Explainable Machine Learning

Neural Network Interpretation

Shim Jaewoong

jaewoong@seoultech.ac.kr

Neural Network Interpretation

- Model-specific method
  - For neural network

- The methods
  - visualize features and concepts learned by a neural network
  - explain individual predictions
  - simplify neural networks.

- To make predictions with a neural network, the data input is passed through many layers of multiplication with the learned weights and through non-linear transformations.
  - There is no chance that we humans can follow the exact mapping from data input to prediction.

Neural Network Interpretation

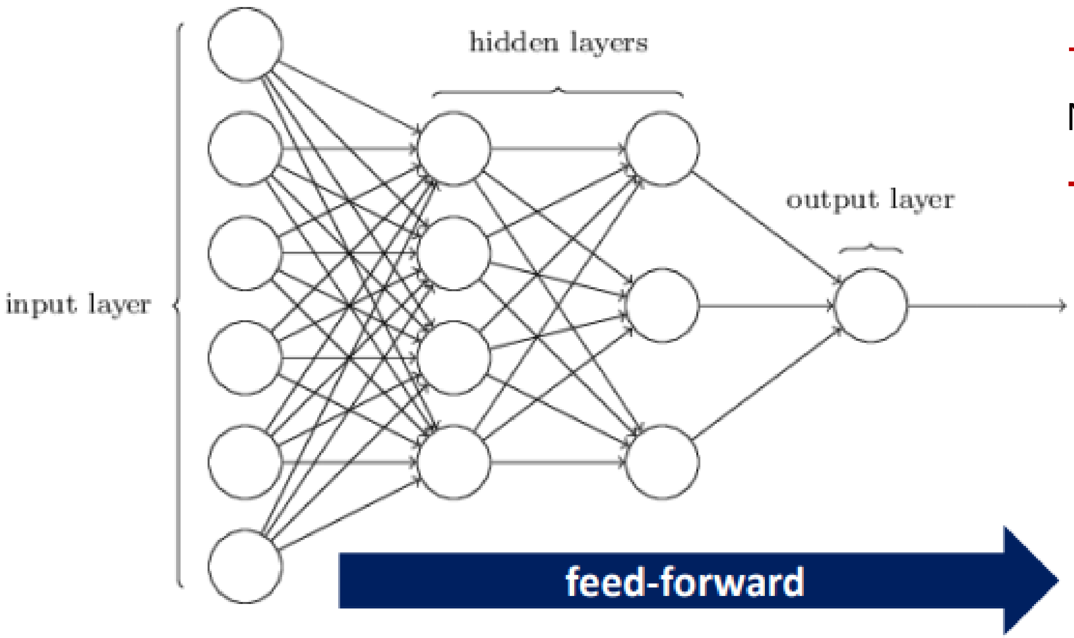
- Why do we consider neural network-specific method?
  - Neural networks learn features and concepts in their <u>hidden layers</u> and we need special tools to uncover them.
  - <u>The gradient can be utilized</u> to implement interpretation methods that are more computationally efficient than model-agnostic methods that look at the model "from the outside".
  - Most other methods in this book are intended for the interpretation of models for tabular data. Image and text data require different methods.

Background:
Neural Network Basic & CNN

Feedforward Neural Network

- **Feedforward Neural Network** (*a.k.a.* MultiLayer Perceptron)
  - Input Layer, Hidden Layers (0 to many), and Output Layer
  - **"feedforward":** information flows through the network from input to output (No feedback/recurrent connections).
  - Multiple layers $f^{(1)}, f^{(2)}, \ldots, f^{(l)}$ are connected in a chain to form

$$f(\boldsymbol{x}) = f^{(l)}\left(\ldots\left(f^{(2)}\left(f^{(1)}(\boldsymbol{x})\right)\right)\right)$$



NN with a single linear output unit and no hidden layer

→ **Linear regression**

NN with a single sigmoid output unit and no hidden layer

→ **Logistic regression**

Feedforward Neural Network

- Training a neural network is not much different from training any other machine learning models.
  - *e.g.*, logistic regression, support vector machine, …

- To train a neural network, we must choose a **cost function** and how to represent the **output/hidden units**.

- The largest difference is that the non-linearity of a neural network causes the **cost function** to become **nonconvex**.
  - many local optima may exist, global optimum cannot be guaranteed.

** Cost function for logistic regression is convex*
→ local optimum = global optimum

Training a Neural Network

- Given a training dataset $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \dots, (\boldsymbol{x}_n, y_n)\}$ such that $\boldsymbol{x}_i = (x_{i1}, \dots, x_{id}) \in \mathbb{R}^d$ is the *i*-th input vector of *d* features and y_i is the corresponding target label.

- The model: $\hat{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$
- The cost function (to be minimized)

$$J(\boldsymbol{\theta}) = \frac{1}{n} \sum_{(x_i, y_i) \in D} L(y_i, \hat{y}_i)$$

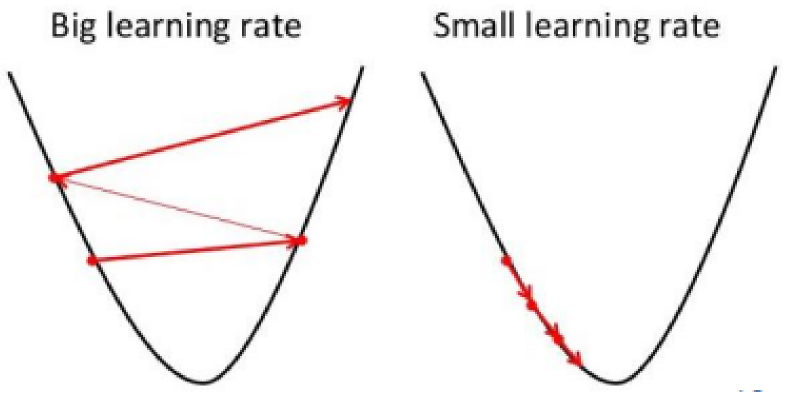
- Training: let's consider simple gradient descent

$$\boldsymbol{\theta} \coloneqq \boldsymbol{\theta} - \epsilon \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

$$\rightarrow \theta_j \coloneqq \theta_j - \epsilon \frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta}), \forall \theta_j \in \boldsymbol{\theta}$$

$\epsilon > 0$ is the learning rate

the cost function for
a deep neural network is non-convex.



how to calculate $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ for a deep neural network?

Training a Neural Network

- Given a training dataset $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \dots, (\boldsymbol{x}_n, y_n)\}$ such that $\boldsymbol{x}_i = (x_{i1}, \dots, x_{id}) \in \mathbb{R}^d$ is the *i*-th input vector of *d* features and y_i is the corresponding target label.

- **For the training set D,**
  - **Forward propagation**: The information from **input** $\boldsymbol{x}$ flows **forward** through the network to get **prediction** $\hat{y}$ and to compute the **cost** $J(\boldsymbol{\theta})$
  - **Backpropagation**: The information from $J(\boldsymbol{\theta})$ flows **backward** through the network to compute the **gradient** of the cost with respect to the parameters $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$

Backpropagation is just a chain rule.

Training a Neural Network

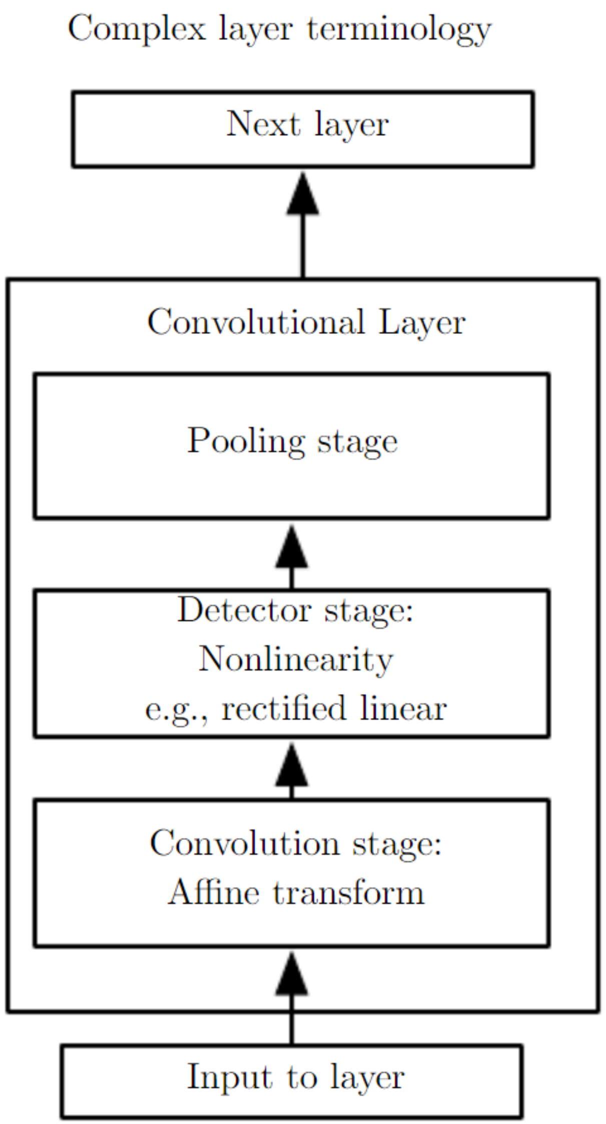
- Step 1. Compute forward propagations for all layers recursively

- Step2. Once done with forward propagation, follow the reverse path.
  - Start from the last layer and for each new layer compute the gradients
  - Cache computations when possible to avoid redundant operations

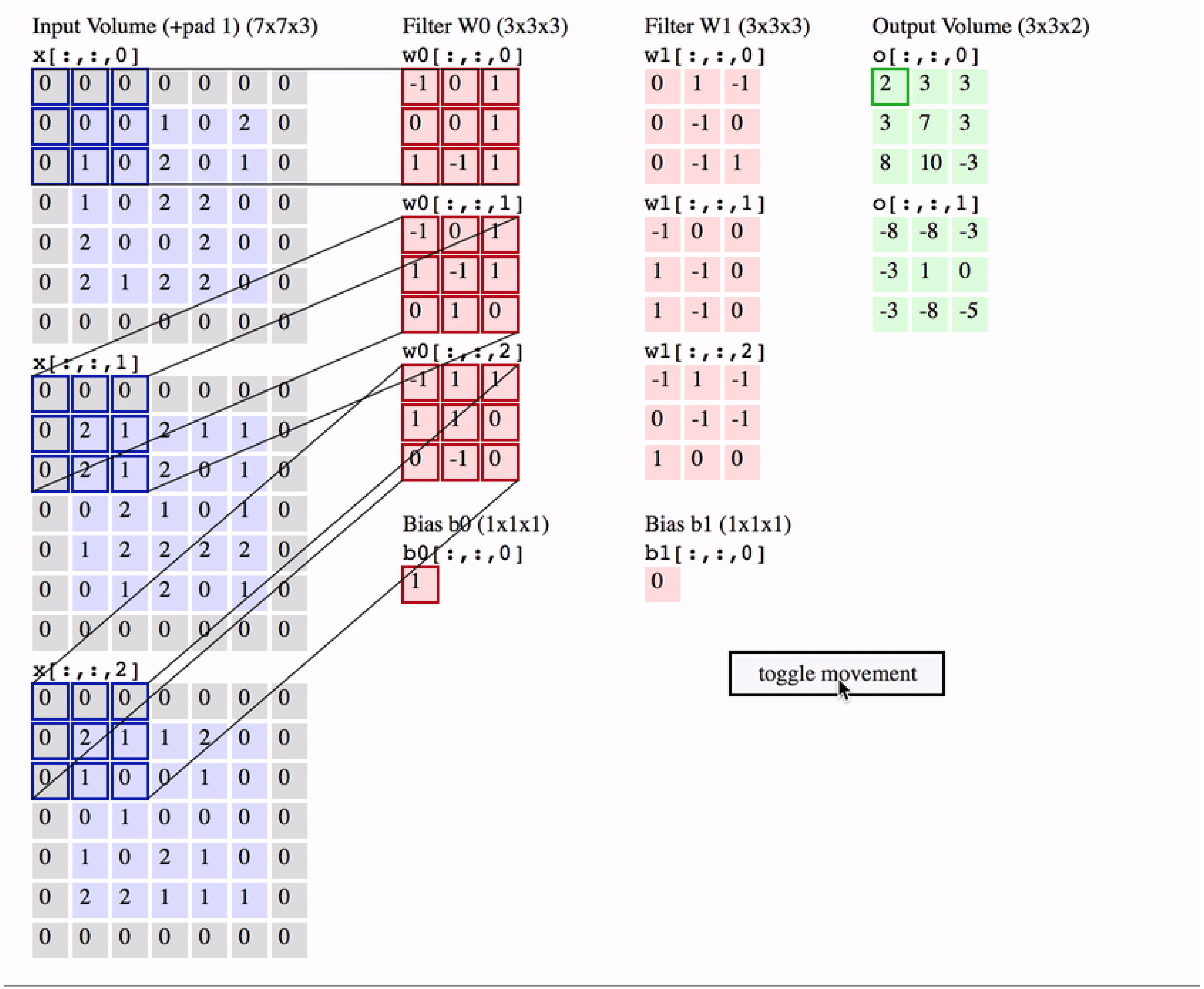
- Step3. Use the gradients ($\frac{\partial L}{\partial \theta}$) with Stochastic Gradient Decent to train

Convolutional Neural Networks

- Convolutional Neural Networks (CNNs)
  - **Neural networks that use convolution in place of general matrix multiplication in at least one of their layers.**
  - Convolutional Layer: Three main operations
    - 1. Convolution
    - 2. Detector (=Activation)
    - 3. Pooling

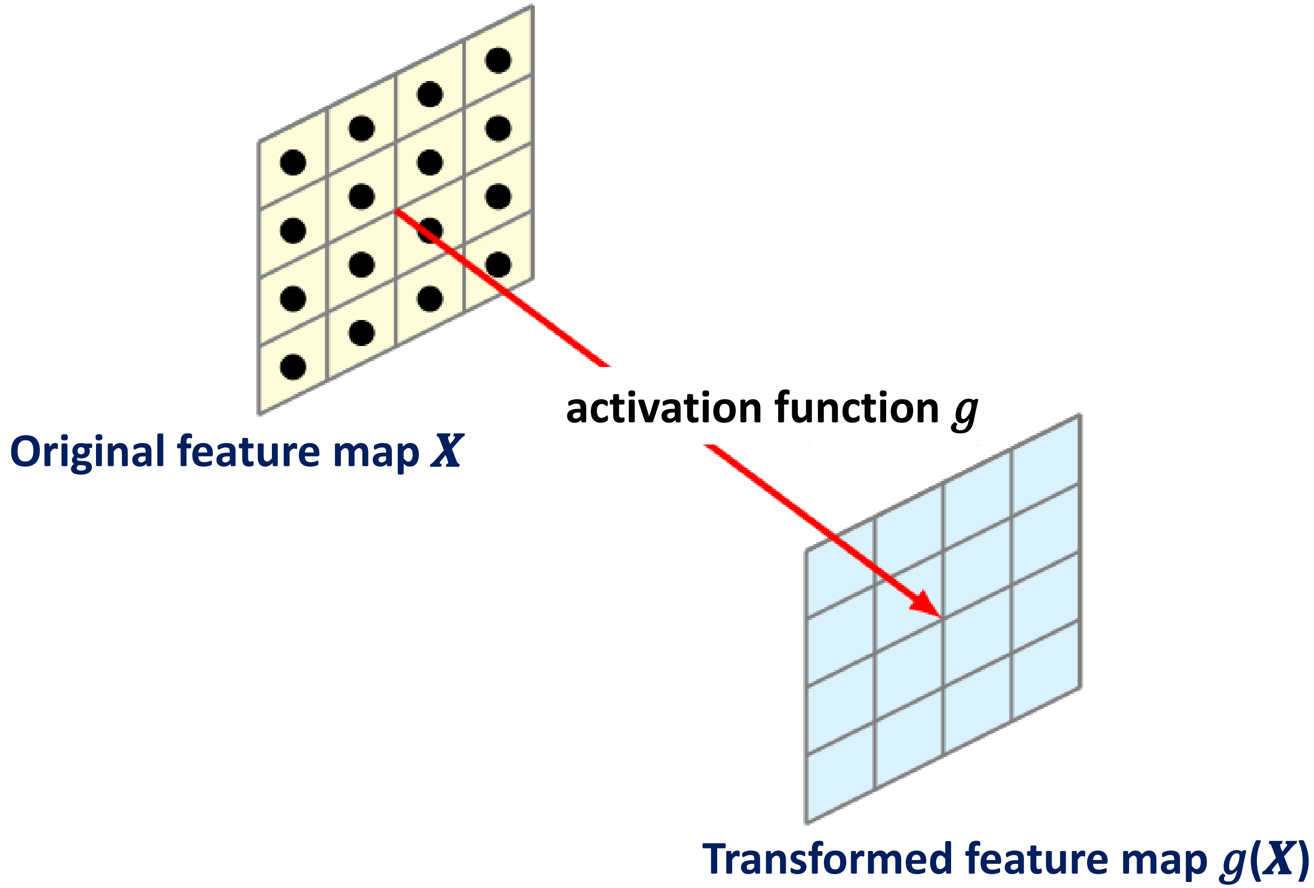


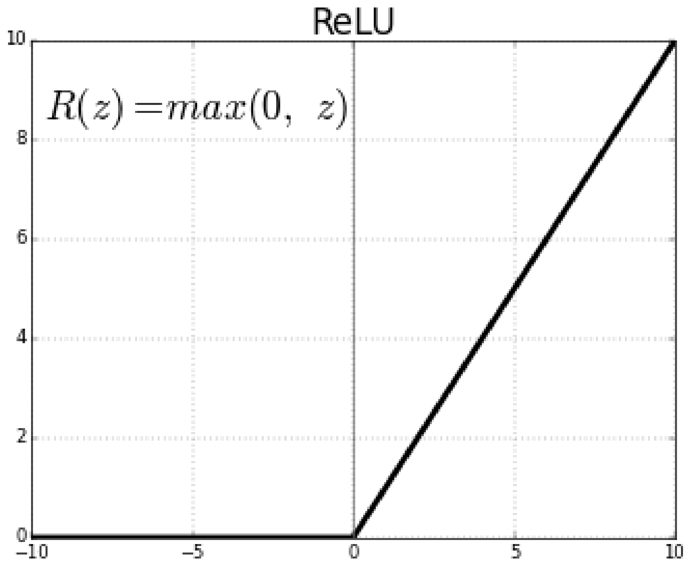
Convolution



Detector (=Activation)

- **Element-wise non-linearity to obtain a transformed feature map**
  - Each feature map is run through a non-linear activation function
  - "ReLU" $g(z)$=max{0,z} is a popular choice.

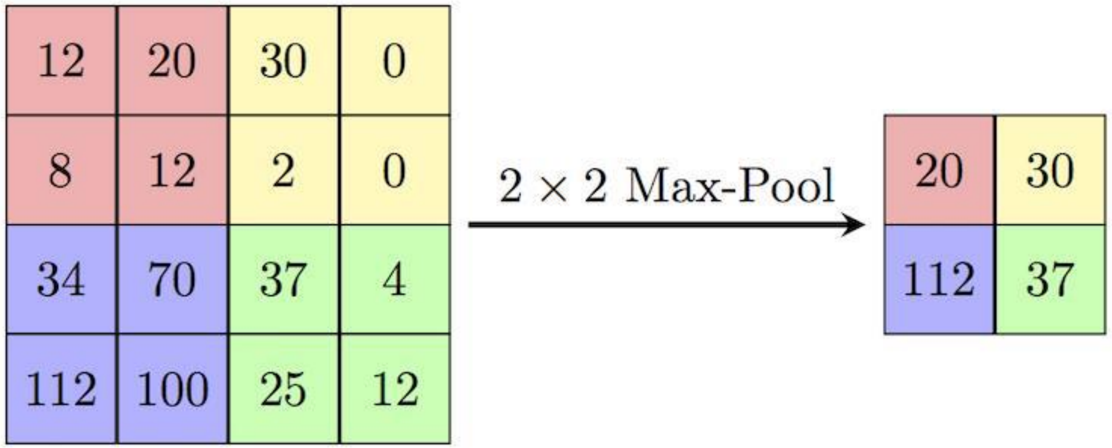


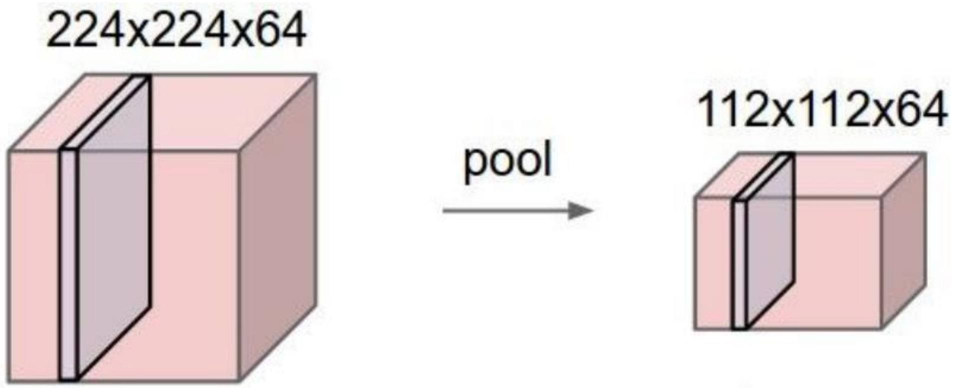


Pooling

- Summarization of each “transformed feature map”
  - makes the representations smaller and more manageable (downsampling)
  - reduces the computational burden on the next layer
  - helps to make the representation *slightly* invariant to small translations of the input.
  - **Various strategies:** max pooling, average pooling, …

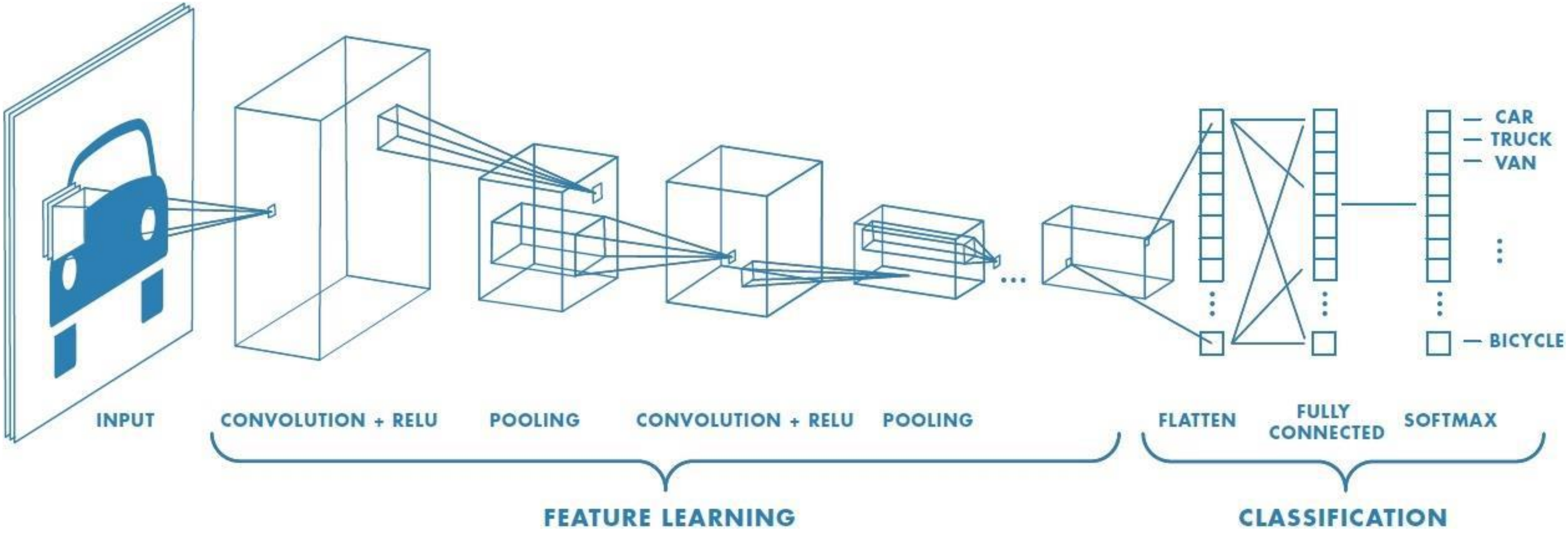
***Example*: 2x2 max pooling with stride (step size) 2**





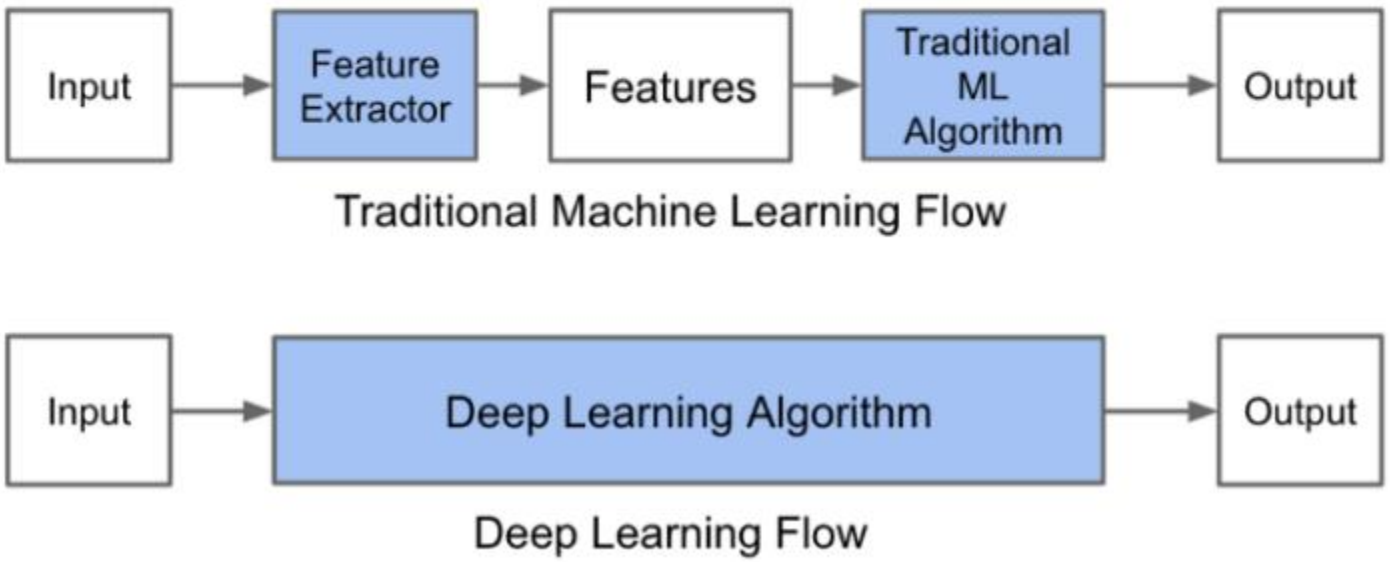
Convolutional Neural Networks

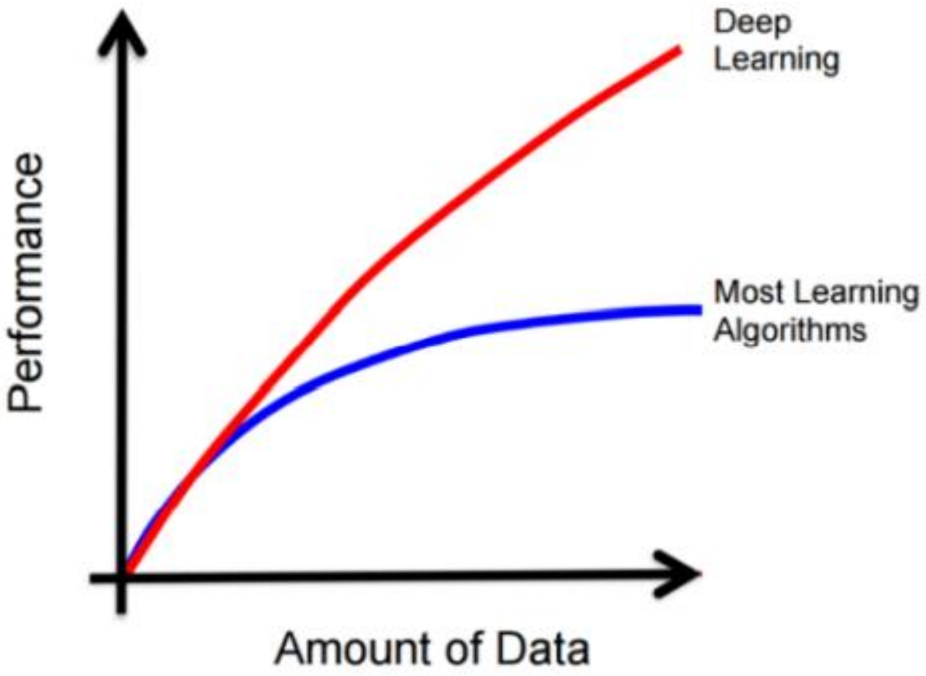
- **A CNN is a stack of convolutional layers (convolution, detector, and pooling) and fully-connected layers**



Deep Learning

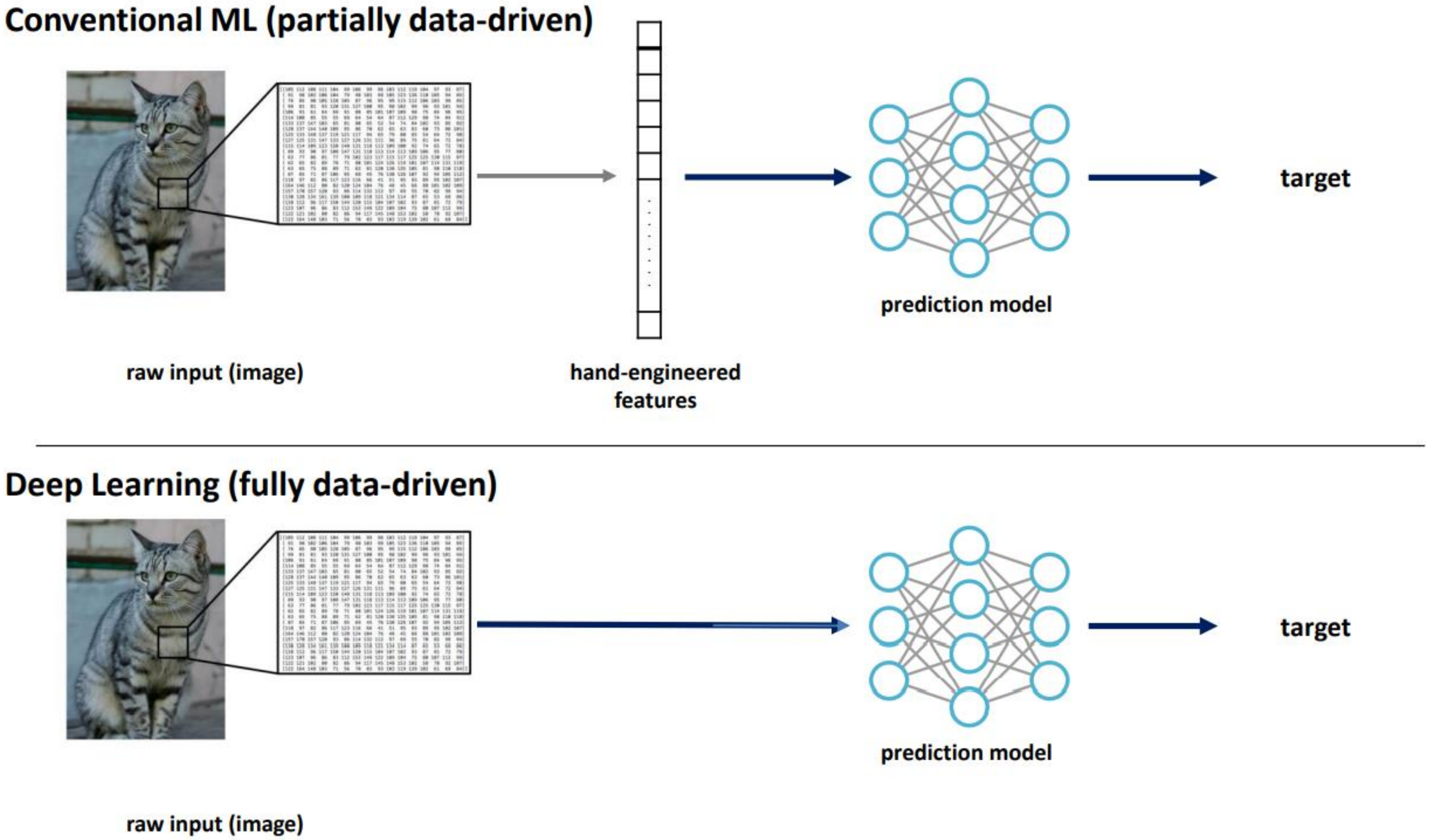
- No need for feature engineering





Deep Learning

- No need for feature engineering
  - SVM: We need to create new features based on color, frequency domain, edge detectors and so on.
  - CNN: The image is fed into the network in its raw form (pixels)

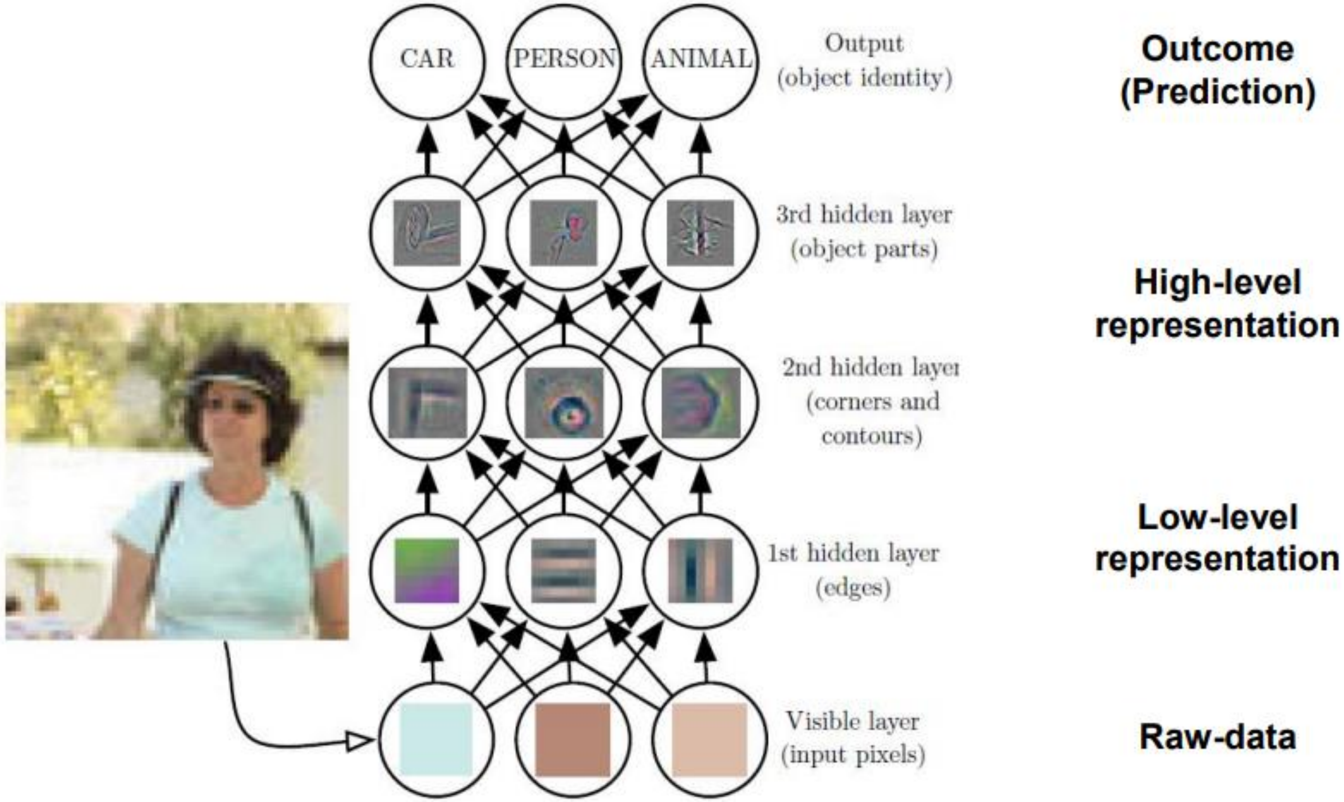


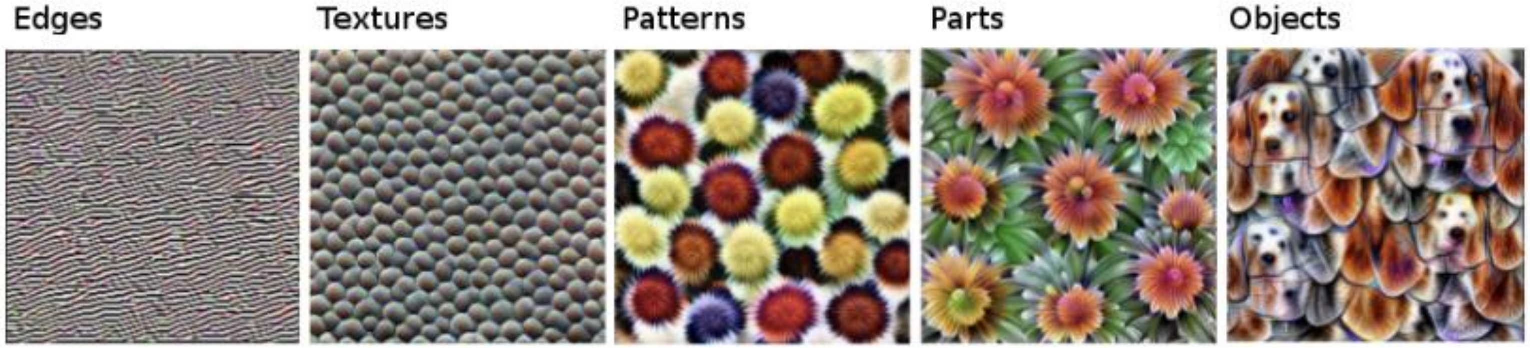
Deep Learning

- **Deep Learning** is based on a cascade of multiple layers of nonlinear processing units for feature extraction and transformation.

- **Higher layers of representation**: amplify important aspects of the input, suppress irrelevant variations.

- The representations are not designed by human experts, but are learned from raw data using a general purpose learning procedure.



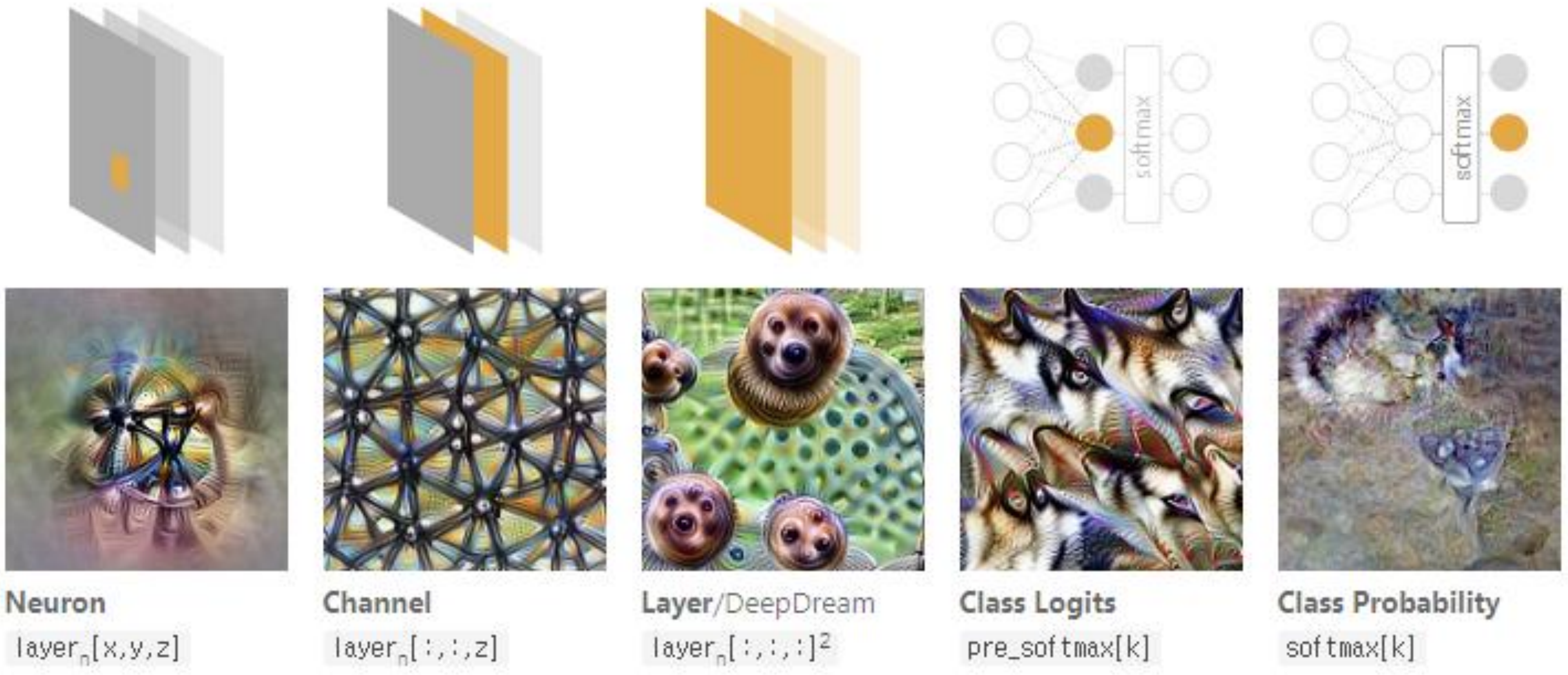


Learned Features

Feature Visualization

https://distill.pub/2017/feature-visualization/

- **Feature Visualization through Optimization**
  - finding the input that **maximizes the activation** of the unit.



If we want to understand individual features, we can search for examples where they have high values — either for a *neuron* at an individual position, or for an entire *channel*.

Feature Visualization

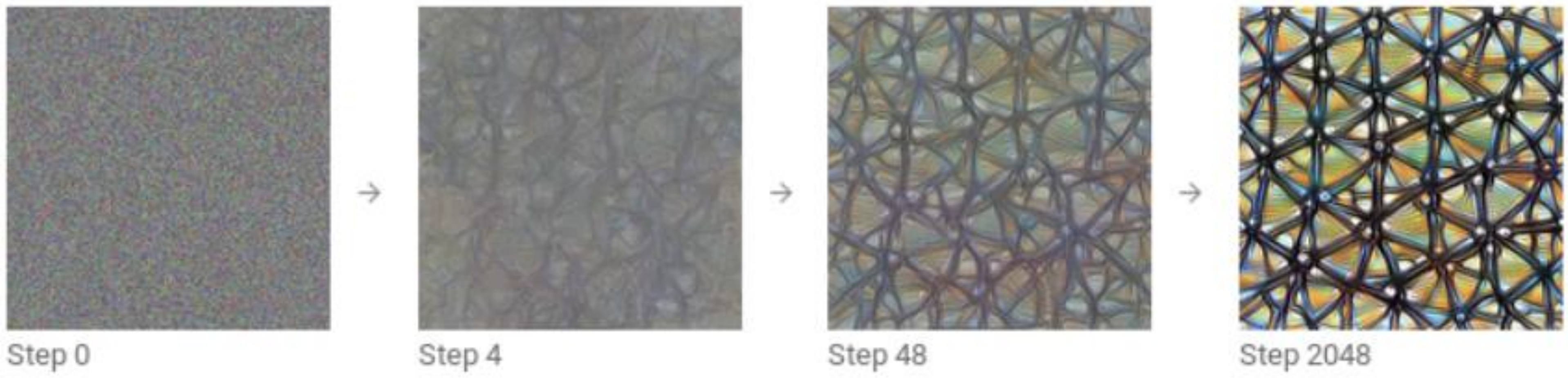
- **Feature Visualization through Optimization**
  - We assume that the weights of the neural network are fixed, which means that the network is trained.
  - We are looking for a new image that maximizes the (mean) activation of a unit

For a single neuron: $$img^* = \arg\max_{img} h_{n,x,y,z}(img)$$

For a single channel: $$img^* = \arg\max_{img} \sum_{x,y} h_{n,x,y,z}(img)$$

Feature Visualization

- **Feature Visualization through Optimization**
  - generate new images, **starting from random noise.**



Starting from random noise, we optimize an image to activate a particular neuron

Feature Visualization

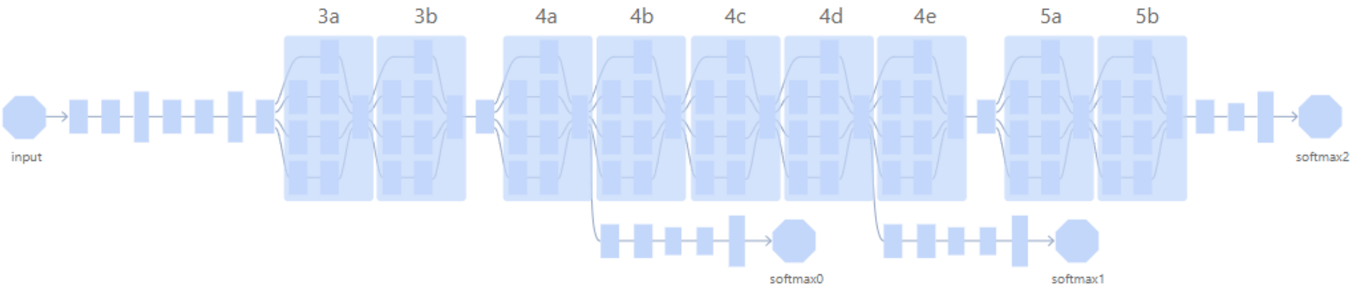
- **Feature Visualization through Optimization**
  - an image full of noise and nonsensical high-frequency patterns that the network responds strongly to.

  - **Regularization options** (to make realistic examples)
    - Frequency penalization
      - explicitly penalize variance between neighboring pixels
    - Transformation robustness
      - find examples that still activate the optimization target highly even if we slightly transform them.
      - Concretely, this means that we stochastically jitter, rotate or scale the image before applying the optimization step.
    - Learned priors
      - Optimization within the latent space.
      - with generative adversarial networks (GANs) or denoising autoencoders.

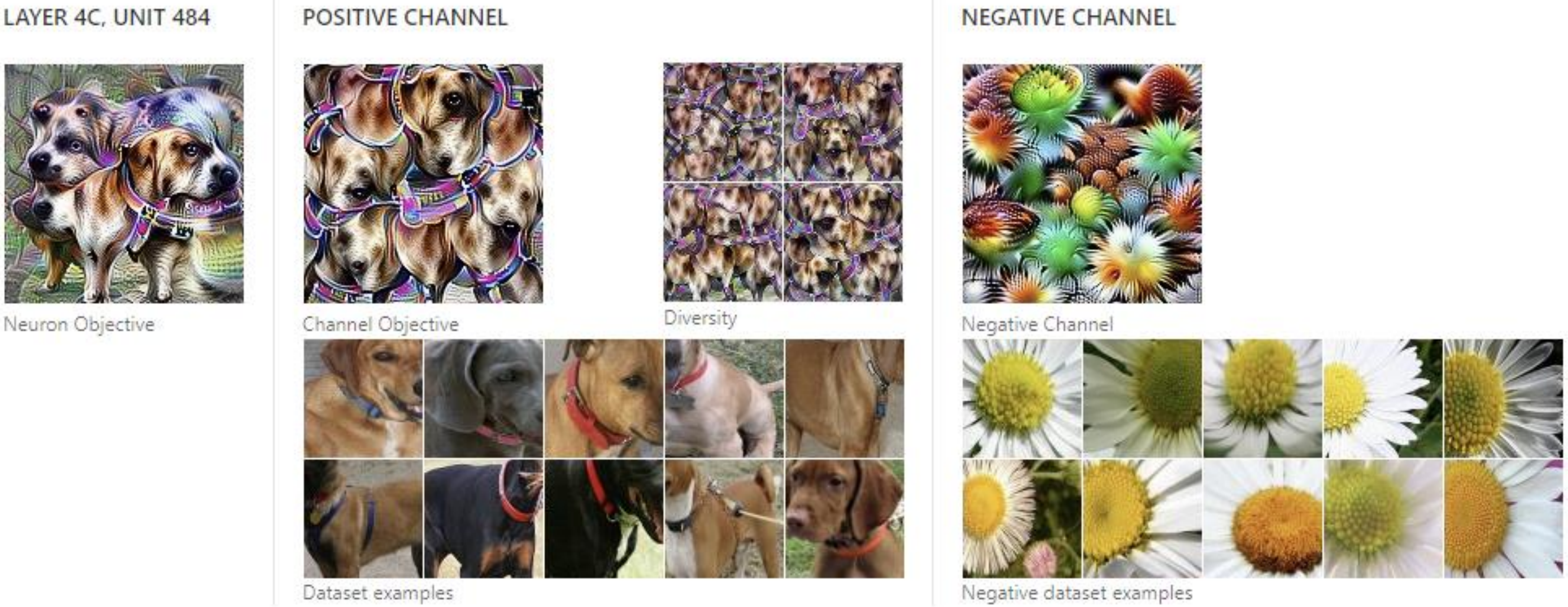
Feature Visualization

- **Feature Visualization through Optimization**

Examples: https://distill.pub/2017/feature-visualization/appendix/



This appendix contains layers 3a through 5b of GoogLeNet.



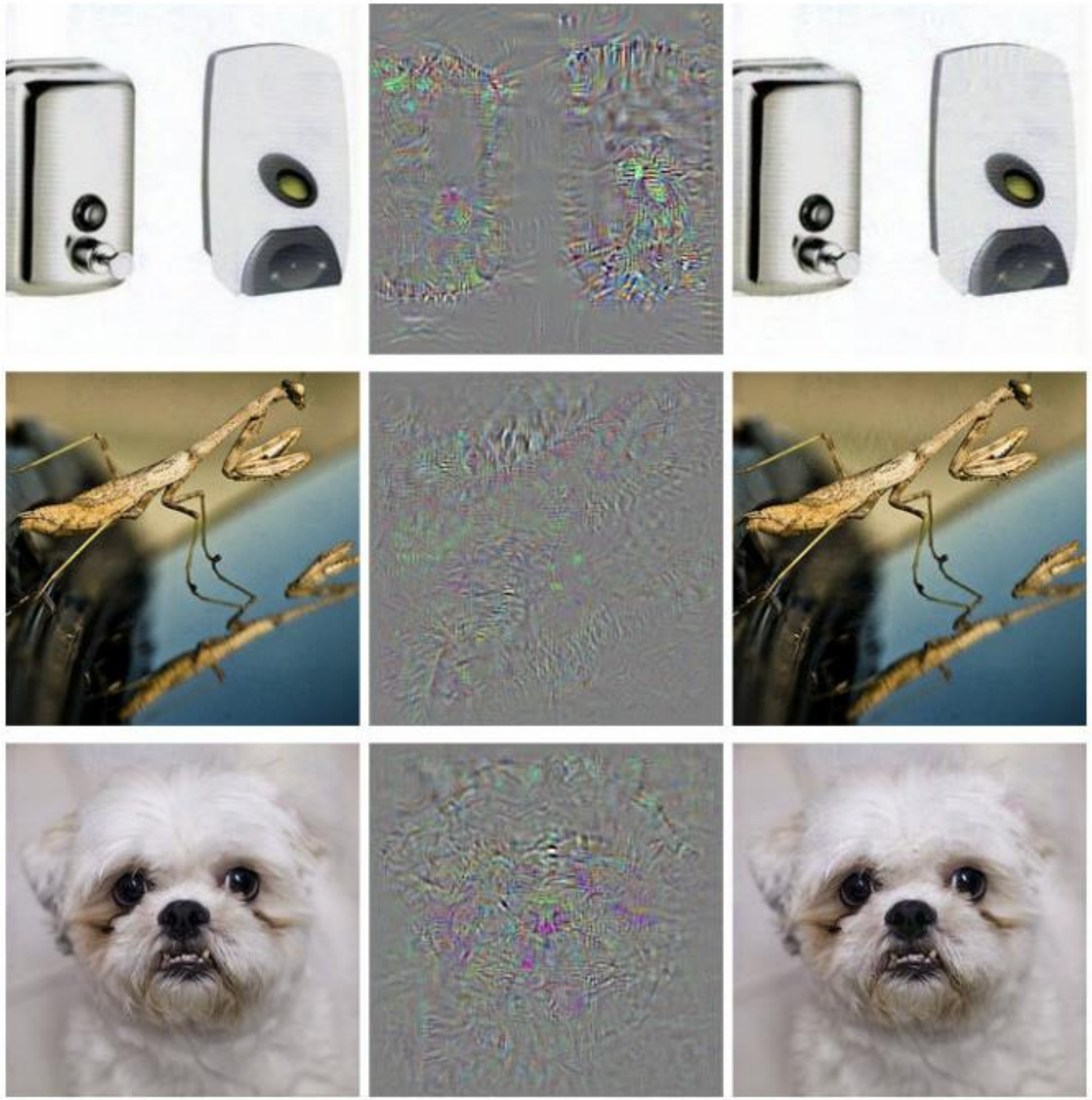
Feature Visualization

Szegedy, Christian, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. “Intriguing properties of neural networks.” arXiv preprint arXiv:1312.6199 (2013)

- **Feature visualization vs Adversarial Examples**
  - Both techniques maximize the activation of a neural network unit
    - Adversarial examples look for the maximum activation of the neuron for the adversarial (= incorrect) class
  - For adversarial examples, we start with the image for which we want to generate the adversarial image.
  - For feature visualization, we start with the random noise.

adversarial examples were generated by minimizing the following function with respect to r:

$$loss(\hat{f}(x+r), l) + c \cdot |r|$$

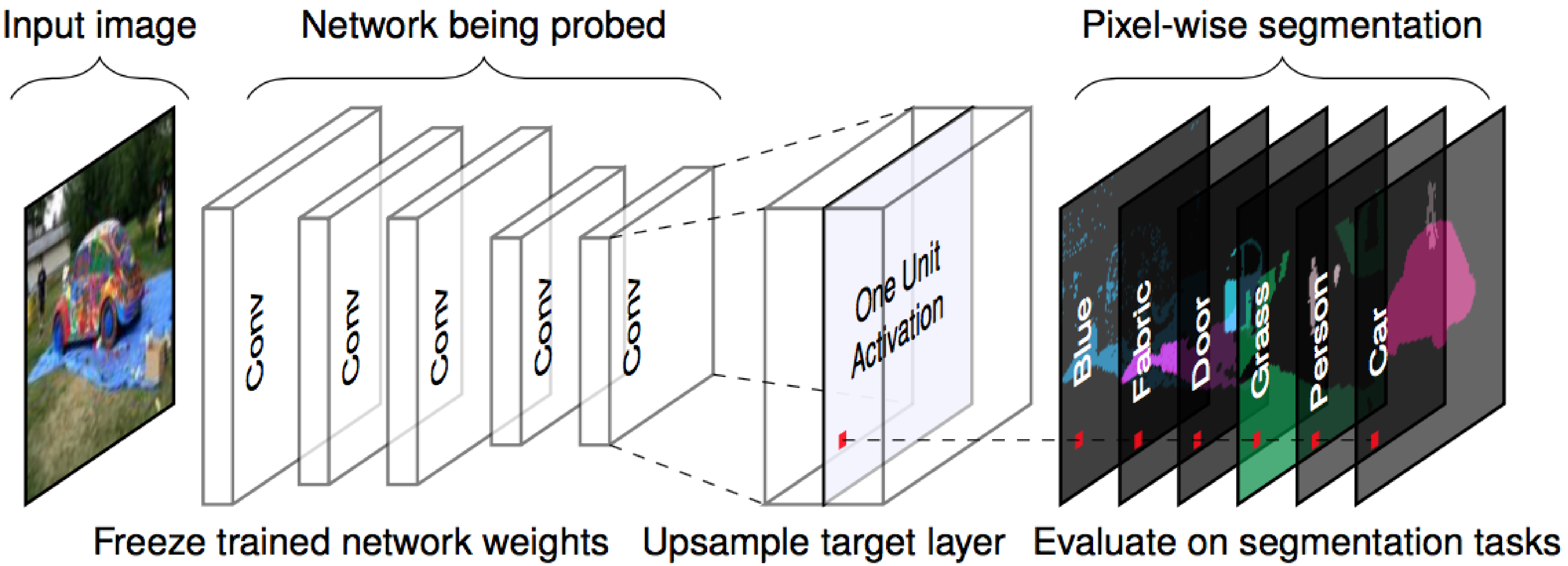


All images in the left column are correctly classified. The middle column shows the (magnified) error added to the images to produce the images in the right column all categorized (incorrectly) as “Ostrich”.

Network Dissection

Bau, David, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. "Network dissection: Quantifying interpretability of deep visual representations." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 6541-6549 (2017).

- **Goal: From Visualization to Interpretation**
  1. Get images with human-labeled visual concepts, from stripes to skyscrapers.
  2. Measure the CNN channel activations for these images.
  3. Quantify the alignment of activations and labeled concepts.



For a given input image and a trained network (fixed weights), we propagate the image forward to the target layer, upscale the activations to match the original image size and compare the maximum activations with the ground truth pixel-wise segmentation.

Network Dissection

http://netdissect.csail.mit.edu/

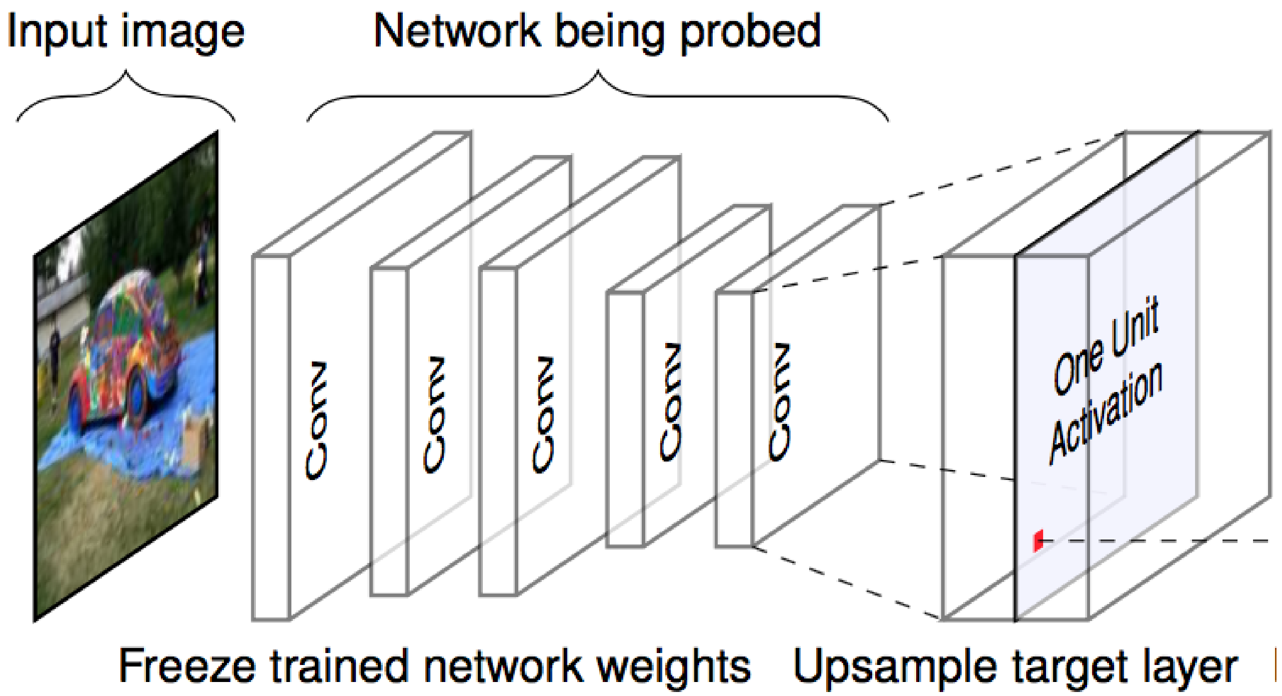
- **Step 1: Broden dataset** (broadly and densely labeled data)
  - pixel-wise labeled images with concepts of different abstraction levels (from colors to street scenes)
  - Bau & Zhou et al. combined a couple of datasets with pixel-wise concepts
  - “Broden” contains 60,000 images with over 1,000 visual concepts in different abstraction levels:
    - 468 scenes, 585 objects, 234 parts, 32 materials, 47 textures and 11 colors.

ADE20K
Zhou et al, CVPR’17
Pascal Context
Mottaghi et al, CVPR’14
Pascal Part
Chen et al, CVPR’14
Open-Surfaces
Bell et al, SIGGRAPH’14
Describable Textures
Cimpoi et al, CVPR’14
Colors

Total = 63,305 images
1,197 visual concepts

Network Dissection

- **Step 2: Retrieve network activations**
  - we create the masks of the top activated areas per channel and per image
    - For convolutional neurons, compute their activation map
    - what is the output of a particular convolutional filter for a given image
    - Threshold this activation map to convert it to a binary activation map

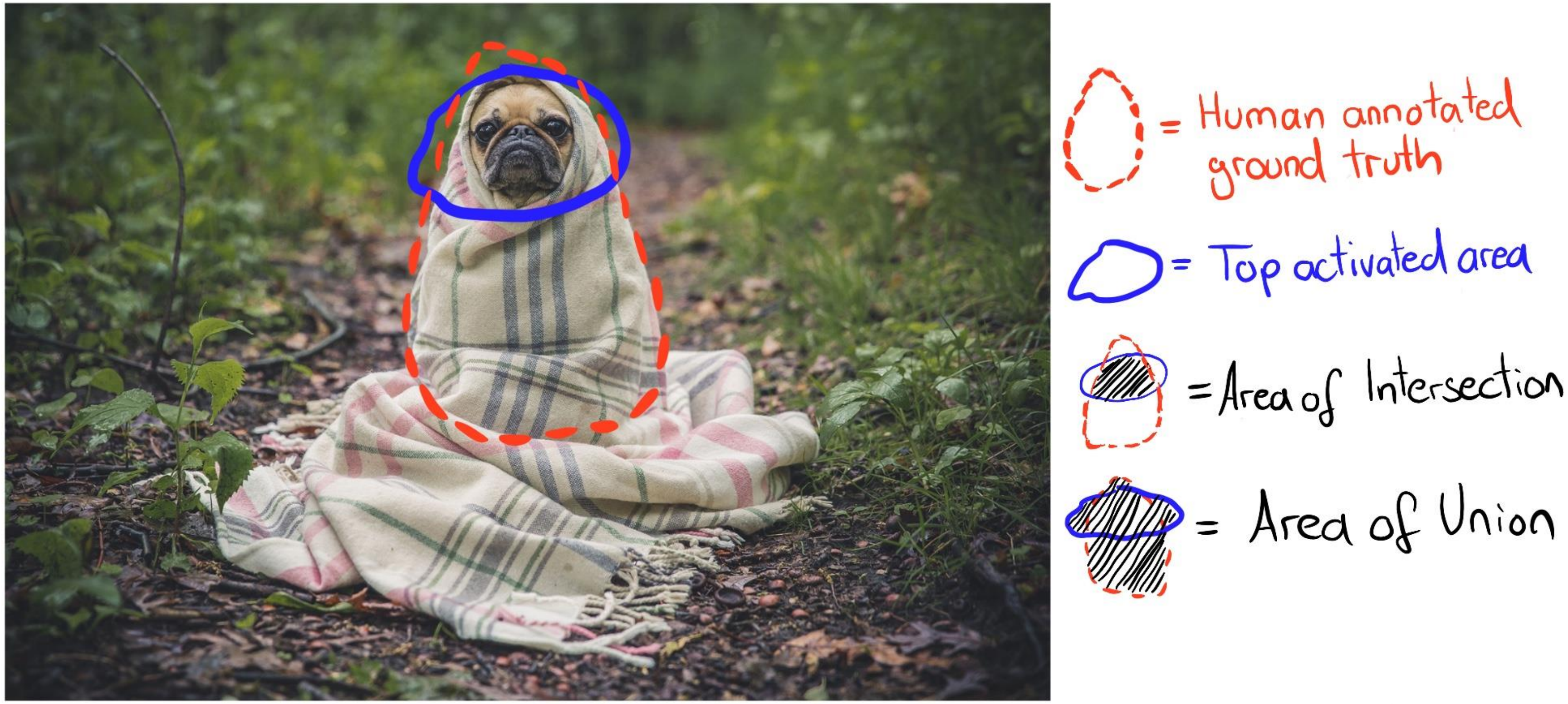


Network Dissection

- **Step 3: Activation-concept alignment**
  - Measure the **IoU** between the binary activation map and the labelled concept images
  - If activation map overlaps highly with a concept, the neuron is a detector for that concept

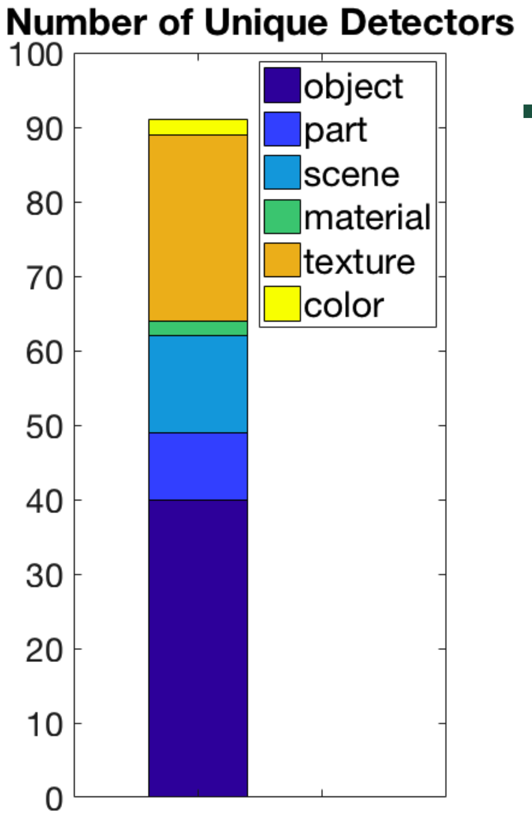
$$IoU_{k,c} = \frac{\sum |M_k(x) \bigcap L_c(x)|}{\sum |M_k(x) \bigcup L_c(x)|}$$

If $IoU_{k,c} > 0.04$. , unit k is a detector of concept c
by Bau & Zhou et al (2017)



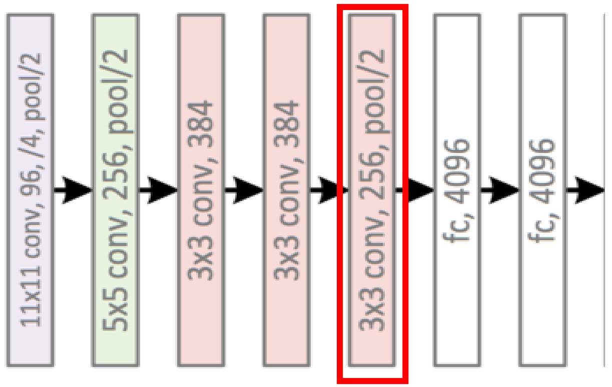
Network Dissection

- **Results** **the number of unique concept detectors (disentangled features)** as a measure of interpretability



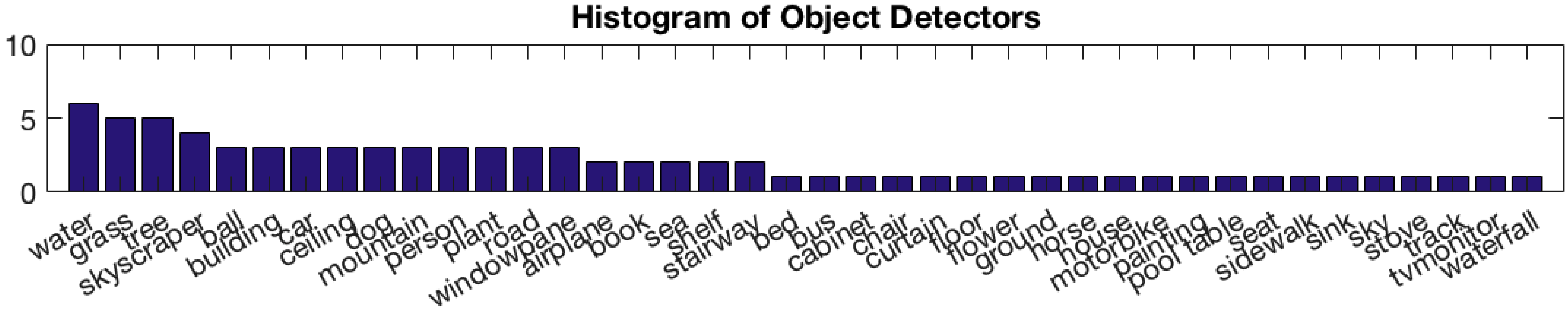
AlexNet trained on places THE SCENE RECOGNITION DATABASE

conv5, 256 units



Living room
Kitchen
Coast
…
365 categories

Histogram of object detectors: Detector:81/256, Unique Detector:40 (Units with IoU>0.04)

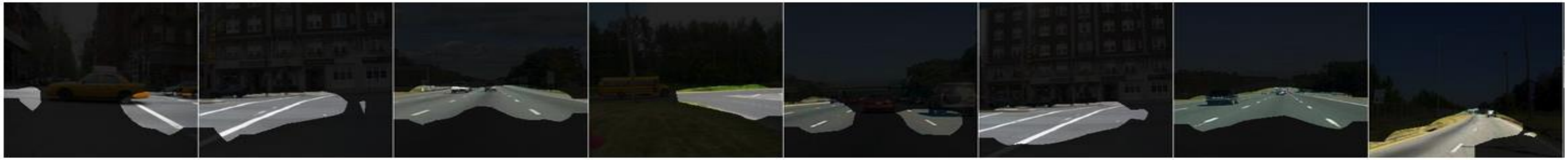


Network Dissection

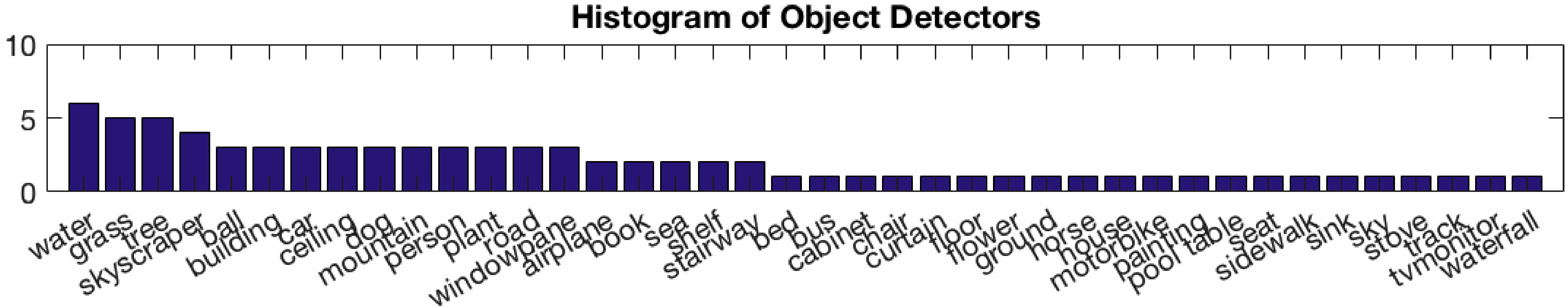
- **Results**

conv5 unit 79 car (object) IoU=0.13

conv5 unit 107 road (object) IoU=0.15

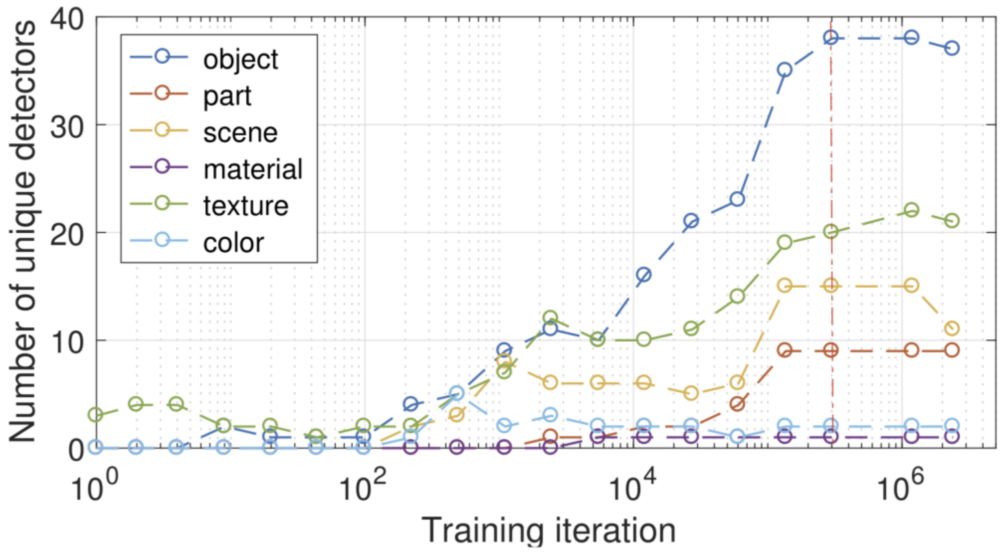


Histogram of object detectors: Detector:81/256, Unique Detector:40 (Units with IoU>0.04)



Network Dissection

- **Results**
  - The networks detect lower-level concepts (colors, textures) at lower layers and higher-level concepts (parts, objects) at higher layers.
  - Batch normalization reduces the number of unique concept detectors.
  - Many units detect the same concept.
    - For example, there are 95 (!) dog channels in VGG trained on ImageNet
  - The number of unique concept detectors increases with the number of training iterations.
  - In transfer learning, the concept of a channel can change. For example, a dog detector became a waterfall detector.



Learned features

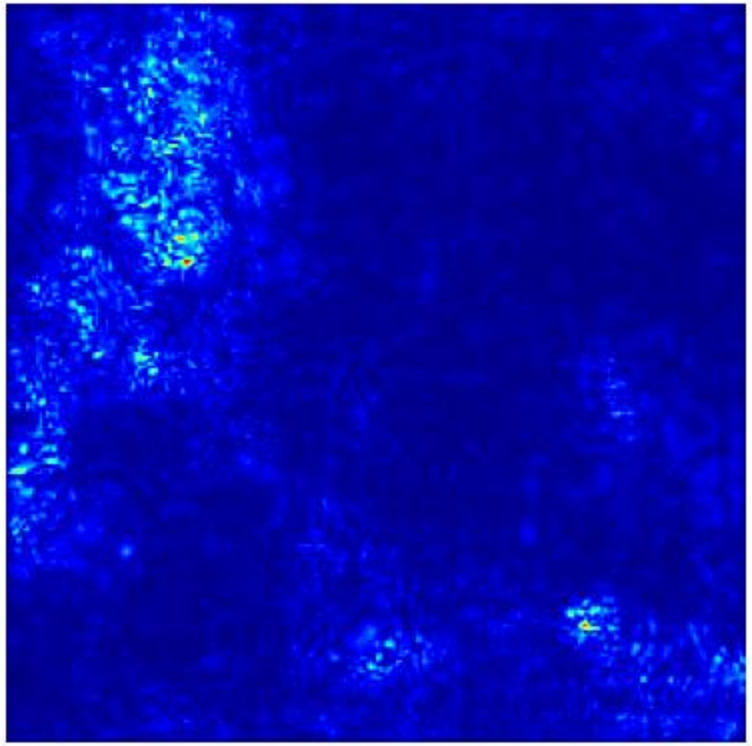
- Advantages
  - Feature visualizations give **unique insight into the working of neural networks**, especially for image recognition.
  - allows us to **automatically link units to concepts**
  - feature visualizations make great desktop wallpapers and T-shirt prints.....

- Disadvantage
  - **Many feature visualization images are not interpretable** at all, but contain some abstract features for which we have no words or mental concept.
  - There are **too many units to look at**, even when “only” visualizing the channel activations.
  - Even if we look at hundreds or thousands of feature visualizations, we cannot understand the neural network.
    - The channels are not completely disentangled and we cannot interpret them in isolation.
  - For Network Dissection, **you need datasets that are labeled on the pixel level** with the concepts.

Pixel Attribution

Pixel Attribution (Saliency map)

- Pixel attribution methods **highlight the pixels that were relevant for a certain image classification** by a neural network.



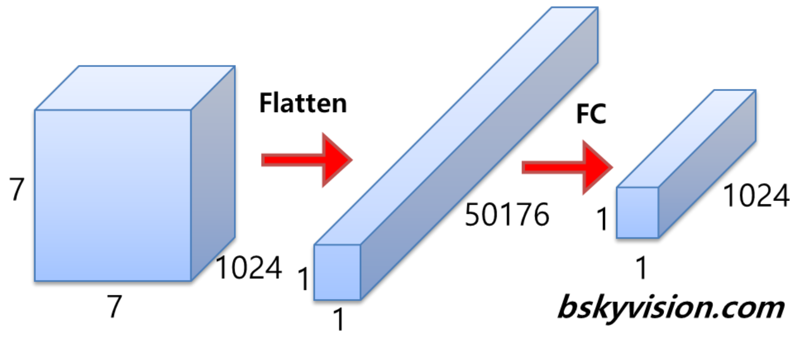
- Pixel attribution is a special case of feature attribution, but for images.
- SHAP, LIME are examples of general feature attribution methods

Grad-CAM

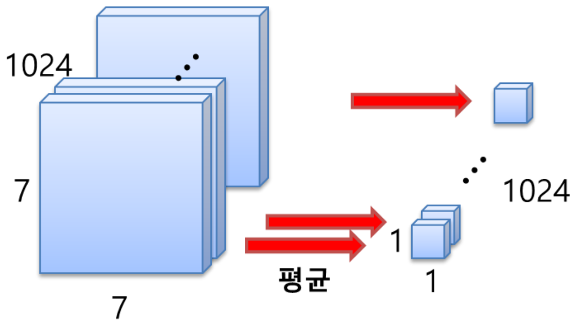
- **Class Activation Map (CAM)**
  - They proposed a network where the fully connected layers at the very end of the model has been replaced by a layer named Global Average Pooling (GAP)

Figure 1. A simple modification of the global average pooling layer combined with our class activation mapping (CAM) technique allows the classification-trained CNN to both classify the image and localize class-specific image regions in a single forward-pass e.g., the toothbrush for *brushing teeth* and the chainsaw for *cutting trees*.

FC 방식



Global average pooling



Grad-CAM

CAM

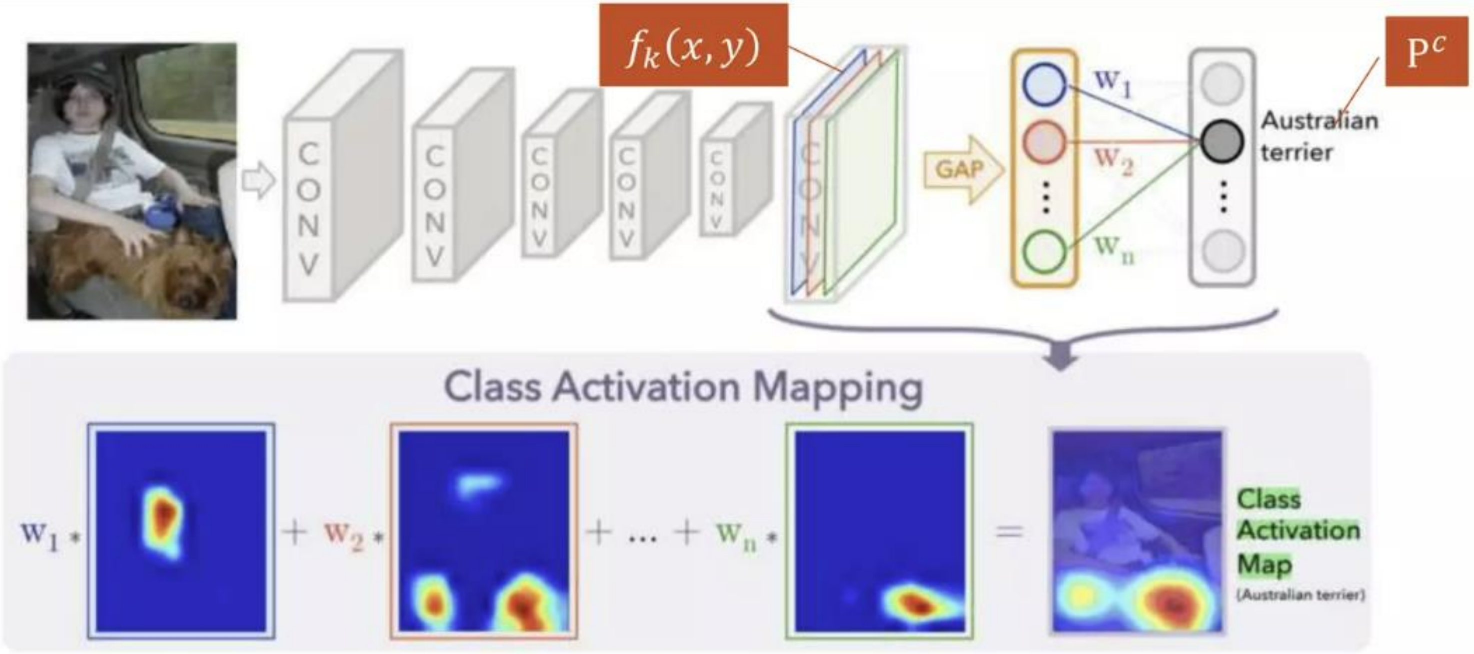


Figure 2. Class Activation Mapping: the predicted class score is mapped back to the previous convolutional layer to generate the class activation maps (CAMs). The CAM highlights the class-specific discriminative regions.

- CNN Architecture
  1. For each feature map ($f_k(x,y), k = 1, \dots, n$) at the last convolutional layer, GAP outputs the spatial average of each feature map
  $$F_k = \sum_{x,y} f_k(x,y)$$
  2. For a given class c, the input for output layer: $S_c = \sum_k w_k^c F_k$ (w_k^c: importance of F_k for class c)
  3. Output score for class c: $P_c = \frac{\exp(S_c)}{\sum_c \exp(S_c)}$ (e.g., softmax)

Grad-CAM

CAM

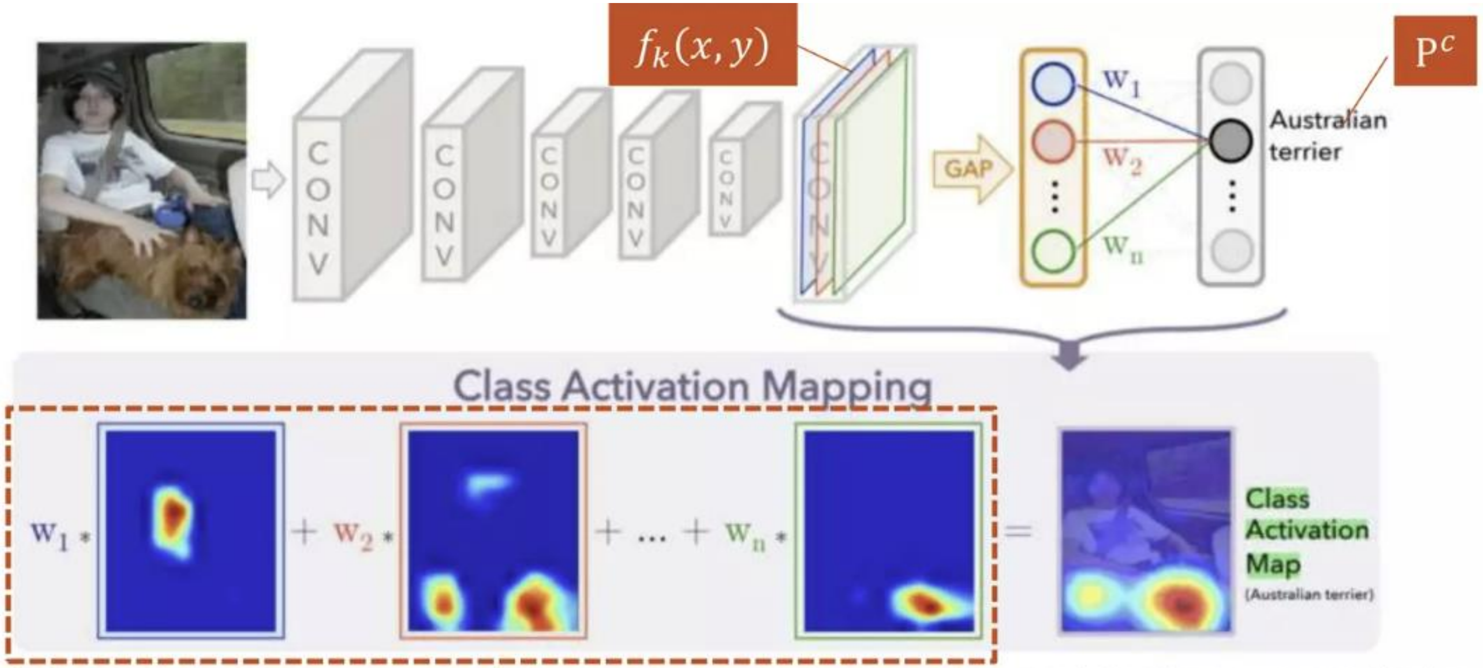


Figure 2. Class Activation Mapping: the predicted class score is mapped back to the previous convolutional layer to generate the class activation maps (CAMs). The CAM highlights the class-specific discriminative regions.

- CAM Procedure
  - Weights $(w_1^c, w_2^c, \ldots, w_n^c)$ of output layer indicate the importance of the image regions (F_k) to a specific class (c)
  - → Compute CAM:

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

Note: if the shape (H, W) of CAM (M_c) is different from that of input images, up-sampling is needed to equalize the shapes

Grad-CAM

- **Class Activation Map (CAM)**
  - Feature map 내의 상대적인 위치는 보존이 됨
  - Feature map 내 pixel 값은 원본 이미지 상의 특정 영역에 의해 activate/deactivate
  - GAP을 통해 feature map이 요약되며, 이어지는 weight가 각 feature map의 중요도

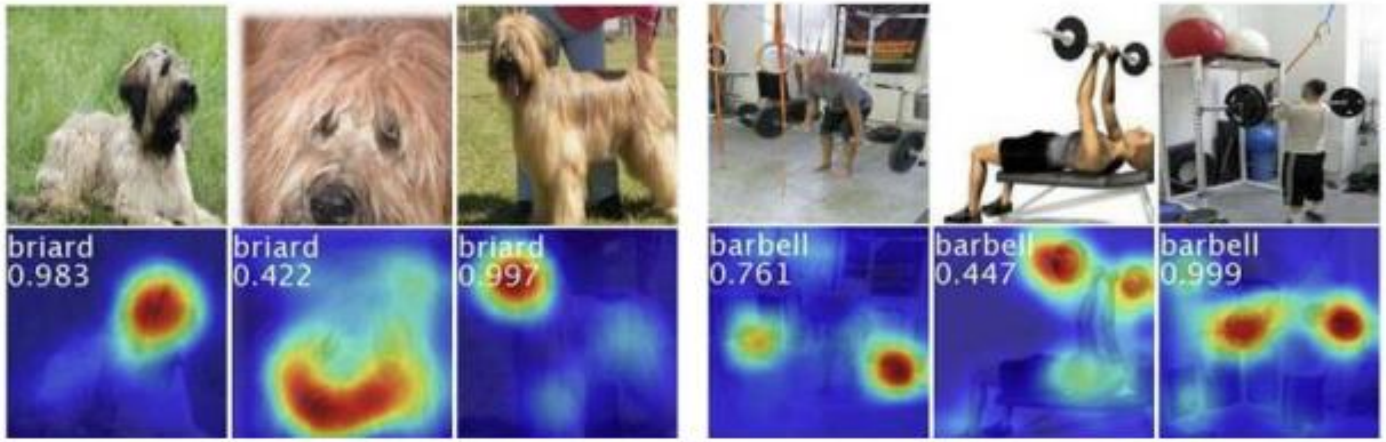
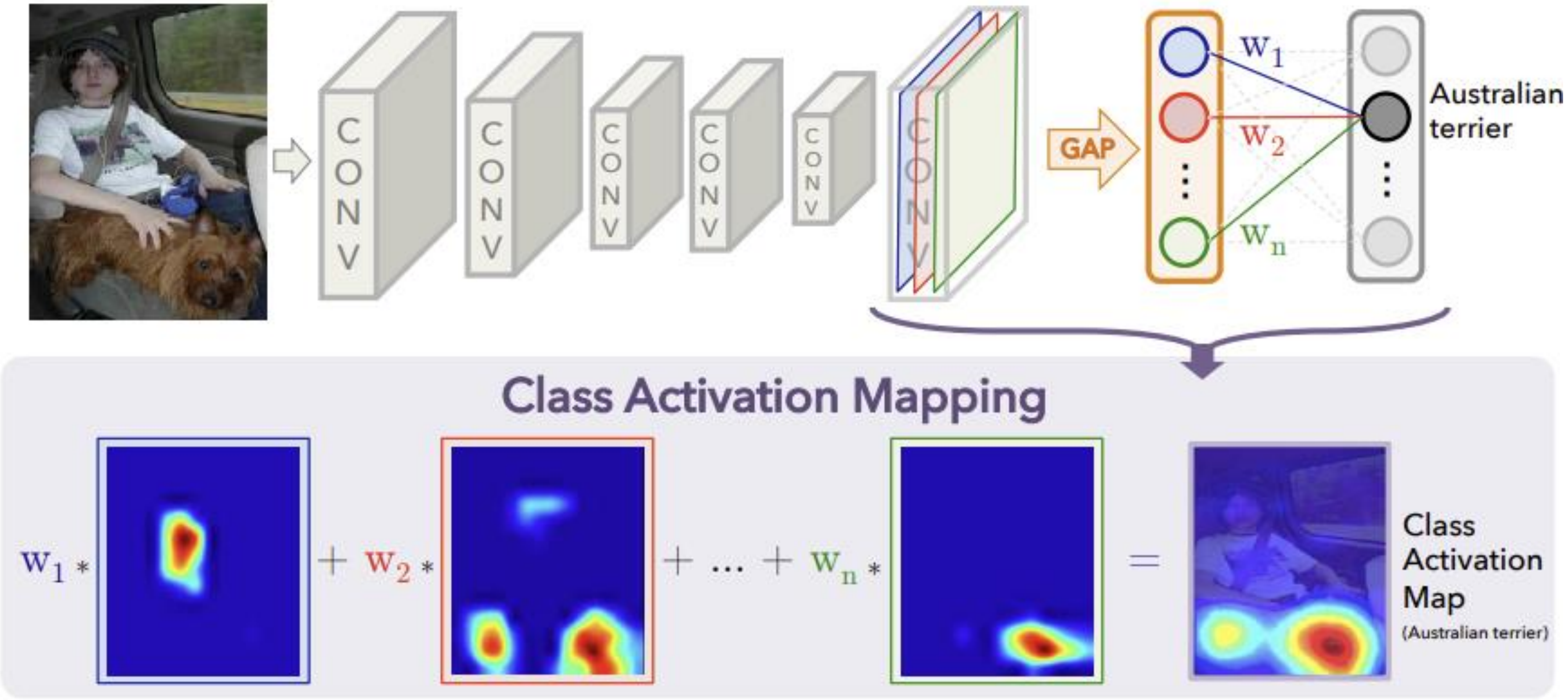


Figure 3. The CAMs of two classes from ILSVRC [21]. The maps highlight the discriminative image regions used for image classification, the head of the animal for *briard* and the plates in *barbell*.

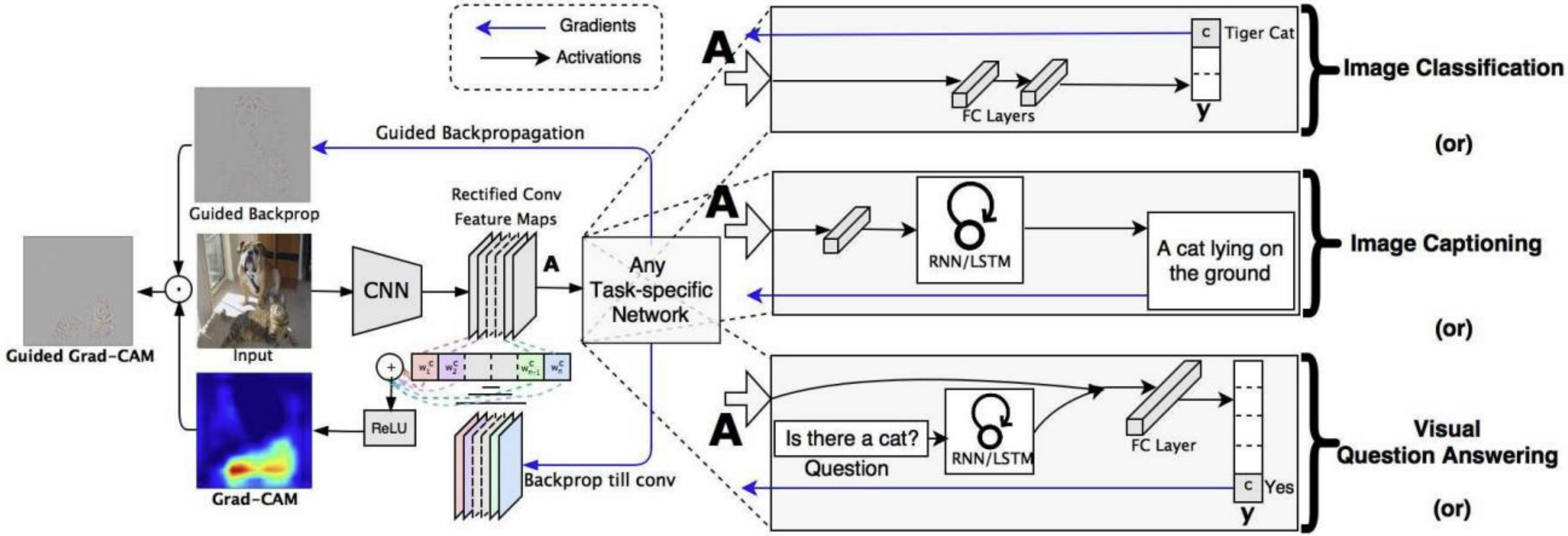
Grad-CAM

- **Class Activation Map (CAM)**
  - GAP을 도입한 후에 재학습 필요



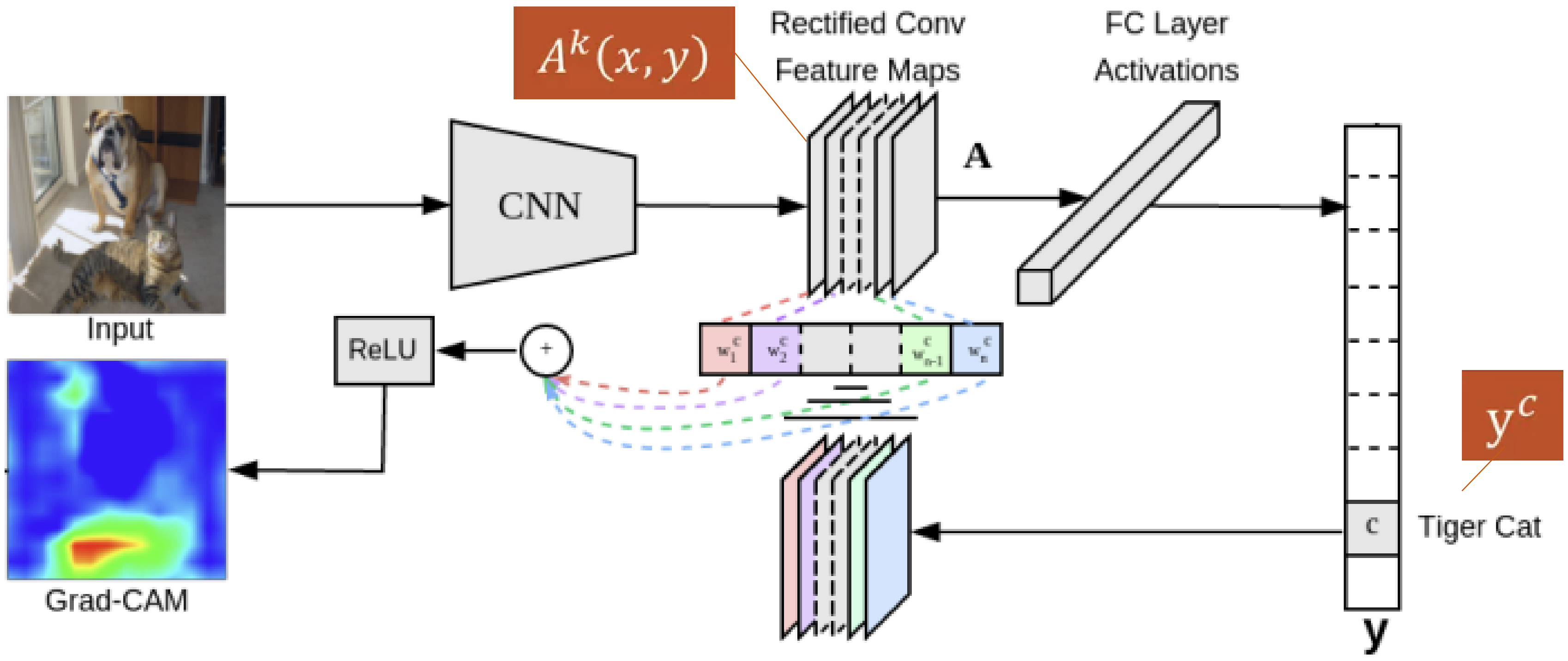
Grad-CAM

- **Grad-CAM (Gradient-weighted Class Activation Mapping)**
  - Generalized version of CAM for <u>any CNN-based architectures</u>
    - **No need for GAP layer, or re-training**
  - Need a way to define weight w without GAP layer
    - <u>Using gradient</u>



https://www.slideshare.net/HsingchuanHsieh/introduction-to-gradcam-complete-version

Grad-CAM

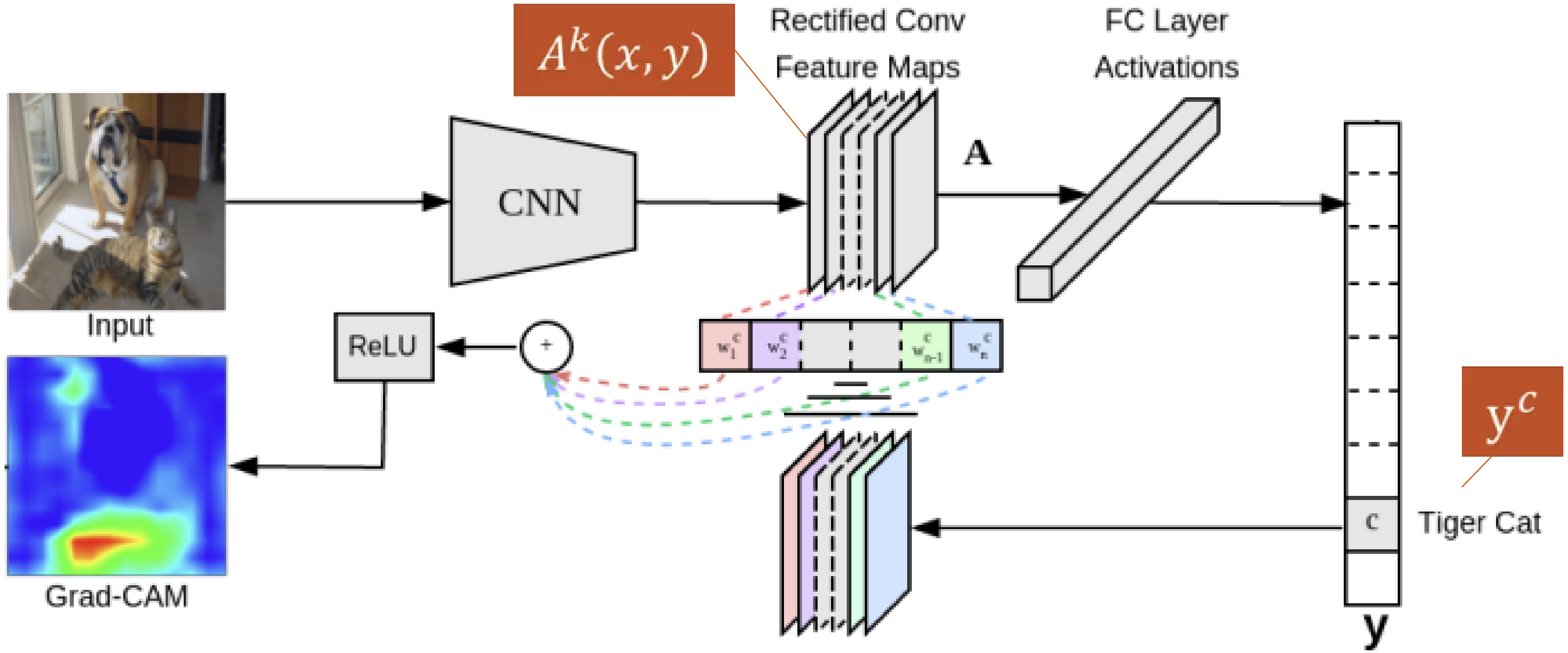


- For a given class c, compute the gradient of its score– y^c (before the softmax), w.r.t. each feature map activations $A^k \in \mathbb{R}^{u \times v}, k = 1, \ldots, n$ of a convolutional layer, i.e. $\frac{\partial y^c}{\partial A^k} \in \mathbb{R}^{u \times v}$ ← Influence of $A^k(x, y)$ to y^c
- Define the importance weights of feature map k via GAP:

$$\alpha_k^c = \frac{1}{Z} \sum_{i \in x} \sum_{j \in y} \frac{\partial y^c}{\partial A_{ij}^k}$$

gradients are globally pooled Gradients via backprop

Grad-CAM



- → Compute Grad-CAM:

$$L^c_{Grad-CAM}(x,y) = ReLU\left(\sum_k \alpha^c_k \; A^k(x,y)\right) \in \mathbb{R}^{u\times v}$$

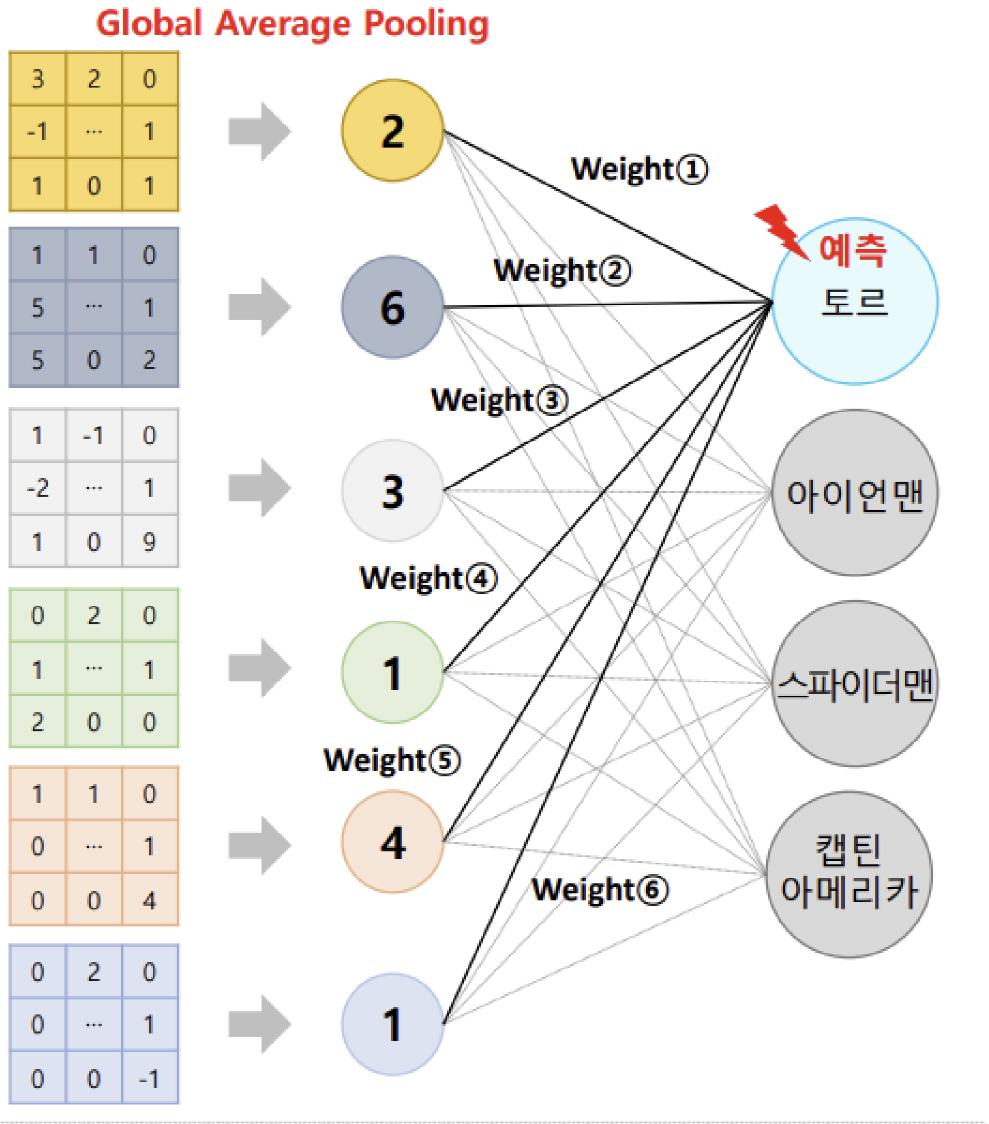
  - ReLU is applied because we are only interested in the features (neurons) that have a positive influence on the class of interest
  - i.e. pixels whose intensity should be increased in order to increase y^c

Note: if the shape (u, v) of $L^c_{Grad-CAM}$ is different from that of input images, up-sampling is needed to equalize the shapes

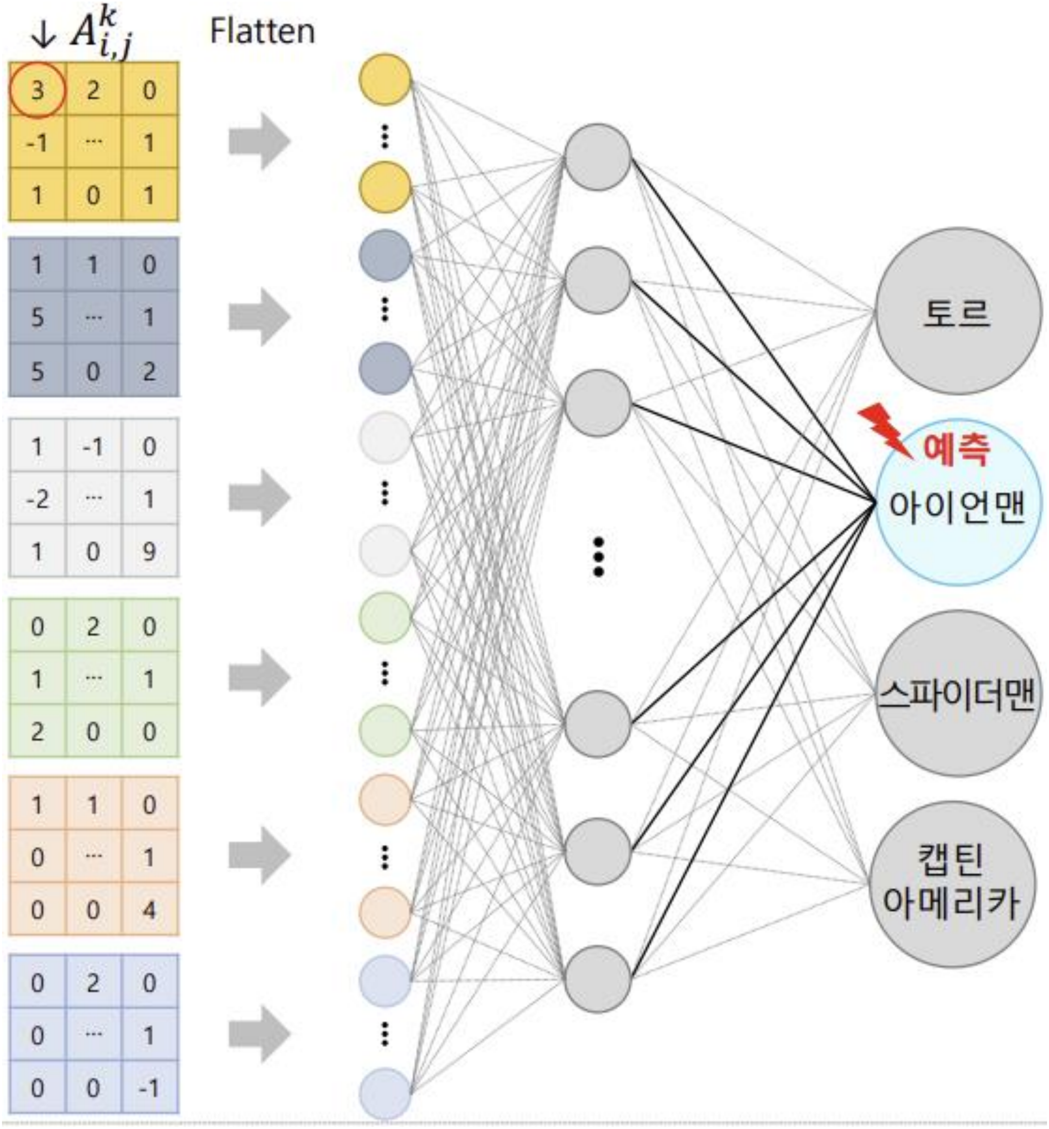
Grad-CAM

http://dmqm.korea.ac.kr/activity/seminar/274

- **CAM**: W를 학습을 통해 계산



Grad-CAM: Gradient를 통해 W와 유사한 역할 수행



$$\frac{1}{Z}\sum_i\sum_j \frac{\partial y^c}{\partial A^1_{i,j}}$$

$$\frac{1}{Z}\sum_i\sum_j \frac{\partial y^c}{\partial A^2_{i,j}}$$

$$\frac{1}{Z}\sum_i\sum_j \frac{\partial y^c}{\partial A^3_{i,j}}$$

$$\frac{1}{Z}\sum_i\sum_j \frac{\partial y^c}{\partial A^4_{i,j}}$$

$$\frac{1}{Z}\sum_i\sum_j \frac{\partial y^c}{\partial A^5_{i,j}}$$

$$\frac{1}{Z}\sum_i\sum_j \frac{\partial y^c}{\partial A^6_{i,j}}$$

Grad-CAM

- **Grad-CAM as a generalization of CAM**
  - <u>GAP가 있는 구조에서는</u>, Grad-CAM이 CAM과 동일

$$Y^c = \sum_k \underbrace{w_k^c}_{\text{class feature weights}} \overbrace{\frac{1}{Z}\sum_i \sum_j}^{\text{global average pooling}} \underbrace{A_{ij}^k}_{\text{feature map}}$$

$$\boxed{F^k = \frac{1}{Z}\sum_i \sum_j A_{ij}^k}$$

$$= \sum_k w_k^c \cdot F^k$$

Feature map k가 class c 예측에 얼마나 영향을 미치나?

$$\frac{\partial Y^c}{\partial F^k} = \frac{\frac{\partial Y^c}{\partial A_{ij}^k}}{\frac{\partial F^k}{\partial A_{ij}^k}}$$

$$w_k^c = Z \cdot \frac{\partial Y^c}{\partial A_{ij}^k}$$

$$\sum_i \sum_j w_k^c = \sum_i \sum_j Z \cdot \frac{Y^c}{A_{ij}^k}$$

$$Z w_k^c = Z \sum_i \sum_j \frac{Y^c}{A_{ij}^k}$$

$$w_k^c = \sum_i \sum_j \frac{Y^c}{A_{ij}^k}$$

➡ Identical to α_k^c of Grad-CAM

Grad-CAM

- **Grad-CAM (Gradient-weighted Class Activation Mapping)**
  - Can be applied on any layer other than the last layer
  - Can help identify the biases

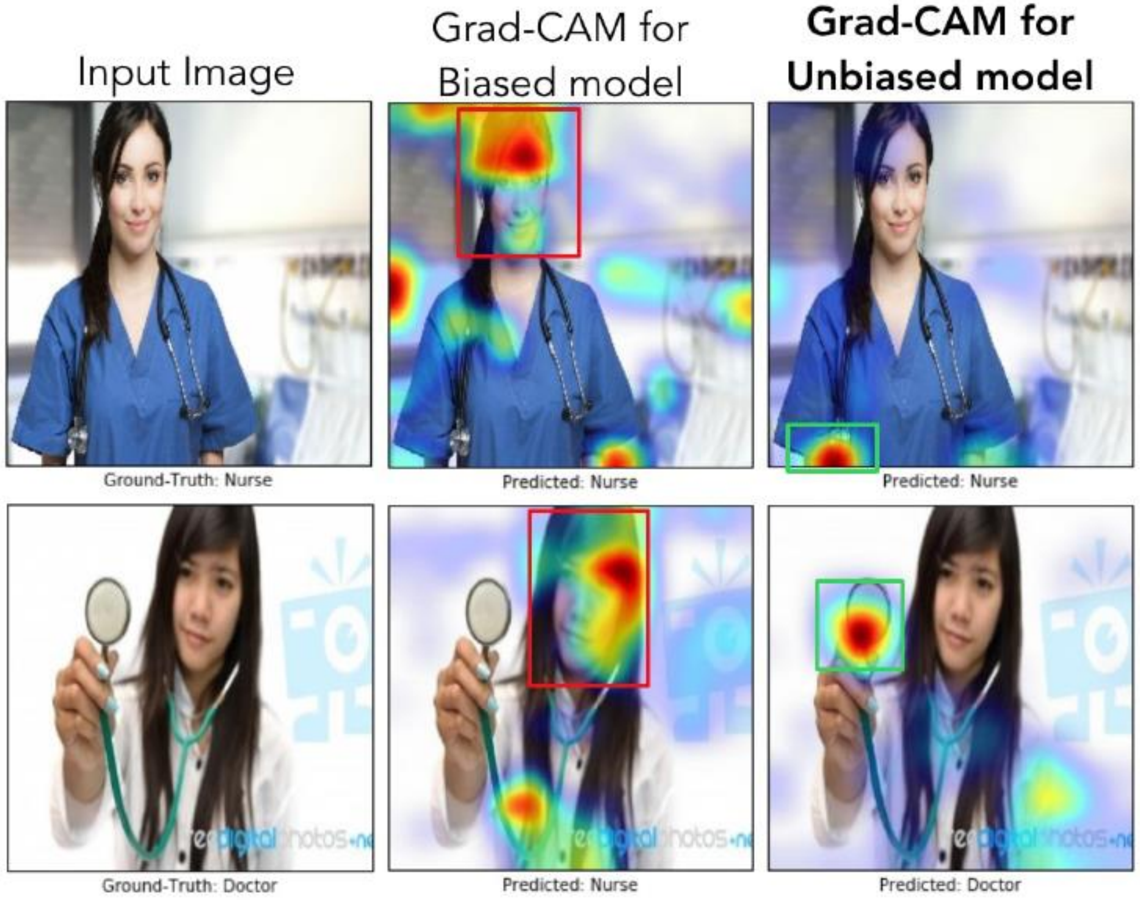
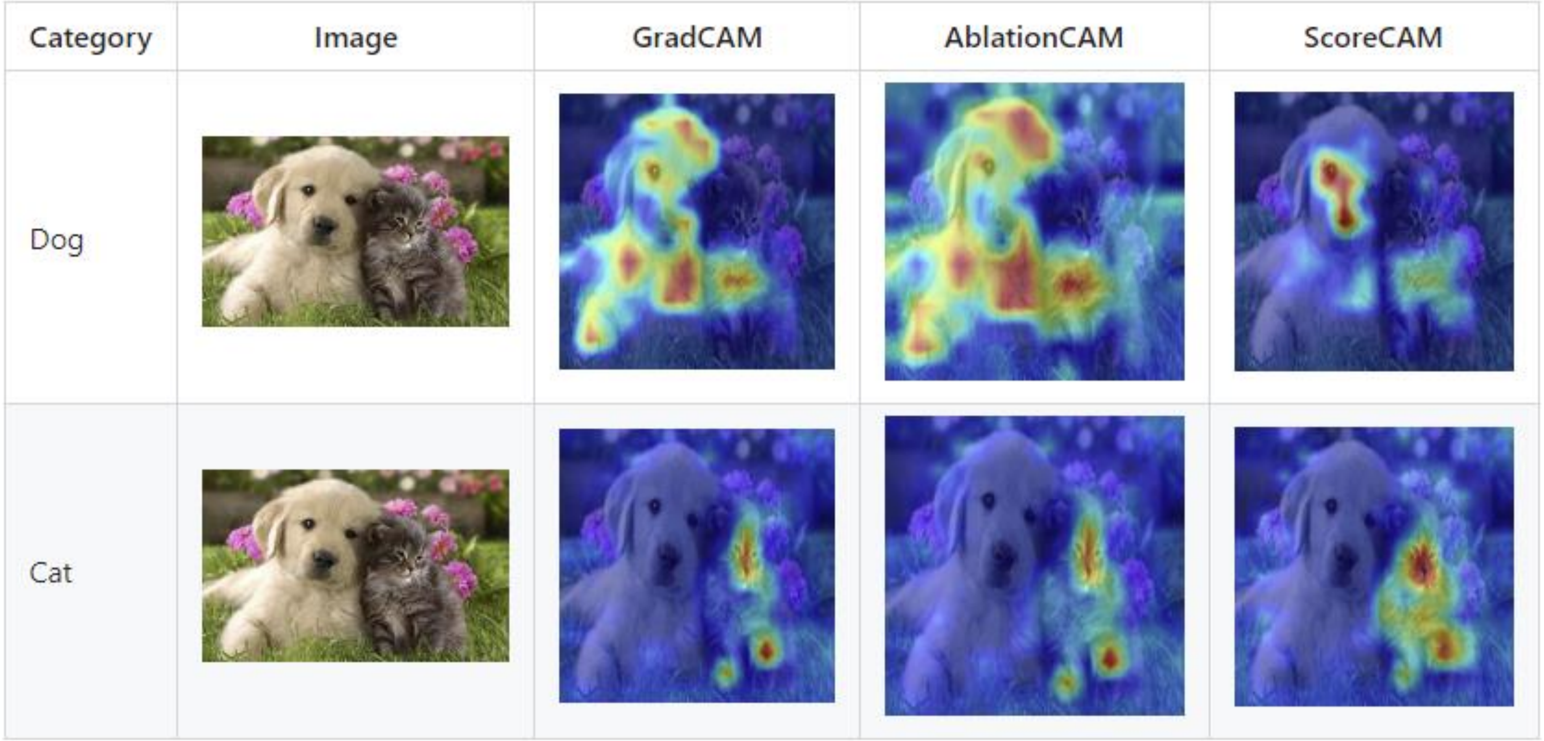


Fig. 8: In the first row, we can see that even though both models made the right decision, the biased model (model1) was looking at the face of the person to decide if the person was a nurse, whereas the unbiased model was looking at the short sleeves to make the decision. For the example image in the second row, the biased model made the wrong prediction (misclassifying a doctor as a nurse) by looking at the face and the hairstyle, whereas the unbiased model made the right prediction looking at the white coat, and the stethoscope.

Grad-CAM

- A lot of variants exist https://github.com/jacobgil/pytorch-grad-cam

Method	What it does
GradCAM	Weight the 2D activations by the average gradient
HiResCAM	Like GradCAM but element-wise multiply the activations with the gradients; provably guaranteed faithfulness for certain models
GradCAMElementWise	Like GradCAM but element-wise multiply the activations with the gradients then apply a ReLU operation before summing
GradCAM++	Like GradCAM but uses second order gradients
XGradCAM	Like GradCAM but scale the gradients b
AblationCAM	Zero out activations and measure how fast batched implementation)
ScoreCAM	Perbutate the image by the scaled activ
EigenCAM	Takes the first principle component of t but seems to give great results)
EigenGradCAM	Like EigenCAM but with class discrimin Activations*Grad. Looks like GradCAM,
LayerCAM	Spatially weight the activations by posi lower layers
FullGrad	Computes the gradients of the biases fi them
Deep Feature Factorizations	Non Negative Matrix Factorization on t



Pixel Attribution

- Advantages
  - The explanations are **visual** and we are quick to recognize images
  - **faster to compute than model-agnostic methods**

- **Disadvantages**
  - As with most interpretation methods, it is **difficult to know whether an explanation is correct**, and a huge part of the evaluation is only qualitative
  - Pixel attribution methods can be very **fragile.**
    - small (adversarial) perturbations to an image, which still lead to the same prediction, can lead to very different pixels being highlighted as explanations.