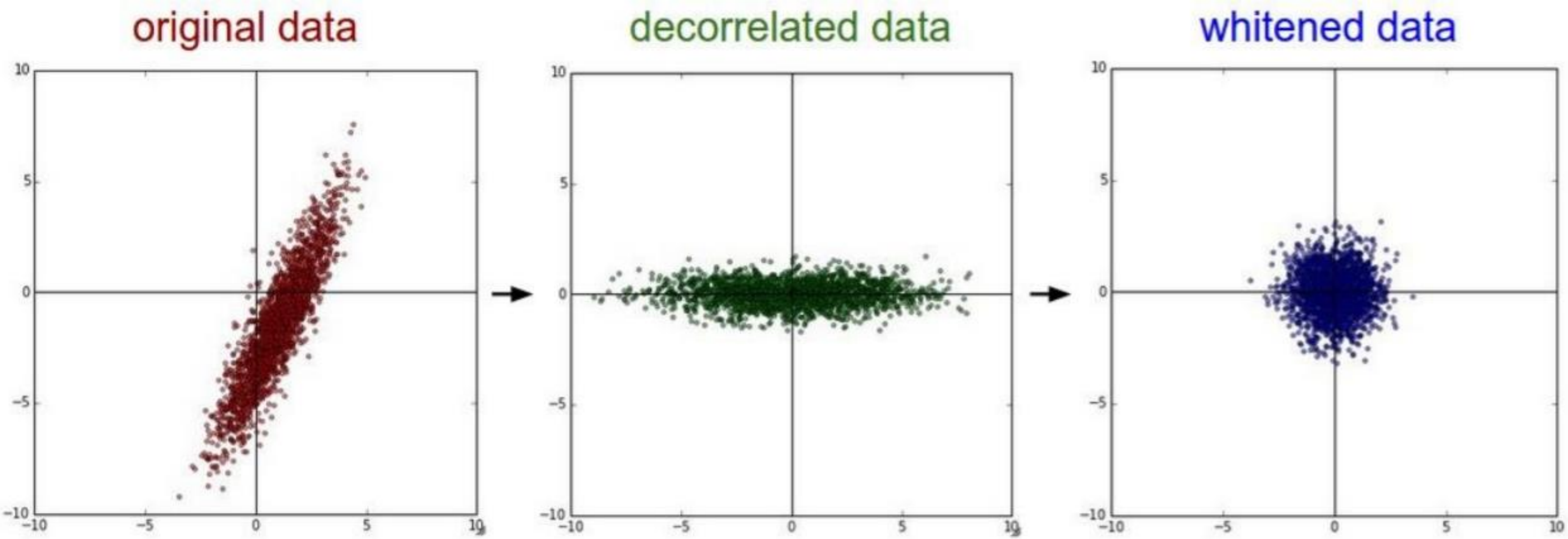
There are different form of preprocessing



[CS231n: Convolutional Neural Networks for Visual Recognition](http://cs231n.github.io/) http://cs231n.github.io/

PCA – based preprocessing

This is performed after having «zero-centered» the data



diagonal covariance matrix

covariance = identity

[CS231n: Convolutional Neural Networks for Visual Recognition](http://cs231n.github.io/) http://cs231n.github.io/

Preprocessing for CNNs: mean subtraction

PCA/Whitening preprocessing are not commonly used with CNN

The most frequent option is to zero-center the data, and it is common to normalize every pixel as well

Consider CIFAR-10 example with [32,32,3] images

- **AlexNet**: Subtract the mean image (*mean image = [32,32,3] array*)
- **VGG**: Subtract per-channel mean (*mean along each channel = 3 numbers*)
- **ResNet**: Subtract per-channel mean and Divide by per-channel std (*mean and std along each channel = 3 + 3 numbers*)

Preprocessing for CNN

Preprocessing and Training:

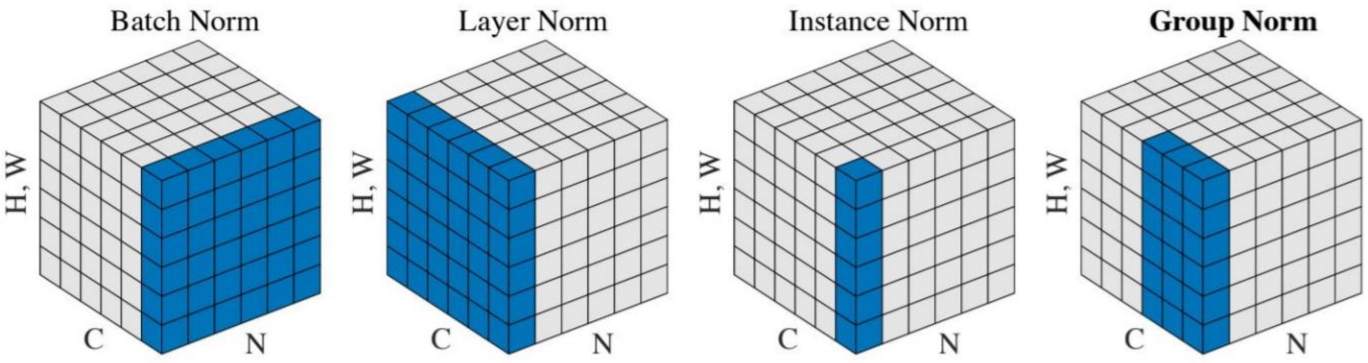
- **Normalization statistics are parameters of your ML model**: Any preprocessing statistics (e.g. the data mean) must be computed on training data, and applied to the validation / test data.
    - Do not normalize first and then split in training, validation, test
- When using pretrained model, remember to import (and use!) their **pre-processing function**.

Batch Normalization

Consider a batch of activations $\{x_i\}$, the following transformation bring these to unit variance and zero mean

$$x_i' = \frac{x_i \; - E[x_i]}{\sqrt{\mathrm{var}[x_i]}}$$

Where $E[x_i]$ and $\sqrt{\mathrm{var}[x_i]}$ are computed from each batch and separately for each channel!



Wu and He, "Group Normalization", ECCV 2018

Can we get more flexibility than zero-mean, unit variance?

Ioffe, S. and Szegedy, C., 2015, June. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning* (pp. 448-456). PMLR.

Batch Normalization

Batch normalization adds after standard normalization

$$x_i' = \frac{x_i - E[x_i]}{\sqrt{\mathrm{var}[x_i]}}$$

a further a parametric transformation

$$y_{i,j} = \gamma_j x_i' + \beta_j$$

Where parameters γ and β are learnable scale and shift parameters.

- We have γ and β for each channel of the input activation.
- The expected value and variance are non trainable parameters.

Rmk: estimates $E[x_i]$ and $\sqrt{\mathrm{var}[x_i]}$ are computed on each minibatch, need to be fixed after training. **After training, these are replaced by (running) averages of values seen during training**.

Batch Normalization

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

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

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

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

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

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

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Ioffe, S. and Szegedy, C., 2015, June. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In International conference on machine learning (pp. 448-456). PMLR.

Batch Normalization

During testing batch normalization becomes a linear operator! Can be fused with the previous fully-connected or conv layer.

In practice networks that use Batch Normalization are significantly more robust to bad initialization.

Typically Batch Normalization is used in between FC layers of deep CNN, but sometimes also between Conv Layers.

Ioffe, S. and Szegedy, C., 2015, June. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning* (pp. 448-456). PMLR.

Batch Normalization

Pros:

- Makes deep networks much easier to train!
- Improves gradient flow
- Allows higher learning rates, faster convergence
- Networks become more robust to initialization
- Acts as regularization during training
- Zero overhead at test-time: can be fused with conv!

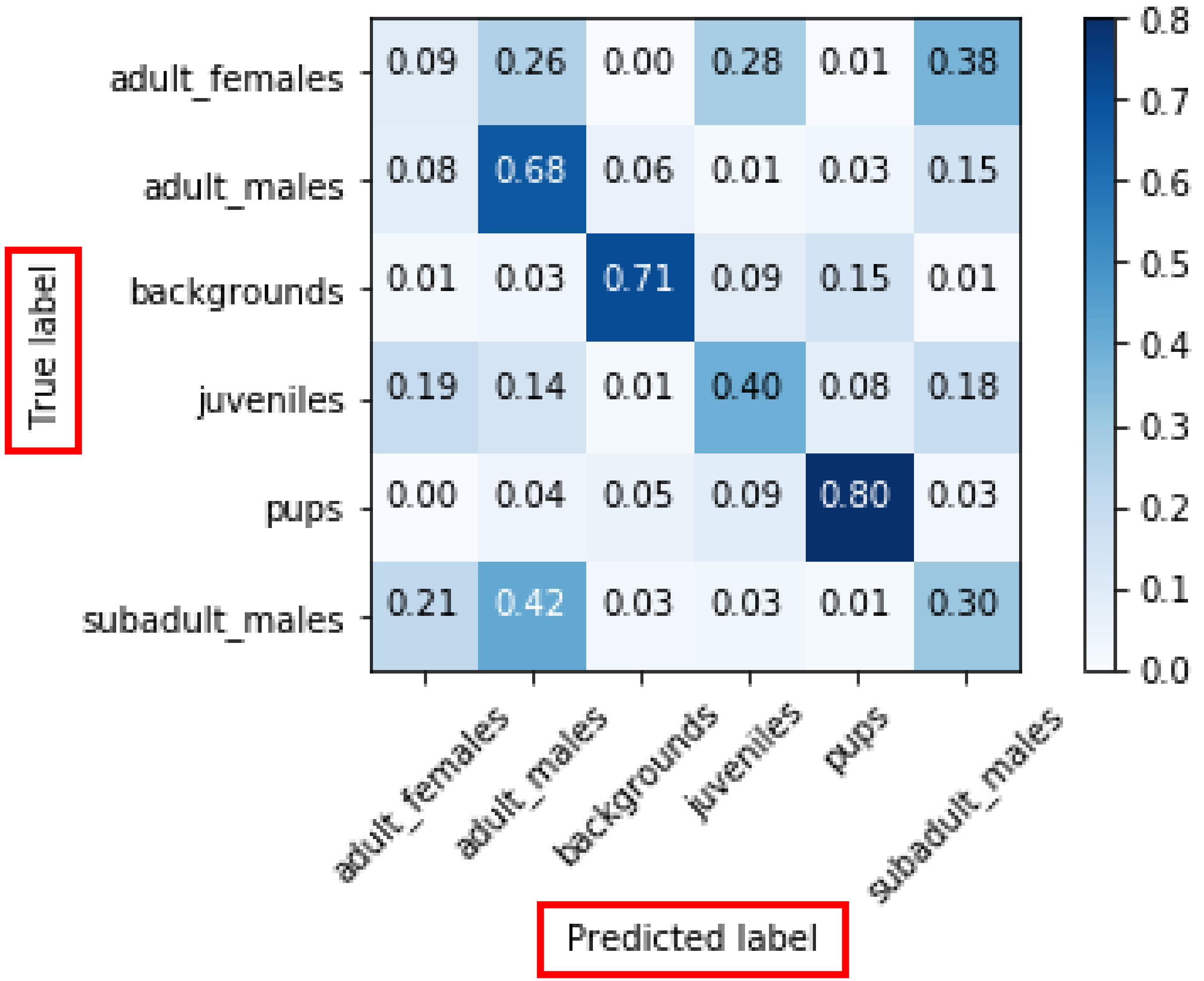
Watch out:

- Behaves differently during training and testing: this is a very common source of bugs!

A bit more of background

Performance measures

and an overview of successful architectures



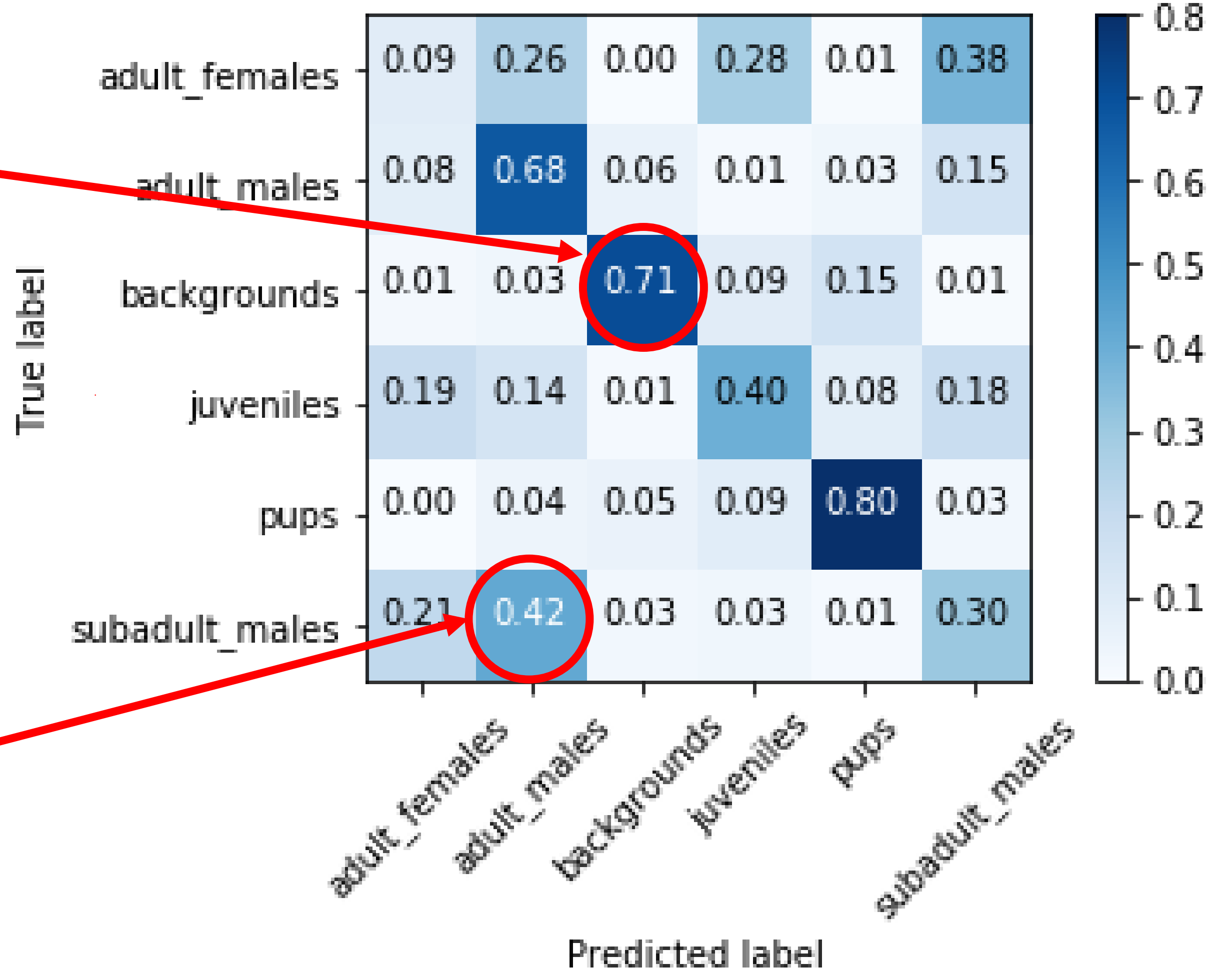
True label
adult_females
adult_males
backgrounds
juveniles
pups
subadult_males
0.09 0.26 0.00 0.28 0.01 0.38
0.08 0.68 0.06 0.01 0.03 0.15
0.01 0.03 0.71 0.09 0.15 0.01
0.19 0.14 0.01 0.40 0.08 0.18
0.00 0.04 0.05 0.09 0.80 0.03
0.21 0.42 0.03 0.03 0.01 0.30
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
adult_females
adult_males
backgrounds
juveniles
pups
subadult_males
Predicted label

Confusion Matrix

The element $C(i,j)$ i.e. at the i-th row and j-th column corresponds to the percentage of elements belonging to class i classified as elements of class j

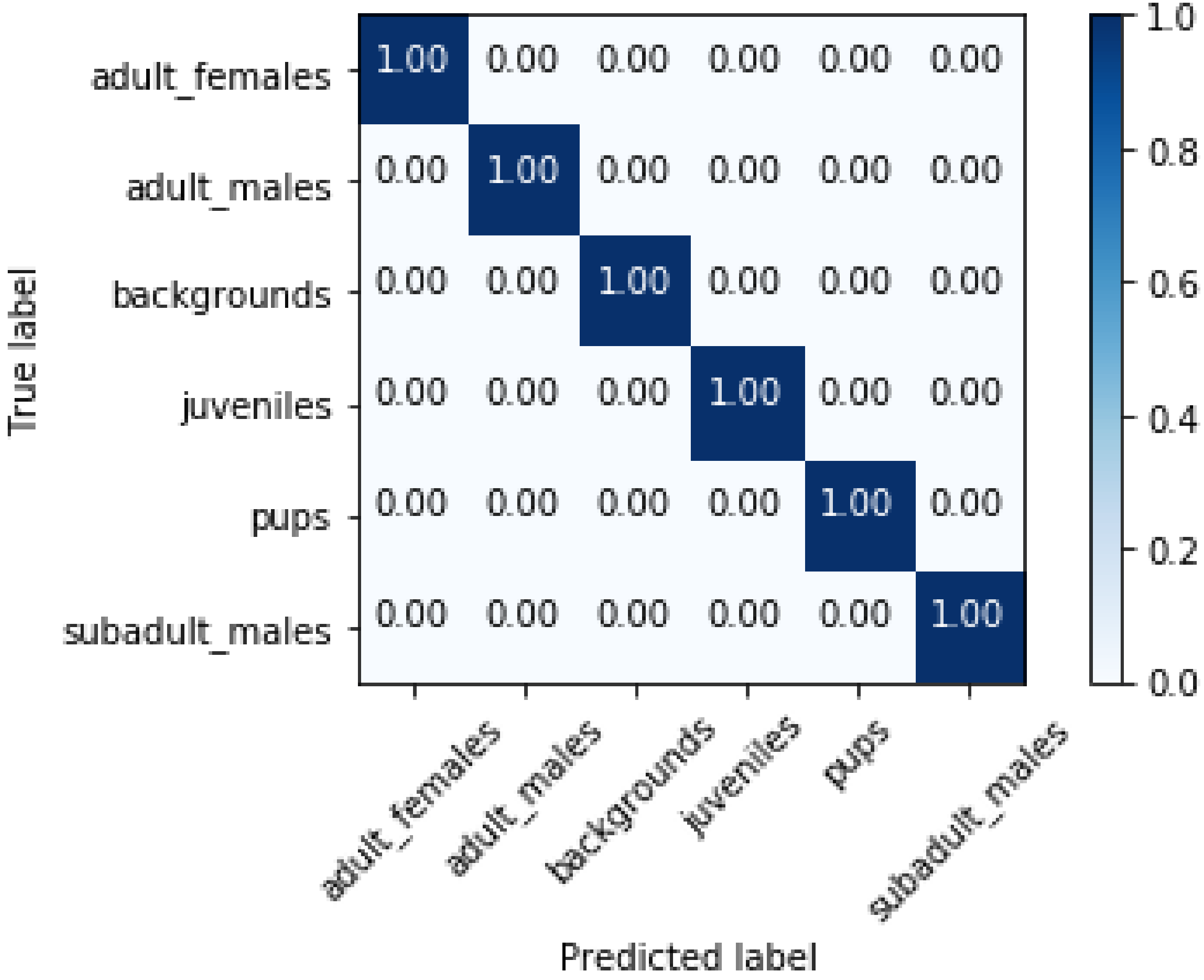
71% of background patches have been correctly classified as background

42% of sub-adult males patches have been wrongly classified as adult-males



… so, the ideal confusion matrix

Which rarely happens



Two-Class Classification

Background:

In a two-class classification problem (binary classification), the **CNN output is equivalent to a scalar,** since

$$CNN(I) = [p, 1 - p]$$

being p the probability of I to belong to the first class.

Thus we can write

$$CNN(I) = p$$

Then, we can decide that I belongs to the first class when

$$CNN(I) > \Gamma$$

and use Γ different from 0.5, which is the standard.

We require stronger evidence before claiming I **belongs to class 1.**

Changing Γ **establishes a trade off between FPR and TPR.**

Two-Class Classification

Classification performance in case of **binary classifiers** can be also measured in terms of the **ROC** (receiver operating characteristic) **curve,** which does not depend on the threshold you set for each class

This is useful in case you plan to modify this and not use 0.5

(FPR, TPR) for a specific parameter

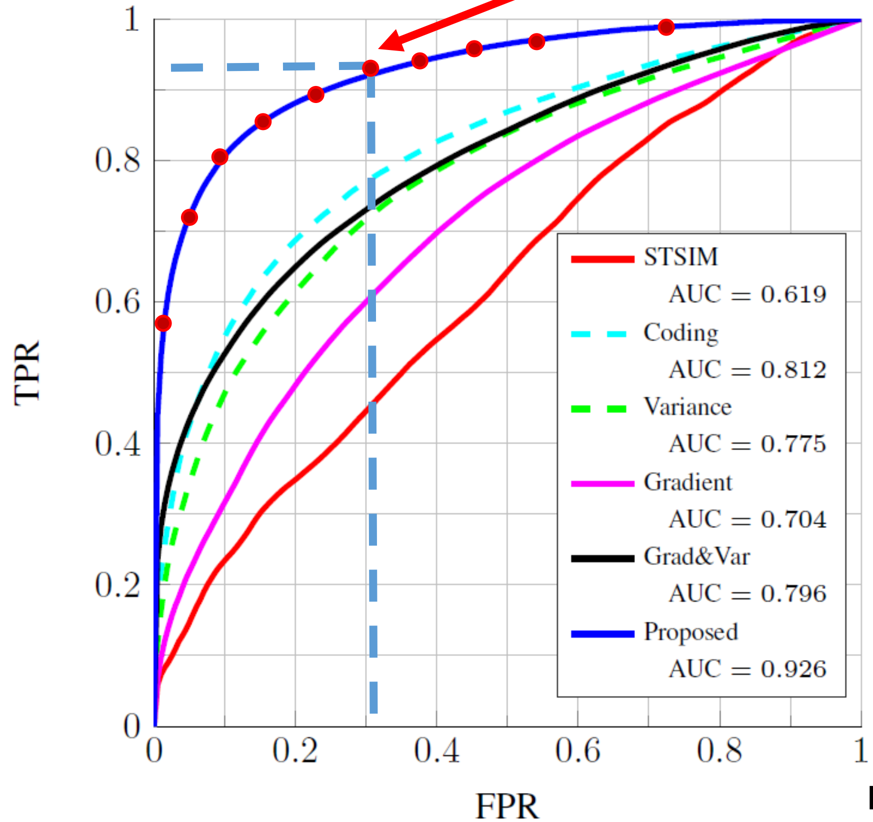
The ideal detector would achieve:

- $FPR = 0\%$,
- $TPR = 100\%$

Thus, the closer to (0,1) the better

The largest the **Area Under the Curve** (AUC), the better

The optimal parameter is the one yielding the point closest to (0,1)



Localization and CNN Explanations

Giacomo Boracchi

giacomo.boracchi@polimi.it

Localization

The Classification Task

The input image contains a single relevant object to be classified in a fixed set of categories

The task is to:

1) assign the object class to the image

hawk

The Classification and Localization Tasks

The input image contains a single relevant object to be classified in a fixed set of categories

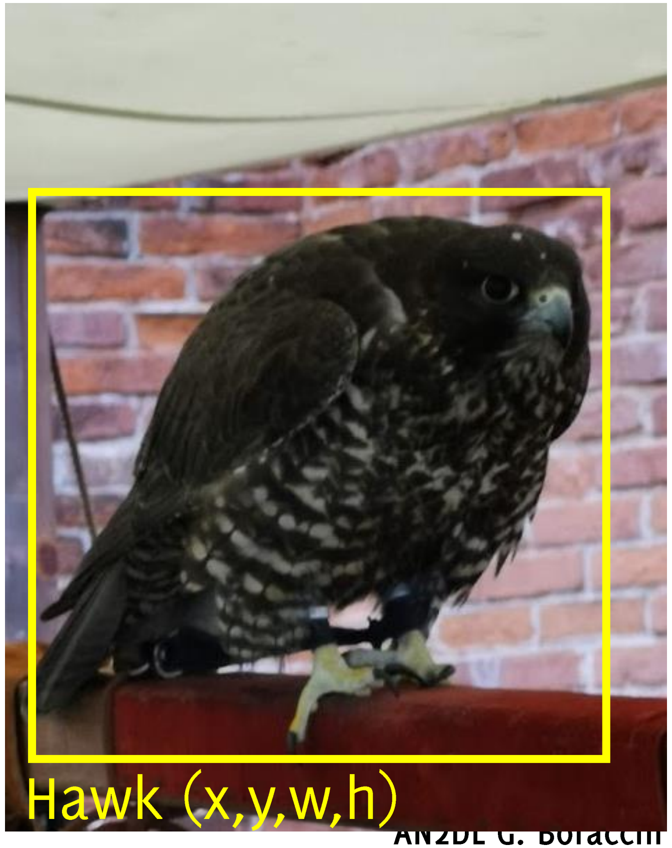
The tasks are:

1) assign the object class to the image
2) locate the object in the image by its bounding box

A training set of annotated images with **label and a bounding box** around each object is required

Extended localization problems involve regression over more complicated geometries (e.g. human skeleton)

hawk



The Classification and Localization Tasks

Class.

categoria

The input image contains a single relevant object to be classified in a fixed set of categories

The tasks are:

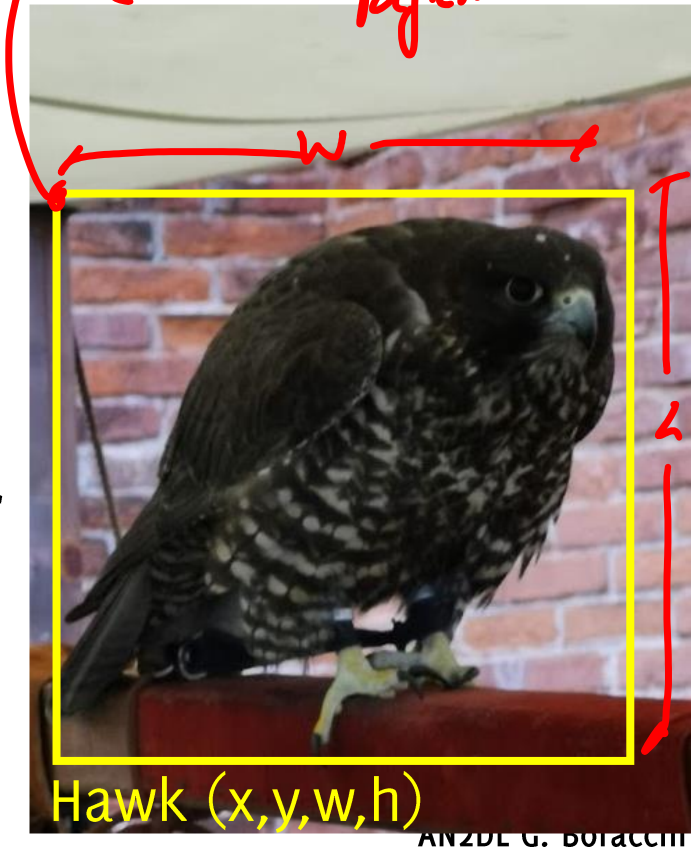
1) assign the object class to the image
2) locate the object in the image by its bounding box

(x,y)

(x,y,w,h)

numerica

hawk



A training set of annotated images with **label and a bounding box** around each object is required

Extended localization problems involve regression over more complicated geometries (e.g. human skeleton)

Bounding Box Estimation, the problem

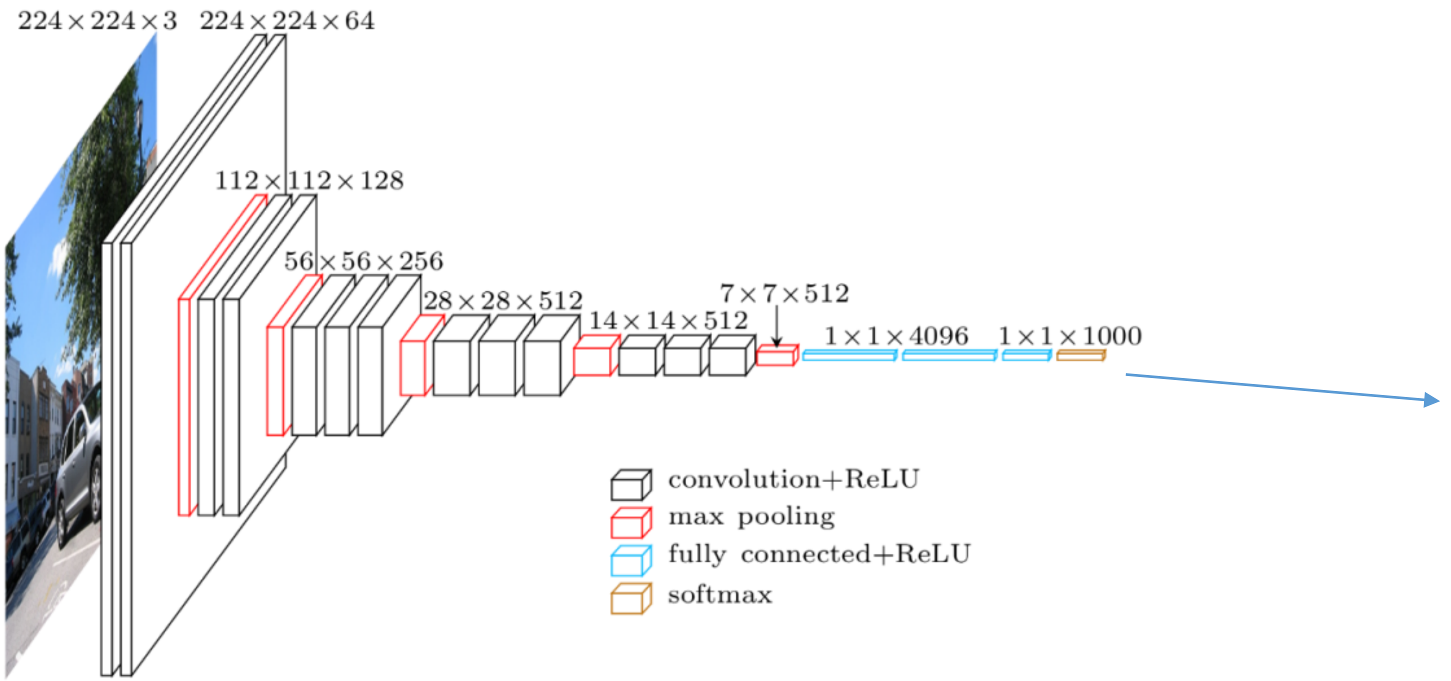
Assign to an input image $I \in \mathbb{R}^{R \times C \times 3}$:

- the coordinates (x, y, h, w) of the bounding box enclosing the object

$$I \to (x, y, h, w)$$

The Simplest Solution

Train a regression network to predict the bounding box



Bounding Box Coordinates

(x, y, w, h)

Regression loss $\mathcal{R}$, e.g. the $\ell^2, \ell^1, ..$ bounding box.

$$\left\| [\hat{x}, \hat{y}, \hat{w}, \hat{h}] - [x, y, w, h] \right\|_2^2$$

How to?

```
# Prepare the output layer, 4 real numbers (bounding box coordinates) and linear activations
output = tfkl.Dense(4, activation='linear', name='regressor')(x)

# Connect input and output through the Model class
regressor_model = tfk.Model(inputs=inputs, outputs=output, name='regressor_model')

# Compile the model using Mean Squared Error (MSE) as loss
regressor_model.compile(loss=tfk.losses.MeanSquaredError(), optimizer=tfk.optimizers.Adam())
```

Classification and Localization, the problem

Assign to an input image $I \in \mathbb{R}^{R \times C \times 3}$:

- a label l from a fixed set of categories

$$\Lambda = \{\text{"wheel"}, \text{"cars"}, \ldots, \text{"castle"}, \text{"baboon"}\}$$

- the coordinates (x, y, h, w) of the bounding box enclosing that object

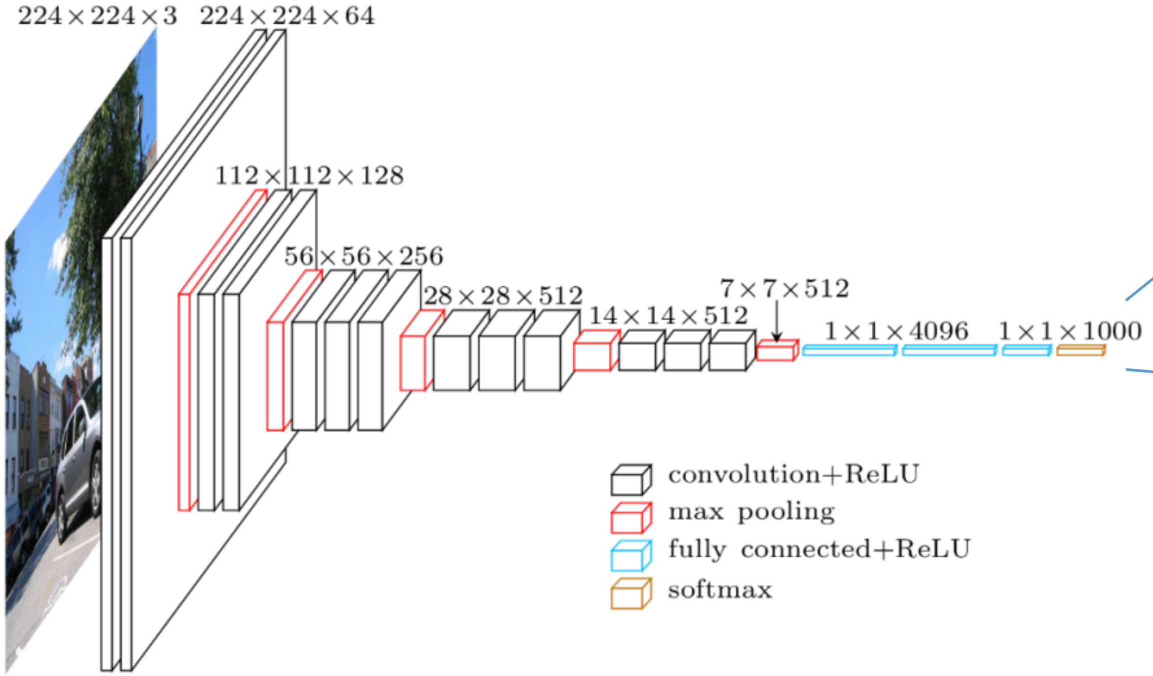
$$I \rightarrow (x, y, h, w, l)$$

This is a multi-task learning problem, as the two outputs have different nature

Multitask Learning

Train a network to predict both the class label and the bounding box

"Two heads"



Target class

Soft-max loss $\mathcal{S}$

Categorical crossentropy loss

Bounding Box Coordinates

(x, y, w, h)

Regression loss $\mathcal{R}$, e.g. the $\ell^2, \ell^1, ..$ bounding box.

$$\left\| [\hat{x}, \hat{y}, \hat{w}, \hat{h}] - [x, y, w, h] \right\|_2^2$$

ESTIMATED ONE   TRUE B.B LOCATION

Multitask Learning

The training loss has to be a single scalar since **we compute gradient of a scalar function** with respect to network parameters.

Minimize a multitask loss to merge two losses:

$$\mathcal{L}(x) = \alpha\, \mathcal{S}(x) + (1 - \alpha)\mathcal{R}(x)$$

and $\alpha \in [0,1]$ is an hyper parameter of the network.

Watch out that $\boldsymbol{\alpha}$ **directly influences the loss definition**, tuning might be difficult. Better to do cross-validation looking at some other loss (loss value for different values of α might be meaningless).

It is also possible to **adopt a pre-trained model** and then train the two FC separately… however it is always better to perform at least some fine tuning to train the two jointly.

“Quick and Dirty” Solution

```
# Add the classifier layer to the MobileNet
inputs = tfk.Input(shape=(img_size,img_size,3))
x = mobile(inputs)
x = tfkl.Dropout(0.5)(x)
# The network has two heads, one for classification using sigmoid (when it is binary classification)
class_output = tfkl.Dense(1, activation='sigmoid', name='classifier')(x)
# The other head has 4 sigmoid activation to predict the bounding box, each number to be considered in [0,1] as its location is normalized w.r.t. the image sizes. The bounding boxes then cannot be predicted outside the image
box_output = tfkl.Dense(4, activation='sigmoid', name='localizer')(x)
```

“Quick and Dirty” Solution

```
# Connect input and output through the Model class. Here the output is the
concatenation of the outputs of the two heads

object_localization_model = tfk.Model(inputs=inputs, outputs=[class_output,
box_output], name='object_localization_model')


# Compile the model using binary cross entropy over the «stacked» outputs.
The labels for training need to stacked accordingly

object_localization_model.compile(loss=tfk.losses.BinaryCrossentropy(), opt
imizer=tfk.optimizers.Adam())

object_localization_model.summary()
```

However, this solution is not:

- Able to handle multi-class classification
- Predict bounding boxes outside the image

To implement a multi-task loss it is necessary to modify the training loop

Human Pose Estimation

Pose estimation is formulated as a **CNN-regression problem towards body joints**. This is a **localization task!**

Represent pose as a set of 14 joint positions:

Left / right foot
Left / right knee
Left / right hip
Left / right shoulder
Left / right elbow
Left / right hand
Neck
Head top

6x2

2

14 locations to estimate
28 output neurons

Alexander Toshev and Christian Szegedy, "DeepPose: Human Pose Estimation via Deep Neural Networks", CVPR 2014

Extension to Human Pose Estimation

Pose estimation is formulated as a **CNN-regression problem towards body joints.**

- **The network receives as input the whole image, capturing the full-context of each body joints.**
- The approach is **very simple to design and train.** Training problems can be **alleviated by transfer learning** of existing classification networks

Pose is defined as a vector of k joints location for the human body, possibly normalized w.r.t. the bounding box enclosing the human.

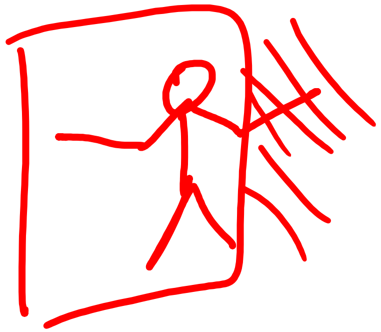
Train a CNN to predict **a $2k$ vector as output** by using an Alexnet-like architecture.

Alexander Toshev and Christian Szegedy, “DeepPose: Human Pose Estimation via Deep Neural Networks”, CVPR 2014

Training Human Pose Estimation Networks

Adopt a ℓ^2 regression loss of the estimated pose parameters over the annotations.

- The network always provide a fixed when a few joints are not visible.



Reduce overfitting by augmentation (translation and flips).

Multiple networks have been trained to improve localization by refining joint position in a crop around the initial detection.

Open Pose

Cao, Z et al.. OpenPose: realtime multi-person 2D pose estimation using Part Affinity Fields. CVPR 2017

Pose Estimation

Real-time Multi-Person 2D Pose Estimation Using Part Affinity Fields

Zhe Cao, Tomas Simon, Shih-En Wei, Yaser Sheikh
Carnegie Mellon University

Cao, Z., Simon, T., Wei, S. E., & Sheikh, Y. (2017). Realtime multi-person 2d pose estimation using part affinity fields. In CVPR2017

Pose Estimation

Cao, Z., Simon, T., Wei, S. E., & Sheikh, Y. (2017). Realtime multi-person 2d pose estimation using part affinity fields. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 7291-7299).

Weakly-Supervised Localization

… Global Averaging Pooling Revisited

… visualizing what matters most for CNN predicitons

Weakly supervised localization

Perform localization over an image without images with annotated bounding box

- Training set provided as for classification with image-label pairs $\{(I, \ell)\}$ where no localization information is provided

2016

Learning Deep Features for Discriminative Localization

Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, Antonio Torralba
Computer Science and Artificial Intelligence Laboratory, MIT
{bzhou,khosla,agata,oliva,torralba}@csail.mit.edu

The GAP revisited

The advantages of GAP layer **extend beyond** simply acting as a structural **regularizer** that prevents overfitting

In fact, **CNNs can retain a remarkable localization ability** until the final layer. By a simple tweak it is possible to easily **identify the discriminative image regions leading to a prediction**.

*A **CNN trained on object categorization** is successfully able to **localize the discriminative regions for action classification** as the objects that the humans are interacting with rather than the humans themselves*

Zhou, Bolei, et al. "Learning deep features for discriminative localization." CVPR 2016.

Class Activation Mapping

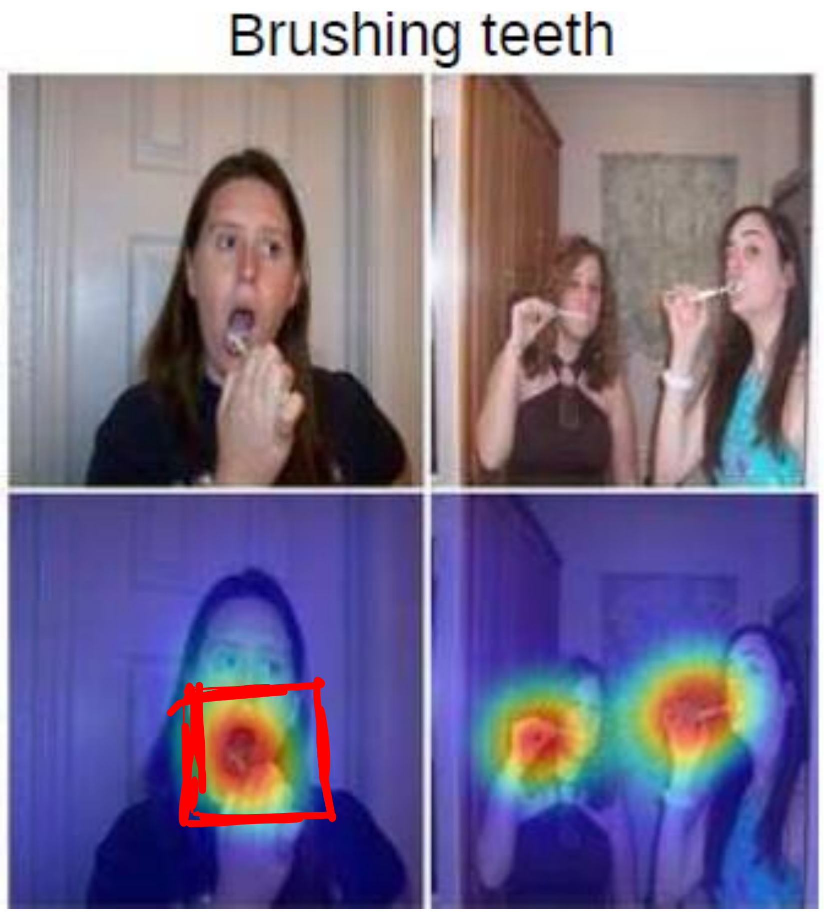
Zhou, Bolei, et al. "Learning deep features for discriminative localization." CVPR 2016.

Class Activation Mapping (CAM)

Identifying exactly **which regions** of an image **are being used for discrimination**.

CAM are very easy to compute. It just requires:

- FC layer after the GAP
- a minor tweak

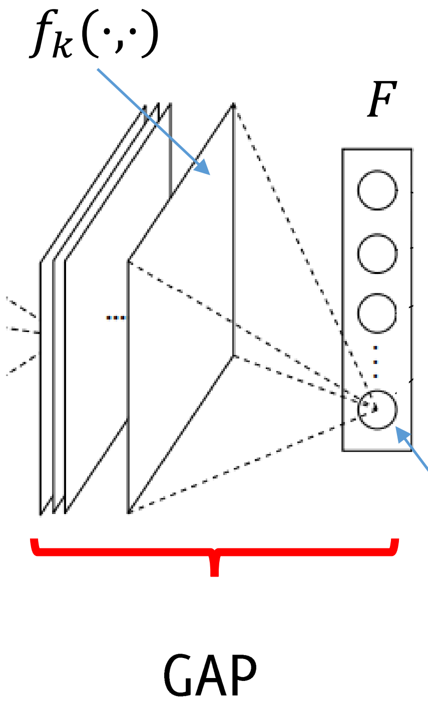


Zhou, Bolei, et al. "Learning deep features for discriminative localization." CVPR 2016.

The Global Averaging Pooling (GAP) Layer

A very simple architecture made only of convolutions and activation functions leads to a final layer having:

- n feature maps $f_k(\cdot,\cdot)$ having resolution “similar” to the input image
- a vector after GAP made of n averages F_k



$$F_k = \sum_{(x,y)} f_k(x, y)$$

Zhou, Bolei, et al. "Learning deep features for discriminative localization.“ CVPR 2016.

The Global Averaging Pooling (GAP) Layer

Add (and train) a **single FC layer** after the GAP.

The FC computes S_c for each class c as the weighted sum of $\{F_k\}$, where weights are defined during training

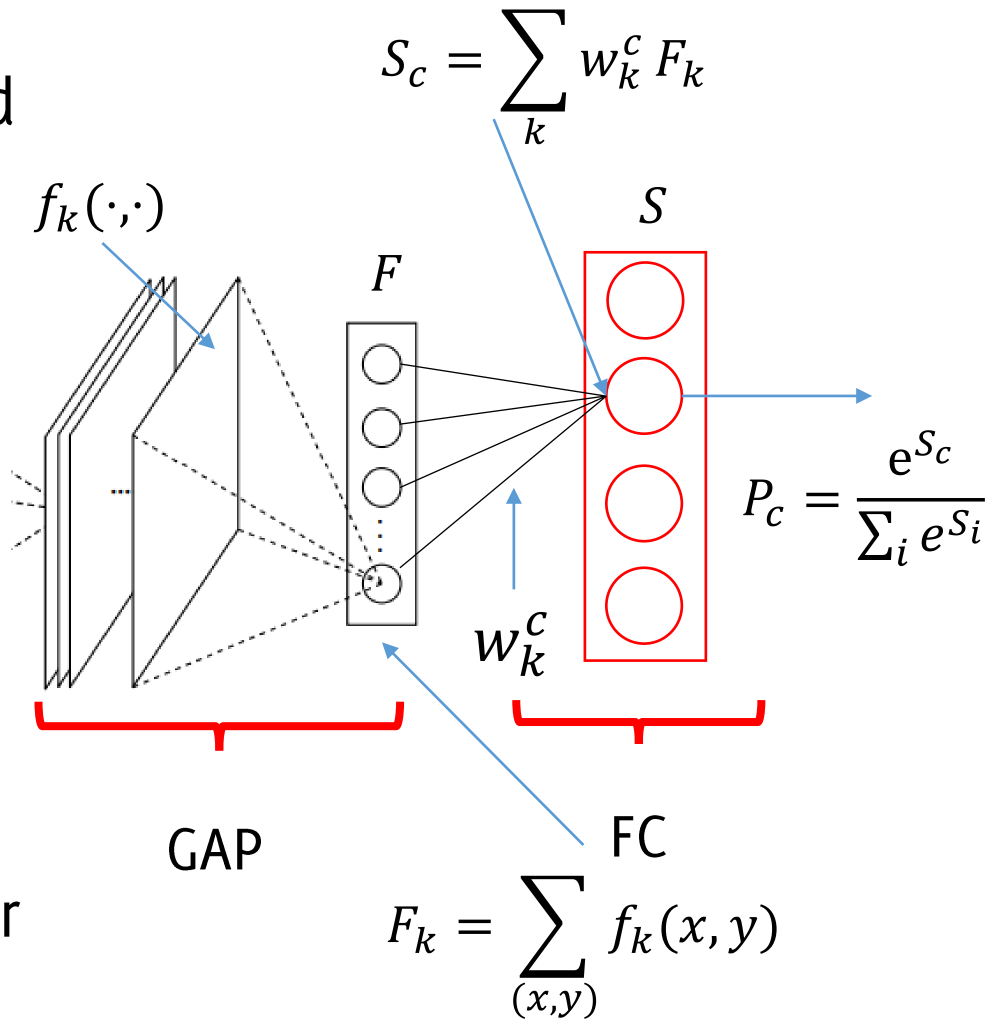
Then, the class probability P_c via soft-max (class c)

Remark: when computing

$$S_c = \sum_k w_k^c \, F_k$$

w_k^c encodes the importance of F_k for the class c,

$\{w_k^c\}_{k,c}$ are all the parameters of the last FC layer



Zhou, Bolei, et al. "Learning deep features for discriminative localization." CVPR 2016.

The Global Averaging Pooling (GAP) Layer

Perspective change in score interpretation

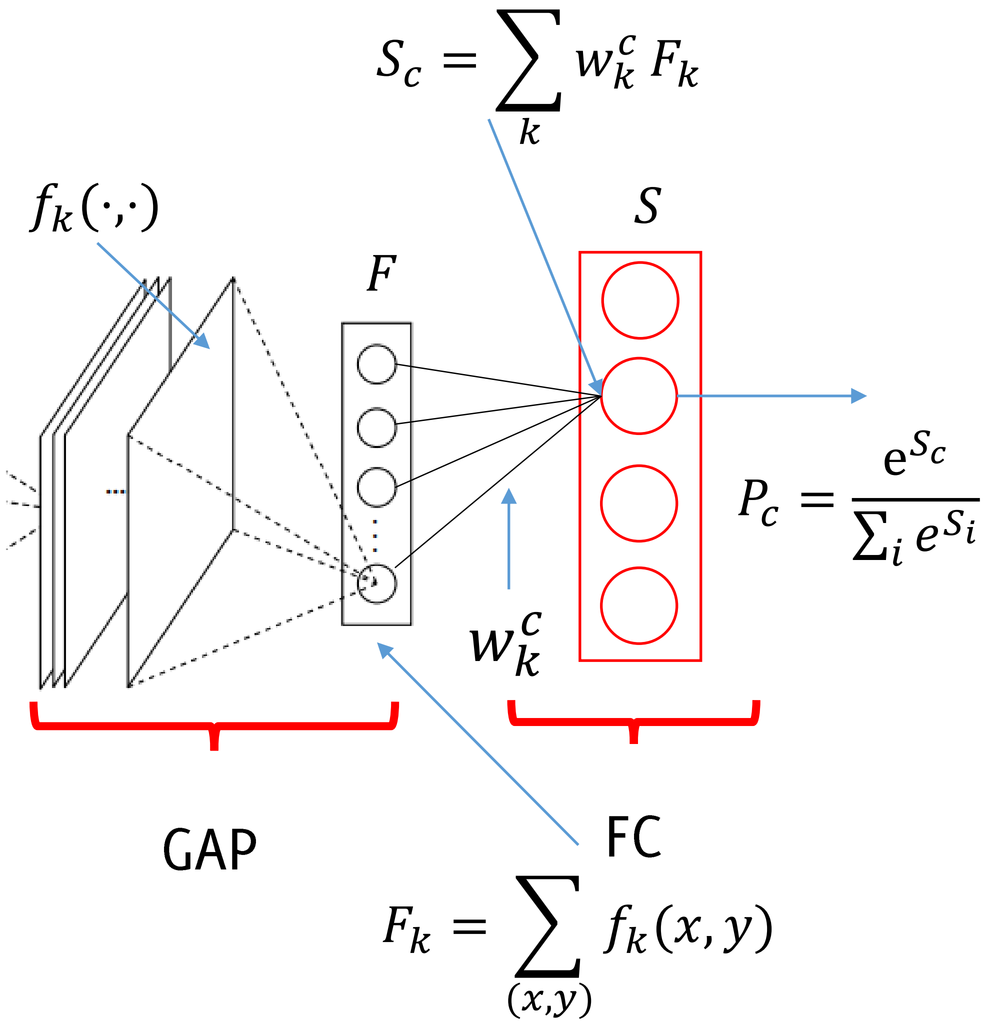
$$S_c = \sum_k w_k^c \sum_{x,y} f_k(x,y) = \sum_{x,y} \boxed{\sum_k w_k^c f_k(x,y)}$$

And CAM is defined as

$$M_c(x,y) = \sum_k w_k^c f_k(x,y)$$

where $M_c(x,y)$ directly indicates the importance of the activations at (x,y) for predicting the class c

Rmk: unlike NiN, thanks to the softmax, **the depth of the last convolutional activations can differ from the number of classes**



Zhou, Bolei, et al. "Learning deep features for discriminative localization." CVPR 2016.

The Global Averaging Pooling (GAP) Layer

FIX c

f_k

$$\boxed{S_c} = \sum_k w_k^c \boxed{\sum_{x,y} f_k(x,y)} = \sum_{x,y} \boxed{\sum_k w_k^c \underline{f_k(x,y)}}$$

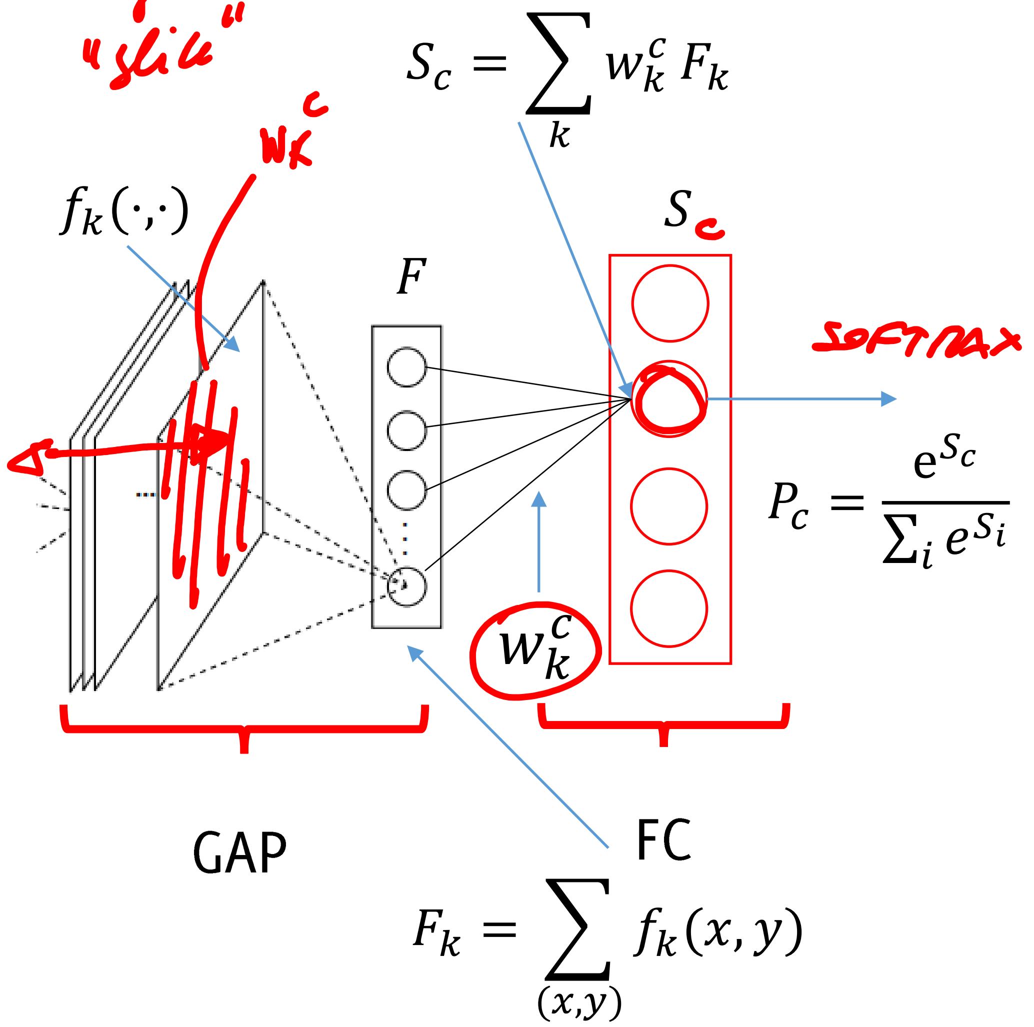
x, y are fixed "slice"

CAM is defined as

$$M_c(x,y) = \boxed{\sum_k w_k^c f_k(x,y)}$$

where $M_c(x,y)$ directly indicates the importance of the activations at (x,y) for predicting the class c

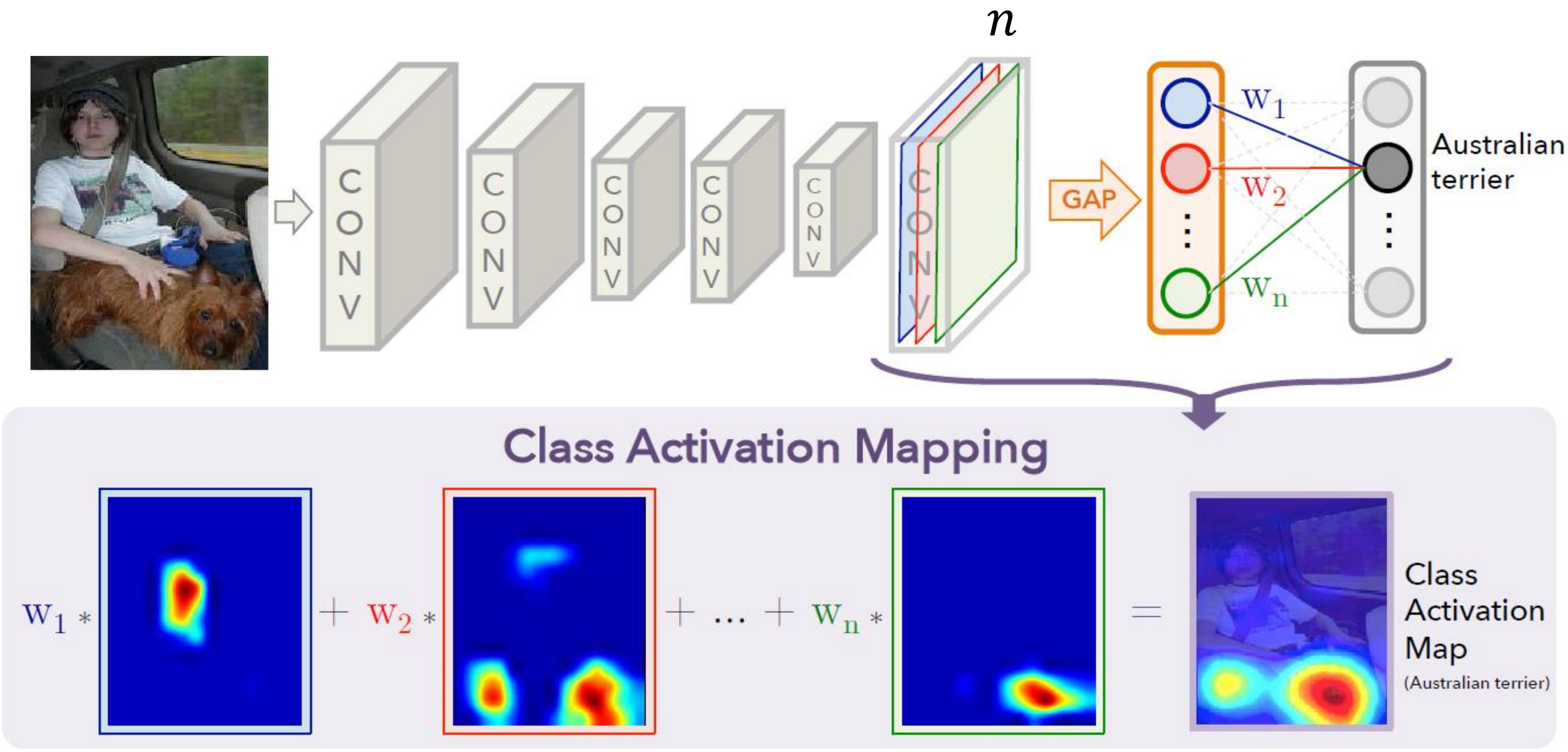
Rmk: unlike NiN, thanks to the softmax, **the depth of the last convolutional activations can differ from the number of classes**



Zhou, Bolei, et al. "Learning deep features for discriminative localization." CVPR 2016.

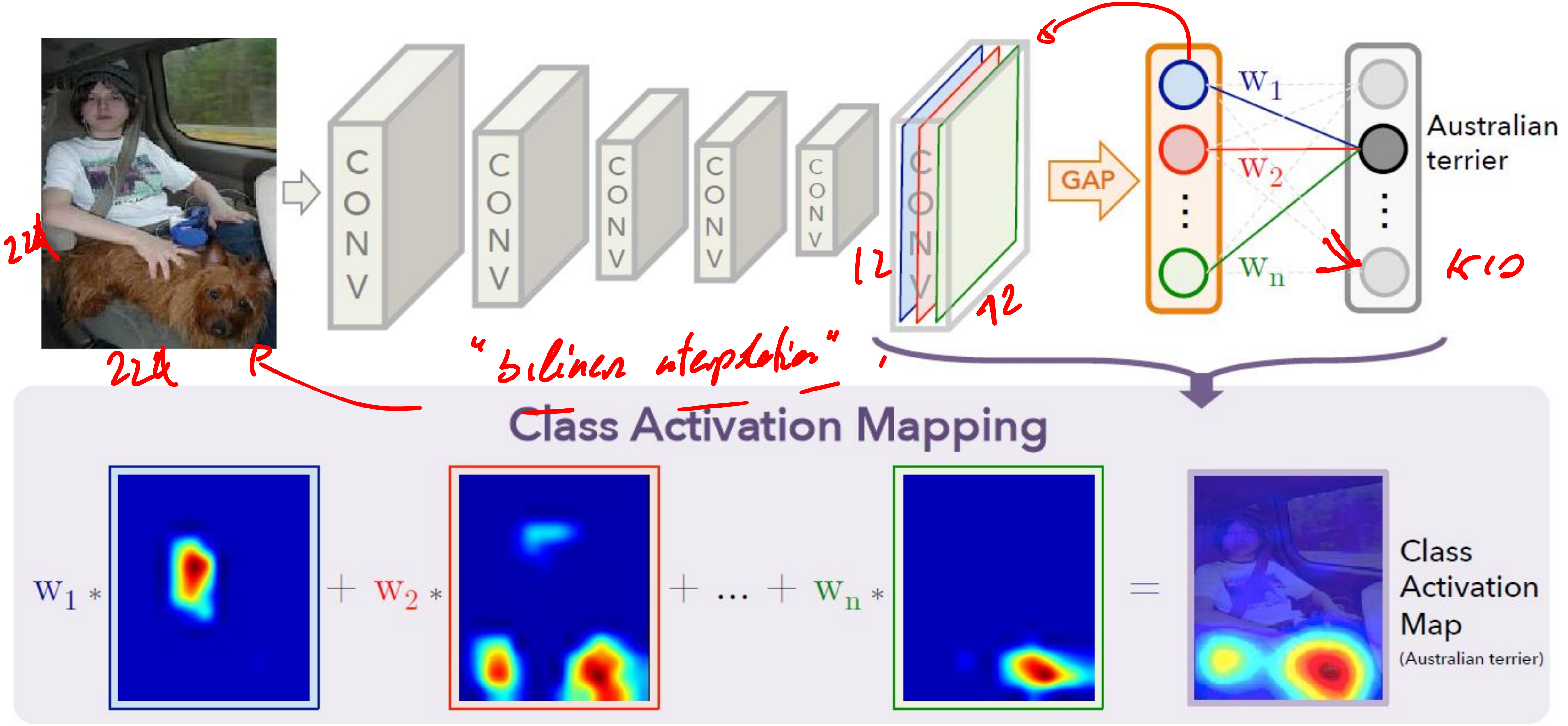
Class Activation Mapping

Now, the weights represents the importance of each feature map to yield the final prediction. Upsampling might be necessary to match the input image



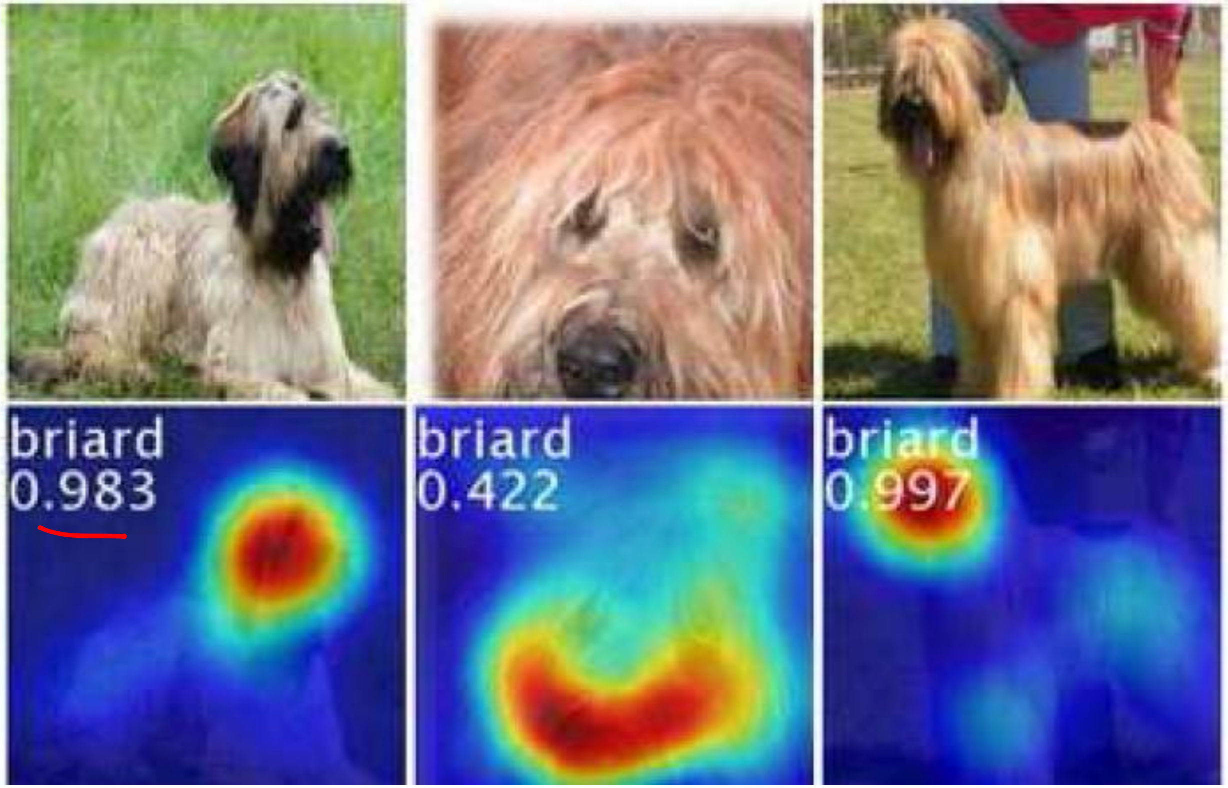
Zhou, Bolei, et al. "Learning deep features for discriminative localization." CVPR 2016.

Class Activation Mapping



Zhou, Bolei, et al. "Learning deep features for discriminative localization." CVPR 2016.

Class Activation Mapping



Zhou, Bolei, et al. "Learning deep features for discriminative localization." CVPR 2016.

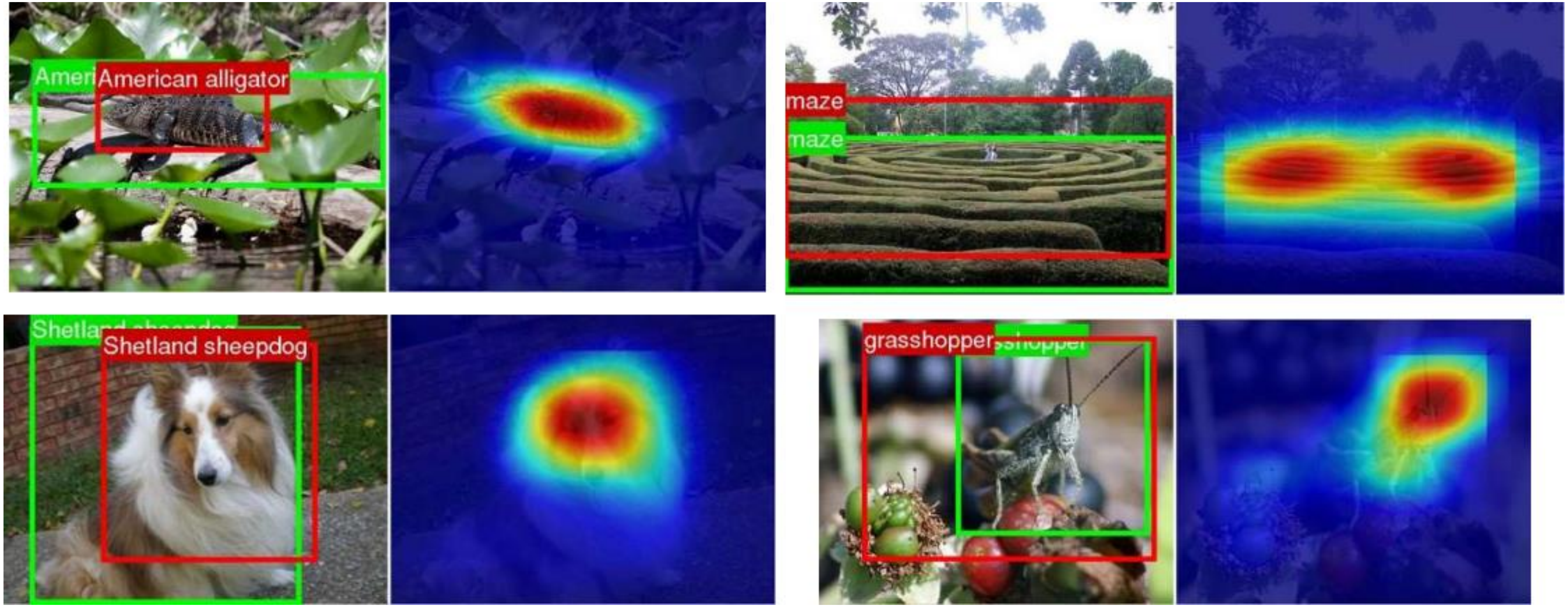
Remarks

- CAM can be **included in any pre-trained network**, as long as all the FC layers at the end are removed
- The **FC used for CAM is simple**, few neurons and no hidden layers
- **Classification performance might drop** (in VGG removing FC means loosing 90% of parameters)
- **CAM resolution** (localization accuracy) can improve by «**anticipating**» GAP to larger convolutional feature maps (but this reduces the semantic information within these layers)
- **GAP**: encourages the **identification of the whole object**, as all the parts of the values in the activation map concurs to the classification
- **GMP (Global Max Pooling)**: it is enough to have a high maximum, thus **promotes specific discriminative features**

Zhou, Bolei, et al. "Learning deep features for discriminative localization." CVPR 2016.

Weakly Supervised Localization

Use thresholding CAM values: > 20% max(CAM), then take the largest componet of the thresholded map (green GT, red estimated location)



Zhou, Bolei, et al. "Learning deep features for discriminative localization." CVPR 2016

CAM in Keras

```
def compute_CAM(model, img):
    # Expand image dimensions to fit the model input shape
    img = np.expand_dims(img, axis=0)

    # Predict to get the winning class
    predictions = model.predict(img, verbose=0)
    label_index = np.argmax(predictions)

    # Get the 1028 input weights to the softmax of the winning class
    # These are the weights of the fully connected after the GAP before the output
    class_weights = model.layers[-1].get_weights()[0]
    # These are the weights related to the winning class
    class_weights_winner = class_weights[:, label_index]
     # Take the MobileNetV2 until the final convolutional layer
    final_conv_layer = tfk.Model(
        model.get_layer('mobilenetv2_1.00_224').input,
        model.get_layer('mobilenetv2_1.00_224').get_layer('Conv_1').output)
```

CAM in Keras

```
…
   # Compute the convolutional outputs and squeeze the dimensions
    conv_outputs = final_conv_layer(img)
    conv_outputs = np.squeeze(conv_outputs)

    # Upsample the convolutional outputs
    mat_for_mult = scipy.ndimage.zoom(conv_outputs, (32, 32, 1), order=1)
    # Flatten the spatial dimension
    mat_for_mult = mat_for_mult.reshape((256*256, 1280))

    # Compute the CAM as the weighted sum of channels, using the weights of dense la
yer as weights of the combination. This is the matrix variant of the formulas seen
before, it is possible to replace this by for loops
    final_output = np.dot(mat_for_mult, class_weights_winner)

    # reshape the CAM
    final_output = final_output.reshape(256,256)

    return final_output, label_index, predictions
```

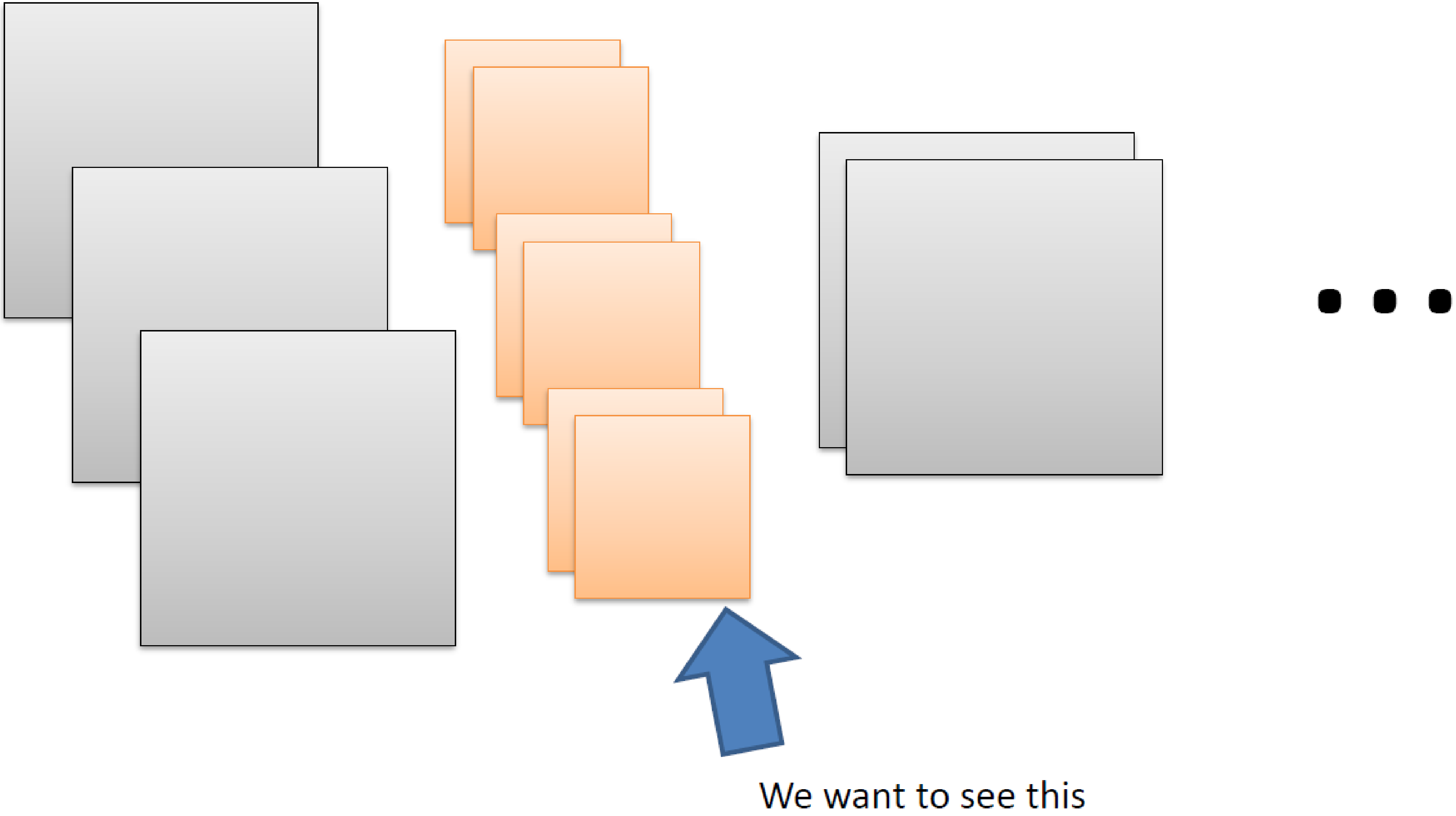
Check The Lab Session on:
Localization, MultiTask Learning, CAM

https://drive.google.com/drive/folders/1AiEwpWhd6Uru08Yerlc83875PZkzbN4W?usp=drive_link

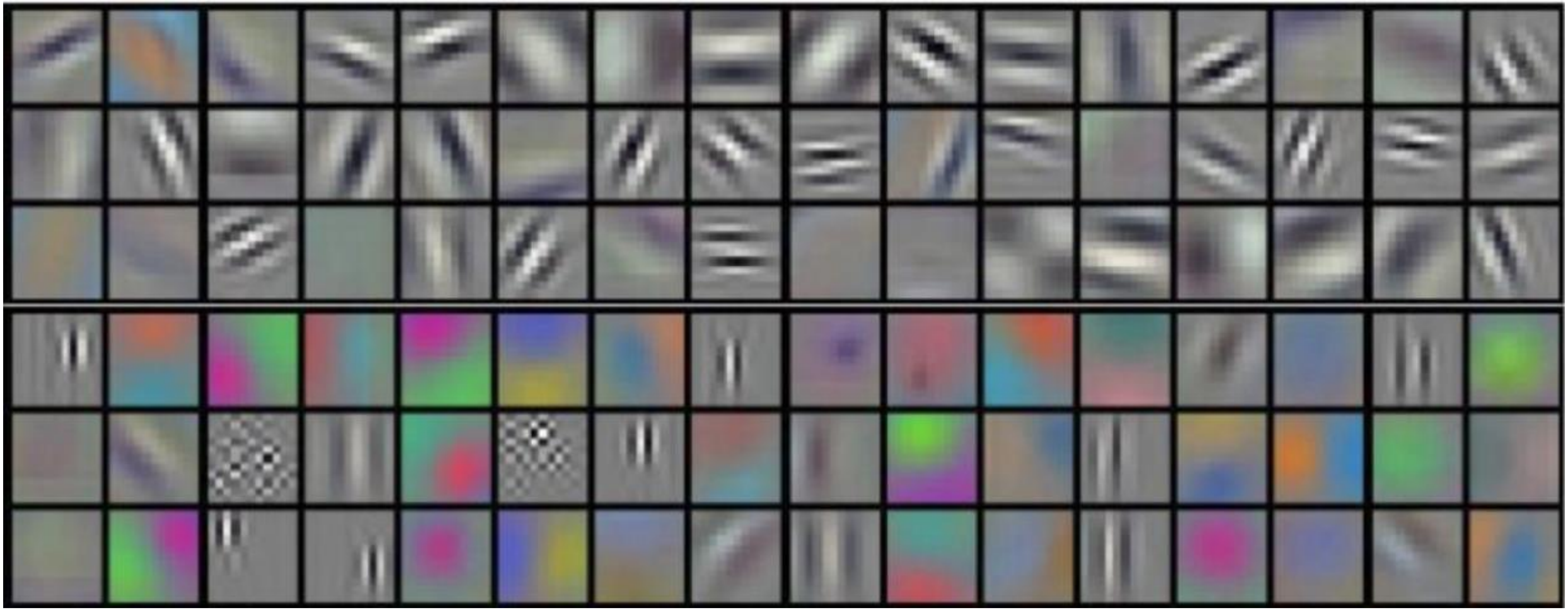
Explaining CNN Outputs

CNN Visualization

Visualizing CNN Filters



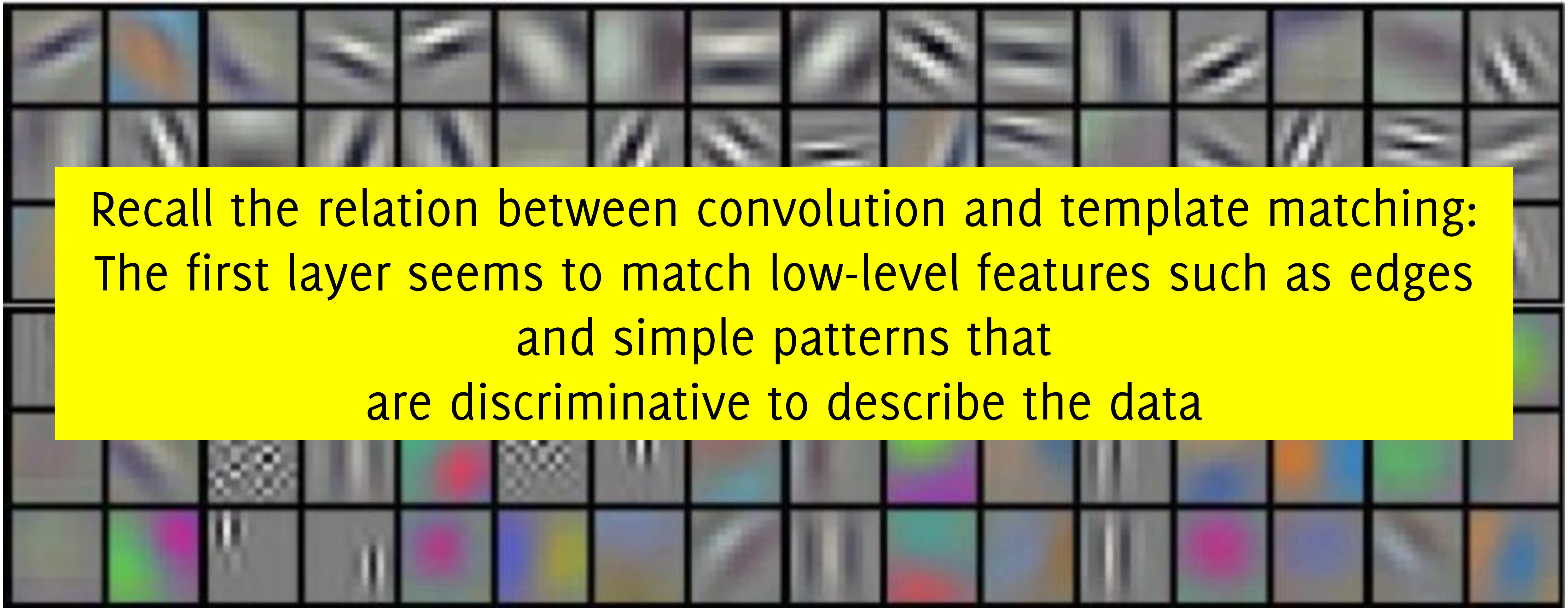
11x11x3 filters (visualized in RGB) extracted from the first convolutional layer



Do you remember?

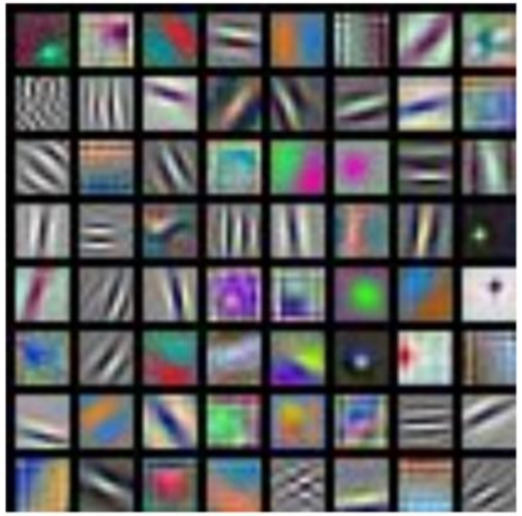
The template matching interpretation of Linear classifiers…

11x11x3 filters (visualized in RGB) extracted from the first convolutional layer



Recall the relation between convolution and template matching:
The first layer seems to match low-level features such as edges
and simple patterns that
are discriminative to describe the data

First layer's filters are often like these

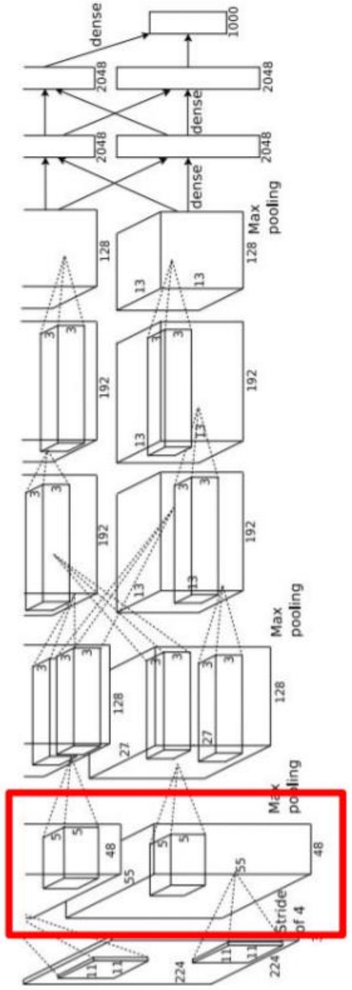


AlexNet:
64 x 3 x 11 x 11

ResNet-18:
64 x 3 x 7 x 7

ResNet-101:
64 x 3 x 7 x 7

DenseNet-121:
64 x 3 x 7 x 7



Krizhevsky, "One weird trick for parallelizing convolutional neural networks", arXiv 2014
He et al, "Deep Residual Learning for Image Recognition", CVPR 2016
Huang et al, "Densely Connected Convolutional Networks", CVPR 2017

Difficult to interpret deeper layers

layer 1 weights

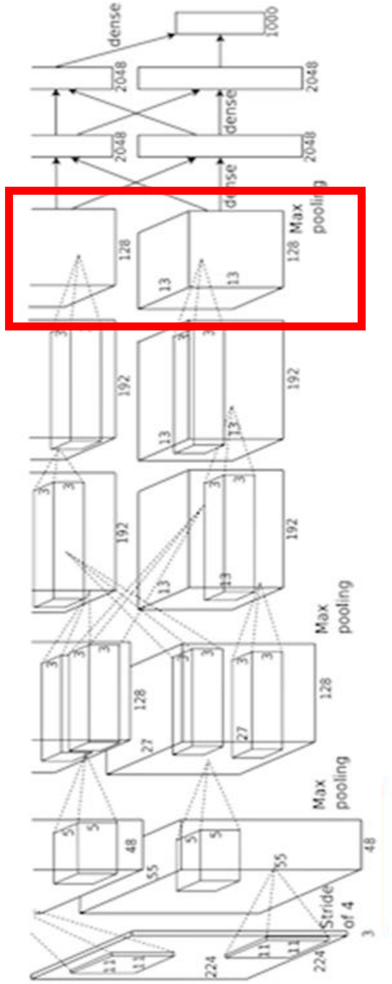
16 x 3 x 7 x 7

layer 2 weights

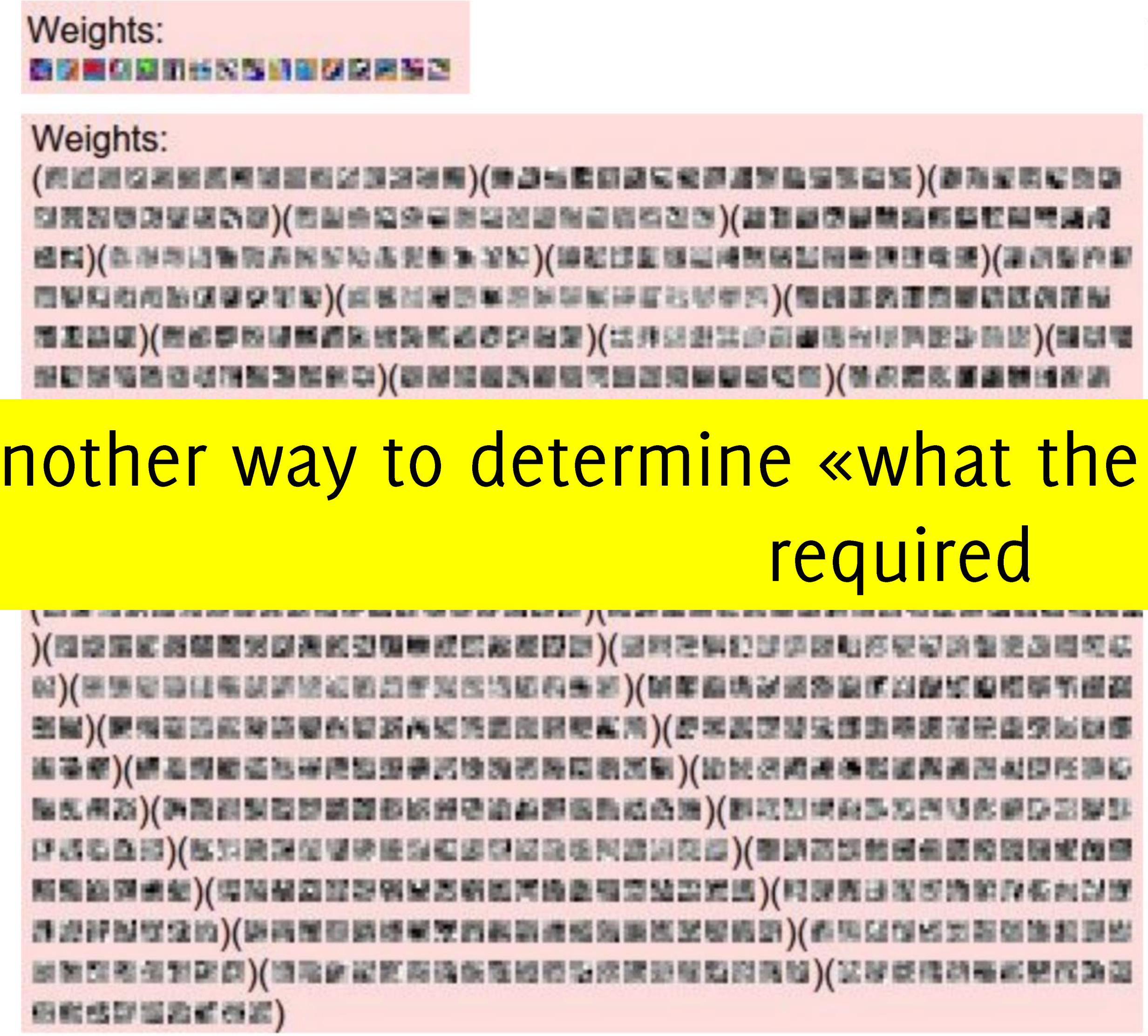
20 x 16 x 7 x 7

layer 3 weights

20 x 20 x 7 x 7



Difficult to interpret deeper layers



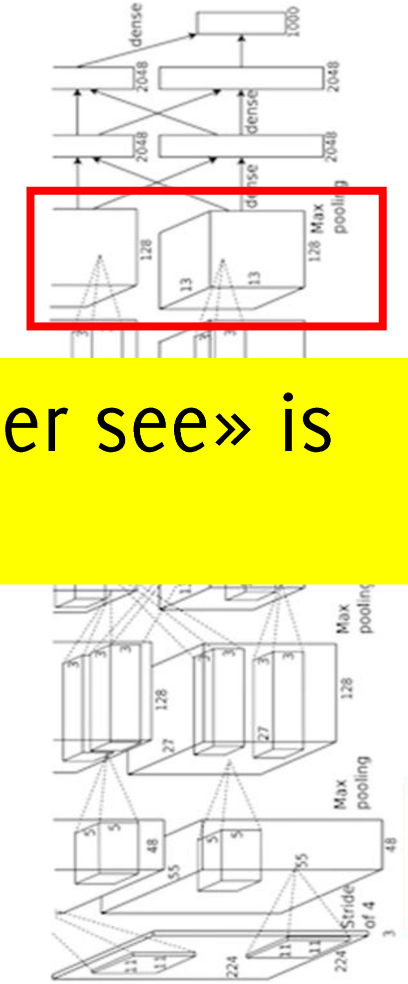
layer 1 weights

16 x 3 x 7 x 7

layer 2 weights

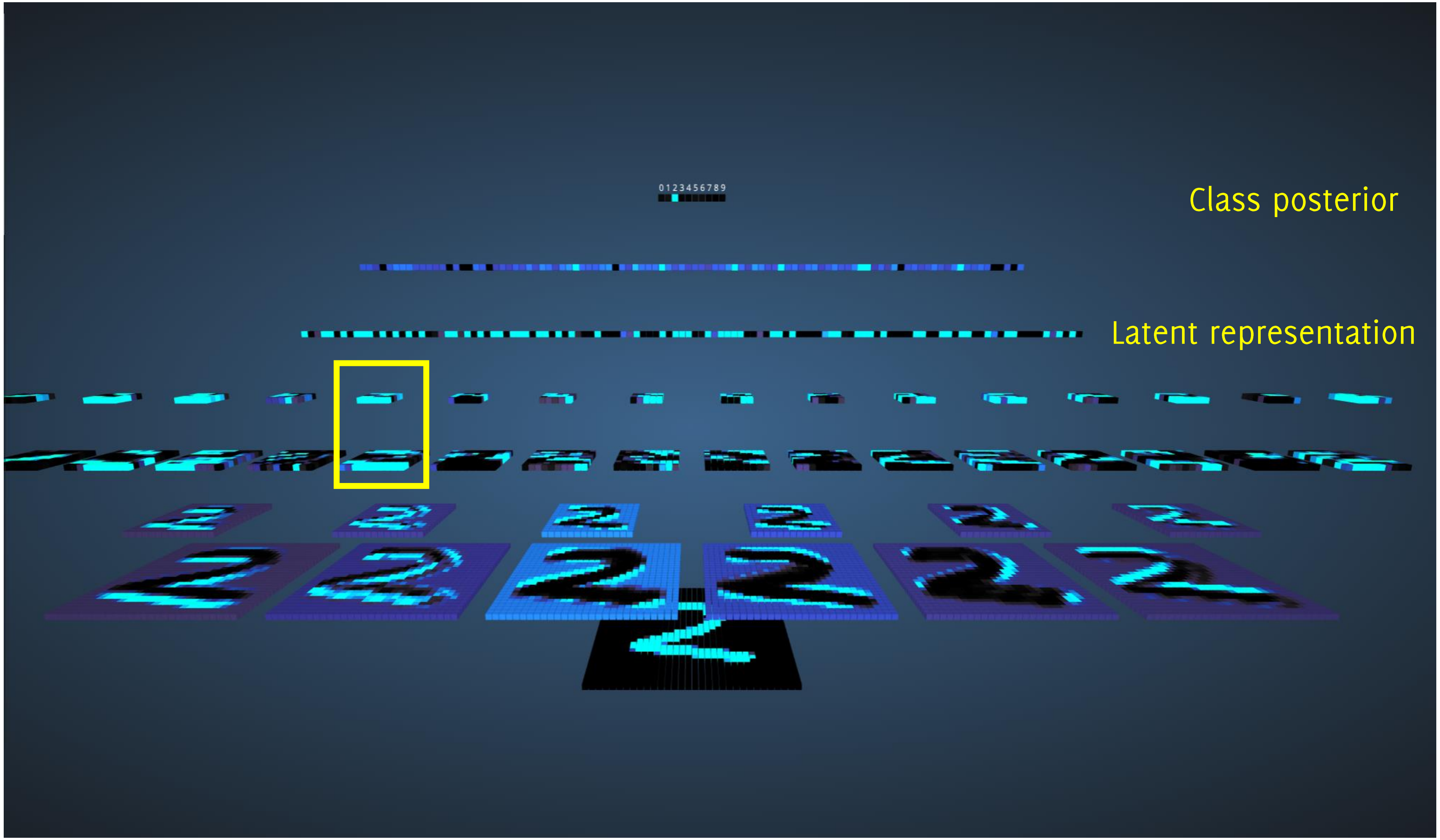
layer 3 weights

20 x 20 x 7 x 7



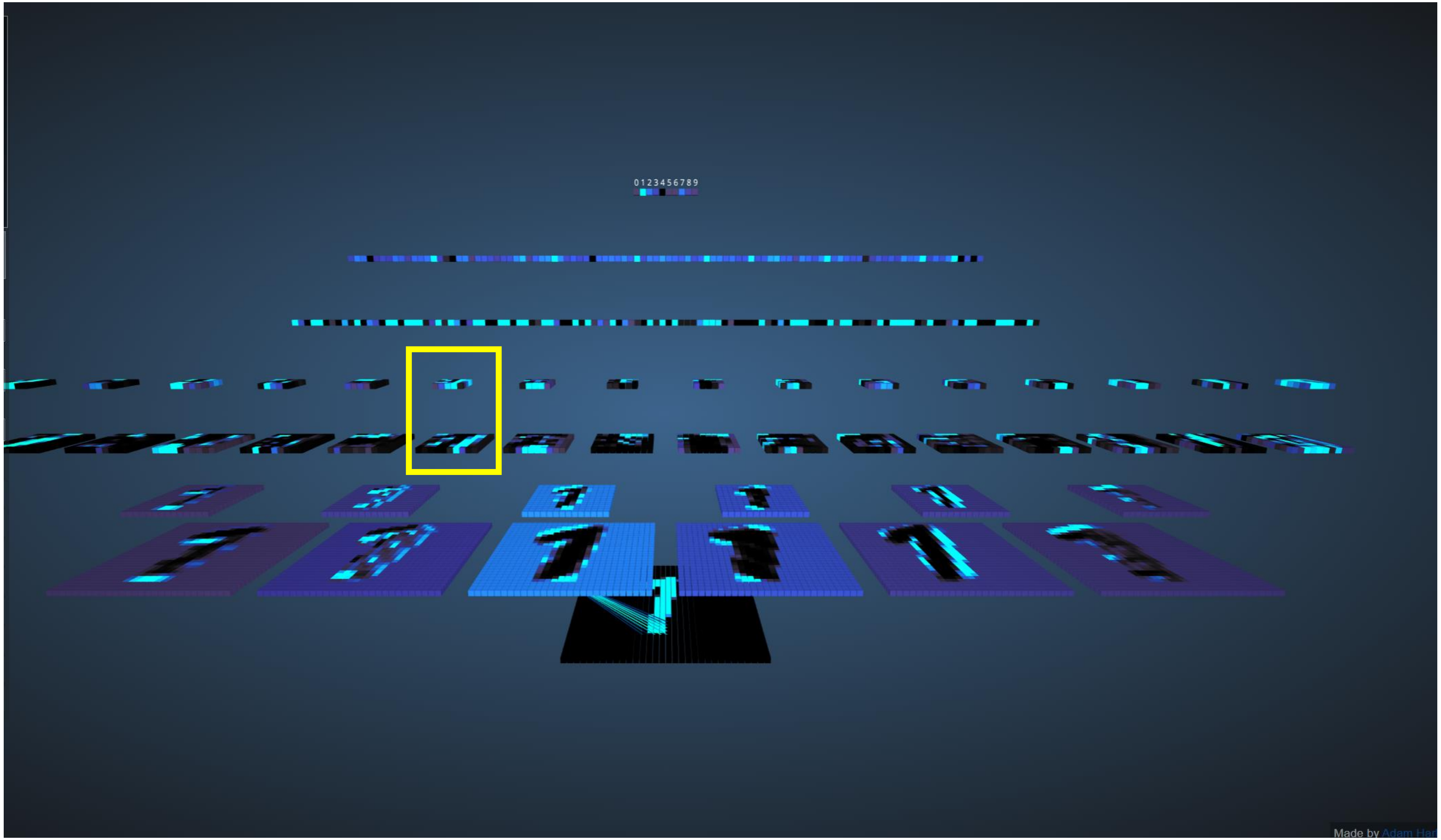
Another way to determine «what the deepest layer see» is required

What if we look at the activations?



https://www.cs.ryerson.ca/~aharley/vis/conv/

What if we look at the activations?



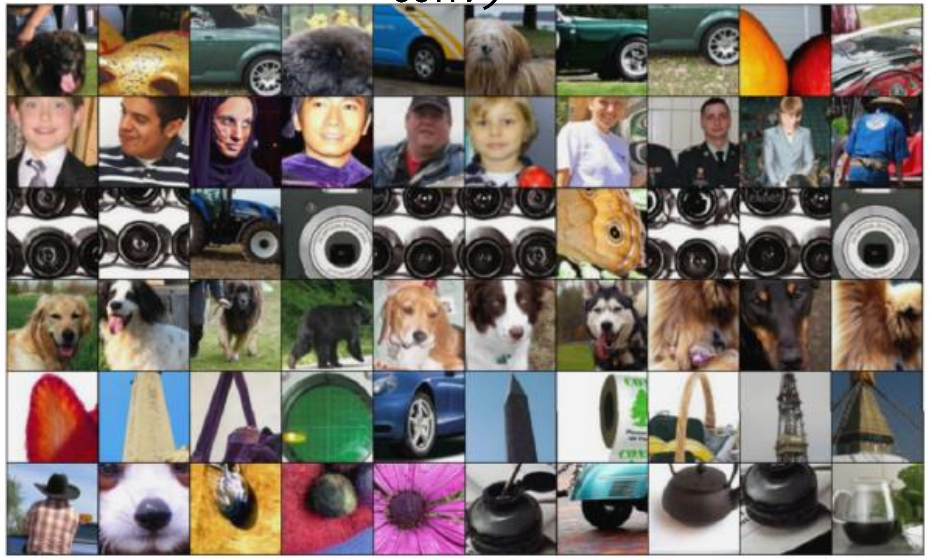
https://www.cs.ryerson.ca/~aharley/vis/conv/

Visualizing Maximally Activating Patches

1. **Select a neuron** in a deep layer of a pre-trained CNN on ImageNet
2. **Perform inference** and **store the activations** for each input image.
3. **Select the image** yielding the **maximum activation**.
4. **Show the regions** (patches) corresponding to the receptive field of the neuron.
5. Iterate for many neurons.

conv6

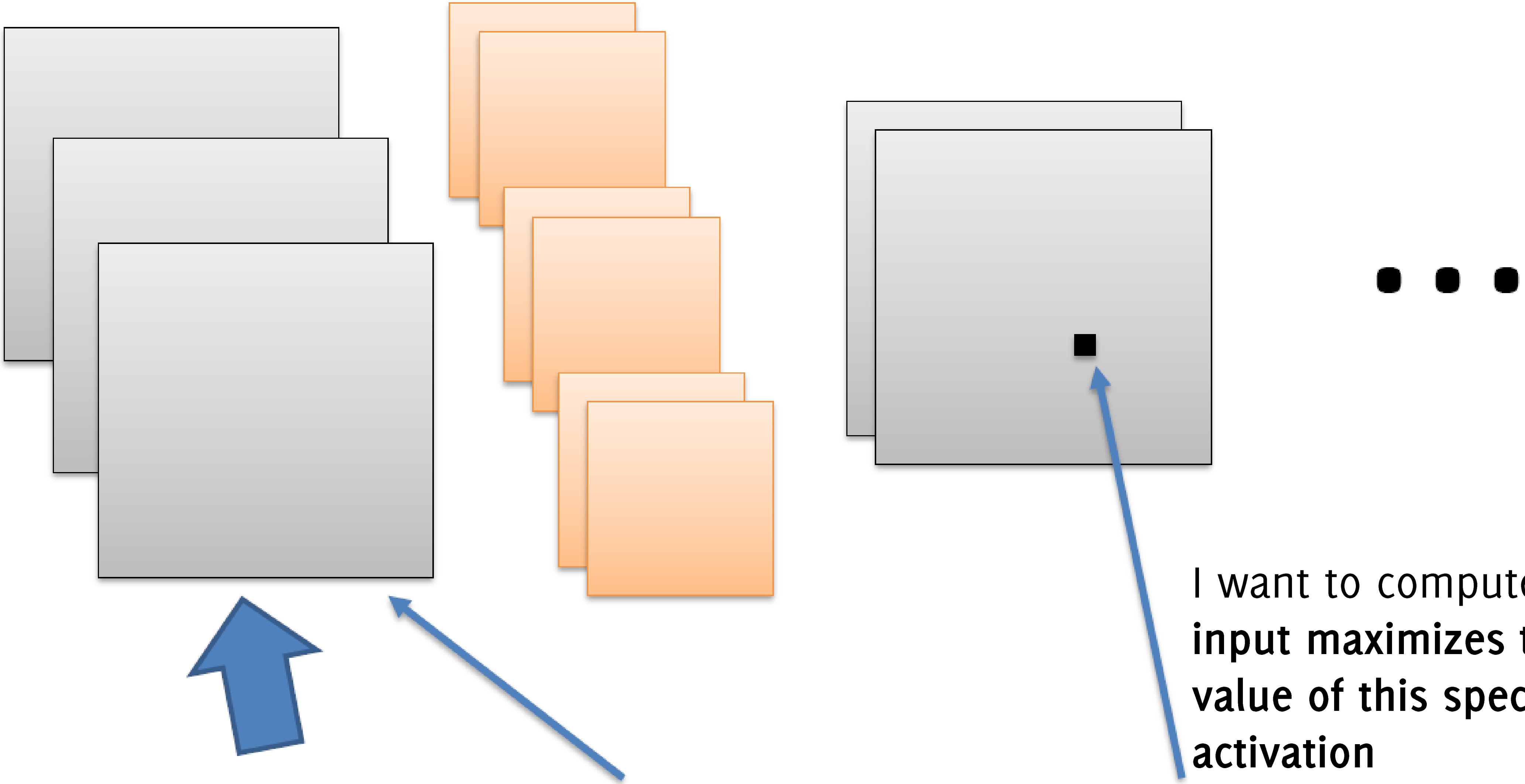
conv9



Each row in these images corresponds to different outputs of the same filter

Springenberg et al, “Striving for Simplicity: The All Convolutional Net”, ICLR Workshop 2015

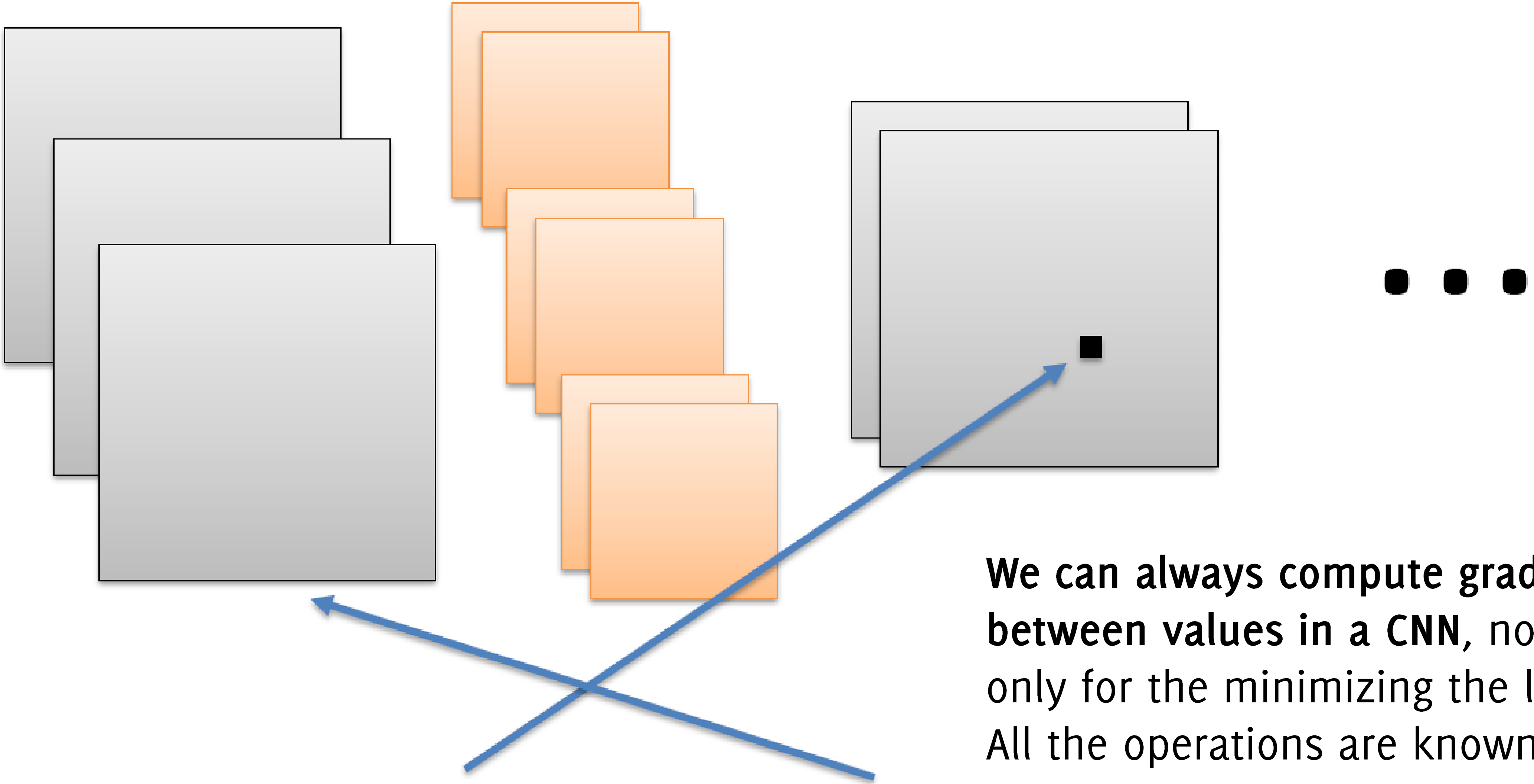
Computing Input maximally activating a neuron



I want to compute **which input maximizes the value of this specific activation**

We want to compute (and see) the input that maximally activates this guy

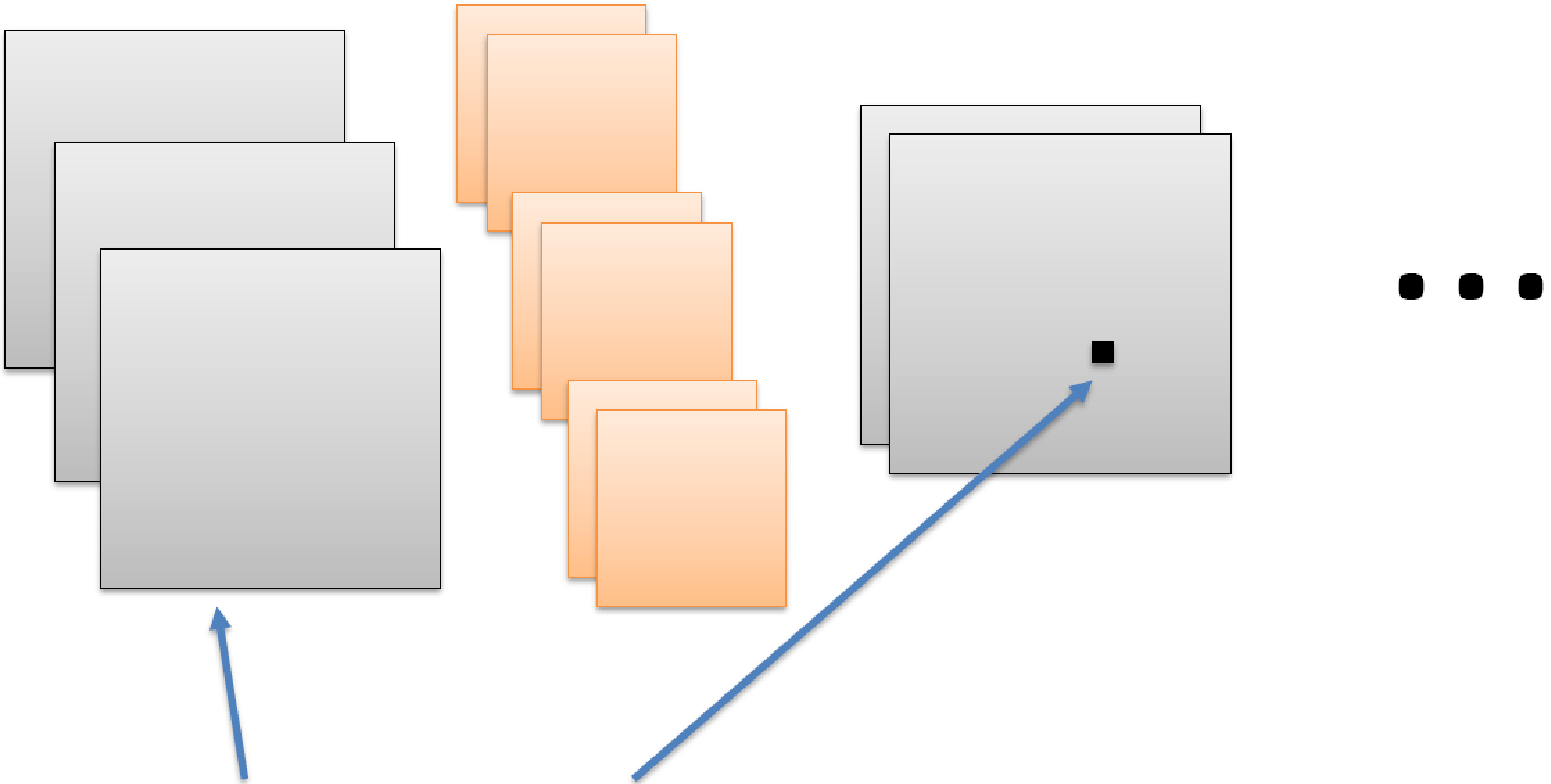
Step 1



Compute the gradient of this with respect to the input

We can always compute gradient between values in a CNN, not only for the minimizing the loss. All the operations are known

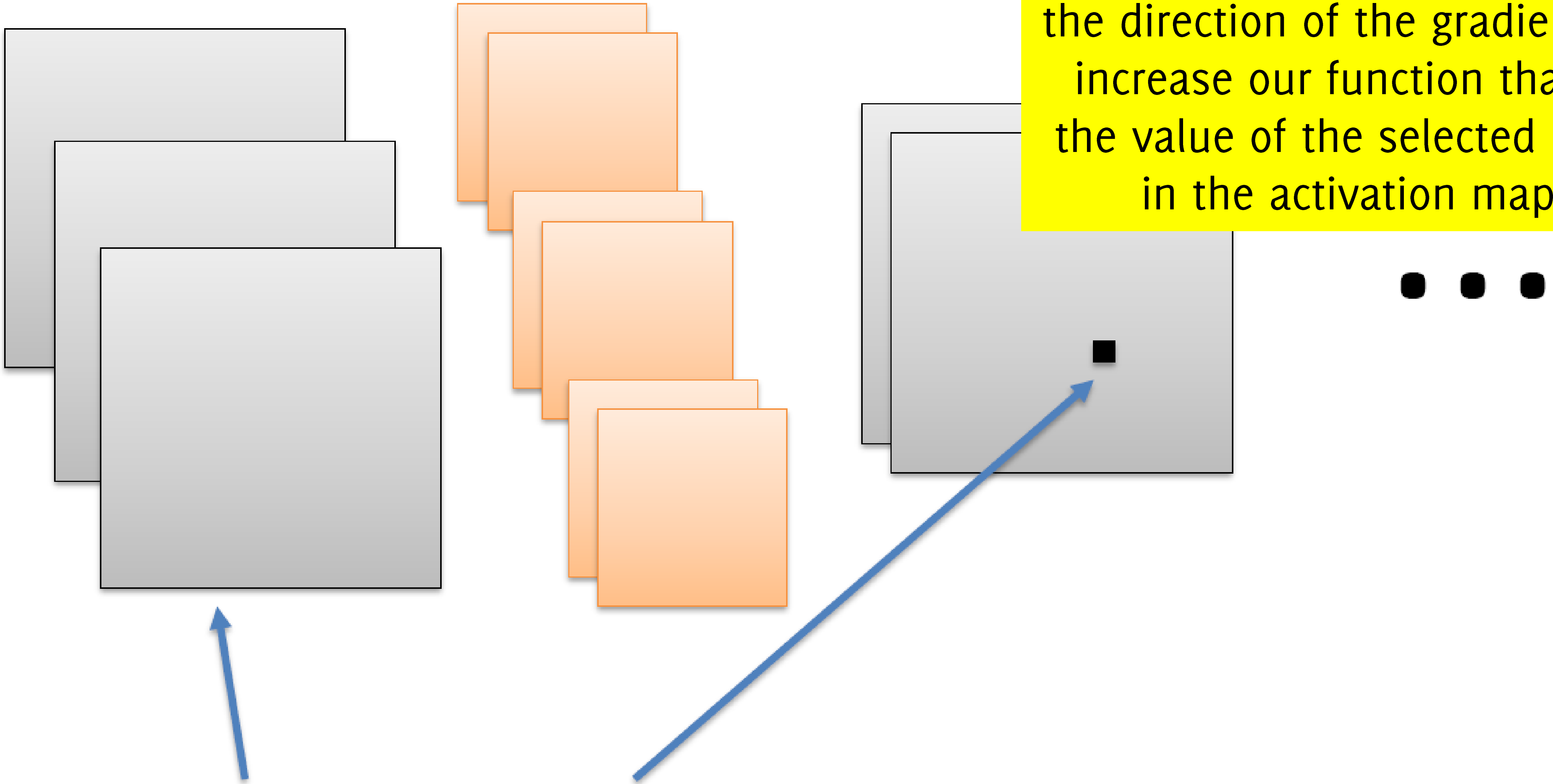
Step 2



Nudge the input accordingly: our guy will increase its activation

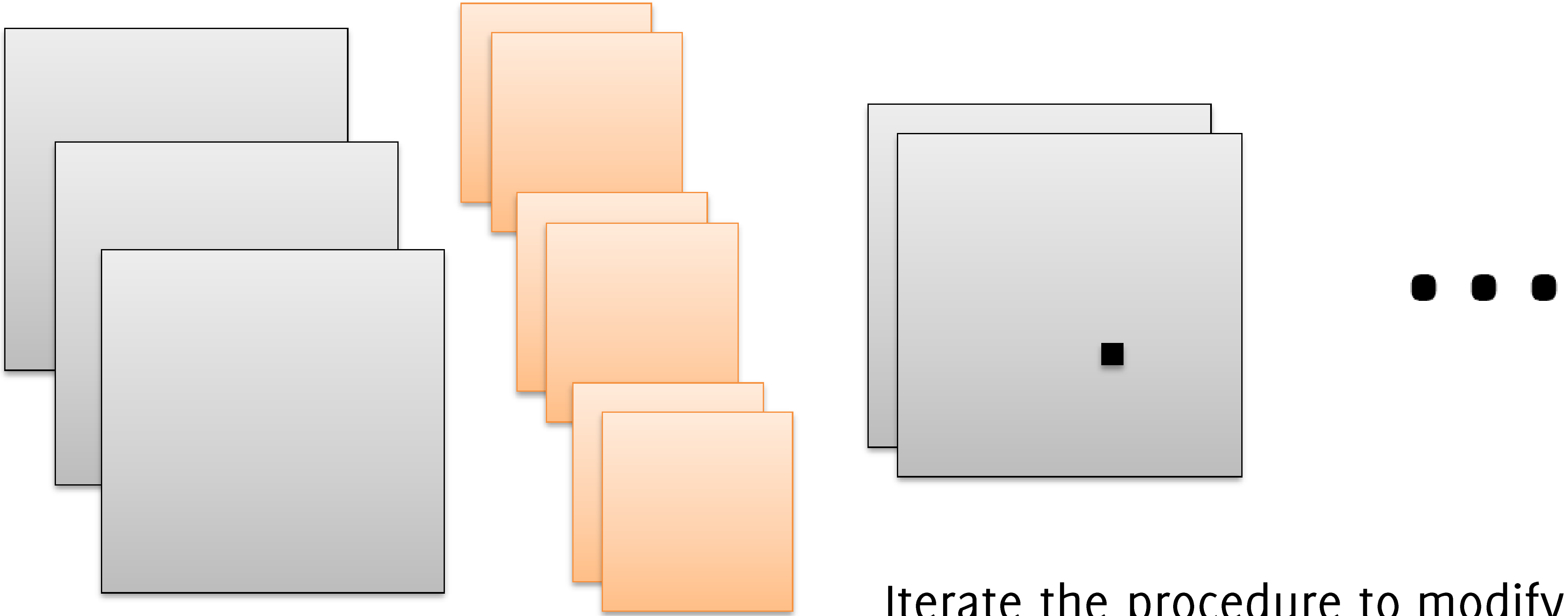
Step 2

Now we perform gradient ascent, we modify the input in the direction of the gradient, to increase our function that is the value of the selected pixel in the activation map



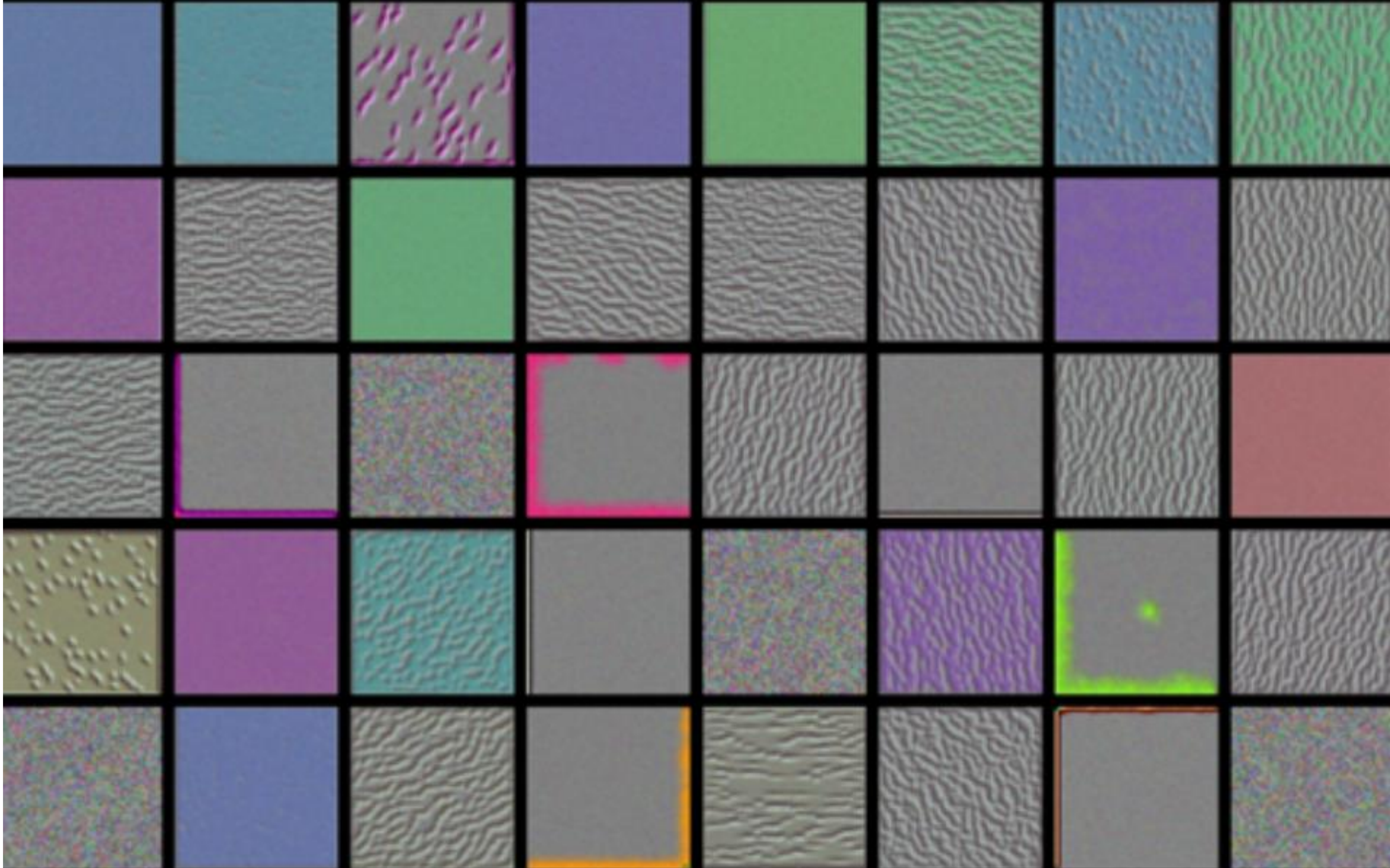
Nudge the input accordingly: our guy will increase its activation

Back to step 1 and iterate

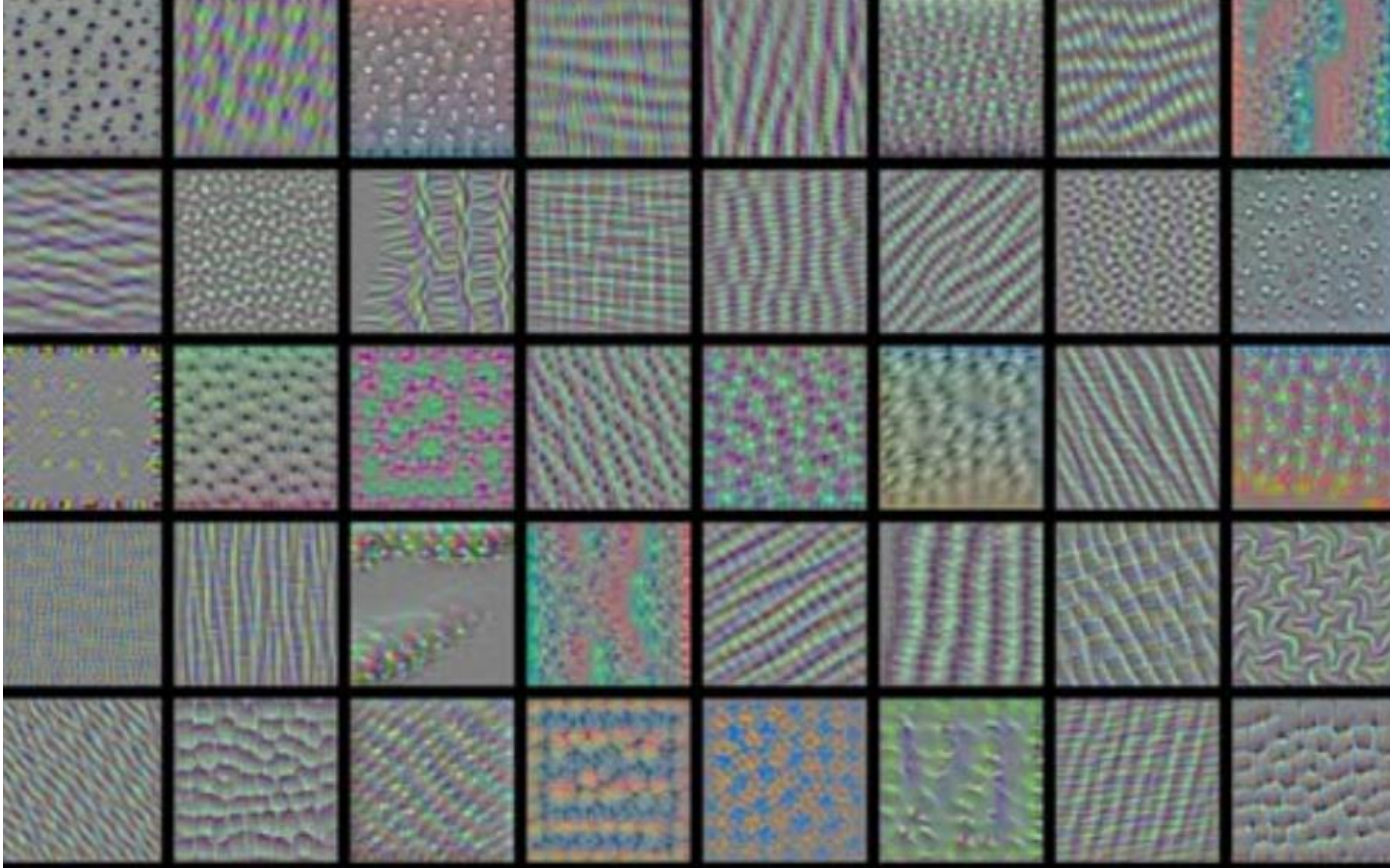


Iterate the procedure to modify the input. Some form of regularization can be added to the selected pixel value to steer the input to look more like a natural image

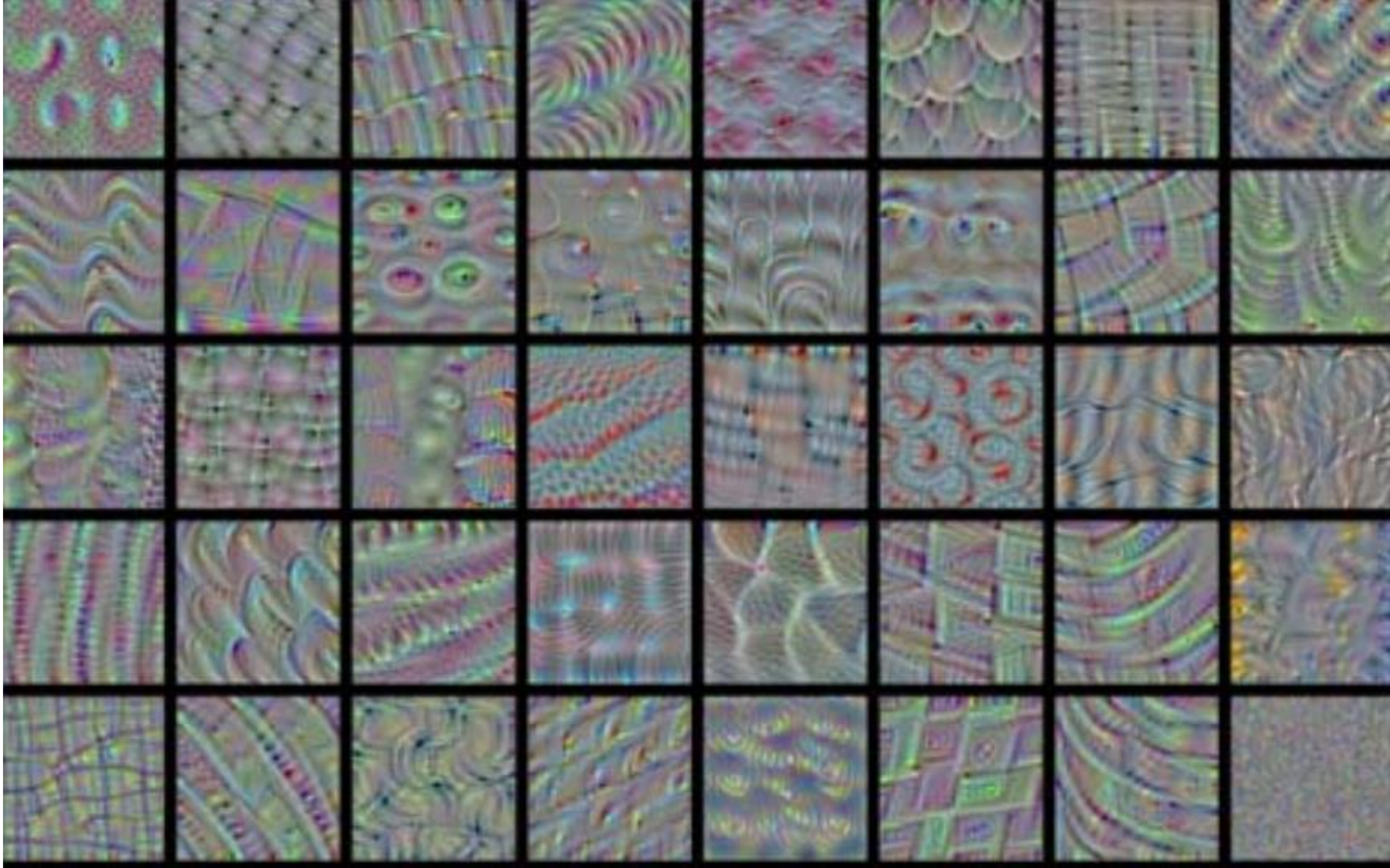
Shallow layers respond to fine, low-level patterns



Intermediate layers ...



Deep layers respond to complex,
high-level patterns



Computing Images maximally activating softmax input

Adopt gradient descent to maximally activate a neuron before the softmax (thus the network «score», which indicates the prediction)

$$\hat{I} = \underset{I}{\operatorname{argmax}}\, S_c(I) + \lambda \,\|I\|_2^2$$

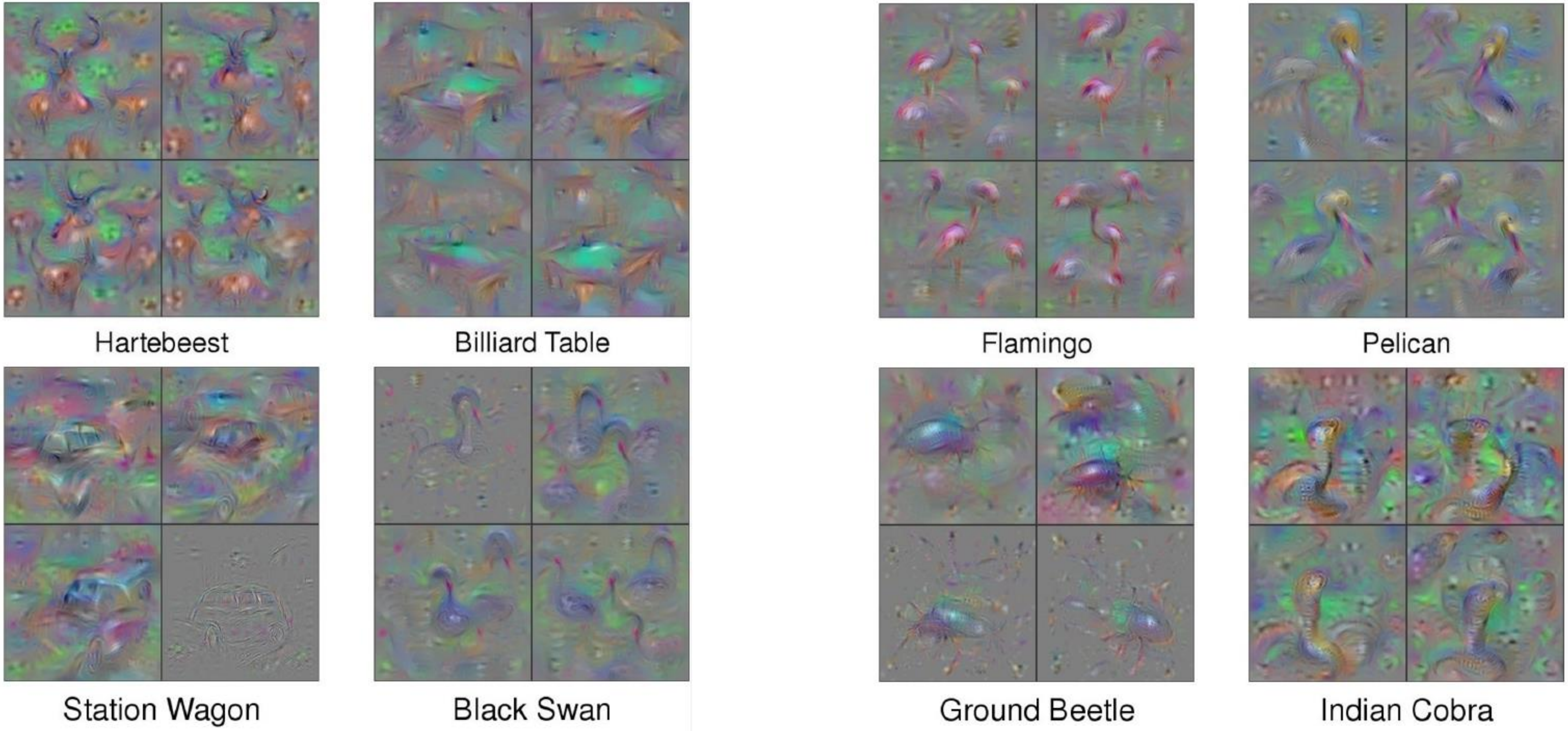
Being $\lambda > 0$ regularization parameter, c is a given output class

We add the regularization term $\lambda \,\|I\|_2^2$ to obtain smoother images

Optimize this through gradient ascent from the network for different classes c

Yosinski et al, "Understanding Neural Networks Through Deep Visualization", ICML DL Workshop 2014.

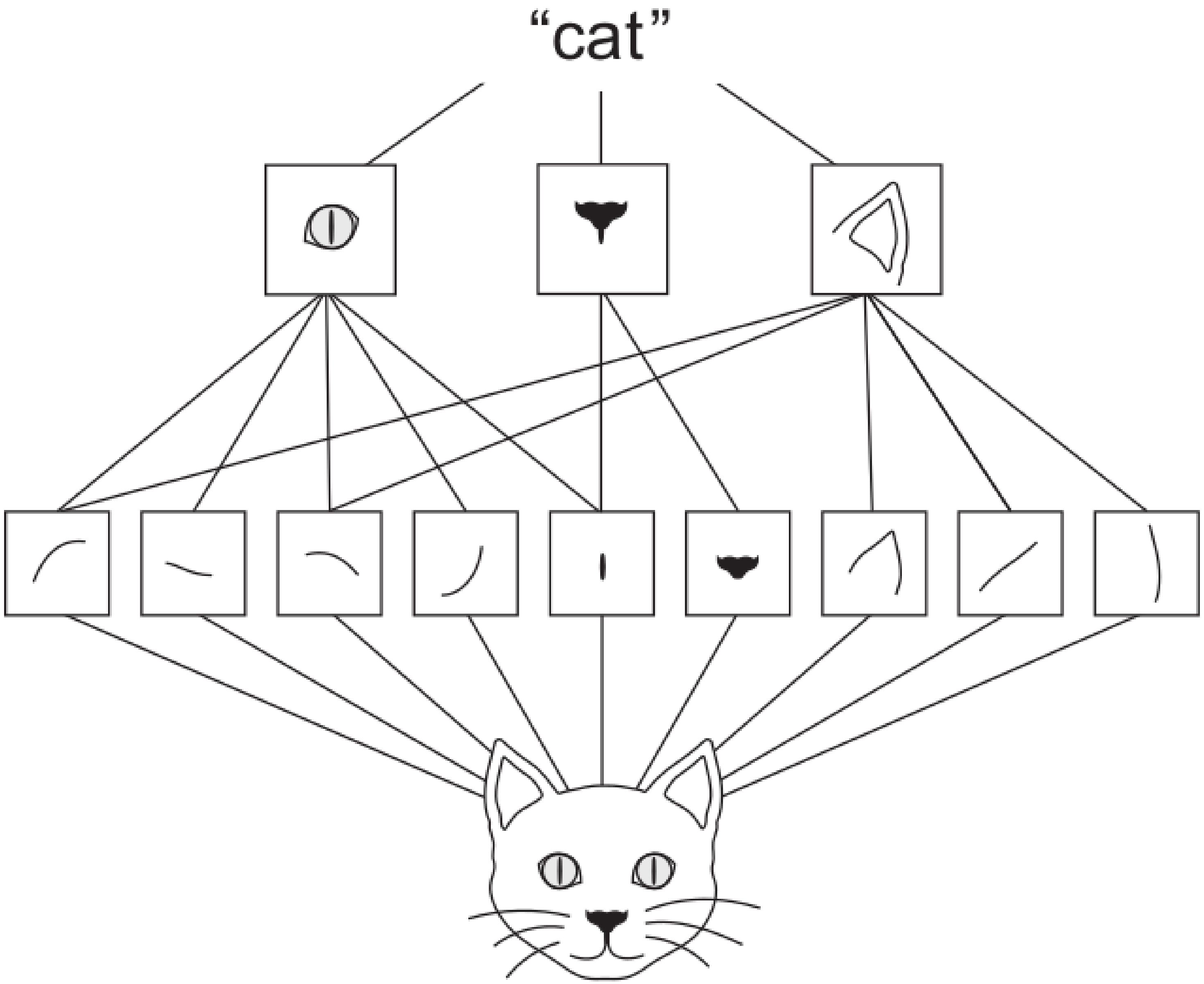
Images maximally activating softmax input



Yosinski et al, “Understanding Neural Networks Through Deep Visualization”, ICML DL Workshop 2014.

Why CNNs work

Convnets learn a **hierarchy** of **translation-invariant** spatial pattern detectors



Explaining Neural Network Predictions

Understanding Deep Neural Networks

Deep Neural Networks have **Million parameters:** their inner functioning is **totally obscure**.

Healthy scepticism to resort to NN decision in critical tasks (e.g. medical domain) or even services (e.g., blocking credit cards).

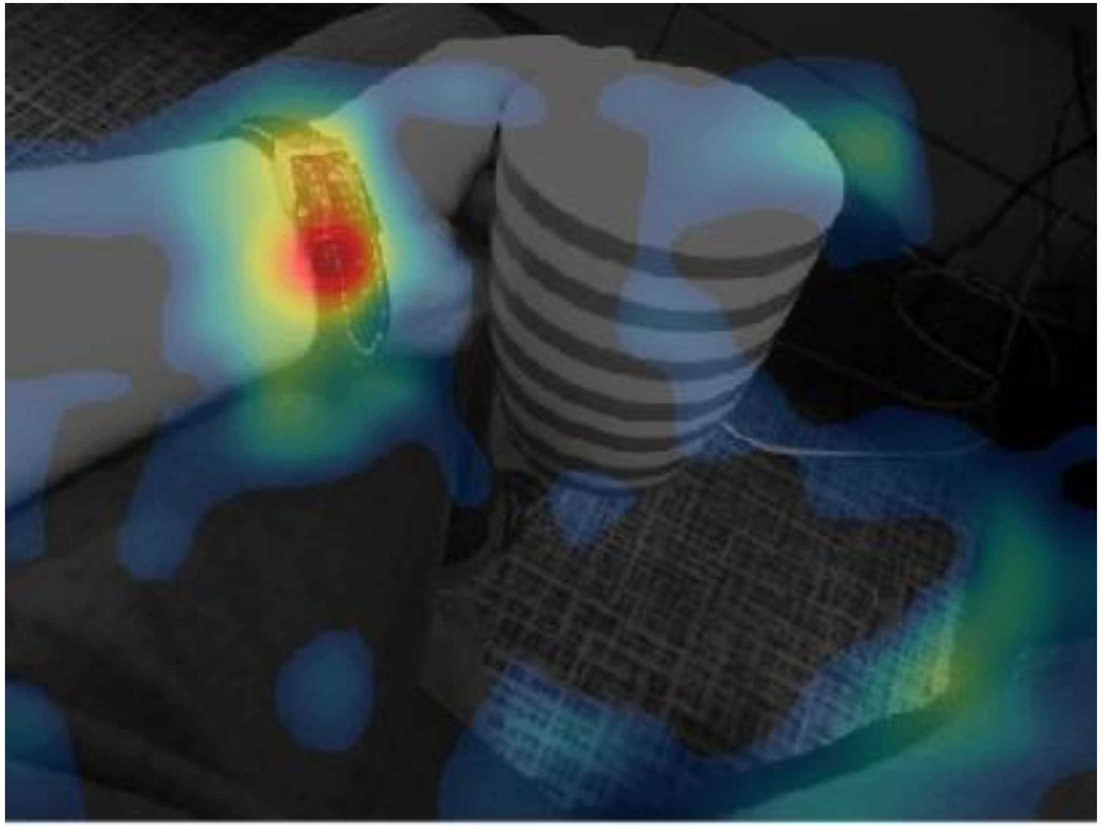
Vivid research activity around **gaining an understanding of Neural Network decision.**

Mispredicted as “buckle”

Saliency Maps to Understand Model Mistakes

Make sense of model mistakes

Mispredicted as “buckle”

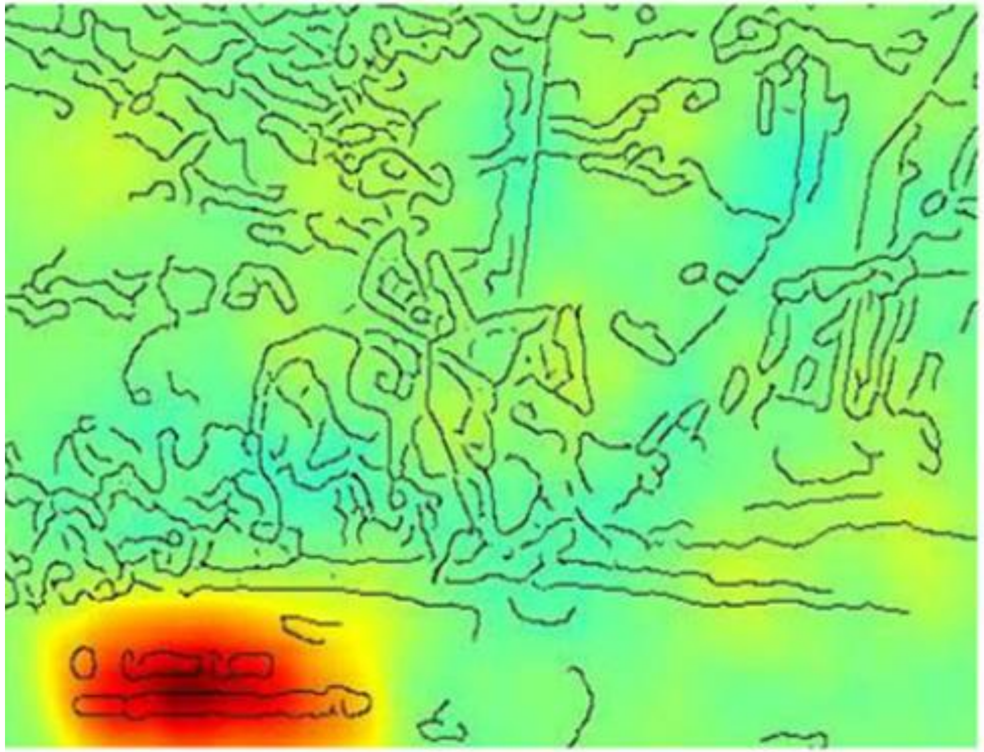


Saliency shows why

Saliency Maps to Discover Systematic Errors

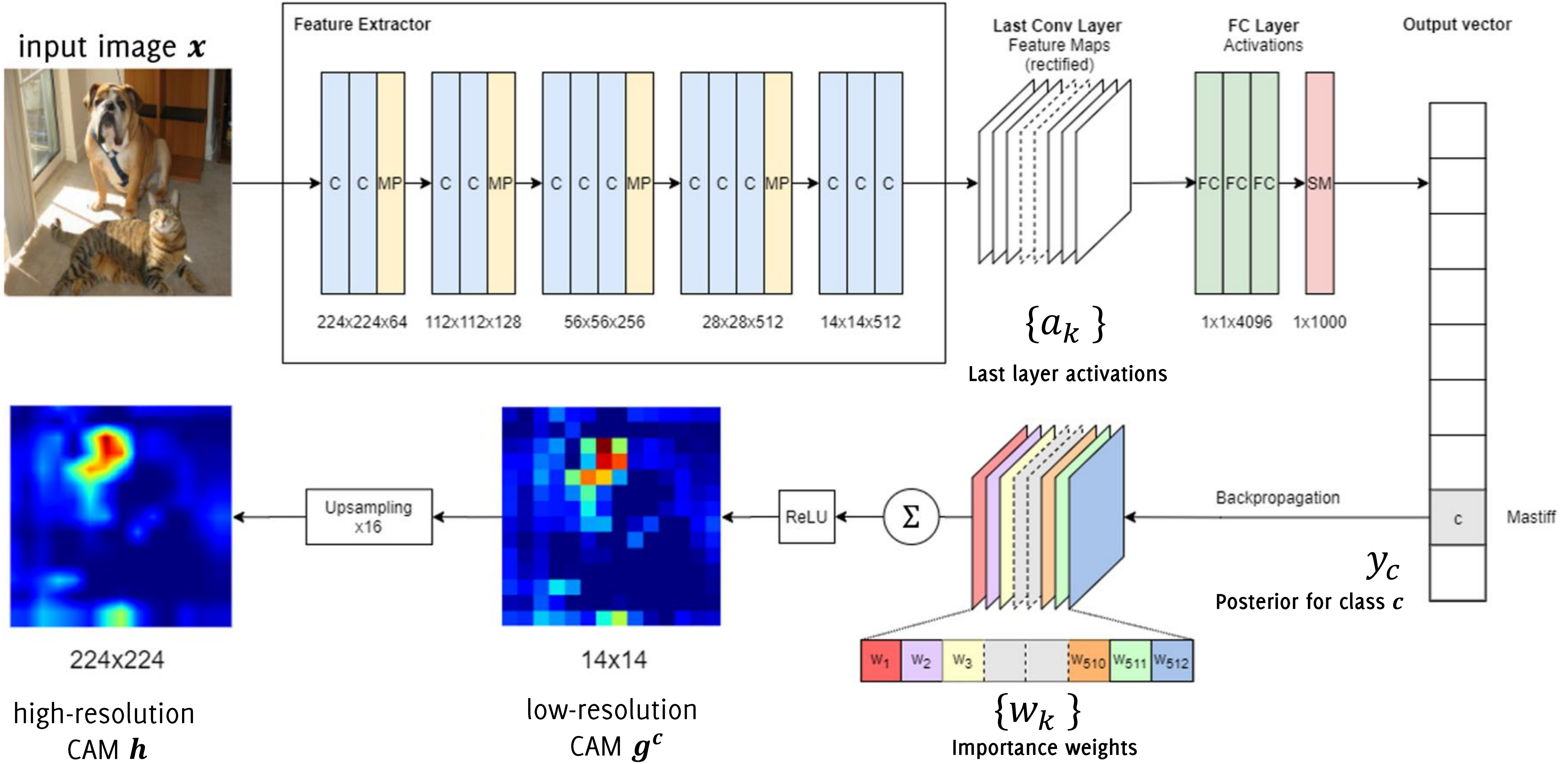
Highlight clever Hans phenomena

Correctly classified as “horse”



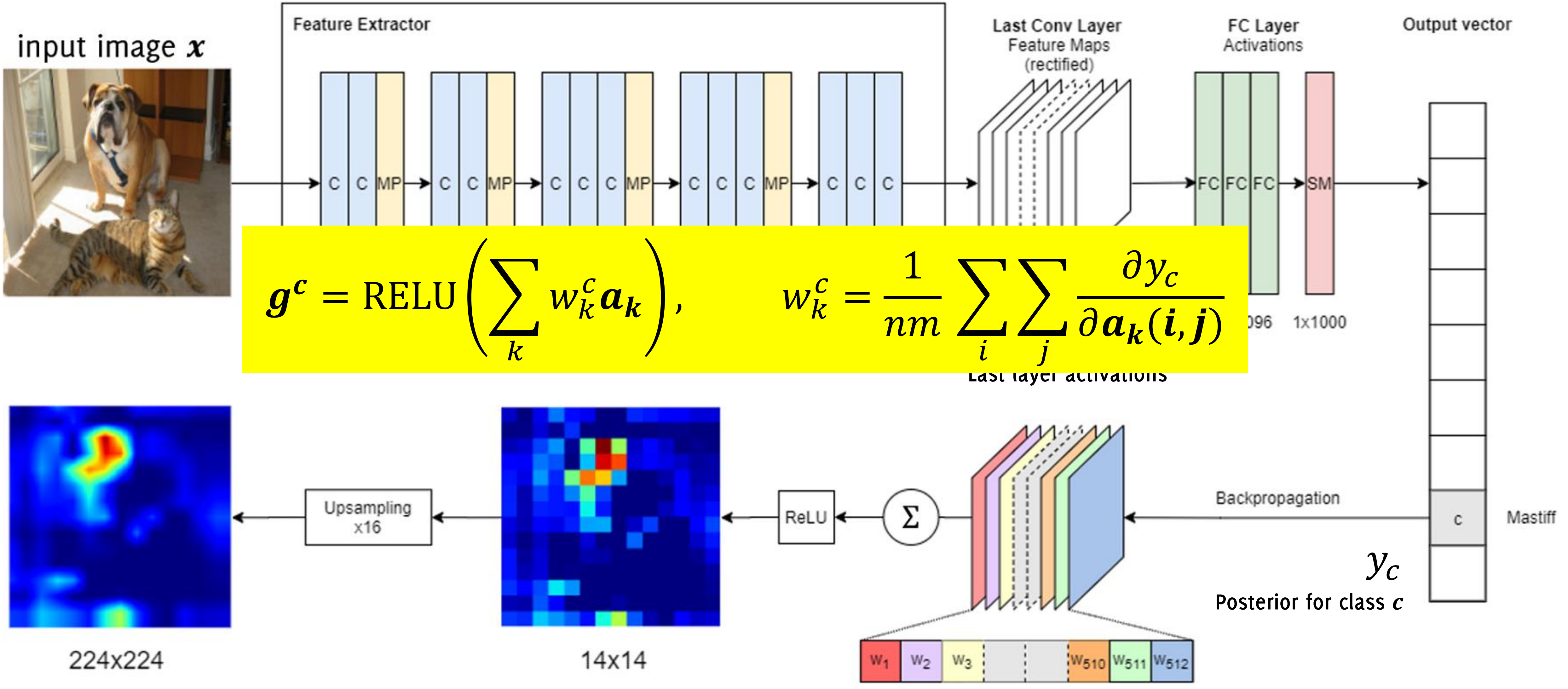
But for the wrong reason

Grad-CAM and CAM-based techniques



Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., & Batra, D. Grad-cam: Visual explanations from deep networks via gradient-based localization. *ICCV2017*

Grad-CAM and CAM-based techniques



$$\boldsymbol{g}^c = \mathrm{RELU}\left(\sum_k w_k^c \boldsymbol{a}_k\right), \qquad w_k^c = \frac{1}{nm}\sum_i \sum_j \frac{\partial y_c}{\partial \boldsymbol{a}_k(\boldsymbol{i},\boldsymbol{j})}$$

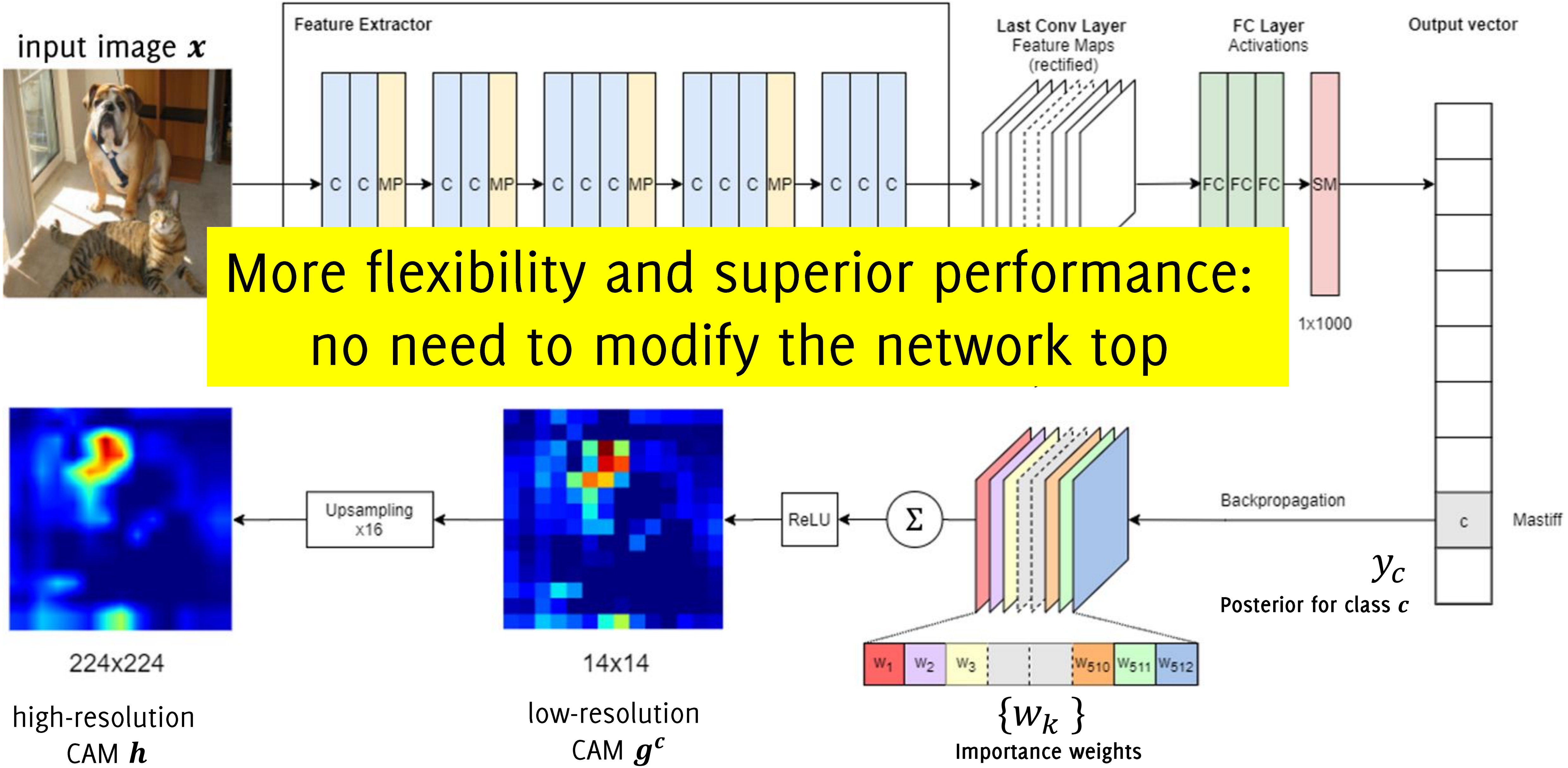
high-resolution CAM $\boldsymbol{h}$

low-resolution CAM $\boldsymbol{g}^c$

$\{w_k\}$ Importance weights

Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., & Batra, D. Grad-cam: Visual explanations from deep networks via gradient-based localization. *ICCV2017*

Grad-CAM and CAM-based techniques



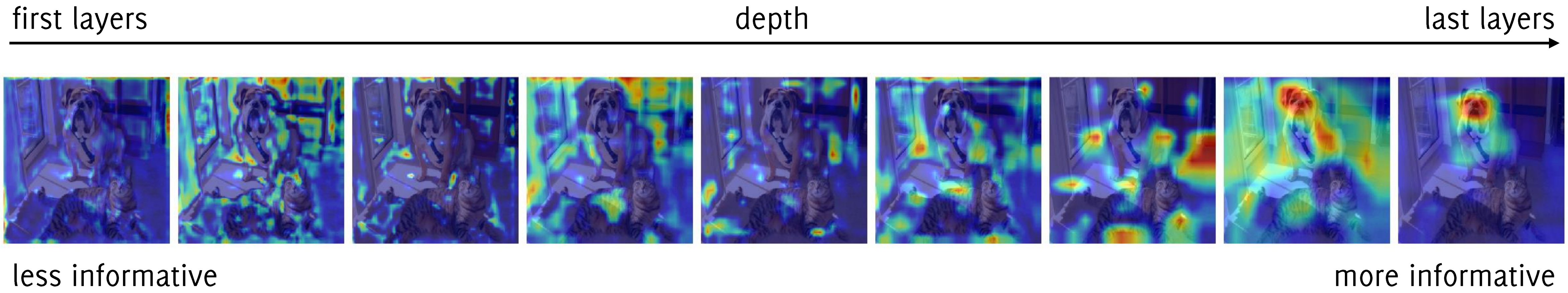
Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., & Batra, D. Grad-cam: Visual explanations from deep networks via gradient-based localization. *ICCV2017*

Heatmaps Desiderata

Should be **class discriminative**

Should **capture fine-grained details** (high-resolution)

- This is critical in many applications (e.g. medical/industrial imaging)

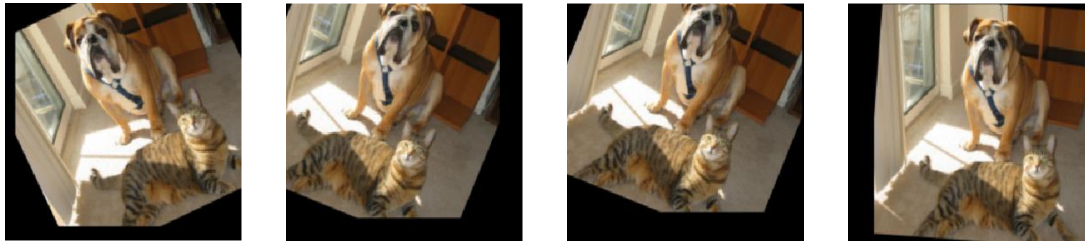


Augmented Grad-CAM

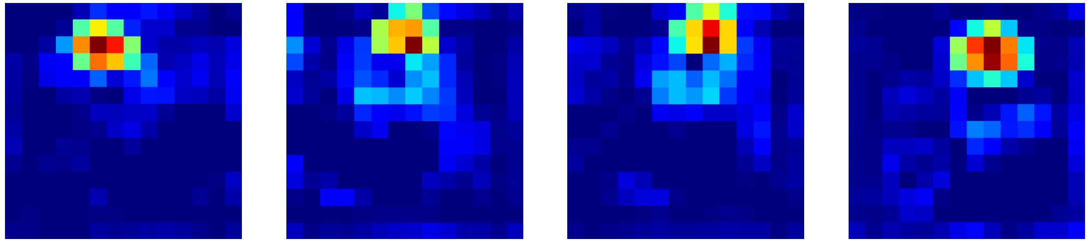
We consider the augmentation operator $\mathcal{A}_l: \mathbb{R}^{N\times M} \to \mathbb{R}^{N\times M}$, including random rotations and translations of the input image $\boldsymbol{x}$

Augmented Grad-CAM: increase heat-maps resolution through image augmentation

All the responses that the CNN generates to the **multiple augmented versions of the same input image** are very informative for reconstructing the high-resolution heat-map $\boldsymbol{h}$



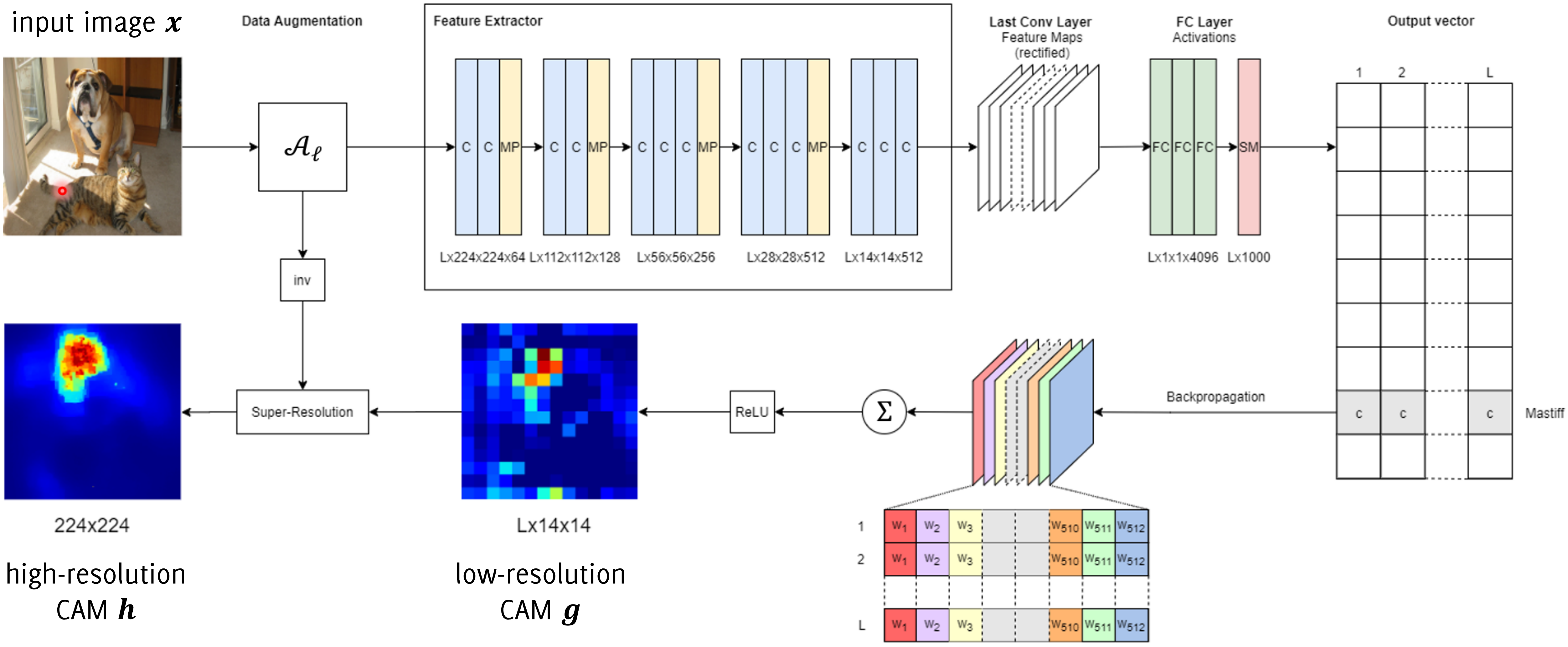
$x_1 = \mathcal{A}_1(x)$ $x_2 = \mathcal{A}_2(x)$ $x_3 = \mathcal{A}_3(x)$ $x_4 = \mathcal{A}_4(x)$



g_1 g_2 g_3 g_4

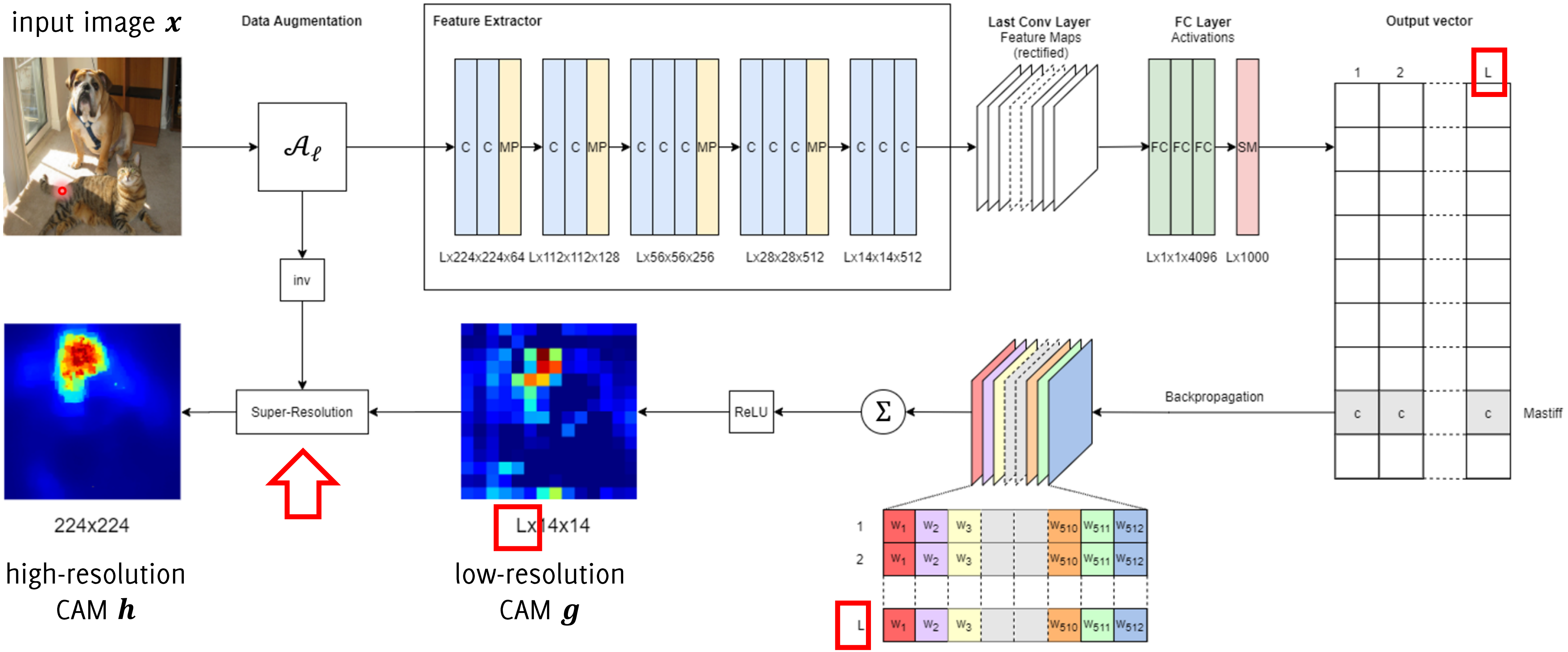
Morbidelli, P., Carrera, D., Rossi, B., Fragneto, P., & Boracchi, G. (2020, May). Augmented Grad-CAM: Heat-Maps Super Resolution Through Augmentation. In *ICASSP 2020*

Augmented Grad-CAM



Morbidelli, P., Carrera, D., Rossi, B., Fragneto, P., & Boracchi, G. (2020, May). Augmented Grad-CAM: Heat-Maps Super Resolution Through Augmentation. In *ICASSP 2020*

Augmented Grad-CAM

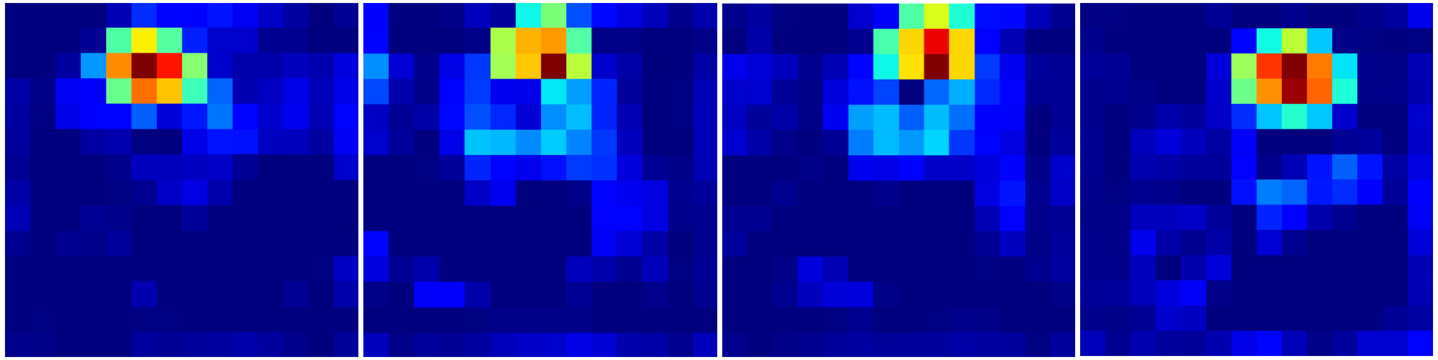


Morbidelli, P., Carrera, D., Rossi, B., Fragneto, P., & Boracchi, G. (2020, May). Augmented Grad-CAM: Heat-Maps Super Resolution Through Augmentation. In *ICASSP 2020*

The Super-Resolution Approach

We perform heat-map Super-Resolution (**SR**) by taking advantage of the information shared in multiple low-resolution heat-maps computed from the **same input under different – but known – transformations**

CNNs are in general invariant to roto-translations, in terms of predictions, but each $\boldsymbol{g}_\ell$ actually contains different information



g_1 g_2 g_3 g_4

General approach, our SR framework can be combined with any visualization tool (not only Grad-CAM)

Morbidelli, P., Carrera, D., Rossi, B., Fragneto, P., & Boracchi, G "Augmented Grad-CAM: Heat-Maps Super Resolution Through Augmentation". ICASSP 2020

The Super-Resolution Formulation

We model heat-maps computed by Grad-CAM as the result of linear **downsampling operator** $\mathcal{D} : \mathbb{R}^{N\times M} \rightarrow \mathbb{R}^{n\times m}$ applied to an unknown high resolution heat-map $\boldsymbol{h}$

Then, heat-map superresolution consists in solving an inverse problem

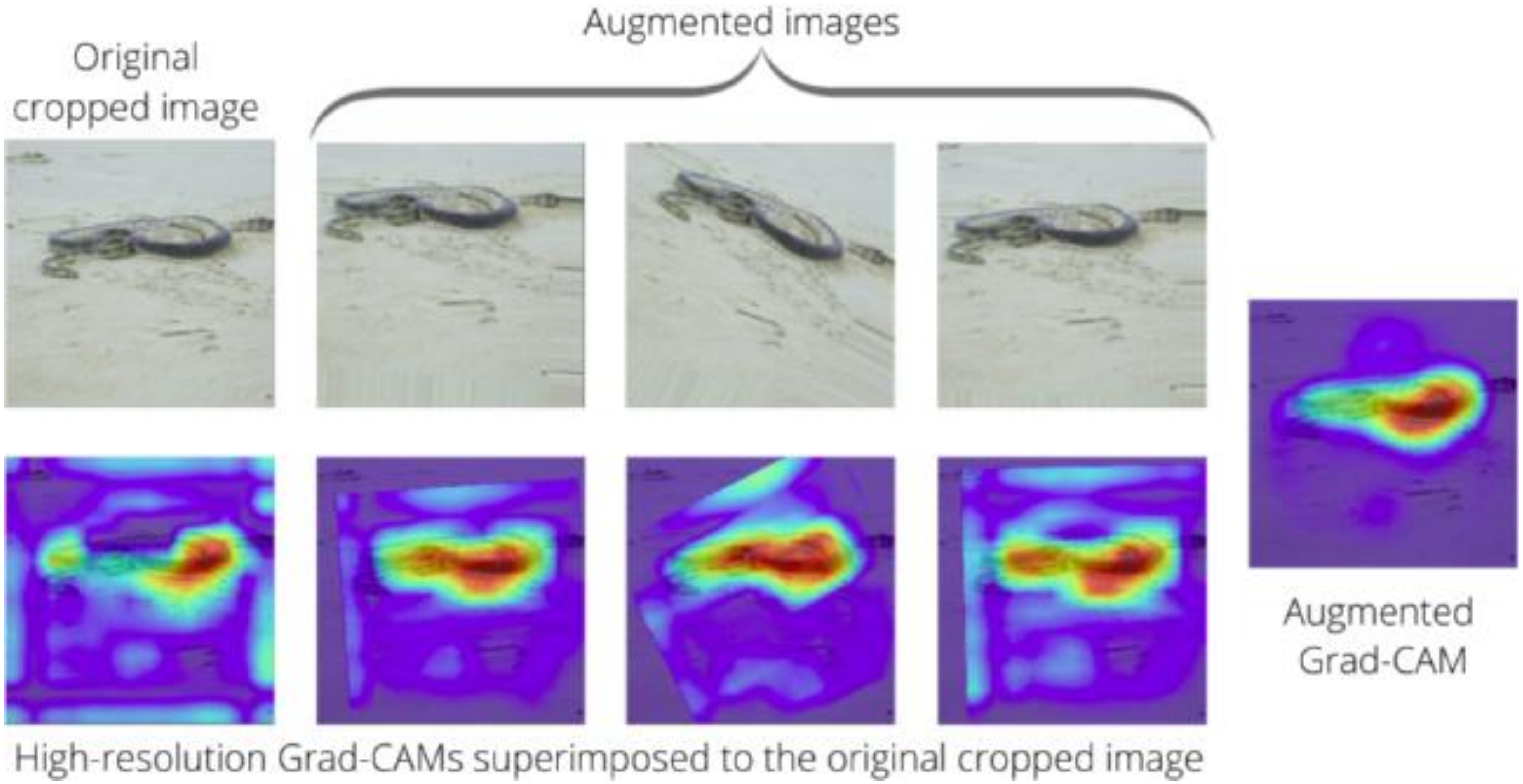
$$\underset{h}{\operatorname{argmin}} \frac{1}{2}\sum_{l=1}^{L} \|\mathcal{D}\mathcal{A}_{\ell}h - g_{\ell}\|_2^2 + \lambda TV_{\ell_1}(h) + \frac{\mu}{2}\|h\|_2^2 \quad (1)$$

TV_{ℓ_1}: Anistropic Total Variation regularization is used to preserve the edges in the target heat-map (high-resolution)

$$TV_{\ell_1}(\boldsymbol{h}) = \Sigma_{i,j}\|\partial_x \boldsymbol{h}(i,j)\| + \left\|\partial_y \boldsymbol{h}(i,j)\right\| \quad (2)$$

This is solved through Subgradient Descent since the function is convex and non-smooth

Morbidelli, P., Carrera, D., Rossi, B., Fragneto, P., & Boracchi, G "Augmented Grad-CAM: Heat-Maps Super Resolution Through Augmentation". ICASSP 2020



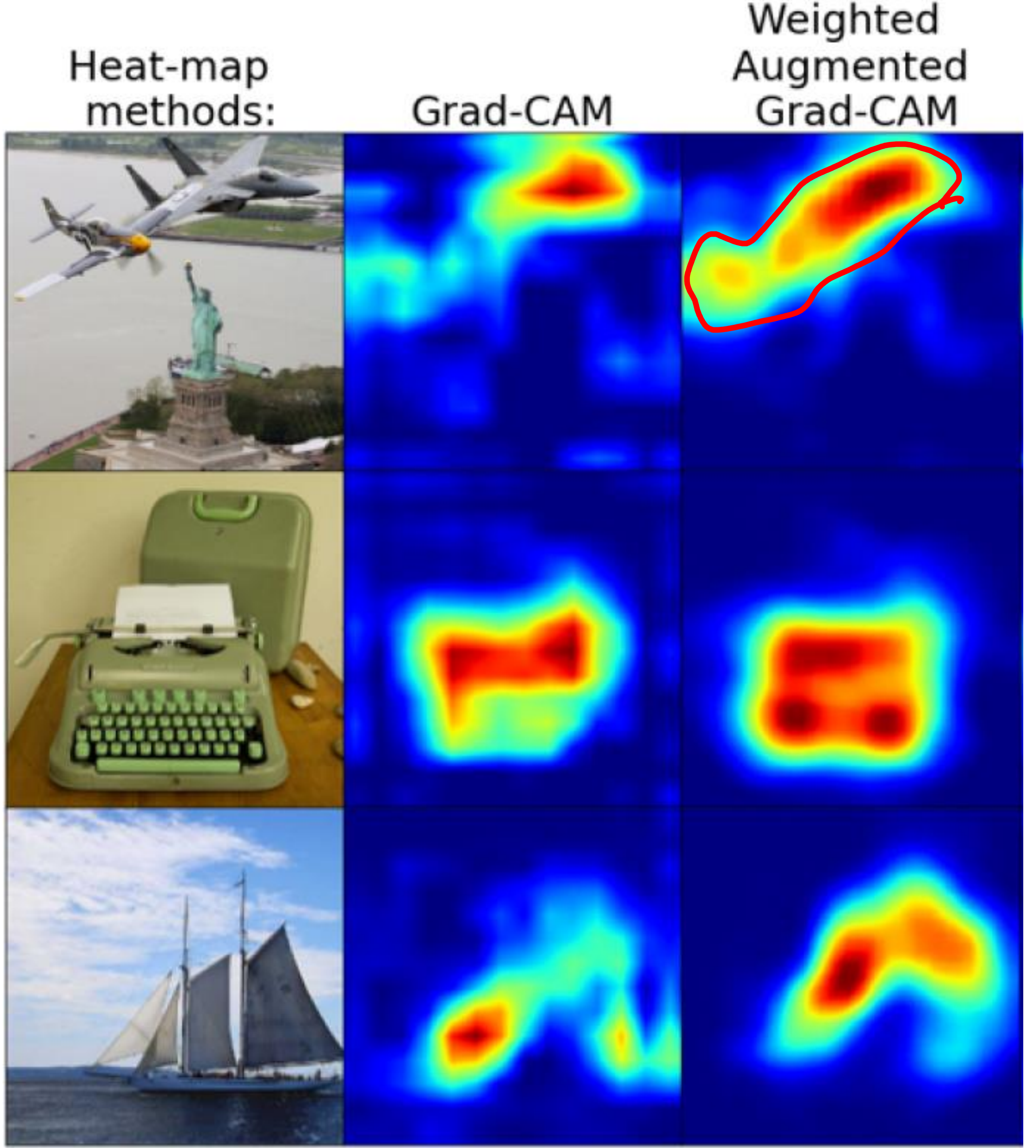
Morbidelli, P., Carrera, D., Rossi, B., Fragneto, P., & Boracchi, G "Augmented Grad-CAM: Heat-Maps Super Resolution Through Augmentation". ICASSP 2020

Augmented Grad-CAM («Mastiff» class)

(a) Grad-CAM. (b) Augmented Grad-CAM.

Morbidelli, P., Carrera, D., Rossi, B., Fragneto, P., & Boracchi, G. (2020, May). Augmented Grad-CAM: Heat-Maps Super Resolution Through Augmentation. In *ICASSP 2020*

Augmented Grad-CAM



Other Gradient-based Saliency Maps

Grad-CAM++ : Same formulation of Grad-CAM, but weights are computed by **higher-order derivatives of the class score** with respect to the feature maps. Increases the localization accuracy of the heat-maps in presence of multiple occurrence of the same object in the image.

Sharpen Focus: highlights only the pixels where the gradients are positive.

$$w_k^c = \frac{1}{nm} \sum_i \sum_j \mathrm{RELU}\left(\frac{\partial y_c}{\partial \boldsymbol{a_k}(\boldsymbol{i}, \boldsymbol{j})}\right)$$

Smooth Grad-CAM++: it averages multiple heat-maps corresponding to noisy versions of the same input image.

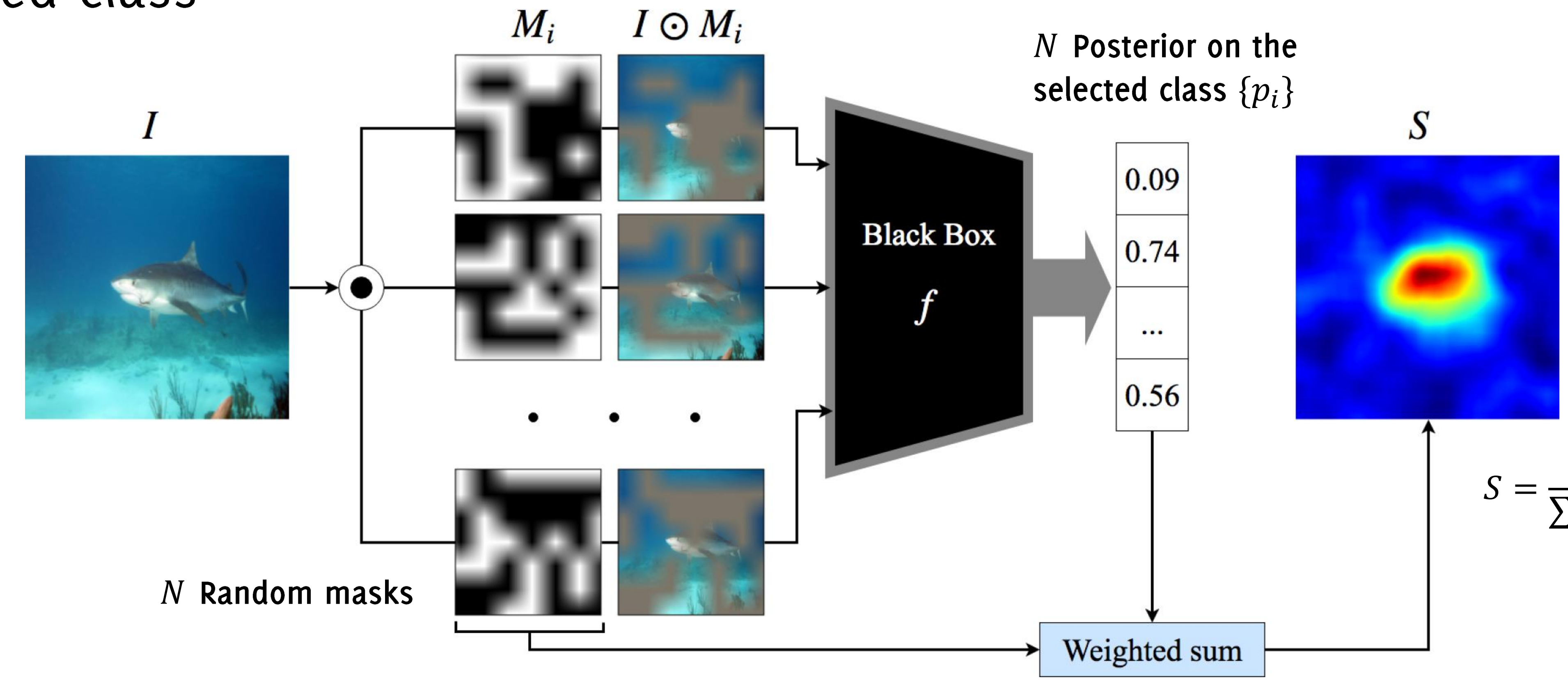
A. Chattopadhay, A. Sarkar, P. Howlader, and V. N. Balasubramanian, "Grad-CAM++: Generalized gradient-based visual explanations for deep convolutional networks," WACV, 2018.

D. Omeiza, S. Speakman, C. Cintas, and K. Weldermariam, "Smoothgrad-CAM++: An enhanced inference level visualization technique for deep convolutional neural network models"

Other Perturbation-based Saliency Maps

Idea: Perturb the input image and assess how the class score changes.

RISE: use random perturbations to identify the most influential regions for a selected class



$$S = \frac{1}{\sum \boldsymbol{M_i}} .* \sum_{i=1}^{N} p_i * \boldsymbol{M_i}$$

Petsiuk, Vitali, Abir Das, and Kate Saenko. "Rise: Randomized input sampling for explanation of black-box models." BMVC (2018).

Limitations of Saliency Maps

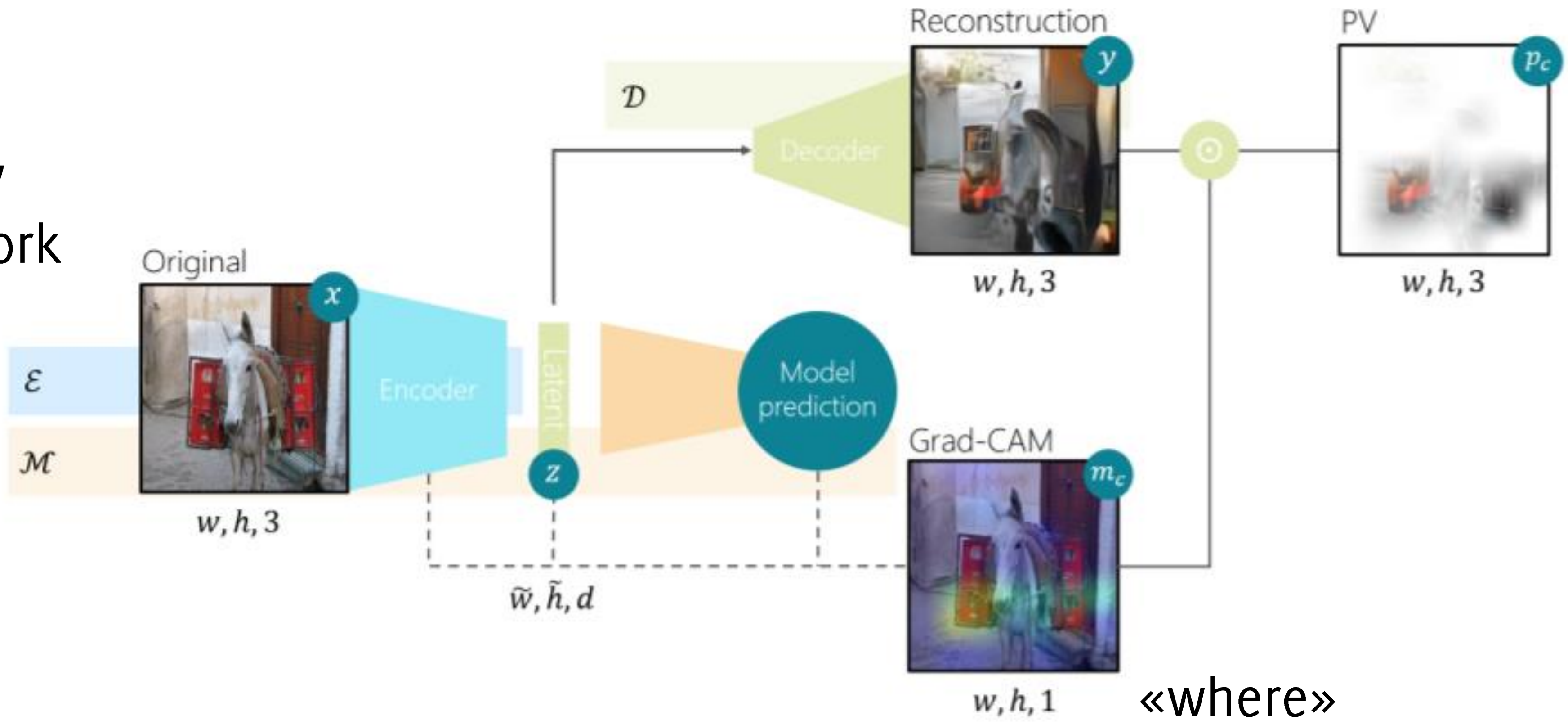
Figure 1: Based on saliency maps it is unclear why this image is labelled as a *cat* rather than a *laundry basket*. Grad-CAM [27] explanations are essentially the same for both classes.

L. Giulivi M.J. Carman G. Boracchi "Perception Visualization: Seeing Through the Eyes of a DNN" BMVC 2021

Perception Visualization

Perception Visualization: provides explanations by exploiting a neural network to invert latent representations

«what»



«where»

Figure 3: An overview of our method and interactions between the models involved. Encoder $\mathcal{E}$ is a truncation of the model $\mathcal{M}$ which we want to explain, decoder $\mathcal{D}$ is trained to reconstruct the encoder's latent representations. From these, we compute Grad-CAM saliency maps and reconstructions, which are then combined to obtain PV.

L. Giulivi M.J. Carman G. Boracchi "Perception Visualization: Seeing Through the Eyes of a DNN" BMVC 2021

Perception Visualization

Misclassified as “boat”

«where»

Saliency doesn’t say much

«where and what»

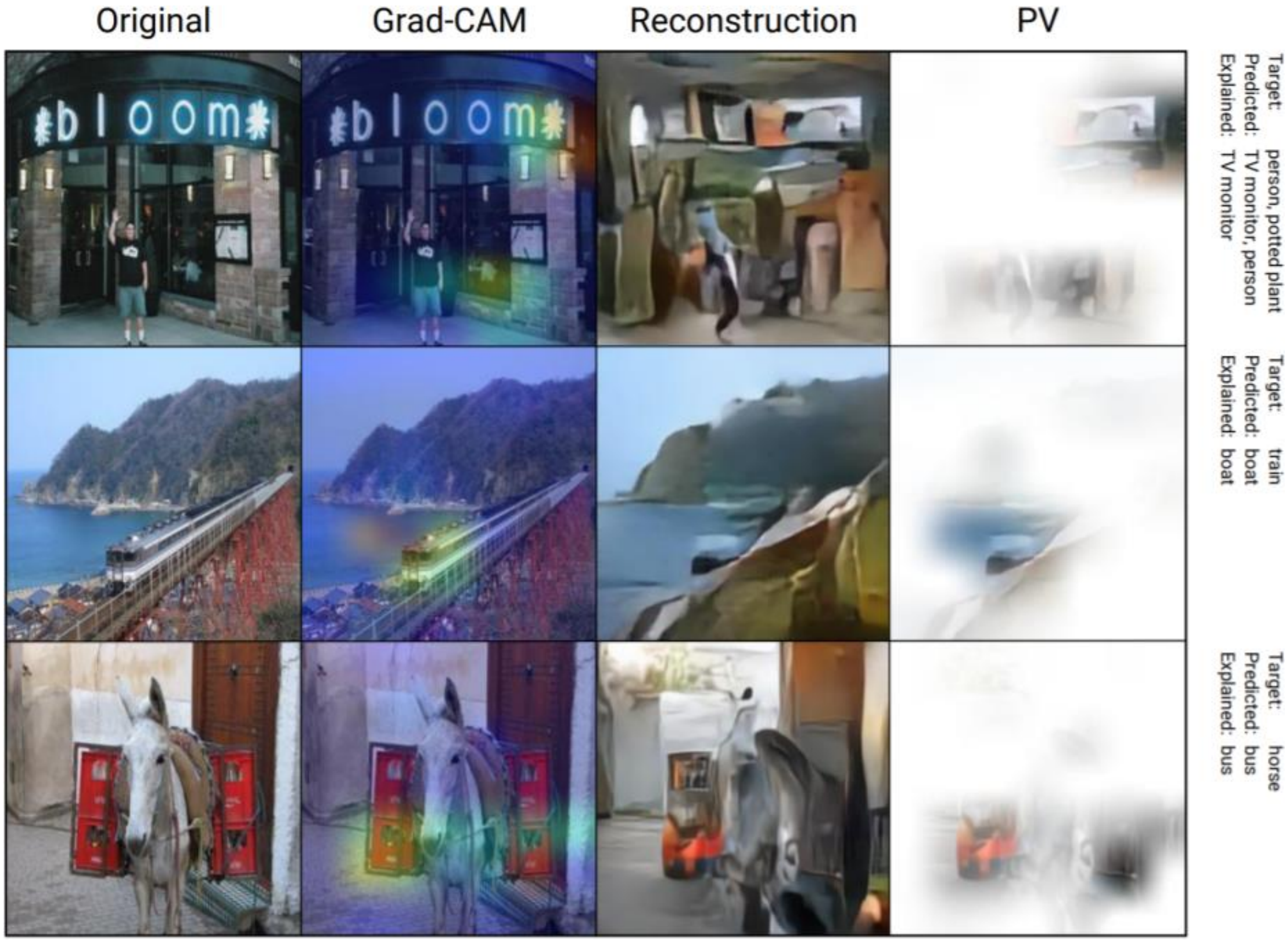
PV shows why

L. Giulivi M.J. Carman G. Boracchi “Perception Visualization: Seeing Through the Eyes of a DNN” BMVC 2021

Perception Visualization

Give better insight on the model's functioning than what was previously achievable using only saliency maps.

A study on circa 100 subjects shows that PV is able to **help respondents better determine the predicted class** in cases where the **model had made an error**



L. Giulivi M.J. Carman G. Boracchi "Perception Visualization: Seeing Through the Eyes of a DNN" BMVC 2021