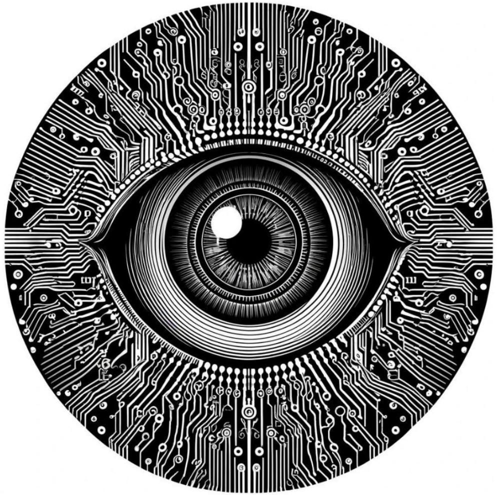
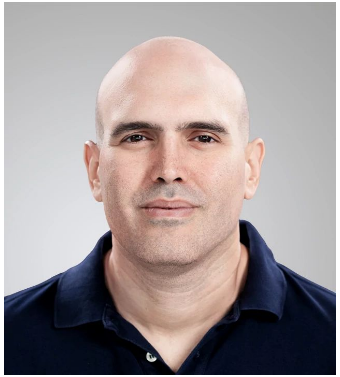
Workshop 7 - Interpretability with Class Activation Mapping



Antonio Rueda-Toicen

About me

- AI Researcher at Hasso Plattner Institute, AI Engineer & DevRel for Voxel51
- Organizer of the Berlin Computer Vision Group
- Instructor at Nvidia's Deep Learning Institute and Berlin's Data Science Retreat
- Preparing a MOOC for OpenHPI (now open for registration)



LinkedIn

How to use our Discord channel during the workshop

- Our channel is **#practical-computer-vision-workshops.** Please ask all questions there instead of the Zoom chat. Through Discord we can have better and more detailed discussions.

- **Step 1 - Use the Discord invite on the Voxel51 website** https://discord.com/invite/fiftyone-community
- **Step 2 - Access channel** (use the direct link below or search) http://bit.ly/3YmvPXG

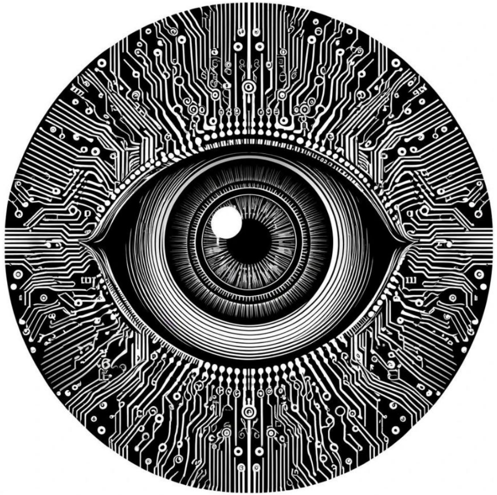
Agenda

- Pooling in Neural Networks
- Interpretability with Class Activation Mapping

Notebook

- Class Activation Mapping - ResNet34 (Google Colab)

Pooling in Neural Networks

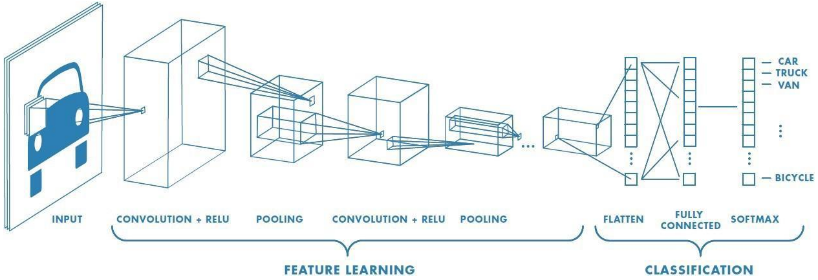


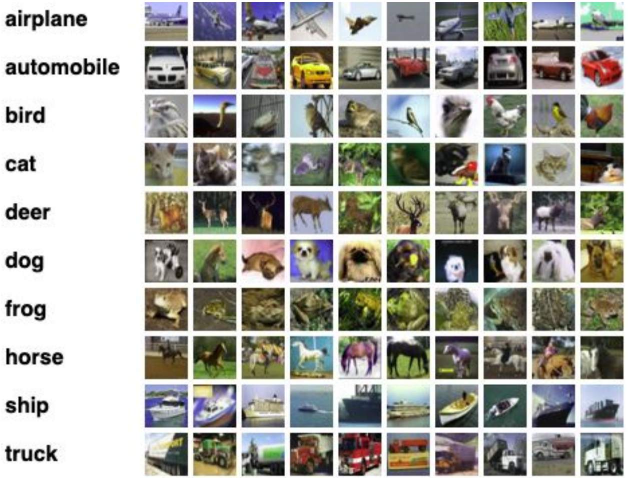
Learning goals

- Explore image downsampling and reduction of network parameters with mean, max, and global pooling

Architecture of a convolutional network with pooling

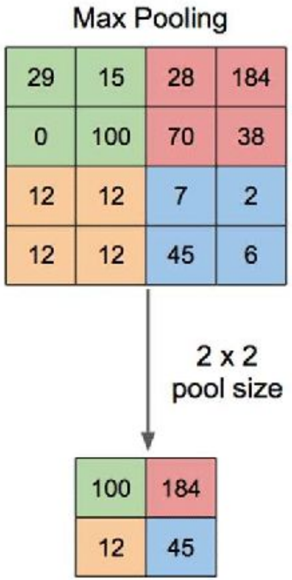
```
nn.Sequential(
            nn.Conv2d(3, 16, kernel_size=5, stride=1, padding=2),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),

            nn.Conv2d(16, 32, kernel_size=5, stride=1, padding=2),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),

            nn.Flatten(),
            nn.Linear(32 * 8 * 8, 120),
            nn.ReLU(),
            nn.Linear(120, 84),
            nn.ReLU(),
            nn.Linear(84, 10), # 10 classes, we are working with CIFAR 10
        )
```

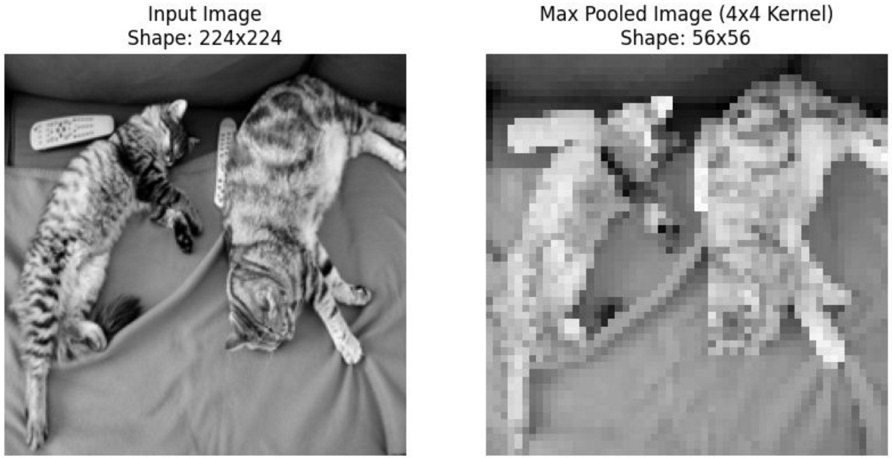




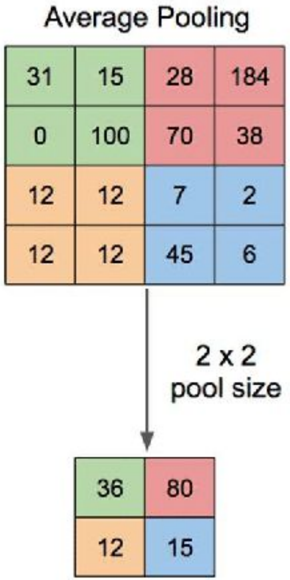
[The CIFAR-10 dataset]

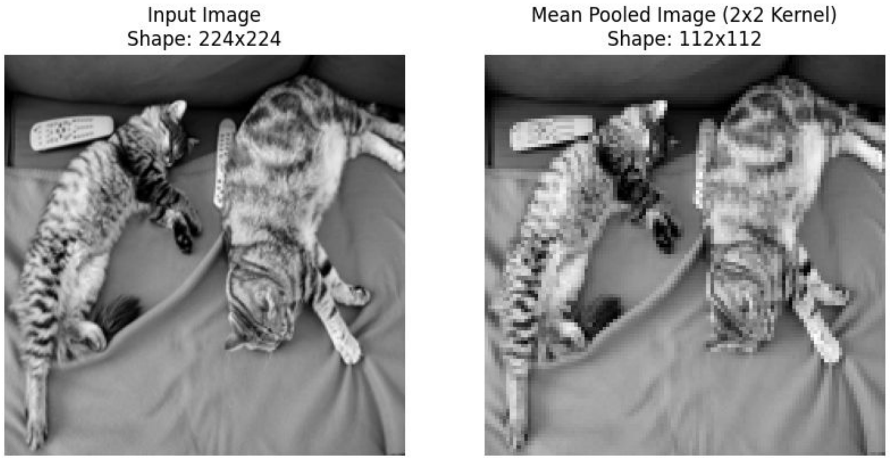
Max Pooling


Max Pooling
29
15
28
184
0
100
70
38
12
12
7
2
12
12
45
6
2 x 2
pool size
100
184
12
45

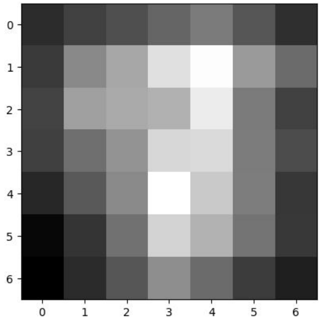

Input Image
Shape: 224x224
Max Pooled Image (4x4 Kernel)
Shape: 56x56

Mean Pooling



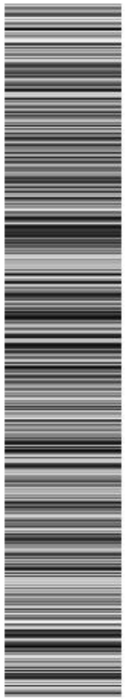


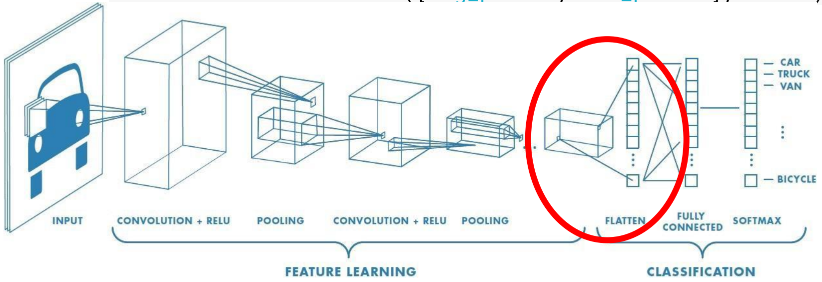
Global pooling aka adaptive pooling




```python
import torch.nn as nn


class GlobalPooling(nn.Module):
    def __init__(self):
        super().__init__()
        # Computes the average value of each activation map
        self.avg_pool = nn.AdaptiveAvgPool2d((1, 1))
        # Selects the max value of each activation map
        self.max_pool = nn.AdaptiveMaxPool2d((1, 1))

    def forward(self, x):
        avg_pooled = self.avg_pool(x).flatten(1)
        max_pooled = self.max_pool(x).flatten(1)
        # Concatenates both poolings to produce a feature vector
        return torch.cat([avg_pooled, max_pooled], dim=1)
```

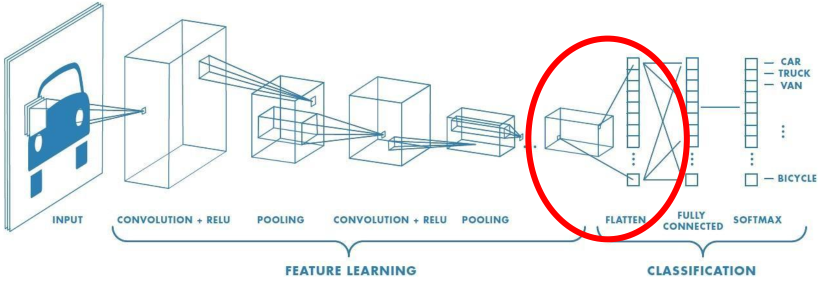
Feature vector

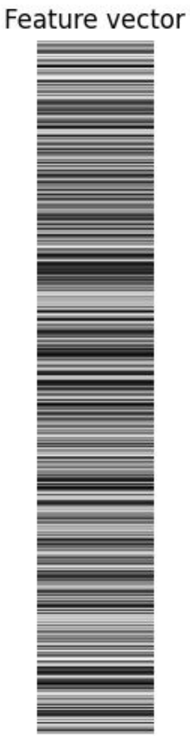




Using global pooling to reduce parameters

```python
import torch
import torch.nn as nn

model = nn.Sequential(
    nn.Conv2d(3, 16, kernel_size=5, stride=1, padding=2),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size= 2, stride=2),

    nn.Conv2d(16, 32, kernel_size=5, stride=1, padding=2),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size= 2, stride=2),

    GlobalPooling(),   # This replaces the Flatten layer

    nn.Linear(64, 120),
    nn.ReLU(),
    nn.Linear(120, 84),
    nn.ReLU(),
    nn.Linear(84, 10)
)
```





Summary

Pooling is a way to compress information

- Pooling allows us to do lossy compression while retaining important visual features
- Convolutions with stride > 1 achieve a similar effect at the cost of a higher number of parameters

Global pooling is an alternative to flattening activation maps

- We can create feature vectors (aka embeddings) using the global mean and/or max pooling operations

Further reading and references

A guide to convolution arithmetic for deep learning

- https://arxiv.org/abs/1603.07285

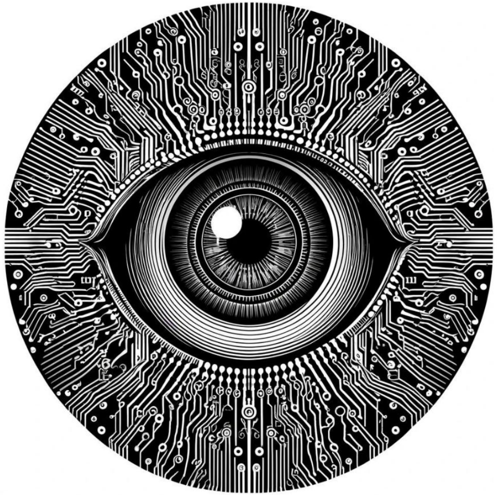
Network in network (1x1 convolutions)

- https://arxiv.org/abs/1312.4400

Hypercolumns for object segmentation and fine-grained localization

- https://openaccess.thecvf.com/content_cvpr_2015/papers/Hariharan_Hypercolumns_for_Object_2015_CVPR_paper.pdf

Interpretability with Class Activation Mapping



Learning goals

- Use class activation mapping to interpret the output of classifiers
- Describe tradeoffs between CAM and Grad-CAM

Interpreting a prediction

A pretrained Resnet34 with Imagenet1K_V1 weights says

“tabby, tabby cat” with

probability = 0.59

The resnet has 512 activations of shape $H = W = 7$ on its last layer, when we apply the 224x224 Resize on the input

Class Activation Mapping

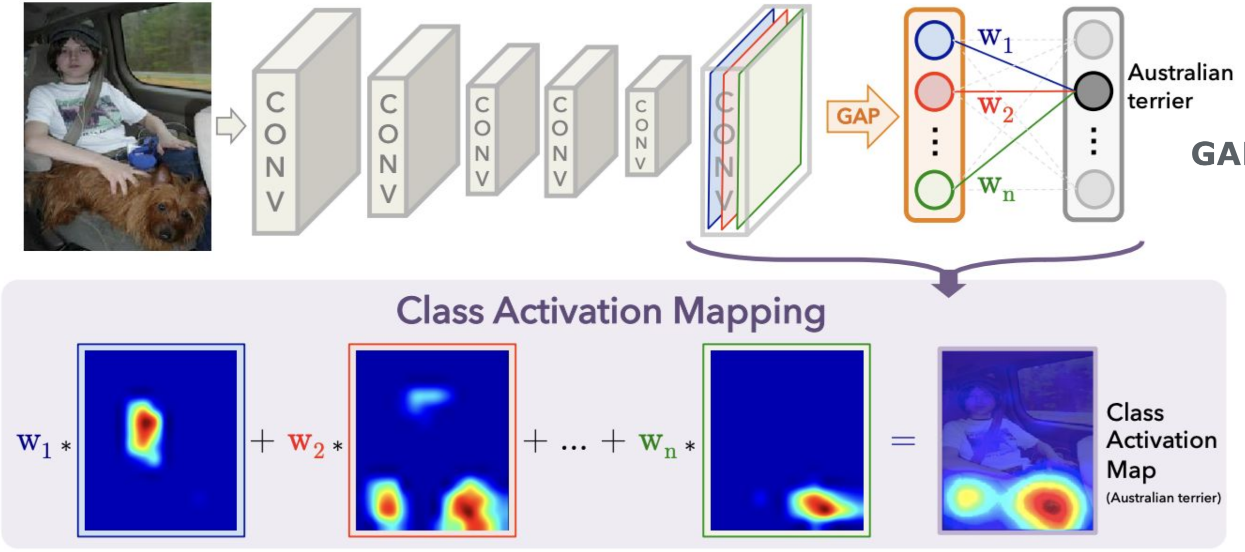


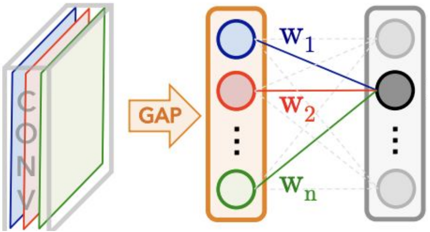
Figure 2. Class Activation Mapping: the predicted class score is mapped back to the previous convolutional layer to generate the class activation maps (CAMs). The CAM highlights the class-specific discriminative regions.

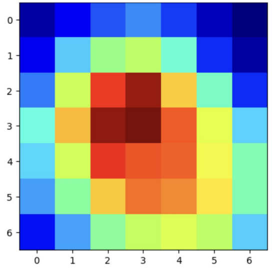
GAP = Global Average Pooling

Image from [Learning Deep Features for Discriminative Localization]()

CAM with global average pooling

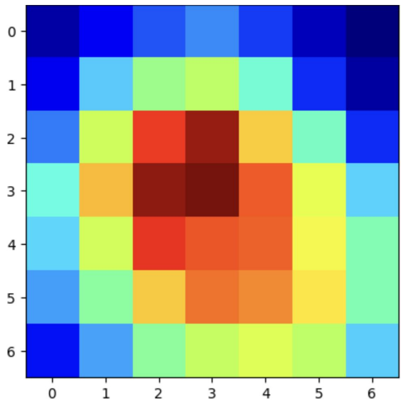
```python
import torch
import torch.nn.functional as F

# Suppose fc_weights has shape [num_classes, channels] and activations has shape [1, batch_size, channels, H, W].
# We'll just show the relevant slices for the single 'class_idx' and the first image in the batch.
weight = fc_weights[class_idx]         # shape [channels]
act = activations[0][0]                # shape [channels, H, W] - 7x7 in the case of resnet34

# ----------------------------------------------------------------
# 1) The "global average pooling" from a usual forward pass:
#    collapses (H, W) -> 1x1, giving us one value per channel.
# ----------------------------------------------------------------
pooled = F.adaptive_avg_pool2d(act.unsqueeze(0), 1)   # shape [1, channels, 1, 1]
pooled = pooled.squeeze(0).squeeze(-1).squeeze(-1)    # shape [channels]
# 'pooled' is the channel-wise average. Multiplying by 'weight' then summing would give the final logit for 'class_idx'.

score = (pooled * weight).sum()  # The single scalar logit for class_idx

# ----------------------------------------------------------------
# 2) Building the CAM:
#    multiply each channel map by its weight, then sum across channels.
# ----------------------------------------------------------------

cam = (act * weight.view(-1, 1, 1)).sum(dim=0)  # shape [H, W]

print(cam.shape)  # [H, W] this is now shape 7x7
```

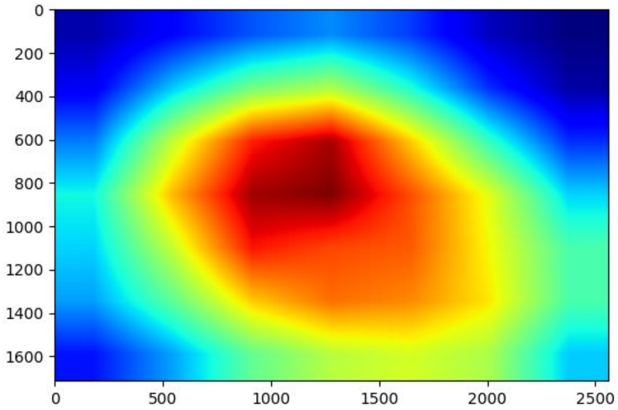




Resizing the map

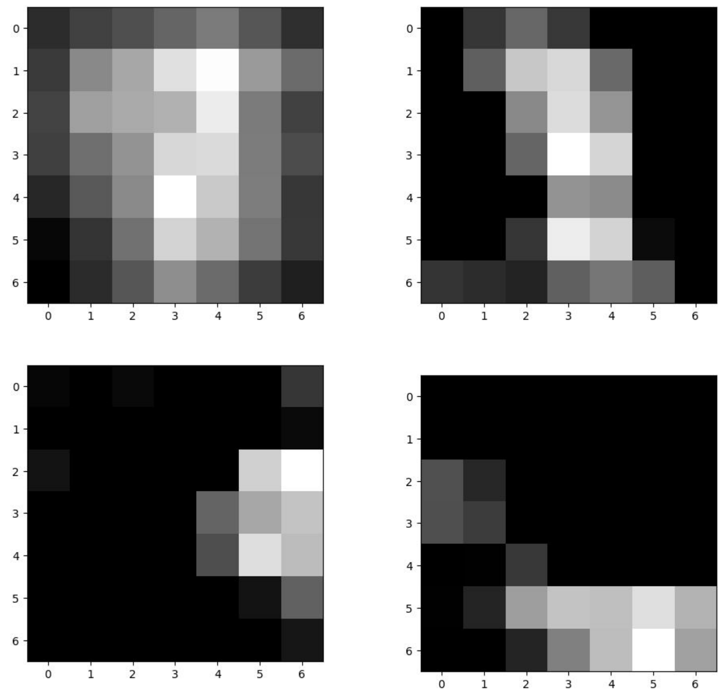
```
cam_resized = np.array(Image.fromarray(cam).resize(img.size, resample=Image.BILINEAR))
plt.imshow(cam_resized, cmap="jet");
```





Extracting the activations with a hook

```
# Hook for extracting the activations from the last convolutional layer
activations = []
def hook_fn(module, input, output):
    activations.append(output)

# Register the hook
layer_name = 'layer4'  # Last convolutional block
hook = model._modules.get(layer_name).register_forward_hook(hook_fn)

# Forward pass
output = model(input_tensor)

# Remove the hook
hook.remove()

# Get the weights of the fully connected layer
fc_weights = model.fc.weight.detach()

# Select the class index (e.g., 0 for 'tench')
class_idx = torch.argmax(output, dim= 1).item()
```

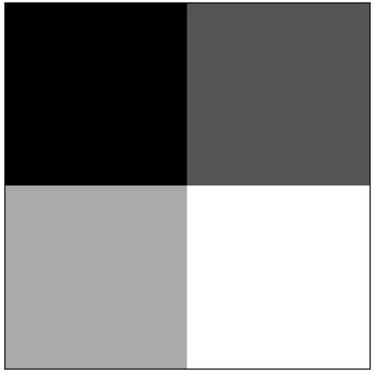


Bilinear interpolation

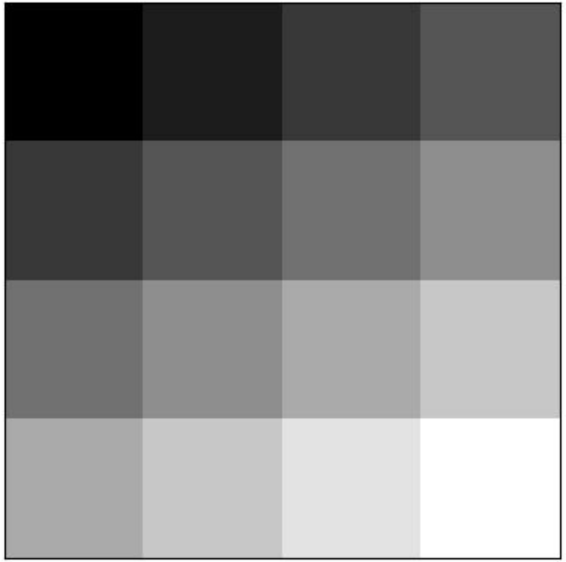
$$w_x = \frac{x - x_1}{x_2 - x_1}$$

$$w_y = \frac{y - y_1}{y_2 - y_1}$$

input



output



$(x_1, y_1) = (0, 0)$ coordinates of Q_{11}

$(x_2, y_2) = (1, 1)$ coordinates of Q_{22}

$(x, y) = (0.6, 0.7)$ target point

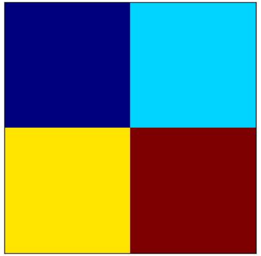
$$Q = \begin{bmatrix} 10 & 20 \\ 30 & 40 \end{bmatrix}$$

$$f(x, y) = \begin{bmatrix} 1 - w_x & w_x \end{bmatrix} \begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix} \begin{bmatrix} 1 - w_y \\ w_y \end{bmatrix}$$

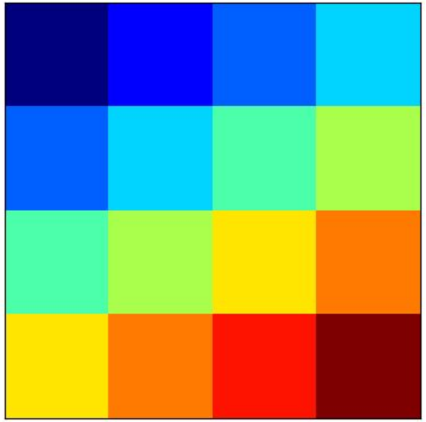
Bilinear interpolation in PyTorch

```python
import torch
import torch.nn.functional as F

# Create a 1x1x2x2 tensor (batch_size x channels x height x width)
# Notice that the batch size and channel dimensions are created by wrapping
# the height and width tensor with two pairs of extra square brackets
input = torch.tensor([[[[10, 20],
                        [30, 40]]]],
                     dtype=torch.float32)

# Upscale to 4x4
output = F.interpolate(input, size=(4, 4), mode='bilinear',
align_corners=True)

import matplotlib.pyplot as plt
# The colormap is just for illustration of corner alignment
plt.imshow(input.squeeze(), cmap="jet")
plt.imshow(output.squeeze(), cmap="jet")
```

input (with colormap)
shape: 2x2



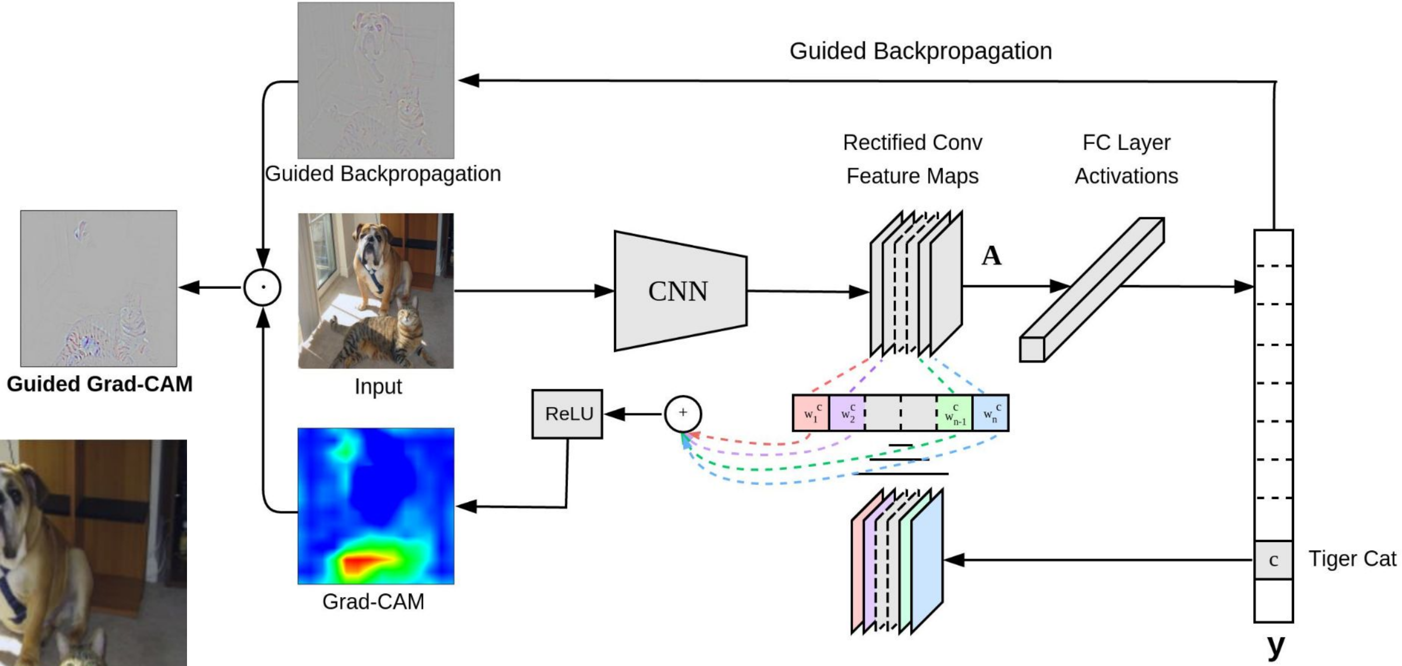
output (with colormap)
shape: 4x4



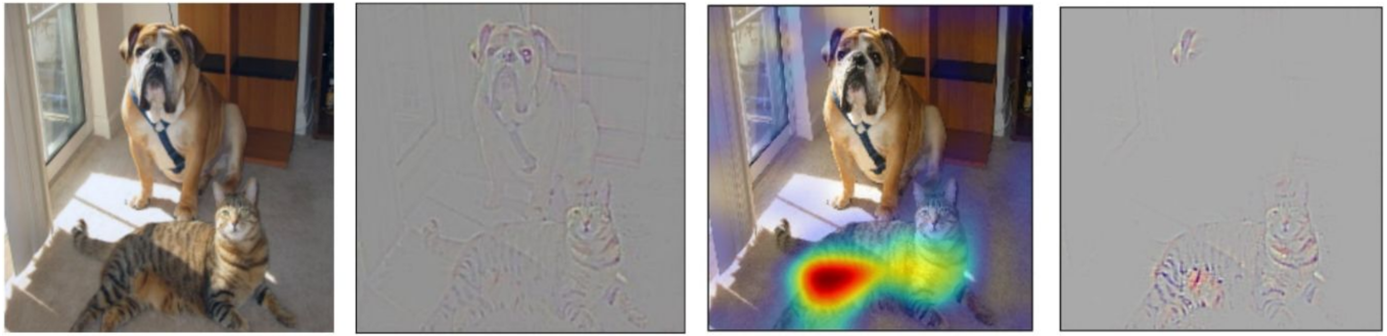
Overlaying the resized CAM on the image

```
# Compute CAM
# fc_weights[class_idx] is 1-dimensional, but the einsum equation 'ij' expects 2 dimensions.
# We need to unsqueeze to add a dimension to fc_weights[class_idx]
# The einsum operation here is a one  liner that is equivalent to Global Average Pooling
cam = torch.einsum( 'ij,jkl->ikl' , fc_weights[class_idx].unsqueeze( 0), activations[ 0][0])

# Normalize CAM
cam = cam - cam. min()
cam = cam / cam. max()

# Resize CAM to match the input image
cam = cam.detach().numpy()
# Squeeze the cam array to remove the first dimension and convert to uint8
cam = cam.squeeze()  # remove the first dimension
cam = (cam *  255).astype(np.uint8)   # scale to 0-255 and convert to uint8
cam_resized = np.array(Image.fromarray(cam).resize(img.size, resample=Image.BICUBIC))

# Overlay CAM on the image
plt.imshow(img)
plt.imshow(cam_resized, cmap= 'jet', alpha=0.5)
plt.axis('off')
plt.show()
```

Guided Grad-CAM



Combining Grad-CAM and Guided Backprop



(a) Original Image (b) Guided Backprop 'Cat' (c) Grad-CAM 'Cat' (d)Guided Grad-CAM 'Cat'

(g) Original Image (h) Guided Backprop 'Dog' (i) Grad-CAM 'Dog' (j)Guided Grad-CAM 'Dog'

CAM vs GradCAM

- **CAM needs a Global Average Pooling layer to be added to the model, GradCAM works with any architecture <u>without changes</u>**

- **GradCAM can visualize outputs of any layer, CAM is limited to the final layer**

- **CAM runs faster and requires less memory**

Berlin 'lioness': Wild animal probably a boar, authorities say

21 July 2023

Share Save

Kathryn Armstrong
BBC News

Michael Grubert, mayor of Kleinmachnow, said the spotted animal on the loose was most likely a boar

Image from BBC News

Lion or boar?

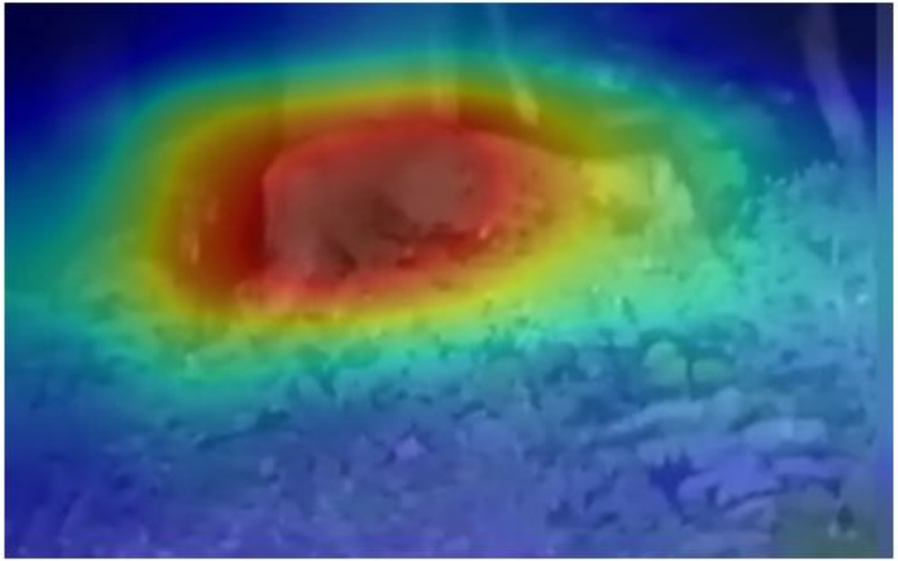


Image from Berlin 'lioness' on loose 'is a wild boar’ (BBC News)

Summary

Class Activation Mapping (CAM) is a tool for explainability

- CAM helps us understand whether our decisions are well supported or based on spurious correlations
- CAM exposes hidden connections between inputs and decisions that affect model reliability and safety
- Grad-CAM allows us to extend the method to any network without changes to the architecture

CAM implementations steps

- Extract feature maps with hooks from final convolutional layer
- Project class weights onto activation maps
- Upsample and overlay heatmaps on input images

References

Learning Deep Features for Discriminative Localization

- **https://openaccess.thecvf.com/content_cvpr_2016/html/Zhou_Learning_Deep_Features_CVPR_2016_paper.html**

Class Activation Mapping explained

- https://github.com/fastai/fastbook/blob/master/18_CAM.ipynb

Basic guide to Numpy's einsum

- https://ajcr.net/Basic-guide-to-einsum/

Class activation mapping on fiftyone

- https://voxel51.com/blog/exploring-gradcam-and-more-with-fiftyone/

Resources

- Github Repository
- YouTube playlist
- Discord channel
  #practical-computer-vision-workshops