Visualization vs. Interpretability

1. “CNN” Model Summary

June 11th, 2025

Lab 1. CAM (Self-defined, sequential)

Dense units = 10

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 28, 16)	160
max_pooling2d (MaxPooling2D)	(None, 14, 14, 16)	0
conv2d_1 (Conv2D)	(None, 14, 14, 32)	4,640
max_pooling2d_1 (MaxPooling2D)	(None, 7, 7, 32)	0
conv2d_2 (Conv2D)	(None, 7, 7, 64)	18,496
max_pooling2d_2 (MaxPooling2D)	(None, 3, 3, 64)	0
conv2d_3 (Conv2D)	(None, 3, 3, 128)	73,856
global_average_pooling2d (GlobalAveragePooling2D)	(None, 128)	0
dense (Dense)	(None, 10)	1,290

Total params: 98,442 (384.54 KB)
Trainable params: 98,442 (384.54 KB)
Non-trainable params: 0 (0.00 B)

Model: "functional_9"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 28, 28, 1)	0	–
conv2d (Conv2D)	(None, 28, 28, 16)	160	input_layer[0][0]… input_layer[0][0]
max_pooling2d (MaxPooling2D)	(None, 14, 14, 16)	0	conv2d[6][0], conv2d[8][0]
conv2d_1 (Conv2D)	(None, 14, 14, 32)	4,640	max_pooling2d[5]… max_pooling2d[7]…
max_pooling2d_1 (MaxPooling2D)	(None, 7, 7, 32)	0	conv2d_1[4][0], conv2d_1[6][0]
conv2d_2 (Conv2D)	(None, 7, 7, 64)	18,496	max_pooling2d_1[… max_pooling2d_1[…
max_pooling2d_2 (MaxPooling2D)	(None, 3, 3, 64)	0	conv2d_2[2][0], conv2d_2[4][0]
conv2d_3 (Conv2D)	(None, 3, 3, 128)	73,856	max_pooling2d_2[… max_pooling2d_2[…
global_average_poo… (GlobalAveragePool…	(None, 128)	0	conv2d_3[2][0]
dense (Dense)	(None, 10)	1,290	global_average_p…

Total params: 98,442 (384.54 KB)
Trainable params: 98,442 (384.54 KB)
Non-trainable params: 0 (0.00 B)

Loss I: sparse_categorical_crossentropy

Label format: Integer (sparse)

1D array of shape (batch_size,)

Lab 2. CAM (Self-defined, sequential)

Dense units = 1

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 300, 300, 16)	448
max_pooling2d (MaxPooling2D)	(None, 150, 150, 16)	0
conv2d_1 (Conv2D)	(None, 150, 150, 32)	4,640
max_pooling2d_1 (MaxPooling2D)	(None, 75, 75, 32)	0
conv2d_2 (Conv2D)	(None, 75, 75, 64)	18,496
max_pooling2d_2 (MaxPooling2D)	(None, 37, 37, 64)	0
conv2d_3 (Conv2D)	(None, 37, 37, 128)	73,856
global_average_pooling2d (GlobalAveragePooling2D)	(None, 128)	0
dense (Dense)	(None, 1)	129

Total params: 97,569 (381.13 KB)
Trainable params: 97,569 (381.13 KB)
Non-trainable params: 0 (0.00 B)

Model: "functional_9"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 300, 300, 3)	0	–
conv2d (Conv2D)	(None, 300, 300, 16)	448	input_layer[0][0… input_layer[0][0]
max_pooling2d (MaxPooling2D)	(None, 150, 150, 16)	0	conv2d[6][0], conv2d[8][0]
conv2d_1 (Conv2D)	(None, 150, 150, 32)	4,640	max_pooling2d[5]… max_pooling2d[7]…
max_pooling2d_1 (MaxPooling2D)	(None, 75, 75, 32)	0	conv2d_1[4][0], conv2d_1[6][0]
conv2d_2 (Conv2D)	(None, 75, 75, 64)	18,496	max_pooling2d_1[… max_pooling2d_1[…
max_pooling2d_2 (MaxPooling2D)	(None, 37, 37, 64)	0	conv2d_2[2][0], conv2d_2[4][0]
conv2d_3 (Conv2D)	(None, 37, 37, 128)	73,856	max_pooling2d_2[… max_pooling2d_2[…
global_average_poo… (GlobalAveragePool…	(None, 128)	0	conv2d_3[2][0]
dense (Dense)	(None, 1)	129	global_average_p…

Total params: 97,569 (381.13 KB)
Trainable params: 97,569 (381.13 KB)
Non-trainable params: 0 (0.00 B)

Loss III: **binary_crossentropy**

Label format: multi-hot encoded

2D array of shape (batch_size, num_classes)

Lab 3. Saliency (Self-defined, sequential)

Dense units = 2

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 300, 300, 16)	448
max_pooling2d (MaxPooling2D)	(None, 150, 150, 16)	0
conv2d_1 (Conv2D)	(None, 150, 150, 32)	4,640
max_pooling2d_1 (MaxPooling2D)	(None, 75, 75, 32)	0
conv2d_2 (Conv2D)	(None, 75, 75, 64)	18,496
max_pooling2d_2 (MaxPooling2D)	(None, 37, 37, 64)	0
conv2d_3 (Conv2D)	(None, 37, 37, 128)	73,856
global_average_pooling2d (GlobalAveragePooling2D)	(None, 128)	0
dense (Dense)	(None, 2)	258

Total params: 97,698 (381.63 KB)
Trainable params: 97,698 (381.63 KB)
Non-trainable params: 0 (0.00 B)

Assignment use this model too~

Loss II:
categorical_crossentropy

Label format: One-hot encoded

2D array of shape (batch_size, num_classes)

Lab 4. GradCAM (VGG16)

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1,792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36,928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73,856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147,584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295,168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590,080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590,080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
global_average_pooling2d (GlobalAveragePooling2D)	(None, 512)	0
dense (Dense)	(None, 2)	1,026

Total params: 14,715,714 (56.14 MB)
Trainable params: 7,080,450 (27.01 MB)
Non-trainable params: 7,635,264 (29.13 MB)

Loss I: sparse_categorical_crossentropy

Label format: Integer (sparse)

1D array of shape (batch_size,)

Visualization vs. Interpretability

2. Topics. CAM vs. Grad-CAM

June 11th, 2025

Grad-CAM

Grad-CAM (Gradient-weighted Class Activation Mapping) is a popular **visual explanation technique** for interpreting the decisions of **Convolutional Neural Networks (CNNs)**, especially in **image classification and computer vision**tasks.

Q1. What does Grad-CAM do?

- Grad-CAM helps you visualize **which regions of an input image** were most influential in a CNN's decision for a particular class.
- It produces a **heatmap** over the image, highlighting the "important" areas the network used to make its prediction.

Example Use Case

Suppose a CNN predicts "dog" for an image. Grad-CAM can show you **which parts of the image (e.g., dog's face or tail)** led to that prediction.

Grad-CAM

Q2. How does Grad-CAM work?

1. **Forward Pass**: Run the input image through the CNN to get the prediction.

2. **Backward Pass**: Compute the gradient of the score for the target class (e.g., "cat") with respect to the **feature maps** in the last convolutional layer.

3. **Weight Computation**: These gradients are global-average-pooled to obtain **importance weights** for each feature map.

4. **Weighted Sum**: Multiply each feature map by its corresponding weight and sum them to get a **class-discriminative heatmap**.

5. **ReLU**: Apply ReLU to focus only on the features that positively influence the class of interest.

Q3. CAM vs. Grad-CAM

🔁 Summary Table: CAM vs Grad-CAM

Feature	CAM (Class Activation Mapping)	Grad-CAM (Gradient-weighted CAM
Introduced in	2016	2017
Requires model modification?	✅ Yes – requires a special architecture	❌ No – works with most CNNs as-is
Architecture required	Global Average Pooling (GAP) before softmax	Any CNN with convolutional layers
How it works	Uses the weights of the final FC layer over feature maps	Uses gradients of class score w.r.t. feature maps
Flexibility	Limited – only works with specific CNN architectures	Flexible – works with pre-trained networks like ResNet, VGG
Uses gradients?	❌ No	✅ Yes
Layer used	Last convolutional layer before GAP	Any convolutional layer (usually last)
Interpretability	Good (but less flexible)	Better and more general

Q4. Conceptual Differences

CAM

- Requires modifying the model so that the final feature maps go directly to a **Global Average Pooling** layer, then to softmax.
- CAM-friendly networks, only
- The class activation map is computed directly using the **learned weights** from the classification layer.
- **Analogy**: **CAM** is like using the model's built-in roadmap.

Grad-CAM

- Works **without modifying the architecture**.
- Uses the **gradient of the class score** with respect to the feature maps to compute importance.
- More **general-purpose**, and works with most modern CNN architectures out of the box.
- **Analogy: Grad-CAM** “If you want more of this class, where should you look in the image?” — and it answers based on gradients.

Q5. When to Use Which?

- Use **CAM** if you're designing a model from scratch and can control the architecture.
- Use **Grad-CAM** if you're working with existing pre-trained models like **ResNet**, **VGG**, or **Inception**, and need **explainability without retraining**.

Visualization vs. Interpretability

3. Review. Label format vs. Last Dense Layer # of Units
Label format vs. Loss

June 11th, 2025

Loss - 1. sparse_categorical_crossentropy

- You have **single-label** classification
- Labels are given as **integer class indices** (not one-hot)
- Predicted probabilities: `[0.1, 0.2. 0.7]`

$$\text{loss} = -\log(p_{\text{true class}})$$

Loss - 2. categorical_crossentropy

- You have **single-label** classification
- Labels are **one-hot encoded**
- One-hot label: `[0, 0, 1]`
- Predicted probabilities: `[0.1, 0.2, 0.7]`

$$\text{loss} = -\sum_{i=1}^{N} y_i \log(p_i)$$

Loss - 3. binary_crossentropy

- You have **multi-label classification** (multi-hot labels)
- Each class is treated as a **separate binary problem** (present or not)
- Multi-hot label: `[1, 0, 1]`
- Prediction: [0.9, 0.2, 0.8]

$$\text{loss} = -\sum_{i=1}^{N} [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)]$$

Loss vs. Label Dimension

Summary Table

Task Type	Label Format	Label Shape	Loss Function
Single-label	Integer	`(batch_size,)`	`sparse_categorical_crossentropy`
Single-label	One-hot	`(batch_size, num_classes)`	`categorical_crossentropy`
Multi-label	Multi-hot	`(batch_size, num_classes)`	`binary_crossentropy`