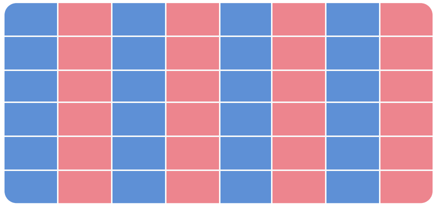
Gradient-based Methods: Computing Feature Sensitivity

📐 Vanilla Gradients



$\partial y/\partial x$

Basic sensitivity, noisy

∫ Integrated Gradients

$\int_0^1 \nabla f(x' + \alpha(x-x'))\ d\alpha$

Path integral, smooth

〰️ SmoothGrad

$E[\nabla f(x + N(0,\sigma^2))]$

Averaged over noise

🔥 Grad-CAM

$\mathrm{ReLU}(\Sigma\ \alpha_k\ A_k)$

Class activation map

Method Comparison

Vanilla	Fast, noisy ✗
Integrated	Smooth, slow ✓
SmoothGrad	Less noise ✓
Grad-CAM	CNN specific ✓

Strengths

✓ Model-agnostic
✓ Fast computation
✓ Differentiable

Limitations

✗ Saturation issue
✗ Noise in vanilla
✗ Requires gradients

PyTorch Implementation

```
Vanilla Gradients
x.requires_grad_()
output = model(x)
output.backward()
gradients = x.grad
```

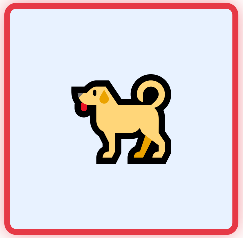
```
Integrated Gradients
# Path from baseline to input
alphas = torch.linspace(0, 1, steps=50)
for alpha in alphas:
  x_step = baseline + alpha * (x - baseline)
  # Compute gradients at each step
```

```
SmoothGrad
# Average over noisy samples
for _ in range(n_samples):
```

```
  noise = torch.randn_like(x) * sigma
  grads += compute_grad(x + noise)
smooth_grad = grads / n_samples
```

🔍 Interactive CAM Visualization Demo

📷 Select Image



🖌 Method Selection

- **Grad-CAM**
- Integrated Gradients
- SmoothGrad
- Vanilla Gradients

🎨 Heatmap Settings

Opacity 60%

Focus Intensity 70%

CAM Visualization on Image

High → Low Activation