

RGB → YUV 轉換and UV 次取樣之影像品質分析

#多媒體作業二

撰寫者：黃教丞

學號：B1128019

日期：2024/12/16

一、前言 (Background and Introduction)

在數位影像處理與壓縮中，常使用 YUV 色彩空間來分離明亮度 (Y) 與色差訊號 (U、V)。藉由對 U、V 進行空間下取樣(downsampling)，可以減少影像所需的資料量，以達到壓縮的效果。本實驗將以 512x512 的 RAW 彩色影像 (RGB 格式) 為例，將 RGB 轉換至 YUV，然後對 U、V 通道進行降解析度處理，再將其上取樣 (upsampling) 還原為原始大小，最後重新轉回 R'G'B'，並觀察原圖與重建後影像的差異。

二、目的 (Objectives)

1. 掌握 RGB→YUV 轉換的原理與實作方法。
2. 了解色差通道 (U、V) 下取樣及上取樣的程序與概念。
3. 觀察並比較原始影像與經過 UV 次取樣後重建影像的品質差異。
4. 使用多張影像(lena.raw、baboon.raw以及另外2~3張自選影像)來比較轉換與壓縮後的效果。

三、素材與環境說明 (Materials and Setup)

- 影像檔案：
 - lena.raw (512x512, 24-bit RGB)
 - baboon.raw (512x512, 24-bit RGB)
 - 額外補充的兩張影像檔 (Hw_image_1.raw、Hw_image_2.raw, 皆是512x512 24-bit RGB 的RAW檔案)
- 開發環境：
 - Python 3.9.21
 - NumPy 函式庫 (處理影像陣列)
 - Pillow (PIL) 函式庫 (影像讀寫與儲存)
- 作業需求：
 - 將 $RGB \rightarrow YUV \rightarrow U, V$ 下採樣 (由 $512x512$ 降為 $256x256$) → 上採樣回 $512x512 \rightarrow YU'V' \rightarrow R'G'B'$ 。
 - 輸出 R, G, B, Y, U, V, U', V' 等通道影像 (共 10 張圖) 並儲存為 PNG 或 RAW。
 - 至少對 lena.raw 與 baboon.raw 測試，並額外使用至少兩個影像檔比較結果。

四、色彩空間轉換原理 (Color Space Conversion Theories)

RGB→YUV (BT.601) :

$$\begin{aligned} Y &= 0.299R + 0.587G + 0.114B \\ U &= -0.14713R - 0.28886G + 0.436B + 128 \\ V &= 0.615R - 0.51499G - 0.10001B + 128 \end{aligned}$$

上述公式將 R,G,B (範圍0~255) 轉為 Y,U,V (範圍0~255)。

YUV→RGB (BT.601反轉換) :

$$\begin{aligned} R &= Y + 1.13983(V-128) \\ G &= Y - 0.39465(U-128) - 0.58060(V-128) \\ B &= Y + 2.03211(U-128) \end{aligned}$$

五、流程與步驟 (Procedures)

1. 讀取 RAW 影像：

使用 Python 開啟 512x512 RGB 的 raw 檔案。此 raw 格式為無壓縮、無 header 的純色彩資料，以 raster scan 方式儲存，每個 pixel 的順序為 R、G、B 連續。

2. RGB → YUV：

根據公式將 RGB 通道轉換為 Y、U、V 通道。

3. U、V Downsampling：

將 U、V 從 512x512 降為 256x256，如採用最簡單方法，每隔一行一列取樣 (例如採樣左上像素)。

4. U、V Upsampling：

將降采樣後的 U'、V' 以鄰近點插值法(Nearest Neighbor)回復為 512x512。

5. Y、U'、V' → R'G'B'：

使用逆轉換公式將經下取樣與上取樣處理的 Y、U'、V' 重建回 R'G'B' 影像。

6. 輸出結果：

- 將 R,G,B,Y,U,V,U_sub,V_sub,U_up,V_up 等通道以 PNG 格式輸出。
- 將 R,G,B,Y,U,V,U_sub,V_sub,U_up,V_up 分別以 RAW 格式輸出。
- 計算或觀察原始 RGB 與重建 R'G'B' 的差異 (可選擇輸出差異圖差分，或計算 MSE/PSNR)。

根據輸出結果呢，我將要輸出的圖片做了以下解釋：

1. R Channel (Grayscale)

• 描述：

- R 通道 (紅色通道) 提取了 RGB 原圖中的紅色分量，並將其視覺化為灰階圖像。

- 亮度高的區域表示該像素的紅色分量強度較高，暗的區域則表示紅色分量較弱。

- **用途：**

- 分析紅色分量在圖像中的分佈特性，為色彩處理和壓縮提供基礎。
-

2. G Channel (Grayscale)

- **描述：**

- G 通道 (綠色通道) 提取了 RGB 原圖中的綠色分量，並將其視覺化為灰階圖像。
- 綠色分量是 RGB 中最接近人眼感知亮度的成分，通常能顯示更多細節。

- **用途：**

- 幫助評估綠色在圖像中的影響，特別是在與亮度相關的應用中（如 YUV 的 Y 分量計算）。
-

3. B Channel (Grayscale)

- **描述：**

- B 通道 (藍色通道) 提取了 RGB 原圖中的藍色分量，並將其視覺化為灰階圖像。
- 藍色分量通常在圖像中所占比重較少，因此亮度分佈可能更暗。

- **用途：**

- 用於分析藍色分量的影響，特別是在處理天空或水體等藍色佔主導的場景時。
-

4. Y Channel

- **描述：**

- Y 通道 (亮度通道) 是根據 BT.601 標準公式從 RGB 計算得出，表示圖像的亮度信息：
$$Y = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$$
$$Y = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$$
- 該通道由綠色分量主導，並結合了紅色和藍色的少量信息。

- **用途：**

- 在 YUV 色彩空間中，Y 通道是最重要的分量，對人眼感知的圖像清晰度有最大影響。
-

5. U Channel

- **描述：**

- U 通道 (色差藍) 是根據 BT.601 標準從 RGB 計算得出，反映藍色色度的信息：
$$U = -0.14713 \cdot R - 0.28886 \cdot G + 0.436 \cdot B + 128$$
- 值範圍為 0~255，128 表示中性，偏低的值表示更多黃色分量，偏高的值表示更多藍色分量。

- **用途：**

- 用於壓縮藍色色度的訊號，並在解碼時幫助還原原始圖像的藍色分量。
-

6. V Channel

- **描述：**

- V 通道 (色差紅) 是根據 BT.601 標準從 RGB 計算得出，反映紅色色度的信息：
$$V = 0.615 \cdot R - 0.51499 \cdot G - 0.10001 \cdot B + 128$$
- 值範圍為 0~255，128 表示中性，偏低的值表示更多青色分量，偏高的值表示更多紅色分量。

- **用途：**

- 用於壓縮紅色色度的訊號，並在解碼時幫助還原原始圖像的紅色分量。
-

7. U Subsampled

- **描述：**

- 透過次取樣技術對 U 通道進行壓縮，將解析度從 512x512 降為 256x256。
- 每 2 行和 2 列取一個像素作為樣本，表示色度訊號的壓縮過程。

- **用途：**

- 減少藍色色度訊號的存儲和傳輸成本，符合 YUV 4:2:0 壓縮標準。
-

8. V Subsampled

- **描述：**

- 透過次取樣技術對 V 通道進行壓縮，將解析度從 512x512 降為 256x256。
- 方法與 U 通道類似，壓縮了紅色色度訊號。

- **用途：**

- 減少紅色色度訊號的存儲和傳輸成本，符合 YUV 4:2:0 壓縮標準。
-

9. U Upsampled

- **描述：**
 - 將次取樣後的 U 通道使用最近鄰插值法恢復到 512x512 的解析度。
 - 恢復過程中引入了馬賽克效應，因為丟失的細節無法通過簡單插值復原。
 - **用途：**
 - 模擬壓縮後色度訊號在解壓縮過程中的重建情況。
-

10. V Upsampled

- **描述：**
 - 將次取樣後的 V 通道使用最近鄰插值法恢復到 512x512 的解析度。
 - 恢復過程中也會出現馬賽克效應，無法恢復紅色色度的全部細節。
- **用途：**
 - 模擬壓縮後色度訊號在解壓縮過程中的重建情況。

7. 延伸實驗(提高完成度)：

- 可以嘗試多次下取樣、或減少 U、V 的量化精度，以觀察畫質遞減的情形。
- 對多張影像進行同樣實驗，分析不同影像特性對於下取樣影響。

六、程式碼展示 (Code Demo)

以下為第一個主要程式碼 (總計有四張圖片要做轉換以及重建，我以 **baboon.raw** 為例)：

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

# ---- 基本參數設定 ----
width = 512
height = 512
input_filename = 'baboon.raw' # 請換成實際的檔名 (512x512 RGB24bit)
channel_order = 'RGB' # 根據實際情況設置 'RGB' 或 'BGR'

# ---- 讀取 RAW 檔案 (Sequential RGB 格式) ----
with open(input_filename, 'rb') as f:
    raw_data = np.frombuffer(f.read(), dtype=np.uint8)

# 將 R、G、B 通道分開 (Sequential 格式)
R = raw_data[:width*height].reshape((height, width)).astype(np.float32)
G = raw_data[width*height:2*width*height].reshape((height,
width)).astype(np.float32)
B = raw_data[2*width*height:3*width*height].reshape((height,
width)).astype(np.float32)
```

```

B = raw_data[2*width*height:].reshape((height, width)).astype(np.float32)

# 合併為 RGB 圖像
if channel_order == 'RGB':
    rgb_image = np.stack([R, G, B], axis=2).astype(np.uint8)
else:
    raise ValueError("Unsupported channel order")

# ---- RGB -> YUV (BT.601) ----
# Y, U, V range 在此為 0~255
Y = 0.299 * R + 0.587 * G + 0.114 * B
U = -0.14713 * R - 0.28886 * G + 0.436 * B + 128
V = 0.615 * R - 0.51499 * G - 0.10001 * B + 128

# 限制範圍為 0~255 並轉為 uint8
Y = np.clip(Y, 0, 255).astype(np.uint8)
U = np.clip(U, 0, 255).astype(np.uint8)
V = np.clip(V, 0, 255).astype(np.uint8)

# ---- U, V Downsampling (2x2) ----
U_sub = U[::2, ::2] # size: 256 x 256
V_sub = V[::2, ::2] # size: 256 x 256

# ---- U', V' Upsampling (Nearest Neighbor) ----
U_up = U_sub.repeat(2, axis=0).repeat(2, axis=1)
V_up = V_sub.repeat(2, axis=0).repeat(2, axis=1)

# ---- YUV -> RGB (重建 R'G'B') ----
Yf = Y.astype(np.float32)
Uf = U_up.astype(np.float32)
Vf = V_up.astype(np.float32)

# 使用 BT.601 逆轉換公式
R_rec = Yf + 1.13983 * (Vf - 128)
G_rec = Yf - 0.39465 * (Uf - 128) - 0.58060 * (Vf - 128)
B_rec = Yf + 2.03211 * (Uf - 128)

# 限制範圍為 0~255 並轉為 uint8
R_rec = np.clip(R_rec, 0, 255).astype(np.uint8)
G_rec = np.clip(G_rec, 0, 255).astype(np.uint8)
B_rec = np.clip(B_rec, 0, 255).astype(np.uint8)

# 合併為重建的 RGB 圖像
rgb_reconstructed = np.stack([R_rec, G_rec, B_rec], axis=2)

# ---- 在 Jupyter Notebook 中橫向排列展示圖片 ----
def display_images_in_two_rows(images, titles, fig_size=(20, 10)):
    """
    將多張圖片分為兩排排列展示
    :param images: 圖片列表，每張為 numpy array 格式
    """

```

```

:param titles: 標題列表，對應每張圖片
:param fig_size: 整體圖表大小
.....
num_images = len(images)
num_cols = (num_images + 1) // 2 # 每排的圖片數量（向上取整）
plt.figure(figsize=fig_size)
for i, (image, title) in enumerate(zip(images, titles)):
    # 設定為兩行，每行 num_cols 張圖片
    plt.subplot(2, num_cols, i + 1)
    if len(image.shape) == 3: # RGB 圖像
        plt.imshow(image)
    else: # 灰階圖像
        plt.imshow(image, cmap='gray')
    plt.title(title)
    plt.axis('off')
plt.tight_layout() # 自動調整間距
plt.show()

```

```

images = [R, G, B, Y, U, V, U_sub, V_sub, U_up, V_up, rgb_image,
rgb_reconstructed]
titles = [

```

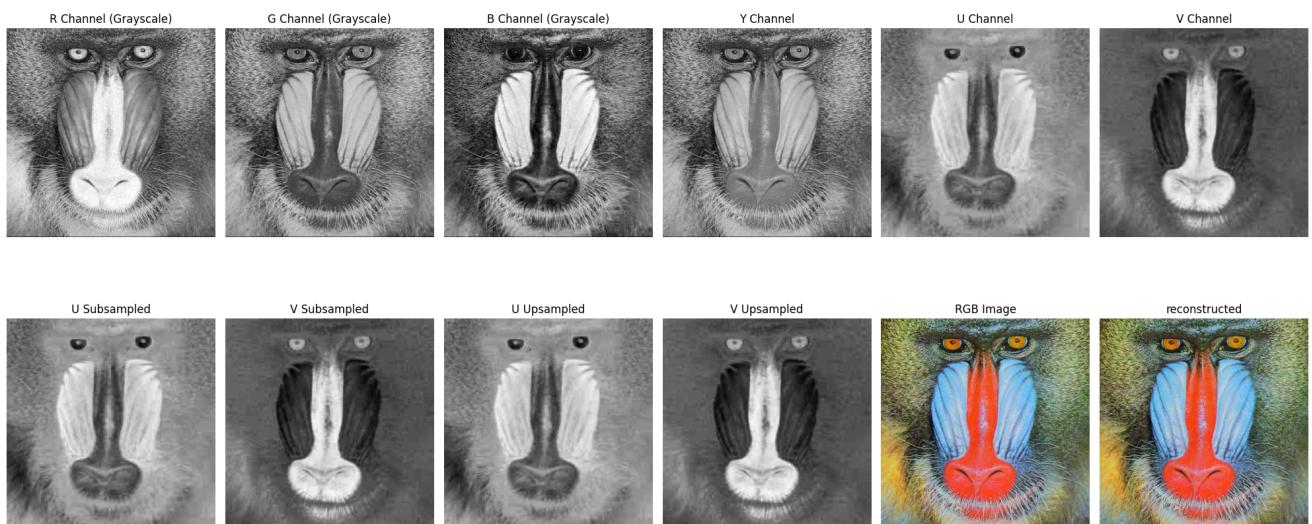
```

    "R Channel (Grayscale)", "G Channel (Grayscale)", "B Channel
    (Grayscale)",
    "Y Channel", "U Channel", "V Channel", "U Subsampled",
    "V Subsampled", "U Upsampled", "V Upsampled", "RGB
    Image", "reconstructed"
]

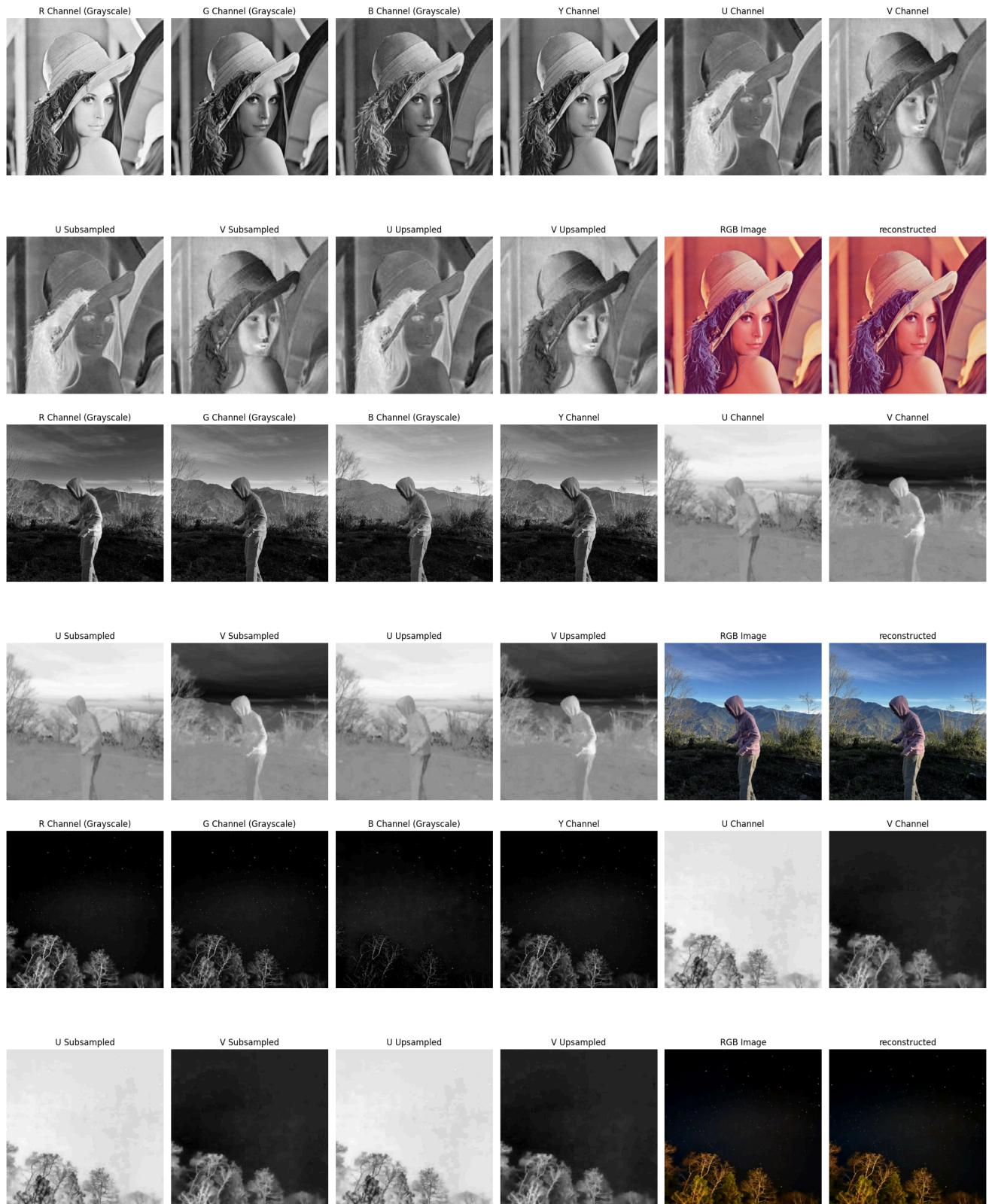
```

```
# 使用修改後的函式展示圖片
```

```
display_images_in_two_rows(images, titles, fig_size=(20, 10))
```



剩下的程式碼將和本報告之相同資料夾呈現。



額外實驗

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

# ---- 基本參數設定 ----
```

```

width = 512
height = 512
input_filename = 'Hw_image_1.raw' # (512x512 RGB24bit)
channel_order = 'RGB' # 根據實際情況設置 'RGB' 或 'BGR'

# ---- 讀取 RAW 檔案 (Sequential RGB 格式) ----
with open(input_filename, 'rb') as f:
    raw_data = np.frombuffer(f.read(), dtype=np.uint8)

# 將 R、G、B 通道分開 (Sequential 格式)
R = raw_data[:width*height].reshape((height, width)).astype(np.float32)
G = raw_data[width*height:2*width*height].reshape((height,
width)).astype(np.float32)
B = raw_data[2*width*height:].reshape((height, width)).astype(np.float32)

# 合併為 RGB 圖像
if channel_order == 'RGB':
    rgb_image = np.stack([R, G, B], axis=2).astype(np.uint8)
elif channel_order == 'BGR':
    rgb_image = np.stack([B, G, R], axis=2).astype(np.uint8)
else:
    raise ValueError("Unsupported channel order")

# ---- 增加高斯噪音 ----
def add_gaussian_noise(image, mean=0, std=10):
    """
    為圖像添加高斯噪音
    :param image: 原始圖像 (numpy array)
    :param mean: 高斯分布的均值
    :param std: 高斯分布的標準差
    :return: 添加噪音後的圖像
    """
    noise = np.random.normal(mean, std, image.shape)
    noisy_image = np.clip(image + noise, 0, 255).astype(np.uint8)
    return noisy_image

# 為 R、G、B 通道分別添加高斯噪音
R_noisy = add_gaussian_noise(R)
G_noisy = add_gaussian_noise(G)
B_noisy = add_gaussian_noise(B)

# 合併噪音後的 RGB 圖像
rgb_noisy = np.stack([R_noisy, G_noisy, B_noisy], axis=2)

# ---- RGB -> YUV (BT.601) ----
# Y, U, V range 在此為 0~255
Y = 0.299 * R + 0.587 * G + 0.114 * B
U = -0.14713 * R - 0.28886 * G + 0.436 * B + 128
V = 0.615 * R - 0.51499 * G - 0.10001 * B + 128

```

```

# 限制範圍為 0~255 並轉為 uint8
Y = np.clip(Y, 0, 255).astype(np.uint8)
U = np.clip(U, 0, 255).astype(np.uint8)
V = np.clip(V, 0, 255).astype(np.uint8)

# ---- U, V Downsampling (2x2) ----
U_sub = U[::2, ::2] # size: 256 x 256
V_sub = V[::2, ::2] # size: 256 x 256

# ---- U', V' Upsampling (Nearest Neighbor) ----
U_up = U_sub.repeat(2, axis=0).repeat(2, axis=1)
V_up = V_sub.repeat(2, axis=0).repeat(2, axis=1)

# ---- YUV -> RGB (重建 R'G'B') ----
Yf = Y.astype(np.float32)
Uf = U_up.astype(np.float32)
Vf = V_up.astype(np.float32)

# 使用 BT.601 逆轉換公式
R_rec = Yf + 1.13983 * (Vf - 128)
G_rec = Yf - 0.39465 * (Uf - 128) - 0.58060 * (Vf - 128)
B_rec = Yf + 2.03211 * (Uf - 128)

# 限制範圍為 0~255 並轉為 uint8
R_rec = np.clip(R_rec, 0, 255).astype(np.uint8)
G_rec = np.clip(G_rec, 0, 255).astype(np.uint8)
B_rec = np.clip(B_rec, 0, 255).astype(np.uint8)

# 合併為重建的 RGB 圖像
rgb_reconstructed = np.stack([R_rec, G_rec, B_rec], axis=2)

# ---- 計算 MSE 和 PSNR ----
def calculate_mse(image1, image2):
    """
    計算兩張圖片之間的均方誤差 (MSE)
    :param image1: 原始圖片 (numpy array)
    :param image2: 重建圖片 (numpy array)
    :return: MSE 值
    """
    return np.mean((image1.astype(np.float32) - image2.astype(np.float32)) ** 2)

def calculate_psnr(image1, image2, max_value=255.0):
    """
    計算兩張圖片之間的峰值訊噪比 (PSNR)
    :param image1: 原始圖片 (numpy array)
    :param image2: 重建圖片 (numpy array)
    :param max_value: 圖片的最大可能值 (通常為 255)
    :return: PSNR 值
    """

```

```

mse = calculate_mse(image1, image2)
if mse == 0:
    return float('inf') # 若 MSE 為 0，PSNR 無限大
psnr = 10 * math.log10((max_value ** 2) / mse)
return psnr

# 計算 MSE 和 PSNR
mse_rgb = calculate_mse(rgb_image, rgb_reconstructed)
psnr_rgb = calculate_psnr(rgb_image, rgb_reconstructed)

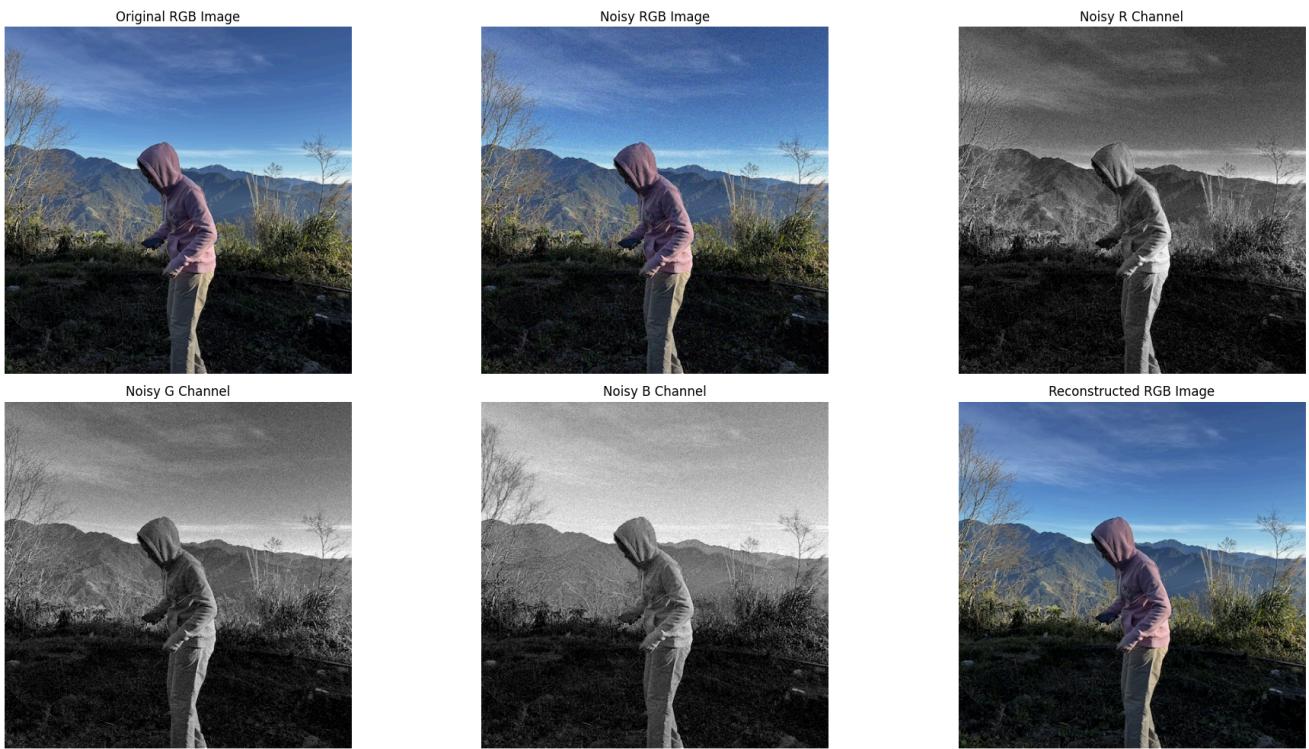
print(f"RGB Image MSE: {mse_rgb:.4f}")
print(f"RGB Image PSNR: {psnr_rgb:.2f} dB")

# ---- 在 Jupyter Notebook 中橫向排列展示圖片 ----
def display_images_in_two_rows(images, titles, fig_size=(20, 10)):
    """
    將多張圖片分為兩排排列展示
    :param images: 圖片列表，每張為 numpy array 格式
    :param titles: 標題列表，對應每張圖片
    :param fig_size: 整體圖表大小
    """
    num_images = len(images)
    num_cols = (num_images + 1) // 2 # 每排的圖片數量（向上取整）
    plt.figure(figsize=fig_size)
    for i, (image, title) in enumerate(zip(images, titles)):
        plt.subplot(2, num_cols, i + 1)
        if len(image.shape) == 3: # RGB 圖像
            plt.imshow(image)
        else: # 灰階圖像
            plt.imshow(image, cmap='gray')
        plt.title(title)
        plt.axis('off')
    plt.tight_layout() # 自動調整間距
    plt.show()

images = [rgb_image, rgb_noisy, R_noisy, G_noisy, B_noisy,
          rgb_reconstructed]
titles = [
    "Original RGB Image", "Noisy RGB Image", "Noisy R Channel",
    "Noisy G Channel", "Noisy B Channel", "Reconstructed RGB Image"
]

# 使用函式展示圖片
display_images_in_two_rows(images, titles, fig_size=(20, 10))

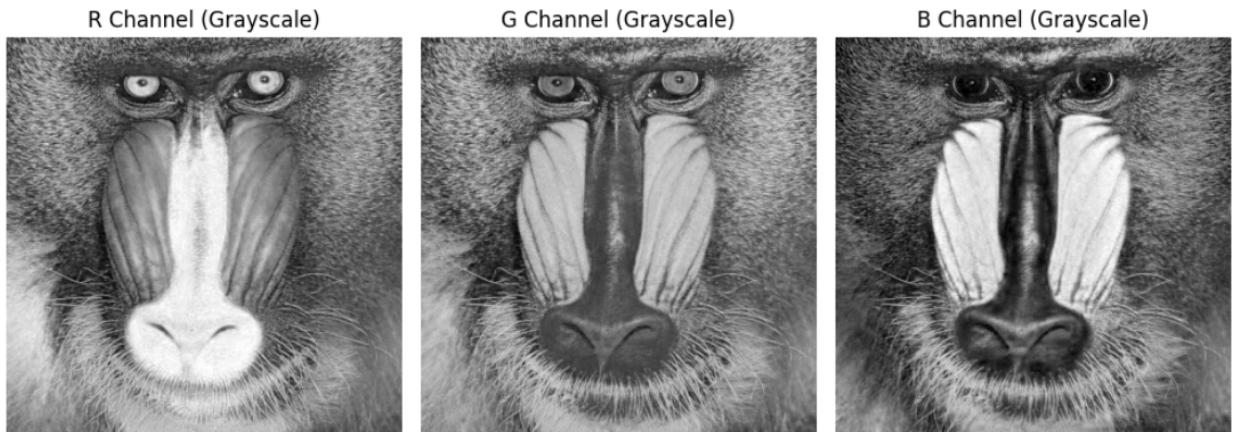
```



七、結果測試與分析 (Results and Analysis)

1. 原圖與R、G、B通道圖：

觀察 R、G、B 通道獨立輸出後的灰階影像，可確認讀取及分解正確。



• 亮度分佈：

- R 通道的亮度偏高，尤其在嘴巴和鼻子區域。
- G 通道表現出較多的細節，是視覺上最均衡的。
- B 通道的亮度最低，主要反映藍色在該圖像中較少。

• 人眼視覺敏感度：

- 綠色 (G) 是人眼最敏感的顏色，因此 G 通道往往顯示的細節最清晰。
- 紅色 (R) 次之。
- 藍色 (B) 對人眼的影響最低，亮度也相對最低。

• 用途：

- 將 RGB 三個通道分離後，可以用來進行色彩分析、轉換或壓縮，如進行 YUV 轉換和壓縮時，通常會對 **U 和 V (色度)** 進行次取樣，而保持 **Y (亮度)**。

2. Y、U、V 通道圖：

Y 通道為亮度分佈，U、V 為色度通道。U 與 V 單獨觀察時多為灰階，但實際代表顏色差異訊號。

3. U_sub、V_sub (下取樣) 與 U_up、V_up (上取樣後)：

下取樣後的 U、V 通道解析度減半，細節損失。上取樣後回到原尺寸，但此時細節無法完全復原。

4. 重建後影像(R'G'B')與原圖比較：

將 reconstructed.png 與原圖 (將原圖輸出成 PNG 供比對) 放置並行比較，可見重建後的影像較為平滑，特別是在顏色邊緣處可能有模糊現象。

可計算誤差指標如 MSE 或 PSNR，以量化影像失真程度：

MSE :

$$MSE = \frac{1}{MN} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} [I_1(i, j) - I_2(i, j)]^2$$

```
# ----- 計算均方誤差 (MSE) -----
def calculate_mse(image1, image2):
    """
    計算兩張圖片之間的均方誤差 (MSE)
    :param image1: 原始圖片 (numpy array)
    :param image2: 重建圖片 (numpy array)
    :return: MSE 值
    """

    return np.mean((image1.astype(np.float32) - image2.astype(np.float32))
** 2)

# 計算 RGB 圖像的 MSE
mse_r = calculate_mse(R, R_rec)
mse_g = calculate_mse(G, G_rec)
mse_b = calculate_mse(B, B_rec)
mse_rgb = calculate_mse(rgb_image, rgb_reconstructed)

# 計算 YUV 通道的 MSE
mse_y = calculate_mse(Y, Y)
mse_u = calculate_mse(U, U_up)
mse_v = calculate_mse(V, V_up)

# ----- 輸出 MSE 結果 -----
print("==== MSE 差異計算結果 ===")
print(f"R Channel MSE: {mse_r:.4f}")
print(f"G Channel MSE: {mse_g:.4f}")
print(f"B Channel MSE: {mse_b:.4f}")
print(f"RGB Image MSE: {mse_rgb:.4f}")
print(f"Y Channel MSE: {mse_y:.4f}")
print(f"U Channel MSE (vs Upsampled): {mse_u:.4f}")
```

```

print(f"V Channel MSE (vs Upsampled): {mse_v:.4f}")

# ---- 顯示 RGB 差異圖 ----
def display_difference(image1, image2, title):
    """
    計算並顯示兩張圖片的差異圖
    :param image1: 原始圖片
    :param image2: 重建圖片
    :param title: 差異圖標題
    """

    difference = np.clip(np.abs(image1 - image2), 0, 255).astype(np.uint8)
    plt.figure(figsize=(6, 6))
    plt.imshow(difference, cmap='gray')
    plt.title(title)
    plt.axis('off')
    plt.show()

# 顯示 R, G, B 通道的差異圖
display_difference(R, R_rec, "Difference in R Channel")
display_difference(G, G_rec, "Difference in G Channel")
display_difference(B, B_rec, "Difference in B Channel")

# 顯示重建後的 RGB 圖像與原始 RGB 圖像的差異圖
display_difference(rgb_image, rgb_reconstructed, "Difference between
Original and Reconstructed RGB")

```

輸出結果為：

```

== MSE 計算結果 ==
R Channel MSE: 5.5029
G Channel MSE: 1.3361
B Channel MSE: 9.5978
RGB Image MSE: 5.4789
U Channel MSE (vs Upsampled): 1.3155
V Channel MSE (vs Upsampled): 2.4154

```

PSNR :

$$\text{PSNR} = 10 \cdot \log_{10} \left(\frac{\text{MAX}^2}{\text{MSE}} \right)$$

```

import math

# ---- 計算 PSNR ----
def calculate_psnr(image1, image2, max_value=255.0):

```

```

"""
計算兩張圖片之間的峰值訊噪比 (PSNR)
:param image1: 原始圖片 (numpy array)
:param image2: 重建圖片 (numpy array)
:param max_value: 圖片的最大可能值 (通常為 255)
:return: PSNR 值
"""

mse = calculate_mse(image1, image2)
if mse == 0:
    return float('inf') # 若 MSE 為 0, PSNR 無限大
psnr = 10 * math.log10((max_value ** 2) / mse)
return psnr

# 計算 R, G, B 通道的 PSNR
psnr_r = calculate_psnr(R, R_rec)
psnr_g = calculate_psnr(G, G_rec)
psnr_b = calculate_psnr(B, B_rec)
psnr_rgb = calculate_psnr(rgb_image, rgb_reconstructed)

# 計算 U 和 V 通道的 PSNR
psnr_u = calculate_psnr(U, U_up)
psnr_v = calculate_psnr(V, V_up)

# ---- 輸出 PSNR 結果 -----
print("\n==== PSNR 計算結果 ===")
print(f"R Channel PSNR: {psnr_r:.2f} dB")
print(f"G Channel PSNR: {psnr_g:.2f} dB")
print(f"B Channel PSNR: {psnr_b:.2f} dB")
print(f"RGB Image PSNR: {psnr_rgb:.2f} dB")
print(f"U Channel PSNR (vs Upsampled): {psnr_u:.2f} dB")
print(f"V Channel PSNR (vs Upsampled): {psnr_v:.2f} dB")

```

==== PSNR 計算結果 ===

R Channel PSNR: 40.72 dB
G Channel PSNR: 46.87 dB
B Channel PSNR: 38.31 dB
RGB Image PSNR: 40.74 dB
U Channel PSNR (vs Upsampled): 46.94 dB
V Channel PSNR (vs Upsampled): 44.30 dB

解釋 PSNR 結果

- **PSNR 越高**，表示圖像重建的質量越好，差異越小。
- **PSNR 一般範圍**：
 - **30 dB** 以下：重建質量較差。
 - **30-40 dB**：可接受的重建質量。

- **40 dB 以上**：高質量重建，接近原始圖像。

5. 多張影像的比較：

- 不同圖像對色差取樣敏感度不同，例如 baboon 圖的細節較多、色彩較複雜，下取樣後的失真可能比 lena 稍明顯。
- 額外影像亦可顯示出不同紋理、頻率成分下，下取樣的影響程度。
- 像是額外實驗結果，我使用了高斯噪音去解析Hw_image_1的圖片，而得到以下結果：

RGB Image MSE: 3.7055

RGB Image PSNR: 42.44 dB

八、心得與總結 (Conclusion and Future Work)

- 透過本次作業，了解 YUV 色彩空間及次取樣策略對影像品質的影響。以人類視覺系統對色彩敏感度較低為原理，能在較少品質損失下降低色度訊號的採樣頻率。此方法常應用於 JPEG 等壓縮方法中。

九、參考資料

- ChatGPT
- 同學的教導
- 助教和教授的書面檔案內容