

14/11/2024

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

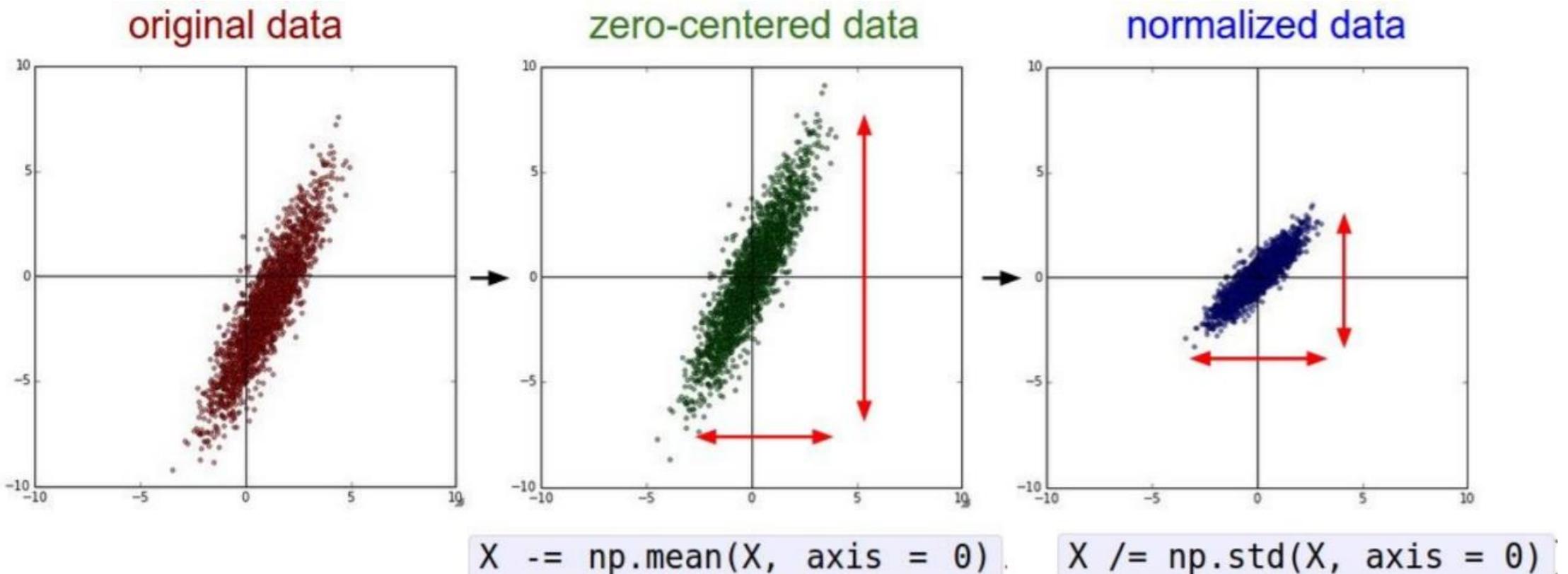
In general, normalization is useful in gradient-based optimizers.

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

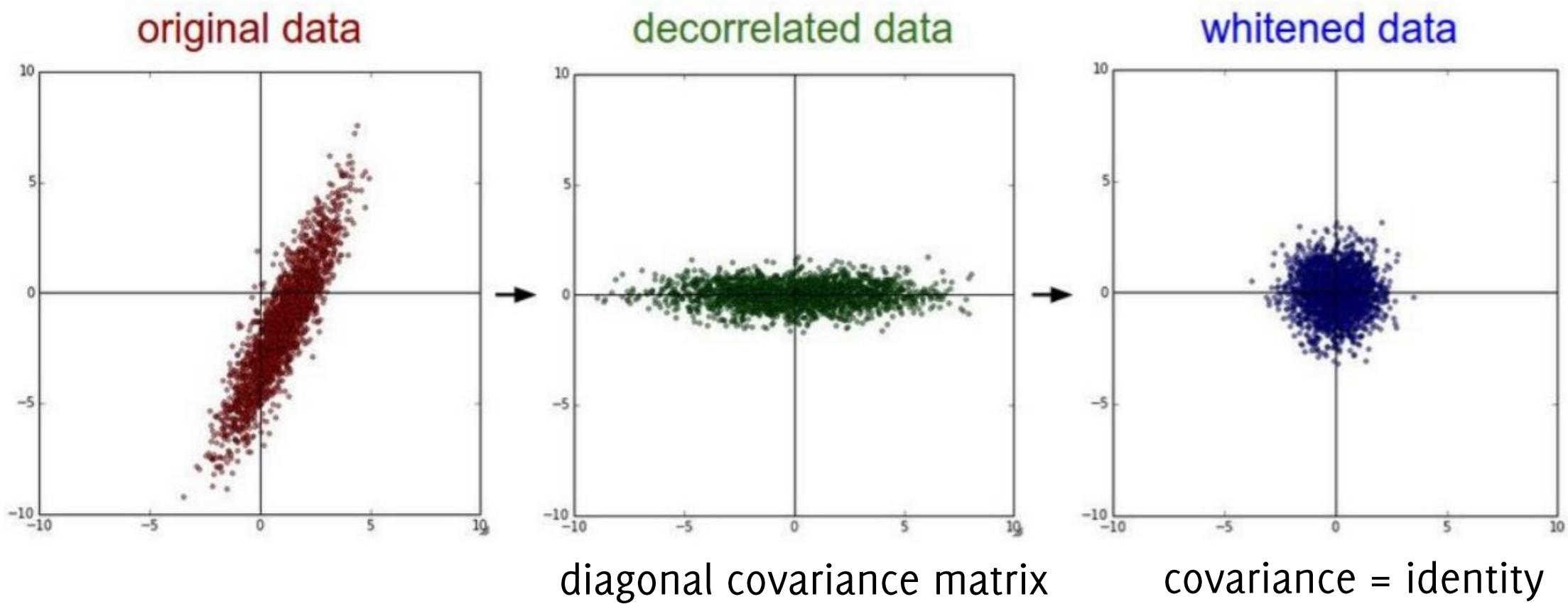
There are several options

There are different form of preprocessing



PCA – based preprocessing

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



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.



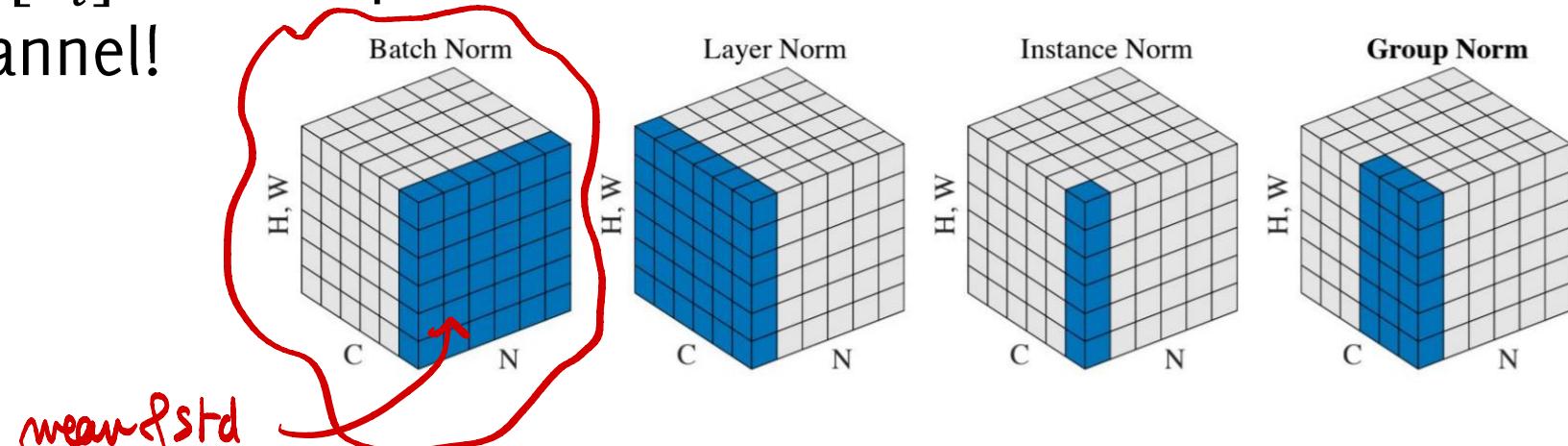
When you do Transfer Learning : no need to ask yourself "how should I normalize" } the pre-training set is given WITH the model!

Batch Normalization

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

$$x'_i = \frac{x_i - E[x_i]}{\sqrt{\text{var}[x_i]}}$$

Where $E[x_i]$ and $\sqrt{\text{var}[x_i]}$ 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

$$x'_i = \frac{x_i - E[x_i]}{\sqrt{\text{var}[x_i]}}$$

a further a parametric transformation

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

batch normalization
adds 2 parameters.

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

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

Rmk: estimates $E[x_i]$ and $\sqrt{\text{var}[x_i]}$ 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: $\mathcal{B} = \{x_1 \dots m\}$;

Parameters to be learned: γ, β

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

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

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

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

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

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

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.

Fully connected
↓

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

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!

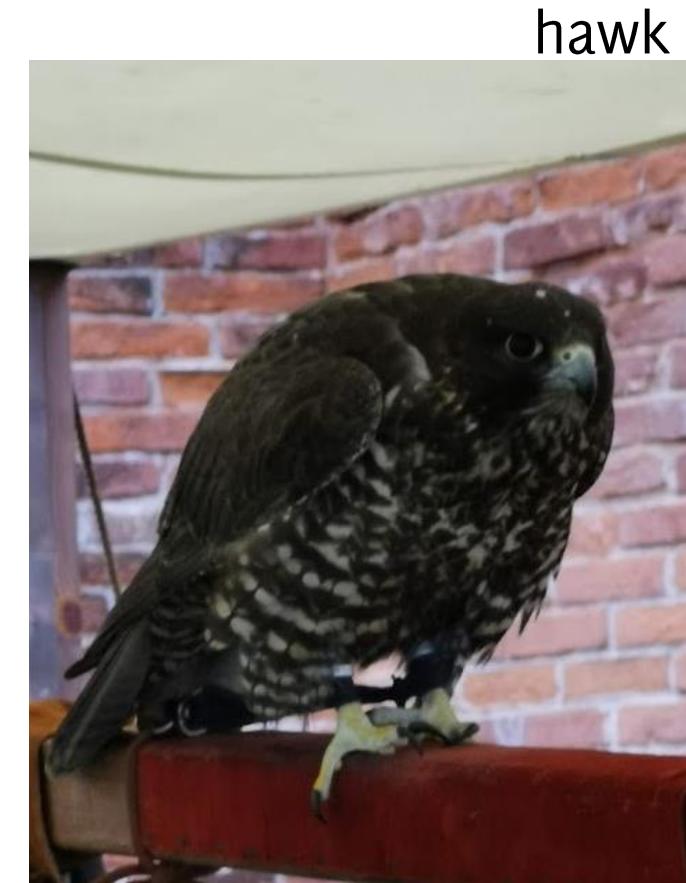
Localization

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



The Classification and Localization Tasks

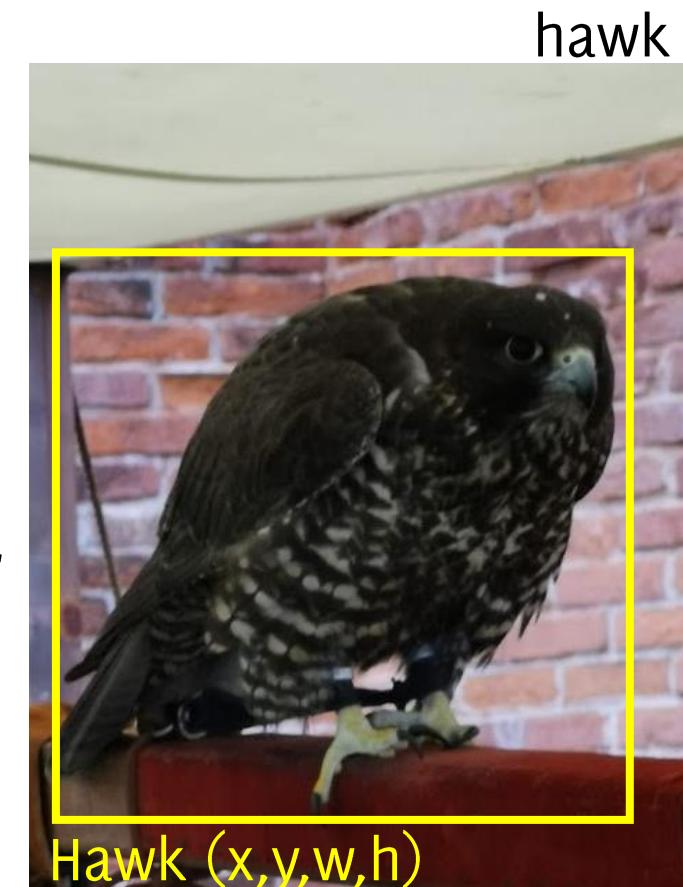
The input image contains a single relevant object to be classified in a fixed set of categories

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)



The Classification and Localization Tasks

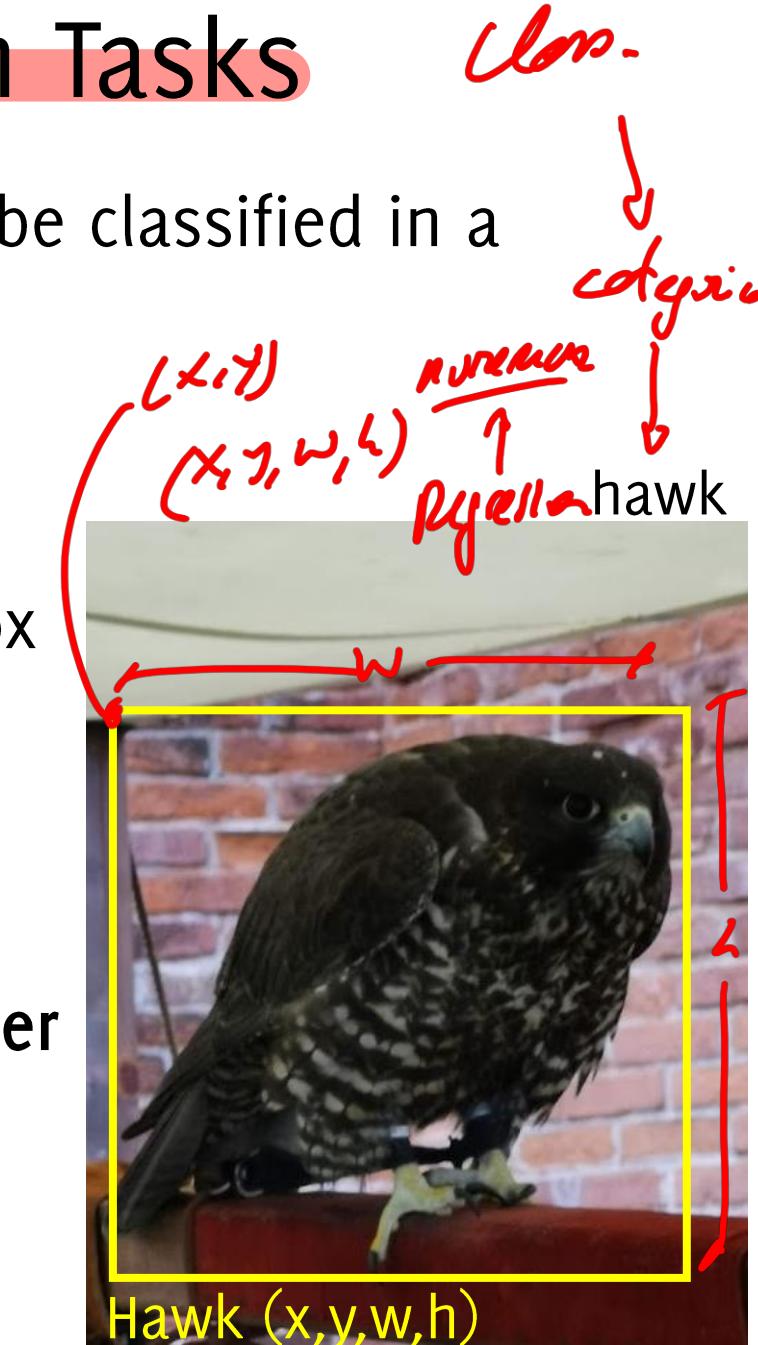
The input image contains a single relevant object to be classified in a fixed set of categories

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)



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

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

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

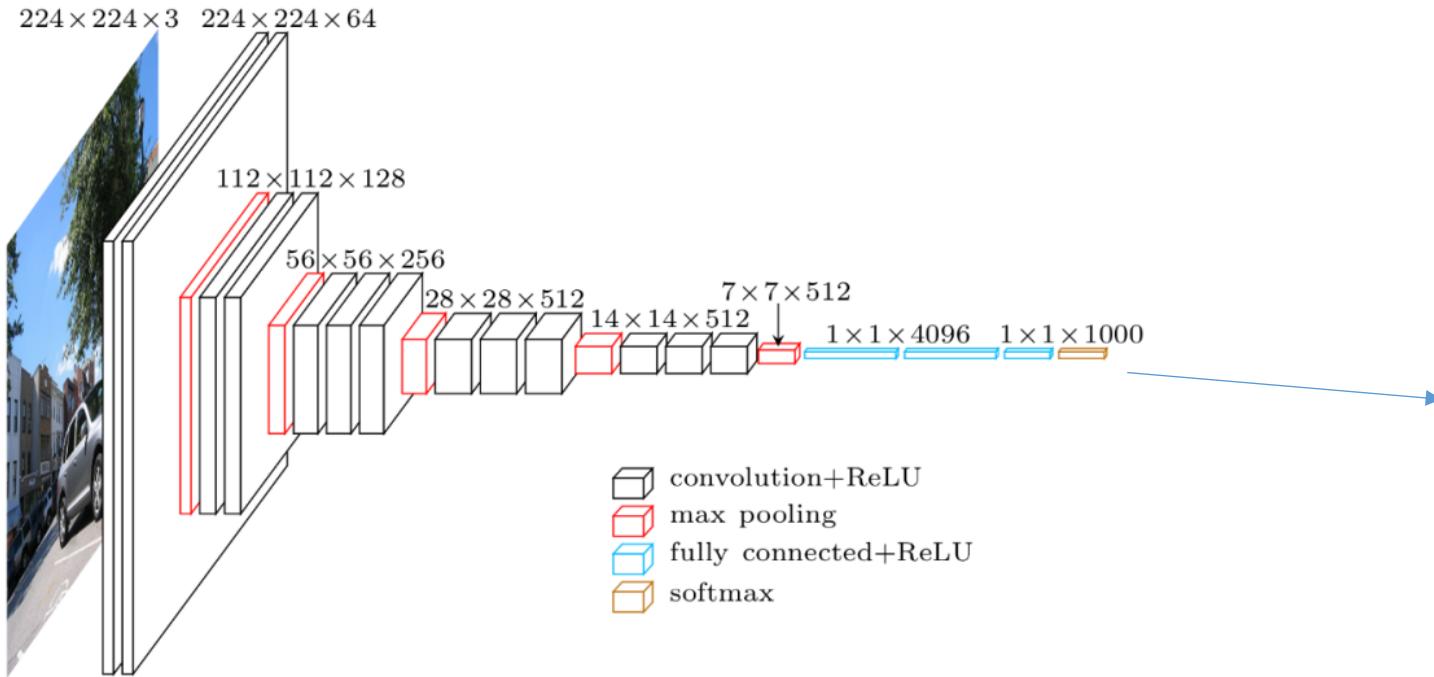
regressor_model = tfk.Model(inputs=inputs, outputs=output, name='re
gressor_model')

# Compile the model using Mean Squared Error (MSE) as loss

regressor_model.compile(loss=tfk.losses.MeanSquaredError(),
optimizer=tfk.optimizers.Adam())
```

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
 ℓ^2, ℓ^1, \dots bounding box.

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

Classification and Localization, the problem

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

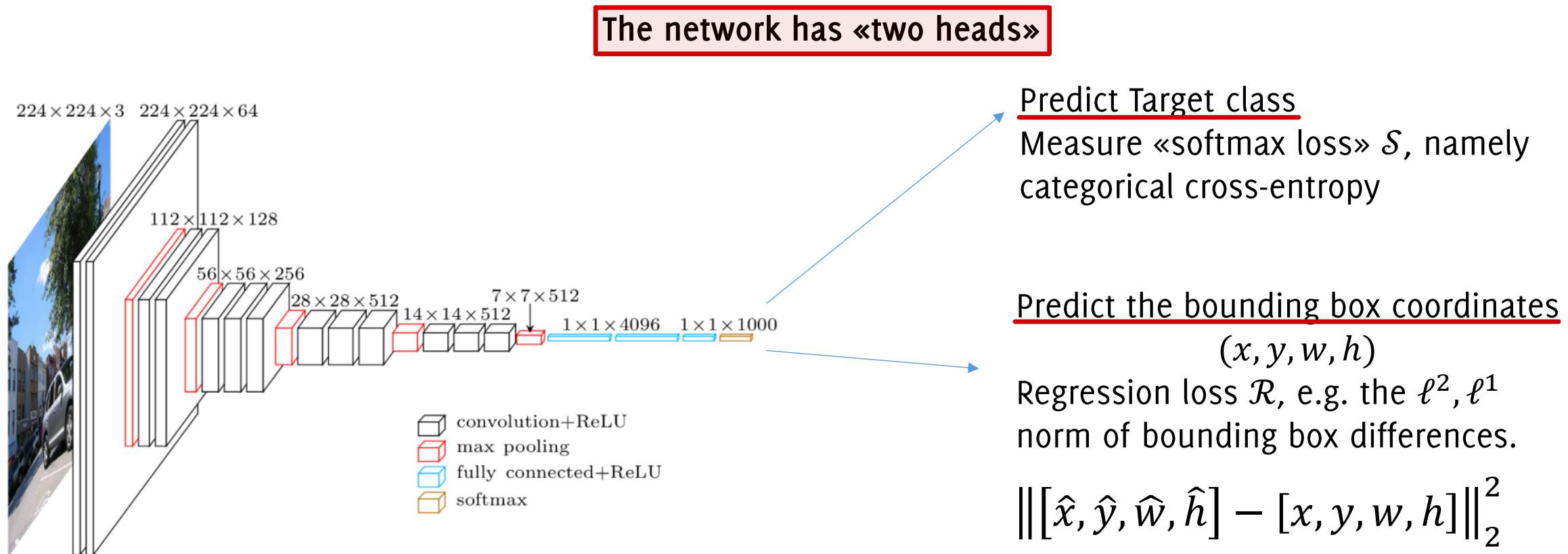
- a label l from a fixed set of categories
 $\Lambda = \{"wheel", "cars", \dots, "castle", "baboon"\}$
- the coordinates (x, y, h, w) of the bounding box enclosing that object

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

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



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 α 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.

Sum the 2 losses

Programming Solution

```
# Define inputs and add MobileNet as a feature extractor
inputs = tfk.Input(shape=train_images.shape[1:])
x = mobile(inputs)

# Add a classification head for object (binary/multiclass) classification
class_outputs = tfkl.Dense(2, activation='softmax', name='classifier')(x)

# Add localization head for bounding box prediction
box_outputs = tfkl.Dense(4, activation='linear', name='localizer')(x)

# Create the model connecting inputs to classification and localisation
outputs (it is enough to put the output in a list)
object_localization_model = tfk.Model(inputs=inputs,
outputs=[class_outputs, box_outputs], name='object_localization_model')

# Compile the model with categorical crossentropy and mean squared error
losses (make sure the two losses are in the same order as the heads.
The i-th item of the output list is assessed using the i-th item of the
loss list
object_localization_model.compile(loss=[tfk.losses.CategoricalCrossentropy(
), tfk.losses.MeanSquaredError()], optimizer=tfk.optimizers.Adam())
```

“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(),  
optimizer=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

This image is licensed under CC-BY 2.0

Even if not visible, we will have a prediction. {Imagine there is a car in front:} the output will give an answer! 14 locations to estimate 28 output neurons

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

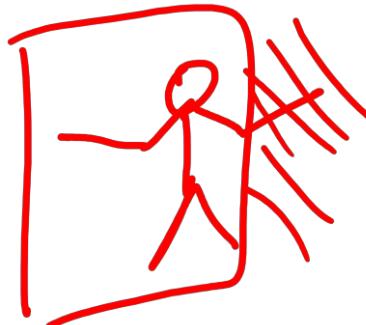
Pose is defined as a vector of k joints location for the human body, possibly normalized w.r.t. the bounding box enclosing the human.

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

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



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

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).

Saliency Maps and Weakly-Supervised Localization

... Global Averaging Pooling Revisited

... visualizing what matters most for CNN predictions

Weak Supervision

In supervised learning a model \mathcal{M} performing inference in Y

$$\mathcal{M}: X \rightarrow Y$$

requires a training set $TR \subset X \times Y$, namely training couples are of the same type as classifier input-output.



For some tasks (e.g. segmentation) these type of annotations are very expensive to gather.

gathering label can become an issue.

Weak Supervision:

Obtain a model able to solve a task in Y , but using labels that are easier to gather in a different domain K . Therefore, \mathcal{M} after training perform inference as

$$\mathcal{M}: X \rightarrow Y$$

but it is trained using $TR \subset X \times K$, where $K \neq Y$.

Weakly Supervised Localization

Weakly Supervised Localization: Perform localization

$$\mathcal{M}: X \rightarrow \mathbb{R}^4$$

without images training images annotated bounding box.

The training set is typically annotated for classification, thus

$$TR \subset X \times \Lambda$$

with image-label pairs $\{(I, \ell)\}$ whith no localization information provided.

Basically, you train a classifier and you get also localization estimates.



This CVPR paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the version available on IEEE Xplore.

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 (as shown in the NiN paper)

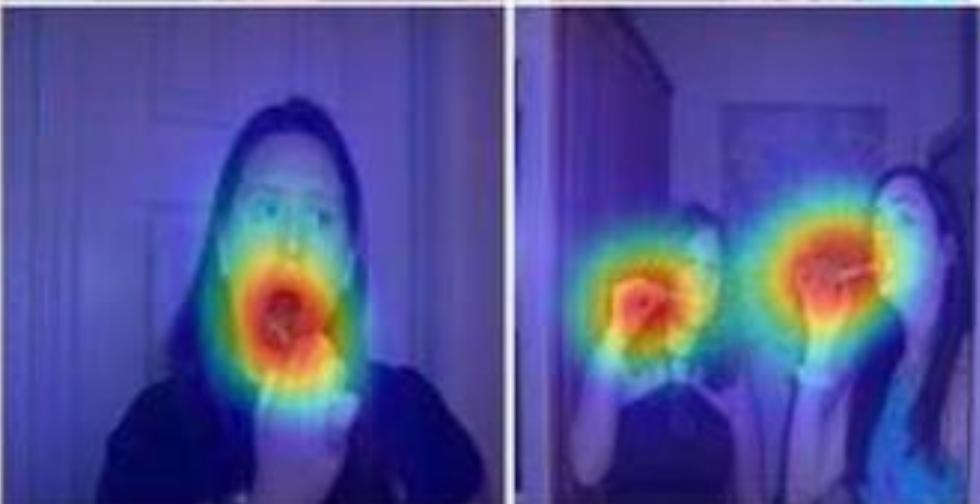
↑ *Network in Network*.

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.

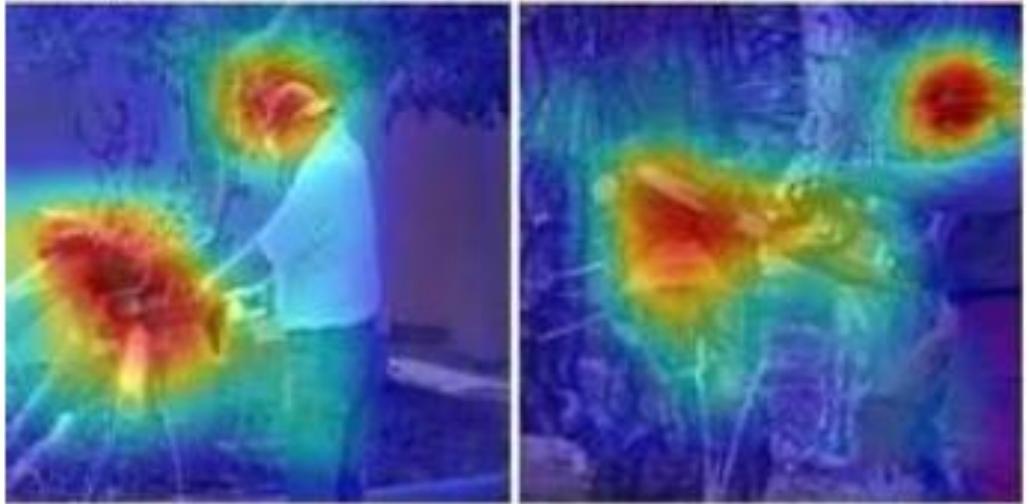
A CNN trained on object categorization is successfully able to localize the discriminative regions for action classification as the objects that the humans are interacting with rather than the humans themselves

Class Activation Mapping

Brushing teeth



Cutting trees



Class Activation Mapping (CAM)

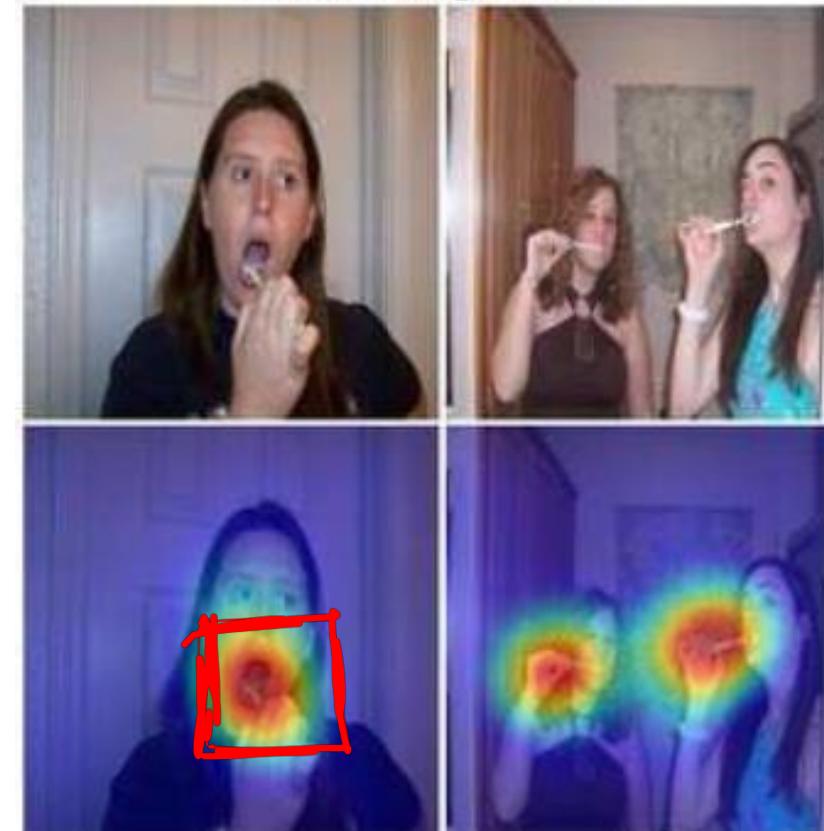
Identifying exactly which regions of an image are being used for discrimination.



CAM are very easy to compute. It just requires:

- A classifier trained with a GAP layer
- FC layer after the GAP
- A minor tweak to obtain saliency maps

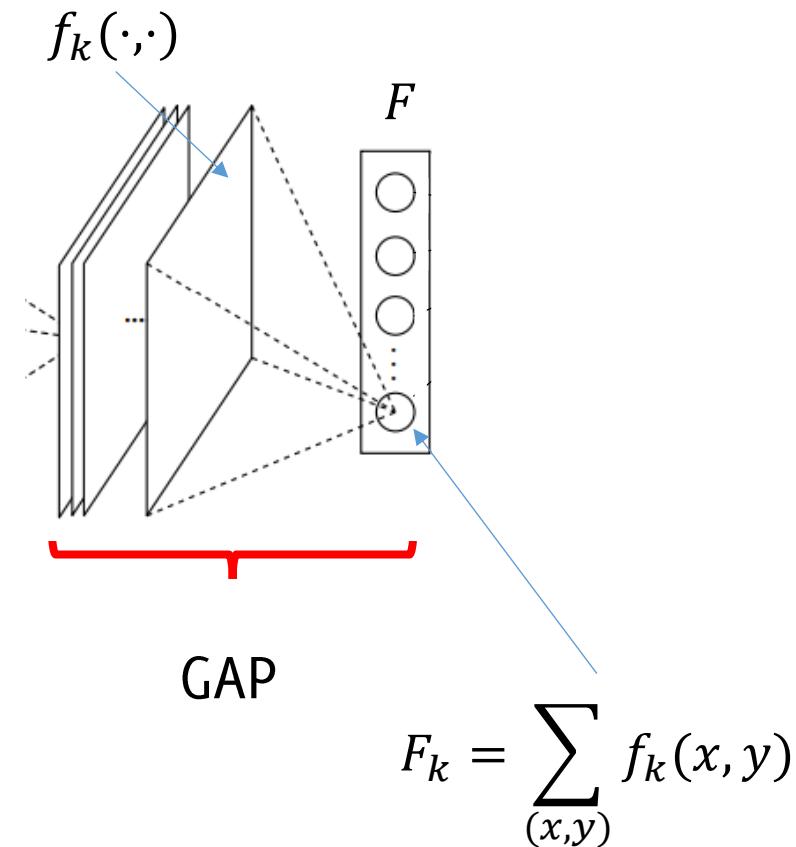
Brushing teeth



The Global Averaging Pooling (GAP) Layer

Assume you have trained a CNN architecture having at the end of the convolutional block:

- n feature maps $f_k(\cdot, \cdot)$ having resolution “similar” to the input image
- a GAP layer that computes n averages F_k , $k = 1, \dots, n$



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

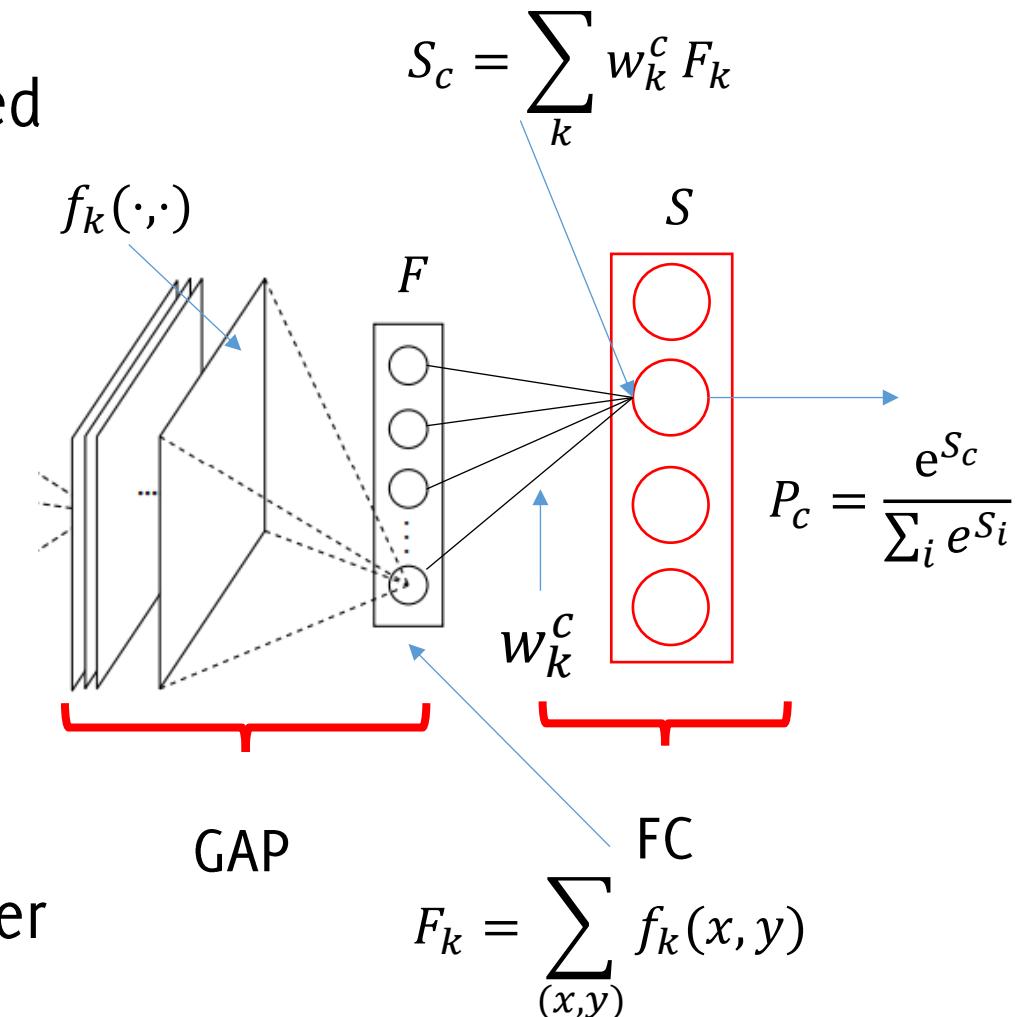
Then, the class probability P_c via soft-max (class c)

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



The Global Averaging Pooling (GAP) Layer

Perspective change in score interpretation

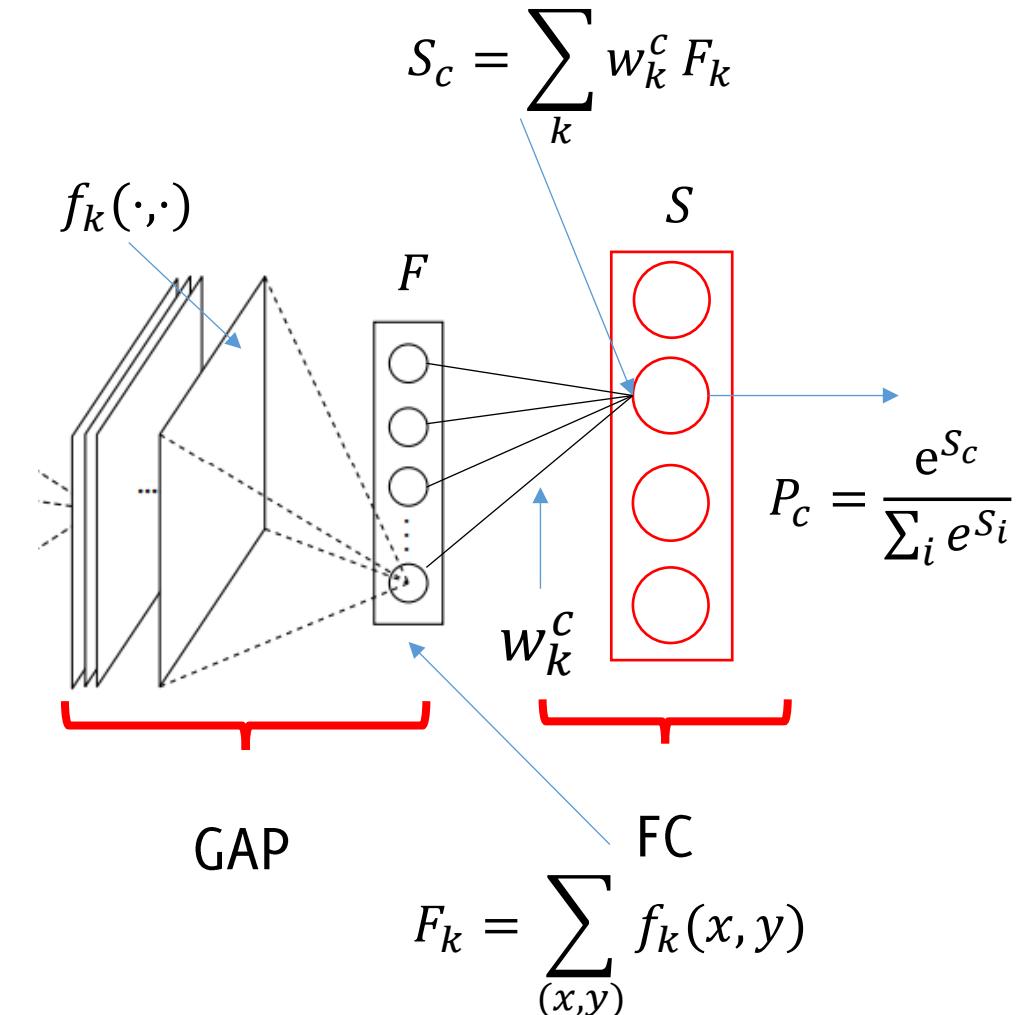
$$S_c = \sum_k w_k^c \sum_{x,y} f_k(x,y) = \sum_{x,y} \sum_k w_k^c f_k(x,y)$$

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 FC layer, the number of channels of the last convolutional layer can differ from the number of classes



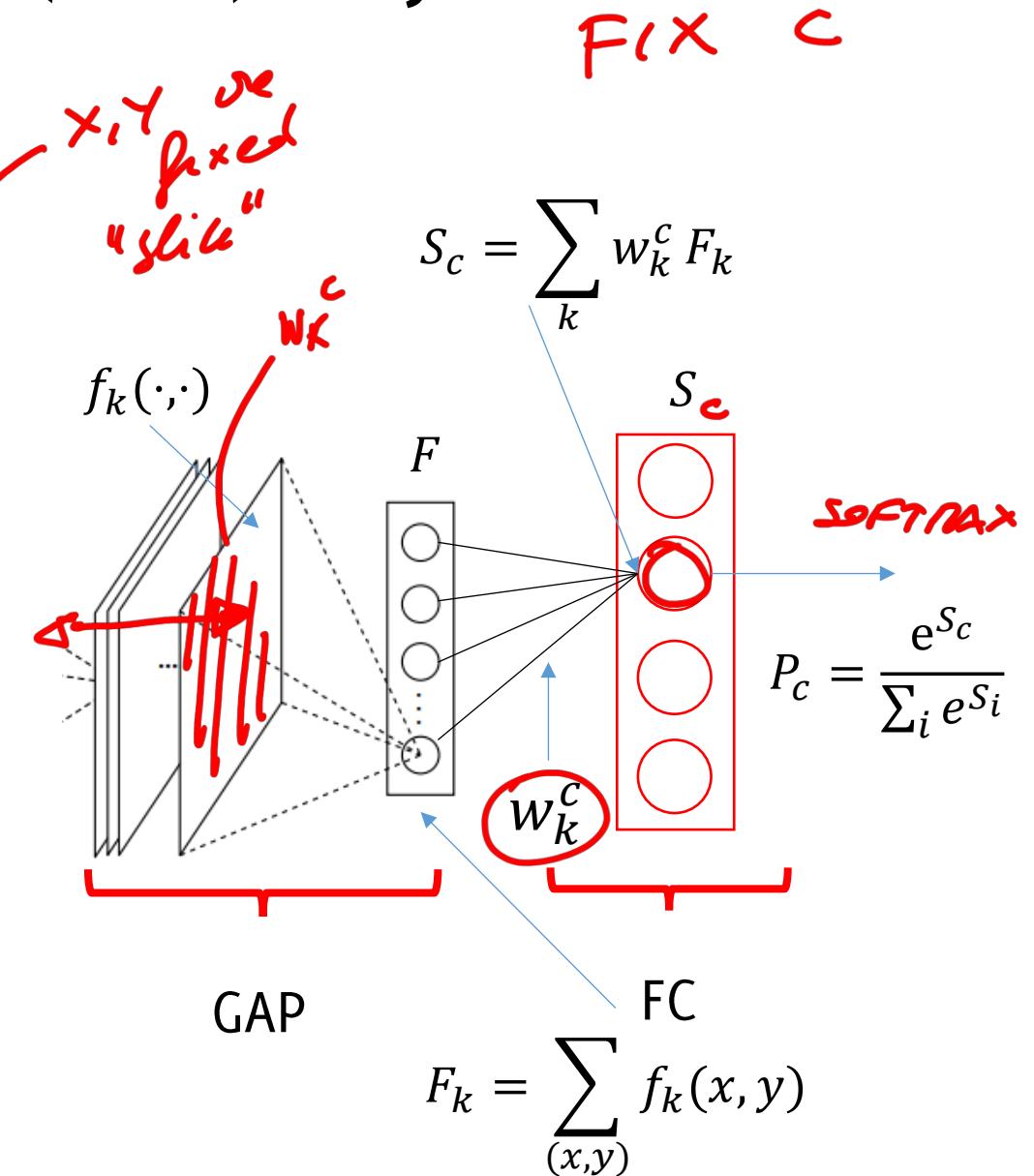
The Global Averaging Pooling (GAP) Layer

$$S_c = \sum_k w_k^c \left(\sum_{x,y} f_k(x,y) \right) = \sum_{x,y} \left[\sum_k w_k^c f_k(x,y) \right]$$

CAM is defined as

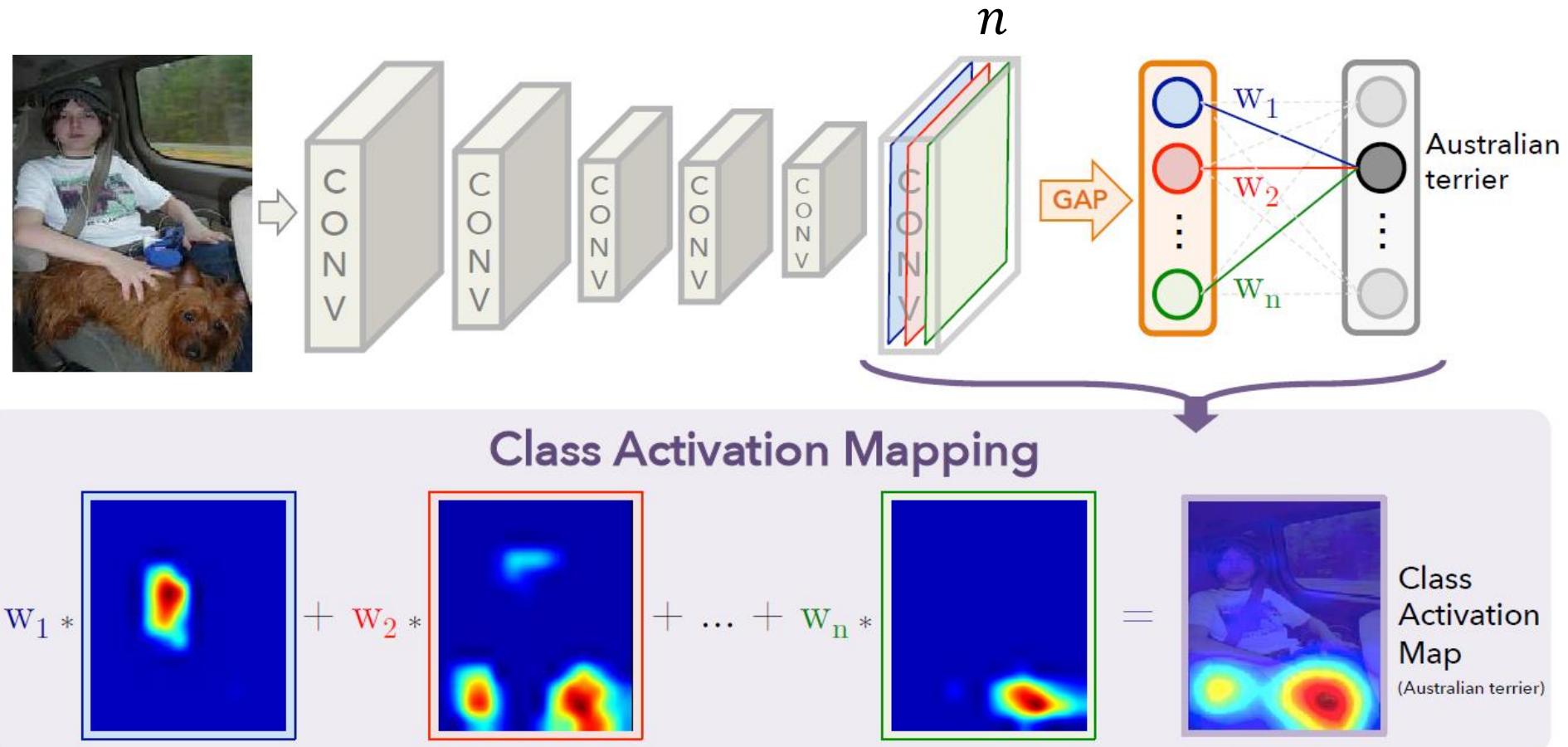
$$M_c(x,y) = \sum_k w_k^c f_k(x,y)$$

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

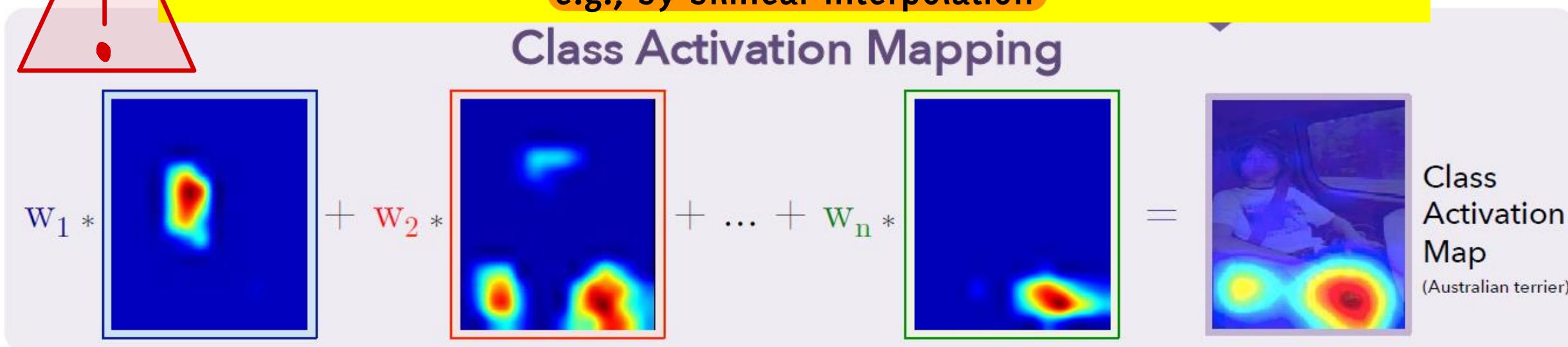
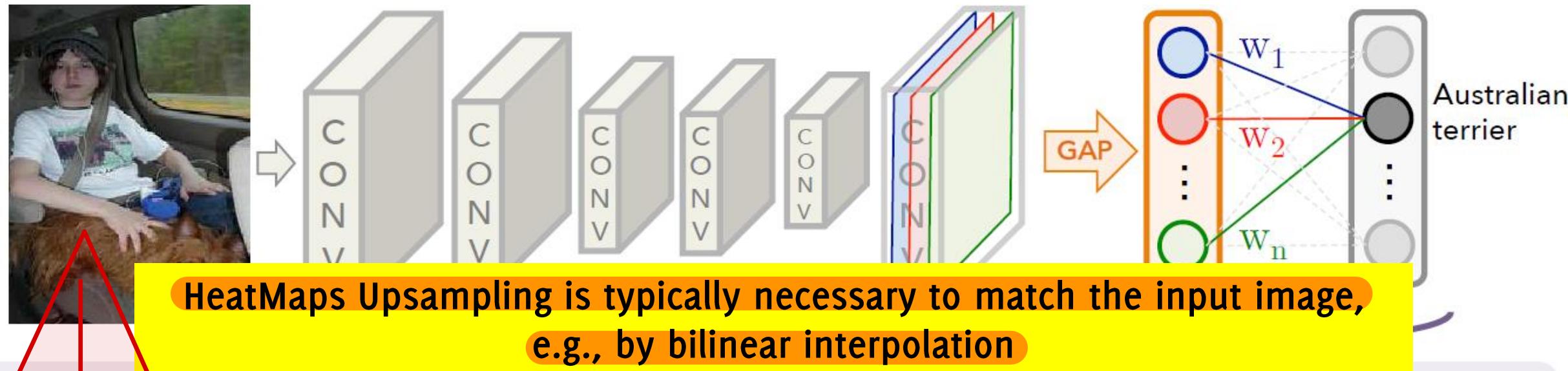


Class Activation Mapping

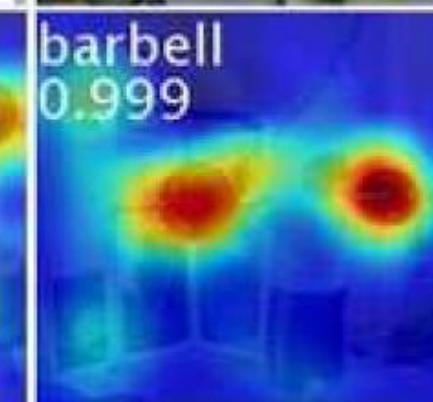
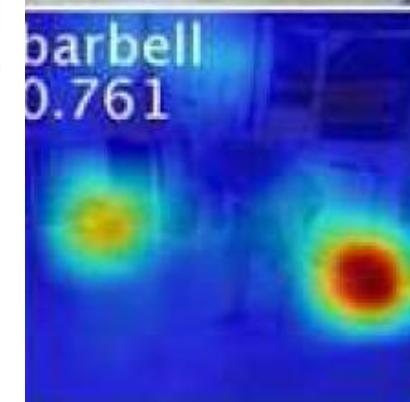
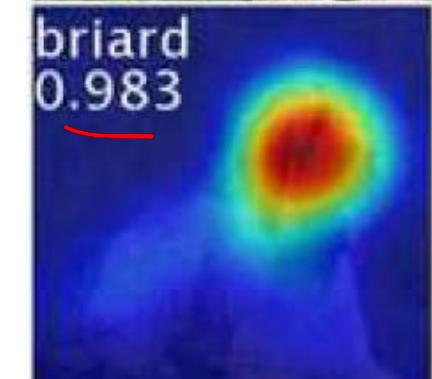
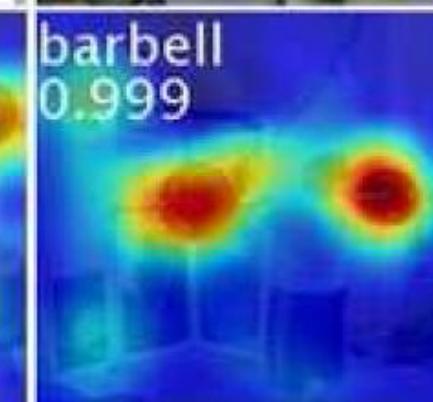
Last layer weights $\{w_k^c\}$ encode how relevant each feature map is to yield the final prediction.



Class Activation Mapping



Class Activation Mapping

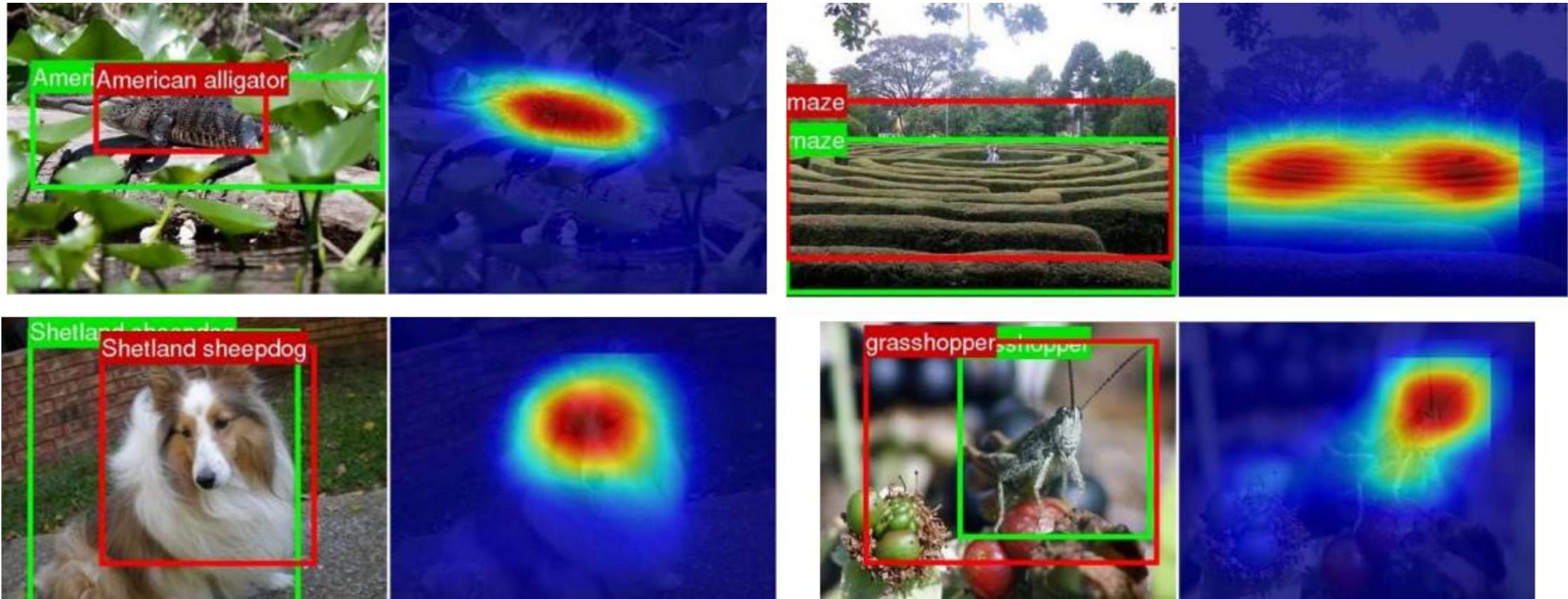


Remarks CAM can be used as an *explainability* tool.

- 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

Weakly Supervised Localization

Use thresholding CAM values: $> 20\% \text{ max}(\text{CAM})$, then take the largest component of the thresholded map (green GT, red estimated location)



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 layer 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

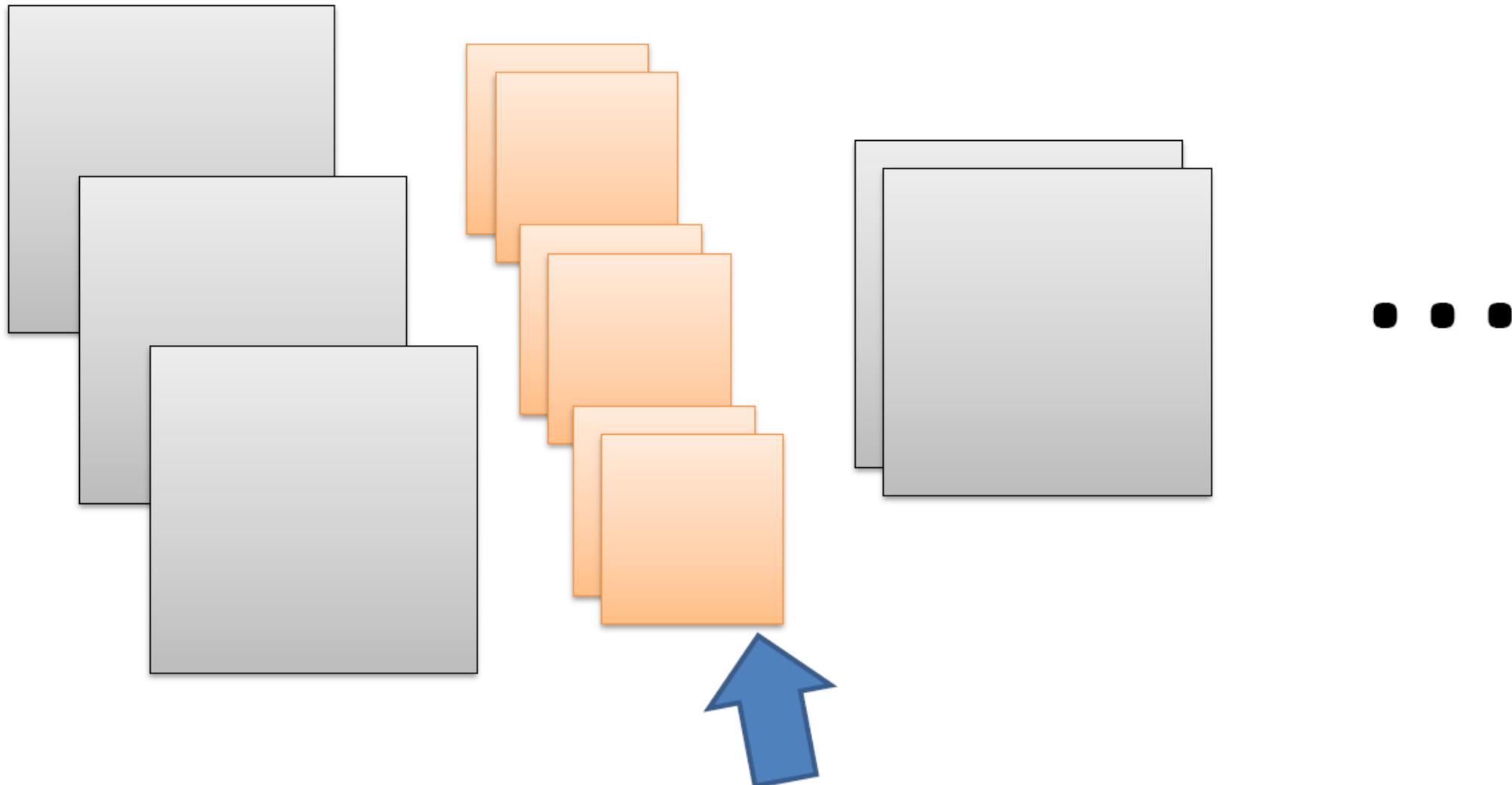
https://drive.google.com/drive/folders/1AiEwpWhd6Uru08Yerlc83875PZkzbN4W?usp=drive_link



Explaining CNN Outputs

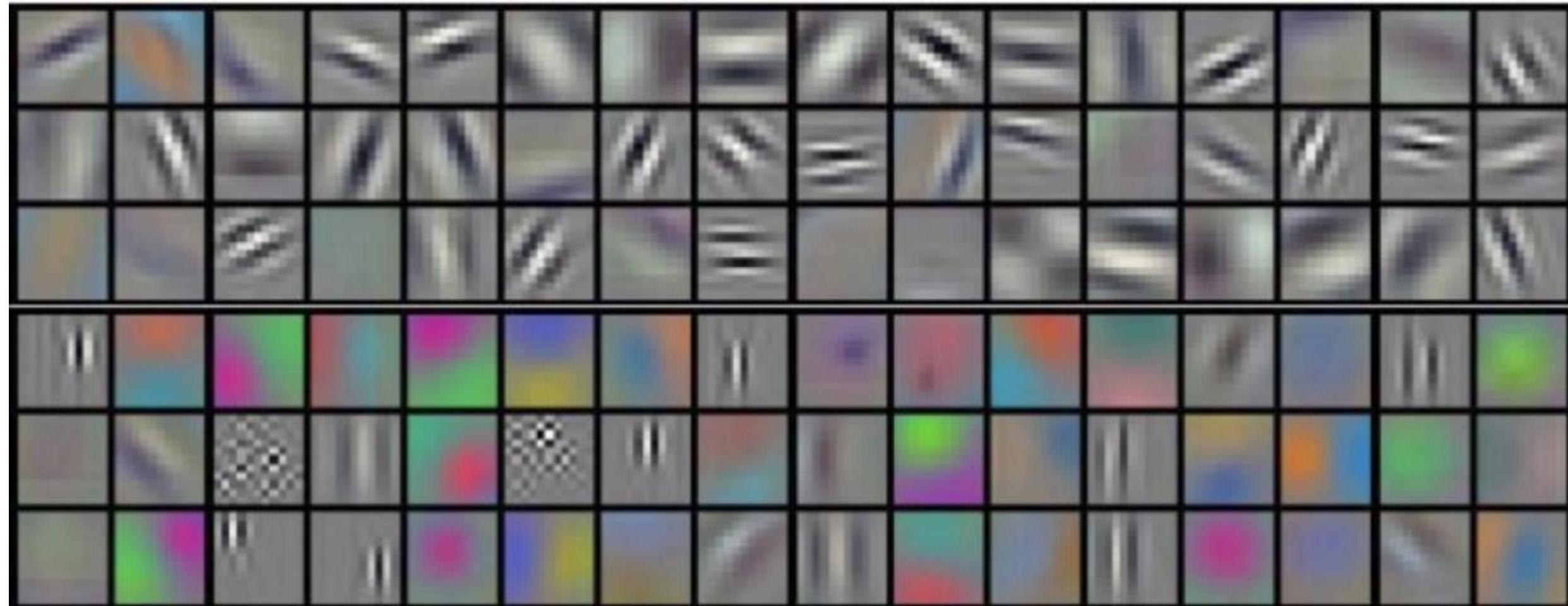
CNN Visualization

Visualizing CNN Filters



We want to see this

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



Do you remember?

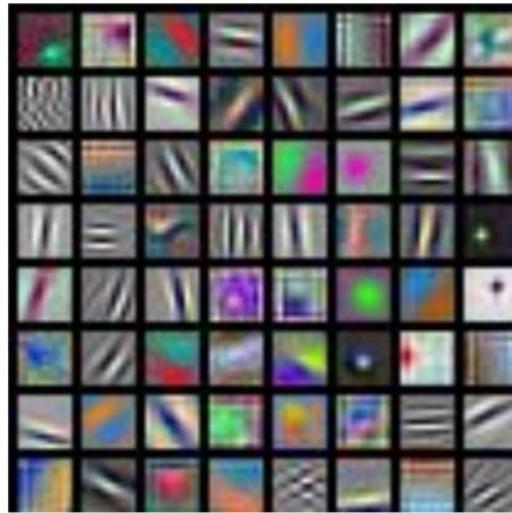
The template matching interpretation of Linear classifiers...



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

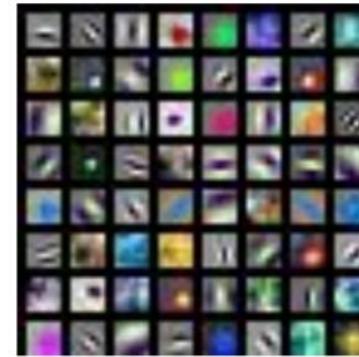
First layer's filters are often like these



AlexNet:
 $64 \times 3 \times 11 \times 11$



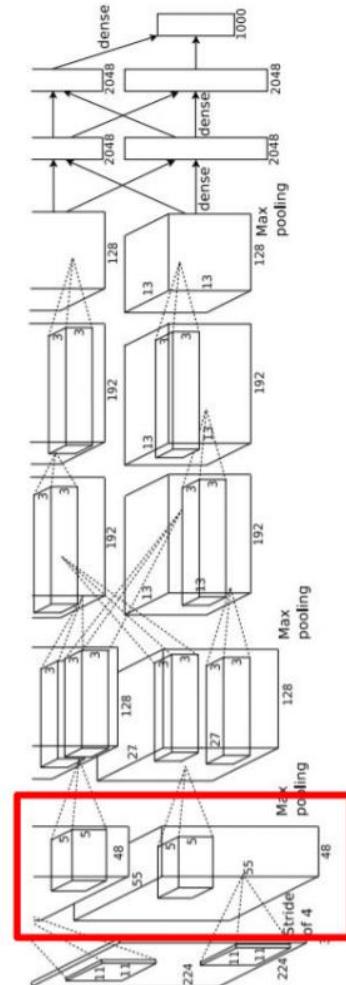
ResNet-18:
 $64 \times 3 \times 7 \times 7$



ResNet-101:
 $64 \times 3 \times 7 \times 7$



DenseNet-121:
 $64 \times 3 \times 7 \times 7$



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

Weights:



Weights:

(模糊的黑色背景)(模糊的黑色背景)(模糊的黑色背景)
(模糊的黑色背景)(模糊的黑色背景)(模糊的黑色背景)
(模糊的黑色背景)(模糊的黑色背景)(模糊的黑色背景)
(模糊的黑色背景)(模糊的黑色背景)(模糊的黑色背景)
(模糊的黑色背景)(模糊的黑色背景)(模糊的黑色背景)
(模糊的黑色背景)(模糊的黑色背景)(模糊的黑色背景)
(模糊的黑色背景)(模糊的黑色背景)(模糊的黑色背景)
(模糊的黑色背景)(模糊的黑色背景)(模糊的黑色背景)
(模糊的黑色背景)(模糊的黑色背景)(模糊的黑色背景)

Weights:

(模糊的黑色背景)(模糊的黑色背景)(模糊的黑色背景)
(模糊的黑色背景)(模糊的黑色背景)(模糊的黑色背景)
(模糊的黑色背景)(模糊的黑色背景)(模糊的黑色背景)
(模糊的黑色背景)(模糊的黑色背景)(模糊的黑色背景)
(模糊的黑色背景)(模糊的黑色背景)(模糊的黑色背景)
(模糊的黑色背景)(模糊的黑色背景)(模糊的黑色背景)
(模糊的黑色背景)(模糊的黑色背景)(模糊的黑色背景)
(模糊的黑色背景)(模糊的黑色背景)(模糊的黑色背景)
(模糊的黑色背景)(模糊的黑色背景)(模糊的黑色背景)

layer 1 weights

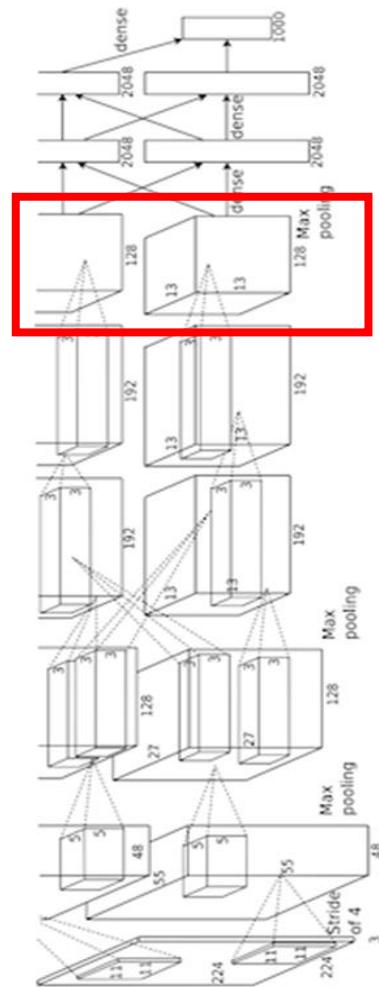
$16 \times 3 \times 7 \times 7$

layer 2 weights

$20 \times 16 \times 7 \times 7$

layer 3 weights

$20 \times 20 \times 7 \times 7$



Difficult to interpret deeper layers

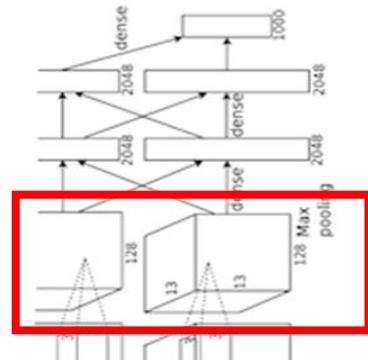


Weights:

(128x128x3x16) (128x128x16x16) (128x128x16x16)
(128x128x16x16) (128x128x16x16) (128x128x16x16)
(128x128x16x16) (128x128x16x16) (128x128x16x16)
(128x128x16x16) (128x128x16x16) (128x128x16x16)
(128x128x16x16) (128x128x16x16) (128x128x16x16)

layer 1 weights

$16 \times 3 \times 7 \times 7$



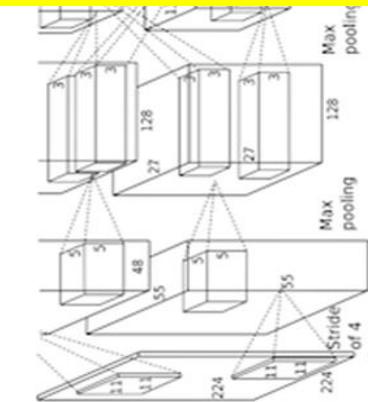
layer 2 weights

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

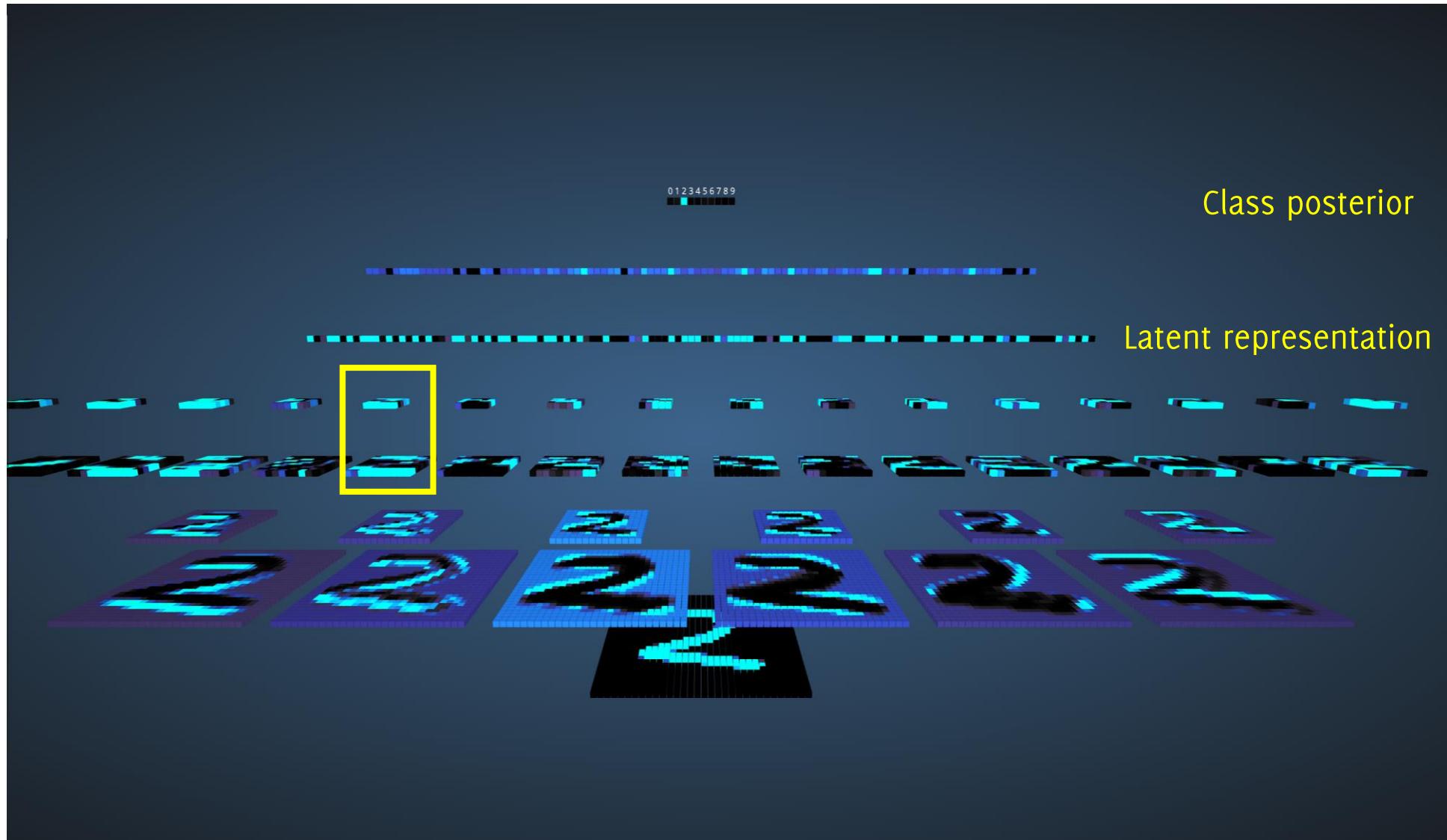
(128x128x16x16) (128x128x16x16) (128x128x16x16)
(128x128x16x16) (128x128x16x16) (128x128x16x16)
(128x128x16x16) (128x128x16x16) (128x128x16x16)
(128x128x16x16) (128x128x16x16) (128x128x16x16)
(128x128x16x16) (128x128x16x16) (128x128x16x16)

layer 3 weights

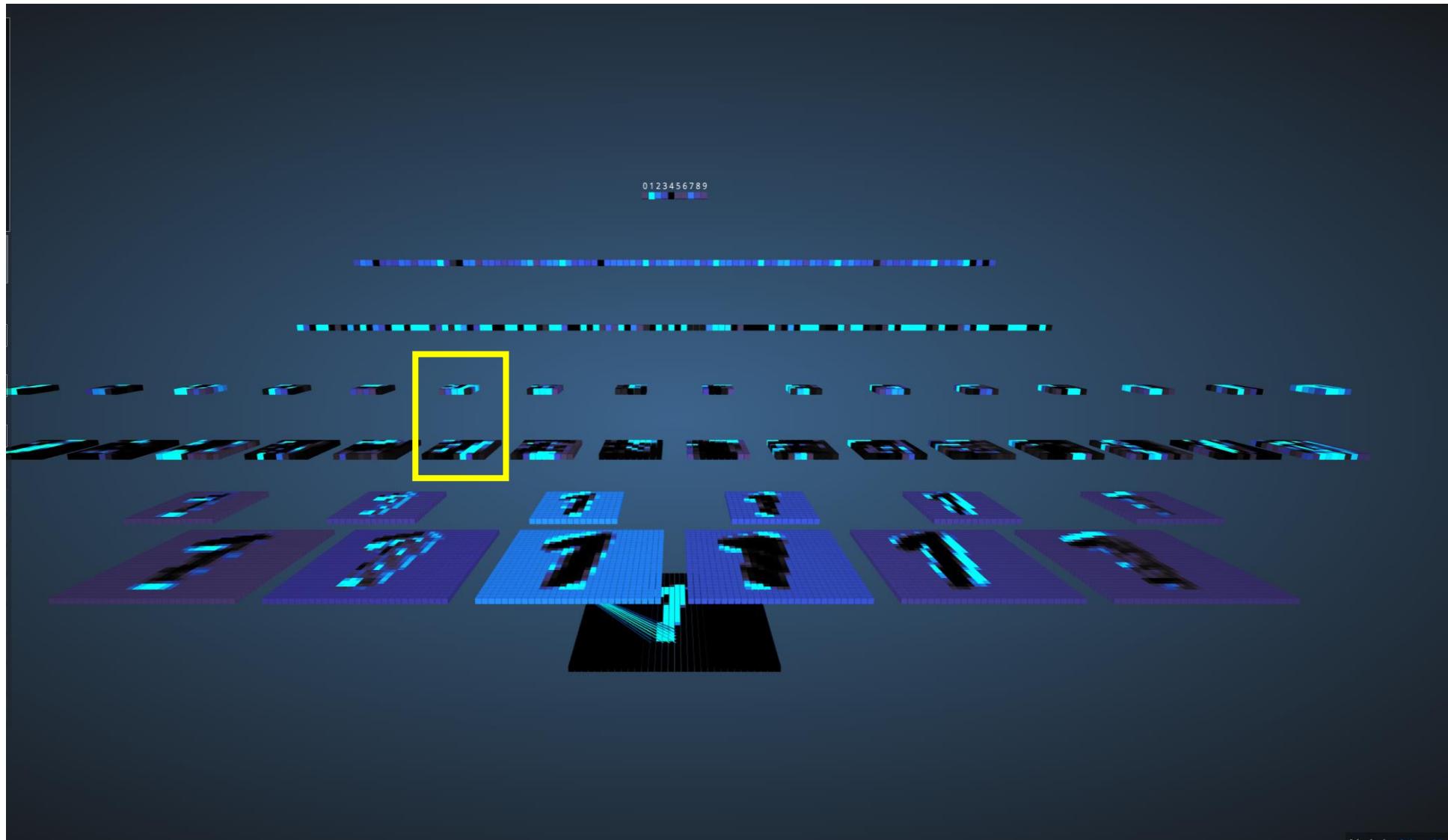
$20 \times 20 \times 7 \times 7$



What if we look at the activations?



What if we look at the activations?



Made by Adam Harley



Visualizing Maximally Activating Patches

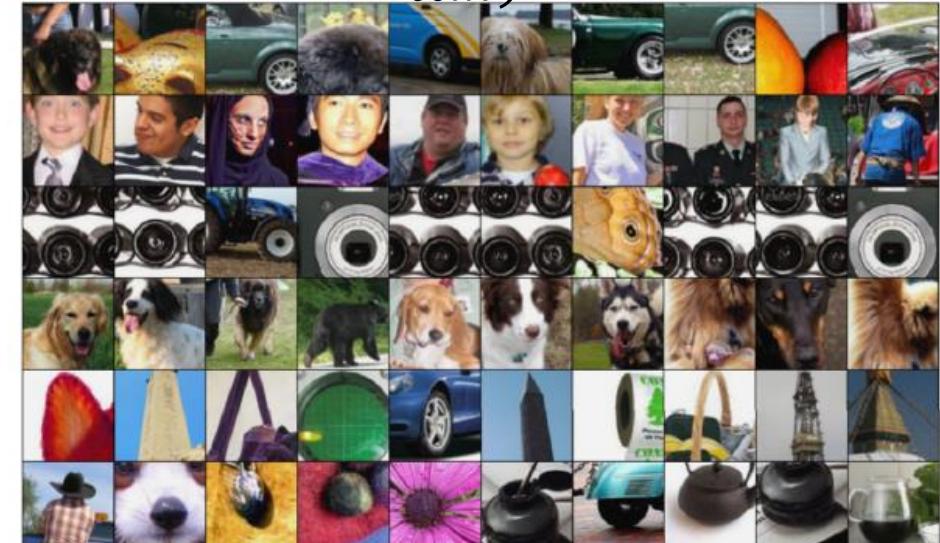
conv6

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.

receptive
field ↑

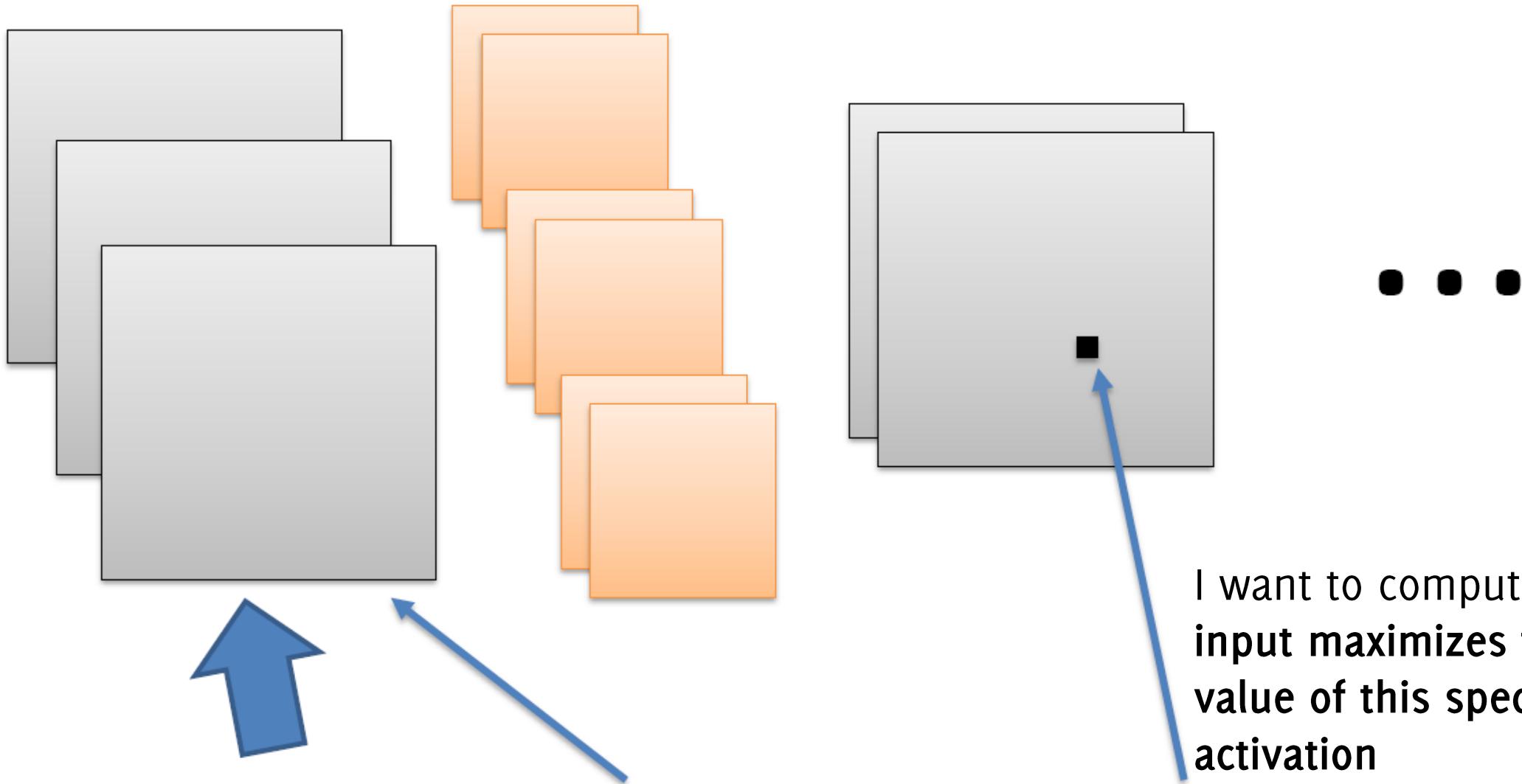


conv9

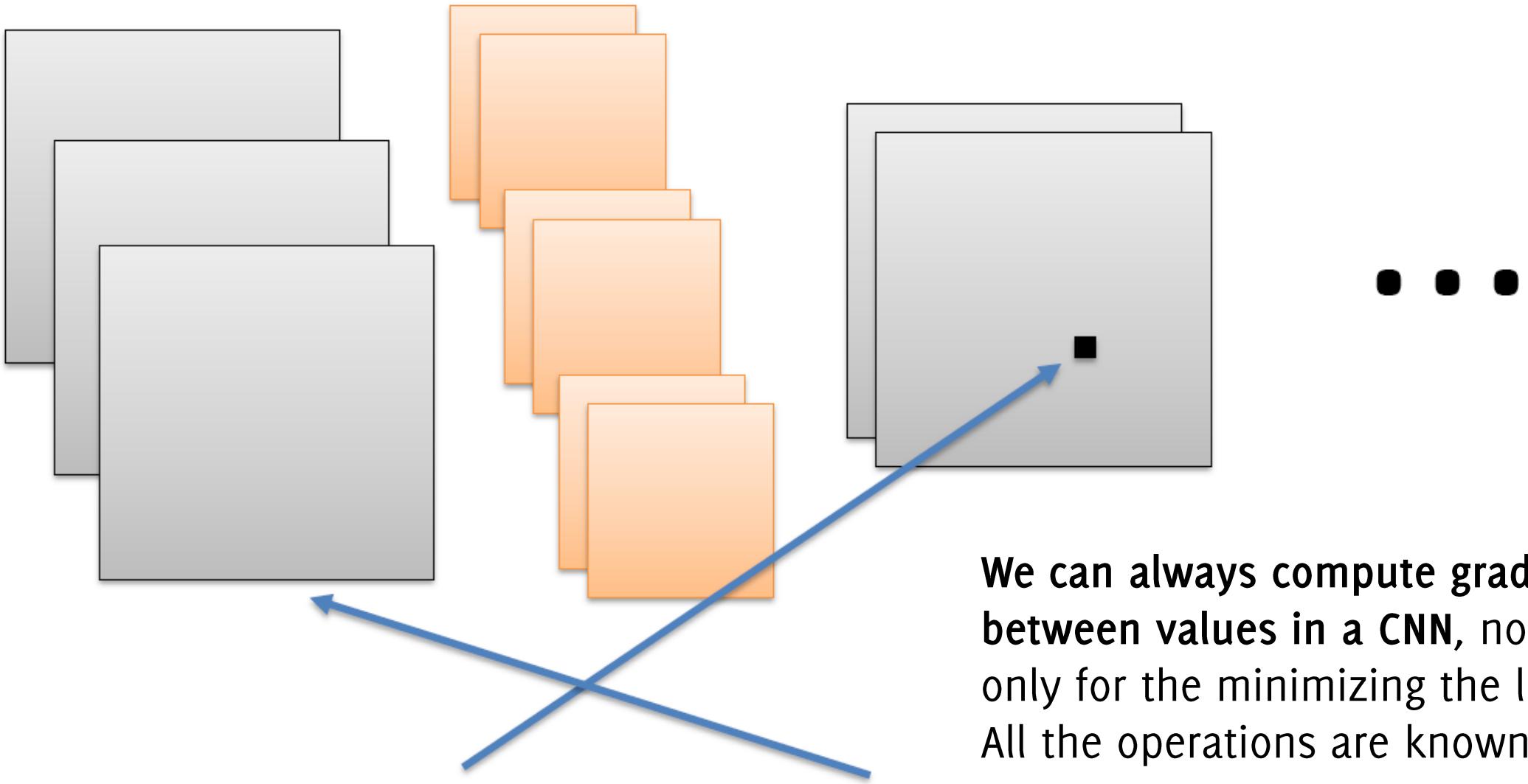


Each row in these images corresponds to different outputs of the same filter

Computing Input maximally activating a neuron



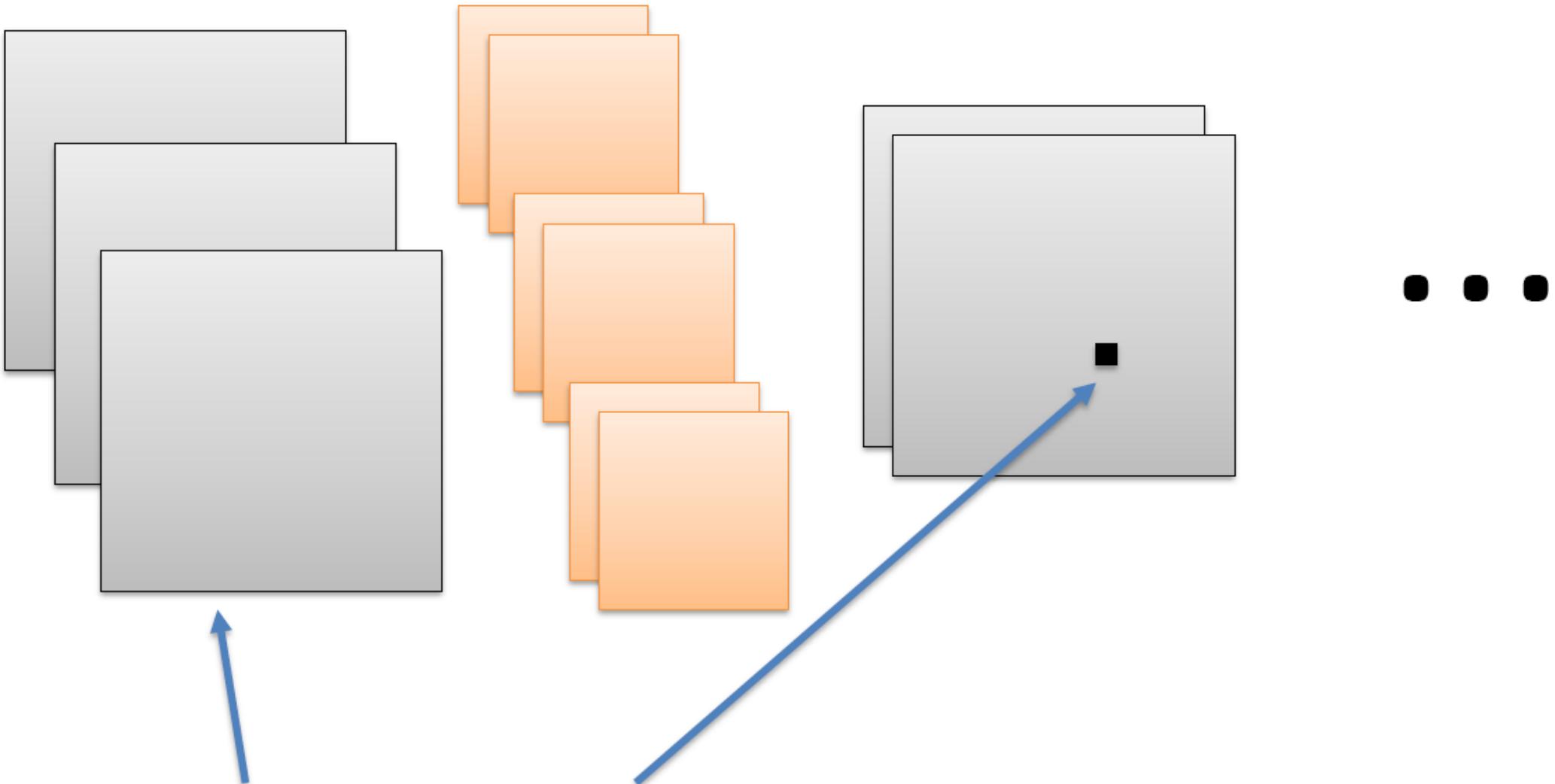
Step 1



Compute the gradient of this with respect to the input

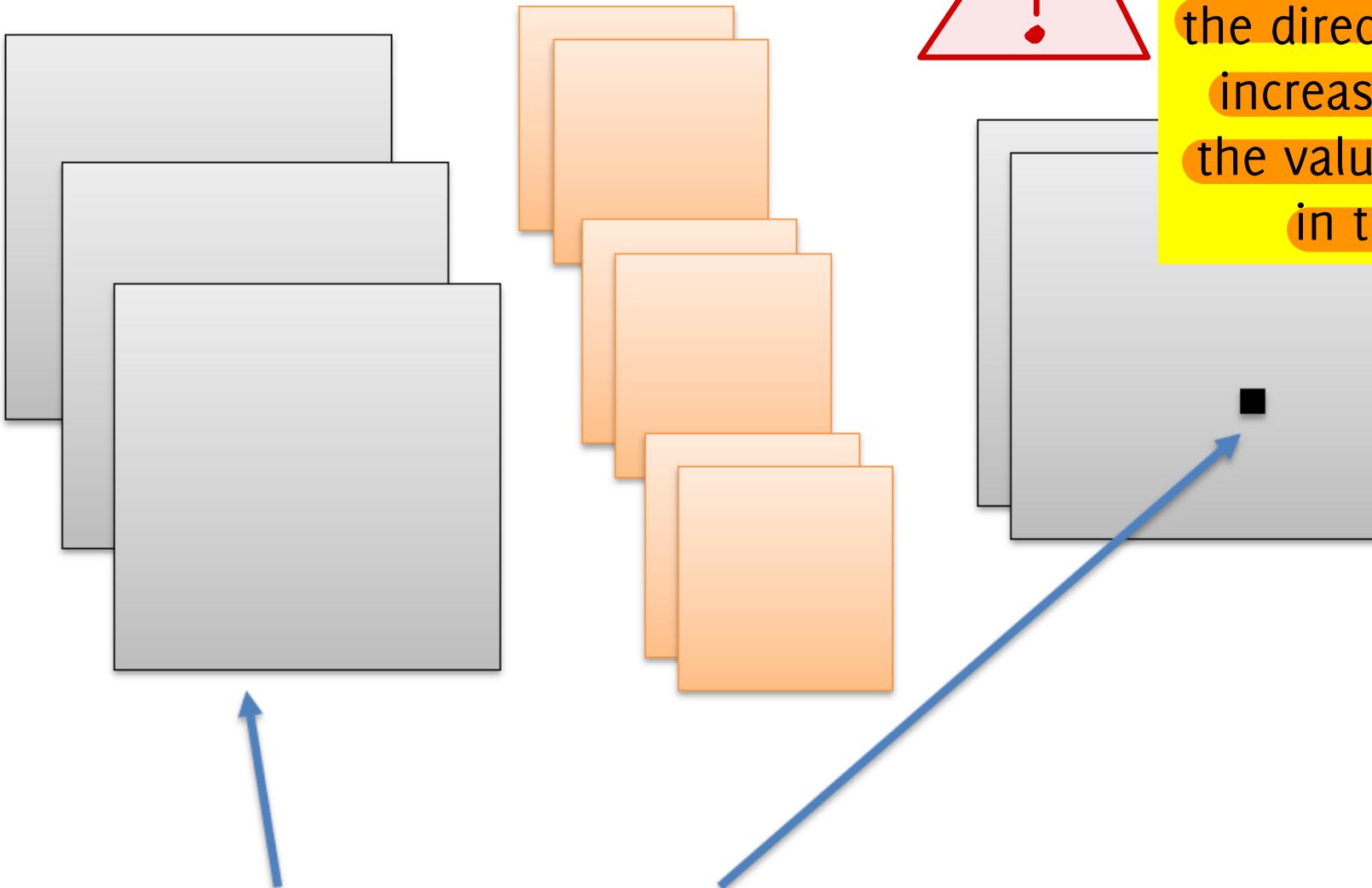
We can always compute gradient between values in a CNN, not only for the minimizing the loss. All the operations are known

Step 2



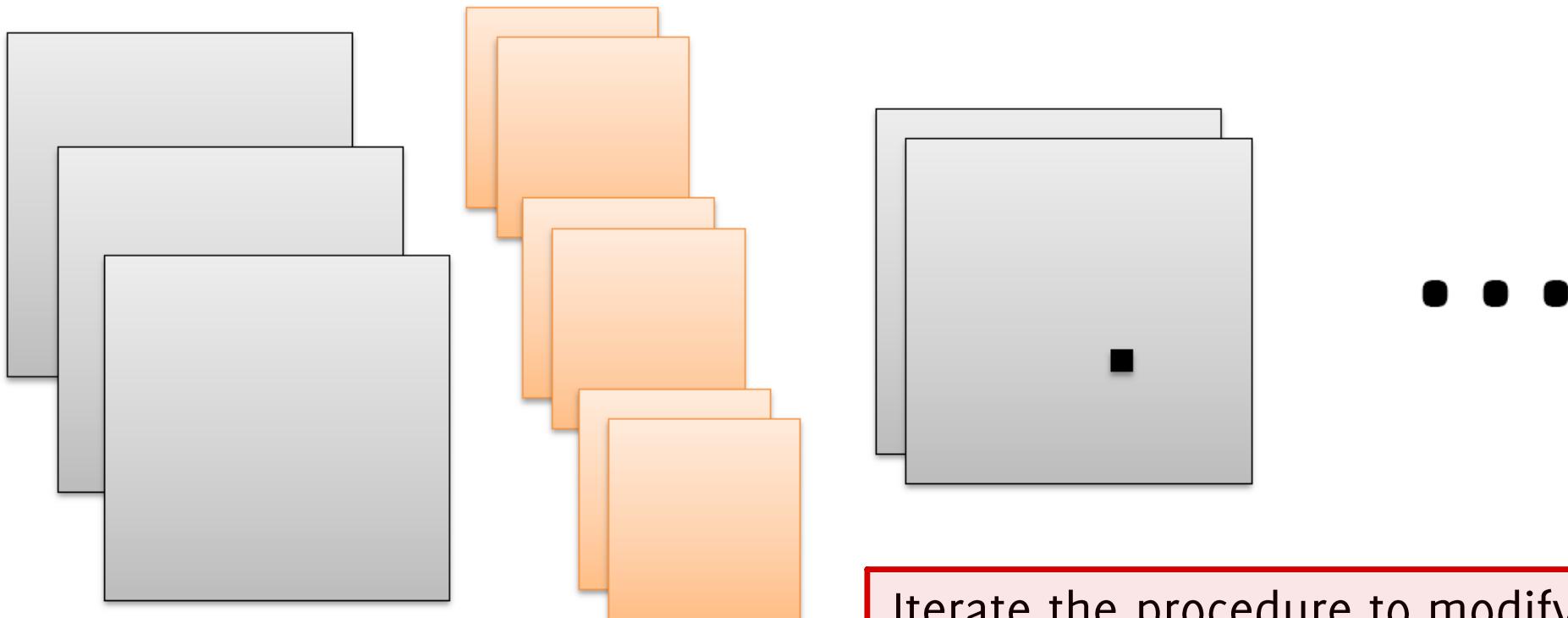
Nudge the input accordingly: our guy will increase its activation

Step 2



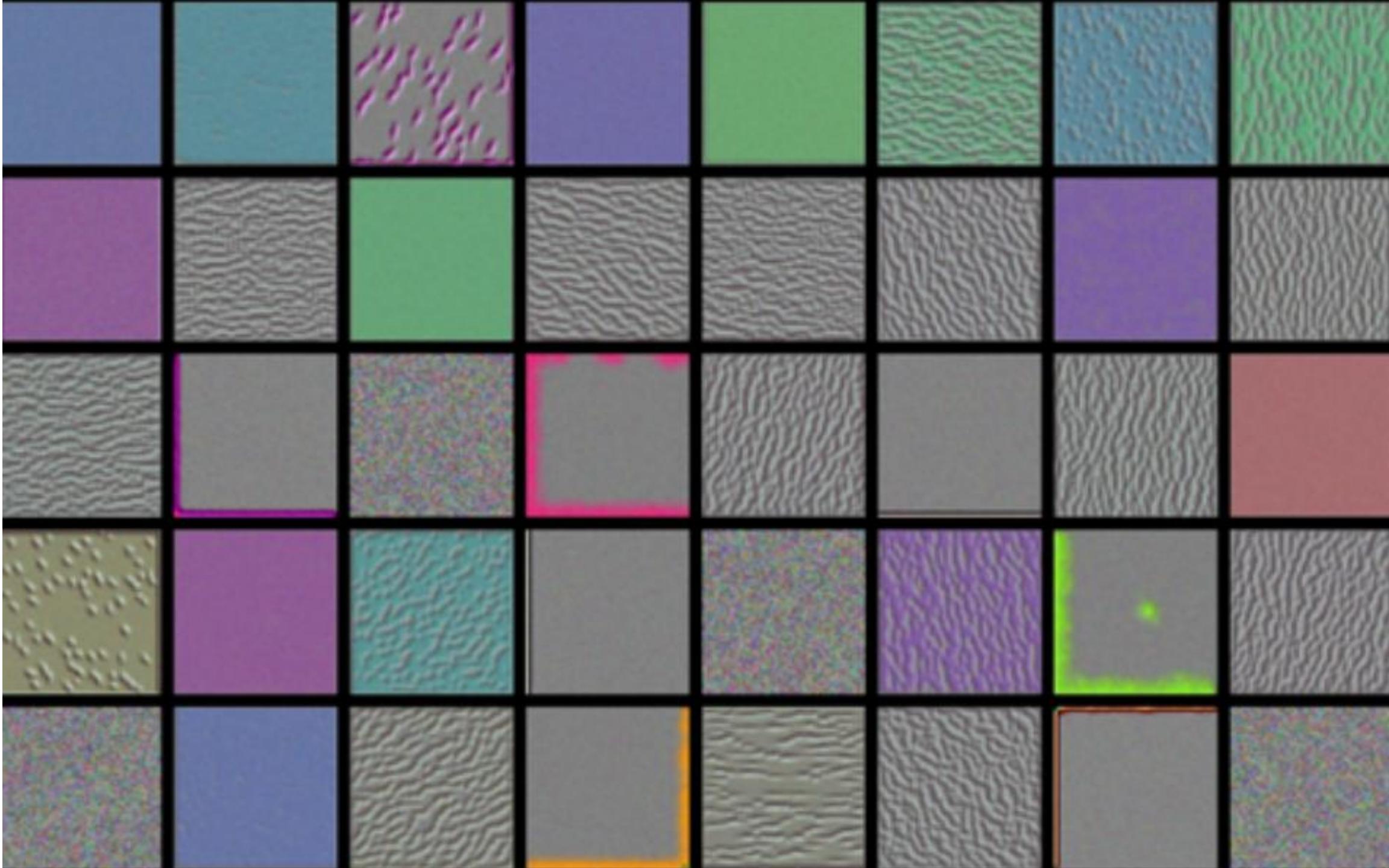
Now we perform gradient ascent, we modify the input in the direction of the gradient, to increase our function that is the value of the selected pixel in the activation map

Back to step 1 and iterate

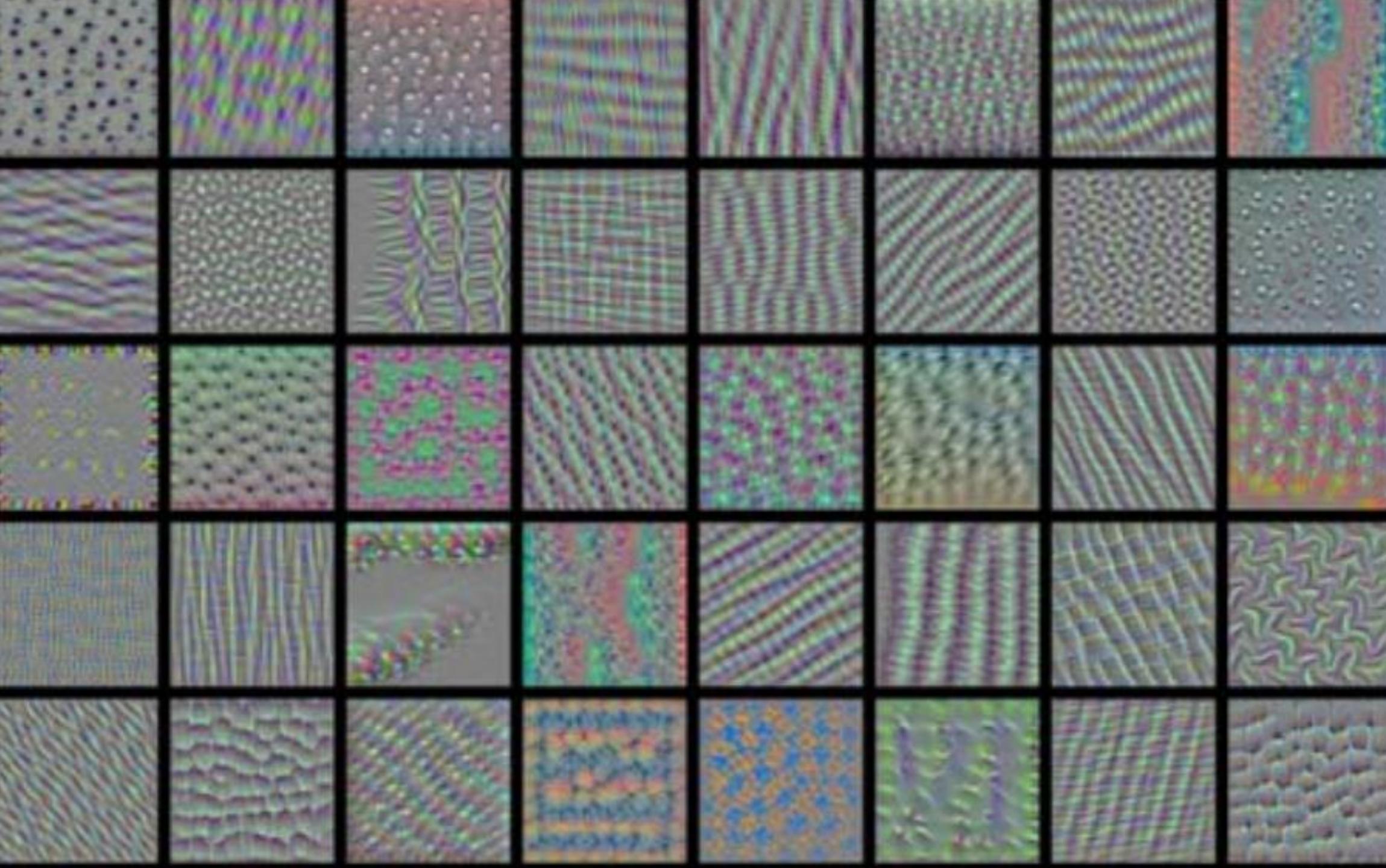


Iterate the procedure to modify the input. Some form of regularization can be added to the selected pixel value to steer the input to look more like a natural image

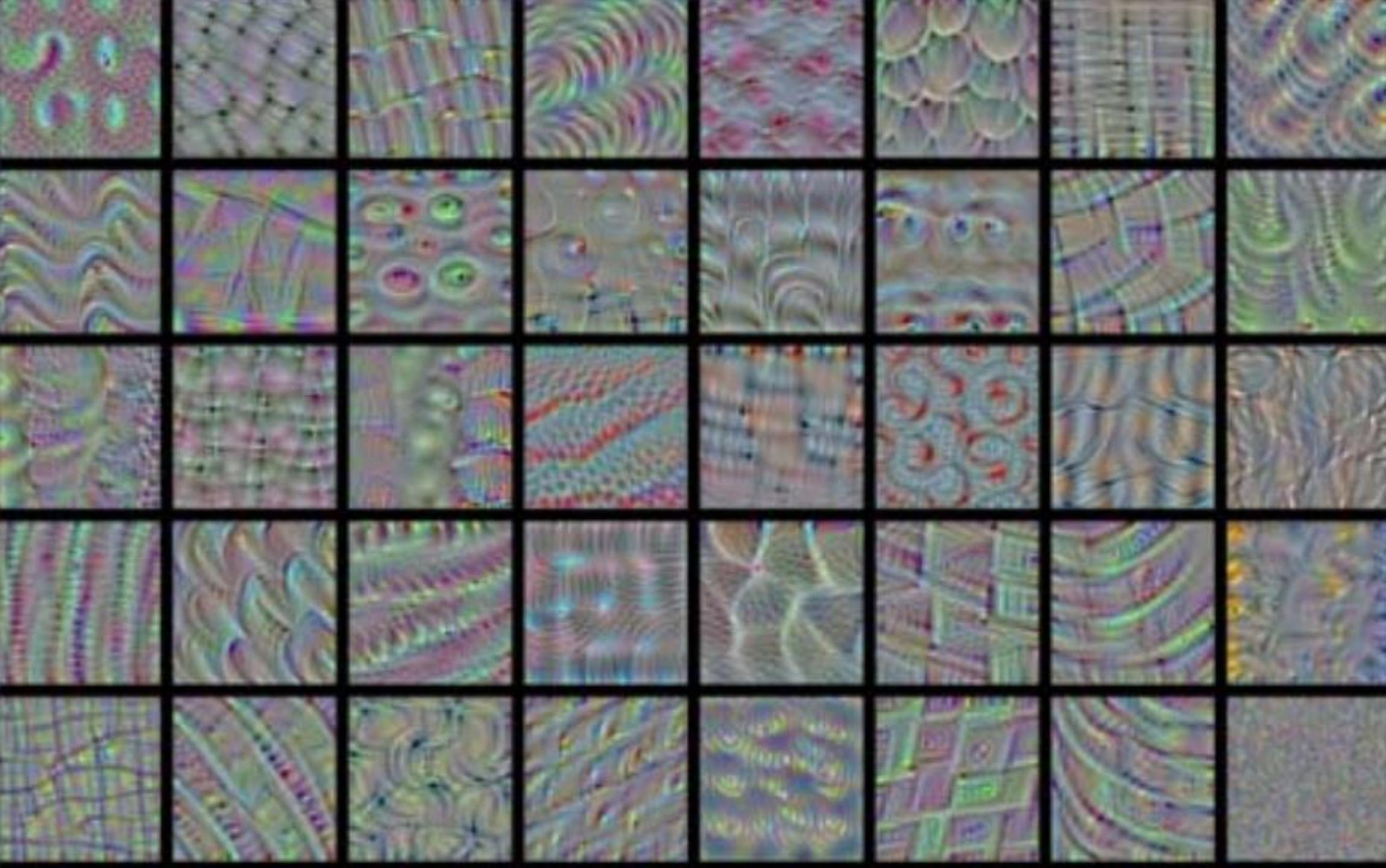
Shallow layers respond to fine, low-level patterns



Intermediate layers ...



Deep layers respond to complex,
high-level patterns



Computing Images maximally activating softmax neuron

Adopt gradient descent to maximally activate a neuron before the softmax, thus the network «score»: the higher the most likely the network predicts that class

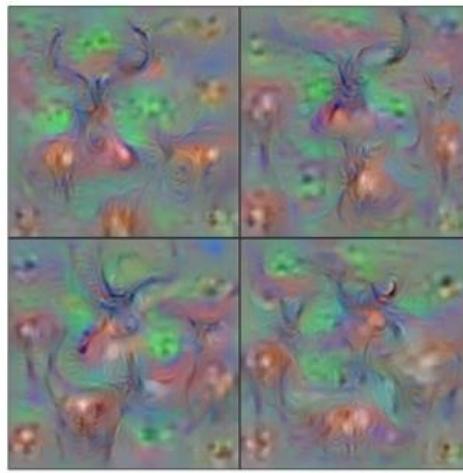
$$\hat{I} = \operatorname{argmax}_I S_c(I) + \lambda \|I\|_2^2$$

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

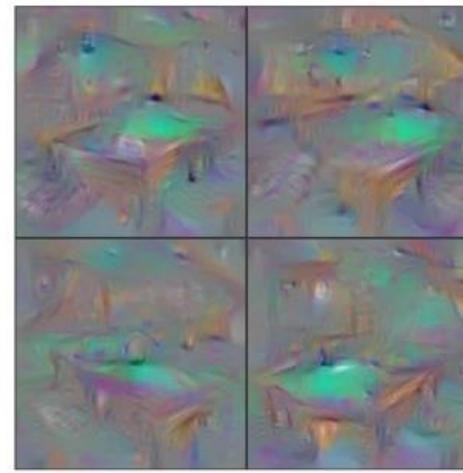
To improve the smoothness of acquired imgs is necessary to add a regularization term $\lambda \|I\|_2^2$.

Maximize this function through gradient ascent over the network input. Repeat this operation for neurons corresponding to different classes c .

Computing Images maximally activating softmax neuron



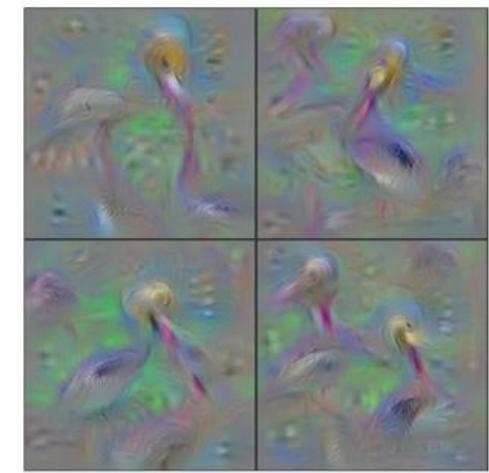
Hartebeest



Billiard Table



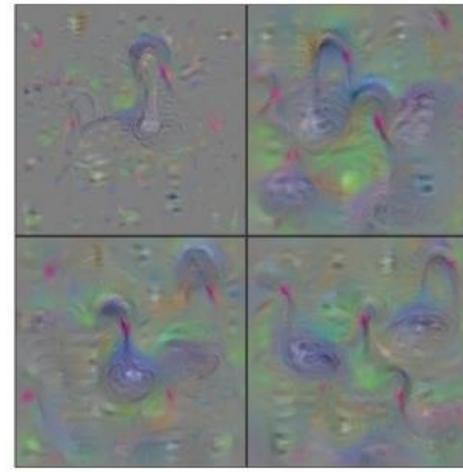
Flamingo



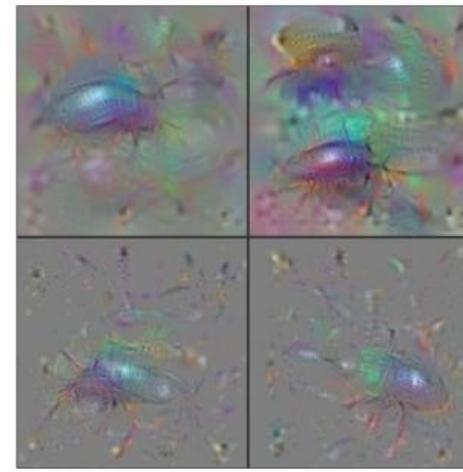
Pelican



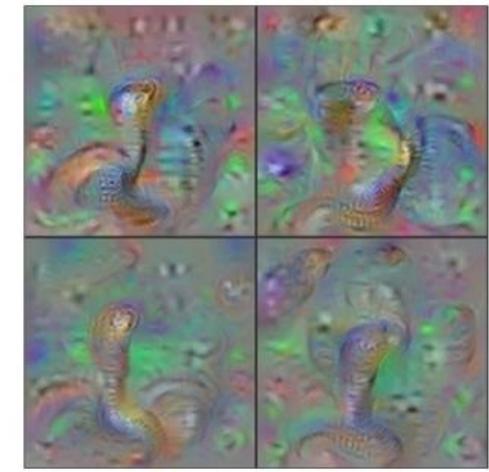
Station Wagon



Black Swan



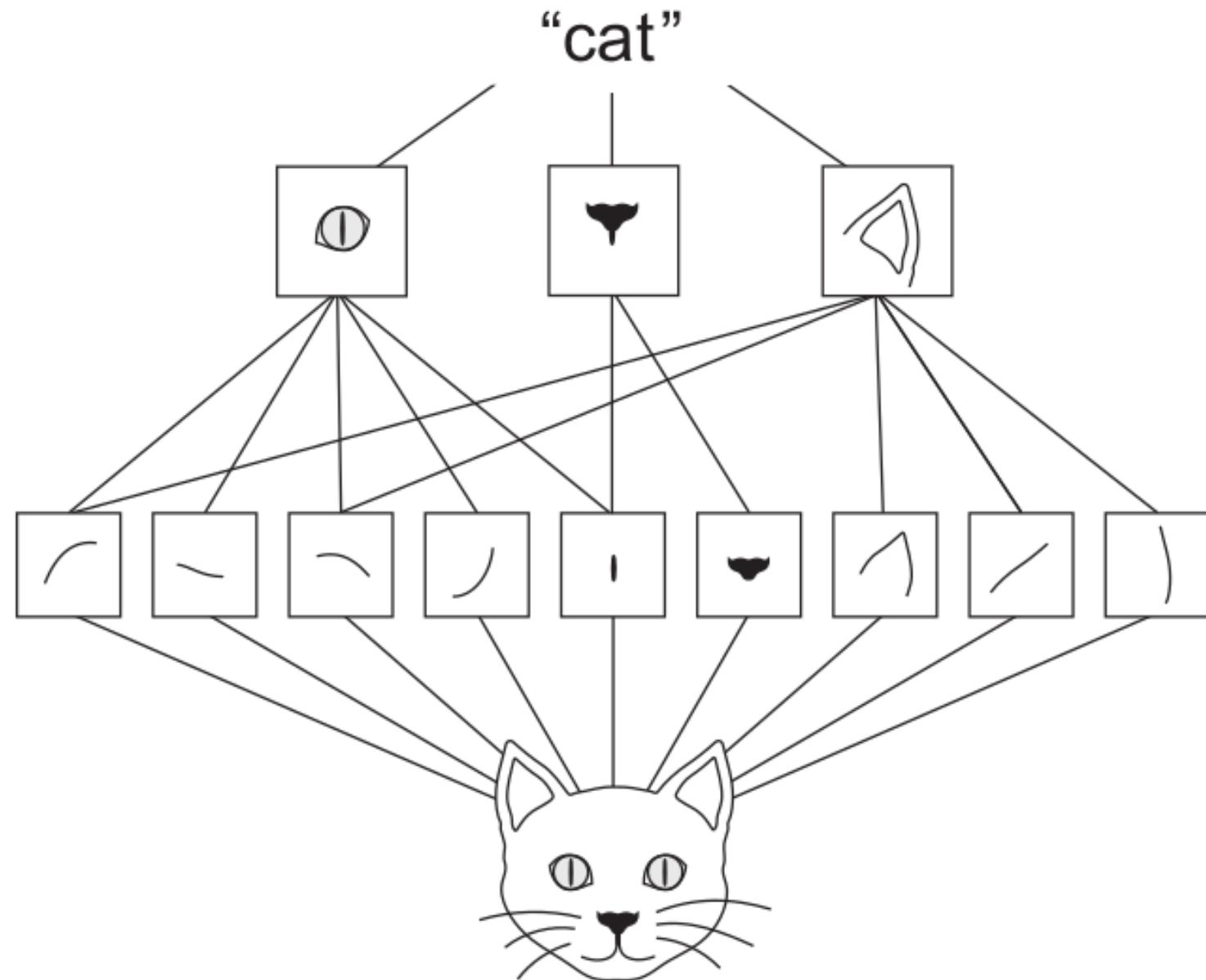
Ground Beetle



Indian Cobra

Why CNNs work

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



Explaining Neural Network Predictions

Saliency Maps and Others

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”

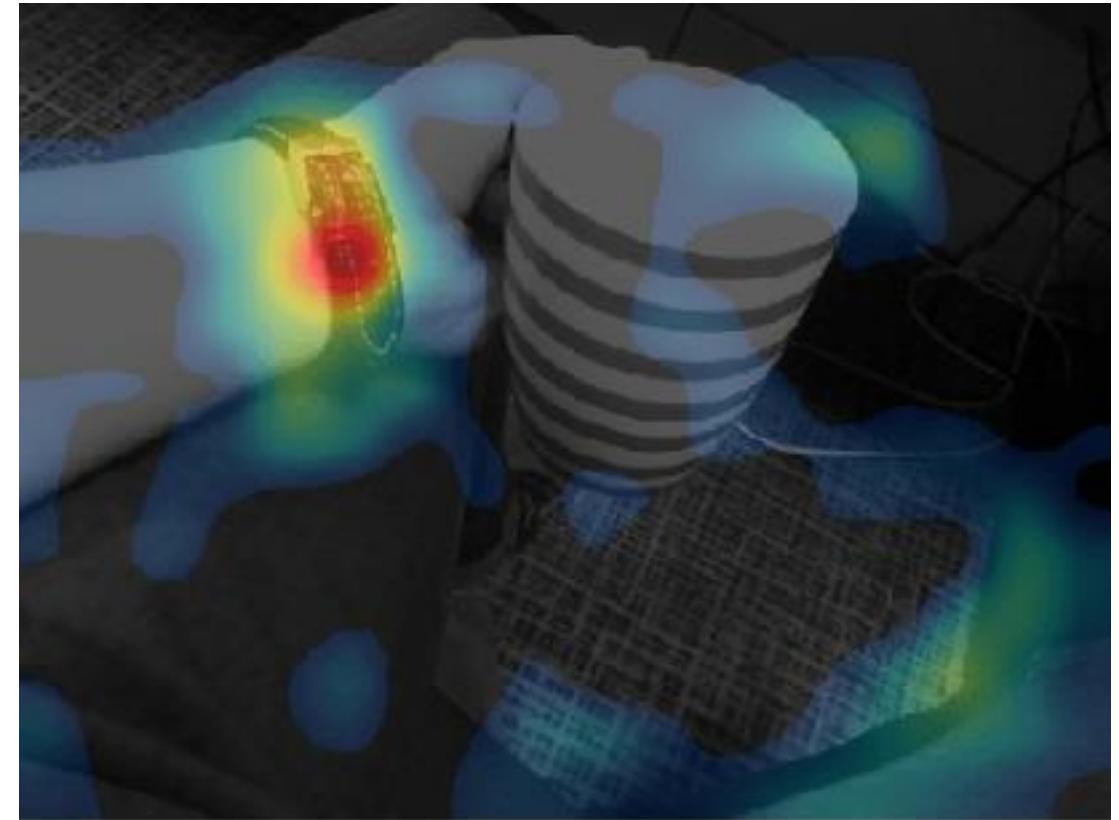
Why? Let's check the CAM!

Saliency Maps to Understand Model Mistakes

Make sense of model mistakes



Mispredicted as “buckle”



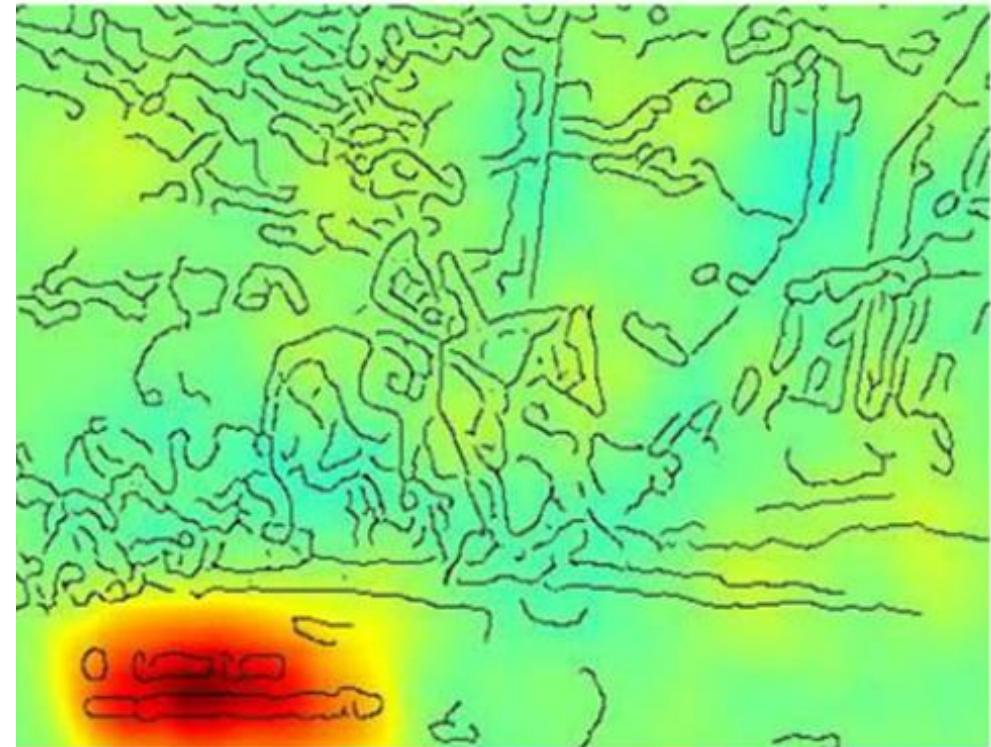
Saliency shows why

Saliency Maps to Discover Systematic Errors

Highlight clever Hans phenomena



Correctly classified as “horse”

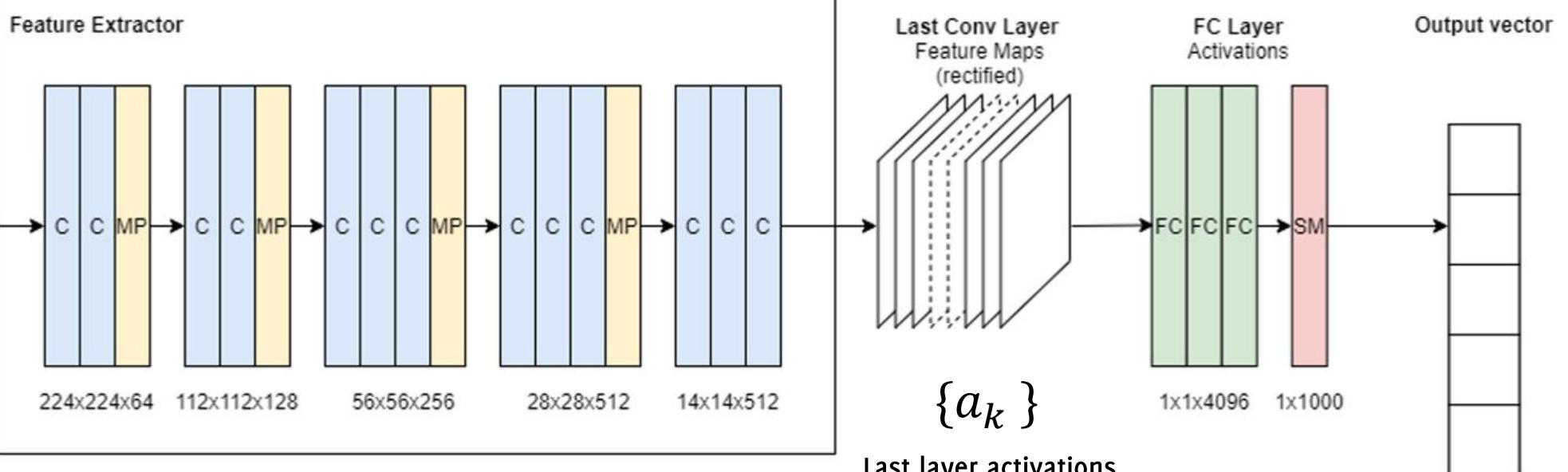


But for the wrong reason

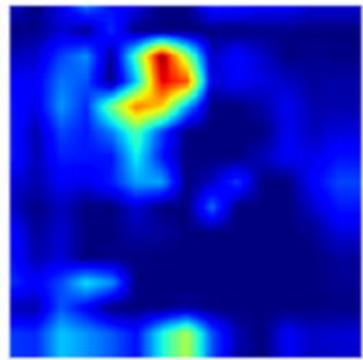


Grad-CAM and CAM-based techniques

input image x

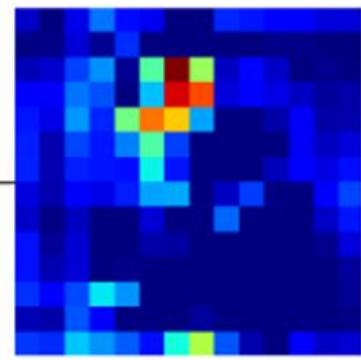


Last layer activations



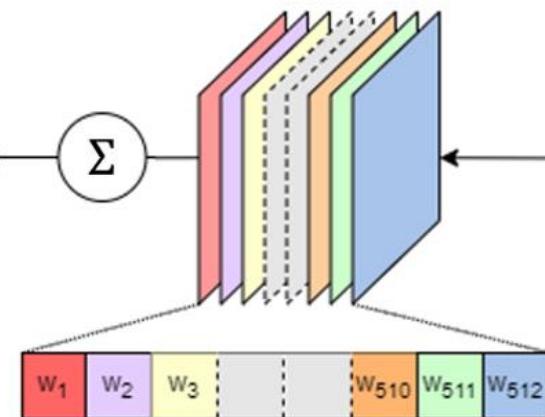
224x224

high-resolution
CAM h

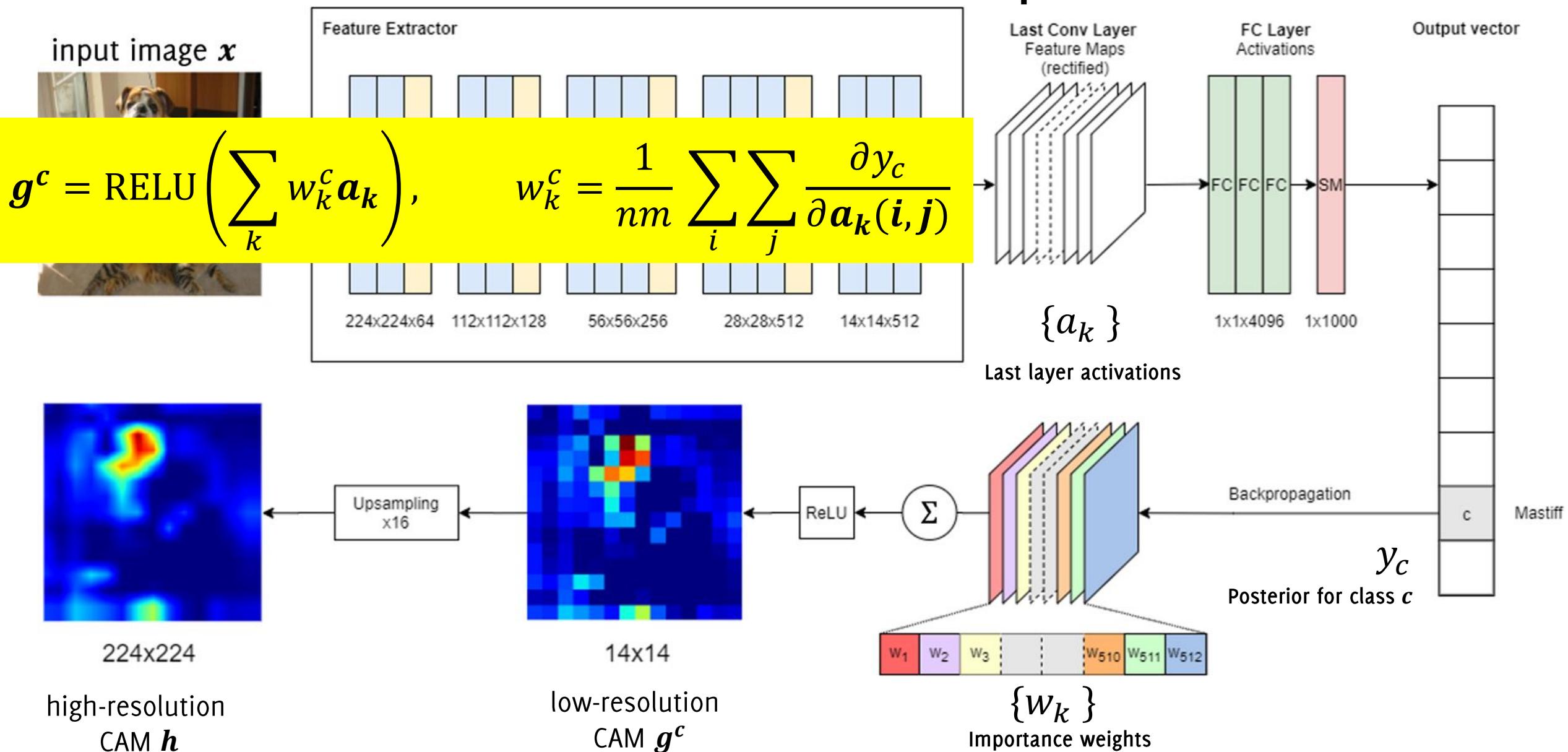


14x14

low-resolution
CAM g^c



Grad-CAM and CAM-based techniques

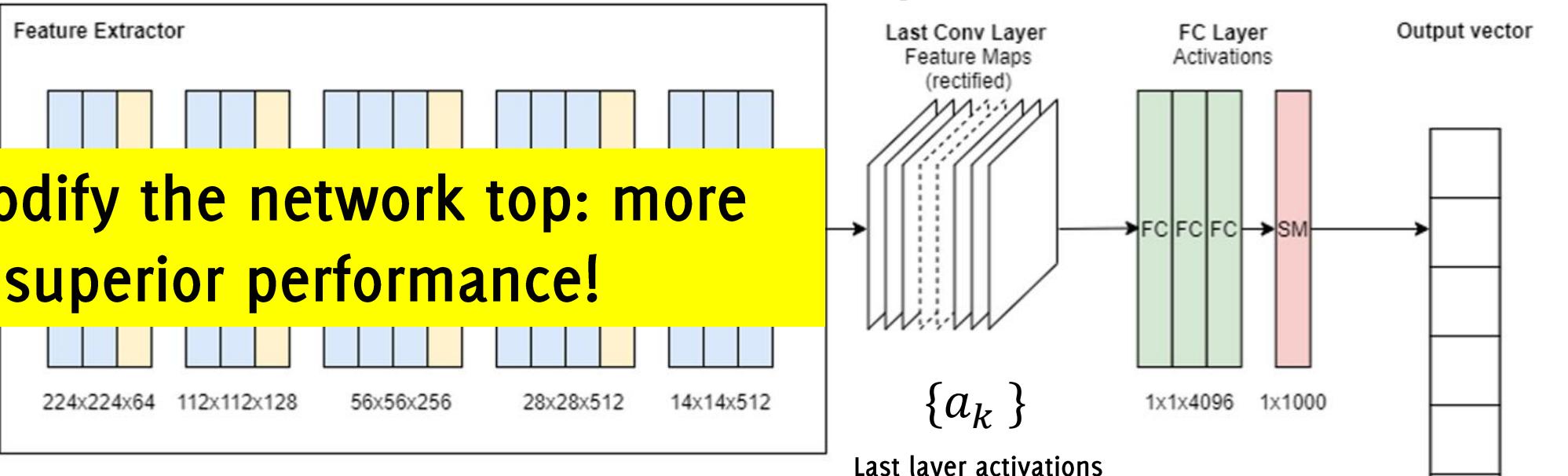


Grad-CAM and CAM-based techniques

input image x



No need to modify the network top: more flexibility and superior performance!



$$\{a_k\}$$

Last layer activations

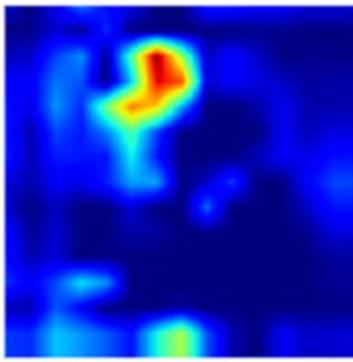
$$1 \times 1 \times 4096$$

$$1 \times 1000$$

Mastiff

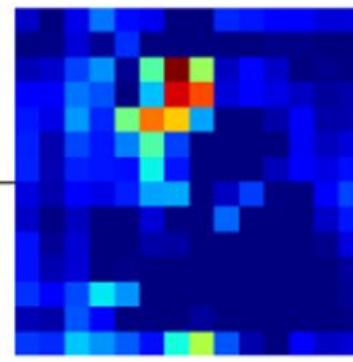
$$y_c$$

Posterior for class c



224x224

high-resolution
CAM h



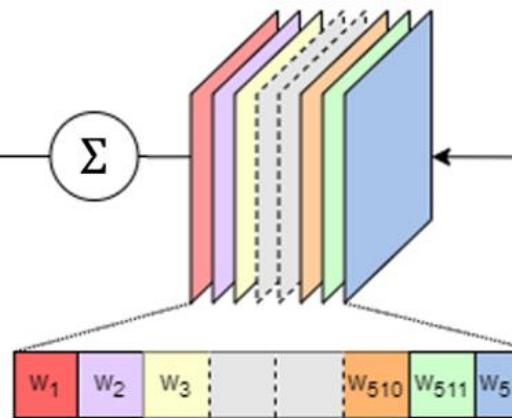
14x14

low-resolution
CAM g^c

$$W_1 \quad W_2 \quad W_3 \quad | \quad | \quad | \quad W_{510} \quad W_{511} \quad W_{512}$$

$$\{w_k\}$$

Importance weights

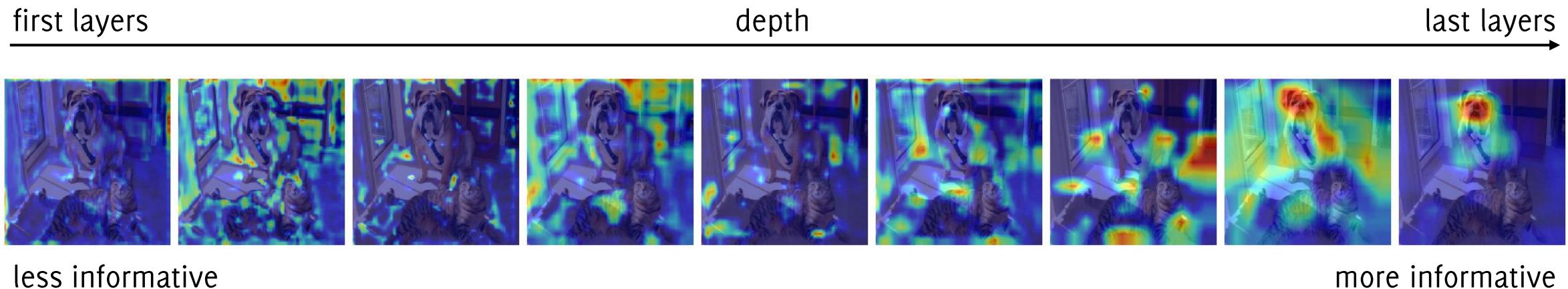


Heatmaps Desiderata

Should be **class discriminative**

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

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

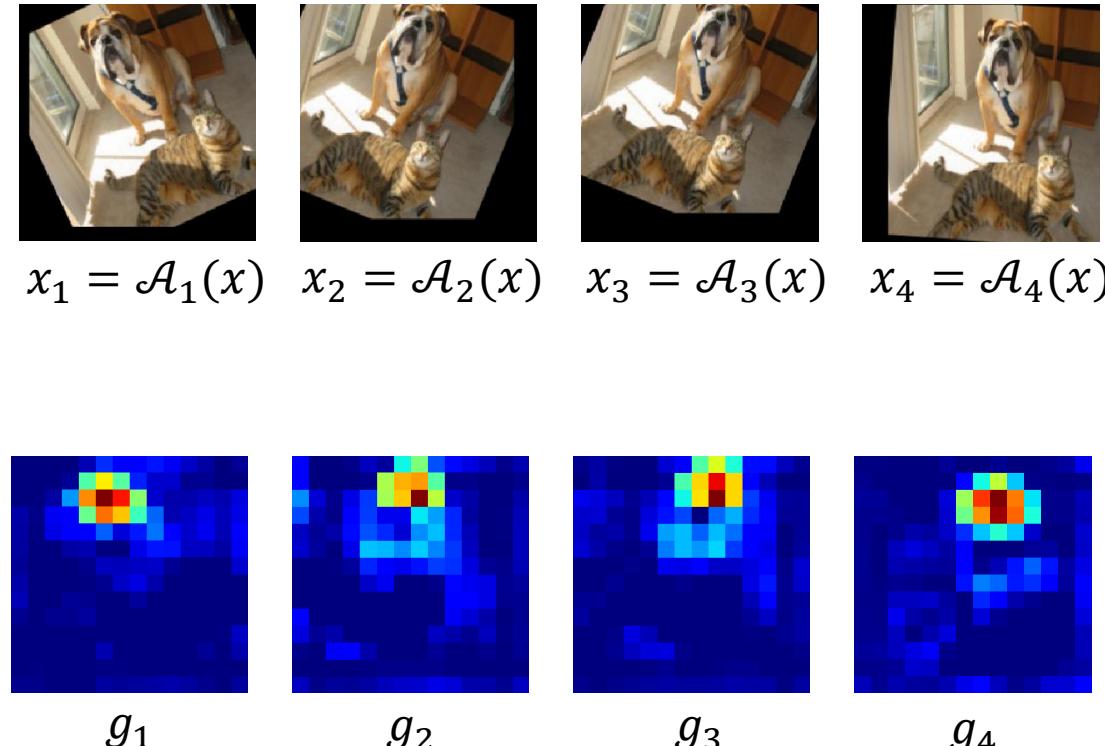


Augmented Grad-CAM

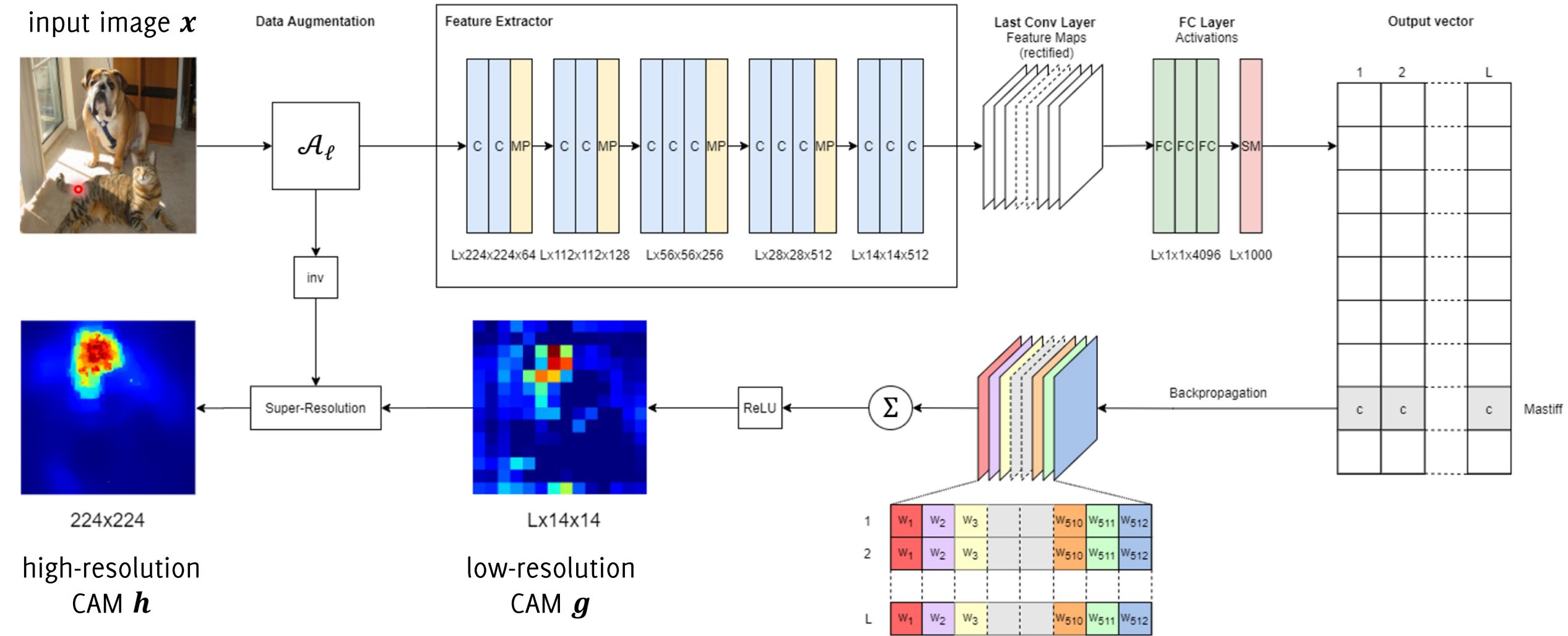
Intuition behind augmented Grad-CAM:
increase the heat-maps resolution through
image augmentation.

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

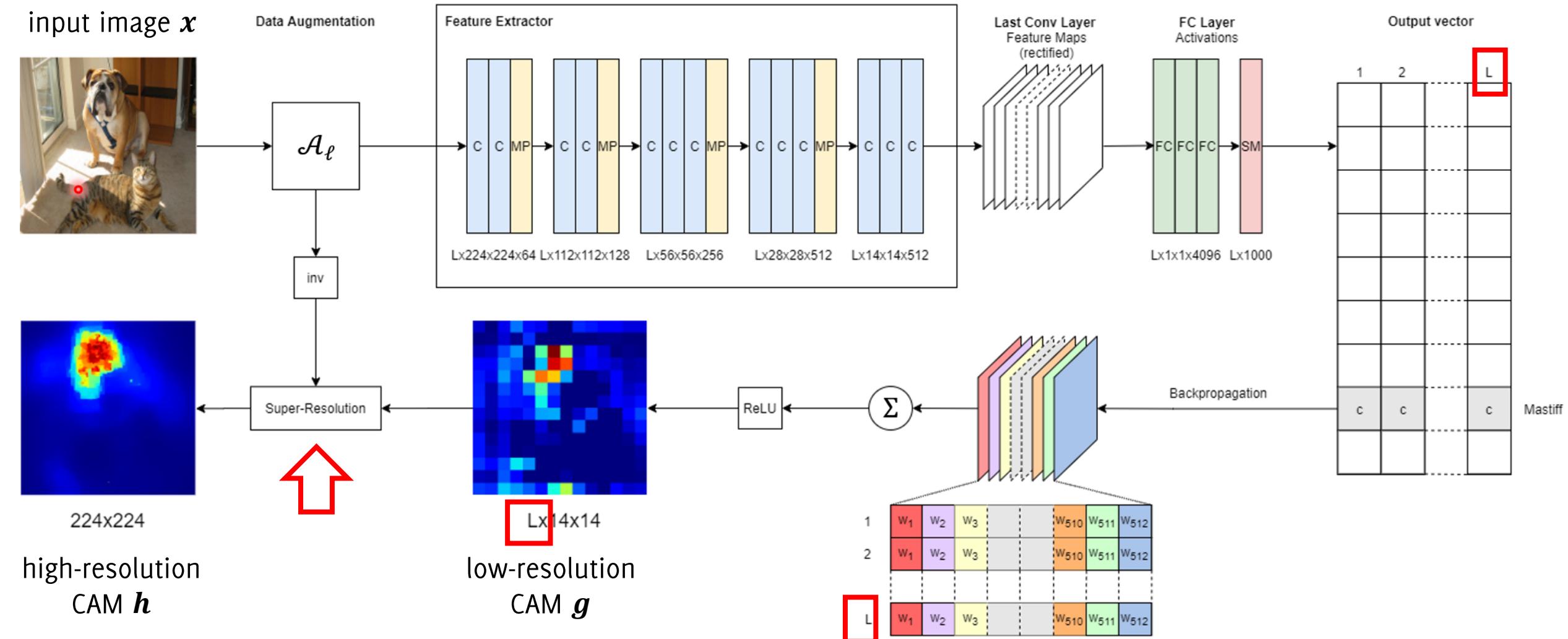
All the Grad-CAM that the CNN generates
when fed with **multiple augmented
versions of the same input image**, are very
informative for reconstructing the high-
resolution heat-map h .



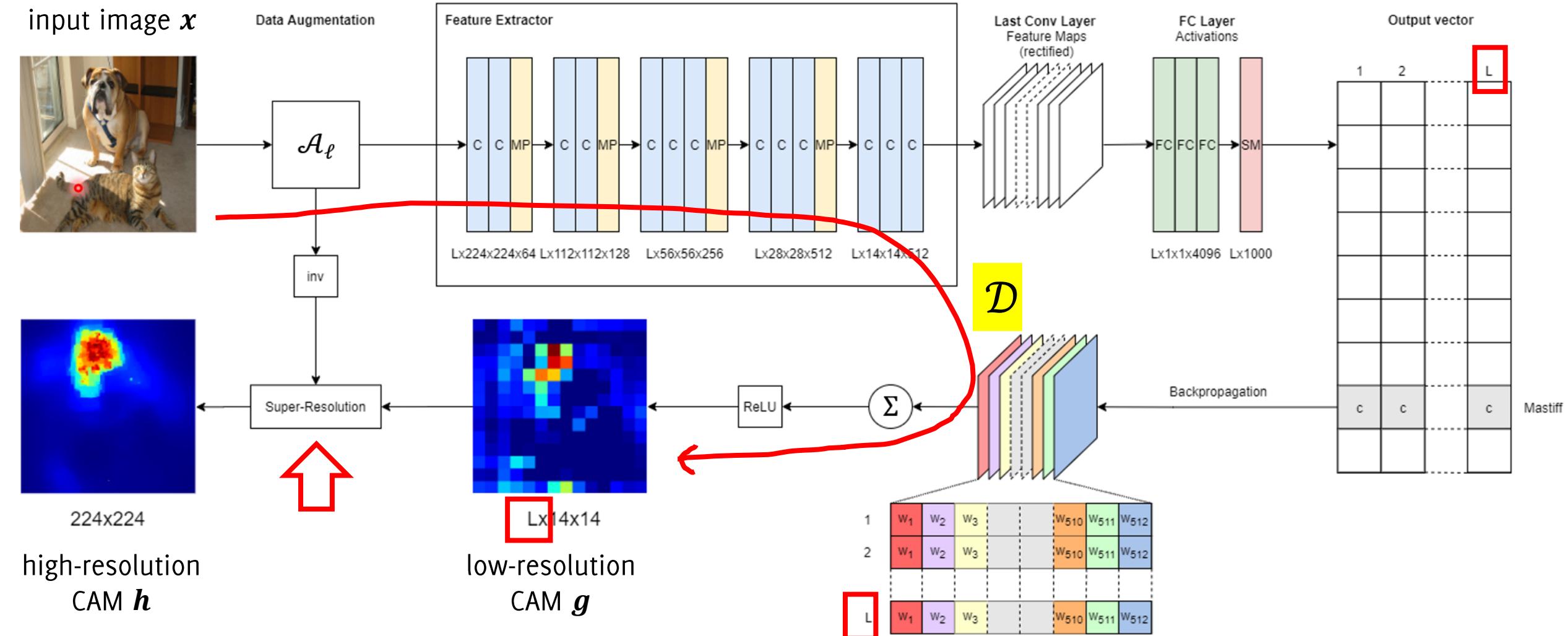
Augmented Grad-CAM



Augmented Grad-CAM



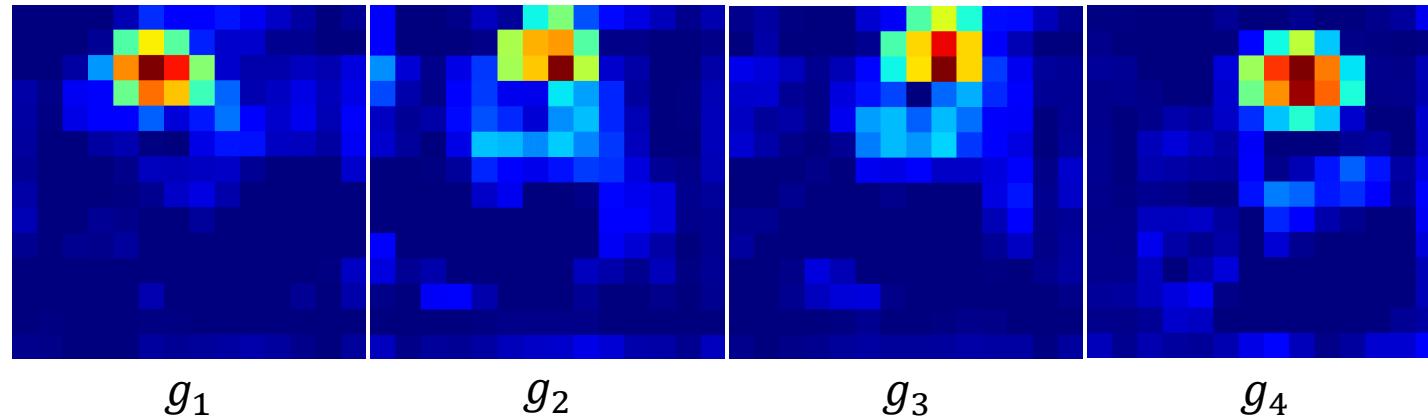
The entire pipeline from the augmented image to the generation of a low-resolution saliency map is seen as the result of the same downsampling operator \mathcal{D} applied to different perturbed variants of the same high-resolution saliency map $\mathcal{A}_l(\mathbf{h})$.



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 \mathcal{A}_l**

CNNs are in general invariant to roto-translations, in terms of predictions, but each g_ℓ actually contains different information



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

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 \mathbf{h}

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

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

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

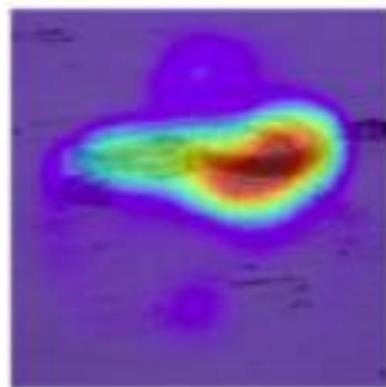
$$TV_{\ell_1}(\mathbf{h}) = \sum_{i,j} \|\partial_x \mathbf{h}(i,j)\| + \|\partial_y \mathbf{h}(i,j)\| \quad (2)$$

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

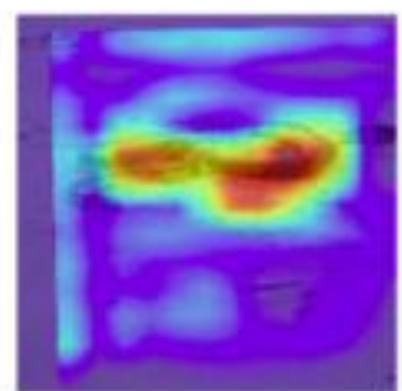
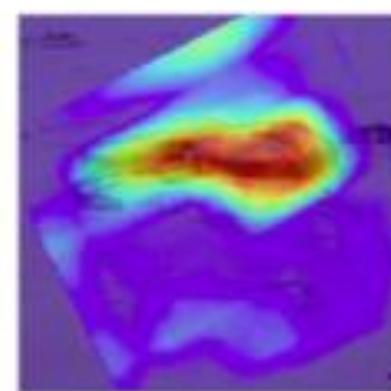
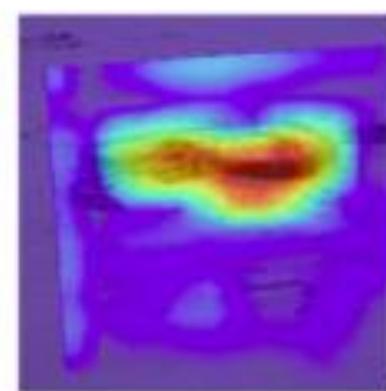
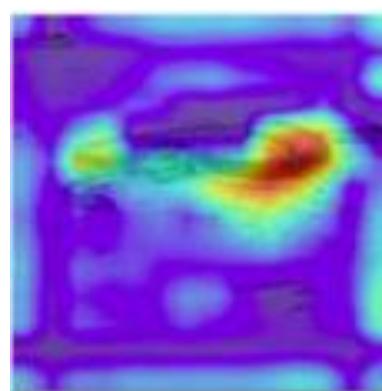
Original
cropped image



Augmented images

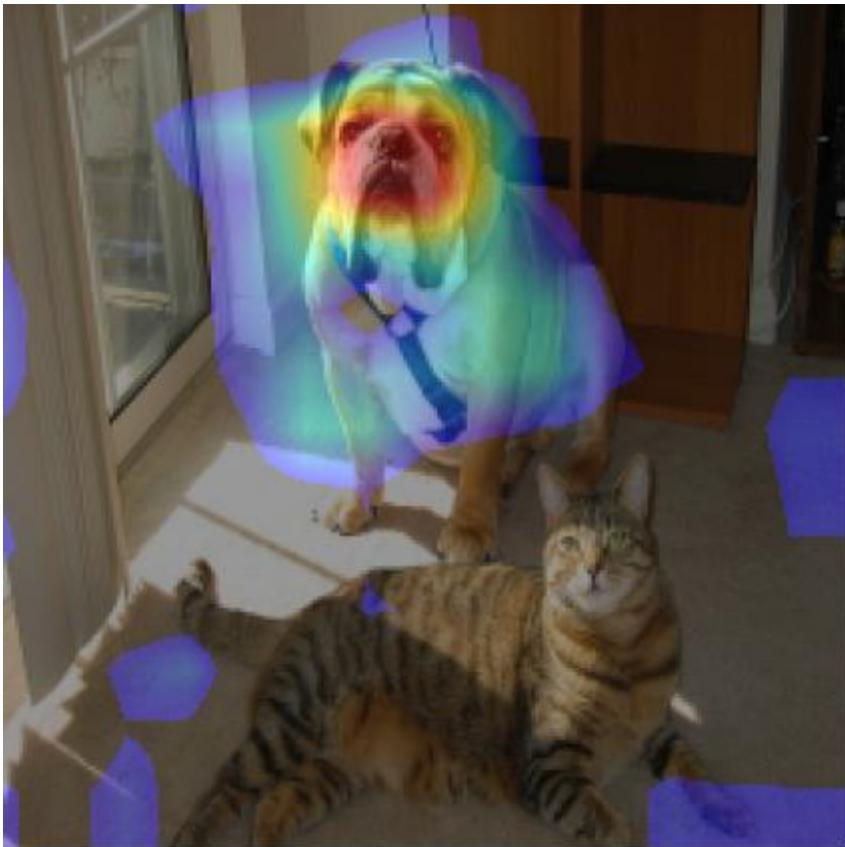


Augmented
Grad-CAM



High-resolution Grad-CAMs superimposed to the original cropped image

Augmented Grad-CAM («Mastiff» class)



(a) Grad-CAM.



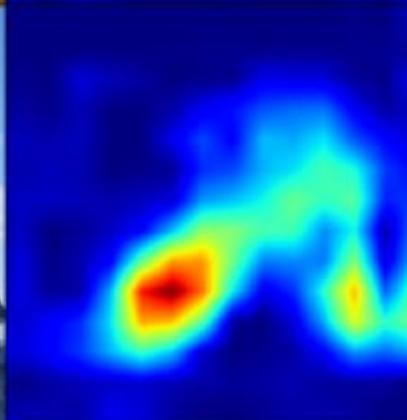
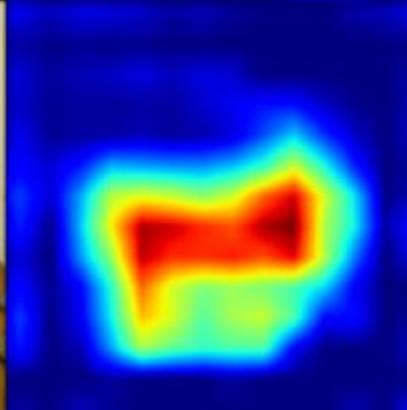
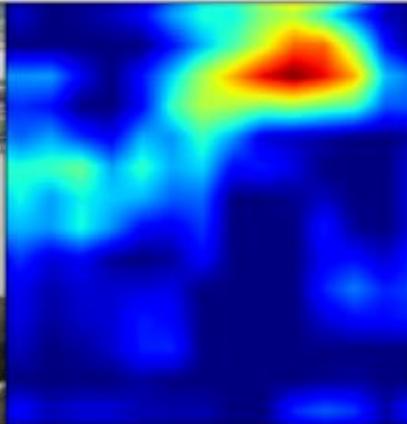
(b) Augmented Grad-CAM.

Augmented Grad-CAM

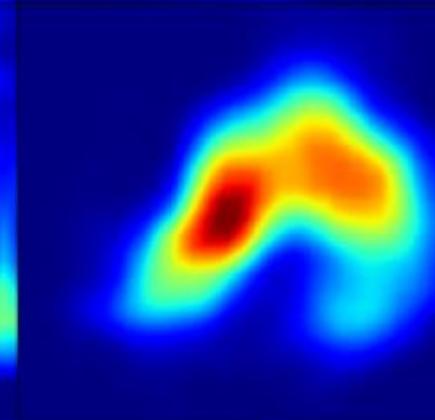
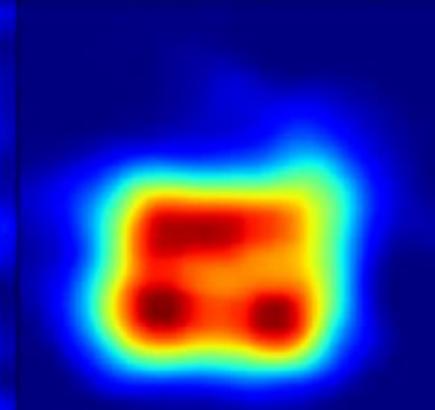
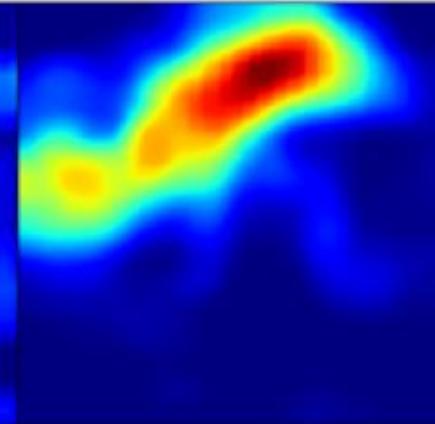
Heat-map
methods:



Grad-CAM



Weighted
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.

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

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

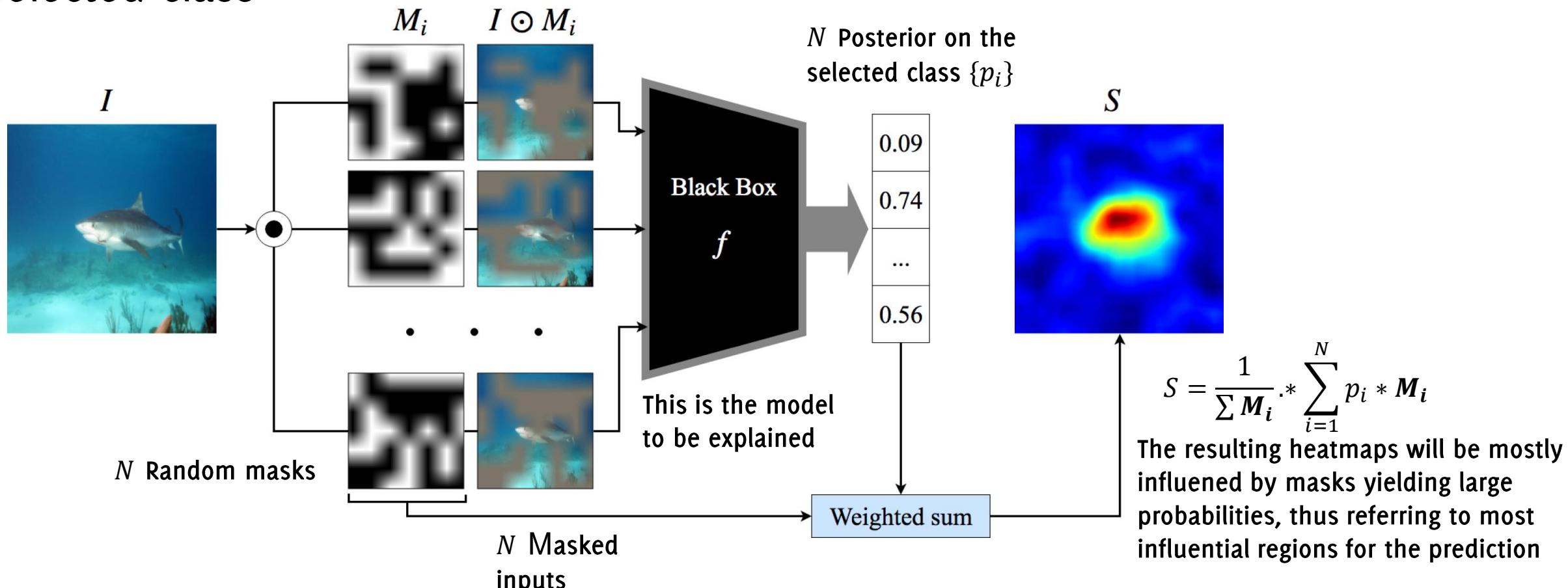
A. Chattpadhyay, A. Sarkar, P. Howlader, and V. N. Balasubramanian, “Grad-CAM++: Generalized gradient-based visual explanations for deep convolutional networks,” WACV, 2018.

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



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.

Perception Visualization

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

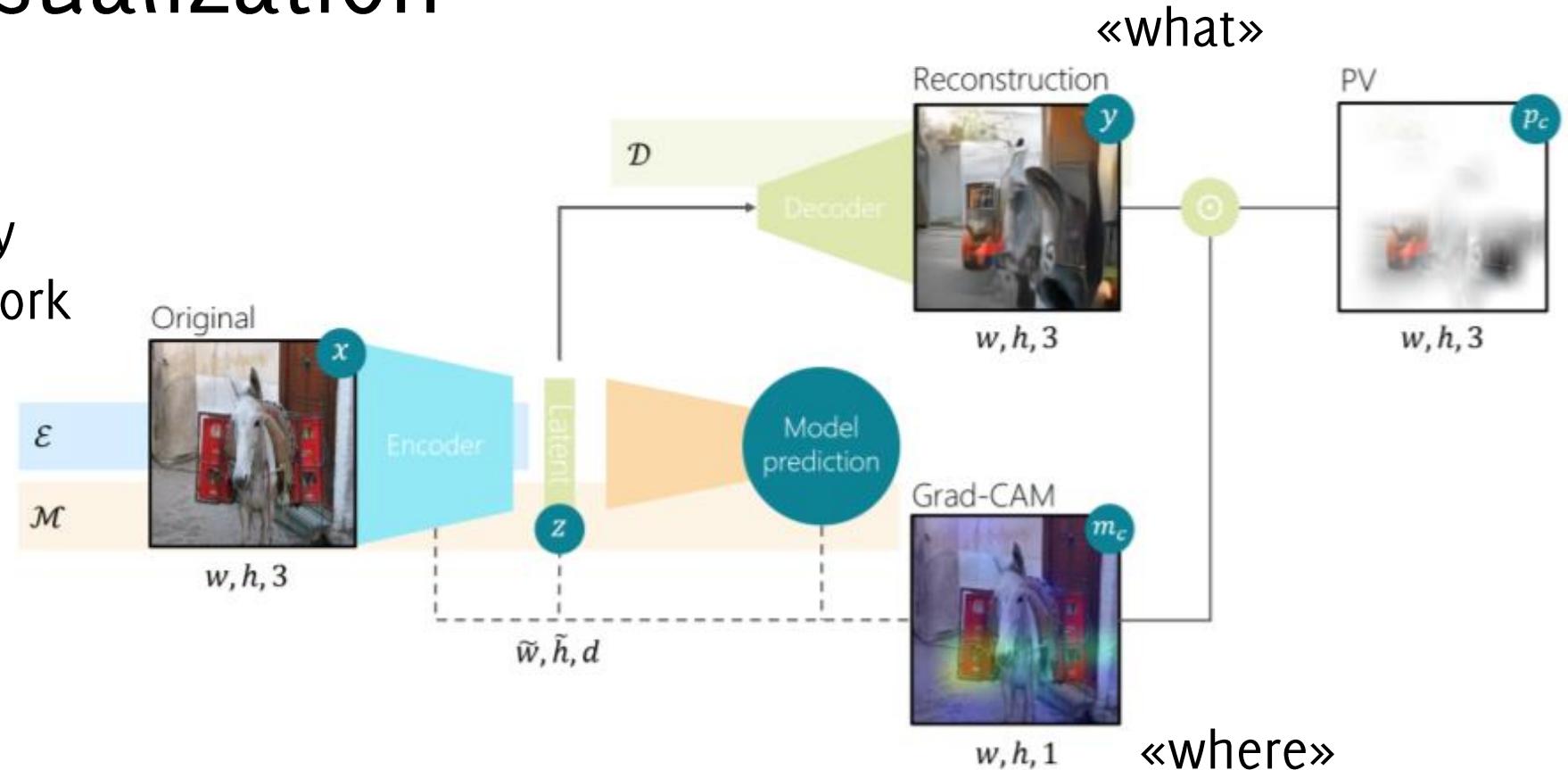


Figure 3: An overview of our method and interactions between the models involved. Encoder \mathcal{E} is a truncation of the model \mathcal{M} which we want to explain, decoder \mathcal{D} is trained to reconstruct the encoder's latent representations. From these, we compute Grad-CAM saliency maps and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

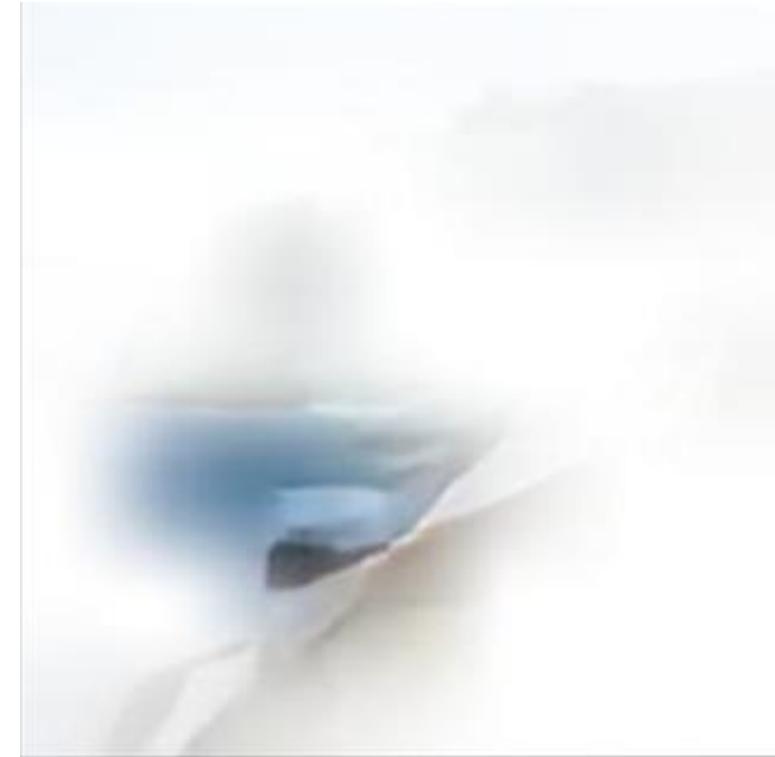
Perception Visualization



Misclassified as “boat”



Saliency doesn't say much

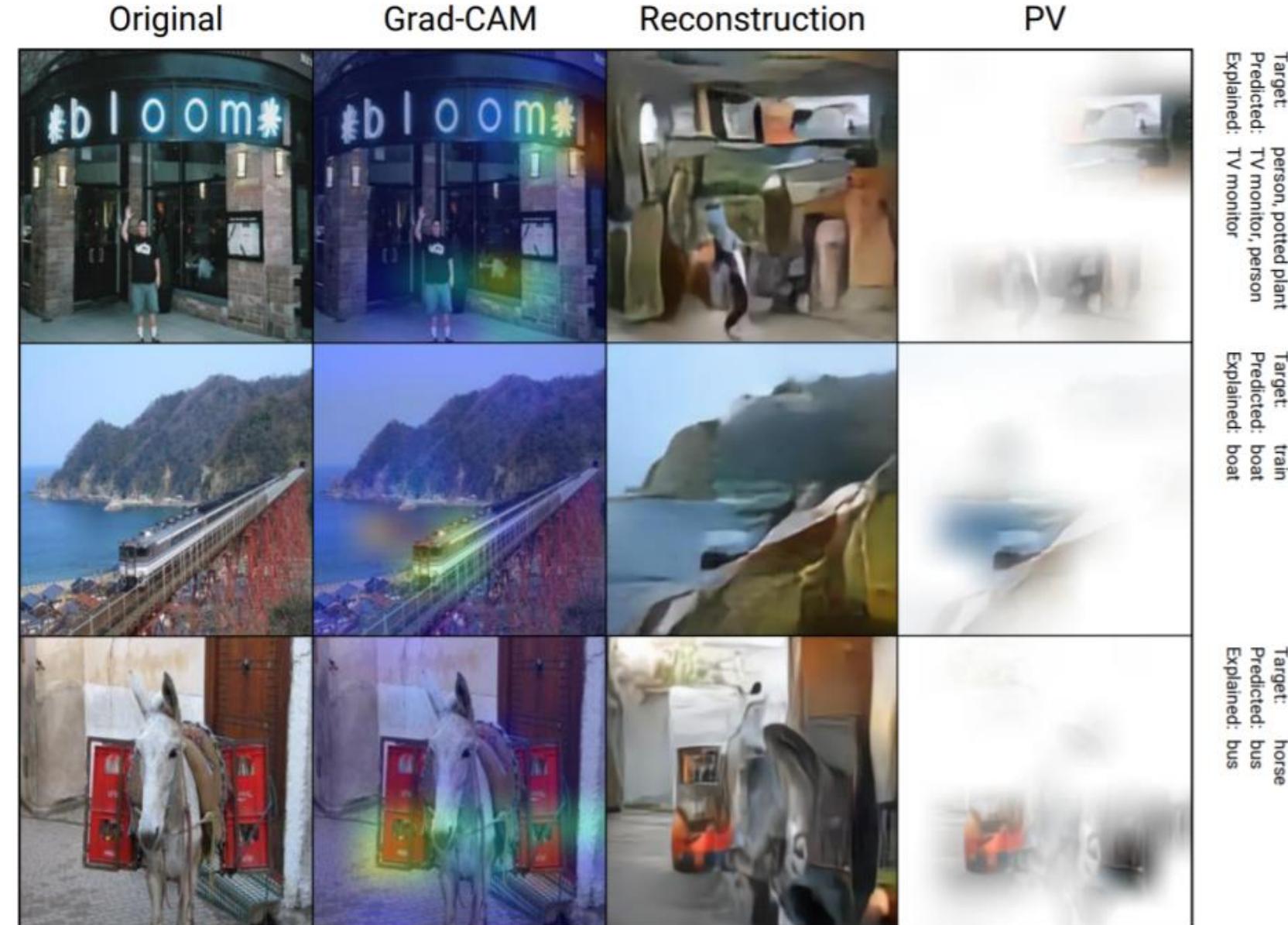


PV shows why

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



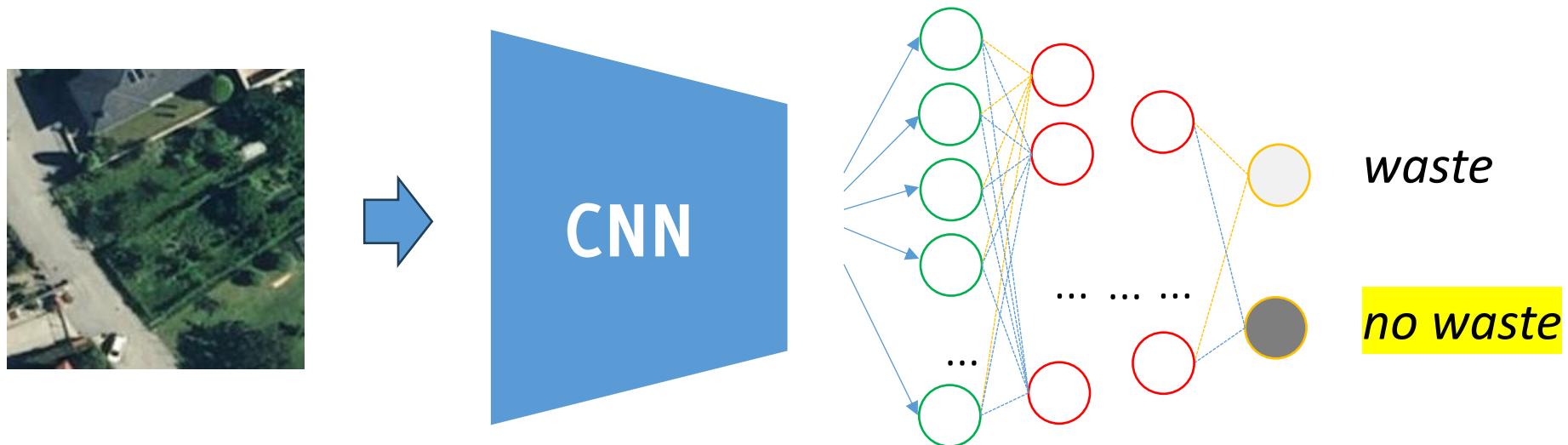
A Project on Weakly Supervised Localization

An European Project Concerning Fight on
Environmental Crime

Deep Learning to inspect Remote Sensing Images

EU funded project aiming at fighting environmental crime

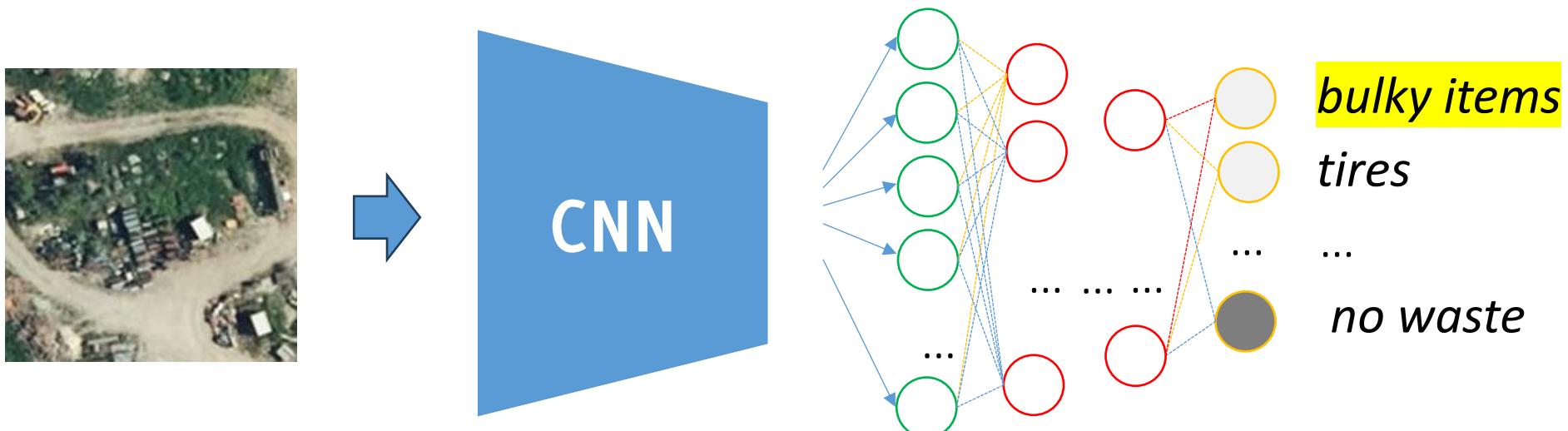
- Illegal landfills detection by DL models analyzing remote sensing images (VHR satellite images)
- Sparse annotations provided (landfills locations)

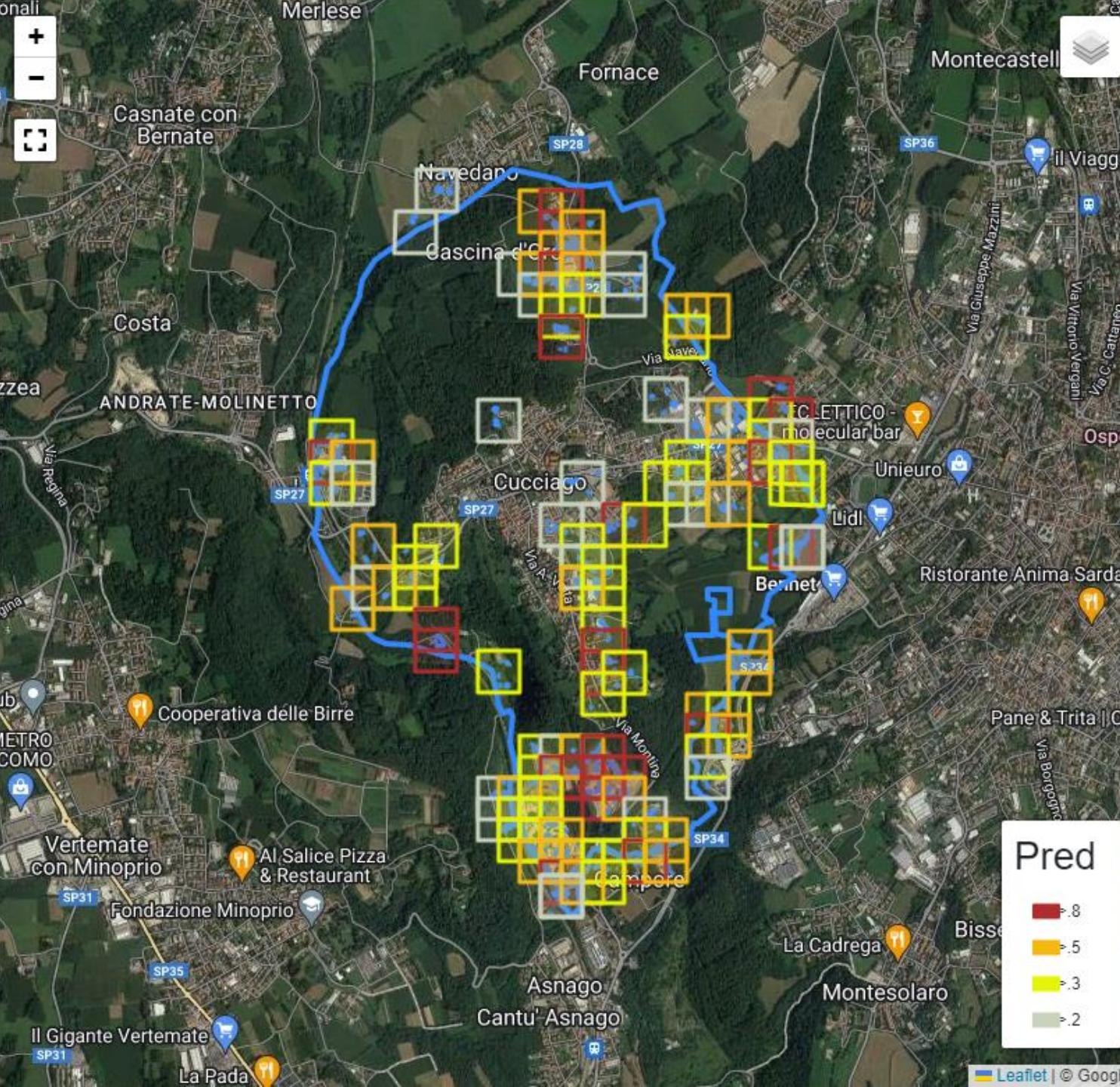


Deep Learning to inspect Remote Sensing Images

Addressing classification waste detection by patch classification

- extensions to segmentation by shift and stitch and *saliency maps* (see *next lecture*)

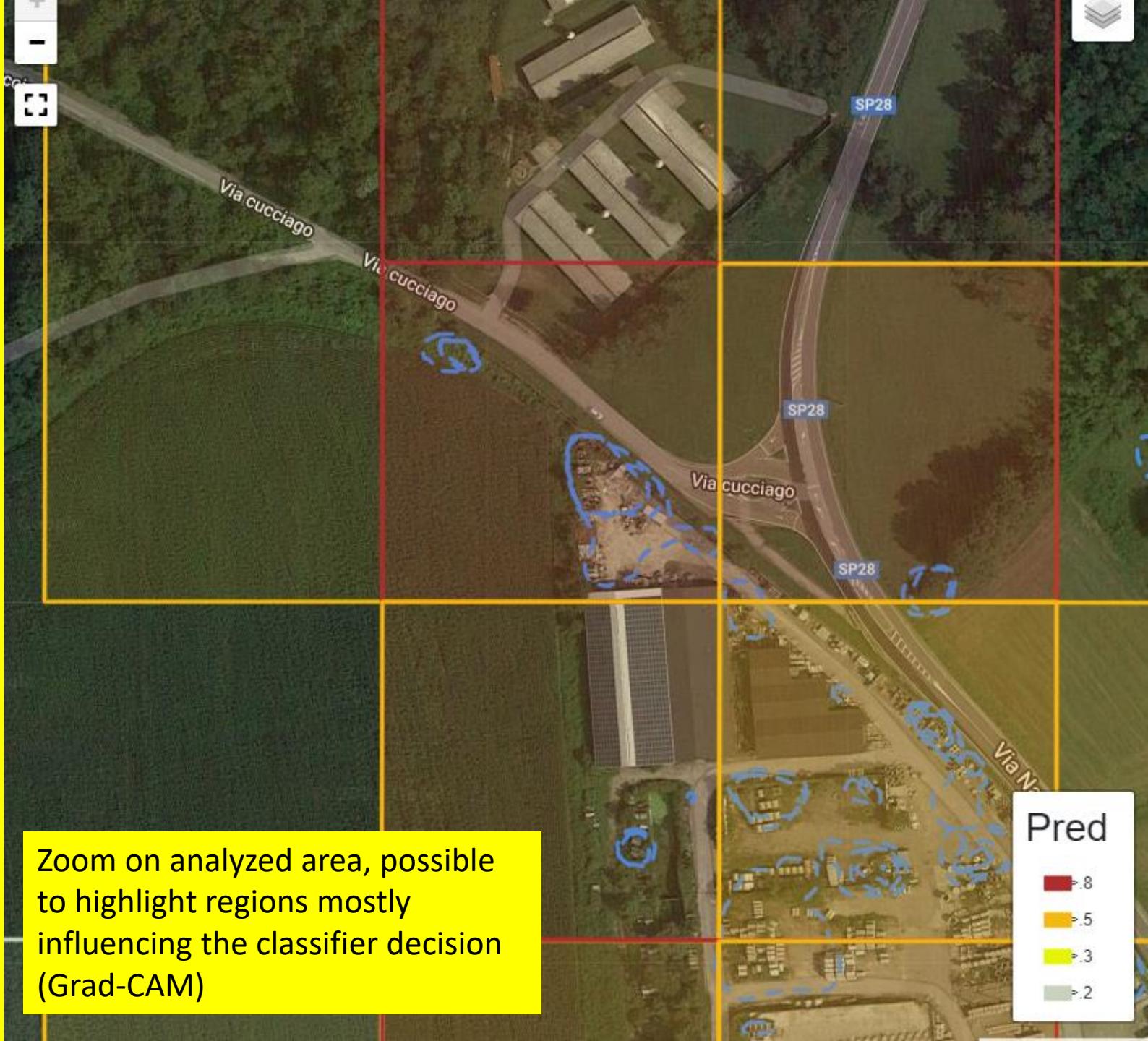
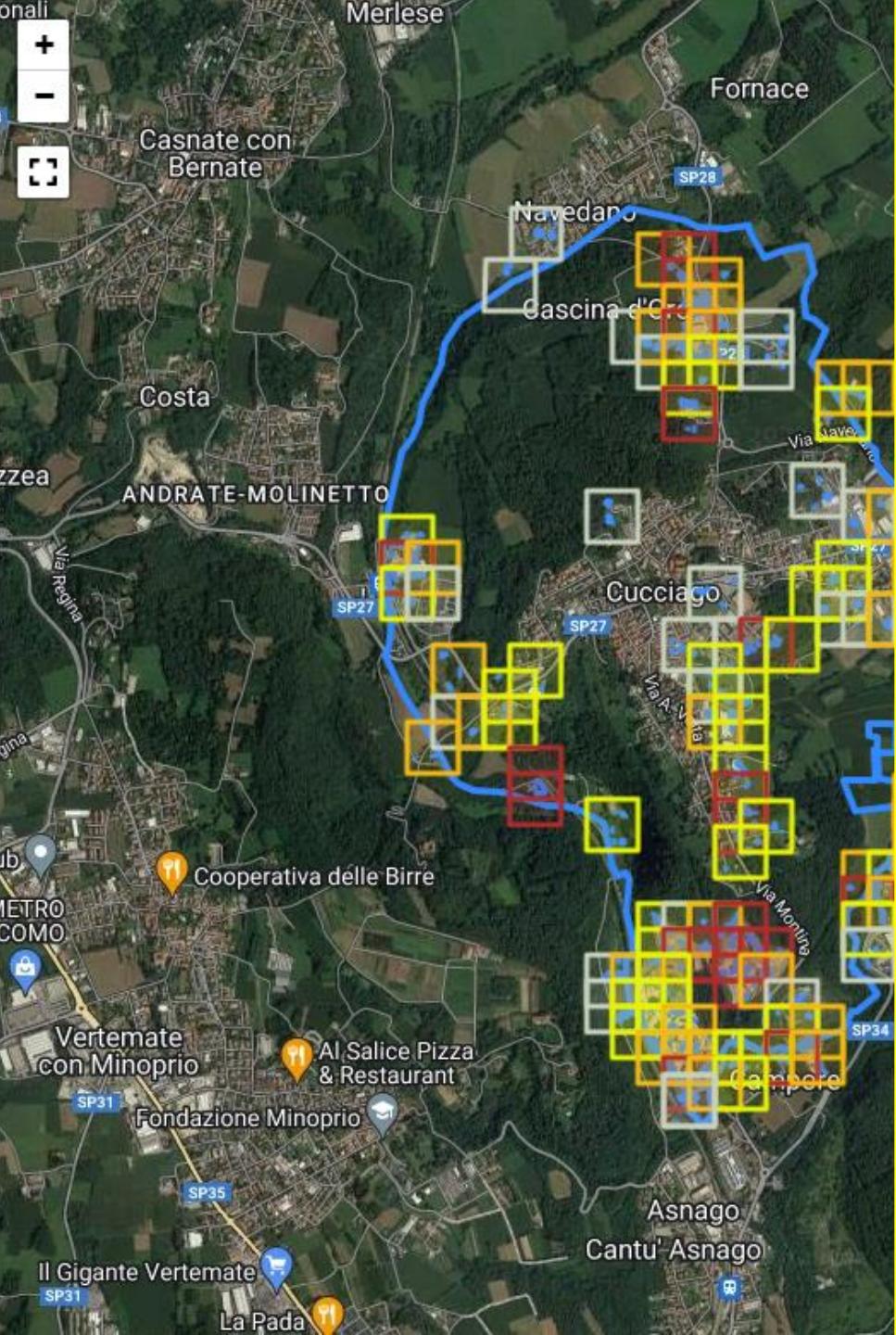




The ODIN Platform within Perivallon

- Tight collaboration with ARPA Lombardia to get annotations
- 100+ municipalities automatically scanned by our system on Google Earth images
- Regular surveys are planned to detect both existing and new waste disposal sites

T. R. Nahime, F. Milani, and P. Fraternali. "ODIN: Pluggable Meta-annotations and Metrics for the Diagnosis of Classification and Localization." *Int. Conf. on Machine Learning, Optimization, and Data Science*.



Examples of segmentation

