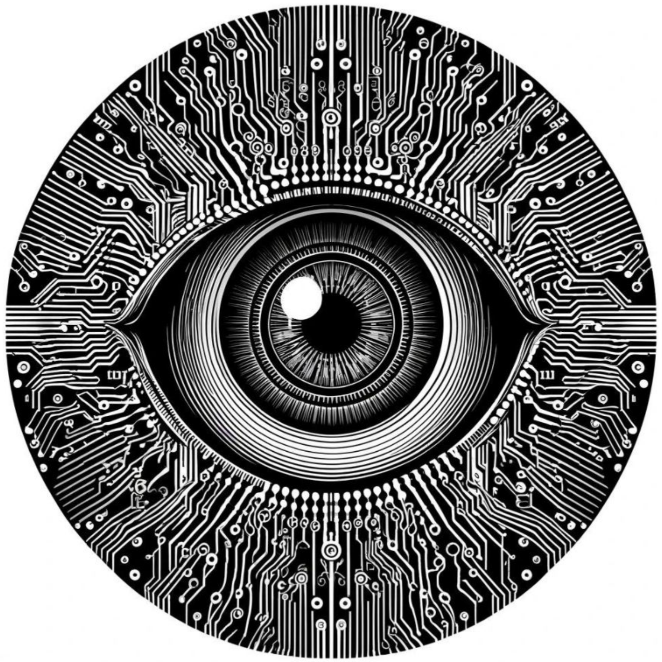
Interpretability with Class Activation Mapping



Antonio Rueda-Toicen

SPONSORED BY THE
Federal Ministry
of Education
and Research

KI Service Zentrum
by Hasso-Plattner-Institut

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

Class Activation Mapping

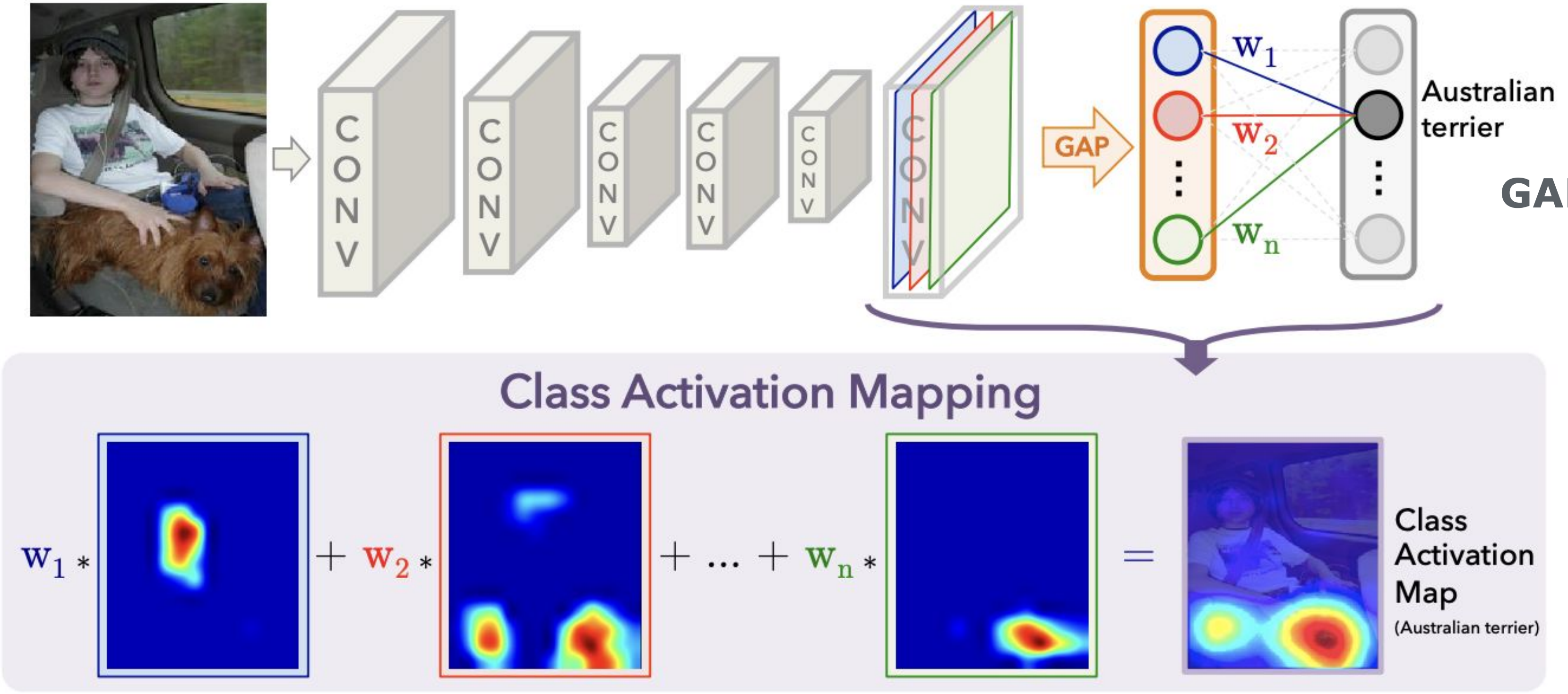


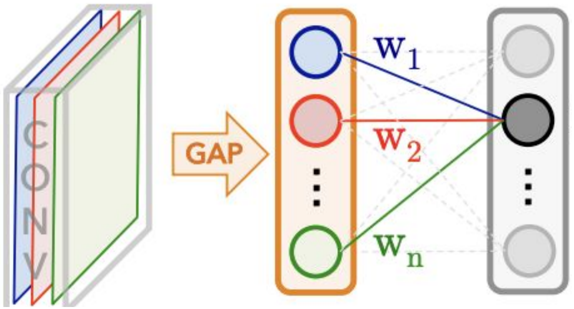
Figure 2. Class Activation Mapping: the predicted class score is mapped back to the previous convolutional layer to generate the class activation maps (CAMs). The CAM highlights the class-specific discriminative regions.

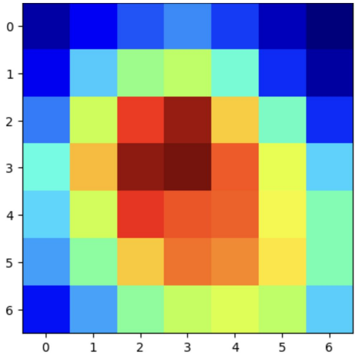
GAP = Global Average Pooling

Image from [Learning Deep Features for Discriminative Localization]

CAM with global average pooling

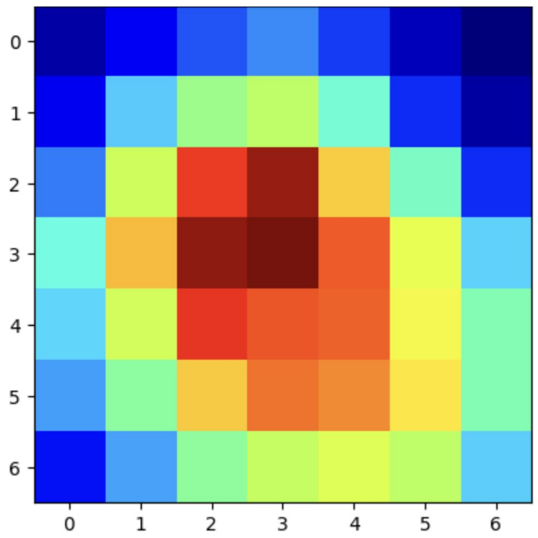
```python
import torch
import torch.nn.functional as F

# 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
```

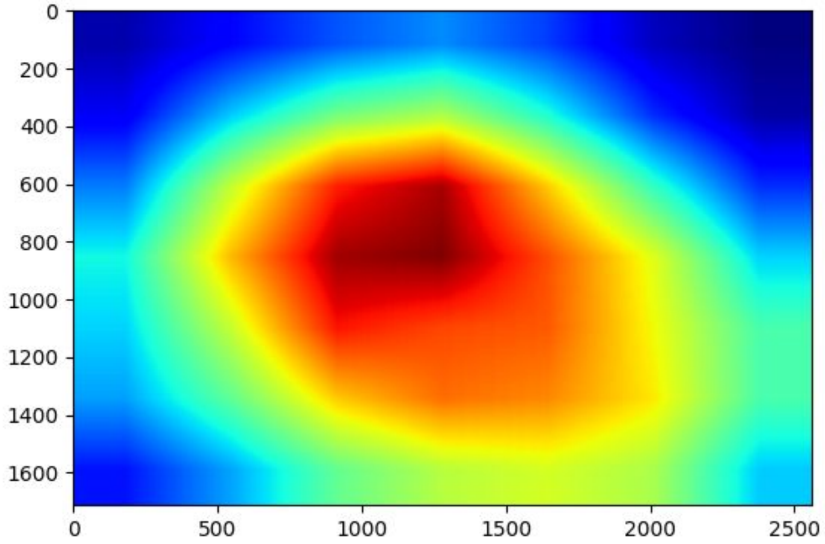




Resizing the map

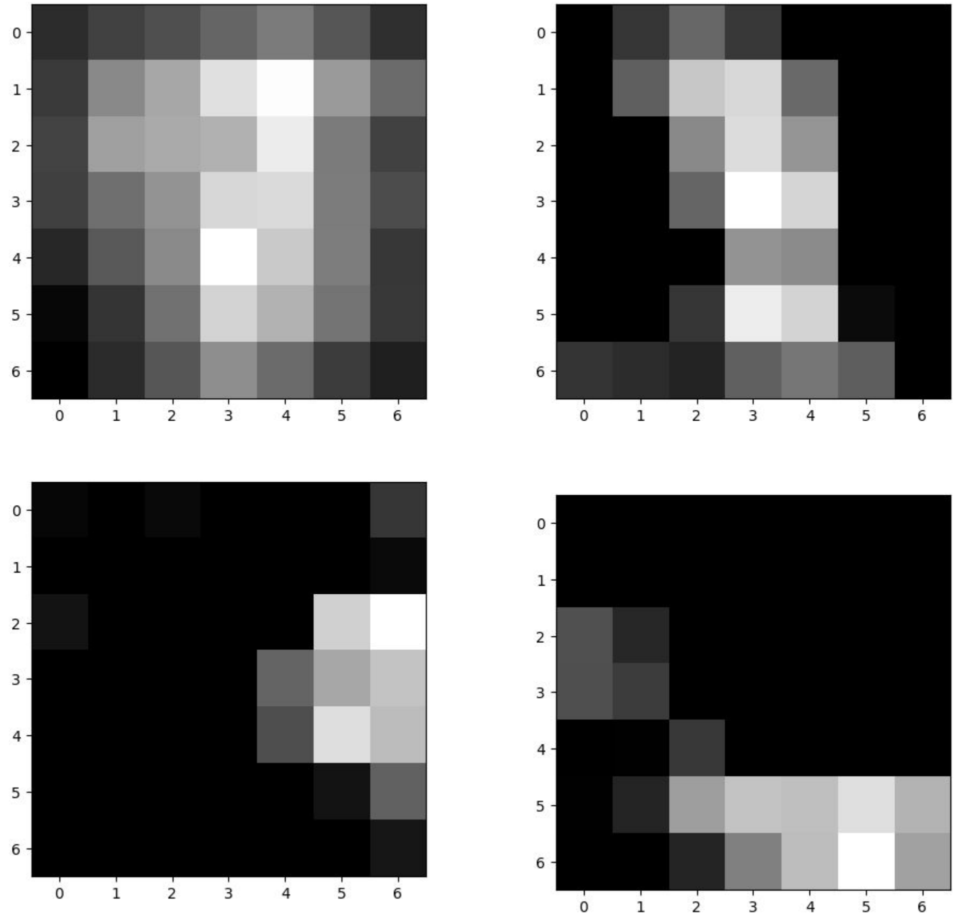
```
cam_resized = np.array(Image.fromarray(cam).resize(img.size, resample=Image.BILINEAR))
plt.imshow(cam_resized, cmap="jet");
```





Extracting the activations with a hook

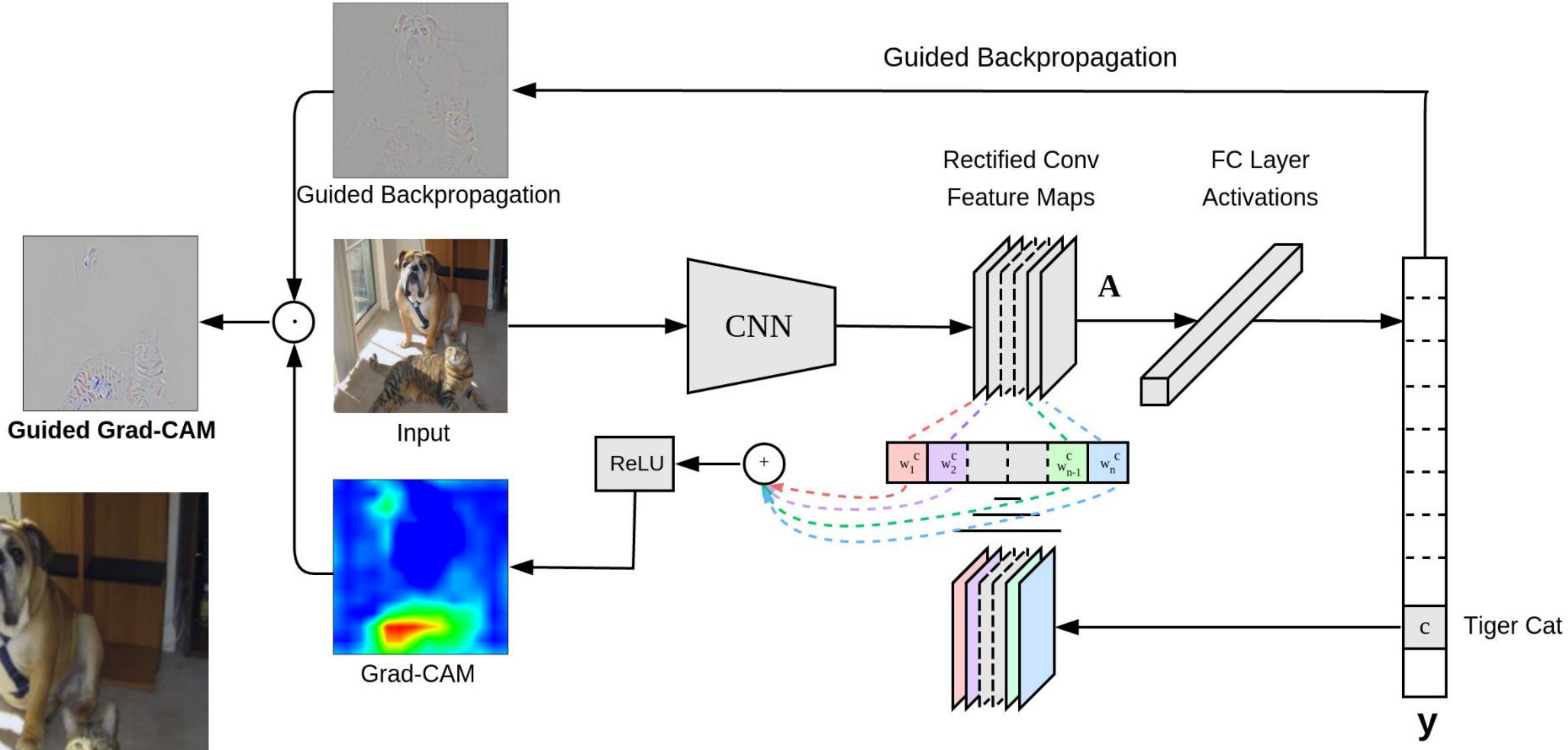
```
# Hook for extracting the activations from the last convolutional layer
activations = []
def hook_fn(module, input, output):
    activations.append(output)

# 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()
```



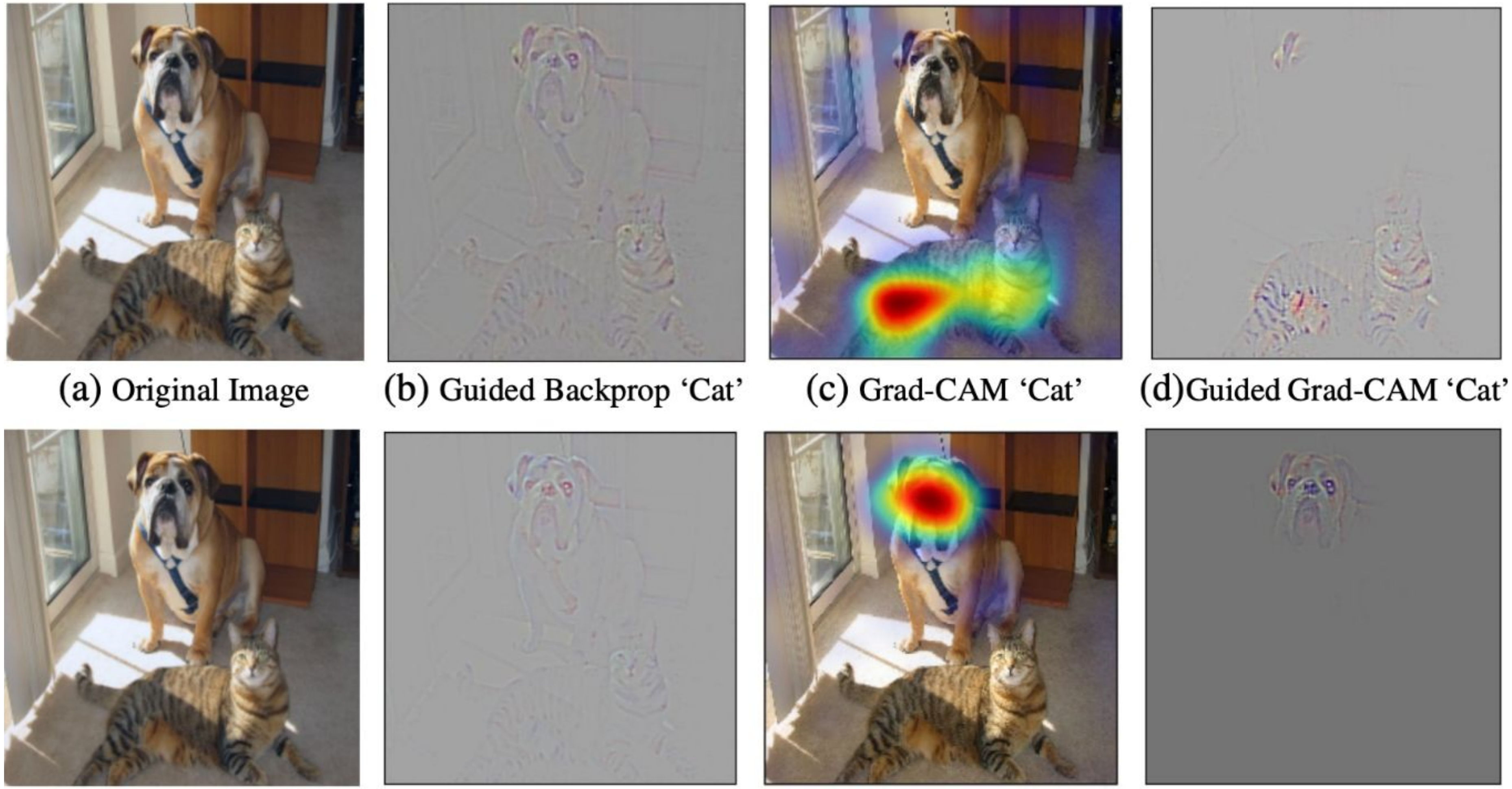
Overlaying the resized CAM on the image

```
# 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]
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?

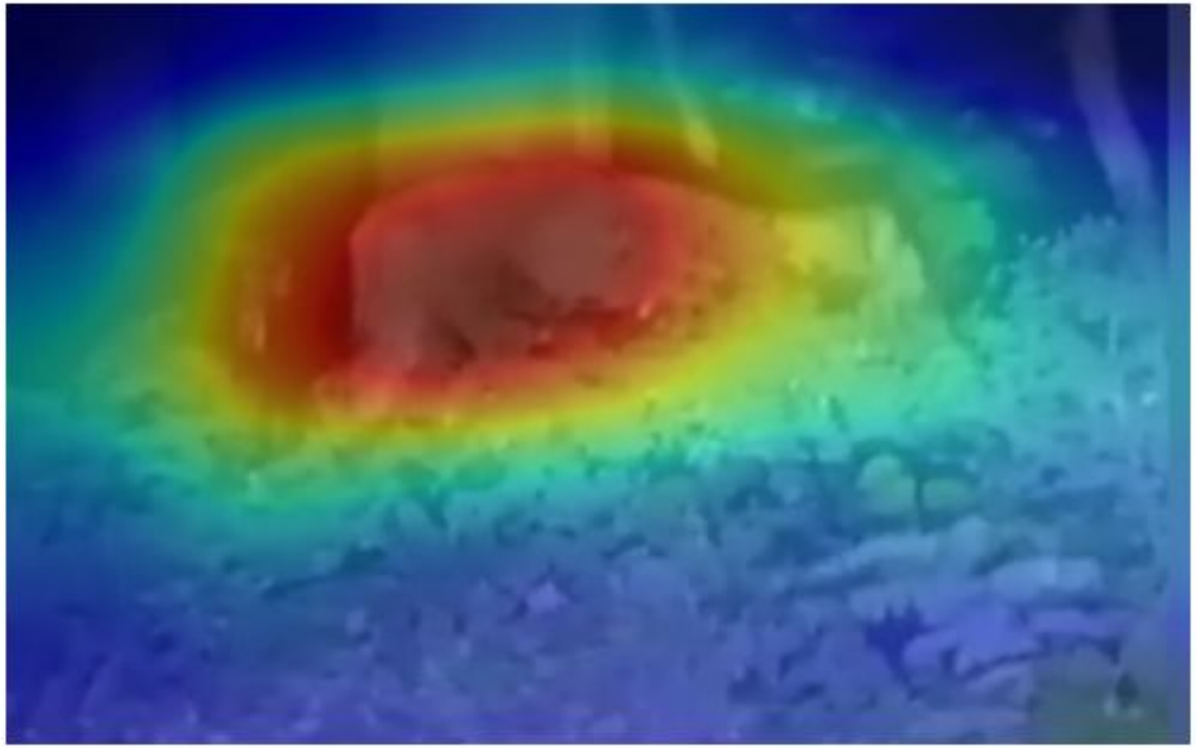


Image from Berlin 'lioness' on loose 'is a wild boar' (BBC News)

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

SPONSORED BY THE

Federal Ministry
of Education
and Research

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/

SPONSORED BY THE

Federal Ministry
of Education
and Research