基于卷积神经网络图像分类任务的可解释性分析方法复现

任务要求

经典的可解释性方法有LIME, RISE, Grad-CAM, Grad-CAM++, ScoreCAM, LayerCAM等，阅读原始论文
简述每种可解释性方法的基本原理，并对各类可解释性方法进行分析、比较；
复现（也可运行开源）LIME代码，得到可解释性分析结果；
复现（也可运行开源）Grad-CAM++代码，得到可解释性分析结果；
复现（也可运行开源）ScoreCAM代码，得到可解释性分析结果；
对比三种方法的可解释性分析结果，分析各自的优缺点；

引言

在卷积神经网络（CNN）于图像分类任务取得巨大成功的同时，其 “黑盒” 特性使得理解模型决策过程变得至关重要。可解释性方法能够帮助我们深入理解模型为何做出特定预测，从而增加模型的透明度与可信度。本文将详细探讨 LIME、RISE、Grad - CAM、Grad - CAM++、ScoreCAM 和 LayerCAM 这几种经典的可解释性方法，包括它们的基本原理、实现过程、实验结果分析以及相互之间的比较。

可解释性方法原理

LIME 的原理解释：

LIME（Local Interpretable Model-agnostic Explanations） 是一种用于解释任何机器学习模型预测的局部可解释性方法。它的核心思想是通过在局部区域内近似复杂模型的决策过程，生成一个简单且可解释的模型（如线性模型），从而帮助用户理解模型在特定输入上的决策依据。

1. 局部近似

LIME 的核心思想是：**在某个输入点附近，复杂的模型可以被一个简单的可解释模型近似。**具体来说，LIME 通过在输入点附近生成一组扰动样本，并使用这些样本的预测结果来拟合一个简单的模型（如线性模型）。这个简单模型在局部区域内近似复杂模型的行为。

- **输入点**：假设我们有一个输入点 x ，模型 f 对该输入的预测为f(x)。
- **扰动样本**：LIME 在输入点 x 附近生成一组扰动样本 z。这些样本是通过对 x 进行微小的修改得到的。
- **简单模型**：LIME 使用这些扰动样本 z 和它们的预测结果 f(z) 来拟合一个简单的可解释模型 g (如线性模型)。这个简单模型 g 在局部区域内近似复杂模型 f 的行为。

2. 可解释表示

LIME 使用一种**可解释的表示**来描述输入。对于不同的数据类型，可解释的表示方式不同：

- **文本数据**：可解释的表示可以是词袋模型（Bag of Words），即一个二进制向量，表示某个词是否出现在文本中。
- **图像数据**：可解释的表示可以是超像素（Superpixel），即图像中的连续像素块。LIME 将图像分割成若干超像素，每个超像素表示一个特征。
- **表格数据**：可解释的表示可以是原始特征的一个子集。

通过这种可解释的表示，LIME 能够将复杂的高维输入（如图像或文本）转换为低维的、人类可理解的表示。

3. **扰动样本生成

LIME 通过在输入点 x 附近生成扰动样本来探索模型的局部行为。具体来说：

- **扰动样本**：LIME 通过对输入 x 进行随机扰动生成一组样本 z 。例如，对于文本数据，可以通过随机删除某些词来生成扰动样本；对于图像数据，可以通过随机遮挡某些超像素来生成扰动样本。
- **样本权重**：LIME 为每个扰动样本 z 分配一个权重 \pi_x(z) ，表示该样本与原始输入 x 的相似度。通常使用指数核函数来计算权重：

$$\pi_x(z) = \exp\left(-\frac{D(x,z)^2}{\sigma^2}\right)$$

其中 `D(x, z)` 是输入 `x` 和扰动样本 `z` 之间的距离，`sigma` 是核函数的宽度参数。

4. 简单模型拟合

LIME 使用扰动样本 z 和它们的预测结果 f(z) 来拟合一个简单的可解释模型 g 。通常LIME 使用线性模型作简单模型：

$$g(z) = w_g \cdot z$$

其中 w_g 是线性模型的权重向量。

LIME 的目标是最小化简单模型 g 与复杂模型 f 在局部区域内的差异，同时保持简单模型的可解释性。具体来说，LIME 通过以下优化问题来拟合简单模型：

$$\xi(x) = \arg\min_{g \in G} \mathcal{L}(f, g, \pi_x) + \Omega(g)$$

其中：

$$\mathcal{L}(f, g, \pi_x)$$

是简单模型 (g) 与复杂模型 (f) 在局部区域内的差异，通常使用加权平方损失：

$$\mathcal{L}(f, g, \pi_x) = \sum_{z,z'} \pi_x(z)\big(f(z) - g(z')\big)^2$$

Omega(g) 是简单模型 g 的复杂度惩罚项，用于确保模型的可解释性。例如，对于线性模型，Omega(g) 可以是权重向量的 L1 范数，以确保模型的稀疏性。

5. 解释生成

通过拟合简单模型 g ，LIME 可以生成对输入 x 的解释。具体来说：

- **特征重要性**：简单模型 g 的权重向量 w_g 表示每个特征对模型预测的贡献。对于文本数据，权重表示每个词的重要性；对于图像数据，权重表示每个超像素的重要性。
- **解释输出**：LIME 输出一个解释，通常是特征及其权重的列表。例如，对于文本分类任务，LIME 可能会输出一组词及其对预测结果的贡献（正贡献或负贡献）。

6. **子模优化

为提供对模型的全局理解，LIME 还引入**子模优化**（Submodular Pick）方法，用于选择一组代表性的解释。具体来说：

- **解释矩阵**：LIME 构建一个解释矩阵W，其中每一行表示一个输入的解释，每一列表示一个特征的重要性。
- **子模优化**：LIME 通过子模优化选择一组解释，使得这些解释能够覆盖尽可能多的特征，同时保持解释的多样性。子模优化的目标是最大化解释的覆盖范围：

$$\text{Pick}(\mathcal{W}, I) = \arg\max_{V, |V| \le B} c(V, \mathcal{W}, I)$$

> 其中 `c(V, W, I)` 是解释集 `V` 的覆盖范围， `B` 是用户愿意查看的解释数量。

LIME 的优点是模型无关、灵活且易于理解，适用于多种数据类型和模型。然而，LIME 的解释是局部的，可能无法反映模型的全局行为，且依赖于扰动样本的生成策略。

Grad-CAM和Grad-CAM++的原理解释

Grad-CAM++ 是 **Grad-CAM** 的改进版本，旨在提供更精确的视觉解释，特别是在处理多目标定位和对象完整性方面**Grad-CAM++** 通过引入像素级加权和更高阶的梯度信息，改进了 Grad-CAM 的局限性。

1. Grad-CAM

在理解 Grad-CAM++ 之前，首先需要回顾 Grad-CAM 的基本原理：

- **目标**：Grad-CAM 的目标是生成一个热力图（Heatmap），显示模型在做出某个类别预测时关注的图像区域。

- **方法**：Grad-CAM 通过计算目标类别得分对最后一层卷积层特征图的梯度，来衡量每个特征图的重要性。具体公式：

$$w_k^c = \frac{1}{Z} \sum_i \sum_j \frac{\partial Y^c}{\partial A_{ij}^k}$$

  其中 Z 是特征图的像素数量， A_{ij}^k 是第 k 个特征图在位置 (i,j) 的激活值。
- **热力图生成**：Grad-CAM 的热力图 (L^c) 是通过对特征图进行加权求和得到的：

$$L^c = \text{ReLU}\left(\sum_k w_k^c A^k\right)$$

  其中 ReLU 用于确保只保留对目标类别有正贡献的区域。

Grad-CAM 的主要问题是：

1. **多目标定位问题**：当图像中有多个同类对象时，Grad-CAM 往往只能定位其中一个对象。
2. **对象完整性**：Grad-CAM 的热力图可能无法完整覆盖整个对象，导致部分区域被忽略。

2. Grad-CAM++

Grad-CAM++ 通过引入像素级加权和高阶梯度信息，解决了 Grad-CAM 的上述问题。以下是 Grad-CAM++ 的核心改进：

2.1 像素级加权

Grad-CAM++ 引入了像素级加权，即每个像素的梯度贡献不再是简单的平均值，而是通过加权组合来衡量。具体来说，Grad-CAM++ 使用以下公式计算特征图 (A^k) 的权重：

$$w_k^c = \sum_i \sum_j \alpha_{ij}^{kc} \cdot \text{ReLU}\left(\frac{\partial Y^c}{\partial A_{ij}^k}\right)$$

其中 alpha_{ij}^{kc} 是像素 (i,j) 的权重系数，用于衡量该像素对目标类别 c 的重要性。

2.2 权重系数的计算

Grad-CAM++ 通过引入高阶梯度信息来计算权重系数。计算公式为：

$$\alpha_{ij}^{kc} = \frac{\frac{\partial^2 Y^c}{(\partial A_{ij}^k)^2}}{2\frac{\partial^2 Y^c}{(\partial A_{ij}^k)^2} + \sum_a \sum_b A_{ab}^k \cdot \frac{\partial^3 Y^c}{(\partial A_{ij}^k)^3}}$$

其中：

$$\frac{\partial^2 Y^c}{(\partial A_{ij}^k)^2}$$

是目标类别得分对特征图的二阶偏导数。

$$\frac{\partial^3 Y^c}{(\partial A_{ij}^k)^3}$$

是目标类别得分 对特征图的三阶偏导数。

通过引入高阶梯度信息，Grad-CAM++ 能够更精确地衡量每个像素对目标类别的重要性，从而生成更准确的热力图。

2.3 热力图生成

Grad-CAM++ 的热力图 L^c 是通过对特征图进行加权求和得到的：

$$L^c = \mathrm{ReLU}\left(\sum_k w_k^c A^k\right)$$

其中 w_k^c 是通过像素级加权和高阶梯度信息计算得到的权重。

3. Grad-CAM++ 的优势和局限性

优势

1. **多目标定位**：通过像素级加权，Grad-CAM++ 能够更好地定位图像中的多个同类对象。
2. **对象完整性**：Grad-CAM++ 的热力图能够更完整地覆盖整个对象，避免部分区域被忽略。
3. **高阶梯度信息**：通过引入二阶和三阶梯度信息，Grad-CAM++ 能够更精确地衡量每个像素对目标类别的重要性。
   局限性
4. **计算复杂度**：由于引入了高阶梯度信息，Grad-CAM++ 的计算复杂度较高，尤其是在处理大模型时。
5. **梯度饱和问题**：Grad-CAM++ 仍然依赖于梯度信息，可能受到梯度饱和问题的影响。

Grad-CAM++ 通过引入像素级加权和高阶梯度信息，改进了 Grad-CAM 的多目标定位和对象完整性问题。它的核心思想是通过更精确的梯度计算，生成更准确的热力图，从而帮助用户理解模型的决策过程。尽管 Grad-CAM++ 在计算复杂度上有所增加，但它在多个任务中表现出了优越的性能。

Score-CAM 的原理解释

Score-CAM 是一种基于类激活映射（CAM）的后处理视觉解释方法。与 Grad-CAM 和 Grad-CAM++ 不同，Score-CAM 不依赖于梯度信息，而是通过前向传播的得分来衡量每个激活图的重要性。

1. Score-CAM 的核心思想

Score-CAM 的核心思想是：**通过前向传播的得分来衡量每个激活图对目标类别的贡献。**具体来说，Score-CAM 通过将激活图上采样到输入图像大小，并将其作为掩码应用于输入图像，计算目标类别的得分变化，从而确定每个激活图的重要性。

2. Score-CAM 的步骤

Score-CAM 的实现可以分为以下几个步骤：

2.1 激活图提取

假设我们有一个卷积神经网络 f，输入图像为 X，目标类别为 c 。Score-CAM 首先从模型的某一卷积层（通常是最后一层卷积层）提取激活图 A^k，其中 k 表示第 k 个通道的激活图。

2.2 激活图上采样

Score-CAM 将每个激活图 A^k 上采样到输入图像的大小，得到上采样后的激活图 M^k

$$M^k = \text{Upsample}(A^k)$$

上采样的目的是将激活图的空间分辨率与输入图像对齐，以便后续的掩码操作。

2.3 激活图归一化

为了生成平滑的掩码，Score-CAM 对每个上采样后的激活图 M^k 进行归一化，将其值映射到 [0, 1] 范围内

$$s(M^k) = \frac{M^k - \min(M^k)}{\max(M^k) - \min(M^k)}$$

归一化后的激活图 s(M^k) 将作为掩码应用于输入图像。

2.4 掩码应用与得分计算

Score-CAM 将归一化后的激活图 s(M^k) 作为掩码，应用于输入图像 X，生成掩码后的图像 X^k

$$X^k = s(M^k) \circ X$$

其中逐元素相乘是Hadamard 积

然后，Score-CAM 将掩码后的图像 X^k 输入模型 f，计算目标类别 c 的得分 f^c(X^k) 。这个得分反映了激活图对目标类别的贡献。

2.5 权重计算

Score-CAM 通过比较掩码后的图像 X^k 和基线图像 X_b （通常是全零图像）的得分，计算每个激活图 A^k 的权重 alpha_k^c：

$$\alpha_k^c = f^c(X^k) - f^c(X_b)$$

其中 f^c(X_b) 是基线图像的得分，通常接近于零。

为了确保权重的归一化，Score-CAM 对权重进行 softmax 归一化：

$$\alpha_k^c = \frac{\exp(\alpha_k^c)}{\sum_k \exp(\alpha_k^c)}$$

2.6 热力图生成

Score-CAM 的热力图 L^c 是通过对激活图进行加权求和得到的：

$$L^c = \text{ReLU}\left(\sum_k \alpha_k^c A^k\right)$$

其中 ReLU 用于确保只保留对目标类别有正贡献的区域。

---

3. Score-CAM 的优势和局限性

Score-CAM 的主要优势在于：

1. **不依赖于梯度**：Score-CAM 通过前向传播的得分来衡量激活图的重要性，避免了梯度饱和和噪声问题。
2. **直观的解释**：Score-CAM 的权重直接反映了激活图对目标类别的贡献，解释更加直观。
3. **多目标定位**：Score-CAM 能够更好地定位图像中的多个同类对象。
4. **类区分能力**：通过 softmax 归一化，Score-CAM 能够区分不同类别的贡献，即使目标类别的预测概率较低。
   尽管 Score-CAM 在多个方面表现优越，但它仍然存在一些局限性：
5. **计算复杂度**：Score-CAM 需要对每个激活图进行前向传播，计算开销较大，尤其是在处理大模型时。
6. **激活图选择**：Score-CAM 的效果依赖于激活图的选择，如果选择的激活图不够代表性，解释可能不够准确。

4. Score-CAM 与 Grad-CAM 的比较

特性	Score-CAM	Grad-CAM
依赖信息	前向传播得分	梯度信息
计算复杂度	较高（需要多次前向传播）	较低（只需一次反向传播）
梯度问题	不依赖梯度，避免梯度饱和和噪声问题	依赖梯度，可能受梯度饱和问题影响
多目标定位	较好	较差

特性	Score-CAM	Grad-CAM
类区分能力	较好（通过 softmax 归一化）	一般

> Score-CAM 通过前向传播的得分来衡量每个激活图的重要性，提供了一种不依赖梯度的视觉解释方法。它的核心步骤包括激活图提取、上采样、归一化、掩码应用、得分计算和热力图生成。Score-CAM 在图像分类、目标定位和模型调试等任务中表现出了优越的性能，尤其是在多目标定位和类区分能力方面。尽管计算复杂度较高，但 Score-CAM 提供了一种更直观、更稳定的解释方法。

RISE 的原理解释

RISE是一种用于解释黑盒模型的视觉解释方法，特别适用于图像分类任务。RISE 的核心思想是通过随机掩码对输入图像进行扰动，观察模型输出的变化，从而估计每个像素对模型预测的重要性。以下是 RISE 的详细原理解释：

1. RISE 的核心思想

RISE 的目标是生成一个**重要性图（Saliency Map）**，显示每个像素对模型预测的贡献。与白盒方法（如 Grad-CAM）不同，RISE 不需要访问模型的内部结构（如梯度、权重或特征图），而是通过随机掩码对输入图像进行扰动，观察模型输出的变化来估计像素的重要性。

2. RISE 的实现步骤

RISE 的实现可以分为以下几个步骤：

2.1 随机掩码生成

RISE 通过生成一组随机掩码 M_i 来对输入图像 I 进行扰动。每个掩码 M_i 是一个二值矩阵（0 或 1），表示是否保留该像素。掩码的生成过程如下：

1. **生成小尺寸掩码**：首先生成一个较小的二值掩码（如 7 x 7 ），每个像素以概率 p 设置为 1，否则为 0。
2. **上采样**：将小尺寸掩码通过双线性插值上采样到输入图像的大小（如 224 x 224 ）。
3. **随机平移**：对上采样后的掩码进行随机平移，以增加掩码的多样性。

2.2 掩码应用与模型输出

对于每个生成的掩码 (M_i)，RISE 将掩码应用于输入图像 (I)，生成掩码后的图像 (I \odot M_i)（其中 (\odot) 表示逐元素相乘）。然后将掩码后的图像输入黑盒模型 (f)，得到模型对目标类别的置信度得分 (f(I \odot M_i))。

2.3 重要性图计算

RISE 通过加权平均所有掩码 (M_i) 来生成重要性图 (S)，其中权重是模型对掩码后图像的置信度得分 (f(I \odot M_i))。具体公式为：

$$S(\lambda) = \frac{1}{N} \sum_{i=1}^{N} f(I \odot M_i) \cdot M_i(\lambda)$$

其中：

- S(lambda) 是像素 lambda 的重要性得分。
- M_i(lambda) 是掩码 M_i 在像素 lambda 处的值。
- N 是生成的掩码数量。

通过这种方式，RISE 能够估计每个像素对模型预测的贡献。

---

3. RISE 的数学原理

RISE 的数学原理基于条件期望的概念。RISE 定义像素 lambda 的重要性得分为：

$$S(\lambda) = \mathbb{E}_M[f(I \odot M) | M(\lambda) = 1]$$

即，在像素 lambda 被保留的条件下，模型得分的期望值。

通过蒙特卡洛采样，RISE 使用生成的掩码 (M_i) 来近似计算这个期望值：

$$S(\lambda) \approx \frac{1}{N} \sum_{i=1}^{N} f(I \odot M_i) \cdot M_i(\lambda)$$

4. RISE 的优势和局限性

优势：

1. **黑盒解释**：RISE 不需要访问模型的内部结构（如梯度或权重），适用于任何黑盒模型。
2. **灵活性**：RISE 可以应用于任何输入图像和模型架构，具有广泛的适用性。
3. **直观的解释**：通过随机掩码和模型输出的变化，RISE 能够生成直观的重要性图，显示每个像素对模型预测的贡献。
   局限性：
4. **计算开销**：RISE 需要对每个掩码进行前向传播，计算开销较大，尤其是在处理大模型时。
5. **掩码生成的质量**：RISE 的效果依赖于掩码生成的质量，如果生成的掩码不够多样化，解释可能不够准确。

5. RISE 与其他方法的比较

特性	RISE	Grad-CAM	LIME
依赖信息	模型输出	梯度信息	局部线性近似
计算复杂度	较高（需要多次前向传播）	较低（只需一次反向传播）	中等（需要多次前向传播）
适用性	任何黑盒模型	仅适用于可微模型	任何黑盒模型
解释粒度	像素	特征图	超像素

RISE 通过随机掩码对输入图像进行扰动，观察模型输出的变化，生成重要性图来解释黑盒模型的决策过程。它的核心思想是通过条件期望估计每个像素对模型预测的贡献。尽管计算复杂度较高，但 RISE 提供了一种灵活且直观的解释方法，适用于任何黑盒模型。

LayerCAM 的原理详解

LayerCAM 是一种基于类激活映射（Class Activation Mapping, CAM）的视觉解释方法，旨在为卷积神经网络（CNN）的决策提供更细粒度的解释。与传统的 CAM 和 Grad-CAM 不同，LayerCAM 通过利用多个卷积层的激活图来生成更精确的热力图，从而更好地捕捉模型在不同层次上的特征表示。以下是 LayerCAM 的详细原理解释：

1. LayerCAM 的核心思想

LayerCAM 的核心思想是：**通过结合多个卷积层的激活图，生成更细粒度的热力图。**传统的 CAM 和 Grad-CAM 通常只使用最后一层卷积层的激活图，而 LayerCAM 则利用多个层次的激活图，从而能够捕捉到不同层次的特征表示（如低层次的边缘、纹理信息和高层次的语义信息）。

2. LayerCAM 的步骤

LayerCAM 的实现可以分为以下几个步骤：

2.1 激活图提取

假设我们有一个卷积神经网络 f ， 输入图像为 X ， 目标类别为 c 。LayerCAM 从多个卷积层（通常是多个中间层和最后一层）提取激活图 A_l^k ， 其中 l 表示第 l 层，k 表示第 k 个通道的激活图。

2.2 梯度计算

对于每个卷积层 l ， LayerCAM 计算目标类别得分 Y^c 对激活图 A_l^k 的梯度

$$\frac{\partial Y^c}{\partial A_l^k}$$

这些梯度反映了每个激活图 A_l^k 对目标类别 c 的贡献。

2.3 权重计算

LayerCAM 通过梯度信息计算每个激活图 A_l^k 的权重 w_l^k

$$w_l^k = \frac{1}{Z} \sum_i \sum_j \frac{\partial Y^c}{\partial A_{l,ij}^k}$$

其中 Z 是激活图的像素数量， A_{l,ij}^k 是第 l 层第 k 个激活图在位置 (i,j) 的激活值。

2.4 热力图生成

对于每个卷积层 l ，LayerCAM 生成一个局部热力图 L_l^c

$$L_l^c = \mathrm{ReLU}\left(\sum_k w_l^k A_l^k\right)$$

其中 ReLU 用于确保只保留对目标类别有正贡献的区域。

2.5 多层热力图融合

LayerCAM 通过加权求和的方式将多个卷积层的热力图 L_l^c 融合成一个最终的热力图 L^c

$$L^c = \sum_l \alpha_l L_l^c$$

其中 alpha_l 是第 l 层热力图的权重，通常根据层的深度或经验值进行设置。

---

3. LayerCAM 的优势和局限性

优势

1. **多层次的激活图**：通过结合多个卷积层的激活图，LayerCAM 能够捕捉到不同层次的特征表示，从而生成更细粒度的热力图。
2. **更精确的定位**：由于利用了多个层次的信息，LayerCAM 能够更精确地定位图像中的目标对象。
3. **适用于复杂场景**：在复杂场景中，LayerCAM 能够更好地处理多目标定位和对象完整性。
   局限性
4. **计算复杂度**：由于需要计算多个卷积层的激活图和梯度，LayerCAM 的计算开销较大。
5. **权重选择**：多层热力图的融合权重 (\alpha_l) 需要根据经验或实验进行选择，可能影响最终的热力图质量。

4. LayerCAM 与 Grad-CAM 的比较

特性	LayerCAM	Grad-CAM
依赖信息	多个卷积层的激活图和梯度信息	最后一层卷积层的激活图和梯度信息

特性	LayerCAM	Grad-CAM
计算复杂度	较高（需要计算多个卷积层的梯度）	较低（只需计算最后一层的梯度）
定位精度	较高	一般
多目标定位	较好	较差
类区分能力	较好	一般

LayerCAM 通过结合多个卷积层的激活图，生成更细粒度的热力图，提供了一种更精确的视觉解释方法。它的核心步骤包括激活图提取、梯度计算、权重计算、热力图生成和多层热力图融合。LayerCAM 在图像分类、目标定位和模型调试等任务中表现出了优越的性能，尤其是在多目标定位和复杂场景中。尽管计算复杂度较高，但 LayerCAM 提供了一种更直观、更精确的解释方法。

复现结果与分析

这里使用创建的一个简单CNN用于作为使用MNIST数据做数字识别的网络作为例子演示三种解释性方法的结果

```
import torch
import os
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader
from torchvision import datasets, transforms

# 定义CNN模型
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=5)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=5)
        self.fc1 = nn.Linear(1024, 1024)
        self.fc2 = nn.Linear(1024, 10)
        self.dropout = nn.Dropout(0.5)
        self.pool = nn.MaxPool2d(2, 2)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.pool(self.relu(self.conv1(x)))
        x = self.pool(self.relu(self.conv2(x)))
        x = x.view(-1, 1024)
        x = self.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)
        return x

# 数据预处理
```

```
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.1307,), (0.3081,))
])
# 文件位置安排
current_dir = os.path.dirname(os.path.abspath(__file__))
model_path = os.path.join(current_dir, 'mnist_cnn_model.pth')
# 加载MNIST数据集
train_dataset = datasets.MNIST(root=current_dir, train=True, download=True,
transform=transform)
test_dataset = datasets.MNIST(root=current_dir, train=False,
transform=transform)

train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=1000, shuffle=False)

# 初始化模型、损失函数和优化器
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = CNN().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters())

# 训练模型
epochs = 5
for epoch in range(epochs):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = criterion(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % 100 == 0:
            print(f'Train Epoch: {epoch} [{batch_idx *
len(data)}/{len(train_loader.dataset)} ({100. * batch_idx /
len(train_loader):.0f}%)]\tLoss: {loss.item():.6f}')

# 测试模型
model.eval()
test_loss = 0
correct = 0
with torch.no_grad():
    for data, target in test_loader:
        data, target = data.to(device), target.to(device)
        output = model(data)
        test_loss += criterion(output, target).item()
        pred = output.argmax(dim=1, keepdim=True)
        correct += pred.eq(target.view_as(pred)).sum().item()
```

```
    test_loss /= len(test_loader.dataset)

    print(f'\nTest set: Average loss: {test_loss:.4f}, Accuracy:
    {correct}/{len(test_loader.dataset)} ({100. * correct /
    len(test_loader.dataset):.0f}%)\n')

    # 保存模型
    torch.save(model.state_dict(), model_path)
```

这里模型的结果保存为mnist_cnn_model.pth文件

同时这里使用MNIST数据集的一张图片做例子展示出结果，提取例子如下

```
import torch
from torchvision import datasets, transforms
from PIL import Image
import os
# 数据预处理
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.1307,), (0.3081,))
])

# 加载MNIST数据集
current_dir = os.path.dirname(os.path.abspath(__file__))
test_dataset = datasets.MNIST(root=current_dir, train=False, download=True,
transform=transform)

# 选择指定索引的图片（例如索引为10的图片）
index = 10  # 你可以修改这个索引
image, label = test_dataset[index]

# 将张量转换为PIL图像并保存
image = transforms.ToPILImage()(image)
image.save('example.png')

print(f"Saved image with index {index} (label: {label}) as example.png")
```

LIME 复现

LIME是模型无关的可解释性方法，适用于任何类型的模型。对于图像数据，我们将使用 `LimeImageExplainer` 来处理卷积神经网络（CNN）模型

首先我们需要pip引入lime
再写复现的代码

```
  pip install lime
```

```python
import torch
import numpy as np
from PIL import Image
from torchvision import transforms
from lime import lime_image
from skimage.segmentation import mark_boundaries
import matplotlib.pyplot as plt
from source import CNN  # 导入CNN模型

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = CNN().to(device)
model.load_state_dict(torch.load('mnist_cnn_model.pth',
map_location=device), strict=False)  # 修复警告
model.eval()

transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.1307,), (0.3081,))
])

# 加载图像
image = Image.open('example.png').convert('L')  # 读取灰度图像
image = transform(image).unsqueeze(0).to(device)
# LIME解释器
def batch_predict(images):
    model.eval()
    # 将输入图像从3通道RGB转换回1通道灰度
    batch = torch.stack([transform(Image.fromarray((img[:, :, 0] *
255).astype(np.uint8)).convert('L')) for img in images], dim=0).to(device)
    logits = model(batch)
    probs = torch.nn.functional.softmax(logits, dim=1)
    return probs.detach().cpu().numpy()
# 将单通道图像转换为3通道图像
image_np = image.squeeze().cpu().numpy()
image_rgb = np.stack([image_np] * 3, axis=-1)  # 复制单通道为3通道
# 创建LIME解释器
explainer = lime_image.LimeImageExplainer()
explanation = explainer.explain_instance(image_rgb, batch_predict,
top_labels=5, hide_color=0, num_samples=1000)
# 可视化解释结果
temp, mask = explanation.get_image_and_mask(explanation.top_labels[0],
positive_only=True, num_features=5, hide_rest=False)
plt.imshow(mark_boundaries(temp / 2 + 0.5, mask))
plt.title('LIME Explanation')
plt.show()
```

Grad - CAM++ 复现

```python
import torch
import numpy as np
from PIL import Image
from torchvision import transforms
import matplotlib.pyplot as plt
from source import CNN  # 导入CNN模型
# 加载模型
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = CNN().to(device)
model.load_state_dict(torch.load('mnist_cnn_model.pth',
map_location=device), strict=False)  # 修复警告
model.eval()

# 图像预处理
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.1307,), (0.3081,))
])
# 加载图像
image = Image.open('example.png').convert('L')  # 读取灰度图像
image = transform(image).unsqueeze(0).to(device)

# 获取卷积层的输出和梯度
conv_output = None
def hook_fn(module, input, output):
    global conv_output
    conv_output = output
# 注册钩子到最后一个卷积层
model.conv2.register_forward_hook(hook_fn)
# 前向传播
output = model(image)
target_class = output.argmax(dim=1).item()
print(f"Target class: {target_class}")

# 反向传播计算梯度
output[:, target_class].backward()
# 获取梯度
gradients = model.conv2.weight.grad  # 形状是 [64, 32, 5, 5]
activations = conv_output.detach()  # 形状是 [1, 64, 8, 8]

# Grad-CAM++
# 1. 计算 alpha
# 对每个输出通道的梯度求和，得到形状为 [64, 1, 1]
alpha = torch.sum(gradients, dim=(2, 3), keepdim=True)  # 形状是 [64, 32, 1,
1]
alpha = torch.mean(alpha, dim=1, keepdim=True)  # 对输入通道取平均，形状是 [64,
1, 1, 1]
alpha = torch.nn.functional.relu(alpha)  # 只保留正梯度
alpha = alpha / (torch.sum(alpha, dim=1, keepdim=True) + 1e-10)  # 归一化
```

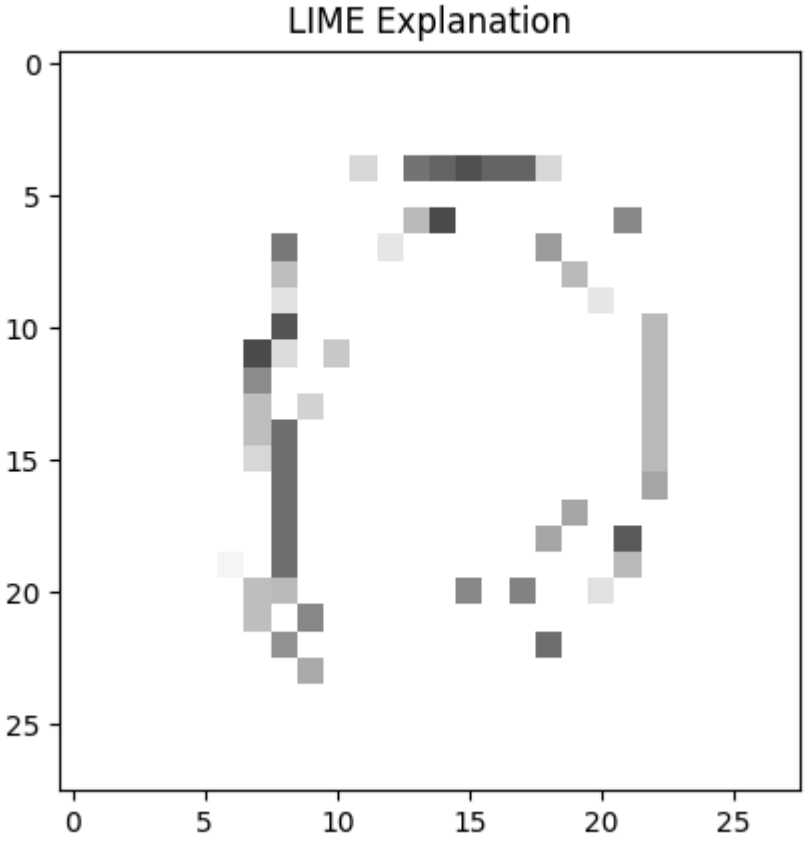
```
# 2. 调整 alpha 的形状以匹配 activations
alpha = alpha.permute(1, 0, 2, 3)  # 形状变为 [1, 64, 1, 1]
# 3. 检查形状是否匹配
if alpha.shape[1] != activations.shape[1]:
    raise ValueError(f"Alpha shape {alpha.shape} does not match activations
shape {activations.shape}")
# 4. 计算加权激活
weighted_activations = torch.sum(activations * alpha, dim=1, keepdim=True)
# 5. 计算热力图
heatmap = weighted_activations.squeeze().cpu().numpy()
# 7. 归一化热力图
heatmap = (heatmap - np.min(heatmap)) / (np.max(heatmap) - np.min(heatmap) +
1e-10)
# 可视化热力图
plt.matshow(heatmap, cmap='viridis')
plt.title('Grad-CAM++ Heatmap')
plt.colorbar()
plt.show()
```

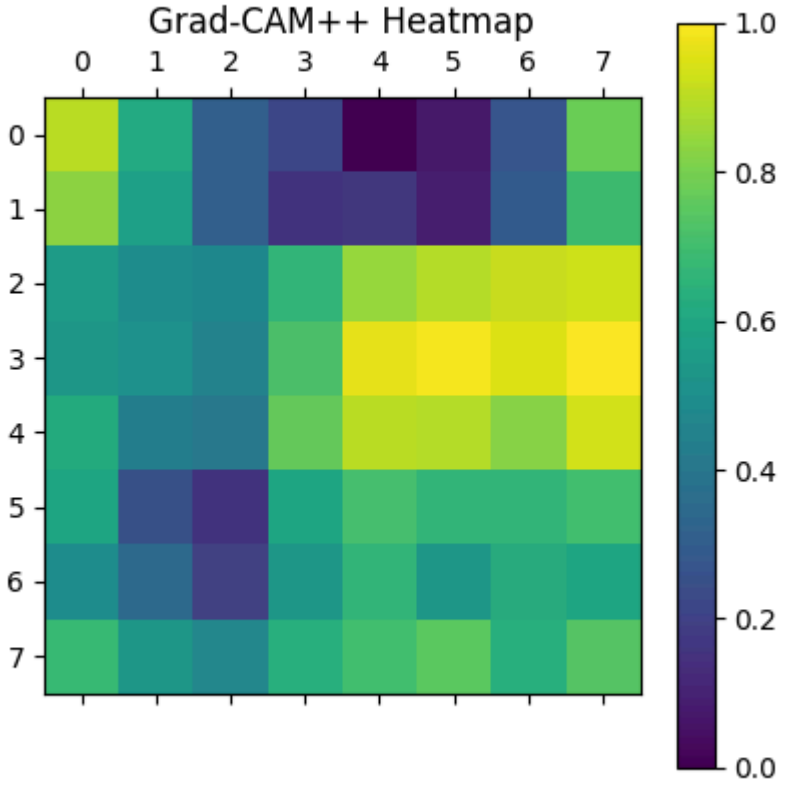
ScoreCAM 复现

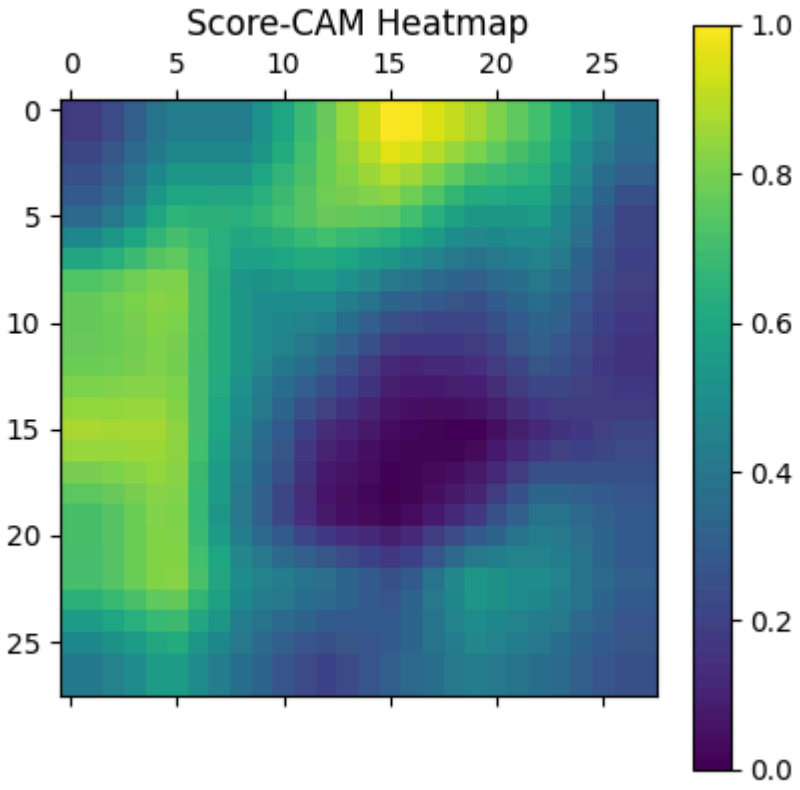
```
import torch
import numpy as np
from PIL import Image
from torchvision import transforms
import matplotlib.pyplot as plt
from source import CNN  # 导入CNN模型
# 加载模型
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = CNN().to(device)
model.load_state_dict(torch.load('mnist_cnn_model.pth',
map_location=device), strict=False)  # 修复警告
model.eval()
# 图像预处理
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.1307,), (0.3081,))
])
# 加载图像
image = Image.open('example.png').convert('L')  # 读取灰度图像
image = transform(image).unsqueeze(0).to(device)
# 获取卷积层的输出
conv_output = None
def hook_fn(module, input, output):
    global conv_output
    conv_output = output
# 注册钩子到最后一个卷积层
model.conv2.register_forward_hook(hook_fn)
```

```
# 前向传播
output = model(image)
target_class = output.argmax(dim=1).item()
print(f"Target class: {target_class}")
# 获取激活图
activations = conv_output.squeeze().detach().cpu()  # 形状是 [64, 8, 8]
# Score-CAM 计算
weights = []
for i, activation in enumerate(activations):
    # 对激活图进行上采样，使其尺寸与输入图像一致
    activation = torch.nn.functional.interpolate(
        activation.unsqueeze(0).unsqueeze(0),  # 增加 batch 和 channel 维度
        size=(28, 28),  # 上采样到 28x28
        mode='bilinear',  # 双线性插值
        align_corners=False
    ).squeeze()  # 去掉多余的维度
    # 将激活图与输入图像相乘
    masked_image = image * activation.to(device)
    # 前向传播计算得分
    output = model(masked_image)
    score = output[:, target_class].item()
    weights.append(score)
# 权重的统计信息
weights = np.array(weights)
# 计算加权激活图
heatmap = np.sum(weights[:, None, None] * activations.numpy(), axis=0)
# 对热力图进行上采样，使其尺寸与输入图像一致
heatmap = torch.nn.functional.interpolate(
    torch.tensor(heatmap).unsqueeze(0).unsqueeze(0).float(),  # 增加 batch 和
channel 维度
    size=(28, 28),  # 上采样到 28x28
    mode='bilinear',  # 双线性插值
    align_corners=False
).squeeze().numpy()
# 归一化热力图到 [0, 1] 范围
heatmap = (heatmap - np.min(heatmap)) / (np.max(heatmap) - np.min(heatmap) +
1e-10)
# 可视化热力图
plt.matshow(heatmap, cmap='viridis')
plt.title('Score-CAM Heatmap')
plt.colorbar()
plt.show()
```

这里给出对比图片


LIME Explanation
0
5
10
15
20
25
0
5
10
15
20
25


Grad-CAM++ Heatmap
0
1
2
3
4
5
6
7
0
1
2
3
4
5
6
7
1.0
0.8
0.6
0.4
0.2
0.0



可以发现通过三种可解释性方法都将可以解释，CNN在这张图片上都提取出什么用来分类了，但是从原理上来看：

- LIME 可以用于任何模型，不依赖于模型的结构，而且解释是像素级的，可以直观地看到哪些区域对模型的预测贡献最大。但是LIME 只能解释单个样本，无法提供模型整体的行为解释
- Grad-CAM 只需要一次前向传播和一次反向传播，计算效率高，可以解释模型对某一类别的整体行为。但是在复现的过程中出现了最后一次relu（heatmap）产生了梯度小和为负数的情况导致第一次产生的热度图是同一的0.00值，如果模型的梯度为 0 或较小，Grad-CAM 的解释效果较差。而且热力图分辨率受卷积层输出尺寸的限制，通常较低，这在这次复现的情况下也很明显。
- Score-CAM 通过对卷积层的激活图进行加权求和，生成热力图，与 Grad-CAM 不同不使用梯度，避免了梯度消失或爆炸的问题。Score-CAM 通过模型输出的得分加权，通常能生成更准确的热力图。而且需要对每个激活图进行前向传播，计算成本较高。