會說話的圖片

第 四 組

4110056030 鄭詠謙 / 4110056003 郭庭言 / 4110056008 林昀萱 / 4110056029 劉菡侖

一、系統設計

本文的主要創新點是針對彩色影像設計的可逆資料隱藏（RDH）方案，確保灰階版本保持不變。以下是所使用的幾個技術和方法：

灰階不變性：

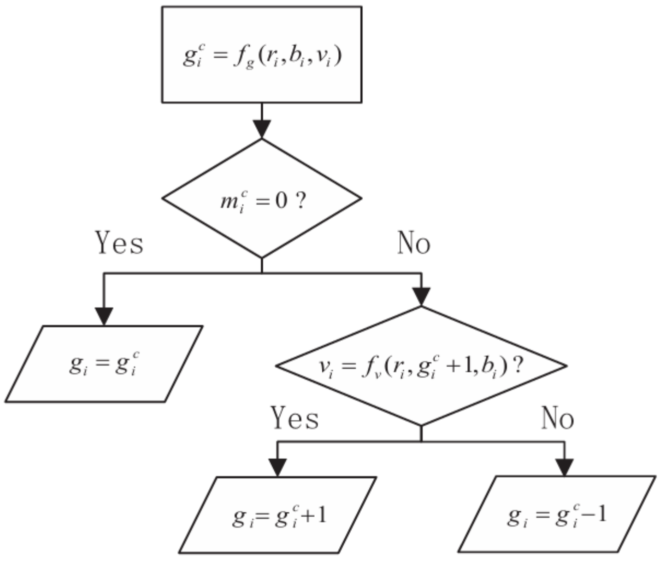
由於彩色影像的訊息比灰階影像更多更複雜，因此用此方法可確保彩色影像的灰階版本在嵌入隱藏資料後保持不變。

嵌入方式：

此篇論文採用 LSB 替換方法並將 LSB 技術納入更廣泛的框架中，確保影像的灰階版本保持不變。做法是把訊息嵌入到彩色影像的紅色 (R) 和藍色 (B) 通道中。

$$e' = 2e+m$$

然後調整綠色 (G) 通道，以確保紅色和藍色通道的修改不會改變灰階值。



調整是自適應的，以保持灰度不變性。嵌入糾錯 bit 以確保綠色通道的調整可以完美逆轉，從而保持資料隱藏過程的可逆性。這樣就可保持影像灰階不變，又可以比較好的保持影像品質。

糾錯與可逆性：

使用糾錯碼來確保綠色通道在提取隱藏訊息後能夠完美恢復。糾錯位元的遞歸嵌入是保持資料隱藏過程可逆性的關鍵步驟。實際做法是糾錯碼將會與隱藏資料一起嵌入，以確保為了保持灰階不變性而對 G 通道所做的任何更改都可以準確地被逆轉。影像復原的步驟是先從 LSB 擷取資料，接著恢復原始 pixel 值，最終使用驗證技術來檢視復原的影像和所提取出來的資料完整性。這樣的方法安全且難以察覺，保持較高的資料保留。

像素選擇與預測誤差 (PE) 直方圖：

該演算法利用因果預測器和像素選擇策略來產生具有用於嵌入的尖銳直方圖的預測誤差（PE）序列。這有助於最大限度地減少嵌入失真。使用 PE 值作為真值與預測值的連接橋梁必要性是減少處理圖片中的數值數量，並且保留最重要的訊息。其中所使用到的 edges detecting 方法是中值邊緣預測(Median Edged Detector , MED)。選擇有較小局部方差的 pixel 方法是透過檢查 PE 值，選擇小的 PE 值並在該 pixel 處插入訊息，PE 值會正比於局部方差。為了避免插入訊息後產生大的誤差會失真，因此當遷入訊息後定義 $D_{i,j}$ 來檢查師真的嚴重度，

$$D_{i,j} = \sqrt{(r_{i,j} - r'_{i,j})^2 + (g_{i,j} - g'_{i,j})^2 + (b_{i,j} - b'_{i,j})^2}$$

自適應調整和直方圖偏移：

常用的嵌入方法有差分擴展和直方圖平移。例如，差分擴展用於嵌入二進位 bit。

總結此方法的一些優點：

完整性和安全性：此方法確保原始影像可以完全恢復，這對於需要高完整性的應用（ex: 醫療和軍事影像）至關重要。

影像處理：灰階不變性確保依賴灰階版本的進一步影像處理和應用不受影響，保持影像的品質和可用性。

二、實作程式碼

執行動作：

加密 -> 執行 encode.py

加密後的圖片 : encoded_image.png

解密 -> 執行 decode.py

畫出灰階比較圖 -> 執行 compare.py

圖片名稱 : Grayscale.jpg

encode.py

```
from collections import defaultdict
from fractions import Fraction
import cv2
from functions import *

def build_prob(input_codes):
    counts = defaultdict(int)

    for code in input_codes:
        counts[code] += 1

    counts[256] = 1

    output_prob = dict()
    length = len(input_codes)
    cumulative_count = 0

    for code in sorted(counts, key=counts.get, reverse=True):
        current_count = counts[code]
        prob_pair = Fraction(cumulative_count, length), Fraction(current_count,
length)
        output_prob[code] = prob_pair
        cumulative_count += current_count

    return output_prob


def encode_fraction_range(input_codes, input_prob):
    start = Fraction(0, 1)
    width = Fraction(1, 1)

    for code in input_codes:
        d_start, d_width = input_prob[code]
        start += d_start * width
        width *= d_width

    return start, start + width


def find_binary_fraction(input_start, input_end):
    output_fraction = Fraction(0, 1)
    output_denominator = 1

    while not (input_start <= output_fraction < input_end):
        output_numerator = 1 + ((input_start.numerator * output_denominator) //
input_start.denominator)
        output_fraction = Fraction(output_numerator, output_denominator)
        output_denominator *= 2

    return output_fraction

def encode(msg):
    return ''.join([f"{bin(ord(i))[2:]:>08}" for i in msg])


if __name__ == '__main__':

    Dt = 20
    rhoT = 0
    msg = 'welovecryptography'
    mesL = len(encode(msg))
    print(f"Message you want to encode : {msg}")

    org_img = cv2.imread('input.jpg')
    img = cv2.cvtColor(org_img, cv2.COLOR_BGR2RGB)
    gray = cvtGray(img)

    print(f'{img}\nFinish reading image!')
    predict, pError, rho = PEs(gray, img)
    print(f'Finish calculating predication error!')

    # 根据訊息長度初選 ρ
    while np.count_nonzero(rho < rhoT) <= mesL:
        if np.count_nonzero(rho < rhoT) == rho.size:
            print('The picture is too small! Exit!')
            exit()
        rhoT += 1
    # 考慮参數後再選 ρ
    enough = 0
    while not enough:
        selected = [n + 2 for n in np.where(rho[2:-2, 2:-2] < rhoT)]
        if selected[0].size >= (img.shape[0] - 4)**2:
            print('The picture is too small! Exit!')
            exit()
        enough, lastEc, La, N, tagsCode = embedMsg(img, gray, encode(msg), mesL, 
selected, predict, pError, Dt)
        rhoT += 0 if enough else 1
    print(f'Finish embeding msg with the critical value of ρ being {rhoT}')
    img, gray, pError = enough

    # 在邊框中嵌入参數
    border = sorted(
        list(
            set(map(tuple, np.argwhere(gray == gray))) -
            set(map(tuple,
                    np.argwhere(gray[1:-1, 1:-1] == gray[1:-1, 1:-1]) + 1))))
    border = list(filter(lambda xy: invariant(img[xy]), border))
    if len(border) < 56:
        print('The size of image is too small to contain the necessary 
parameters')
        exit()
    for char, loc in zip(f'{rhoT:016b}' + f'{lastEc:08b}' + f'{La:016b}' + 
f'{N:016b}',
                         filter(lambda xy: invariant(img[xy]), border)):
        img[loc][2] = 2 * (img[loc][2] // 2) + int(char)

    cv2.imwrite("encoded_image.png", cv2.cvtColor(img, cv2.COLOR_RGB2BGR))

    print("encode -> PSNR : ", PSNR(org_img, cv2.cvtColor(img, 
cv2.COLOR_RGB2BGR)))
```

主要步驟包括讀取圖片並轉換為灰度圖，計算預測誤差，選擇合適的參數以確保圖片能夠容納訊息，使用 embedMsg 函數嵌入訊息，然後在圖片邊框中嵌入一些必要的參數，最後將加密後的圖片存起來。

decode.py

```
from collections import defaultdict
```

```python
import cv2
import numpy as np
from fractions import Fraction
from functions import *

def build_prob(input_codes):
    counts = defaultdict(int)

    for code in input_codes:
        counts[code] += 1

    counts[256] = 1

    output_prob = dict()
    length = len(input_codes)
    cumulative_count = 0

    for code in sorted(counts, key=counts.get, reverse=True):
        current_count = counts[code]
        prob_pair = Fraction(cumulative_count, length), Fraction(current_count,
length)
        output_prob[code] = prob_pair
        cumulative_count += current_count

    return output_prob

def find_binary_fraction(input_start, input_end):
    output_fraction = Fraction(0, 1)
    output_denominator = 1

    while not (input_start <= output_fraction < input_end):
        output_numerator = 1 + ((input_start.numerator * output_denominator) //
input_start.denominator)
        output_fraction = Fraction(output_numerator, output_denominator)
        output_denominator *= 2

    return output_fraction


def decode_fraction(input_fraction, input_prob):
    output_codes = []
    code = 257

    while code != 256:
        for code, (start, width) in input_prob.items():
            if 0 <= (input_fraction - start) < width:
                input_fraction = (input_fraction - start) / width

                if code < 256:
                    output_codes.append(code)
                break

    return ''.join([chr(code) for code in output_codes])

def decode(msg):
    return ''.join([chr(int(i, 2)) for i in (msg[i:i + 8] for i in range(0,
len(msg), 8))])

if __name__ == '__main__':

    # 讀取遷入訊息並計算 predication error
    encode_img = cv2.imread("encoded_image.png")
    imgRcv = cv2.cvtColor(encode_img, cv2.COLOR_BGR2RGB)
    grayRcv = cvtGray(imgRcv)
    predictRcv, pErrorRcv, rhoRcv = PEs(grayRcv, imgRcv)
    print(f'Finish reading embeded image and calculating predication error!')

    border = sorted(
        list(
            set(map(tuple, np.argwhere(grayRcv == grayRcv))) -
            set(map(tuple,
                    np.argwhere(grayRcv[1:-1, 1:-1] == grayRcv[1:-1, 1:-1]) +
1))))
    border = [str(imgRcv[loc][2] % 2) for loc in filter(lambda xy:
invariant(imgRcv[xy]), border)]
    rhoT = int(''.join(border[:16]), 2)
    lastEc = int(''.join(border[16:24]), 2)
    La = int(''.join(border[24:40]), 2)
    N = int(''.join(border[40:56]), 2)
    selected = [tuple(n + 2) for n in np.argwhere(rhoRcv[2:-2, 2:-2] < rhoT)]
    tagsCode = [imgRcv[value][2] % 2
                for value in filter(lambda xy: invariant(imgRcv[xy]),
selected[N:])][:La] if La != 1 else [0] * N

    # 根據參數提取嵌入的訊息
    candidate = reversed([selected[:N][index] for index, value in
enumerate(tagsCode) if value == 0])
    predictRcv = imgRcv.copy().astype(np.int32)
    pErrorRcv = np.zeros(imgRcv.shape)
    msgRcv = ''
    for i in candidate:
        rM = np.array([imgRcv[i[0] + 1, i[1], 0], imgRcv[i[0], i[1] + 1, 0],
                       imgRcv[i[0] + 1, i[1] + 1, 0]]).reshape(3, 1)
        bM = np.array([imgRcv[i[0] + 1, i[1], 2], imgRcv[i[0], i[1] + 1, 2],
                       imgRcv[i[0] + 1, i[1] + 1, 2]]).reshape(3, 1)
        grM = np.array([grayRcv[i[0] + 1, i[1]], grayRcv[i[0], i[1] + 1],
grayRcv[i[0] + 1, i[1] + 1]]).reshape(3, 1)
        X = np.mat(np.column_stack(([1] * 3, grM, grM**2)))
        predictRcv[i][0] = predictV(rM, grayRcv[i], X)
        predictRcv[i][2] = predictV(bM, grayRcv[i], X)
        pErrorRcv[i] = imgRcv[i] - predictRcv[i]

        msgRcv += str(int(pErrorRcv[i][0]) % 2)

        nextEc = pErrorRcv[i][2] % 2
        pErrorRcv[i] = pErrorRcv[i] // 2
        imgRcv[i] = predictRcv[i] + pErrorRcv[i]
        imgRcv[i][1] = np.round((grayRcv[i] - imgRcv[i][0] * RGB[0] -
imgRcv[i][2] * RGB[2]) / RGB[1])
        if lastEc != 0:
            if np.round(np.array([imgRcv[i][0], imgRcv[i][1] + lastEc,
imgRcv[i][2]]).dot(RGB)) == grayRcv[i]:
                imgRcv[i][1] += lastEc
            elif np.round(np.array([imgRcv[i][0], imgRcv[i][1] - lastEc,
imgRcv[i][2]]).dot(RGB)) == grayRcv[i]:
                imgRcv[i][1] -= lastEc
        else:
            if np.round(np.array([imgRcv[i][0], imgRcv[i][1],
imgRcv[i][2]]).dot(RGB)) != grayRcv[i]:
                print(f"index {i} has no matched ec")
        lastEc = abs(nextEc)
    print(f"=> Received Message: {decode(msgRcv[::-1])}")
```

先讀取加密後的影像並轉換為灰階，再計算預測值和預測誤差。從灰階影像中提取邊界，並從中解碼嵌入的參數。然後選擇候選像素，根據預測誤差提取訊息。解碼過程中，依據預測值更新影像數據，最後可提取出藏在影像中的訊息。

compare.py

```
from collections import defaultdict
import cv2
import numpy as np
from fractions import Fraction
from functions import *

def build_prob(input_codes):
    counts = defaultdict(int)

    for code in input_codes:
        counts[code] += 1

    counts[256] = 1

    output_prob = dict()
    length = len(input_codes)
    cumulative_count = 0

    for code in sorted(counts, key=counts.get, reverse=True):
        current_count = counts[code]
        prob_pair = Fraction(cumulative_count, length), Fraction(current_count,
length)
        output_prob[code] = prob_pair
        cumulative_count += current_count

    return output_prob

def find_binary_fraction(input_start, input_end):
    output_fraction = Fraction(0, 1)
    output_denominator = 1

    while not (input_start <= output_fraction < input_end):
        output_numerator = 1 + ((input_start.numerator * output_denominator) //
input_start.denominator)
        output_fraction = Fraction(output_numerator, output_denominator)
        output_denominator *= 2

    return output_fraction


def decode_fraction(input_fraction, input_prob):
    output_codes = []
    code = 257

    while code != 256:
        for code, (start, width) in input_prob.items():
            if 0 <= (input_fraction - start) < width:
                input_fraction = (input_fraction - start) / width

                if code < 256:
                    output_codes.append(code)
                break

    return ''.join([chr(code) for code in output_codes])

def decode(msg):
    return ''.join([chr(int(i, 2)) for i in (msg[i:i + 8] for i in range(0,
len(msg), 8))])

if __name__ == '__main__':

    # 讀取遷入訊息並計算 predication error
    encode_img = cv2.imread("encoded_image.png")
    imgRcv = cv2.cvtColor(encode_img, cv2.COLOR_BGR2RGB)
    grayRcv = cvtGray(imgRcv)
    predictRcv, pErrorRcv, rhoRcv = PEs(grayRcv, imgRcv)
    print(f'Finish reading embeded image and calculating predication error!')

    border = sorted(
        list(
            set(map(tuple, np.argwhere(grayRcv == grayRcv))) -
            set(map(tuple,
                    np.argwhere(grayRcv[1:-1, 1:-1] == grayRcv[1:-1, 1:-1]) +
1))))
    border = [str(imgRcv[loc][2] % 2) for loc in filter(lambda xy:
invariant(imgRcv[xy]), border)]
    rhoT = int(''.join(border[:16]), 2)
    lastEc = int(''.join(border[16:24]), 2)
    La = int(''.join(border[24:40]), 2)
    N = int(''.join(border[40:56]), 2)
    selected = [tuple(n + 2) for n in np.argwhere(rhoRcv[2:-2, 2:-2] < rhoT)]
    tagsCode = [imgRcv[value][2] % 2
                for value in filter(lambda xy: invariant(imgRcv[xy]),
selected[N:])][:La] if La != 1 else [0] * N
    # print(
    #     f'Finish extractig parameters:\n\trhoT: {rhoT}, lastEc: {lastEc}, La:
{La}, N: {N}, tagsCode: {"".join([str(i) for i in tagsCode])}'
    # )
    # 根據參數提取嵌入的訊息
    candidate = reversed([selected[:N][index] for index, value in
enumerate(tagsCode) if value == 0])
    predictRcv = imgRcv.copy().astype(np.int32)
    pErrorRcv = np.zeros(imgRcv.shape)
    msgRcv = ''
    for i in candidate:
        rM = np.array([imgRcv[i[0] + 1, i[1], 0], imgRcv[i[0], i[1] + 1, 0],
                       imgRcv[i[0] + 1, i[1] + 1, 0]]).reshape(3, 1)
        bM = np.array([imgRcv[i[0] + 1, i[1], 2], imgRcv[i[0], i[1] + 1, 2],
                       imgRcv[i[0] + 1, i[1] + 1, 2]]).reshape(3, 1)
        grM = np.array([grayRcv[i[0] + 1, i[1]], grayRcv[i[0], i[1] + 1],
grayRcv[i[0] + 1, i[1] + 1]]).reshape(3, 1)
        X = np.mat(np.column_stack(([1] * 3, grM, grM**2)))
        predictRcv[i][0] = predictV(rM, grayRcv[i], X)
        predictRcv[i][2] = predictV(bM, grayRcv[i], X)
        pErrorRcv[i] = imgRcv[i] - predictRcv[i]

        msgRcv += str(int(pErrorRcv[i][0]) % 2)

        nextEc = pErrorRcv[i][2] % 2
        pErrorRcv[i] = pErrorRcv[i] // 2
        imgRcv[i] = predictRcv[i] + pErrorRcv[i]
        imgRcv[i][1] = np.round((grayRcv[i] - imgRcv[i][0] * RGB[0] -
imgRcv[i][2] * RGB[2]) / RGB[1])
        if lastEc != 0:
            if np.round(np.array([imgRcv[i][0], imgRcv[i][1] + lastEc,
imgRcv[i][2]]).dot(RGB)) == grayRcv[i]:
                imgRcv[i][1] += lastEc
            elif np.round(np.array([imgRcv[i][0], imgRcv[i][1] - lastEc,
imgRcv[i][2]]).dot(RGB)) == grayRcv[i]:
                imgRcv[i][1] -= lastEc
        else:
            if np.round(np.array([imgRcv[i][0], imgRcv[i][1],
imgRcv[i][2]]).dot(RGB)) != grayRcv[i]:
                print(f"index {i} has no matched ec")
        lastEc = abs(nextEc)
    print(f"=> Received Message: {decode(msgRcv[::-1])}")
```
比較加密前後影像的灰度值。

functions.py

```
import numpy as np
from decode import decode
```

```python
from math import log10, sqrt

RGB = np.array([0.299, 0.587, 0.114])

def PSNR(original, compressed):
    mse = np.mean((original - compressed) ** 2)
    if(mse == 0):
        return 100
    max_pixel = 255.0
    psnr = 20 * log10(max_pixel / sqrt(mse))
    return psnr


def predictV(value, grayij, X):
    beta = np.linalg.pinv(X.T * X) * X.T * value
    r_predict = np.linalg.det([1, grayij, grayij**2] * beta)
    if r_predict <= min(value[1, 0], value[0, 0]): r_predict = min(value[1, 0],
value[0, 0])
    elif r_predict >= max(value[1, 0], value[0, 0]):
        r_predict = max(value[1, 0], value[0, 0])
    return np.round(r_predict)


def PEs(gray, img):
    pError = np.zeros(img.shape)
    predict = img.copy().astype(np.int32)
    rho = np.zeros(gray.shape)
    for i in range(2, img.shape[0] - 2):
        for j in range(2, img.shape[1] - 2):
            r = np.array([img[i + 1, j, 0], img[i, j + 1, 0], img[i + 1, j + 1,
0]]).reshape(3, 1)
            b = np.array([img[i + 1, j, 2], img[i, j + 1, 2], img[i + 1, j + 1,
2]]).reshape(3, 1)
            gr = np.array([gray[i + 1, j], gray[i, j + 1], gray[i + 1, j +
1]]).reshape(3, 1)
            X = np.mat(np.column_stack(([1] * 3, gr, gr**2)))
            predict[i, j, 0] = predictV(r, gray[i, j], X)
            predict[i, j, 2] = predictV(b, gray[i, j], X)
            pError[i, j] = img[i, j] - predict[i, j]
            rho[i, j] = np.var([gray[i - 1, j], gray[i, j - 1], gray[i, j],
gray[i + 1, j], gray[i, j + 1]], ddof=1)
    return predict, pError, rho


def invariant(rgb):
    return np.round(rgb[:2].dot(RGB[:2]) + 2 * (rgb[2] // 2) * RGB[2]) ==
np.round(rgb[:2].dot(RGB[:2]) + (2 * (rgb[2] // 2) + 1) * RGB[2])


def embedMsg(img, gray, msg, mesL, selected, predict, pError, Dt):
    IMG, GRAY, pERROR = img.copy(), gray.copy(), pError.copy()
    tags = []
    La = 0
    tagsCode = '0'
    ec = 0
    location = 0
    msgIndex = 0
    for i in zip(*selected):
        if tags.count(0) < mesL:
            # 遍歷滿足 rho < rhoT 的像素點進行插入訊息
            pERROR[i][0] = 2 * pERROR[i][0] + int(msg[msgIndex])
            pERROR[i][2] = 2 * pERROR[i][2] + ec
            ec = abs(int(IMG[i][1] - np.round((GRAY[i] - IMG[i][0] * RGB[0] -
IMG[i][2] * RGB[2]) / RGB[1])))
            rgb = np.array([predict[i][loc] + pERROR[i][loc] for loc in
range(3)])
            rgb[1] = np.floor((GRAY[i] - rgb[0] * RGB[0] - rgb[2] * RGB[2]) /
RGB[1])
            if np.round(rgb.dot(RGB)) != GRAY[i]:
                rgb[1] = np.ceil((GRAY[i] - rgb[0] * RGB[0] - rgb[2] * RGB[2]) /
RGB[1])
            if np.round(rgb.dot(RGB)) != GRAY[i]: print(f'该位置{i}无法满足灰度不变
性')
            D = np.linalg.norm(rgb - IMG[i])
            if np.max(rgb) > 255 or np.min(rgb) < 0 or D > Dt:
                tags.append(1)  # 設置當前的 tag 為非法（tag 為 1）
            else:
                tags.append(0)
                msgIndex += 1
                IMG[i] = rgb
        else:
            if La == 0:
                if np.unique(tags).size > 1:
                    tagsCode, La = ''.join([str(char) for char in tags]),
len(tags)
                else:
                    La = 1
            if location == La: break
            if invariant(IMG[i]):
                IMG[i][2] = 2 * (IMG[i][2] // 2) + int(tagsCode[location])
                location += 1
    if len(tags) < mesL or location < La: return False, ec, La, len(tags),
tagsCode
    print(f"=> Message: {decode(msg)}")
    return (IMG, GRAY, pERROR), ec, La, len(tags), tagsCode


def cvtGray(img):
    gray = np.zeros(img.shape[:-1])
    for i in np.argwhere(img[:, :, -1]):
        gray[i] = np.round(img[i].dot(RGB))
    return gray
```

PSNR : 計算峰值信噪比(PSNR)以評估加密前後的圖像品質

predict : 使用線性回歸預測灰度值和顏色值。

PEs : 計算預測誤差並產生相應的圖像。

invariant : 檢查圖像的灰度不變性。

embedMsg : 在圖像中嵌入訊息，並確保灰度值保持不變。

cvtGray : 將彩色圖像轉換為灰度圖像。

三、實驗結果分析

論文中描述的方法具有多種優勢，特別是透過保持灰階不變性和使用糾錯碼。然而，它也不能避免先進的隱寫分析技術的影響，尤其是那些針對 LSB 嵌入的技術。此方法的安全性可能會透過統計分析、基於機器學習的偵測和已知的掩護攻擊而受到損害。
首先根據直方圖前後變化分析灰階是否擁有不變性，藉由直方圖可以看出嵌入訊息後的像素值難以分辨差別，以 Reversible Data Hiding 與 Grayscale Invariance 修改後的直方圖顯示的是嵌入訊息後影像的灰階值分佈。這個直方圖應該與原始圖像的灰度直方圖非常相似，因為嵌入訊息的過程目的在於保持影像的灰階不變性。這意味著在嵌入信息後，影像的外觀和亮度分佈應該盡可能保持不變。

實作範例：

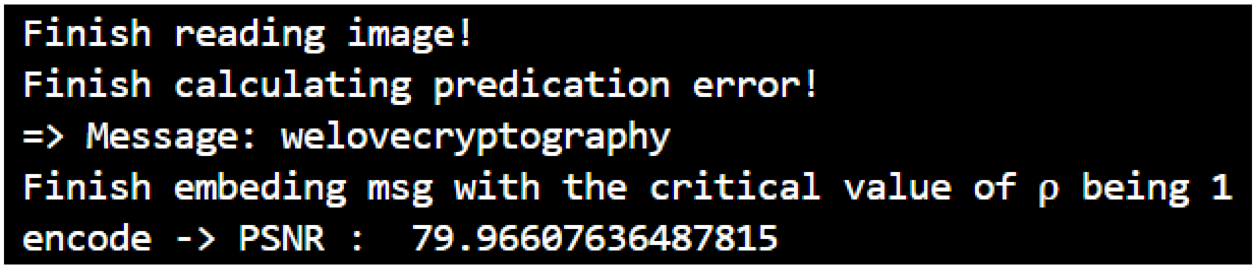
欲隱藏的訊息：weallovecryptography

輸入的影像：

Nomtendo!
A Nintendo Charity Food Zine

加密後的影像：

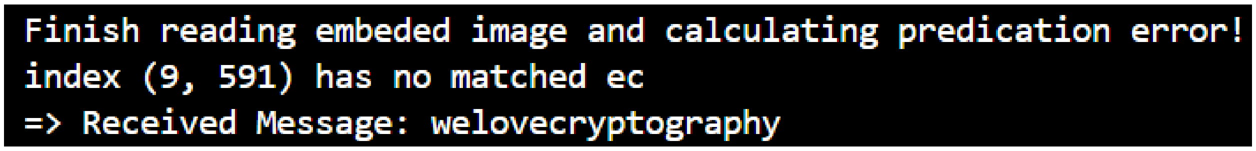
Nomtendo!
A Nintendo Charity Food Zine



```
Finish reading image!
Finish calculating predication error!
=> Message: welovecryptography
Finish embeding msg with the critical value of ρ being 1
encode -> PSNR :  79.96607636487815
```

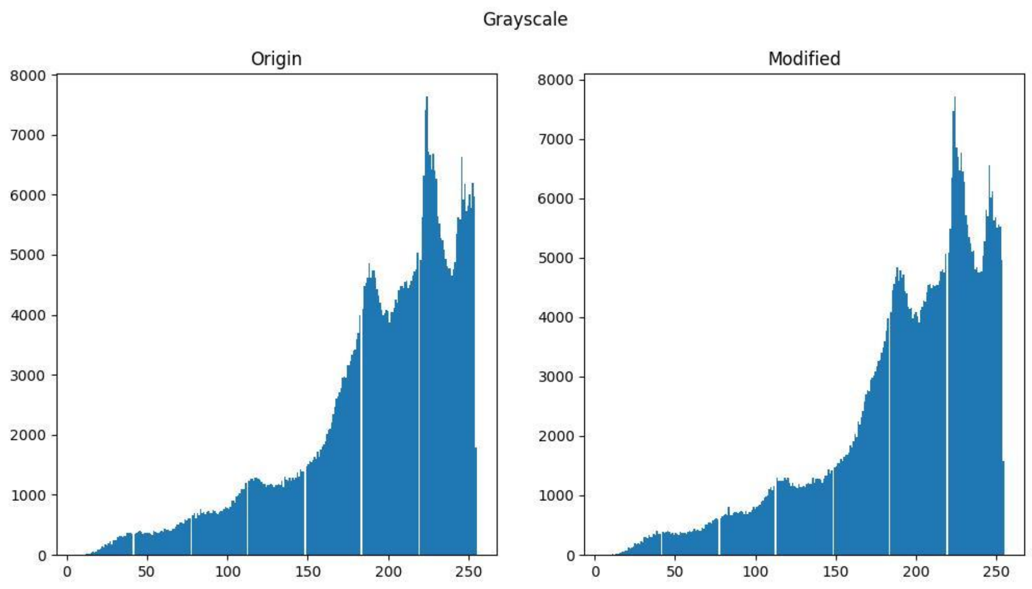
兩者的 PSNR 為 79.96607636487815

解密後得到的訊息：



```
Finish reading embeded image and calculating predication error!
index (9, 591) has no matched ec
=> Received Message: welovecryptography
```

加密前後的灰度值比較



最後，我們認為可以透過以下方式來增強此方法:

1. 與其他技術結合：將 LSB 嵌入與更複雜的隱寫技術結合可以提高安全性。
2. 抗雜訊和抗壓縮性：採取措施承受常見的影像處理操作可以保護隱藏資料免受意外損壞。
3. 高級加密：在嵌入隱藏數據之前對其進行加密可以增加額外的安全層，使攻擊者更難理解提取的數據，即使他們檢測到數據的存在。