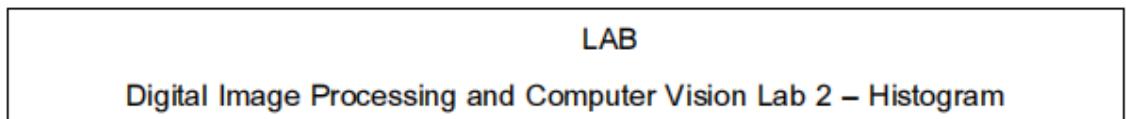
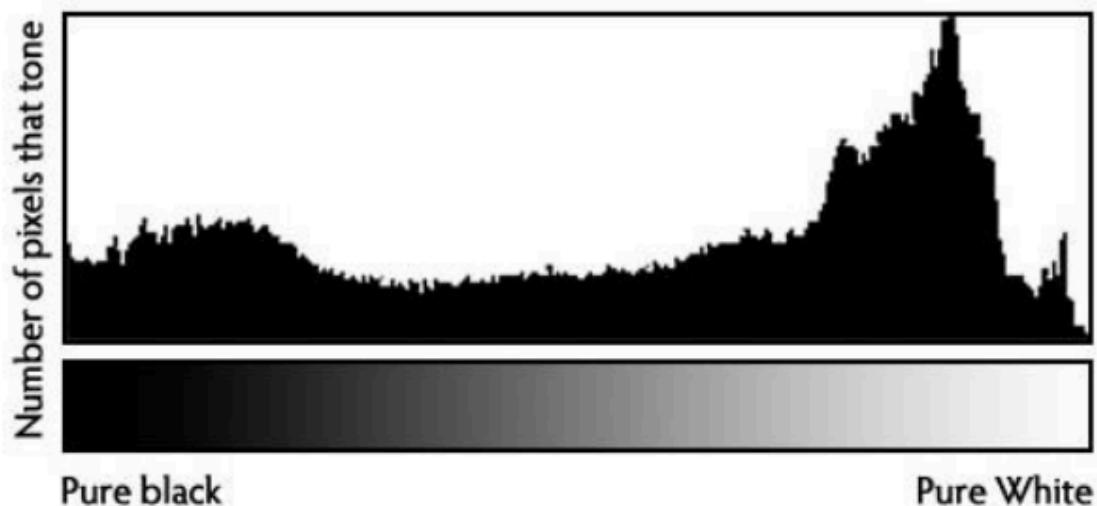


Lab 2

2.1 Histogram



1. Histogram



From the figure that shows the number of pixels in each brightness or the color value data R,G,B of the digital image in the grayscale image. The horizontal axis shows the brightness, which has a brightness of 0-255 (divided into 256 levels of color difference). The left side of the graph has a low brightness value (black), the image will be dark, close to black. The right side has a high brightness, the image will be bright, close to white. The middle of the graph shows the weight of the middle color. The vertical axis shows the number of pixels in each brightness, which has no boundary on the vertical axis. If the image is very dark, the graph will be piled up on the dark left side, without boundary.

Properties of Histogram

1. 标题：数字图像处理与计算机视觉实验2——图像直方图解析

在本次课件的内容中，我们主要探讨了「直方图 (Histogram)」在数字图像处理中的基本概念和性质。通过本次截图可以看到一张典型的灰度直方图示意图，以及对其在图像亮度分布上的说明。下面的笔记将详细拆解图片中涉及的所有知识点，并结合上下文进行深入分析，帮助你在无需额外课本的情况下也能对直方图有完整的理解。

2. 内容详解

1) 直方图的定义和图像示例

- 直方图 (Histogram) 是统计图像中各个灰度等级 (或颜色分量值) 出现频数 (像素数) 的一种图形表示方式。
- 图中显示了一张典型的灰度直方图。横轴通常代表像素的灰度等级 (亮度值)，范围从 0 (纯黑) 到 255 (纯白)。这是因为在 8 位灰度图像中，每个像素的灰度值可取 0~255，共 256 个离散等级。
- 纵轴代表像素数量 (频数)，即图像中每个灰度值所包含的像素数。

2) 横轴：从纯黑到纯白

- 在图中，最左端对应的灰度值接近 0，表示非常暗的像素 (接近纯黑)。
- 最右端对应的灰度值接近 255，表示非常亮的像素 (接近纯白)。
- 中间区域对应着中间调 (较为均衡的灰度值)。

3) 纵轴：像素数目

- 图中纵轴是“Number of pixels that tone”，用来统计并显示某一灰度值在图像中出现的频数。
- 由于实际的图像大小或每个灰度值出现的次数都可能很大，纵轴一般是一个相对值或可动态调节的范围，因此在理论上它并没有一个固定的最大上限 (“no boundary on the vertical axis”)。

4) 图形形状与图像亮度分布的关系

- 如果图像整体偏暗，那么大多数像素集中在灰度值较低的区间，对应的直方图会在左侧出现大量像素堆积。
- 如果图像整体偏亮，那么大多数像素集中在灰度值较高的区间，对应的直方图会在右侧有明显峰值。
- 若图像对比度比较丰富，各个灰度等级都有一定数量的像素，那么直方图会呈现较为分散的分布，左右两侧乃至中间都可能出现不同程度的峰值。
- 在本课件截图的示例中，直方图右侧有一个明显的峰值，说明图像中存在大量的较高亮度（偏白）像素；左侧和中间也存在一定分布，但与右侧的峰值相比相对较少，表明该图像可能整体偏亮，但并非完全缺失暗部或中间亮度。

5) 灰度图与彩色图像中 R、G、B 的联系

- 课件中提到“R, G, B of the digital image in the grayscale image”，这是在说明当我们把彩色图像转换为灰度图像时，也可以得到相应的灰度值（综合了 RGB 三个通道的数值）。
- 对于彩色图像，也可以对每个通道（R、G、B 分别）进行直方图统计，或者将它们先转换为灰度再做综合分析。
- 本图的示例是以灰度形式进行直方图统计：将所有像素按其灰度值 0-255 进行计数并绘制。

6) 直方图的典型性质

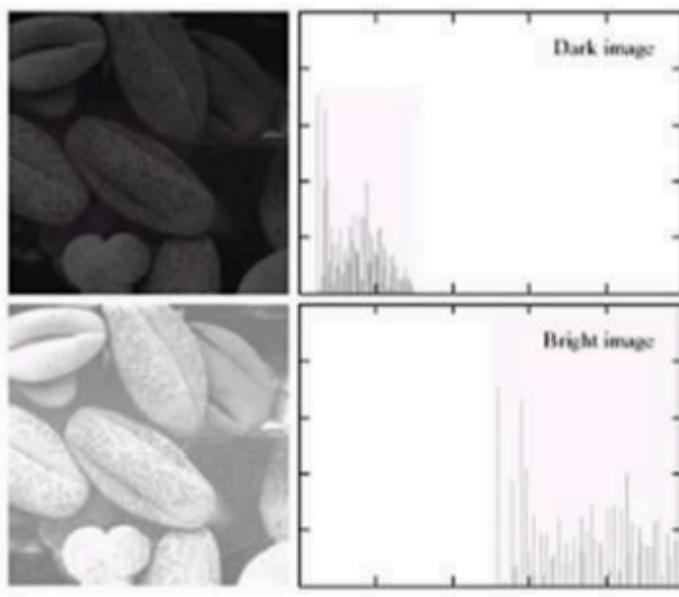
- **非负性：**直方图任何区间或灰度值处的像素数都不会是负数。
- **总像素数：**直方图所有区间（从 0 到 255）的像素数之和，必然等于整幅图像的总像素数。
- **范围可调：**横轴为 0~255 的离散灰度值区间；纵轴则根据图像大小可相应变化。
- **可视化对比度和亮度：**通过查看直方图，可以大致判断出图像是偏暗、偏亮，还是对比度不足（多数像素集中在某些狭窄区域）或对比度过大（像素分布非常分散等）。

7) 如果图像非常暗或非常亮的情形

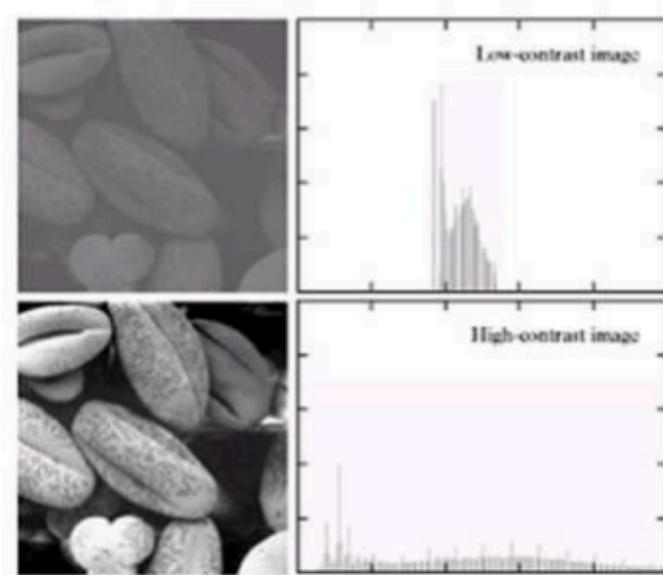
- 课件指出：“If the image is very dark, the graph will be piled up on the dark left side, without boundary.”这表示若图像过暗，直方图会在左端累积很高的峰值甚至贴近 0 灰度值；反之图像非常亮时，则会在右端（接近 255）累积。

3. 本次部分的总结

- 本截图最主要的内容是通过一个典型的灰度直方图说明了什么是直方图、它的横纵轴代表何种含义、以及图形形状和图像整体亮度分布之间的关系。
- 理解直方图的作用，对于后续在图像处理中的亮度调整、对比度增强（比如直方图均衡化）等都有重要意义。
- 总体而言，直方图可作为图像亮度和对比度特征的快速可视化手段，帮助我们直观地理解图像特性，并为后续处理操作提供参考依据。



If most of the distribution is on the left side of the graph, then the image is low in brightness. Conversely, if most of the distribution is on the right side of the graph, then the image is high in brightness.



If the distribution of the graph is narrow, it means that the image is a low-contrast image. If the distribution of the graph is wide or spread out throughout the graph, it means that the image is a high-contrast image.

Benefits of Histogram

1. 标题：灰度分布与图像亮度、对比度的深入解析

以下笔记围绕课件截图中对于“Dark image”“Bright image”“Low-contrast image”和“High-contrast image”四个示例以及对应的直方图展开，帮助你理解如何根据直方图分布来判断图像的亮度与对比度。通过阅读本笔记，你将能在无需课本的情况下，掌握判断图像明暗程度和对比度差异的基本方法。

2. 详细内容解析

1) 示例一：Dark image（暗图像）及其直方图

- **图像特点：**上半部分的图示显示了一幅偏暗的图像，可以看到大多数细节都淹没在深灰色甚至接近黑色的区域中，整体亮度不足。
- **直方图分布：**从右侧的直方图可以观察到，大部分像素值集中在灰度级较低（左侧）的位置，意味着该图像的亮度整体偏暗。
- **原因与效果：**
 - 灰度级越低，说明像素越接近黑色。
 - 当绝大部分像素堆积在左边时，图像整体呈深暗调。
- **总结：**通过这个示例可直观看到，若图像的亮度整体不足，直方图的“峰”会明显向左侧集中。

2) 示例二：Bright image（亮图像）及其直方图

- **图像特点：**下半部分的图示显示了一幅整体偏亮的图像，大多数区域接近白色或高亮灰度。
- **直方图分布：**直方图主要分布在较高灰度值的区域（右侧），说明大部分像素都属于亮度值较大的范围。
- **原因与效果：**
 - 灰度级越高，像素越接近白色。
 - 若图像偏亮，那么直方图会朝右边聚集，甚至出现明显峰值。
- **总结：**该示例说明了当图像整体非常亮时，直方图会向右侧集中，同理可反推若直方图偏右，则图像整体亮度偏高。

3) 示例三：Low-contrast image (低对比度图像) 及其直方图

- **图像特点：**第三张（左侧）图为低对比度图像，能够直观看到图中所有像素看起来灰蒙蒙的，暗部并不够暗，亮部也不够亮，整体层次不分明。
- **直方图分布：**从右侧直方图可以看出，其分布比较“狭窄”，集中在某一有限区间内，并没有覆盖太多的灰度范围。
- **原因与效果：**
 - 对比度低意味着像素亮度分布不够“拉开”，大多数像素都处于灰度级区间的某一小部分范围，缺少深黑或纯白的极端值。
 - 图像看起来灰暗、平淡，没有鲜明的明暗对比。
- **总结：**当直方图在横轴上仅占据很小的一段，说明像素的亮度范围被“压缩”，这是低对比度的典型特征。

4) 示例四：High-contrast image (高对比度图像) 及其直方图

- **图像特点：**第四张（左侧）图为高对比度图像，从视觉上可以看到，亮的区域很亮，暗的区域很暗，细节层次丰富。
- **直方图分布：**右侧直方图分布范围较广，几乎占据了从较低灰度值到较高灰度值的区段，说明该图像既有暗部也有亮部，中间层次也被明显拉开。
- **原因与效果：**
 - 对比度高意味着像素亮度分布在0~255之间更大范围地铺开，暗部更深，亮部更亮，中间也保持足够层次。
 - 这种图像往往层次丰富、视觉效果更鲜明。
- **总结：**当直方图在横轴上覆盖范围很广，说明不同灰度级的像素数相对均衡或分布跨度大，这是高对比度图像的标志。

5) 亮度与对比度的直方图判断总结

- **亮度：**
 - 主要根据直方图整体分布位置来判断：左侧集中=偏暗；右侧集中=偏亮。
- **对比度：**
 - 主要根据直方图横向分布的宽度来判断：范围窄=低对比度；范围宽=高对比度。
 - 两个指标可以结合起来看，一张图既有可能偏亮又低对比度，也有可能偏暗但高对比度，这些在实践中都可以通过直方图仔细分析。

3. 本次部分的总结

- 本课件通过展示四组对比图片和对应的直方图，形象地说明了如何根据灰度直方图的分布位置（左右）来判断图像的亮度，以及根据分布范围（窄宽）来判断图像的对比度高低。
- 在实际的数字图像处理过程中，直方图是判断和调整图像属性（如亮度、对比度、动态范围）的重要工具。理解这些概念对于后续学习直方图均衡化、伽马校正以及高级图像增强算法都至关重要。

1. The image viewed from the monitor may be distorted, but the information from the histogram will tell the true brightness and intensity of the image.
2. This information will help us choose a better shooting mode by helping to choose the compensation for the image's exposure when taking pictures in places with very high or low image brightness.
3. The information can be used to process and adjust the image.

procedure FillHistogram()

for each $b \in H$

$b \leftarrow b \times \alpha$

for each $x \in I$

if $x > T$

for each $b \in h(x)$

$b \leftarrow b + (1 - \alpha)$

Idea for histogram computation algorithm.

Function CreateHistogram(image):

If image is colored (RGB):

Separate the image into three channels: Red, Green, and Blue

Initialize three arrays to store pixel counts for each channel:

histogramRed[256] = {0}

histogramGreen[256] = {0}

histogramBlue[256] = {0}

For each pixel in the image:

Extract Red, Green, Blue values from the pixel

Increment histogramRed[Red value] by 1

Increment histogramGreen[Green value] by 1

Increment histogramBlue[Blue value] by 1

Return histogramRed, histogramGreen, histogramBlue

```
Else if image is grayscale:  
    Initialize an array to store pixel counts:  
        histogramGray[256] = [0]  
  
    For each pixel in the image:  
        Extract the pixel intensity value  
        Increment histogramGray[intensity value] by 1  
  
    Return histogramGray  
  
Else:  
    Print "Unsupported image format"  
    Return None  
  
End Function
```

Pseudocode for histogram computation algorithm.

1. 标题：直方图的应用、算法思路与示例伪代码详解

本次课件截图展示了基于直方图在图像处理中的进一步应用场景以及如何编写计算直方图的示例伪代码。我们既看到了在图像拍摄、后期处理中的价值，也看到了一个用于更新或填充直方图的伪代码示例以及如何对彩色/灰度图像分别进行直方图统计。以下笔记将结合截图中的内容对这些点进行深入解析。

2. 详细内容解析

2.1 直方图在图像处理中的作用和意义

1. 图像拍摄模式的选择

- 截图第1条内容指出，屏幕显示的图像可能会有失真，但通过直方图可以获取图像实际的亮度与强度分布。
- 截图第2条内容说明，直方图信息能够帮助我们在拍照时选择合适的曝光补偿设置，尤其是在极暗或极亮的场景下，避免图像过暗或过曝。

2. 图像后期处理

- 截图第3条内容提到，可以使用直方图对图像进行处理与调整，如亮度、对比度、动态范围的修正等。
- 这些操作都依赖于直方图准确地反映出图像中每个亮度（或颜色）级别所占像素数量的客观分布。

2.2 直方图填充/更新的过程: procedure FillHistogram()

在截图中可以看到一个示例过程 `FillHistogram()`，其伪代码逻辑如下：

```
css
```

```
procedure FillHistogram()
    for each b ∈ H
        b ← b × α

    for each x ∈ I
        if x > T
            for each b ∈ h(x)
                b ← b + (1 - α)
```

⑤ 复制

- 含义拆解：

- `H` 代表直方图 (Histogram) 的数据结构，可能是一组桶 (bins) 或者计数值。
- `b ← b × α`：这是对直方图的所有“桶”进行一次衰减操作，用 `α` (0到1之间的因子) 来缩放现有的直方图值。
- `for each x ∈ I if x > T`：依次遍历图像中的像素 `x`，如果 `x` 大于某个阈值 `T`，则对与之对应的直方图桶 `h(x)` 做相应更新。
- `b ← b + (1 - α)`：将被选中的桶在原有基础上增加一部分值，这里 `(1 - α)` 似乎表示每次更新时的“增量权重”。

- 可能的用途：

- 该算法看起来像是一种边计算边“淡化旧信息、加入新信息”的自适应或动态更新直方图的方法。
- 参数 `α` 控制了旧直方图在迭代更新过程中的保留程度。（若 `α` 较大，则保留的旧信息较多；若 `α` 较小，则新信息占比更大。）
- 阈值 `T` 的出现则表明这段代码只对超过一定值的像素进行统计或强调，可能是一种特定场景下的亮度/阈值分割判断，也可能用来检测特定范围的像素。

2.3 直方图计算的思路与伪代码: Function CreateHistogram(image)

截图中给出了一个更常见、更加“基础”的直方图生成伪代码，用于计算单幅图像的直方图。代码主要流程如下：

1. 判断图像类型

- 如果图像是彩色 (RGB)，则先将图像分离成三个通道：Red、Green、Blue；
- 如果图像是灰度图 (grayscale)，则只需要一个通道；
- 如果是其他格式，可能不支持或直接报错。

2. 初始化直方图数组

- 例如彩色图像时：
 - `histogramRed[256] = {0}`
 - `histogramGreen[256] = {0}`
 - `histogramBlue[256] = {0}`
- 灰度图像时：
 - `histogramGray[256] = {0}`

3. 遍历图像像素

- 对于彩色图像中的每个像素：
 1. 提取该像素的 Red、Green、Blue 分量（例如范围是0~255）。
 2. 分别累加到对应的直方图数组中，例如 `histogramRed[Red] += 1`，表示在红色分量为 Red 的桶中加一。
- 对于灰度图像中的每个像素：
 1. 提取该像素灰度值 Intensity (0~255)。
 2. `histogramGray[Intensity] += 1`。

4. 返回直方图

- 彩色图像则返回三个数组 (`histogramRed`, `histogramGreen`, `histogramBlue`) , 灰度图像则返回一个数组。
 - 总结:
 - 这是最传统、最核心的直方图计算思路：对每一个像素都要根据它的亮度/颜色值来更新计数。
 - 这些计数最终就能反映全图在每个亮度/颜色值范围上出现了多少像素。
-

3. 本次部分的总结

- 直方图信息的重要性：无论是拍照时的曝光补偿还是后期处理中的亮度对比度调节，直方图都能提供客观的数据支持。
 - 直方图的动态/自适应更新：`FillHistogram()` 提供了一个将旧直方图数据与新图像信息相融合的思路，可能应用于实时场景（如视频分析或在线图像处理）。
 - 直方图的基础计算：`createHistogram(image)` 的伪代码演示了如何对不同类型的图像进行简单、直接的直方图统计，以便后续进行可视化或分析。
 - 彩色与灰度的区别：对于彩色图像，需要分别统计 R、G、B 三个通道的直方图；而灰度图像只需单一通道。
-

```
from PIL import Image
import numpy as np

def create_histogram(image_path):
    # Load the image
    image = Image.open(image_path)
    data = np.array(image)

    if len(data.shape) == 3:
        # Initialize histograms
        histogramRed = np.zeros(256)
        histogramGreen = np.zeros(256)
        histogramBlue = np.zeros(256)

        # Update histograms
        for row in data:
            for pixel in row:
                r, g, b = pixel[:3]
```

```
        histogramRed[r] += 1
        histogramGreen[g] += 1
        histogramBlue[b] += 1

    return histogramRed, histogramGreen, histogramBlue

elif len(data.shape) == 2:
    # Initialize histogram
    histogramGray = np.zeros(256)

    # Update histogram
    for row in data:
        for pixel in row:
            histogramGray[pixel] += 1

    return histogramGray

else:
    print("Unsupported image format")
    return None

# Example usage:
histRed, histGreen, histBlue = create_histogram('path_to_color_image.jpg')
histGray = create_histogram('path_to_grayscale_image.jpg')
```

Coding for histogram computation algorithm. [Try running the code , then fix the code and take a screenshot]

1. 标题：使用 Python (PIL + NumPy) 计算图像直方图的代码解析

通过本次课件截图，我们看到了一个基于 Python 的示例函数 `create_histogram(image_path)`，用以加载图像并计算其直方图。该函数能够分别针对彩色图像 (RGB 三通道) 和灰度图像进行统计，示例中还展示了可能的使用方式 (Example usage)。下面的笔记将详细讲解截图代码的每个要点。

2. 详细内容解析

2.1 引入库与函数定义

python

④ 复制

```
from PIL import Image
import numpy as np
```

- **PIL** (Python Imaging Library, 现多使用 Pillow) 与 **NumPy** 是图像和矩阵处理的常用组合。
- `Image` 用于打开/加载图像文件, `np` 用于数组操作 (如创建、遍历等)。

python

④ 复制

```
def create_histogram(image_path):
    # Load the image
    image = Image.open(image_path)
    data = np.array(image)
```

- 定义函数 `create_histogram` 接受一个图像路径 `image_path`。
- `Image.open(image_path)` 打开图片文件并返回一个 PIL 图像对象。
- `np.array(image)` 将 PIL 图像对象转换为 NumPy 数组 `data`，便于后续处理。

2.2 判断图像类型并初始化直方图

python

复制

```
if len(data.shape) == 3:  
    # Initialize histograms  
    histogramRed = np.zeros(256)  
    histogramGreen = np.zeros(256)  
    histogramBlue = np.zeros(256)
```

- `data.shape` 返回图像数组的维度信息。
 - 若 `len(data.shape) == 3`，通常表示彩色图像（一般是 `[height, width, channels]` 的形式）。
 - 初始化三个数组各含 256 个元素，用于统计 R、G、B 三个通道从 0 ~ 255 这 256 个灰度/强度值所对应的像素数量。

2.3 更新直方图：彩色图像 (RGB)

python

复制

```
# Update histograms  
for row in data:  
    for pixel in row:  
        r, g, b = pixel[:3]  
        histogramRed[r] += 1  
        histogramGreen[g] += 1  
        histogramBlue[b] += 1  
  
return histogramRed, histogramGreen, histogramBlue
```

- 逐行遍历图像的每一个像素：
 - 对于彩色图像，每个 `pixel` 通常是 `[R, G, B]` 或 `[R, G, B, A]` (带透明度通道)。此处通过 `pixel[:3]` 只取前三个通道值 (即 R, G, B)。
 - `histogramRed[r] += 1`：若当前像素的红色分量是 `r`，则在 `histogramRed` 中把索引为 `r` 的位置加 1。相同道理应用于 G、B 通道。
- 最终返回三个直方图数组，分别对应红、绿、蓝分量的统计结果。

2.4 更新直方图：灰度图像

python

② 复制

```
elif len(data.shape) == 2:  
    # Initialize histogram  
    histogramGray = np.zeros(256)  
  
    # Update histogram  
    for row in data:  
        for pixel in row:  
            histogramGray[pixel] += 1  
  
    return histogramGray
```

- 若 `len(data.shape) == 2`，则图像数组只有两个维度 `[height, width]`，可视为灰度图像。
- 初始化 `histogramGray` 数组后，遍历所有灰度值 `pixel`（范围 0~255），相应地在直方图对应位置累加 1。
- 返回单独的灰度图直方图数组。

2.5 其他情况

```
python 复制
else:
    print("Unsupported image format")
    return None
```

- 如果图像的通道数并非 3 或者 2，则输出“Unsupported image format”，返回 `None`。
- 这表明本示例仅演示最常见的 RGB 和灰度格式，如需支持更多通道（如带 Alpha 通道或 CMYK）可自行扩展。

2.6 示例用法

```
python 复制
# Example usage:
histRed, histGreen, histBlue = create_histogram('path_to_color_image.jpg')
histGray = create_histogram('path_to_grayscale_image.jpg')
```

- 展示如何调用该函数分别处理彩色与灰度图像。
- 用户可以利用返回的直方图结果进行可视化（如使用 `matplotlib` 绘制）或进一步分析（如直方图均衡化）。

3. 本次内容总结

- **核心逻辑：**本函数首先判断图像是彩色还是灰度，然后针对不同情况分别统计像素的强度值。
- **彩色图像：**需要维护 R/G/B 三个直方图；灰度图像只需要一个直方图。
- **NumPy 的应用：**使用 NumPy 数组极大地简化了遍历和操作过程；在实际实现中，也可用更多矢量化操作进一步加速（如 `np.bincount`）。
- **适用范围：**该示例主要示范直方图统计的基本原理，对于大多数常规格式（RGB、灰度）足够直接使用，也为自定义或扩展方案提供了参考。

fix the code for grayscale images and display the results. :

下面给出一个示例完整代码，既“修复”了对灰度图像的处理方式，也演示了如何将结果可视化。思路如下：

1. 保证能够正确识别并统计灰度图像：

- 有些灰度图可能本身是二维数组 `(height, width)`，也有可能是 `(height, width, 1)`。
- 为了避免因图像模式或通道数出错，我们可以在读取图像后先判断其模式，如不是“L”（灰度模式），则转换为灰度。这样就能确保灰度图一定是二维数组。

2. 显示结果：

- 使用 `matplotlib` 画柱状图（或折线图）来可视化直方图。

下面的示例代码供你参考：

python

⑤ 复制

```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt

def create_histogram(image_path):
    """
    加载图像并计算其直方图。
    若为彩色(RGB)图像，返回三个数组(histRed, histGreen, histBlue);
    若为灰度图像，返回一个数组(histGray);
    否则返回 None。
    """
    # 1. 打开图像
    image = Image.open(image_path)

    # 2. 若图像不是灰度(L)也不是RGB，则可根据需要进行转换或直接报错
    #   这里演示：如果不是RGB就转为灰度
    if image.mode not in ('RGB', 'L'):
        image = image.convert('L')
```

```
# 3. 转换为numpy数组
data = np.array(image)

# 4. 根据数组形状判断是彩色还是灰度
if image.mode == 'RGB' and len(data.shape) == 3:
    # (height, width, 3)
    histogramRed = np.zeros(256, dtype=np.int64)
    histogramGreen = np.zeros(256, dtype=np.int64)
    histogramBlue = np.zeros(256, dtype=np.int64)
    # 遍历每个像素，统计RGB分量
    for row in data:
        for pixel in row:
            r, g, b = pixel[:3]
            histogramRed[r] += 1
            histogramGreen[g] += 1
            histogramBlue[b] += 1
    return histogramRed, histogramGreen, histogramBlue

elif image.mode == 'L' and len(data.shape) == 2:
    # (height, width)
    histogramGray = np.zeros(256, dtype=np.int64)
    for row in data:
        for pixel in row:
            histogramGray[pixel] += 1
    return histogramGray

else:
    print("Unsupported image format or dimension.")
    return None

def show_histogram(hist, title="Histogram", color="blue"):
    """
    将给定的直方图数组可视化。
    hist: 一维numpy数组，长度为256。
    """
    plt.bar(range(256), hist, color=color)
    plt.title(title)
    plt.xlabel("Intensity Value")
    plt.ylabel("Count")
    plt.show()
```

```

# ===== 使用示例 =====

# 1) 处理灰度图像
gray_hist = create_histogram('path_to_grayscale_image.jpg')
if gray_hist is not None and isinstance(gray_hist, np.ndarray):
    show_histogram(gray_hist, title="Grayscale Histogram", color="gray")

# 2) 处理彩色图像
rgb_hist = create_histogram('path_to_color_image.jpg')
if rgb_hist is not None and isinstance(rgb_hist, tuple):
    histRed, histGreen, histBlue = rgb_hist
    # 分别显示三个通道的直方图
    show_histogram(histRed, title="Red Channel Histogram", color="red")
    show_histogram(histGreen, title="Green Channel Histogram", color="green")
    show_histogram(histBlue, title="Blue Channel Histogram", color="blue")

```

代码说明

1. `image = Image.open(image_path)`

- 使用 Pillow 打开图像文件。

2. 模式判断与转换

- 如果图像是 `RGB`，我们就按彩色处理；如果是 `L`（灰度），就按灰度处理；如果是其他模式（例如 `RGBA`、`CMYK` 等），为了演示方便，这里先把它们一律转换为灰度模式 `L`。

3. 统计直方图

- 对于彩色图像：分别准备 3 个大小为 256 的数组 `histogramRed`, `histogramGreen`, `histogramBlue`；遍历每个像素的 (r, g, b) 分量并累加对应的桶。
- 对于灰度图像：准备 1 个大小为 256 的数组 `histogramGray`，遍历像素值并累加。

4. 可视化

- `show_histogram` 函数通过 `matplotlib.pyplot.bar` 绘制柱状图，可以方便地查看像素分布。对于灰度图，用灰色 (`color="gray"`)；对于不同通道，可以用对应的颜色标识。

运行上述代码后，你就能同时看到修复后的灰度图像直方图计算结果，以及彩色图像三个通道的直方图，并在窗口中分别显示出来。这样便完成了「修复灰度图像处理并显示结果」的练习。

修改后 2.1 完整代码：

```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt

def create_histogram(image_path):  # 用法
    """
    加载图像并计算其直方图。
    若为彩色(RGB)图像，返回三个数组(histRed, histGreen, histBlue)；
    若为灰度图像(L)或强制转换后的图像，返回一个数组(histGray)；
    若为其他格式且无法处理，则返回 None。
    """

    # 1. 打开图像
    image = Image.open(image_path)

    # 2. 若图像不是灰度(L)也不是RGB，则可根据需要进行转换或直接报错
    #   这里演示：如果不是RGB就转为灰度
    if image.mode not in ('RGB', 'L'):
        image = image.convert('L')

    # 3. 转换为numpy数组
    data = np.array(image)

    # 4. 根据数组形状和 image.mode 判断是彩色还是灰度
    if image.mode == 'RGB' and len(data.shape) == 3:
        # (height, width, 3) => 统计 RGB
        histogramRed = np.zeros(shape=256, dtype=np.int64)
        histogramGreen = np.zeros(shape=256, dtype=np.int64)
        histogramBlue = np.zeros(shape=256, dtype=np.int64)
        # 遍历每个像素，统计RGB分量
        for row in data:
            for pixel in row:
                r, g, b = pixel[:3]
                histogramRed[r] += 1
                histogramGreen[g] += 1
                histogramBlue[b] += 1
        return histogramRed, histogramGreen, histogramBlue

    elif image.mode == 'L' and len(data.shape) == 2:
        # (height, width) => 灰度图
        histogramGray = np.zeros(shape=256, dtype=np.int64)
        for row in data:
            for pixel in row:
                histogramGray[pixel] += 1
```

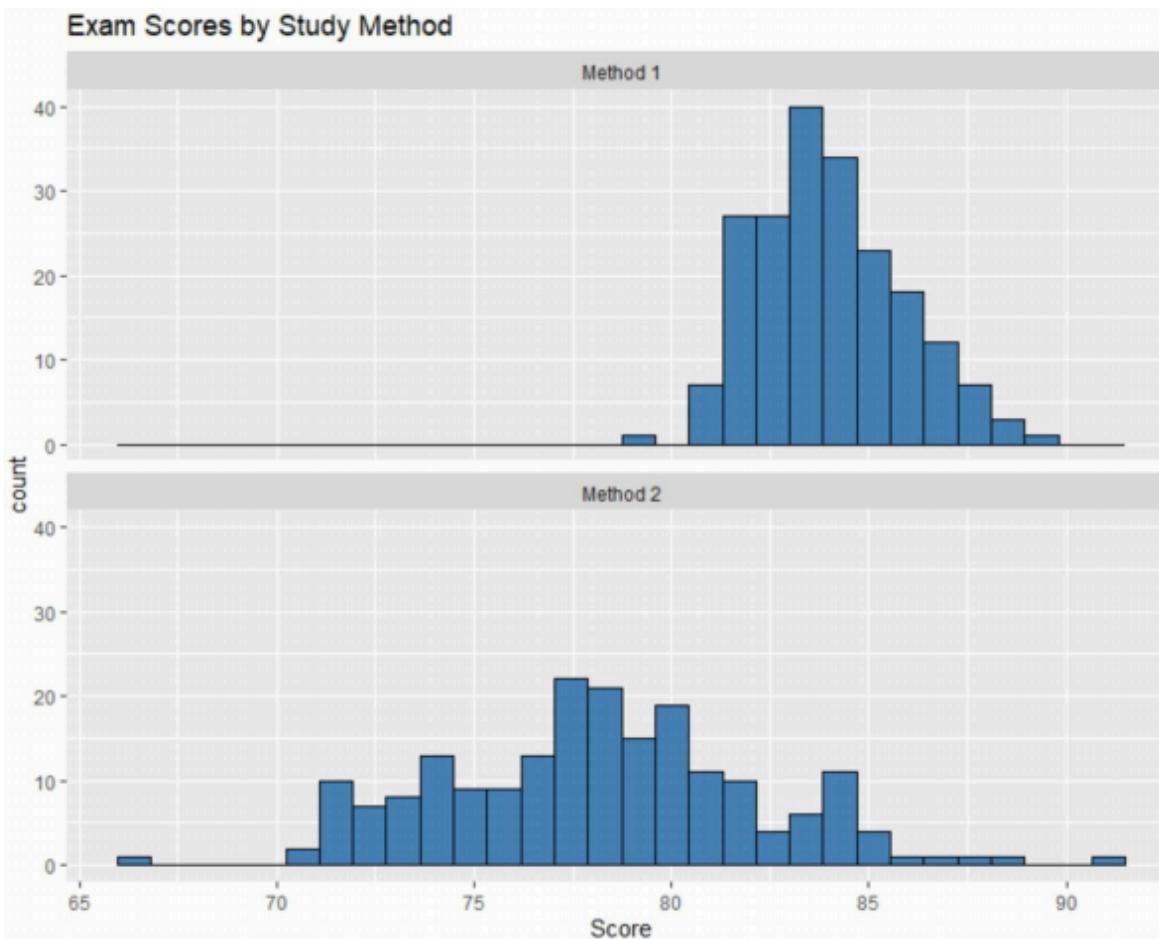
```
44         return histogramGray
45
46     else:
47         print("Unsupported image format or dimension.")
48         return None
49
50
51 def create_grayscale_histogram(image_path): 1个用法
52 """
53 演示：无论原图是什么模式，都强制转为灰度(L)后统计直方图。
54 """
55
56     image = Image.open(image_path).convert('L')
57     data = np.array(image)
58     histogramGray = np.zeros(shape=256, dtype=np.int64)
59     for row in data:
60         for pixel in row:
61             histogramGray[pixel] += 1
62     return histogramGray
63
64
65 def show_histogram(hist, title="Histogram", color="blue"): 5用法
66 """
67 将给定的一维直方图数组可视化（长度为256）。
68 """
69
70     plt.bar(range(256), hist, color=color)
71     plt.title(title)
72     plt.xlabel("Intensity Value")
73     plt.ylabel("Count")
74     plt.show()
75
76
77 # ===== 使用示例 =====
78
79 if __name__ == "__main__":
80
81     # 1) 验证图像是否真的是灰度
82     # 如果图像原本是RGB，但视觉上看似灰度，也会显示 "RGB"
83     test_image = Image.open('pic/grayscale.jpg')
84     print("测试图像模式:", test_image.mode)

# 2) 使用 create_histogram() 函数处理“灰度图像”
```

```
85     gray_hist = create_histogram('pic/grayscale.jpg')
86     if gray_hist is not None and isinstance(gray_hist, np.ndarray):
87         # 显示灰度图的直方图
88         show_histogram(gray_hist, title="Grayscale Histogram", color="gray")
89
90     # 3) 使用 create_histogram() 函数处理“彩色图像”
91     rgb_hist = create_histogram('pic/color.jpg')
92     if rgb_hist is not None and isinstance(rgb_hist, tuple):
93         histRed, histGreen, histBlue = rgb_hist
94         # 分别显示三个通道的直方图
95         show_histogram(histRed, title="Red Channel Histogram", color="red")
96         show_histogram(histGreen, title="Green Channel Histogram", color="green")
97         show_histogram(histBlue, title="Blue Channel Histogram", color="blue")
98
99     # 4) 演示“强制转灰度”的用法
100    forced_gray_hist = create_grayscale_histogram('pic/color.jpg')
101    show_histogram(forced_gray_hist, title="Forced Grayscale Histogram", color="gray")
```

完

2.2 Histogram Comparison



- To compare two histograms (H_1 and H_2), first we have to choose a metric ($d(H_1, H_2)$) to express how well both histograms match.
- OpenCV implements the function `cv::compareHist` to perform a comparison. It also offers 4 different metrics to compute the match:

1. Correlation (`CV_COMP_CORREL`)

$$d(H_1, H_2) = \frac{\sum_I (H_1(I) - \bar{H}_1)(H_2(I) - \bar{H}_2)}{\sqrt{\sum_I (H_1(I) - \bar{H}_1)^2 \sum_I (H_2(I) - \bar{H}_2)^2}}$$

where

$$\bar{H}_k = \frac{1}{N} \sum_J H_k(J)$$

and N is the total number of histogram bins.

2. Chi-Square (`CV_COMP_CHISQR`)

$$d(H_1, H_2) = \sum_I \frac{(H_1(I) - H_2(I))^2}{H_1(I)}$$

3. Intersection (`method=CV_COMP_INTERSECT`)

$$d(H_1, H_2) = \sum_I \min(H_1(I), H_2(I))$$

4. Bhattacharyya distance (`CV_COMP_BHATTACHARYYA`)

$$d(H_1, H_2) = \sqrt{1 - \frac{1}{\sqrt{\bar{H}_1 \bar{H}_2 N^2}} \sum_I \sqrt{H_1(I) \cdot H_2(I)}}$$

1. Start

2. Load Image1 and Image2
 - Ensure both images are in the same color space (e.g., RGB, grayscale)
3. Calculate Histogram for Image1
 - For each pixel in Image1:
 - Increment the histogram bin corresponding to the pixel's value
 - Normalize the histogram (so that the sum of all bins equals 1)
4. Calculate Histogram for Image2
 - Repeat steps similar to step 3 for Image2
5. Compare Histograms
 - Initialize similarity_measure to 0
 - For each bin in the histograms:
 - Calculate the difference between corresponding bins of Image1 and Image2
 - Optionally, use a similarity metric such as:
 - Chi-Squared Test
 - Intersection
 - Bhattacharyya Distance
 - Correlation
 - Update the similarity_measure based on the metric
6. Output the similarity_measure
 - A higher value typically indicates more similarity (depends on the metric used)
7. End

Pseudocode for histogram comparison algorithm.

Compare Histograms using the Chi-Squared Test

```

- Initialize chi_squared_value to 0
- For each bin in the histograms:
  - Calculate expected_value as the average of the corresponding bins from both
    histograms
  - If expected_value is not zero:
    - Calculate observed_value for Image1
    - Compute the squared difference between observed_value and
      expected_value
    - Divide the squared difference by expected_value
    - Add the result to chi_squared_value

```

Pseudocode for Chi-Squared Test algorithm.

1. 标题：直方图相似度度量及卡方检验 (Chi-Squared Test) 在直方图比较中的应用

这部分课件结合了“对考试成绩分布 (Method 1 和 Method 2)”的直方图示例，随后介绍了在图像处理或任意分布对比中常用的几种直方图比较度量方法，包括相关性 (Correlation)、卡方 (Chi-Square)、交集 (Intersection) 以及巴氏距离 (Bhattacharyya Distance)。最后还给出了“如何编写伪代码来对直方图进行比较”的步骤，并以 Chi-Squared (卡方检验) 为例展示了详细算法。下面的笔记将结合截图内容，按顺序为你进行完整解析。

2. 详细内容解析

2.1 课件示例：分数直方图及比较场景

1) Exam Scores by Study Method 1 & 2

- 从截图可看到两幅直方图：
 - Method 1**: 大部分分数集中在 80~85 附近，有少量分数分布在更低（70 左右）或更高（接近 90）的区间。
 - Method 2**: 整体分布比较分散，分数覆盖区间更大，从 65~95 都有出现，不过其中 75~85 之间显著集中。
- 这些柱状图实质上就相当于一个直方图 (histogram)，将“分数”做为横坐标，而“出现次数 (人数)”做为纵坐标进行统计。

2) 比较两组分数分布

- 课件指出，当我们有两个直方图 H_1 和 H_2 时，需要选定一个度量 (metric) 来表达二者的相似度或差异度 $d(H_1, H_2)$ 。
- 在图像处理 (如 OpenCV) 中，典型函数 `cv::compareHist` 就是用此类度量函数来比较不同的图像直方图分布。

2.2 常见的直方图比较度量

课件给出了四种常用度量的数学形式，这些方法同样可以应用在“分数分布”等一般统计场景中。

1) Correlation (相关性, CV_COMP_CORREL)

$$d(H_1, H_2) = \frac{\sum_I [(H_1(I) - \bar{H}_1)(H_2(I) - \bar{H}_2)]}{\sqrt{\sum_I (H_1(I) - \bar{H}_1)^2 \sum_I (H_2(I) - \bar{H}_2)^2}}$$

- 其中 \bar{H}_1 和 \bar{H}_2 分别表示直方图 H_1 和 H_2 的平均值， I 代表直方图的第 i 个 bin (或分数区间 / 灰度等级等)。
- 该公式类似皮尔森相关系数，值域一般为 $[-1, 1]$ ，数值越大表明分布越相似。

2) Chi-Square (卡方检验, CV_COMP_CHISQR)

$$d(H_1, H_2) = \sum_I \frac{(H_1(I) - H_2(I))^2}{H_1(I)}$$

- 当 $H_1(I)$ 和 $H_2(I)$ 相差大时，分母相对较小或出现极端情况，会使得该值迅速变大；表示差异明显。
- 需注意此处的形式与传统统计学中的卡方分布概念类似，但在实际使用中可有多种变形或加上平滑项。
- 值越小越相似（因为分母是 $H_1(I)$ ，若二者接近则差平方会相对较小）。

3) Intersection (交集, CV_COMP_INTERSECT)

$$d(H_1, H_2) = \sum_I \min(H_1(I), H_2(I))$$

- 取所有 bin 的最小值并累加。若两直方图在各 bin 上都有相近的数量，则 min 值较大，整个和也较大。
- 值越大越相似。对于归一化后的直方图，其范围一般在 $[0, 1]$ 之间。

4) Bhattacharyya Distance (巴氏距离, CV_COMP_BHATTACHARYYA)

$$d(H_1, H_2) = \sqrt{1 - \frac{1}{\sqrt{\bar{H}_1 \bar{H}_2}} \sum_I \sqrt{H_1(I) H_2(I)}}$$

- 它是度量两个概率分布相似度的常用方法；与交集概念有点相似，但采用了 $\sqrt{H_1(I) \cdot H_2(I)}$ 的形式来衡量重叠。
 - 值越小越相似。
-

2.3 直方图比较的伪代码流程

课件示例给出了一个通用的直方图比较算法伪代码，可总结为：

1. 同一颜色空间

- 加载图像（或分数序列）*Image1* 和 *Image2*，并确保二者处于相同的维度/格式（如同为 RGB、或者同为灰度、或者像考试成绩就都是“分数”）。

2. 计算直方图

- 对 **Image1**：遍历所有像素/数值，根据对应值更新直方图某个 bin 的计数；
- 归一化：即让所有 bin 的总和为 1，变成概率分布。
- 对 **Image2**：以相同方式计算并归一化。

3. 比较直方图

- 初始化某个相似度或差异度变量（如 `similarity_measure=0`）；
- 对每个 bin，计算 $H_1(I)$ 和 $H_2(I)$ 的差异，并根据所选的度量方法（卡方 / 交集 / 巴氏距离 / 相关系数等）更新 `similarity_measure`；
- 最后输出相似度或距离的数值。
- 注意，不同度量的数值含义不同：有的数值越大表示越相似（如交集、相关系数），有的越小表示越相似（如卡方、巴氏距离）。

2.4 Chi-Squared Test (卡方) 对比的算法思路

在伪代码中，还针对 Chi-Squared 提供了详细步骤：

1. 初始化 `chi_squared_value = 0`

2. 对直方图的每个 bin：

- 计算 “`expected_value`” = $\frac{H_1(I)+H_2(I)}{2}$ ，即两个直方图该 bin 值的平均数；
- 如果 `expected_value` 不为 0：

 1. `observed_value = H_1(I)` (有时也可取 $H_2(I)$ 或不等式看需求)

 2. 计算 $(\text{observed}_value - \text{expected}_value)^2$

 3. 将其除以 `expected_value` 并累加到 `chi_squared_value`

3. 输出 `chi_squared_value`

- 值越大，说明二者差距越大；值越小表明它们越相似。

注：在图像处理中，`cv::compareHist` 的卡方公式与此稍有不同（分母只用 $H_1(I)$ 或加平滑），但核心思想相近：越大越不相似。

3. 本次内容的总结

- 通过“Method 1 vs Method 2”的分数分布示例，课件引出了如何比较两个直方图的问题。
- 在图像处理、统计分析等场景下，经常需要选择合适的度量方式来衡量两个直方图的“差异”或“相似度”。
- 常见方法包括相关性、卡方、交集以及巴氏距离，它们适合不同的应用场景，各自有不同的数值意义（有些大值代表相似，有些大值代表差异）。
- 伪代码流程为：加载并同一化处理数据 → 分别计算直方图 → 选择度量公式 → 得出最终比较结果。
- 卡方检验（Chi-Squared）是其中较常见的一种，因为它对分布差异敏感，计算也相对直接。

练习：

下面是一份“从零到结果”的完整示例代码，演示如何对两幅图像的灰度直方图做归一化后，再按照课件的**Chi-Squared Test**伪代码算法进行比较并计算出卡方值。代码同时包含简要的“伪代码思路”说明，方便你将其扩展或嵌入到自己的工程中。

Chi-Squared Test：伪代码思路

课件中给出的伪代码可总结如下（以两个归一化直方图 H_1 和 H_2 为例）：

```
less ④ 复制
1. 初始化 chi_squared_value = 0
2. 对每个 bin i in [0, ..., N-1]:
   a) 计算 expected_value = (H1[i] + H2[i]) / 2
   b) 如果 expected_value != 0:
      observed_value = H1[i]    # 或根据需求选择 H2[i]
      diff = observed_value - expected_value
      chi_term = (diff^2) / expected_value
      chi_squared_value += chi_term
3. 返回 chi_squared_value
```

- 其中 $H_1[i]$ 和 $H_2[i]$ 通常是已经归一化到 $[0,1]$ 范围的概率分布。
 - 若 χ^2 值越小，则表示两个直方图越相似；反之则差异更大。
-

完整 Python 示例代码

以下代码做了这些事情：

1. 读取并强制转换两张图像为灰度模式；
2. 计算并归一化各自的灰度直方图；
3. 按照伪代码使用卡方公式比较两份直方图；
4. 在控制台打印最终的 Chi-Squared 值。

你也可以用同样的方法去比较彩色图像（对 R/G/B 三个通道分别计算并汇总），这里为了示例清晰，直接使用灰度流程。

python

④ 复制

```
import numpy as np
from PIL import Image

def compute_grayscale_histogram(image_path, bins=256):
    """
    读取图像后强制转为灰度，并计算其直方图。
    返回一个长度为bins的一维数组 hist，表示每个灰度级上出现的像素数量。
    """
    # 1) 打开并转换为灰度
    image = Image.open(image_path).convert('L')

    # 2) 转numpy数组
    data = np.array(image)

    # 3) 初始化直方图
    hist = np.zeros(bins, dtype=np.float64)

    # 4) 统计频次
    for row in data:
        for pixel in row:
            hist[pixel] += 1
```

```
# 5) 归一化 - 转为概率分布
total_pixels = hist.sum()
if total_pixels > 0:
    hist = hist / total_pixels

return hist

def chi_squared_test(hist1, hist2):
    """
    对两个“已归一化”的直方图进行Chi-Squared检验计算。
    伪代码参考：
    chi_squared_value = 0
    for i in range(len(hist1)):
        expected = (hist1[i] + hist2[i]) / 2
        if expected != 0:
            diff = hist1[i] - expected
            chi_squared_value += (diff * diff) / expected
    return chi_squared_value
    """

    chi_squared_value = 0.0
    for i in range(len(hist1)):
        expected = (hist1[i] + hist2[i]) / 2
        if expected != 0:
            diff = hist1[i] - expected
            chi_squared_value += (diff * diff) / expected

    return chi_squared_value

if __name__ == "__main__":
    # ===== 使用示例 =====
```

```
# 1) 分别计算两幅图像的灰度直方图
hist_imageA = compute_grayscale_histogram("imageA.jpg")
hist_imageB = compute_grayscale_histogram("imageB.jpg")

# 2) 对比: 卡方检验
chi_value = chi_squared_test(hist_imageA, hist_imageB)

# 3) 输出结果
print(f"Chi-Squared value between imageA & imageB = {chi_value:.4f}")

# 说明: 值越小, 表示两幅图像的灰度直方图分布越相近
```

代码说明

1. `compute_grayscale_histogram` 函数

- 打开图像并使用 `.convert('L')` 强制灰度;
- 创建一个大小为 256 的数组, 用来记录 0~255 各灰度级出现的次数;
- 将像素值逐一统计, 最后做“归一化”, 使直方图之和为1, 得到概率分布。

2. `chi_squared_test` 函数

- 直接对应伪代码: χ^2 值通过每个 bin 的期望值 $(H_1[i] + H_2[i])/2$ 来衡量二者差异;
- 如果 `expected` 不为 0, 则按公式 $\frac{(observed - expected)^2}{expected}$ 累加到最终的 `chi_squared_value`。

3. 主程序

- 分别对两幅图像计算直方图并归一化;
- 调用 `chi_squared_test()` 得到卡方值并输出。
- 你可以尝试替换不同图像, 或者把同一张图像对比自己来测试结果 (应当非常小或接近 0)。

完整代码：

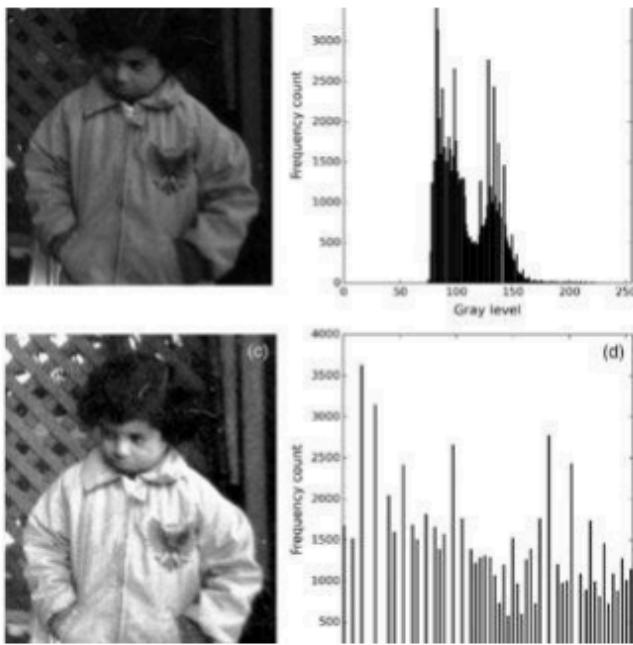
```
lab2.1.py      lab2.2.py ×

1 import numpy as np
2 from PIL import Image
3
4
5 def compute_grayscale_histogram(image_path, bins=256): 2 用法
6     """
7         读取图像(可能是RGB也可能是灰度), 统一强制转为灰度(L),
8         然后统计其直方图(频数), 并归一化为[0,1]的概率分布。
9         返回一个长度为bins的一维数组 hist。
10    """
11
12     # 1) 打开图像并强制转换为灰度(L)
13     image = Image.open(image_path).convert('L')
14
15     # 2) 转为 NumPy 数组
16     data = np.array(image)
17
18     # 3) 初始化直方图
19     hist = np.zeros(bins, dtype=np.float64)
20
21     # 4) 逐像素统计各灰度值(0~255)的出现次数
22     for row in data:
23         for pixel in row:
24             hist[pixel] += 1
25
26     # 5) 归一化 => 转为[0,1]区间的概率分布
27     total_pixels = hist.sum()
28     if total_pixels > 0:
29         hist = hist / total_pixels
30
31
32
33 def chi_squared_test(hist1, hist2): 1 个用法
34     """
35         依据课件中描述的Chi-Squared(卡方)检验伪代码,
36         比较两个“已归一化”的直方图的差异度。
37
38         chi_squared_value = 0
39         对于每个bin i:
40             expected = (hist1[i] + hist2[i]) / 2
41             若 expected != 0:
42                 diff = hist1[i] - expected
43                 chi_squared_value += (diff * diff) / expected
44
45         chi_squared_value = 0.0
```

```
45     chi_squared_value = 0.0
46
47     for i in range(len(hist1)):
48         expected = (hist1[i] + hist2[i]) / 2
49         if expected != 0:
50             diff = hist1[i] - expected
51             chi_squared_value += (diff ** 2) / expected
52
53     return chi_squared_value
54
55
56 ▶ if __name__ == "__main__":
57     # ===== 使用示例 =====
58
59     # 第一张图 (已是灰度图, pic/grayscale.jpg)
60     histA = compute_grayscale_histogram("pic/grayscale.jpg")
61
62     # 第二张图 (假设原本是RGB, 这里也能强制转成灰度)
63     histB = compute_grayscale_histogram("pic/grayscale2.jpg")
64
65     # 进行卡方检验, 比较两张图像(灰度化后)的分布差异
66     chi_value = chi_squared_test(histA, histB)
67
68     # 打印结果
69     print(f"Chi-Squared value = {chi_value:.6f}")
70
71     # 说明:
72     # 值越小 => 两张图在灰度直方图分布上越相似
73     # 值越大 => 二者差异越明显
```

完

2.3. Histogram Equalization



Adjusting the contrast of an image by modifying pixel values often utilizes histogram equalization. This technique aims to redistribute the pixel intensities across the entire range, making the image appear more visually dynamic.

Advantages:

- Improved Contrast: Histogram equalization effectively enhances the contrast of images with limited dynamic range. This can make details more visible in both dark and bright areas.
- Enhanced Visibility: By stretching the histogram, the technique can reveal hidden details that might have been obscured due to poor contrast.

Disadvantages:

- Over-Enhancement: In some cases, histogram equalization can over-enhance the image, leading to unnatural or exaggerated contrast. This can result in a loss of subtle details and create an artificial appearance.
- Distortion of Local Contrast: The method may not preserve local contrast effectively. This can lead to a loss of fine details and textures within specific regions of the image.
- Unpredictable Results: The outcome of histogram equalization can be unpredictable. It may not always produce visually pleasing results, especially for images with already good contrast.

Key Points:

- Cumulative Distribution Function (CDF): While the CDF plays a crucial role in the mathematical implementation of histogram equalization, it's not directly used for "balancing" the image in the way you described.

- "Dynamic" is a relative term: The perceived "dynamism" of an image is subjective and depends on the specific image content and the desired visual effect.

I hope this clarified the concept of histogram equalization and its impact on image contrast!

```
import cv2 as cv
import matplotlib.pyplot as plt

img = cv.imread("data/test_2.jpg")
if img is None:
    print('Could not open or find the image')
    exit(0)

## [Resize image]
scale_percent = 100 # percent of original size
width = int(img.shape[1] * scale_percent / 100)
height = int(img.shape[0] * scale_percent / 100)
dim = (width, height)
src= cv.resize(img, dim, interpolation = cv.INTER_AREA)

## [Convert to grayscale]
src = cv.cvtColor(src, cv.COLOR_BGR2GRAY)
## [Convert to grayscale]

## [Apply Histogram Equalization]
dst = cv.equalizeHist(src)
## [Apply Histogram Equalization]

## [Display results]
cv.imshow('Source image', src)
cv.imshow('Equalized Image', dst)
## [Display results]

## [Wait until user exits the program]
cv.waitKey()
## [Wait until user exits the program]
```

1. 标题：直方图均衡化 (Histogram Equalization) 及其在图像增强中的应用

本次课件围绕“直方图均衡化”这一常见的图像处理技术展开，展示了通过重新分配图像像素的灰度/强度值，从而提升图像整体对比度与可视化效果的过程。下面的笔记将分步骤详细解析截图中的内容，包括基本概念、优缺点、关键点以及基于 OpenCV 的实现示例。

2. 详细内容解析

2.1 直方图均衡化的概念

1) 整体思路

- 直方图均衡化是一种通过修改像素灰度值分布来提升对比度的方法，旨在让原本“分布狭窄”的直方图尽量利用整个动态范围 (0~255)。
- 简言之，它会将像素值在可能的范围内“拉伸/重新分配”，使暗部更暗、亮部更亮，或让隐藏在中间区域的细节更易被人眼察觉。

2) 原理概述

- 核心在于**累计分布函数 (CDF)**的使用：将每个灰度级出现的概率依次累加，然后根据该累积概率来映射到新的像素强度。
- 虽然 CDF 在数学实现中扮演重要角色，但课件也提醒，“CDF 并不直接等同于我们‘平衡图像’的过程”，它是工具而不是目的。

3) 示例对比

- 课件示例中，左侧原图对比度不足，右侧经均衡化后，亮暗区域的细节都得到一定增强，可见人物衣服等细节变得更清晰。

2.2 直方图均衡化的优点与缺点

1) 优点

- **对比度提升：**对于动态范围受限的图像，直方图均衡化能让暗区和亮区都获得更多层次，从而可视化效果更好。
- **细节增强：**在低对比度区域中可能被“淹没”的细节，经过均衡化后可得到突出。

2) 缺点

- **过度增强：**某些场景下，均衡化会使图像显得过分锐利或对比度失真，出现不自然或人工痕迹。
- **局部对比度损失：**该方法对整幅图像采取统一处理，不一定保留局部对比度，容易丢失原本微妙的纹理或细节。
- **结果不可预测：**对原本对比度较好的图像继续做均衡化，可能反而使结果变差或产生噪点。

3) 关键提示

- **CDF 的作用：**它在数学实现中不可或缺，但均衡化并非直接将图像“平衡”到平均亮度，而是重新分配像素值以拉伸直方图。
- **“动态”概念相对性：**用户感受的“动态”或“视觉冲击感”往往与场景内容及主观期望相关，并无绝对标准。

2.3 OpenCV 中的直方图均衡化示例

课件给出了用 OpenCV (cv2) 实现直方图均衡化的代码流程。要点如下：

1) 导入库

```
python  
  
import cv2 as cv  
import matplotlib.pyplot as plt
```

⑤ 复制

- `cv2`：OpenCV 库，用于图像读写、处理；
- `matplotlib`：用于可视化（如想绘制图像、直方图等）。

2) 读取与预处理

```
python  
  
img = cv.imread("data/test_2.jpg")  
if img is None:  
    print('Could not open or find the image;')  
    exit(0)
```

⑤ 复制

- `cv.imread()` 读取原始图像，若失败则退出。
- 课件演示了还可以根据需要缩放图像 (`cv.resize()`)，以下是常见的写法：

```
python  
  
scale_percent = 100  
width = int(img.shape[1] * scale_percent / 100)  
height = int(img.shape[0] * scale_percent / 100)  
dim = (width, height)  
src = cv.resize(img, dim, interpolation = cv.INTER_AREA)
```

⑤ 复制

3) 转换为灰度

```
python  
  
src = cv.cvtColor(src, cv.COLOR_BGR2GRAY)
```

⑤ 复制

- 直方图均衡化通常对灰度图像进行，因此先从 BGR 转为单通道灰度。

4) 应用直方图均衡化

```
python  
  
dst = cv.equalizeHist(src)
```

⑤ 复制

- `cv.equalizeHist()` 是 OpenCV 自带的均衡化函数，输入灰度图像 `src`，输出增强后的结果 `dst`。

5) 结果展示

```
python  
  
cv.imshow('Source image', src)  
cv.imshow('Equalized Image', dst)  
cv.waitKey(0)
```

⑤ 复制

- 弹出窗口分别显示原图与均衡化后的效果；
- `cv.waitKey(0)` 等待键盘按下才关闭图窗。

2.4 补充说明

- 如果想要可视化均衡化前后的直方图，可在 Python 里使用 `matplotlib` 先计算并绘制两幅图的灰度直方图曲线（或柱状图），再观察其分布拉伸情况。
- 对于彩色图像，简单做法是先转为灰度统一处理。如果想对彩色三通道分别均衡，需要对每个通道（B、G、R）独立做 `equalizeHist()`，但这在某些情况下会导致色彩失真，需要更复杂的色彩空间变换（如 YCrCb 模式下仅对亮度通道做均衡）。

3. 本次部分的总结

- 直方图均衡化概念：重新分配像素灰度值，使图像的对比度得以整体提升；
- 优缺点：有效增强暗部或亮部细节，但可能过度增强或破坏局部对比，结果具有不确定性；
- OpenCV 实现：`cv.equalizeHist()` 一行即可完成灰度图均衡化，简单高效。
- 应用场景：适合改善暗光下拍摄或对比度低的图像；但不一定适用于已有良好对比度的场景，需实际观测和选择。

give a code for histogram equalization :

完整代码：

```
lab2.1.py lab2.2.py lab2.3.py x
1 import cv2 as cv
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 def main(): 1个用法
6     # 1) 读取图像 (以灰度模式)
7     # 若原图是彩色，会被自动转换为单通道
8     img = cv.imread('pic/grayscale2.jpg', cv.IMREAD_GRAYSCALE)
9     if img is None:
10         print("Error: Could not open or find 'pic/grayscale.jpg'.")
11         return
12
13     # 2) 进行直方图均衡化
14     equalized = cv.equalizeHist(img)
15
16     # 3) 用 OpenCV 的窗口显示处理结果
17     cv.imshow(winname='Original Image', img=img)
18     cv.imshow(winname='Equalized Image', equalized)
19
20     # 4) 计算并可视化直方图 (使用 Matplotlib)
21     # 计算直方图
22     hist_original = cv.calcHist([img], [0], None, [256], [0,256])
23     hist_equalized = cv.calcHist([equalized], [0], None, [256], [0,256])
24
25     # 画出直方图对比
26     plt.figure(figsize=(10,5))
27
28     # 原图直方图
29     plt.subplot(*args: 1, 2, 1)
30     plt.title('Original Histogram')
31     plt.xlabel('Gray Level')
32     plt.ylabel('Pixel Count')
33     plt.plot(*args: hist_original, color='gray')
34
35     # 均衡化后直方图
36     plt.subplot(*args: 1, 2, 2)
37     plt.title('Equalized Histogram')
38     plt.xlabel('Gray Level')
39     plt.ylabel('Pixel Count')
40     plt.plot(*args: hist_equalized, color='black')
41
42     plt.tight_layout()
43     plt.show()
44
45     # 5) 等待按键再关闭所有窗口
46     cv.waitKey(0)
47     cv.destroyAllWindows()
48
49 D  if __name__ == "__main__":
50      main()
```

完