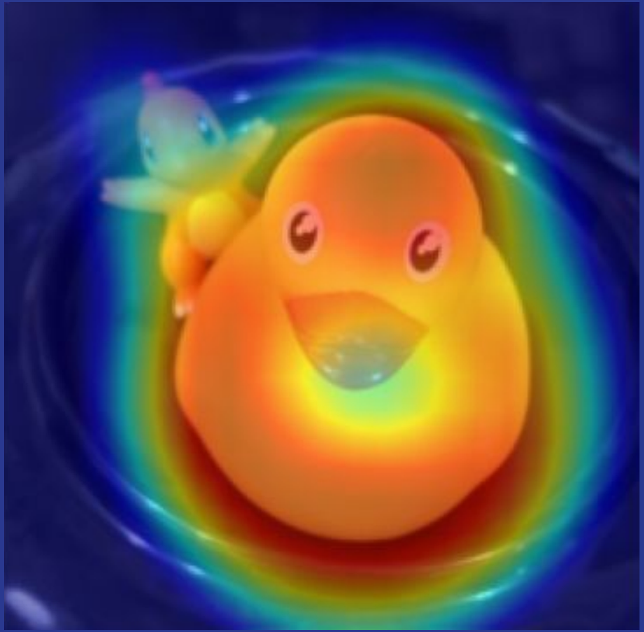
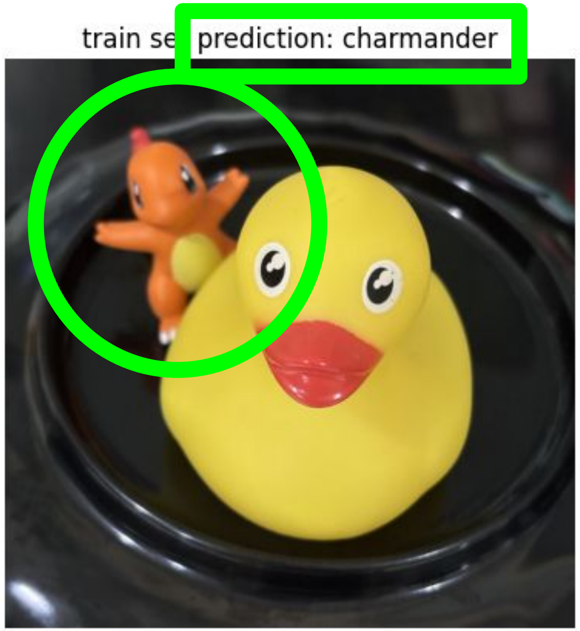
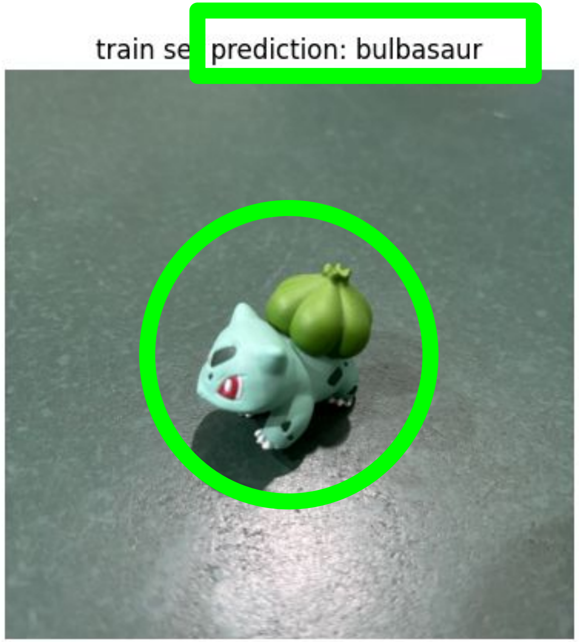
GradCAM in PyTorch

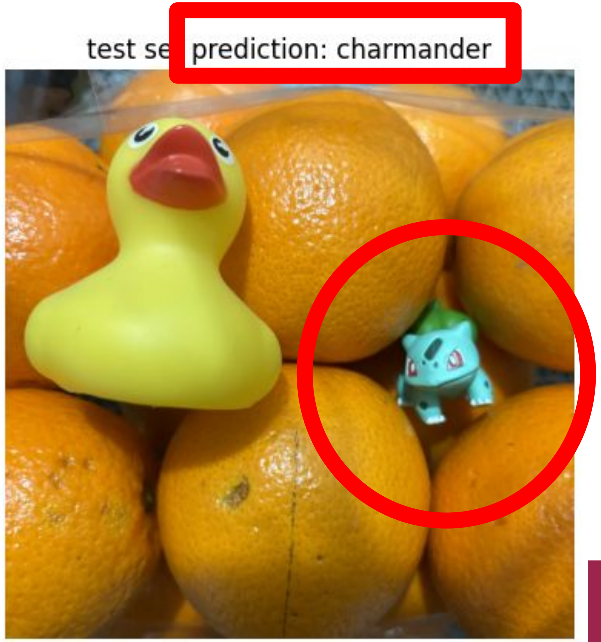
400067-Prut



Why did our model make those predictions?

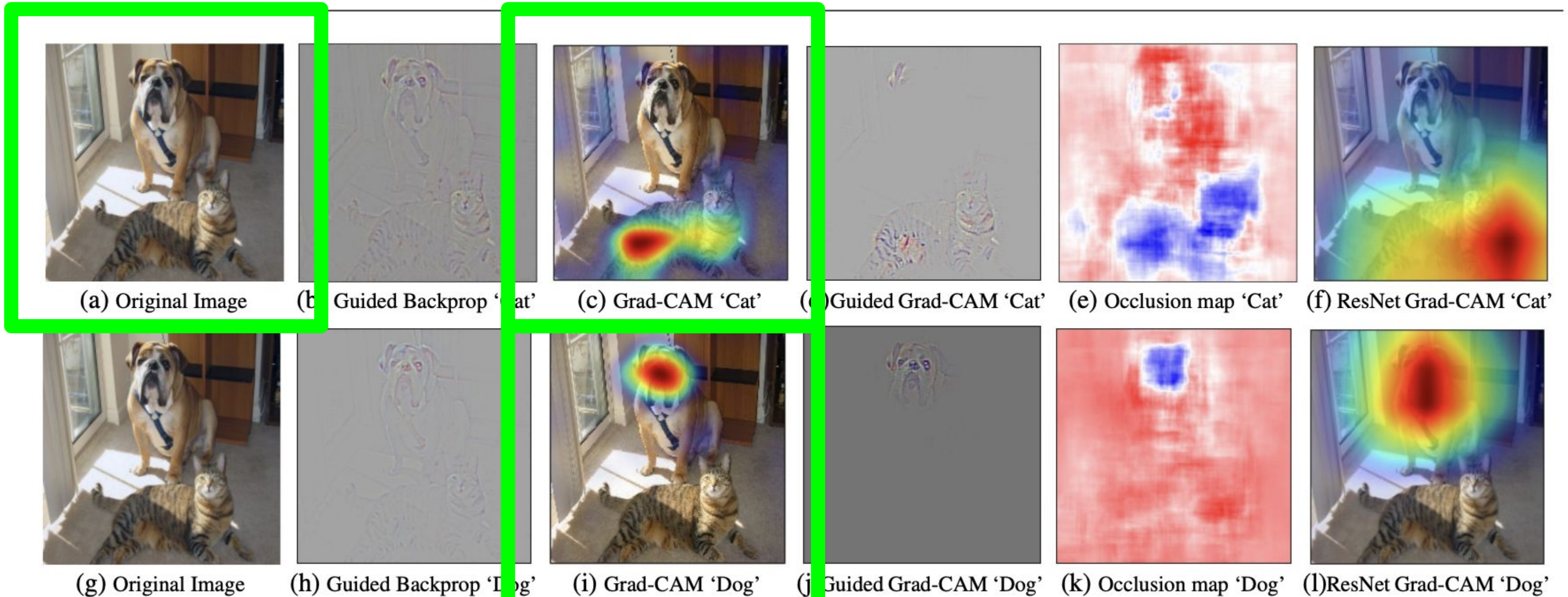






Gradient-weighted Class Activation Mapping

Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization 3

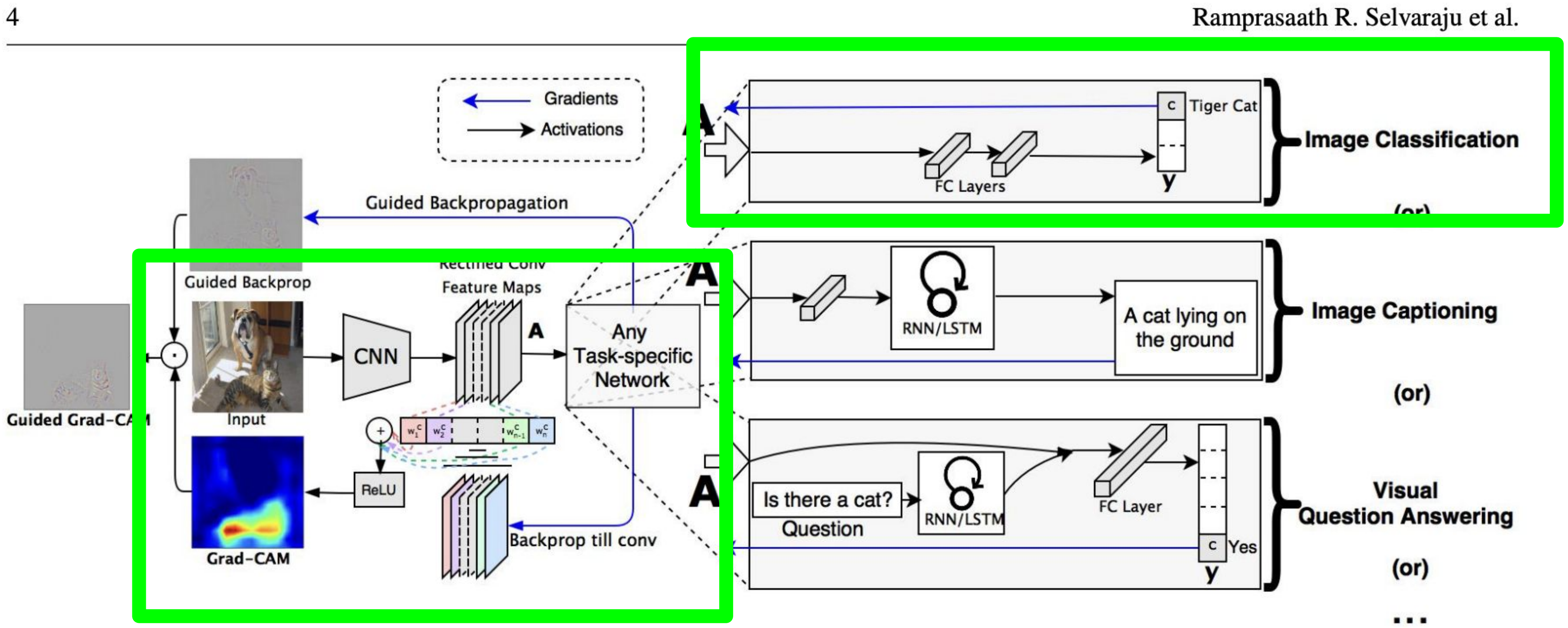


(a) Original Image (b) Guided Backprop 'Cat' (c) Grad-CAM 'Cat' (d) Guided Grad-CAM 'Cat' (e) Occlusion map 'Cat' (f) ResNet Grad-CAM 'Cat'

(g) Original Image (h) Guided Backprop 'Dog' (i) Grad-CAM 'Dog' (j) Guided Grad-CAM 'Dog' (k) Occlusion map 'Dog' (l) ResNet Grad-CAM 'Dog'

Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., & Batra, D. (2019). Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization. Retrieved from https://arxiv.org/abs/1610.02391

GradCAM vs CAM



Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., & Batra, D. (2019). Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization. Retrieved from https://arxiv.org/abs/1610.02391

```
!pip install grad-cam
```

```python
from pytorch_grad_cam import GradCAM
```

https://github.com/jacobgil/pytorch-grad-cam

Code

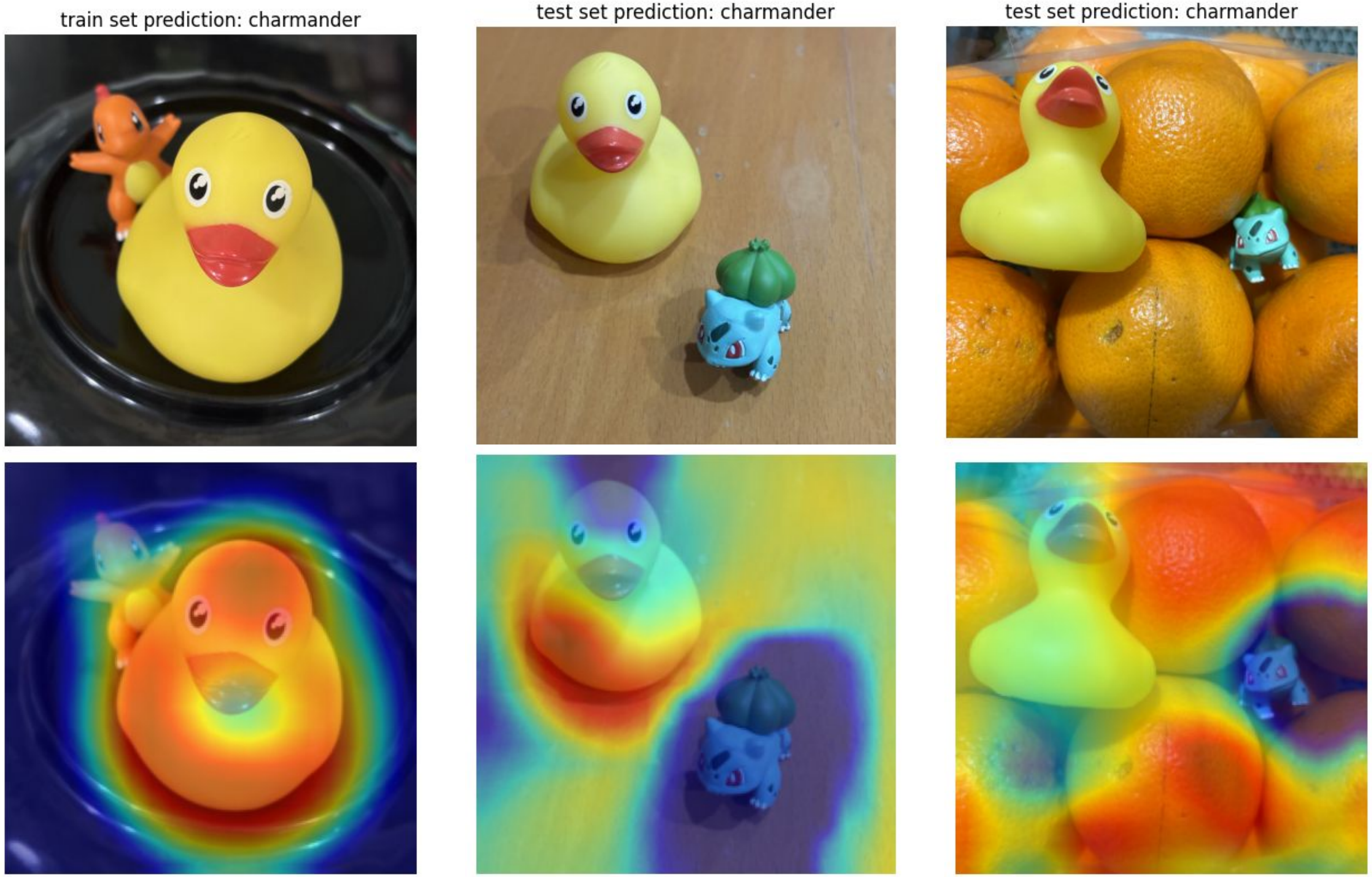
https://www.kaggle.com/code/prutsaowaprut/gradcam-tutorial

Dataset

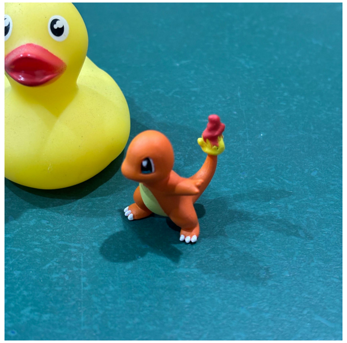
https://www.kaggle.com/datasets/prutsaowaprut/charmander-or-bulbasaur-duck-invasion/data

```
class CNNModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 32, 5)
        self.conv2 = nn.Conv2d(32, 64, 5)
        self.conv3 = nn.Conv2d(64, 128, 3)
        self.conv4 = nn.Conv2d(128, 256, 5)
        self.fc1 = nn.Linear(256, 2)
        self.pool = nn.MaxPool2d(2, 2)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.pool(F.relu(self.conv3(x)))
        x = self.pool(F.relu(self.conv4(x)))
        x = F.adaptive_avg_pool2d(x, 1).reshape(x.shape[0], -1)
        x = self.fc1(x)
        return x
```

```python
from pytorch_grad_cam import GradCAM
from pytorch_grad_cam.utils.model_targets import ClassifierOutputTarget
from pytorch_grad_cam.utils.image import show_cam_on_image


cam = GradCAM(model=model, target_layers=[model.conv4])

targets = [ClassifierOutputTarget(pred_class)]

grayscale_cam = cam(input_tensor=input_tensor.unsqueeze(0), targets=targets, aug_smooth=True)

grayscale_cam = grayscale_cam[0, :]

visualization = show_cam_on_image(torch_to_numpy(input_tensor), grayscale_cam, use_rgb=True)
```


train set prediction: charmander
test set prediction: charmander
test set prediction: charmander

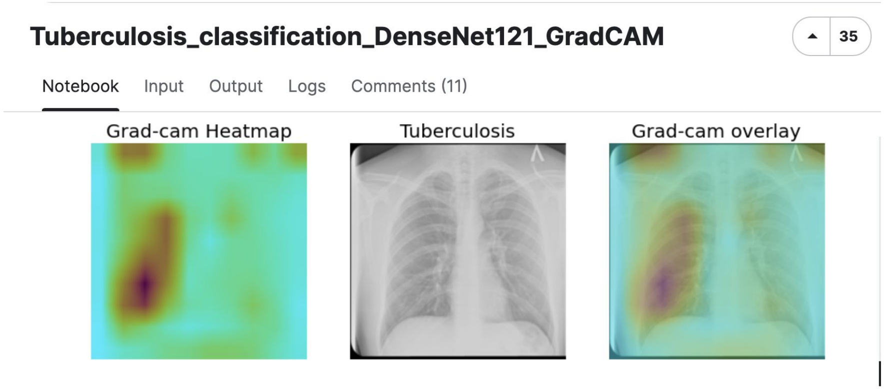
Training set is biased!



https://www.kaggle.com/competitions/rsna-breast-cancer-detection/discussion/372186

Applications

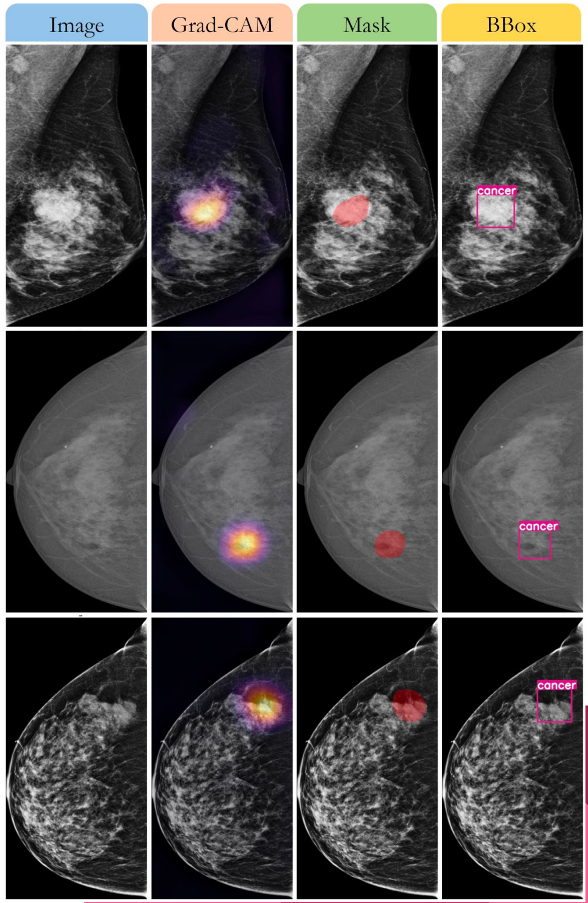
- Object detection
- Image segmentation

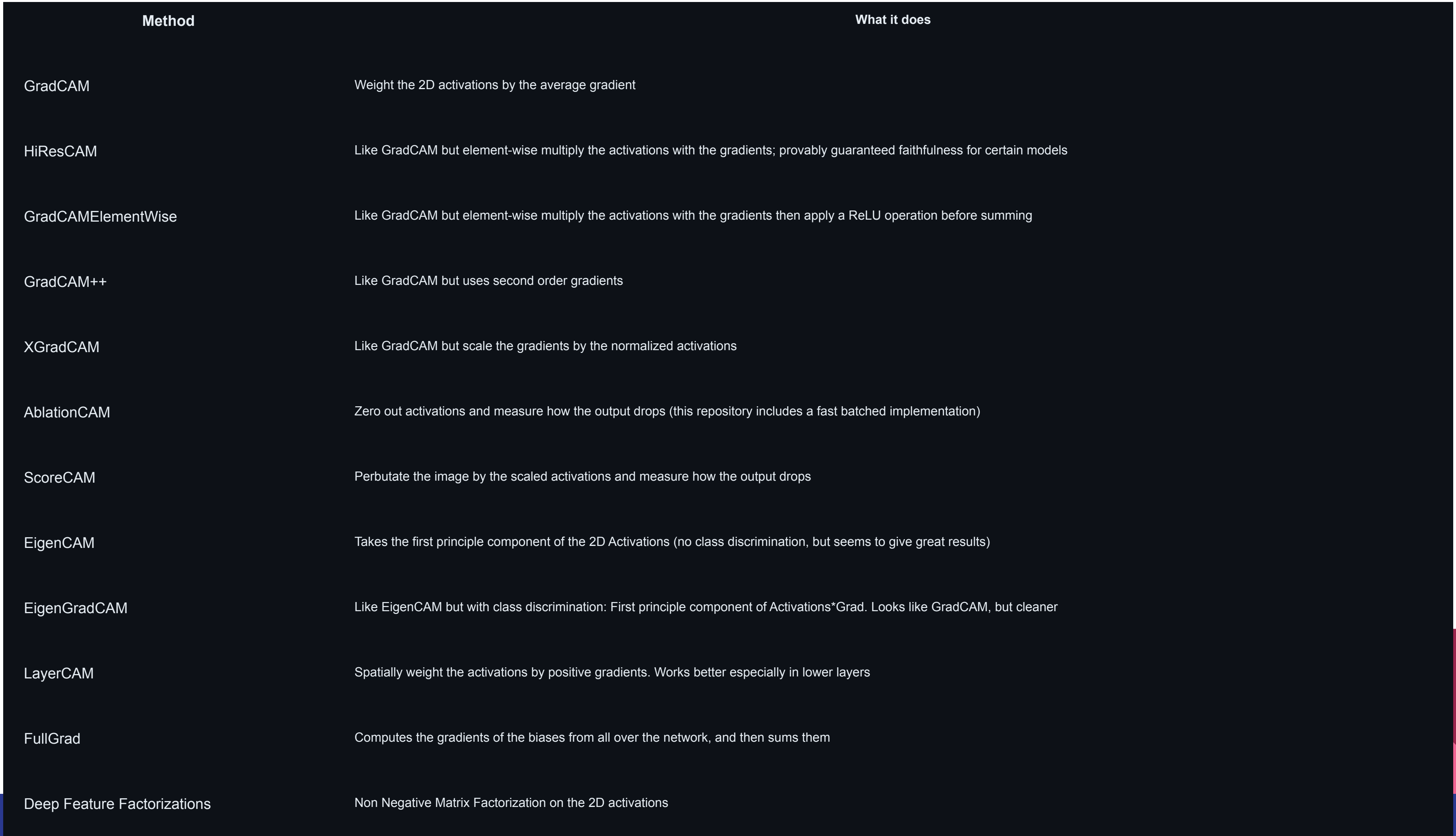

Tuberculosis_classification_DenseNet121_GradCAM

▲ 35

Notebook Input Output Logs Comments (11)

https://www.kaggle.com/code/sanphats/tuberculosis-classification-densenet121-gradcam#Grad-cam-evaluation





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 by the normalized activations
AblationCAM	Zero out activations and measure how the output drops (this repository includes a fast batched implementation)
ScoreCAM	Perbutate the image by the scaled activations and measure how the output drops
EigenCAM	Takes the first principle component of the 2D Activations (no class discrimination, but seems to give great results)
EigenGradCAM	Like EigenCAM but with class discrimination: First principle component of Activations*Grad. Looks like GradCAM, but cleaner
LayerCAM	Spatially weight the activations by positive gradients. Works better especially in lower layers
FullGrad	Computes the gradients of the biases from all over the network, and then sums them
Deep Feature Factorizations	Non Negative Matrix Factorization on the 2D activations

Other Explainable AI (XAI) Methods

- LIME
- SHAP

References

Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., & Batra, D. (2019). Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization. Retrieved from https://arxiv.org/abs/1610.02391

https://www.analyticsvidhya.com/blog/2023/12/grad-cam-in-deep-learning/

https://github.com/jacobgil/pytorch-grad-cam

https://pytorch.org/vision/stable/index.html

https://debuggercafe.com/pytorch-imagefolder-for-training-cnn-models/

https://www.kaggle.com/code/sanphats/tuberculosis-classification-densenet121-gradcam#Grad-cam-evaluation

https://www.kaggle.com/competitions/rsna-breast-cancer-detection/discuss[illegible]n/372186