Layer-Wise Relevance Propagation

Jorge Ángel Piñeiro Acevedo

18/12/2024

TUHH

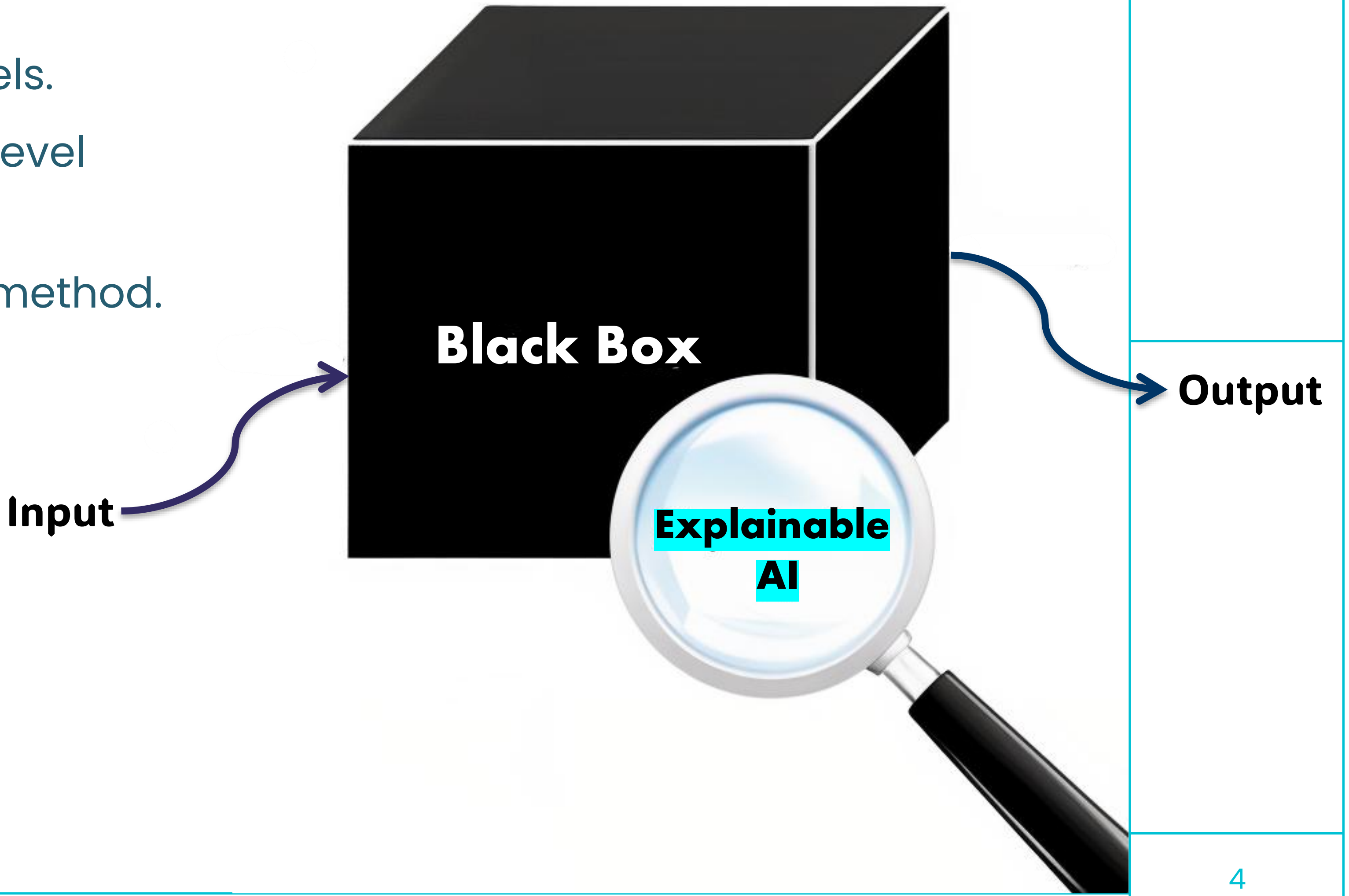
Main Sections

1. Introduction to Layer-Wise Relevance Propagation(LRP).
2. Conceptual Foundations.
3. Mathematical Formulation of LRP.
4. LRP Rules and Their Variants
5. Applying LRP to Different Network Components.
6. Task: Explaining CNN Predictions on CIFAR-10 Dataset using LRP.
7. Limitations Challenges and Future Directions.
8. Conclusion and Takeaways.

Introduction to Layer-Wise Relevance Propagation (LRP)

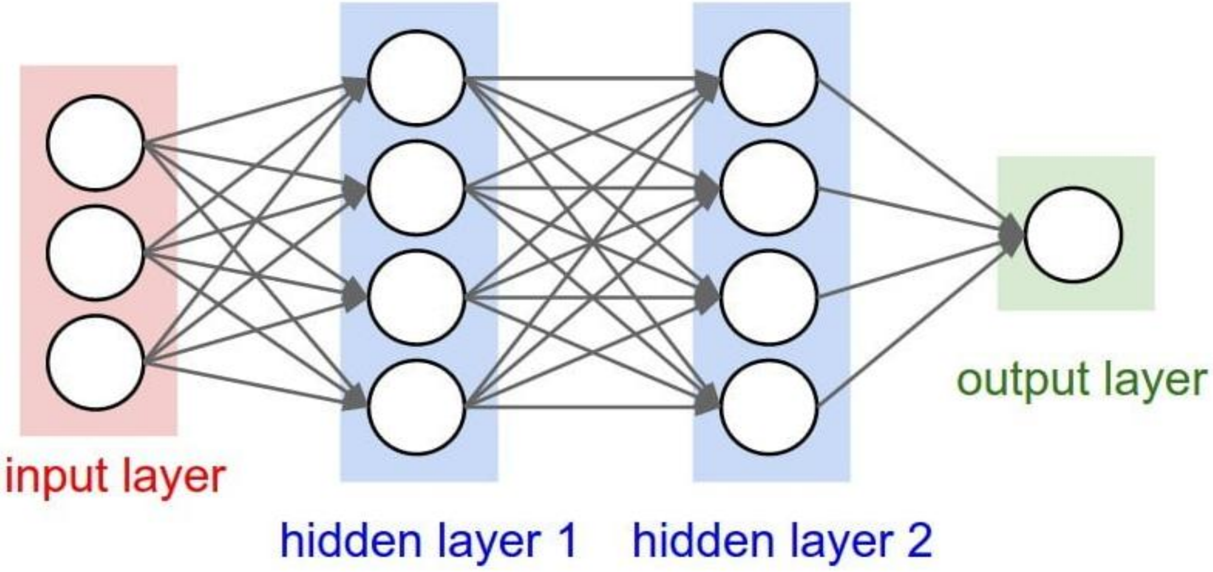
Understanding Layer-Wise Relevance Propagation

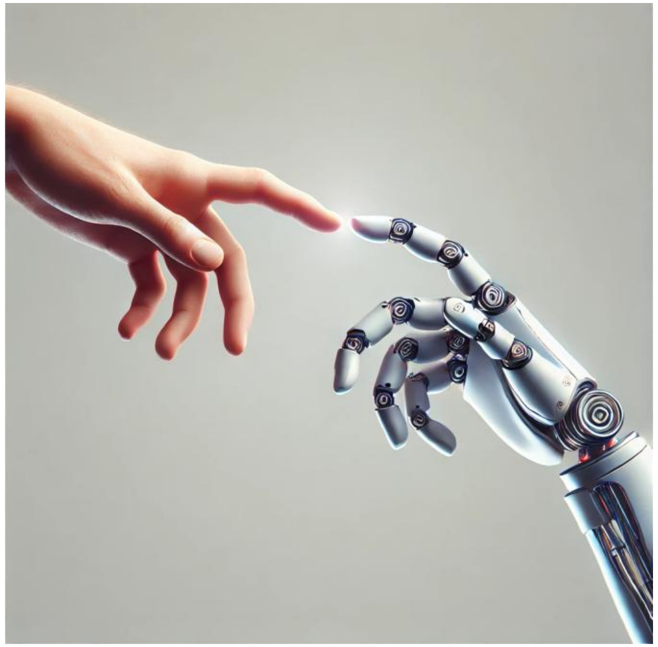
- Explaining complex deep models.
- Importance of local, instance-level explanations.
- LRP as a backward attribution method.



Why LRP?

- Explanation provided after training the models.
- Complements performance metrics.
- Enhances trust and transparency.

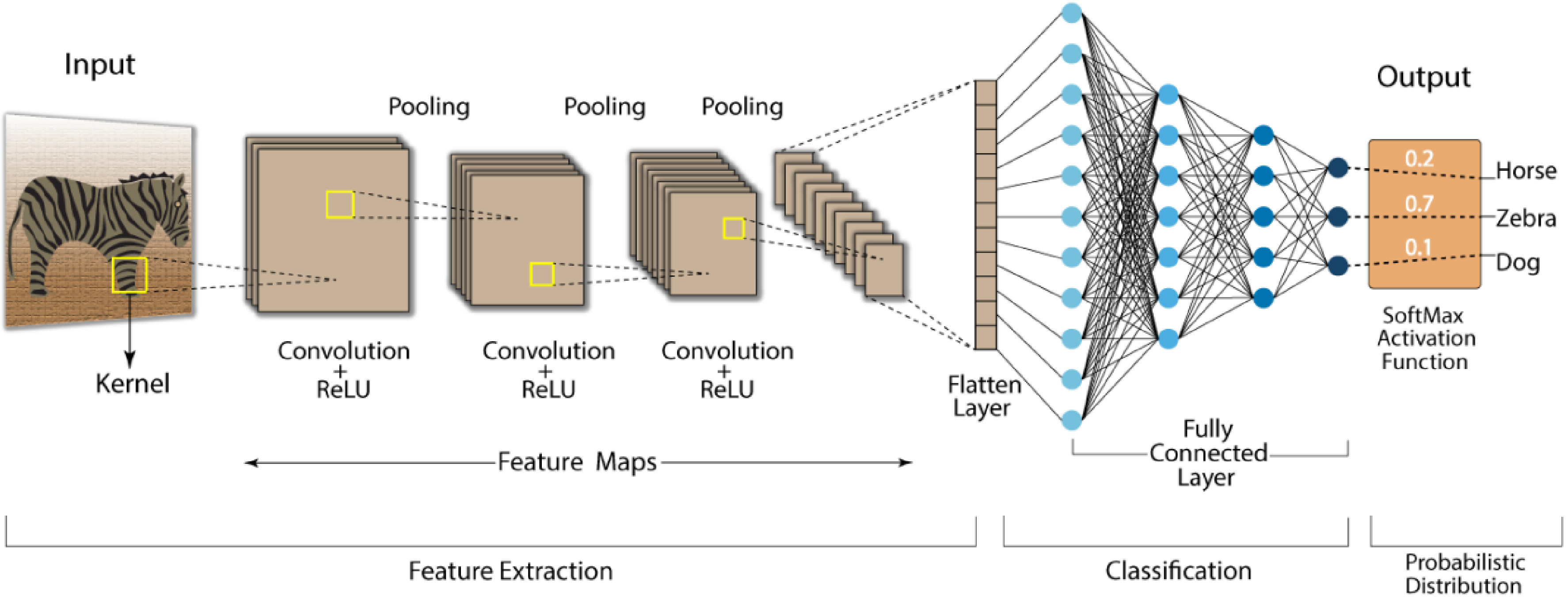




Conceptual Foundations

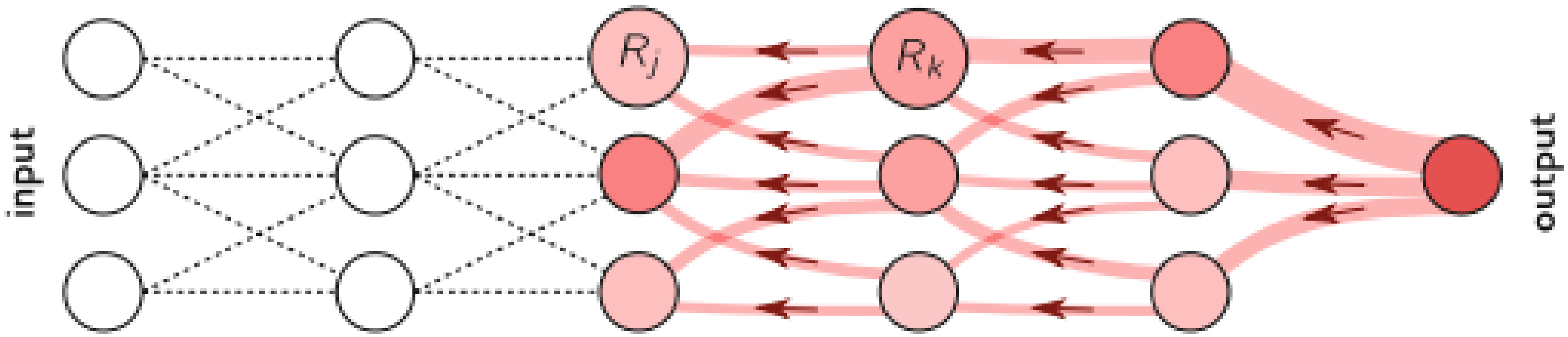
The 'Attribution Problem'

- Complex internal representations.
- Need to assign credit or blame for each feature.
- Bridge between input and output.



Local Explanation Paradigm

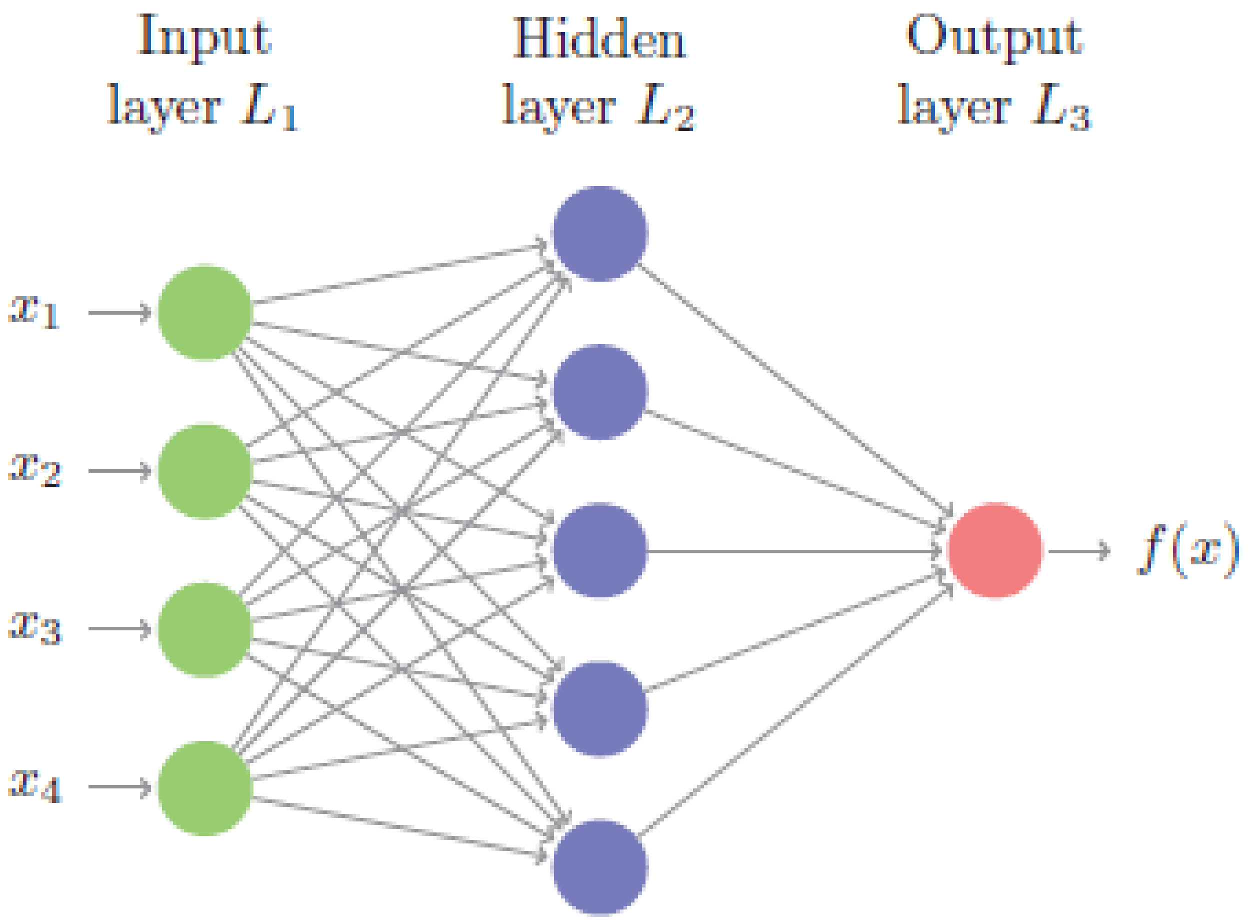
- Focus on single predictions.
- Distinguish from global model summaries.
- Applicable to various data modalities



Mathematical Formulation of LRP

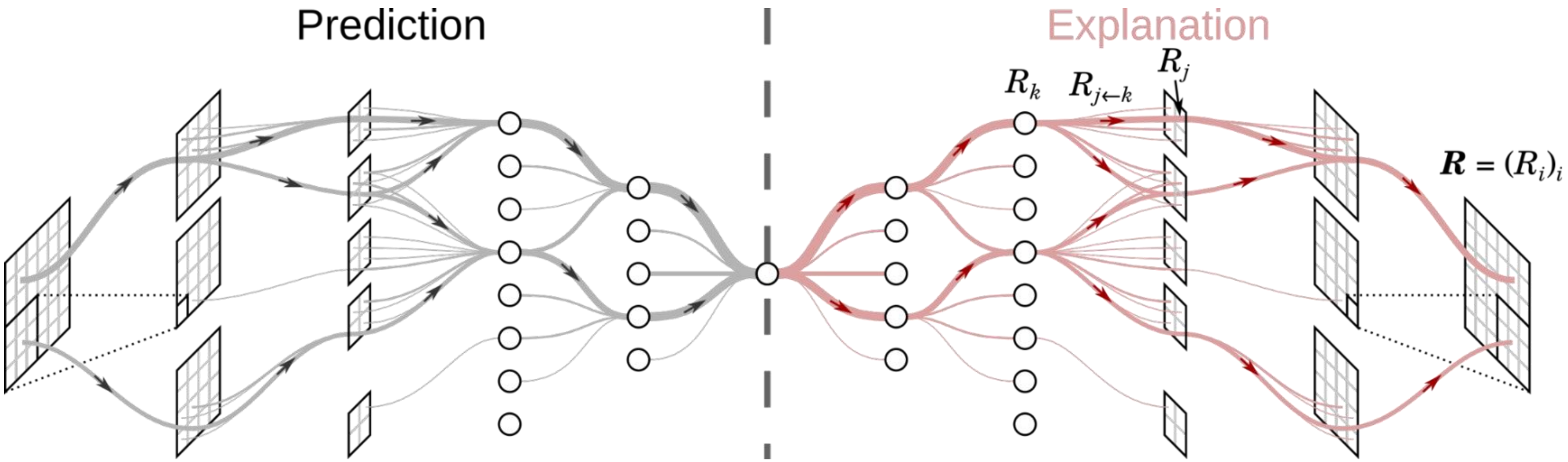
Basic Notation

- Let x be input features.
- Let $f(x)$ be the model's output score.
- Layers indexed by l.



Relevance Propagation Principle

- Conservation principle.
- Back-propagation of relevance.
- From output layer to input layer.



The LRP Decomposition Rule

- Local relevance redistribution rule.
- Uses contributions Z_{Jk} between pairs of neurons.
- Ensures layer-wise conservation.

$$R_j = \sum_k \frac{Z_{jk}}{\sum_j Z_{jk}} R_k$$

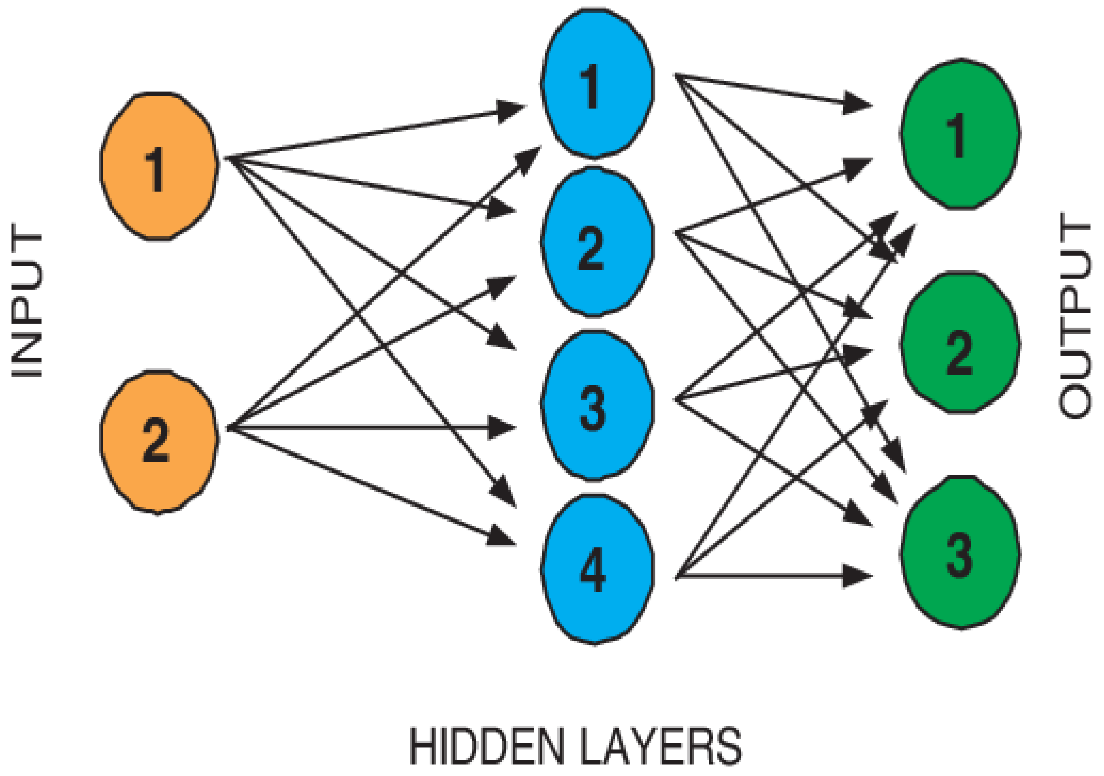
- R_J: Relevance assigned to a neuron j in the current layer.
- R_k: Relevance of neuron k in the upper layer.
- Z_{jk}: Connection between neuron j and k (typically the weight connecting j to k)
- $\sum_j Z_{jk}$: Normalization that ensures the total relevance is conserved between layers.

Example of Relevance Redistribution

- Input Layer: 2 neurons $(x1, x2)$
- Hidden Layer: 4 neurons $(h1, h2, h3, h4)$
- Output layer : 3 neurons $(o1, o2, o3)$

Given final relevances at output layer:

- $R(o1) = 0.5$
- $R(o2) = 0.3$
- $R(o3) = 0.2$



Relevance at the Hidden Layer

- Contributions from $(h1, h2, h3, and\ h4)$ to each output neuron are known.
- We apply the LRP formula.

Example Result:

- $R(h1) = \frac{0.8}{2} \cdot 0.5 + \frac{0.1}{1} \cdot 0.3 + \frac{0.5}{2} \cdot 0.2 = 0.28$
- $R(h2) = 0.27$
- $R(h3) = 0.245$
- $R(h4) = 0.205$

$$R_j = \sum_k \frac{Z_{jk}}{\sum_j Z_{jk}} R_k$$

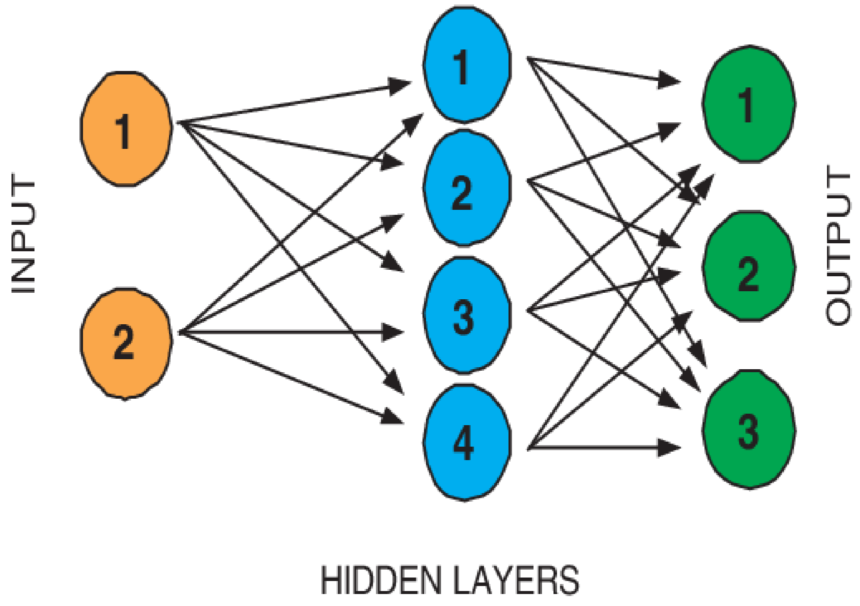
Relevance in the Input Layer

- Repeat the process from the hidden layer to the input layer.
- Suppose $h1$ distribute it's 0.28 relevance to $x1$ and $x2$.
- If $Z_{jk}(x1, h1) = 0.6$ and $Z_{jk}(x2, h1) = 0.4$

$$R(x1)\ from\ h1 = \frac{0.6}{1} \cdot 0.28 = 0.168$$

$$R(x2)\ from\ h1 = \frac{0.4}{1} \cdot 0.28 = 0.112$$

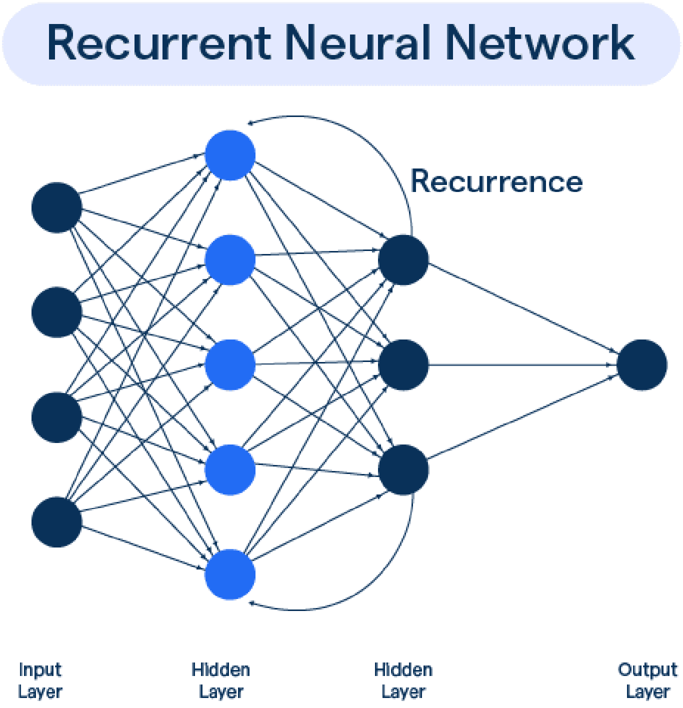
- Summing the contributions for all the hidden layers gives the full picture of how the input layer influenced the final result.



LRP Rules and Their Variants

Beyond the Basic LRP Rule

- Basic LRP-Z assumes only positive weights and activations.
- Real networks often include negative activations.
- Advanced LRP rules improve stability and flexibility.
- They enhance explanations for complex architectures



LRP-0: The baseline Relevance Rule

- Distributes relevance proportional to neuron activations and weights.
- Assumes no biases or stabilizing factors in the calculation.
- Works best for networks with predominantly positive weights.
- Simple yet effective for many dense and shallow networks.

$$R_j = \sum_k \frac{a_j w_{jk}}{\sum_{0,j} a_j w_{jk}} R_k$$

LRP—ε: Stabilizing the Relevance Flow

- Adds a small ε to the denominator to avoid division by values close to zero.
- Reduces sensitivity to numerical instabilities.
- Particularly useful when we are dealing with small or zero activations.

$$R_j = \sum_k \frac{a_j w_{jk}}{\varepsilon + \sum_{0,j} a_j w_{jk}} R_k$$

LRP-γ: Emphasizing Positive Contributions

- Adjusts the propagation by increasing the weight of positive contributions.
- Negatively contributing features are downplayed,.
- Useful when focusing on features that positively influence the final decision.

$$R_j = \sum_k \frac{a_j.(w_{jk} + \gamma w_{jk}^{+})}{\sum_{0,j} a_j.(w_{jk} + \gamma w_{jk}^{+})} R_k$$

- w_{jk}^{+}: Represents the positive part of the weight.
- γ: A scaling factor that increases the weight of positive contributions.

LRP-αβ: Balancing Positive and Negative Evidence

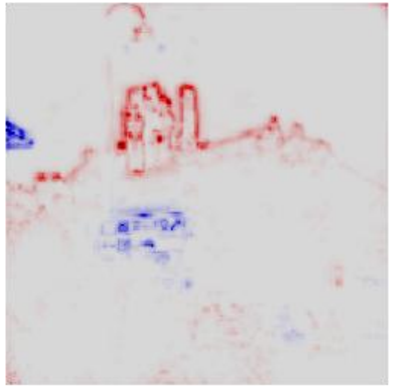
- Splits relevance into positive (α) and negative (β) components.
- Allows precise control over the relevance assigned to positive and negative contributions.
- By adjusting α and β, we can focus more on positive features, negative features, or both.

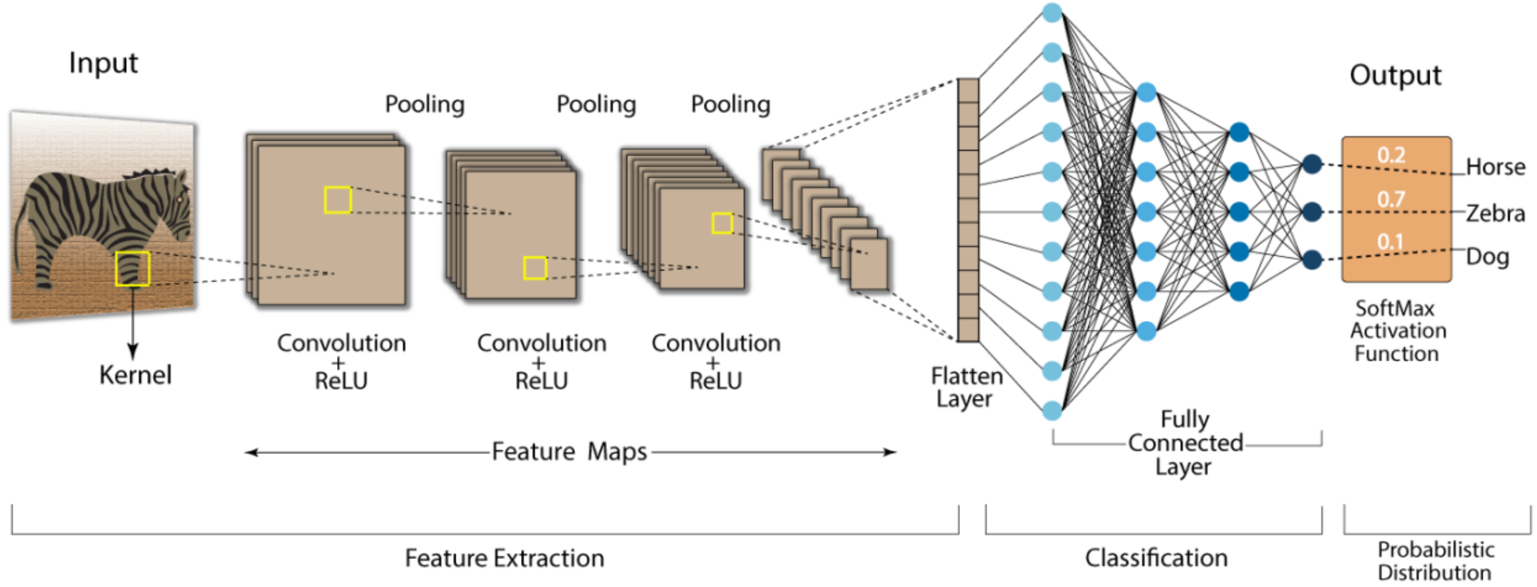
$$R_j = \sum_k \left(\alpha \frac{(a_j w_{jk})^+}{\sum_{0,j}(a_j w_{jk})^+} - \beta \frac{(a_j w_{jk})^-}{\sum_{0,j}(a_j w_{jk})^-}\right) R_k$$

Applying LRP to Different Network Components

Example 1: CNN for Image Classification

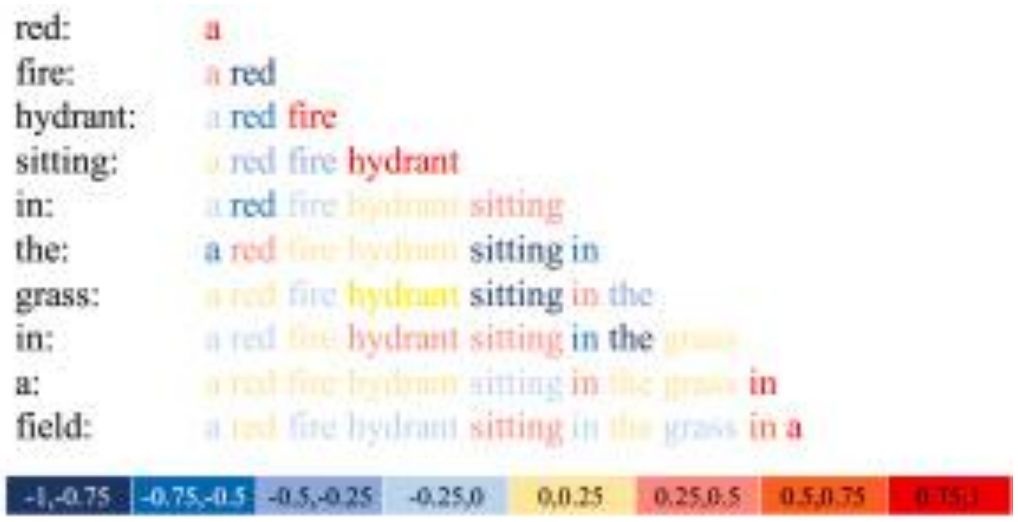
- Input: Image pixels x.
- Layers: Convolutional, ReLU, Pooling and Fully connected.
- Final Relevance map.



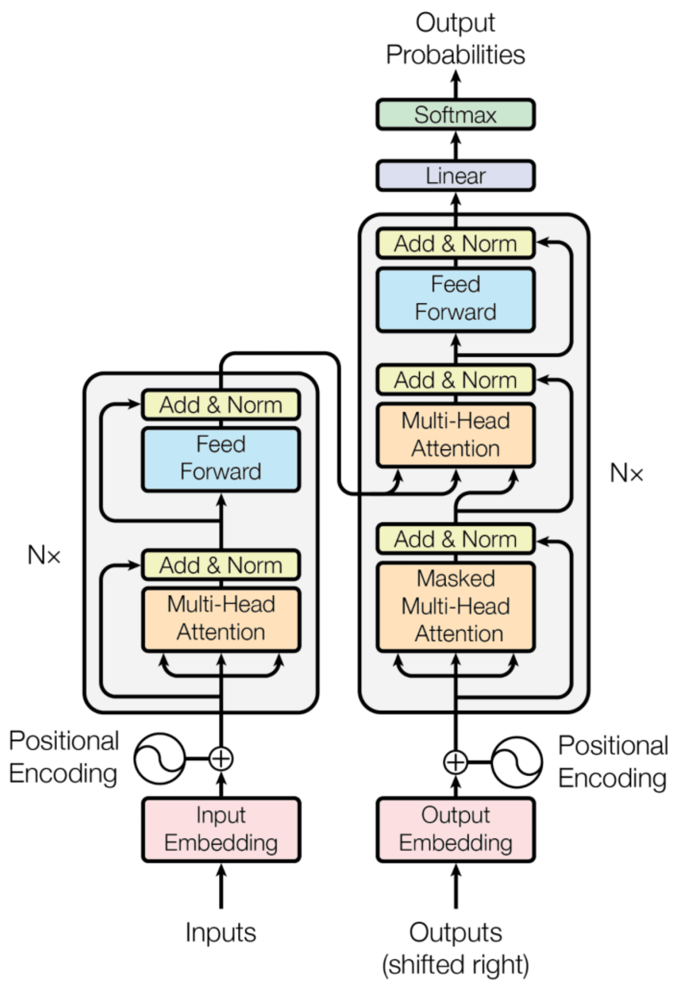


Example 2: LRP in Natural Language Processing (NLP)

- Words as input features.
- Embedding and recurrent layers.
- Highlighting crucial terms.



linguistic explanations of LRP



Example 3: LRP in Tabular/Structure Data

- Numerical and categorical input features.
- Fully connected layers.
- Identifying predictive features.

Parameter	Min value	Max value
Cement (kg/m3)	300	450
Water content (kg/m3)	150	225
Mineral admixture (kg/m3)	0	225
Calcium content (% by mass)	0	70
Silica content (% by mass)	0	70
w/c ratio	0.4	0.6
Curing period (days)	7	180
Specific gravity	1.80	3.15
Compressive strength (MPa)	7	60
Split Strength (MPa)	0.5	5

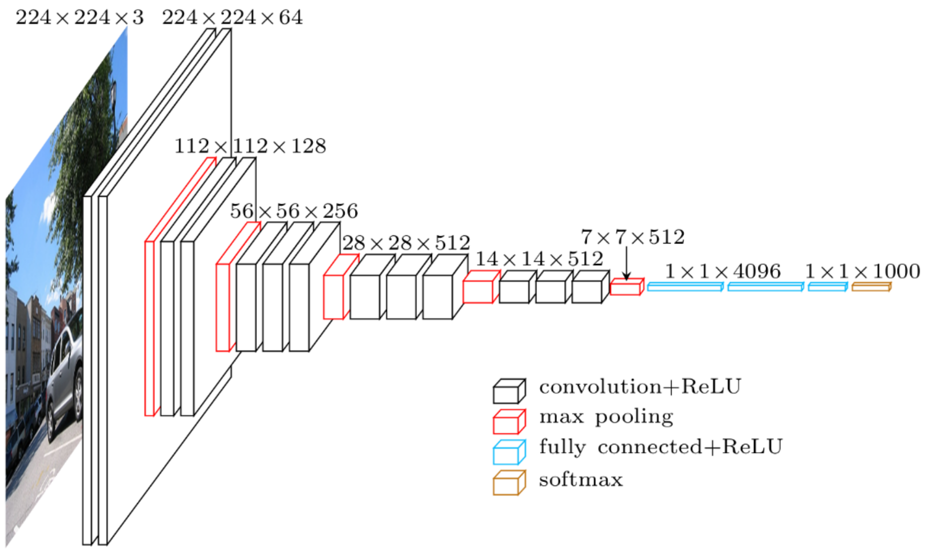
Task: Explaining CNN Predictions on CIFAR-10 Dataset using LRP

The CIFAR-10 Dataset

- 32x32 color images.
- 10 classes.
- Widely used as a Benchmark dataset.

The CNN Model (MiniVGG)

- Custom *MIniVGG* CNN architecture.
- Convolutional layers, ReLU and Pooling
- Fully Connected layers at the end.




```
class MiniVGG(nn.Module):
    def __init__(self, num_classes=10):
        super(MiniVGG, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Dropout(p=0.35),
            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),

            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Dropout(p=0.35),
            nn.Conv2d(128, 128, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2)
        )

        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Linear(8*8*128, 256),
            nn.ReLU(inplace=True),
            nn.Dropout(p=0.45),
            nn.Linear(256, num_classes)
        )
```

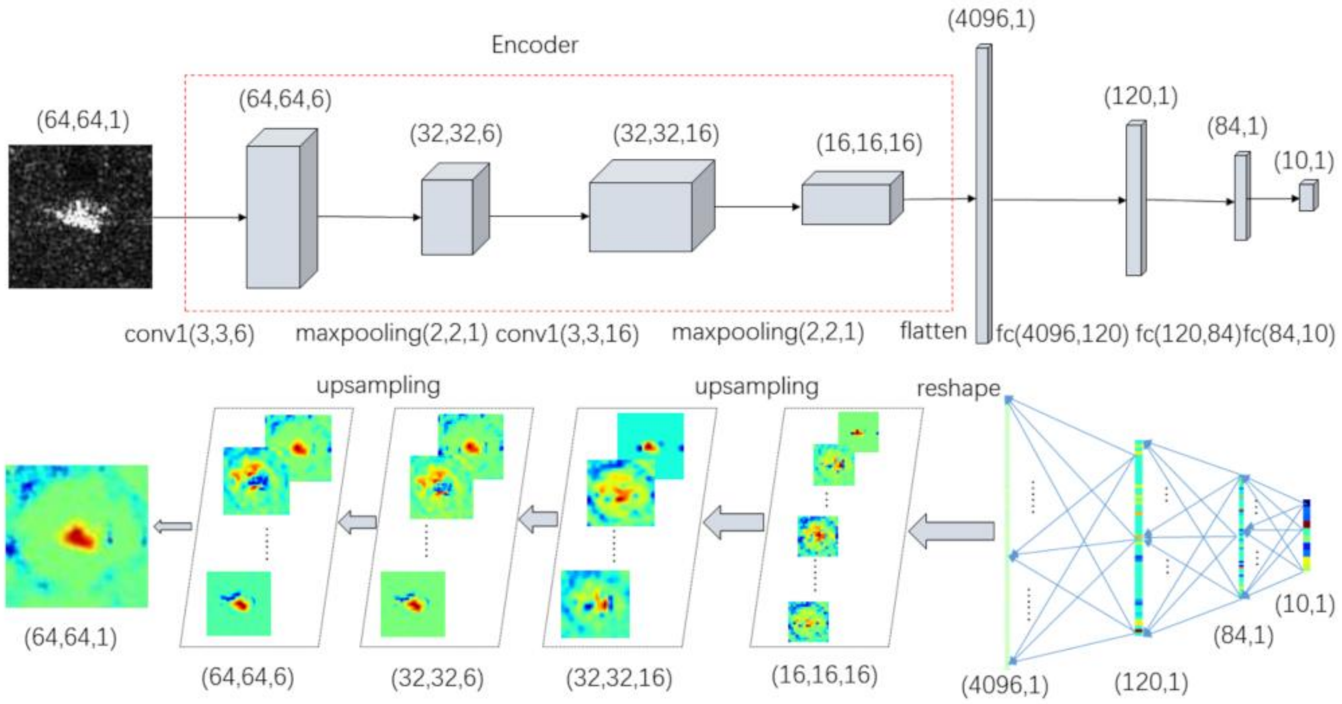
Training and Performance Summary

- Training: 12 epoch, Adam optimizer.
- Achieved around $\sim 80\%$ accuracy.
- Balanced performance on all classes.

```
Epoch [1/12] Training Loss: 1.4701, Validation Loss: 1.0918, Validation Accuracy: 61.09%
Epoch [2/12] Training Loss: 1.0504, Validation Loss: 0.9149, Validation Accuracy: 68.29%
Epoch [3/12] Training Loss: 0.8785, Validation Loss: 0.7783, Validation Accuracy: 73.89%
Epoch [4/12] Training Loss: 0.7595, Validation Loss: 0.7714, Validation Accuracy: 73.05%
Epoch [5/12] Training Loss: 0.6863, Validation Loss: 0.6883, Validation Accuracy: 76.09%
Epoch [6/12] Training Loss: 0.6218, Validation Loss: 0.6783, Validation Accuracy: 76.34%
Epoch [7/12] Training Loss: 0.5707, Validation Loss: 0.6601, Validation Accuracy: 77.48%
Epoch [8/12] Training Loss: 0.5205, Validation Loss: 0.6644, Validation Accuracy: 78.12%
Epoch [9/12] Training Loss: 0.4854, Validation Loss: 0.6492, Validation Accuracy: 77.84%
Epoch [10/12] Training Loss: 0.4569, Validation Loss: 0.6409, Validation Accuracy: 79.23%
Epoch [11/12] Training Loss: 0.4338, Validation Loss: 0.6442, Validation Accuracy: 78.73%
Epoch [12/12] Training Loss: 0.4001, Validation Loss: 0.6939, Validation Accuracy: 78.17%
```

Applying LRP in Our Scenario

- Using LRP to interpret CNN predictions on CIFAR-10.
- Identifies pixels and regions most influential for the model’s decision
- Enables validation and debugging of the model’s reasoning.



The *Zennit* Library and Implementation Details

- *Zennit* is a flexible library for explainability.
- *EpsilonPlusFlat* is the LRP rule chosen.
- Integration with *PyTorch* model.

$$R_j = \sum_k \frac{a_j w_{jk} + \beta}{\varepsilon + \sum_{0,j} a_j w_{jk} + \beta} R_k$$

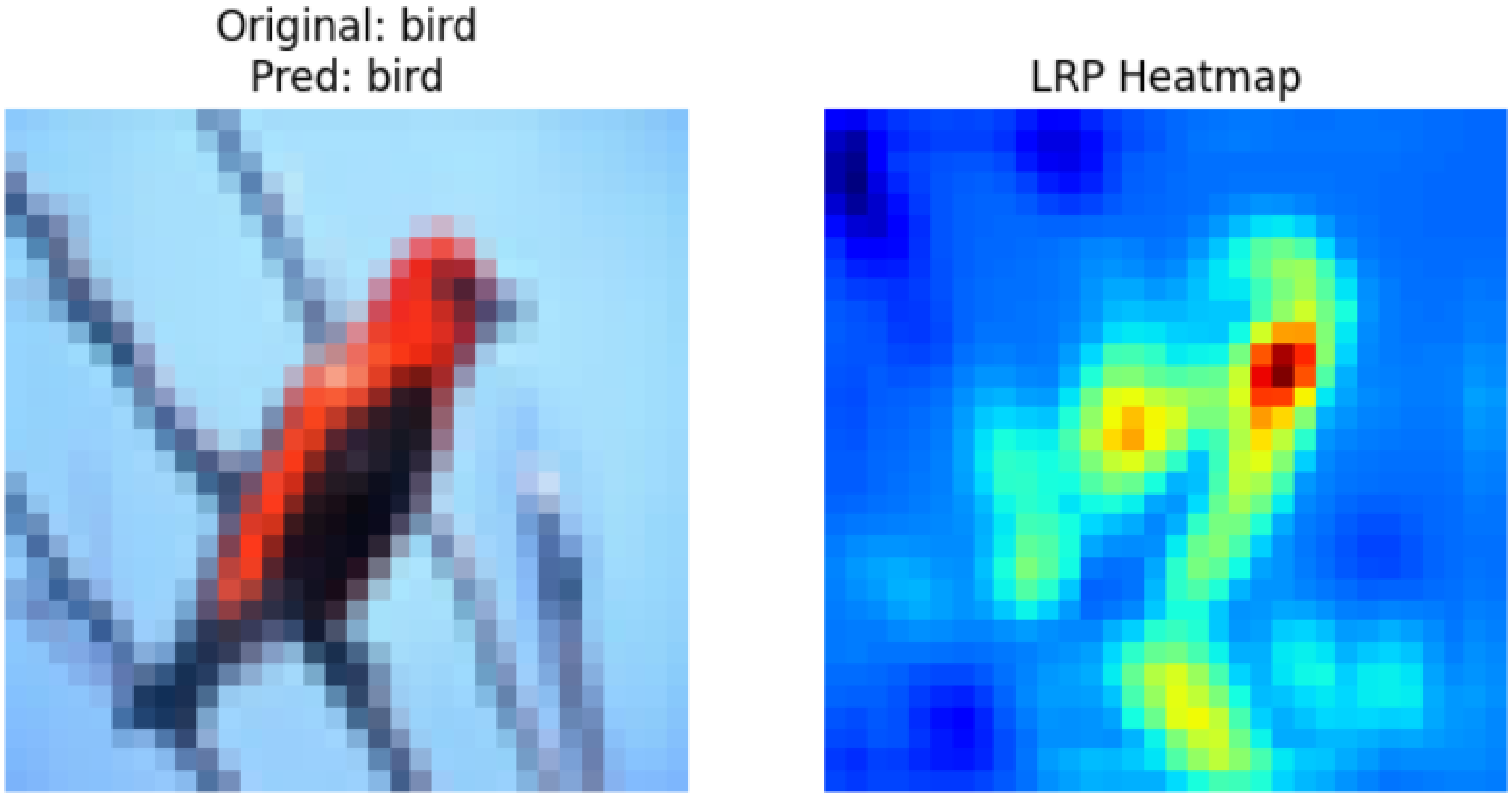
Code Overview for LRP application with Zennit

- Load the model and images.
- Apply attribution.
- Obtain relevance scores.
- Generate Heatmaps.

```
composite = EpsilonPlusFlat(epsilon=1e-6)

# Create an Attribution object
attr = Gradient(model, composite)

example_loader = DataLoader(test_set,
batch_size=5, shuffle=True)
images, labels = next(iter(example_loader))
images, labels = images.cuda(), labels.cuda()

model.eval()
with torch.no_grad():
    outputs = model(images)
_, predicted = outputs.max(1)
```

```
# Define a function to create attribution
def attr_output_fn(output):
    # Create a one-hot tensor
    one_hot = torch.zeros_like(output)
    one_hot[range(len(predicted)), predicted] = 1.0
    return one_hot

# Compute LRP attributions
output, relevance = attr(images,
attr_output=attr_output_fn)
R = relevance.detach().cpu().numpy()

for i in range(len(images)):
    # Summation over channels
    heatmap = R[i].sum(axis=0)
    # Normalize heatmap
    heatmap = (heatmap - heatmap.min()) /
(heatmap.max() - heatmap.min() + 1e-10)
```

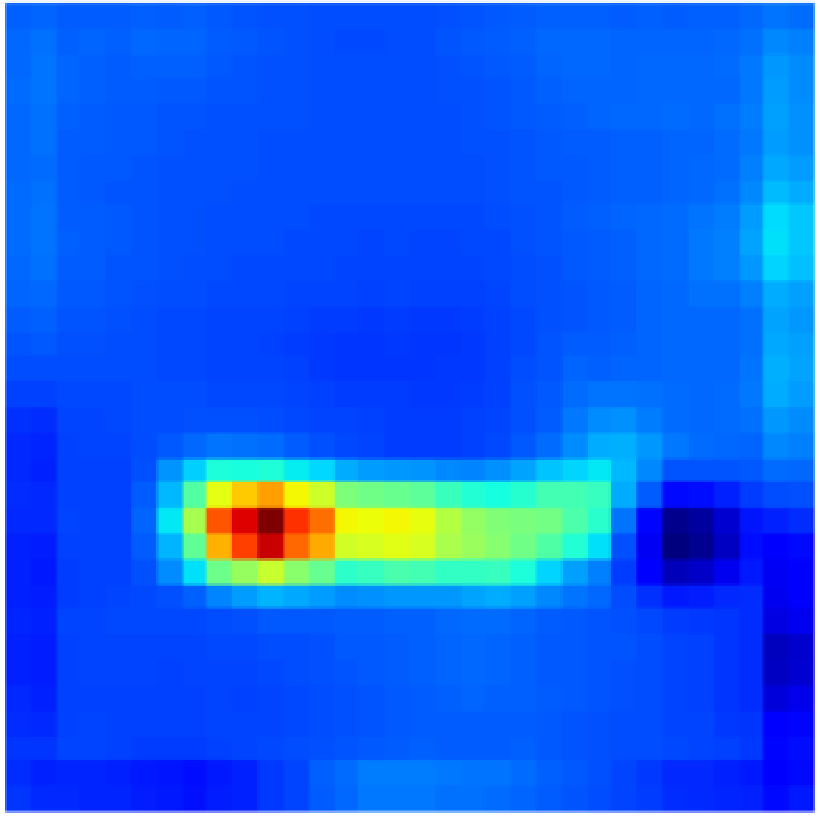
Example 1 – Bird (Correct Classification)



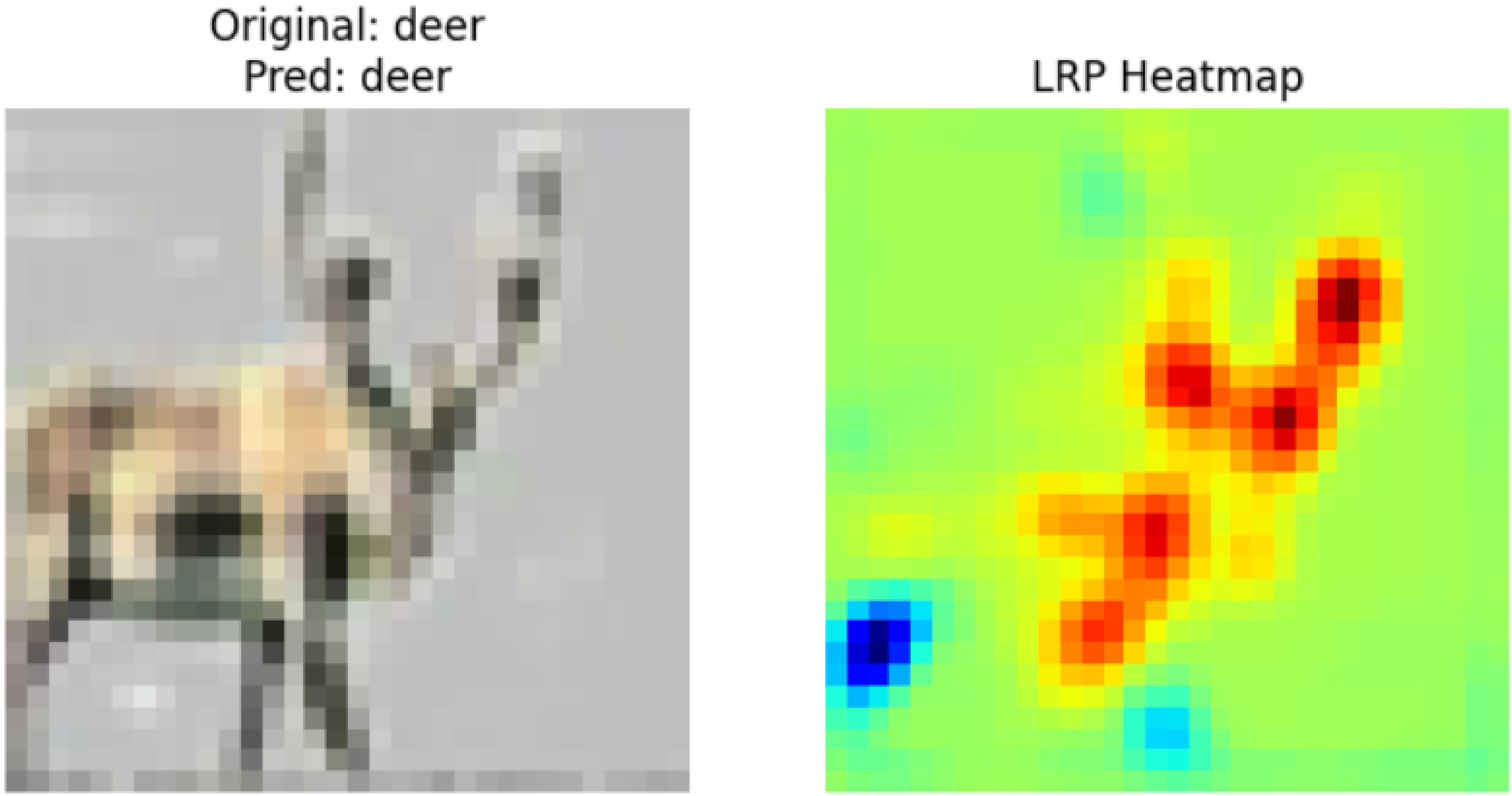
Example 2: Ship (Correct Classification)

Original: ship
Pred: ship

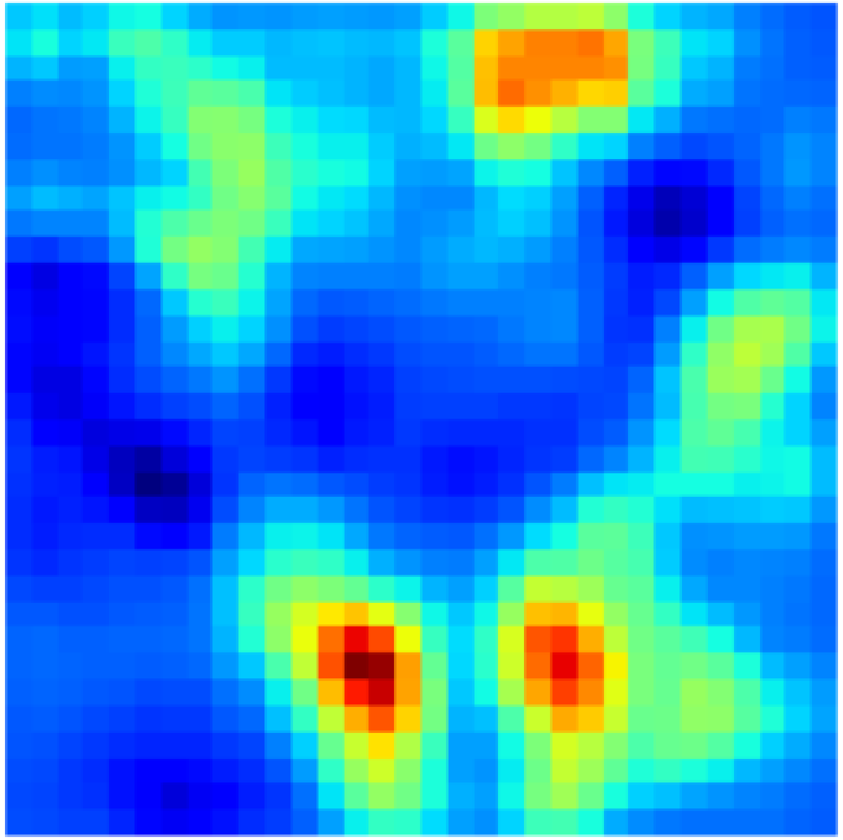
LRP Heatmap



Example 2: Deer (Correct Classification)



Example 4 – Cat Misclassified as Deer



Practical Uses and Task Conclusions

- Confirming model attention on relevant features.
- Helps debug and improve models.
- Informing adjustments to data or model design

Limitations Challenges and Future Directions

Known Limitations and Challenges

- **Limitations:**
  - Interpretability depends on model complexity.
  - Sensitivity to input variations (adversarial examples).
  - Difficult to validate without domain knowledge.

- **Ongoing Research:**
  - More robust LRP rules for complex architectures (Transformers, GNNs)
  - Integration with uncertain quantification.
  - Combining LRP with causal inference Methods.

Conclusion and Takeaways

- LRP helps **visualize model decisions** by redistributing relevance scores.
- Works on diverse data types: images, text, and structured data.
- Useful for **debugging**, identifying errors and improving models.
- Ongoing research focuses on improving LRP for **complex architectures**.
- A key tool for ensuring trust and **interpretability** in AI models