PADERBORN UNIVERSITY

Neural Network Explanations

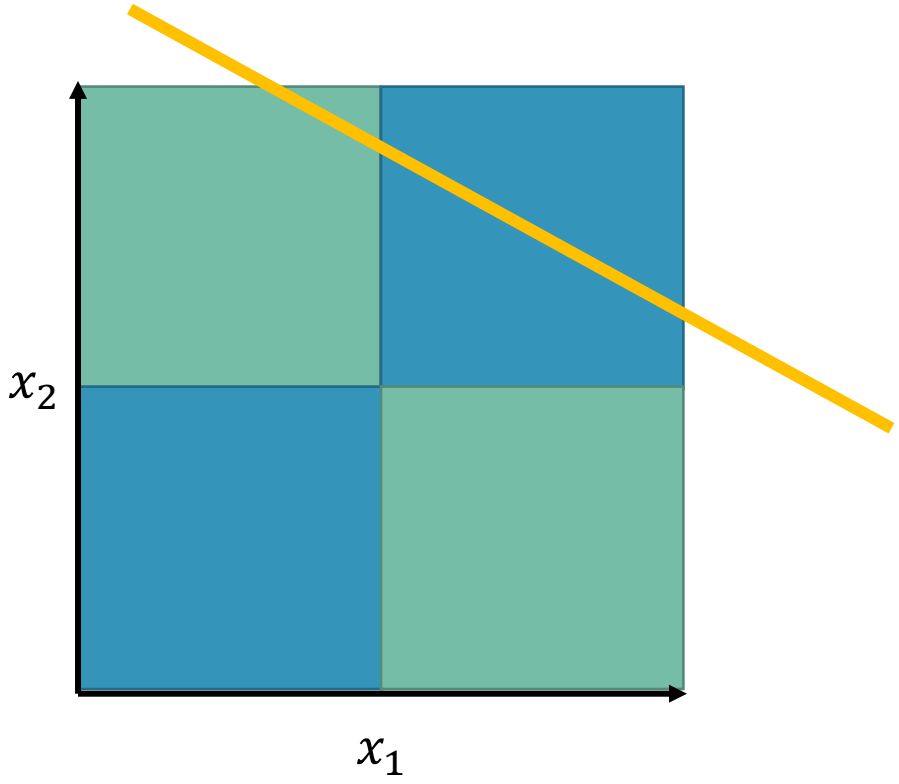
Explainable Artificial Intelligence

Dr. Stefan Heindorf

Background

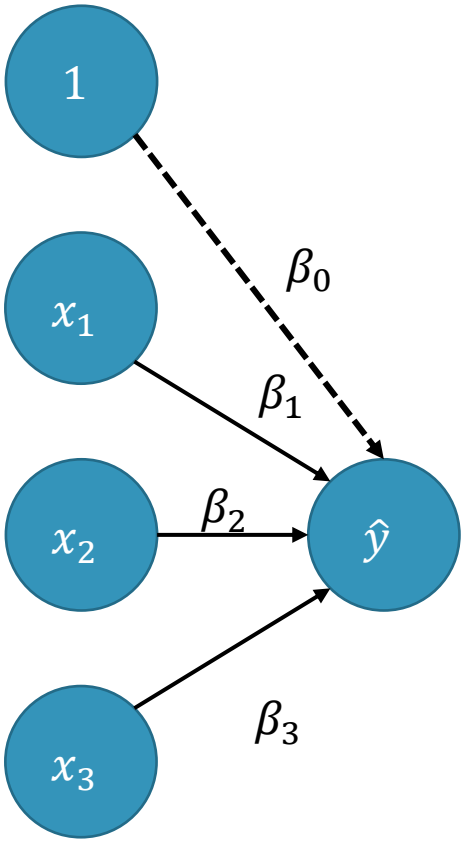
Motivation: Neural networks

Linear models cannot model XOR between two features



Single node

Graphical notation



Mathematical notation

$$\hat{y} = \sigma(\beta_0 + x_1\beta_1 + x_2\beta_2 + x_3\beta_3) = \sigma\left(\mathbf{x}^T\beta\right)$$

where σ is non-linear activation function

Typical activation functions:

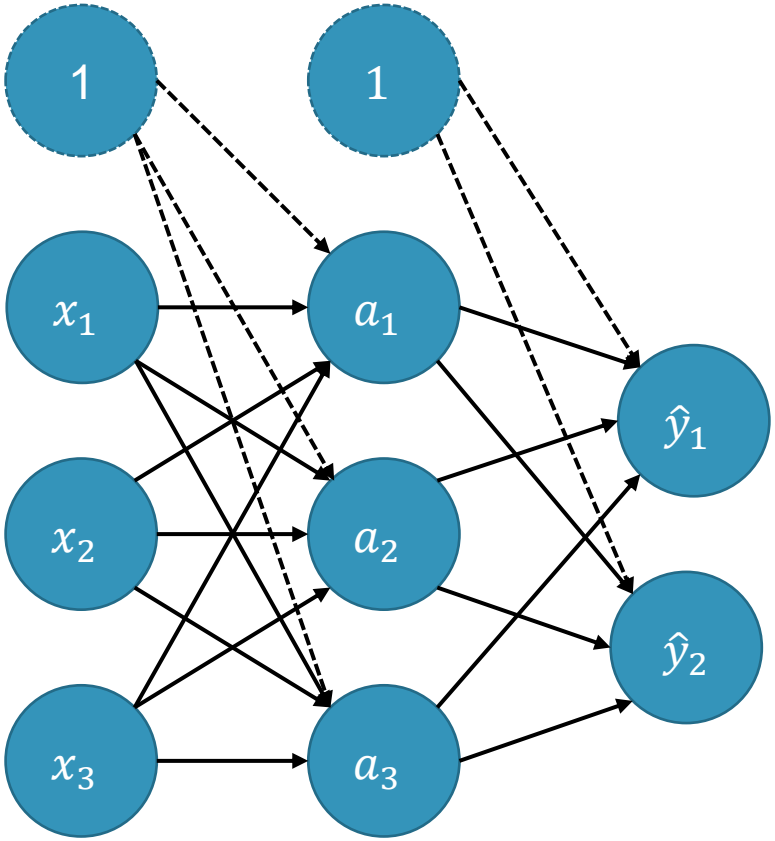
$$\sigma(\eta) = \text{logistic}(\eta) = \frac{1}{1+exp(\eta)}$$

$$\sigma(\eta) = \tanh(\eta)$$

$$\sigma(\eta) = \text{relu}(\eta) = \begin{cases} 0 \text{ for } \eta \leq 0 \\ \eta \text{ for } \eta > 0 \end{cases}$$

Neural network

Graphical notation



Mathematical notation

$$\hat{\mathbf{y}} = \sigma(\sigma(\mathbf{x}^{\mathbf{T}}\mathbf{W}_1)\mathbf{W}_2)$$

where $\mathbf{x}^{\mathbf{T}} = (1, x_1, x_2, x_3)$

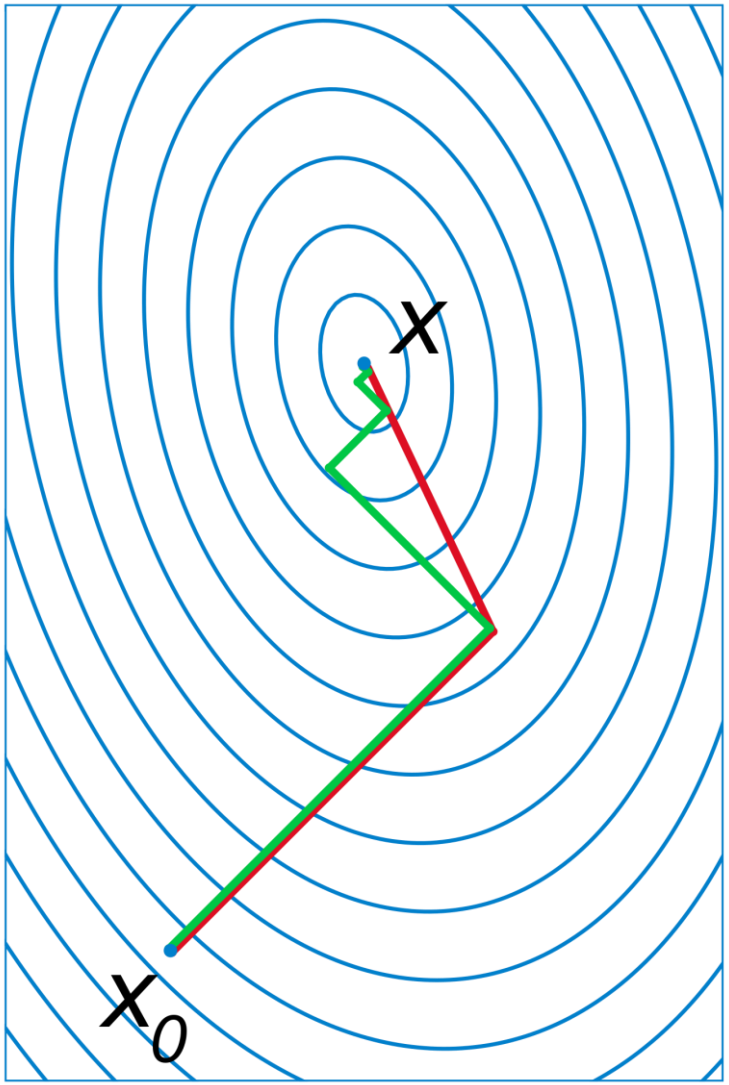
$\mathbf{W}_1 \in \mathbb{R}^{4\times3}$ and $\mathbf{W}_2 \in \mathbb{R}^{4\times2}$ represent weigh matrices

$\hat{\mathbf{y}} = (\hat{y}_1, \hat{y}_2)$ represents multiple outputs

Training via backpropagation and gradient descent

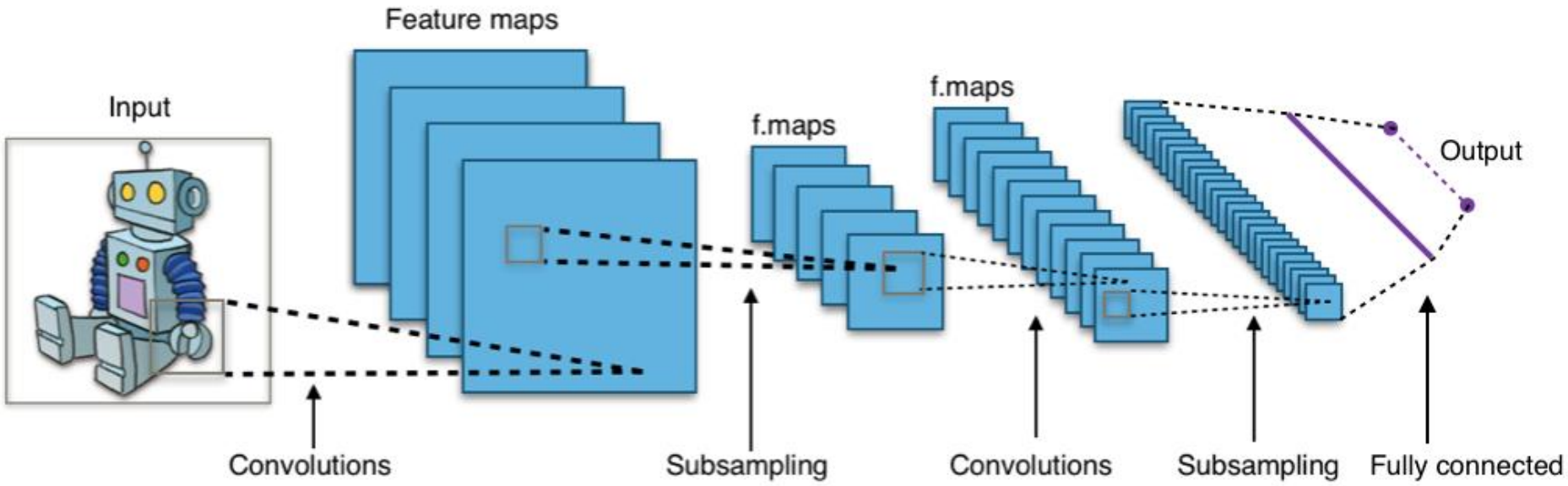
Gradient descent

- Initialize weight matrices randomly
- Make a prediction with the neural network
- Compute loss function
- Compute gradients via backpropagation (backwards one layer at a time)
  - Example: $\hat{\mathbf{y}} = \sigma(\sigma(\mathbf{x}^{\mathbf{T}}\mathbf{W}_1)\mathbf{W}_2)$
  - First compute gradient of W_2
  - Then compute gradient of W_1
- Update weights of neural network (according to their gradients)
- Repeat the whole process



Further details in lectures ML1 / ML2

Convolutional neural network



Note on terminology

- Feature maps = channels
- Subsampling = pooling

Convolutions and subsampling

Convolutions

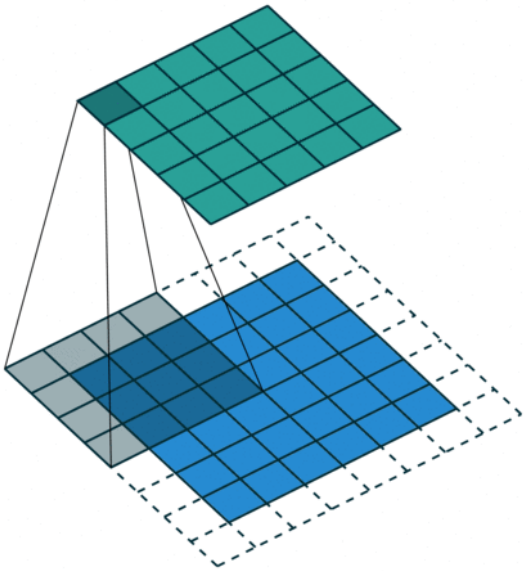
0	1	2
3	4	5
6	7	8

*

0	1
2	3

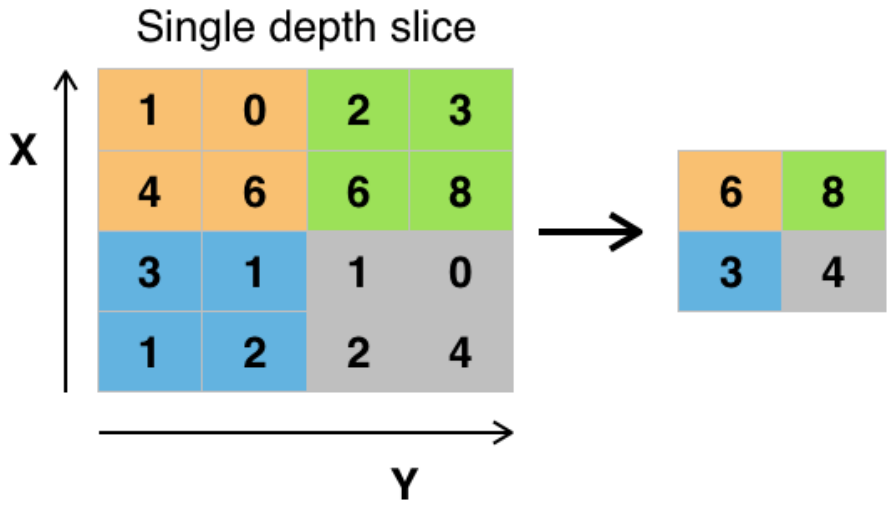
=

19	25
37	43



Vincent Dumoulin, Francesco Visin (https://commons.wikimedia.org/wiki/File:Convolution_arithmetic_-_Arbitrary_padding_no_strides_transposed.gif), MIT License: https://github.com/vdumoulin/conv_arithmetic/blob/master/LICENSE

Subsampling (=Pooling)



Aphex34 (https://commons.wikimedia.org/wiki/File:Max_pooling.png), https://creativecommons.org/licenses/by-sa/4.0/legalcode

ImageNet datasets

Large datasets for visual object recognition software research

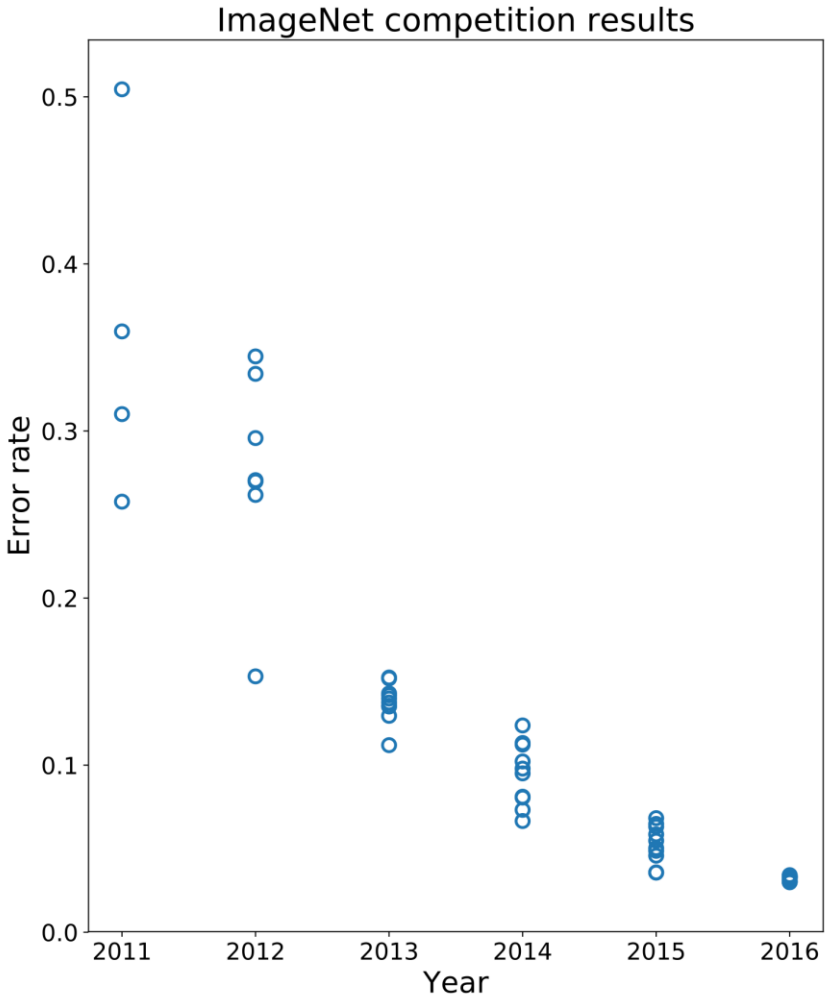
- Hand-annotated
  - Objects in the picture
  - Bounding boxes

Full ImageNet-21k

- 21,841 classes
- 14,197,122 images divided into

Subset ImageNet-1K

- 1,000 classes
- 1,281,167 training images,
- 50,000 validation images
- 100,000 test images.



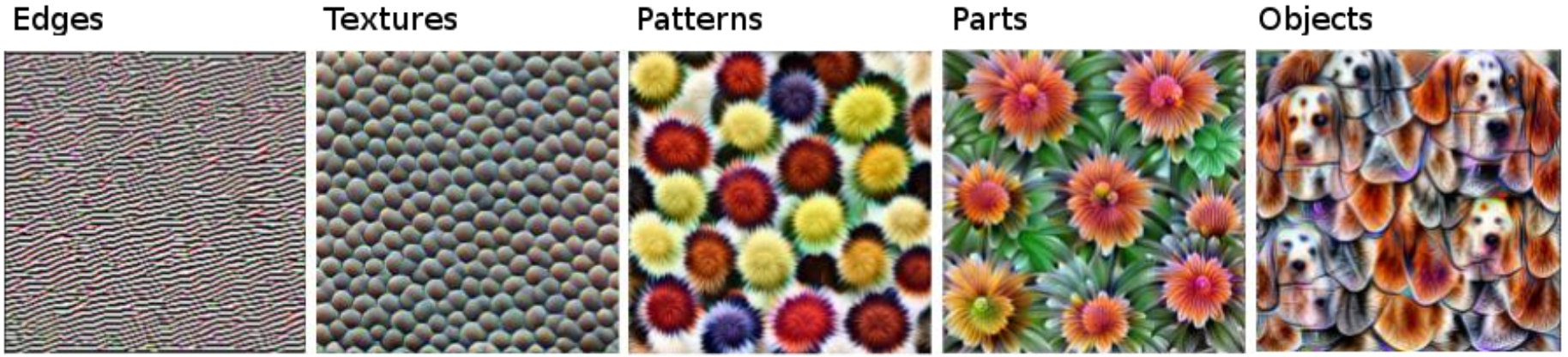
Feature Visualization

Feature visualization

Motivation

Deep neural networks learn high-level features in the hidden layers

- First layers: edges and simple textures
- Next layers: textures and patterns
- Later layers: parts of objects and object
- Last layers: classes to be predicted



Features learned by a convolutional neural network (Inception V1) trained on the ImageNet data. The features range from simple features in the lower convolutional layers (left) to more abstract features in the higher convolutional layers (right).

Figure by Christoph Molnar (https://christophm.github.io/interpretable-ml-book/images/cnn-features.png), https://creativecommons.org/licenses/by-nc-sa/4.0/. Originally from Olah, et al. (2017) https://distill.pub/2017/feature-visualization/appendix/ https://creativecommons.org/licenses/by/4.0/

Feature visualization

Feature visualization: for a unit of a neural network find the input that maximizes the activation of that unit

"Unit" can refer to

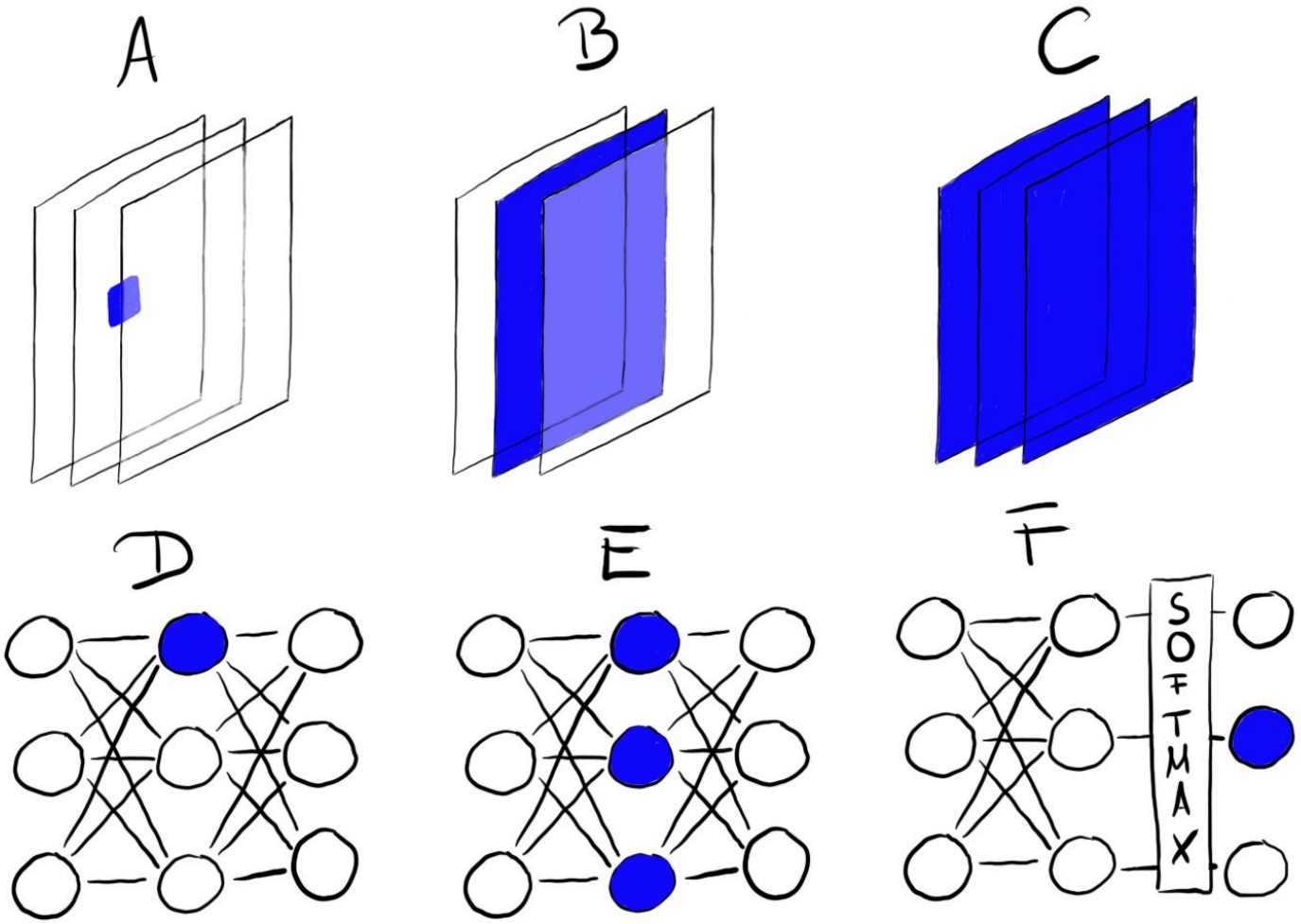
- Individual neurons
- Channels (also called feature maps)
- Entire layers
- Final class probability in classification
  (or the corresponding pre-softmax neuron, which is recommended)

Considerations

- Neural networks often contain millions of neurons
  → millions of visualizations
- Better choice in practice: visualize channels / layers

Feature visualization

Units of neural networks



Feature visualization can be done for different units. A) Convolution neuron, B) Convolution channel, C) Convolution layer, D) Neuron, E) Hidden layer, F) Class probability neuron

Feature visualization through optimization

Assumption: **network is trained and weights are fixed**

Goal: **find image that maximizes (or minimizes) the activation of a single neuron**

$$\mathbf{x}^* = \arg\max_{\mathbf{x}} h_{\boldsymbol{n,u,v,z}}(\mathbf{x})$$

Goal: **find image that maximizes (or minimizes) the mean activation of entire channel**

$$\mathbf{x}^* = \arg\max_{\mathbf{x}} \sum_{u,v} h_{\boldsymbol{n,u,v,z}}(\mathbf{x})$$

h	activation of a neuron
$\mathbf{x}$	input of the network (an image)
u, v	spatial position of the neuron *(e.g., x, y position)*
n	layer of the neuron (in network)
z	channel index of the neuron (in network)

Feature visualization through optimization

Example

Positive (left) and negative (right) activation of Inception V1 neuron 484 from layer mixed4d pre relu. While the neuron is maximally activated by wheels, something which seems to have eyes yields a negative activation.

Christoph Molnar (https://github.com/christophM/interpretable-ml-book/blob/master/manuscript/images/a484.png), https://creativecommons.org/licenses/by-nc-sa/4.0/

Feature visualization through optimization

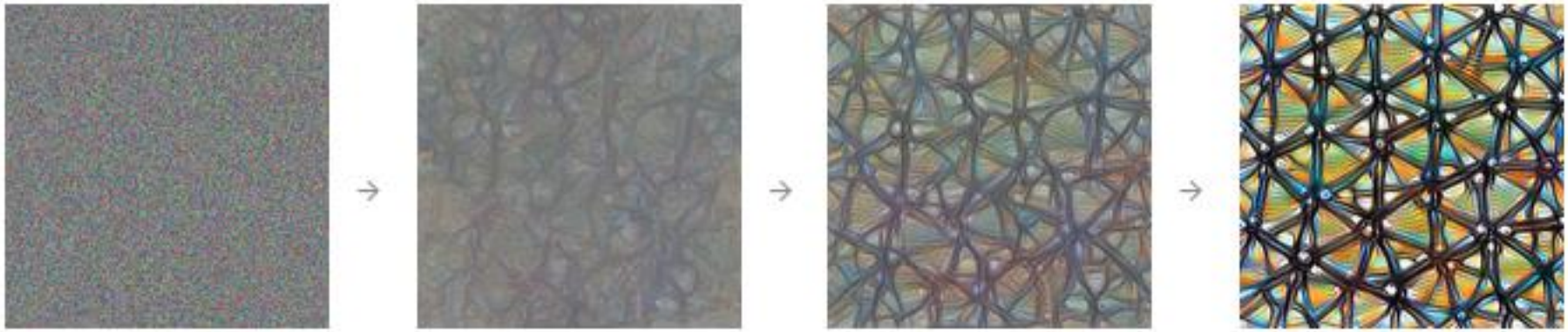
Solving the optimization problem

Select training images that maximize the activation

- Elements on the images can be correlated
  → we cannot see what the neural network is really looking for

Generate new images that maximize the activation

- Start from random noise
- In each step, apply small changes to the image
- Apply regularization to reduce noise in the feature visualization
  (jittering, rotation, scaling, …)



Iterative optimization from random image to maximizing activation.

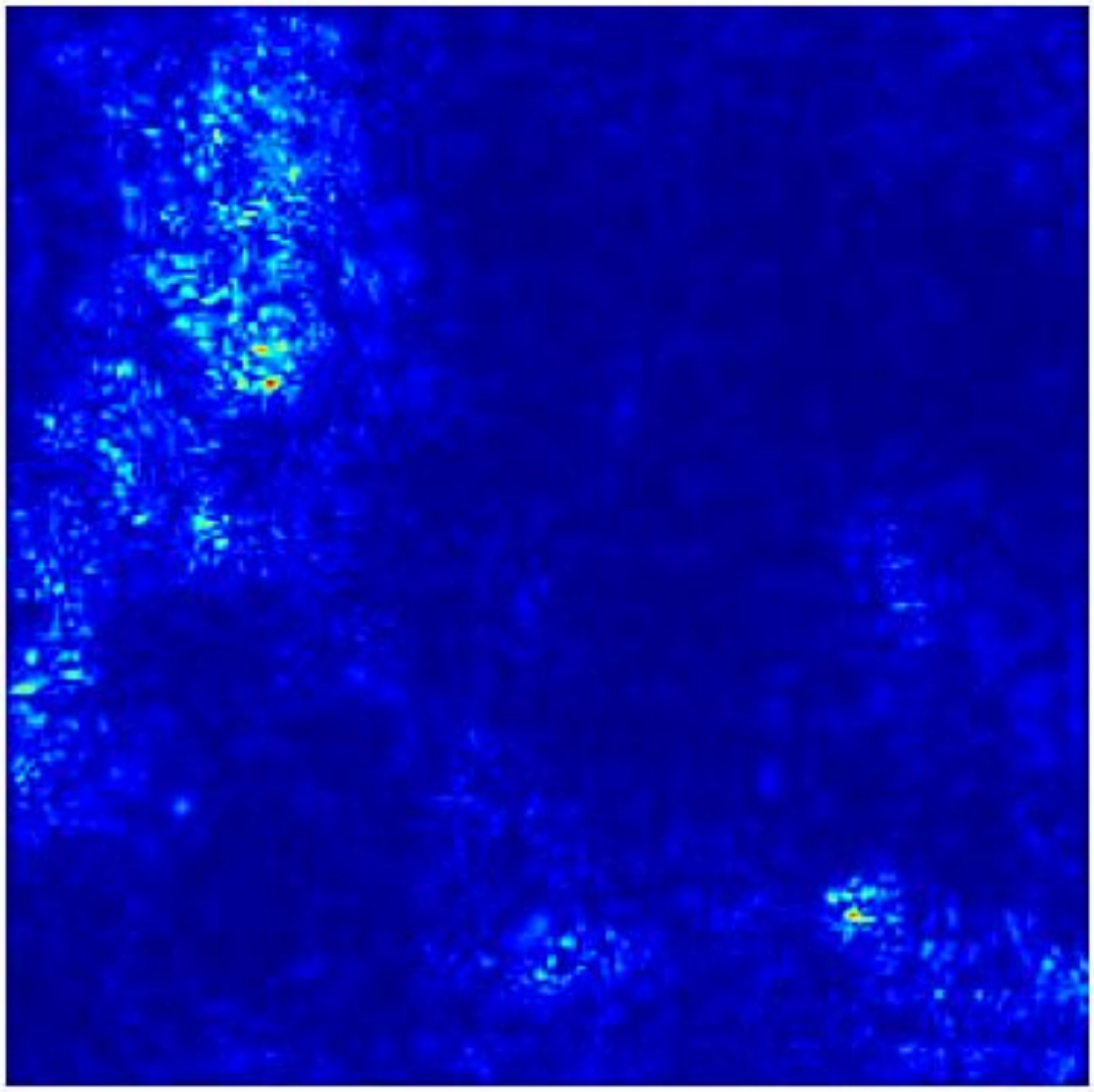
Christoph Molnar (https://christophm.github.io/interpretable-ml-book/images/activation-optim.png), https://creativecommons.org/licenses/by-nc-sa/4.0/
riginally from Olah, et al. (2017) https://distill.pub/2017/feature-visualization/appendix/ https://creativecommons.org/licenses/by/4.0/

Pixel Attribution

Saliency Maps

Pixel attribution (saliency maps)

Example: highlight pixels that are relevant for an image classification



Pixel attribution (saliency maps)

Introduction

Pixel attribution methods

- Also known as: sensitivity map, saliency map, pixel attribution map, gradient-based attribution methods, feature relevance, feature attribution, and feature contribution
- Pixel attribution is a special case of feature attribution, but for images

Machine learning model $\hat{f}$ (typically neural network)

- Input: $\mathbf{x} \in \mathbb{R}^p$, i.e., p features (typically pixels)
- Output: $\hat{\mathbf{y}} = [\boldsymbol{S_1}(\mathbf{x}), \ldots, \boldsymbol{S_c}(\mathbf{x})] \in \mathbb{R}^C$, i.e., C outputs (typically one output per class to be predicted)

Explanation method

- Input: features $\mathbf{x} \in \mathbb{R}^p$ and machine learning model $\hat{f}$
- Output: relevance scores $\mathbf{R}^c = [\mathrm{R}_1^c, \ldots, \mathrm{R}_p^c] \in \mathbb{R}^p$ for each output $c \in C$

Types of pixel attribution approaches

- Occlusion- or perturbation-based (e.g., LIME, SHAP): manipulate parts of the image to generate explanations (model-agnostic)
- Gradient-based: compute the gradient of the prediction with respect to the input features (model-specific)

Vanilla gradient (saliency maps)

[Simonyan et al. 2013]

1. Perform a forward pass of the image of interest
2. Compute the gradient of class score of interest with respect to the input pixels:

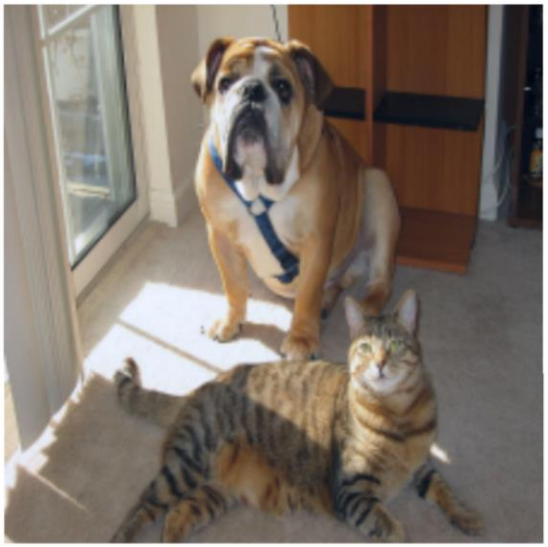
$$E_{grad}(\mathbf{x}_0) = \frac{\delta \boldsymbol{S}_\mathbf{c}}{\delta \mathbf{x}}\Big|_{\mathbf{x}=\mathbf{x}_0}$$

   Here we set all other classes to zero.
3. Visualize the gradients. You can either show the absolute values or highlight negative and positive contributions separately.

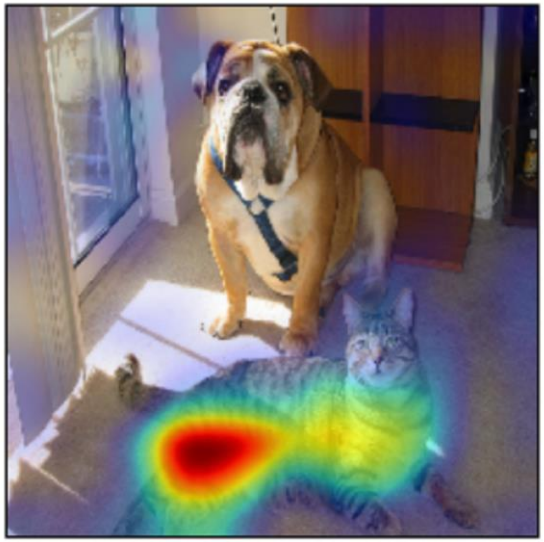
Grad-CAM [Selvaraju et al. 2017]
Introduction

Grad-CAM: Gradient-weighted Class Activation Map

- Provides explanations for convolutional neural networks (CNNs)
- Goal: understand at which parts of an image a convolutional layer “looks” for a certain classification
- Idea: last convolutional layer (before the fully connected layers on the right) has high-level semantic and spatial information



Original Image



Grad-CAM ‘Cat’

Grad-CAM ‘Dog’

Grad-CAM: How does it work? [Selvaraju et al. 2017]

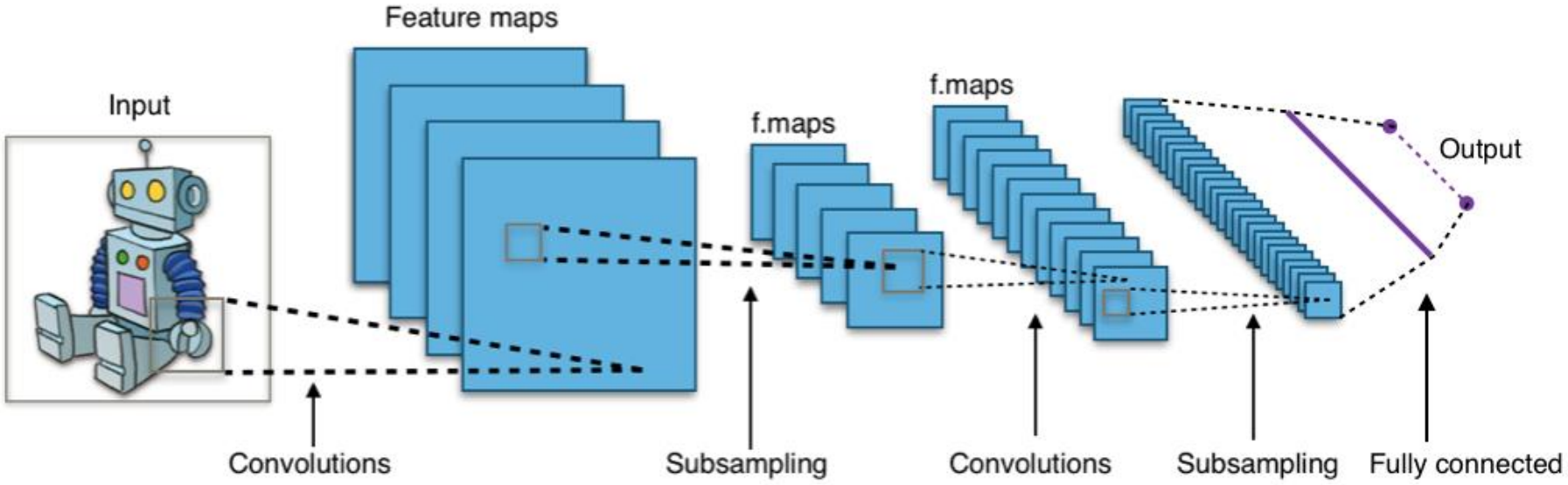
Introduction

- The gradient is not backpropagated all the way back to the image
- But to the last convolutional layer. Grad-CAM analyzes which regions are activated in the feature maps $A_1, A_2, \ldots, A_k$ of the last convolutional layer
- For each class c, decode how important a feature map is for this class
- Each pixel of the k feature maps is weighted according to its gradient

$$L^c_{Grad-CAM} \in \mathbb{R}^{U \times V} = ReLU\left(\sum_k \alpha^c_k A^k\right)$$

where U is the width, V the height of the explanation, and c the class of interest.

- ReLU ensures that only positive values are picked



Grad-CAM

Algorithm

1. Forward-propagate the input image through the convolutional neural network
2. Obtain the raw score for the class of interest
   (the activation of the neuron before the softmax layer)
3. Set all other class activations to zero
4. Back-propagate the gradient of the class of interest to the last convolutional layer before the fully connected layers: $\frac{\delta \hat{y}^c}{\delta A^k}$
5. Weight each feature map "pixel" by the gradient for the class. Indices u and v refer to the width and height dimensions and Z to the number of "pixels":

$$\alpha_k^c = \frac{1}{Z} \sum_u \sum_v \frac{\delta \hat{y}^c}{\delta A_{uv}^k}$$

   This means that the gradients are globally pooled.
6. Calculate an average of the feature maps, weighted per pixel by the gradient
7. Apply ReLU to the averaged feature map
   (we are only interested in activations that positively influence the prediction)
8. For visualization: Scale values to the interval between 0 and 1. Upscale the image and overlay it over the original image

SmoothGrad

[Smilkov et al. 2017]

Idea

- Make gradient-based explanations less noisy by adding noise and averaging over these artificially noisy gradients

SmoothGrad works in the following way:

1. Generate multiple versions of the image of interest by adding noise to it
2. Create pixel attribution maps for all images
3. Average the pixel attribution maps

Why should this work?

- Gradients fluctuate greatly at small scales
- Neural networks have no incentive during training to keep the gradients smooth, their goal is to classify images correctly
- Averaging over multiple maps "smooths out" these fluctuations:

$$R_{sg}(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{n} R(\mathbf{x} + \mathbf{g}_i)$$

$\mathbf{x}$	image
$\mathbf{g}_i \sim N(0, \sigma^2)$	noise vectors sampled from the Gaussian distribution
R	relevance
n	number of samples

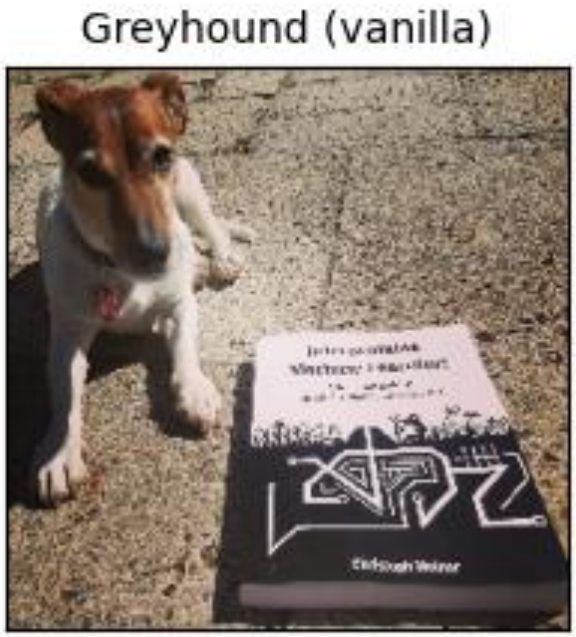
Pixel attribution (saliency maps)

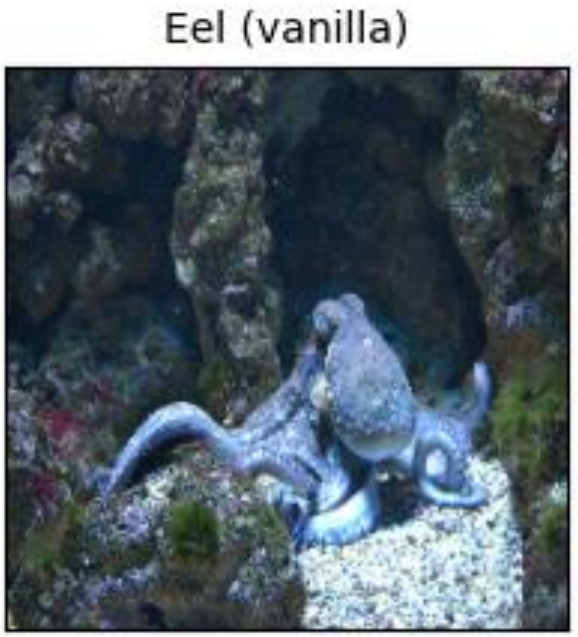
Examples

Neural network: VGG-16 trained on ImageNet

Example images (and their classification)

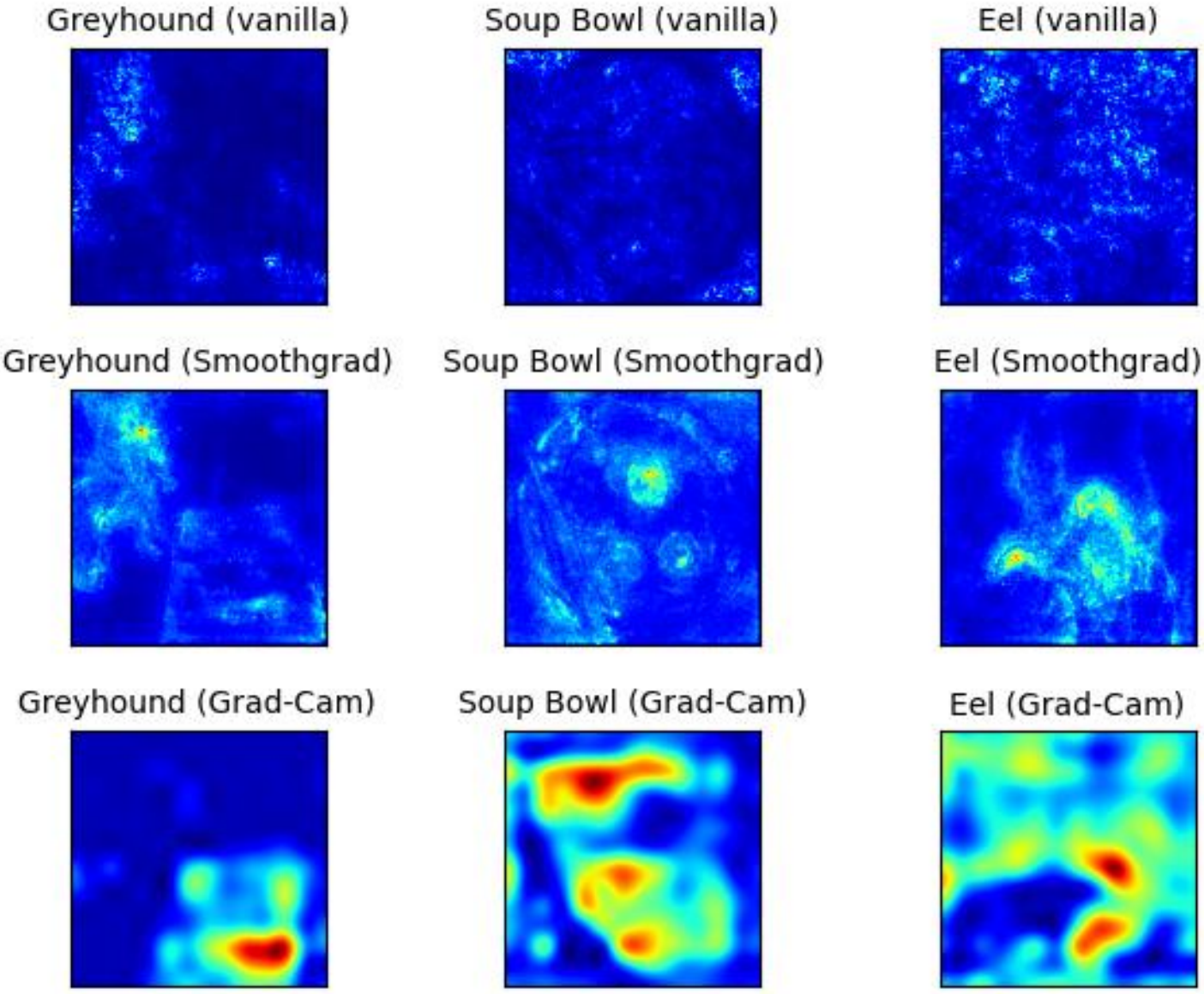
- Left: misclassification as greyhound (35% probability)
- Middle: correct classification as soup bowl (50% probability)
- Right: misclassification of octopus as eel (70% probability)





Christoph Molnar (https://christophm.github.io/interpretable-ml-book/images/original-images-classification.png), https://creativecommons.org/licenses/by-nc-sa/4.0/

Pixel attribution (saliency maps)



Pixel attribution (saliency maps)

Advantages and Disadvantages

Advantages

- Visual explanations: easy to recognize important regions
- Gradient-based methods usually faster than model-agnostic methods (LIME, SHAP)

Disadvantages

- Often no thorough evaluation in literature, evaluation often only with anecdotal examples
- Pixel attribution methods can be **fragile:** Ghorbani et al. (2019) showed that
  - introducing small (adversarial) perturbations to an image, which still lead to the same prediction,
  - can lead to very different pixels being highlighted as explanations

Detecting Concepts

Detecting concepts: TCAV [Kim et al. 2018]

Limitations of pixel attribution methods

- Low-level features are not user-friendly in terms of interpretability
- Example: importance of a single pixel in an image not very meaningful
- Not everything can be expressed in terms of feature (=pixel) importances

Idea: Concept-based approach

- A concept can be any abstraction, such as a color, an object, or even an idea
- Given a user-defined concept, detect that concept in the latent space (although a neural network was not explicitly trained with the concept)

Solution: TCAV - Testing with Concept Activation Vectors

- For any given concept, TCAV measures the extent of that concept's influence on the model's prediction for a certain class
- Example: TCAV can answer questions such as how the concept of "striped" influences a model classifying an image as a "zebra"
- TCAV describes the relationship between a concept and a class (global explanation) instead of explaining a single prediction (local explanation)

💡 Carefullly define concept

Ensure that the concept you're exploring is distinct and well-defined.

Detection concepts: TCAV

Two datasets per concept C required

- A concept dataset which represents C
  Example: for concept “striped”, images of striped objects
- Random dataset that consists of arbitrary data
  Example: random images (mostly) without stripes

Algorithm sketch

- Concept activation vector (CAV) $\mathbf{v}_l^C$ depends on layer l and concept C
- For layer l and concept C, train binary classifier that separates the activations of the concept dataset from the random dataset
- The coefficient vector of this classifier is then the CAV $\mathbf{v}_l^C$
- A CAV is simply the numerical representation that generalizes a concept in the activation space of a neural network layer
- Sensitivity of single data point: given an image input $\mathbf{x}$, we can measure its “conceptual sensitivity” $S_{C,k,l}$ where k is class to be predicted
- Global explanation per class: ratio of inputs with positive conceptual sensitivities to the number of inputs for a class:

$$TCAV_{C,k,l} = \frac{\left|\{\mathbf{x} \in \mathbf{X}_k : S_{C,k,l}(\mathbf{x}) > 0\}\right|}{|\mathbf{X}_k|}$$

Conceptual sensitivity

- We focus on the activations $\hat{f}_l(\mathbf{x})$ of a hidden layer l
- We train a binary classifier g that receives the activations $\hat{f}_l(\mathbf{x})$ of layer l and predicts the concept C
- The coefficient vector of this trained binary classifier g is then the CAV $\mathbf{v}_l^C$
- Given an image input $\mathbf{x}$, we measure its "conceptual sensitivity" by multiplying the gradient of the activations with the CAV

$$S_{C,k,l}(\mathbf{x}) = \nabla h_{l,k}\left(\hat{f}_l(\mathbf{x})\right) \cdot \mathbf{v}_l^C$$

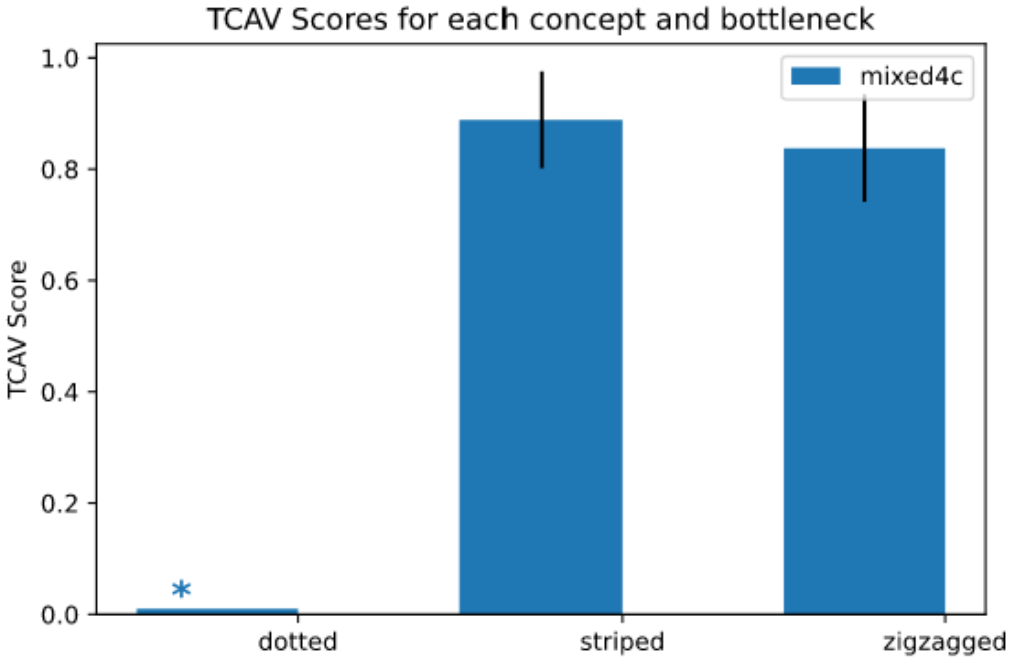
where $\hat{f}$ maps the input $\mathbf{x}$ to the activation vector of the layer l and $h_{l,k}$ maps the activation vector to the logit output of class k

- Remarks:
  - The sign of $S_{C,k,l}(\mathbf{x})$ depends on the angle between $\nabla h_{l,k}\left(\hat{f}_l(\mathbf{x})\right)$ and $\mathbf{v}_l^C$
  - If the angle is less than 90 degrees, $S_{C,k,l}(\mathbf{x})$ will be positive (otherwise negative)
  - Intuitively: $S_{C,k,l}$ indicates whether $\mathbf{v}_l^C$ points in a similar direction like $\nabla h_{l,k}$
  - Thus, $S_{C,k,l}(\mathbf{x}) > \mathbf{0}$ can be interpreted as concept C encouraging the model to classify x into class k

Example: Zebra

How does the concept “striped” influence the model while classifying images as “zebra”?

- Collect data that are labeled as “zebra”
- Calculate conceptual sensitivity for each input image
- TCAV score of the concept C=“striped”and class k=“zebra”: number of “zebra” images with positive sensitivities divided by total number of the “zebra” images
- TCAV score=0.8 indicates that 80% of predictions for “zebra” class are positively influenced by the concept of “striped”



Example of measuring TCAV scores of three concepts for the model predicting “zebra”. The targeted bottleneck is a layer called “mixed4c”. A star sign above “dotted” indicates that “dotted” has not passed the statistical significance test, i.e., having the p-value larger than 0.05. Both “striped” and “zigzagged” have passed the test, and both concepts are useful for the model to identify “zebra” images according to TCAV.

Detecting concepts: Advantages

TCAV does not require users to have machine learning expertise

- Explanations are in terms of human-defined concepts
- This make TCAV particularly suitable for domain-experts

Customizability of explanations beyond feature attribution

- Users can investigate any concept as long as the concept can be defined by its concept dataset
- A user can control the balance between the complexity and the interpretability of explanations based on their needs (e.g., coarse-grained vs fine-grained)

Generates global explanations

- Concepts are related to classes (concepts explain classes globally)
- Example:
  - Classifier predicts “zebra” with a high accuracy
  - TCAV score is higher for the concept “dotted” instead of “striped”
  - This might indicate that the classifier is accidentally trained by an unbalanced dataset
  - Improve model by adding more “striped zebra” images or less “dotted zebra” images to the training dataset

Detecting concepts: Disadvantages

Might perform badly on shallower neural networks

- If a network is too shallow, its layers may not be capable of separating concepts clearly so that TCAV is not applicable

Requires additional annotations for concept datasets

- Annotations might be expensive to obtain (e.g., “striped” vs. not “striped”)

Mostly suitable for image data

- Limited applications in text data and tabular data (as of now)

References

- **Kim**, Been, Martin Wattenberg, Justin Gilmer, Carrie Cai, James Wexler, and Fernanda Viegas. "Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav)." In *International conference on machine learning*, pp. 2668-2677. PMLR, **2018**.
- **Selvaraju**, Ramprasaath R., Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. "Grad-cam: Visual explanations from deep networks via gradient-based localization." In *Proceedings of the IEEE international conference on computer vision*, pp. 618-626. **2017**.
- **Simonyan**, Karen, Andrea Vedaldi, and Andrew Zisserman. "Deep inside convolutional networks: Visualising image classification models and saliency maps." *arXiv preprint arXiv:1312.6034* (**2013**).
- **Smilkov**, Daniel, Nikhil Thorat, Been Kim, Fernanda Viégas, and Martin Wattenberg. "Smoothgrad: removing noise by adding noise." *arXiv preprint arXiv:1706.03825* (**2017**).
- **Olah**, Chris, Alexander Mordvintsev, and Ludwig Schubert. “Feature visualization.” Distill 2, no. 11 (**2017**). https://distill.pub/2017/feature-visualization/