Hot Topics in Machine Learning Safety

Explainability

OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG

Topics

- Introduction and Motivation
- ML Fundamentals
- Testing
- **Explainability** 👉
- Adversarial Machine Learning
- Uncertainty Estimation
- Anomaly Detection
- Alignment

Agenda

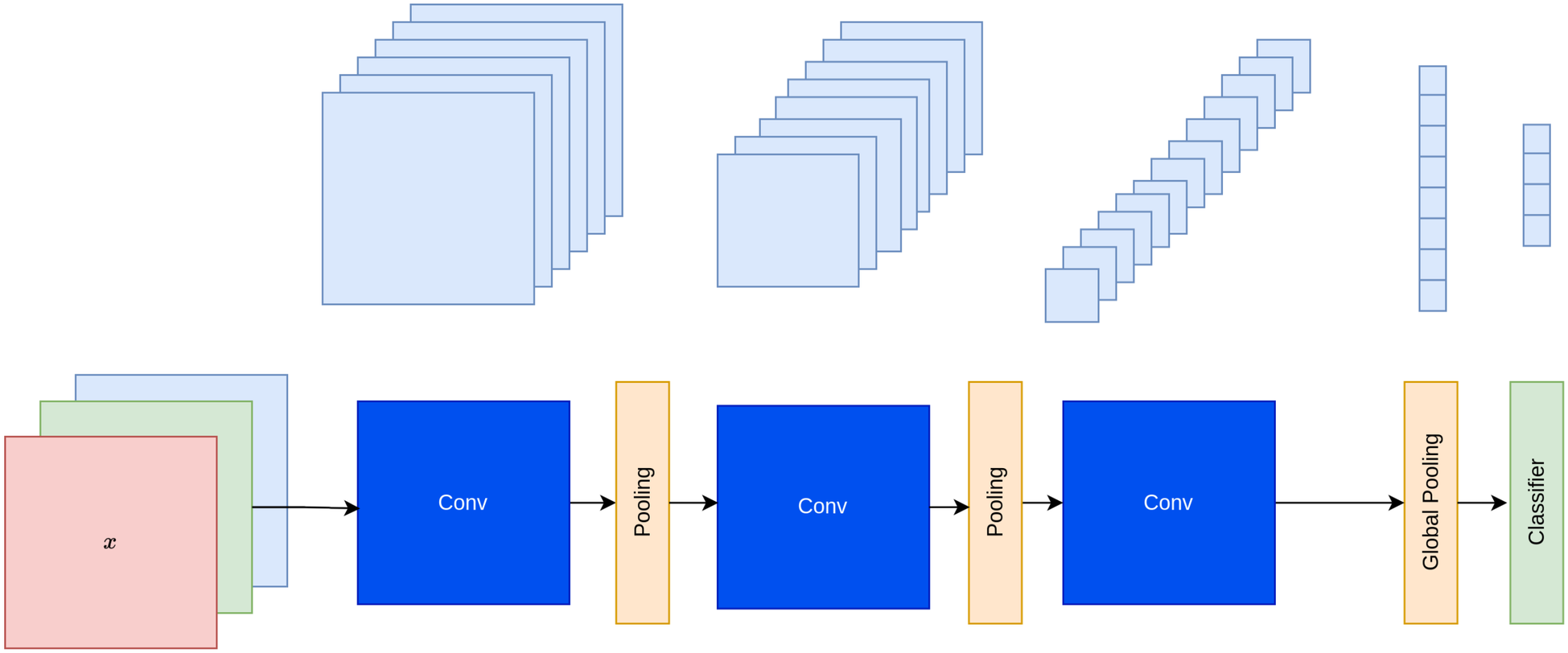
- How do CNNs work?
- What do we mean by explainability?
- Local explainability
  - Occlusion Method
  - Saliency Maps
  - Class Activation Maps

- Global explainability
  - Filter Visualization

Explainability

- Similar concepts: transparency, interpretability
- Generally difficult to define
- Global explainability: we can retrace decisions, we understand the system bottom up
- Local explainability: for a single input, we can say why

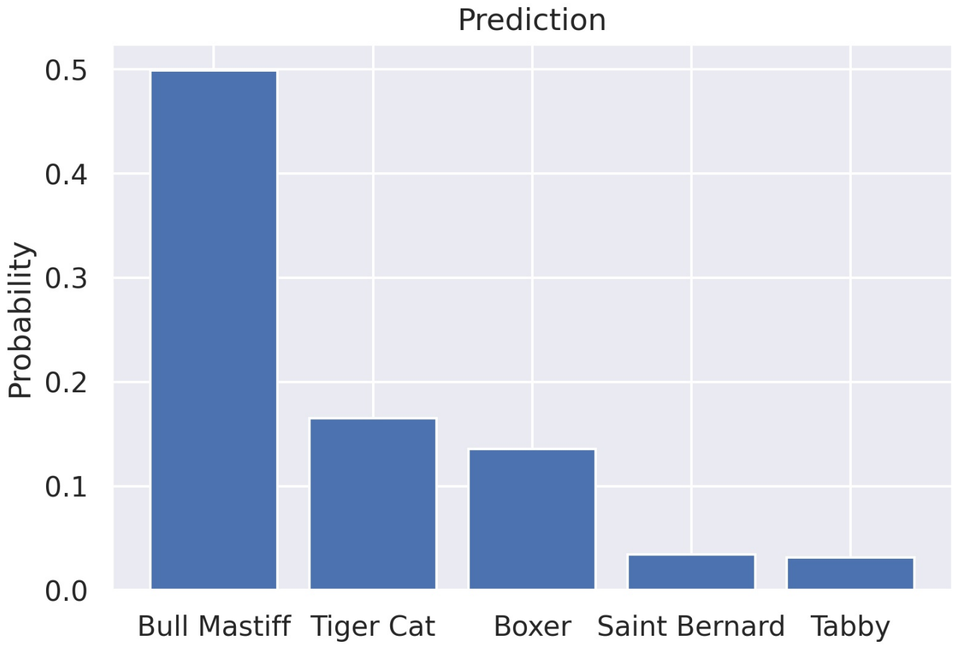
CNNs


x
Conv
Pooling
Conv
Pooling
Conv
Global Pooling
Classifier

Explainability - Why should I care?

Tank Anekdote

Example


Prediction
Probability
0.5
0.4
0.3
0.2
0.1
0.0
Bull Mastiff
Tiger Cat
Boxer
Saint Bernard
Tabby

Ideas?

Occlusion Method

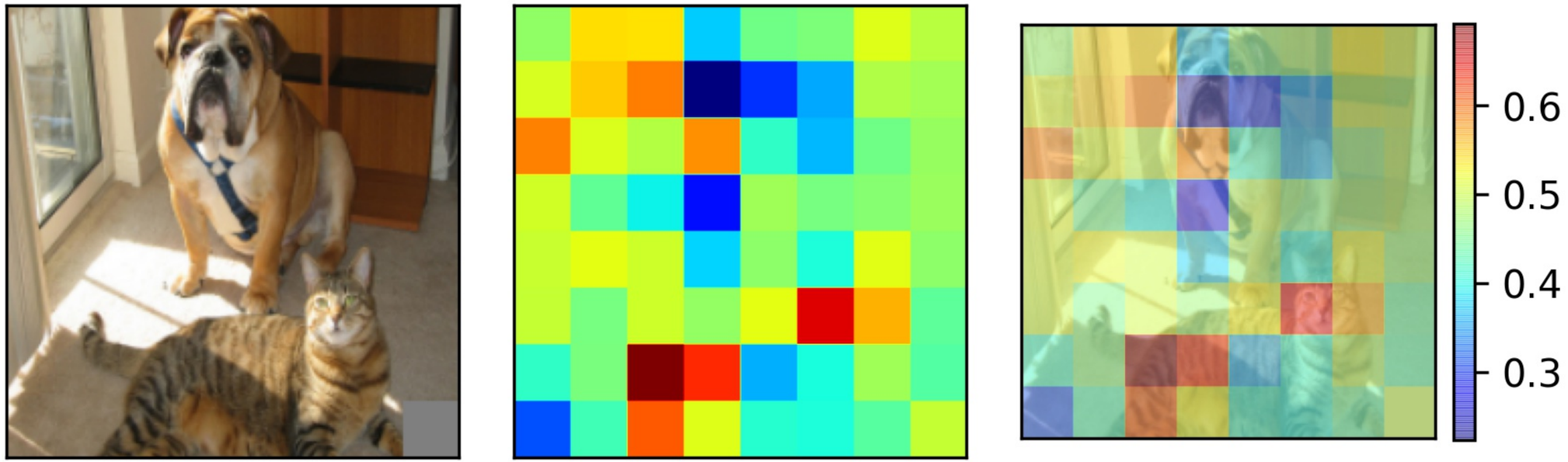
- Select a class (e.g., dog)
- Occlude a portion of your input
- Measure how the class-probability changes
- Repeat until everything was occluded once

OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG

Occlusion Method: Dog Class

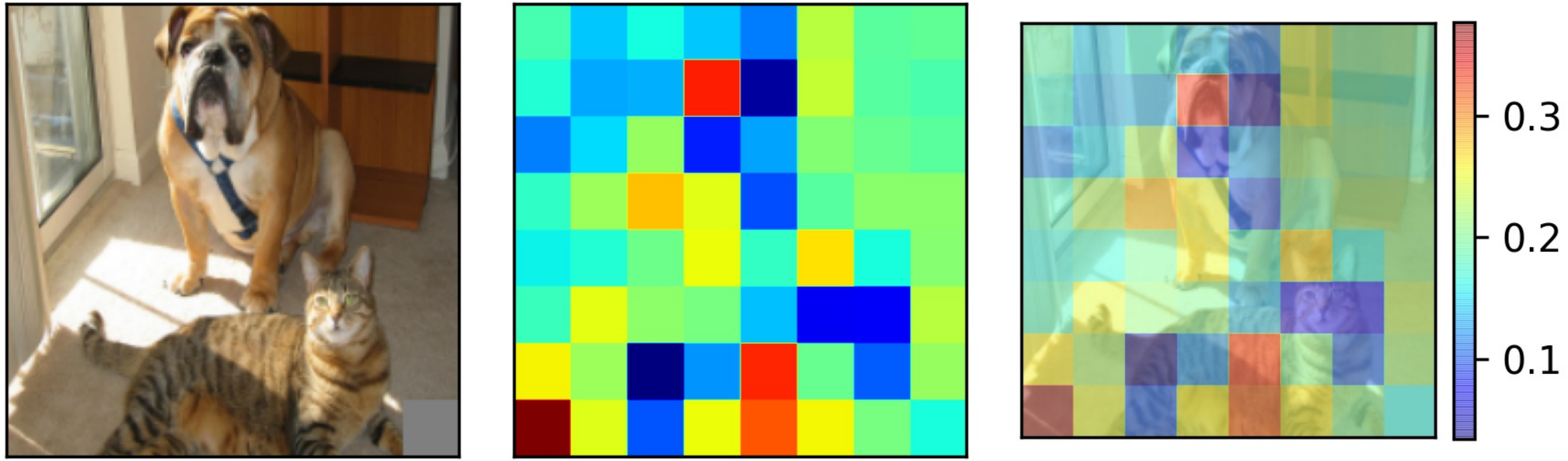
OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG

Occlusion Method: Dog Class

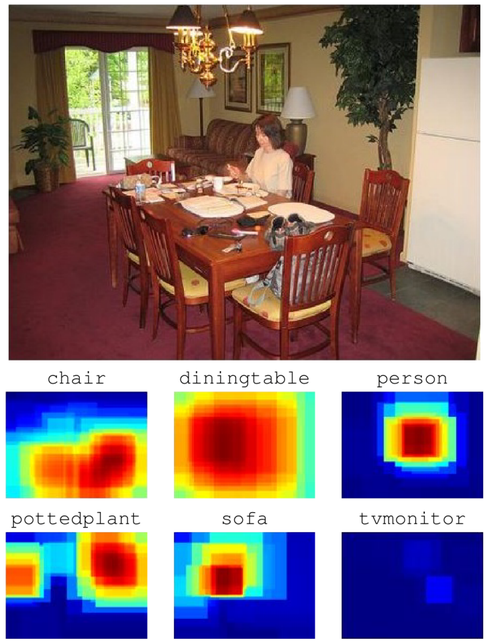


Occlusion Method: Cat Class

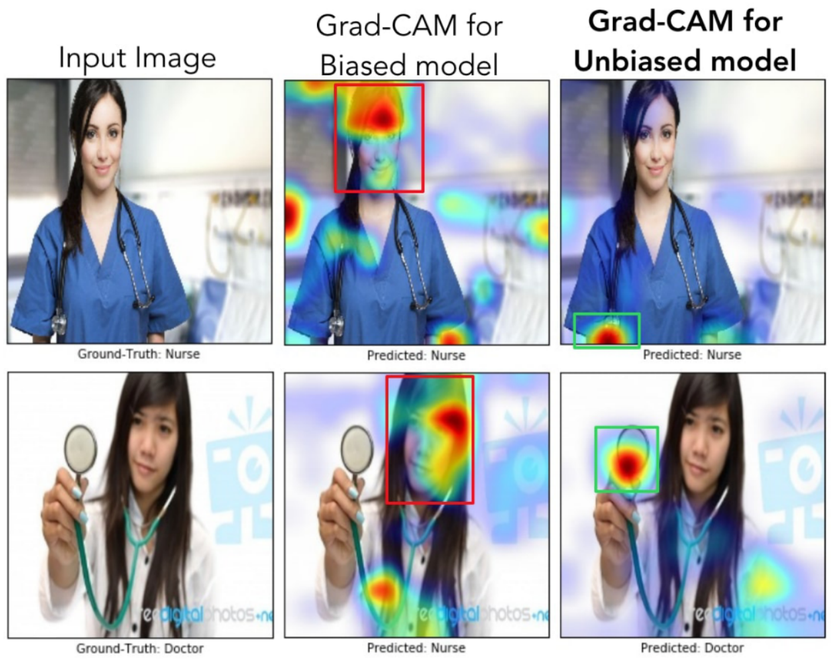
Occlusion Method: Cat Class



Applications: Object Detection



Applications: Model Bias



Occlusion Method

- Easy to implement
- Higher Resolution requires more samples, gets expensive
- Maps do not look too good ...
- They also include "negatives"

Saliency Maps

- Occlusion: How relevant is input x_i for the prediction $\hat{y}_c$?
- In other words: how sensitive is out prediction to changes in x_i
- We can measure this by the derivative $\frac{\partial \hat{y}_c}{\partial x_i}$
- I.e. we calculate the gradient of the **input image** x w.r.t. the prediction $\hat{y}_c$

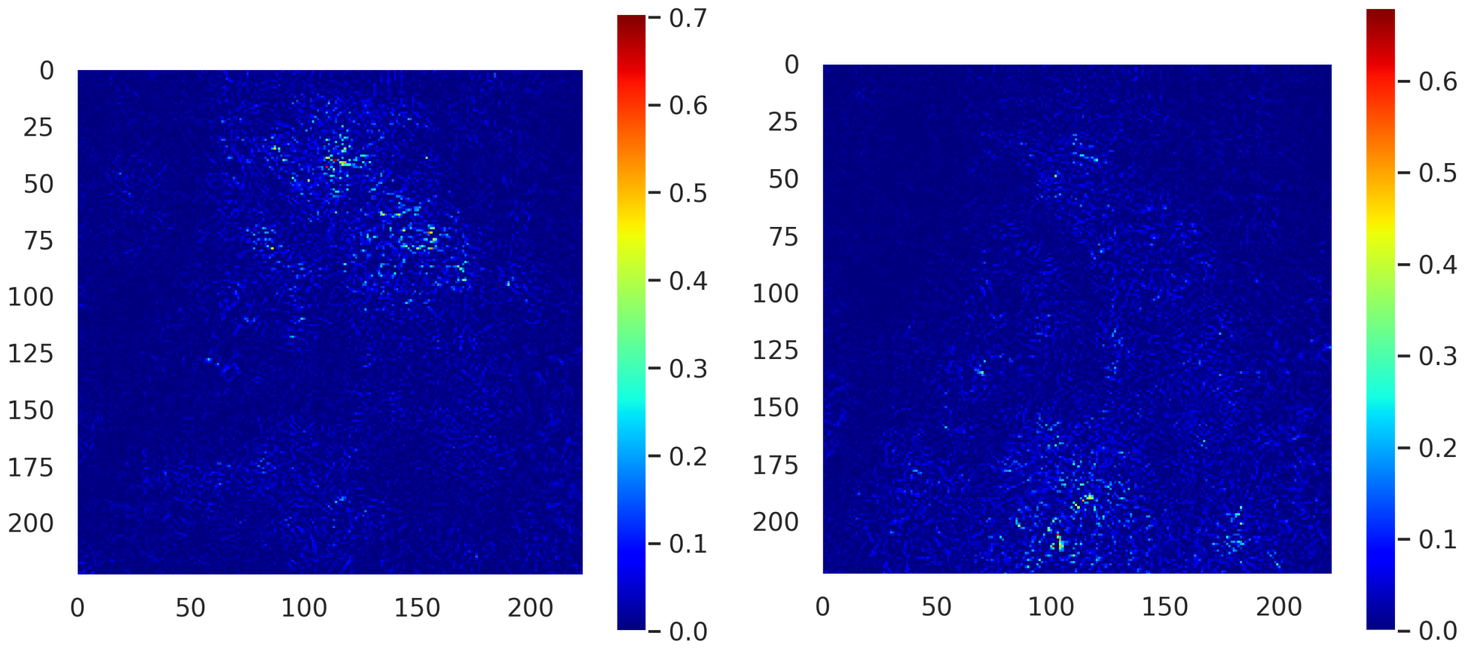
Saliency Maps

Details:

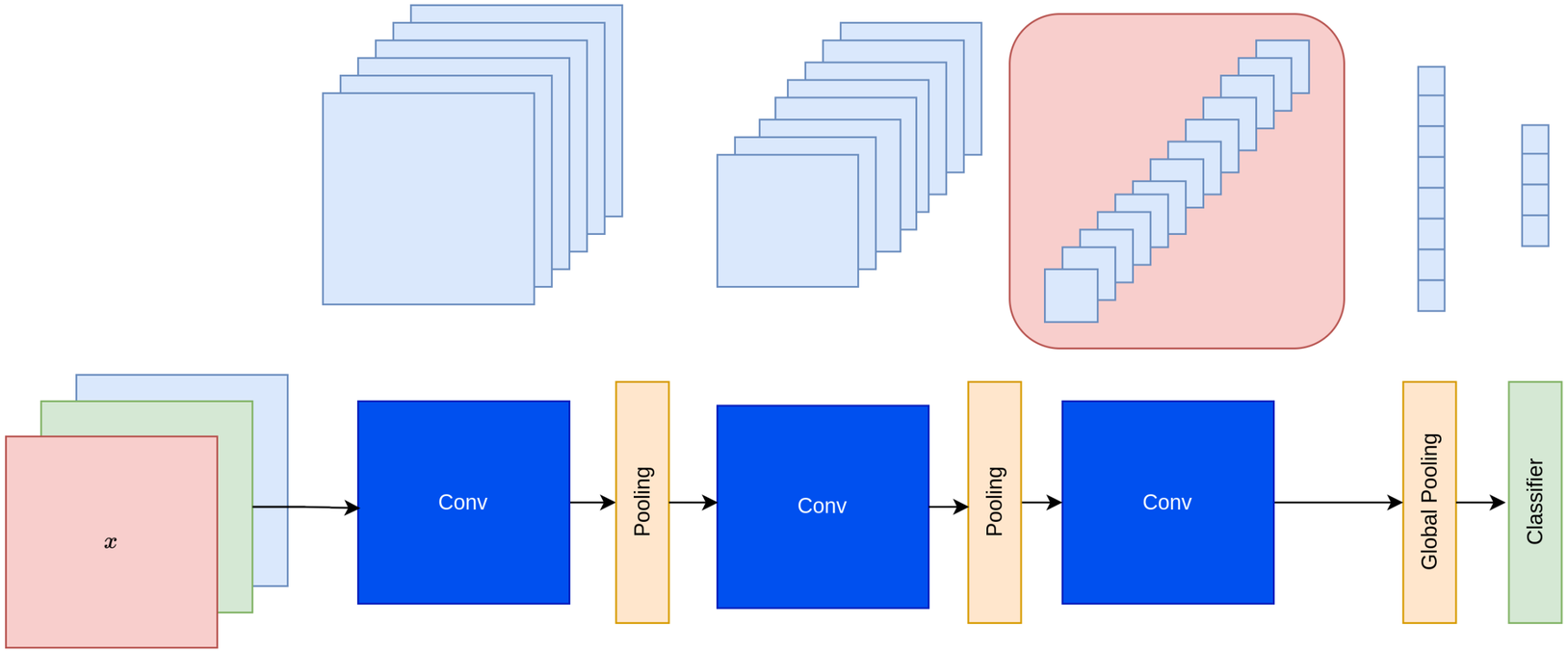
- We are only interested in positive gradients
- We take the mean over all RGB-Pixels

```
1 img = torch.tensor(img, requires_grad=True)
2
3 y_hat = model(img)
4 class_score = y_hat[243] # score for class "dog" y_c
5 class_score.backward()
6
7 grad = img.grad.data # dx/dy_c
8 pos_grads = grad.relu() # only gradients > 0
9 saliency_map = pos_grads.mean(dim=0) # mean over RGB
```

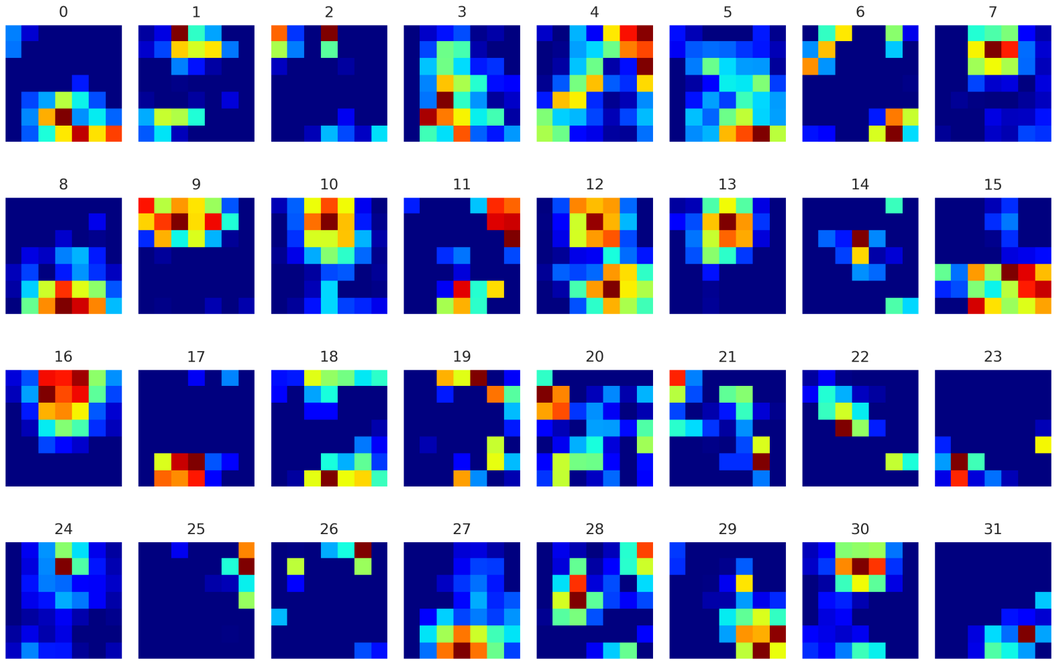
Saliency Maps



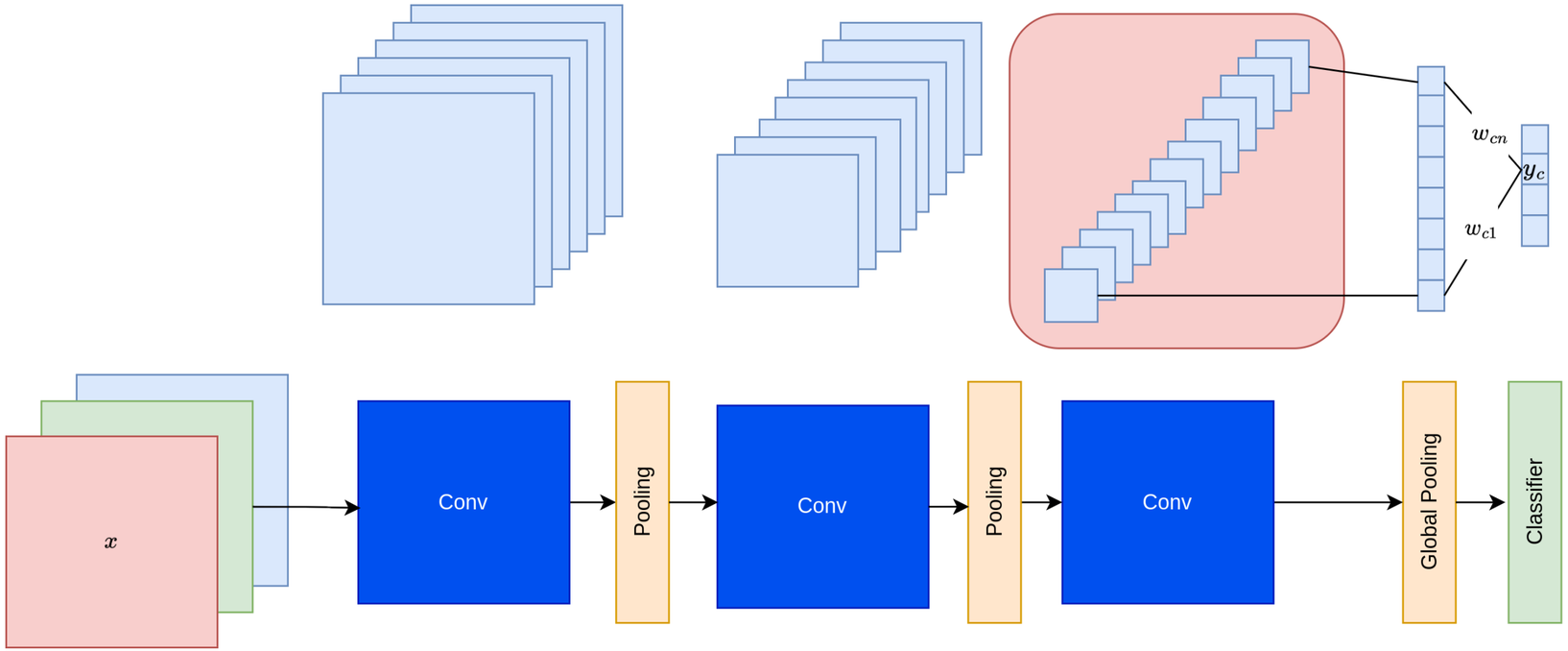
Class Activation Maps



Class Activation Maps



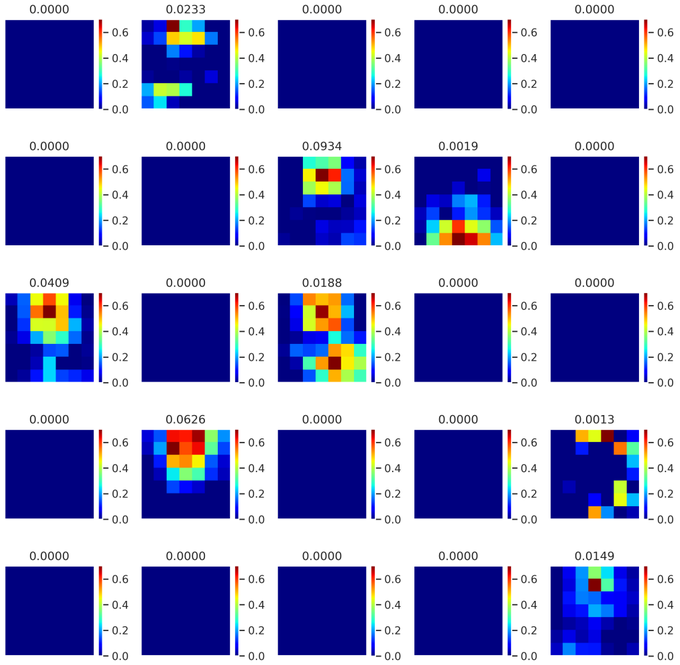
Class Activation Maps

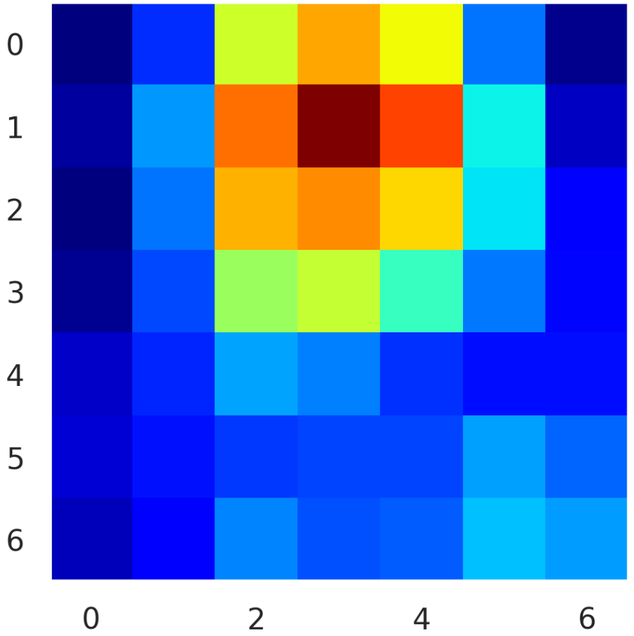


Class Activation Maps

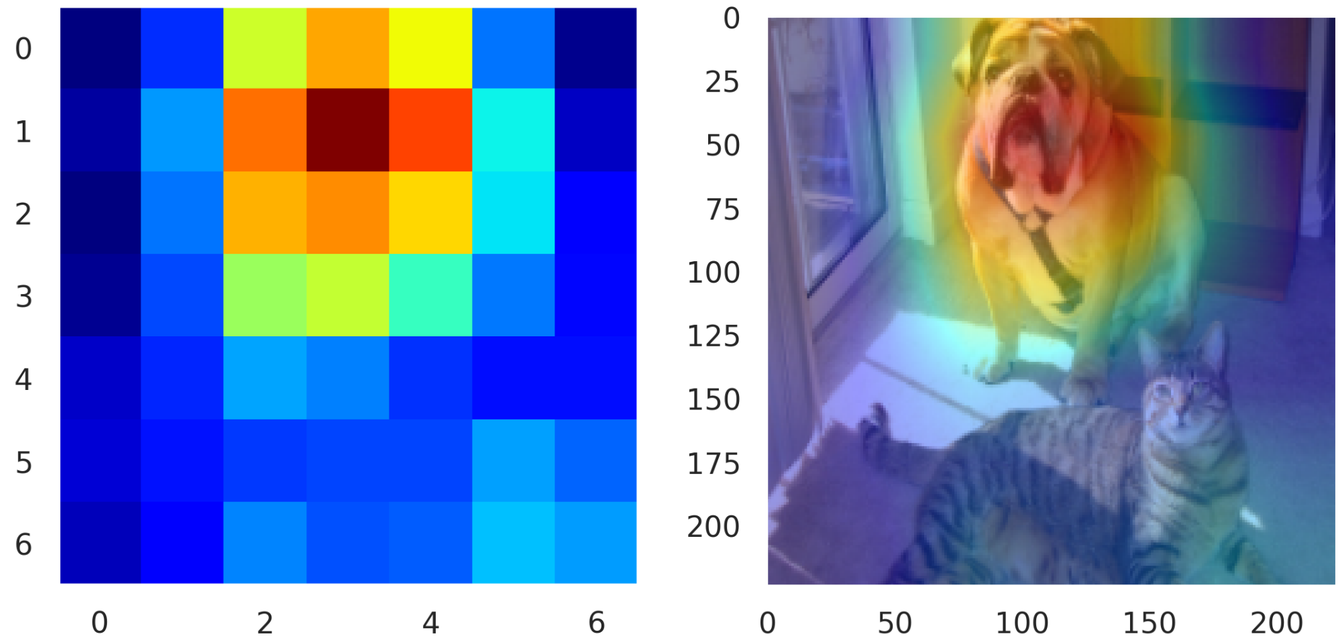
- Take activation maps for input x
- Take weights w_c from classifcation layer for some class c
- Set all negative weight $w_{ci} < 0$ to 0
- Weight each activation map by its corresponding weight w_{ci}
- Calculate mean over all maps
- Scale up to input size

Class Activation Maps - Dog

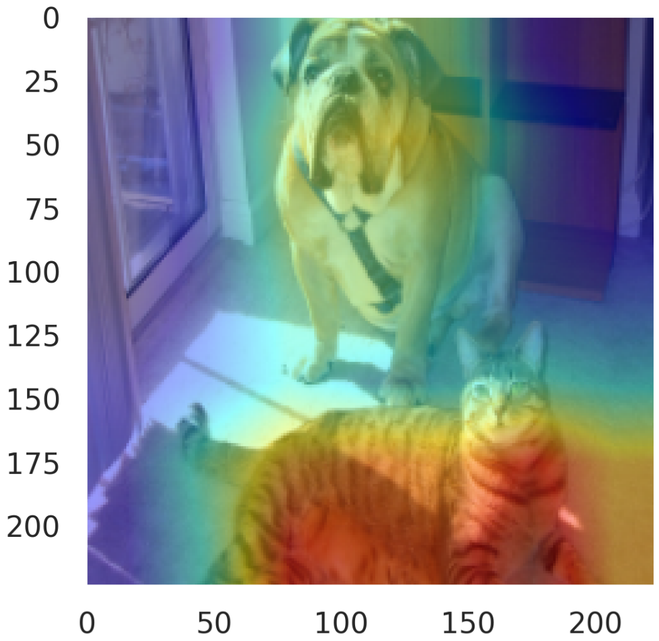




Class Activation Maps - Dog



Class Activation Maps - Cat



Filter Visualization

- How can we know what a convolutional filter is doing?
- Idea: Find pattern that maximally activates the filter

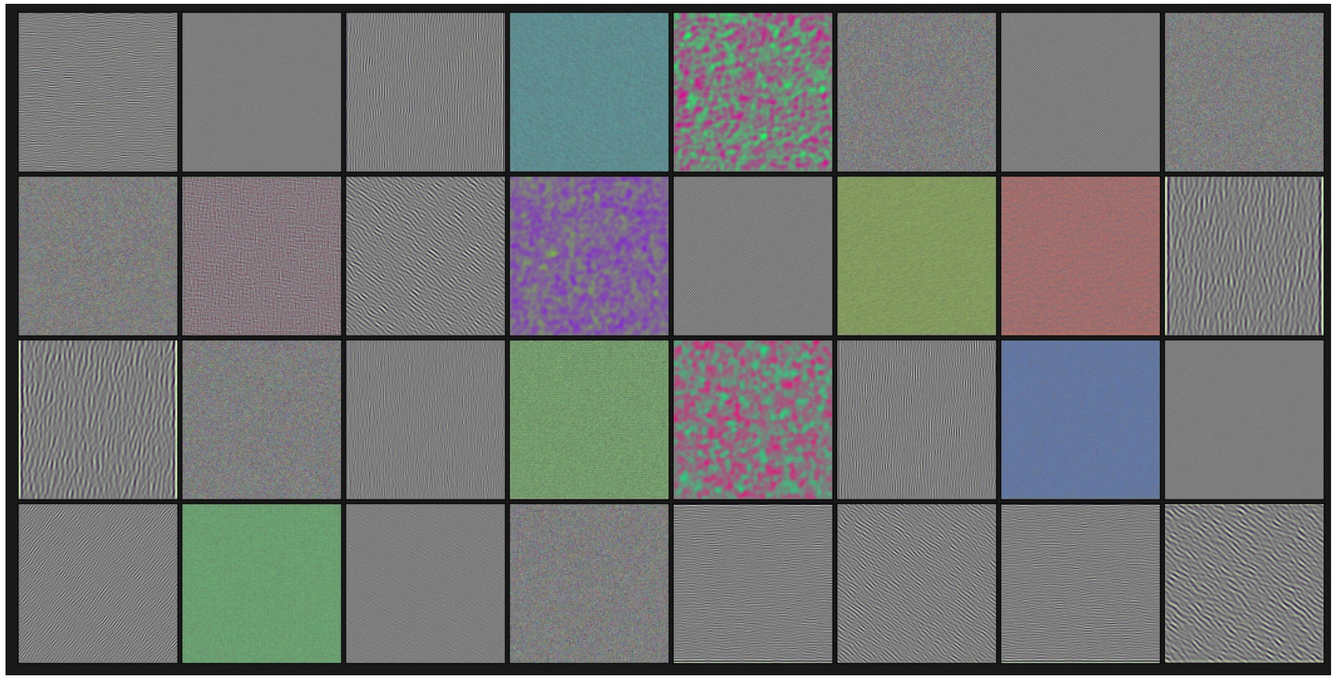
- Start with noise
- Use Gradient ascend to optimize norm of activations

Saliency Maps - Pseudocode

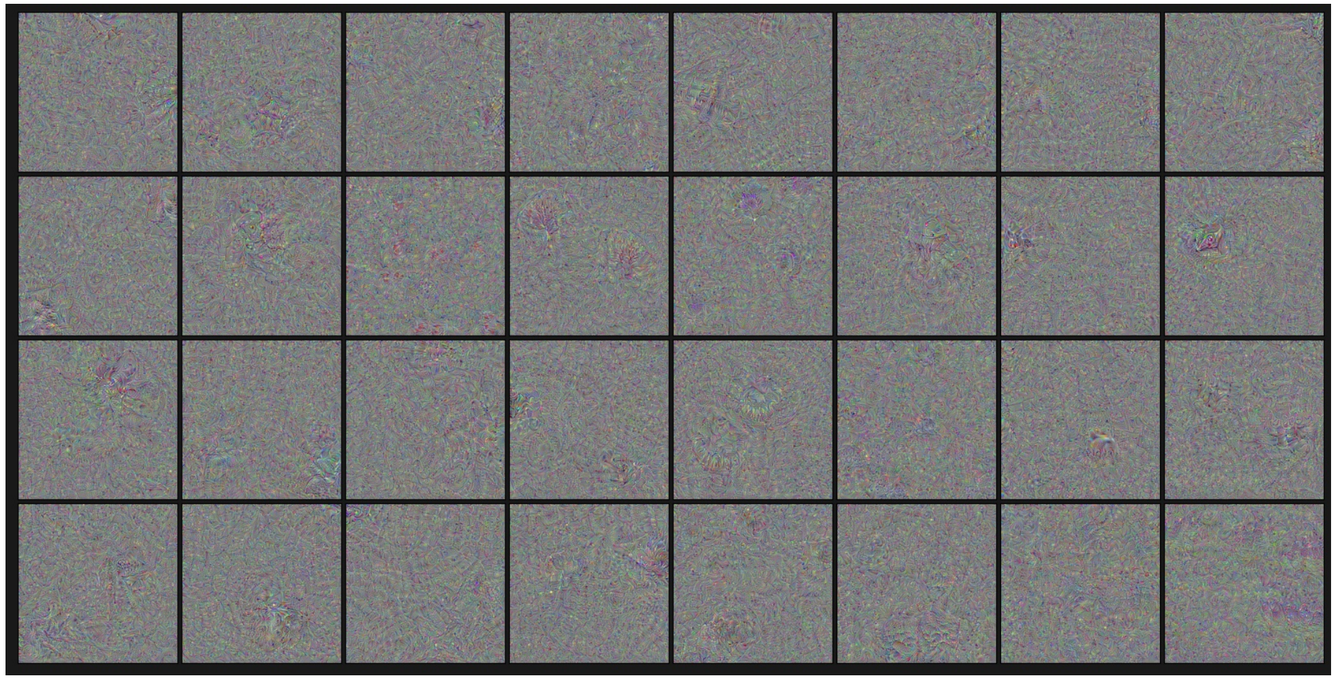
```
img = torch.randn(size=(1, 3,256,256))

x = torch.tensor(img, requires_grad = True)

for i in range(50): # optimization loop
  out = net.conv1(x)
  activations = out[0, filter_no]

  x_v.grad.data = torch.zeros_like(x_v.grad.data) # clear gradients
  loss = activations.pow(2).sum().sqrt() # norm
  loss.backward()

  with torch.no_grad():
    x += x.grad * 0.01
```

OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG

Filter Visualization - Lower Layers



Filter Visualization - Higher Layers



Applications

- Understand what model is doing
- Identify unused filters
- Identify redundant filters

Take Home Assignments (Optional)

- Read "Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization"
  - https://arxiv.org/abs/1610.02391

- E-Learning: PyTorch Notebook
  - Jupyter Notebook with exercises, available soon