影像處理作業

A watermarking-based image ownership and tampering authentication scheme  
基於浮水印技術圖像認證方法

實作

系級:資工碩一

學號:107522099

姓名:詹振宗

2018年12月21日

1. **介紹**

**摘要**

本作業的程式碼保管在github上

<https://github.com/frogben/water_marking>

本作業依照論文

Chin-Chen Chang,Yih-Shin Hu,Tzu-Chuen Lu “[A watermarking-based image ownership and tampering authentication scheme](https://www.sciencedirect.com/science/article/abs/pii/S016786550500259X)”

內的提出的影像浮水印技術實作，此方法不同於以往的浮水印技術無法定位修改處的缺點，此論文題提出的方法可定位出影像被竄改之處，且不影響影像的品質，而為了使程式更容易實作，目前只實作灰階圖片的浮水印認證方法，而且實作以下三個功能

1)將影像加上隱藏式浮水印

2)認證已加上浮水印的影像是否有被竄改

3)可以定位影像被修改之處

1. **相關工作**

依照論文內所提及的方法，影像浮水印技術分為以下兩種

* fragile authentication

加上浮水印的影像上的任何變動都是不允許的，否則此浮水印技術

會將影像視為已經被竄改，本論文提出的方法以及本作業是屬於這種方法。

* robust authentication

可認證經過濾，壓縮，裁剪，銳化，模糊的影像，依舊可以使用浮水印認證。

影像浮水印根據實作方式分為

* labeling-based schemes

需額外檔案作為認證，但是會增加記憶體與硬碟額外的負擔

* watermarking-based

浮水印直接嵌入在多媒體影像內，分為1)浮現式2)隱藏式兩種

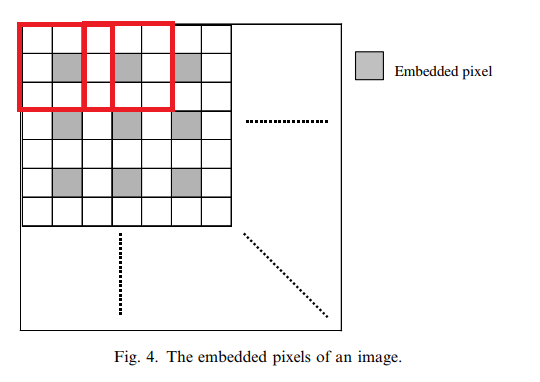
又通常屬於fragile authentication

1. **實驗方法**

**加入浮水印步驟**

1. 將影像切為多個九宮格(9X9)，每個九宮格的左邊三個像素跟上一個九宮格重疊(圖1)

圖1



1. 將每個九宮格的ID(Index of this block )、九宮格的特徵 (P1~P8) (圖2) 、影像ID(Identification of the image)、加密者密碼(User’s secret key) 經過HASH（md5雜湊）成一個長度128 bits的特徵值(圖三)

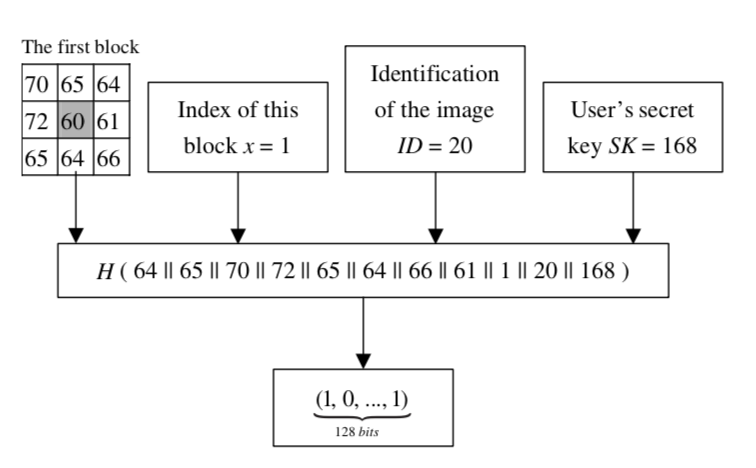
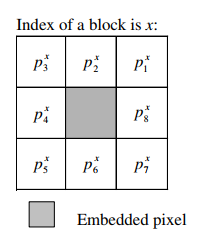


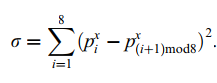
圖2 圖3

1. 需要將128bit特徵值壓縮至2~4 bit以便藏入圖片中，不然會影響影像品質，首先要決定要縮成多短(論文規定2~4個)，依照論文所提出的公式(圖4)，依照每個九宮格像素之間的差異(九宮格內像素的亂度)，算出σ，而作者經實驗發現σ介於

0 ≤ σ < 8 壓縮成2個bit

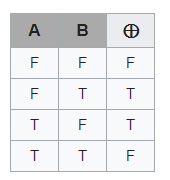
8 ≤ σ <16 壓縮成3個bit

16 ≤ σ ≤ 255 壓縮成4個bit

圖4

亂度越高，保留的資料量越多，決定好要壓縮後的長度後，再使用exclusive or（⊕，如圖5）將128bit特徵值，壓縮至2~4 bit，如何壓縮，請先看簡單例子，例如如果4bit 縮成2bits，相等於2bit與2bit做exclusive or，例如1 0 ⊕ 1 1 => 1 0，所以如果要將128bits 縮成 2bits，等於64個 2bit 依次做exclusive or，

最終得到2 bit

圖5

由於128bit只能被2,4整除，並不能被3整除，所以在壓縮之前會將128bit再加上4個bit(0000)，長度變為132bit，此時就可以被2,3,4整除，讓特徵值皆可被壓縮至2~4bit。

1. 將壓縮後的特徵值塞入九宮格中間的像素的LSB(Least Significant Bit)，避免影響影像品質
2. 完成加入隱藏浮水印

**認證影像步驟**

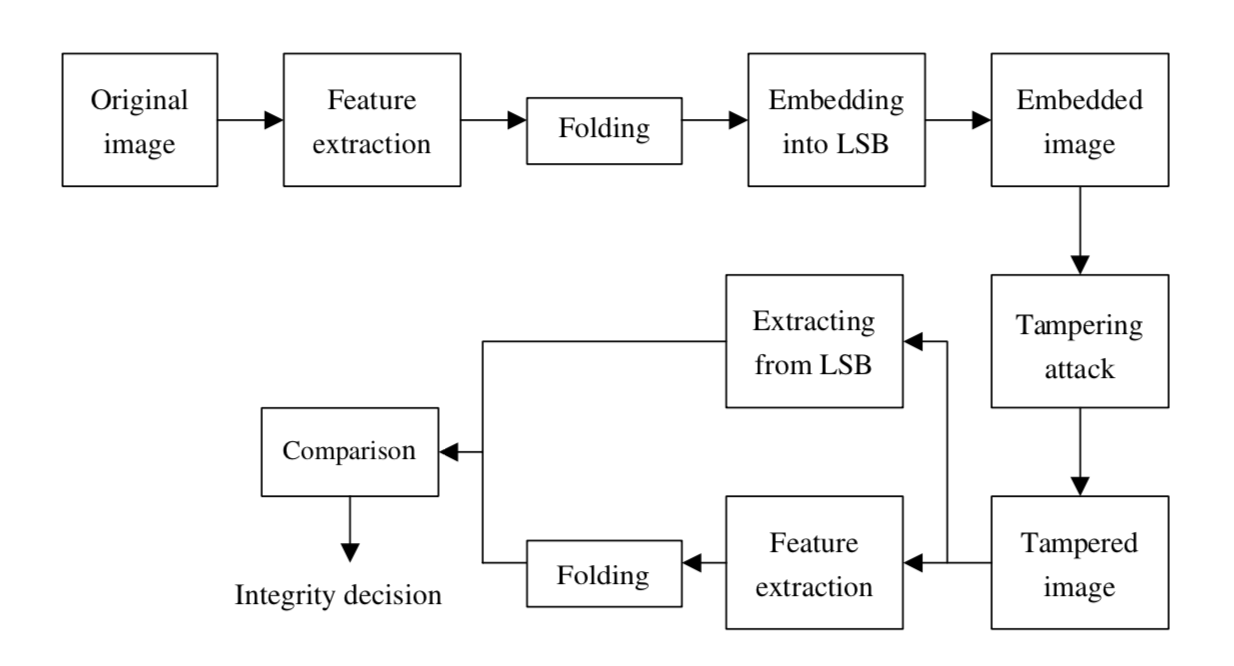
首先得到一張已作浮水印處理之影像

1. 照上述方法每個九宮個的8個Pixel作運算得到特徵值且運算出壓縮長度
2. 運算出的特徵值與九宮格中間的嵌入值(取出壓縮長度的資料)比較

相同=>驗證成功

不相同=>驗證失敗

認證流程如下圖



1. **實作結果**
2. 實驗環境

* 程式語言 python
* 實驗平台 mac
* 相依套件
  + OpenCV-python
  + Hashlib
  + Numpy

1. 程式說明

因爲本實驗屬於fragile authentication，在此浮水印壓縮技術下，任何變動都是不允許的，必須使用不壓縮或壓縮後不失真的影像格式，如bmp,png等影像格式，本作業使用bmp檔案格式，使用圖片原為灰階lena\_gray.bmp，大小為512\*512（高\*寬）如下圖。



檔案說明如下，影像放在data資料夾下，在programs資料夾下，程式檔案有兩個分別有lib.py和main.py，lib.py提供了浮水印的實作細節，而main.py則呼叫lib.py內的函式進行展示，程式皆為自己所寫。

lib.py 講解

lib.py本程式主要提供兩個函式，分別為

* add\_waterMarking()
  + 傳入值為灰階影像像素的二維陣列
  + 回傳已做浮水印處理後的灰階像素的二維陣列
* validate\_waterMarking()
  + 傳入值為灰階影像像素的二維陣列
  + 回傳1)影像是否有被竄改2)被竄改的像素所在的九宮格位置

main.py 講解

1. 第一部分－認證影像，將影像加上浮水印

先使用套件open-cv提供的函式imread讀取尚未加上浮水印的灰階lena\_gray.bmp，再使用lib.py內的自己寫的add\_waterMarking()將圖片加上浮水印，接著存檔成lena\_encode.bmp，程式碼如下圖所示



1. 第二部分－仿造竄改圖片

取出上一步驟加上浮水印後的灰階影像lena\_encode.bmp

修改其中一個像素的值，方便下一步驟驗證，這邊修改位置H=50,W=30的像素，將值更改為10，再存檔成lena\_encode\_modified.bmp，檔案路徑請參照第一部分，程式碼如下圖所示。



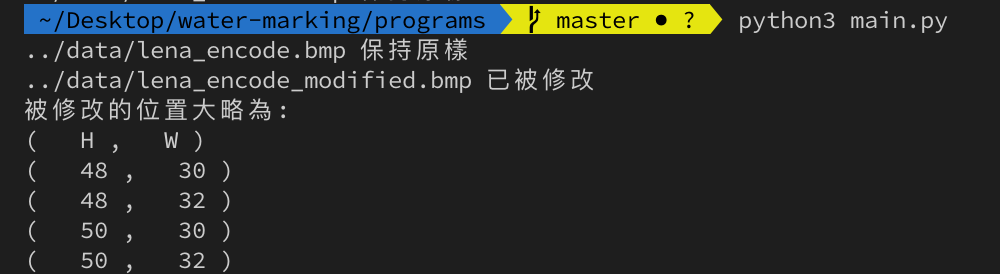
1. 第三部分－驗證圖片

為了測試實作浮水印的驗證能力，分別測試lena\_encode.bmp與lena\_encode\_modified.bmp，作為對照組與實驗組，分別讀取兩個檔案，使用validate\_waterMarking()，驗證檔案是否遭到竄改及被竄改的大略位置，並將結果輸出，輸出結果請見下一部分，程式碼如下圖所示。



1. 結果

結果如下圖所示，結果顯示lena\_encode.bmp保持原樣lena\_encode\_modified.bmp顯示已經被修改，且顯示被修改的大概位置，我們在第二部分修改位置H=50,W=30的像素（仿造竄改圖片），而在這程式定位出四個非常相近的位置，這是由於程式只能定位被竄改相素所屬的九宮格，一個像素最多會屬於四個九宮格，所以會產生四個位置，實驗到這裡可以明顯看出本作業的功能，可以讓一張圖在保持高品質之下加上隱藏的浮水印，並且而定位被竄改的位址。



1. **結論**

在一開始讀這篇論文時，無法理解它部份的實作細節，經由這次作業的機會，將此方法實作出來，有十足的成就感，並且可以更了解論文的細節，依照論文使用的方法的確可以辦別影像是否遭到竄改，以及定位竄改之處，並且保持影像高品質，但此方法不能使用壓縮過後的檔案格式，是一大缺點，因為主流的圖片格式幾乎都是壓縮檔案，使得這個方法的使用場景受限，且目前尚未實作還有一些小缺點，例如目前只實作了灰階影像，且當檔案不能被的最右邊被修改時，由於九宮格的大小關係，並不能識別竄改，這都是這個實作需要改進之處，也從這次實作學到了c++平台上以及python平台上的openCV安裝，且學習到了openCV各種方便的函式，openCV在讀檔寫檔方面的函式非常方便，過往在大學時期，曾經拜讀學長寫的jepg壓縮原始碼，程式相當複雜，檔頭各種欄位，讓我望之卻步，而在openCV一行程式碼已經幫你完成如此複雜的讀檔寫檔工作，讓我體會到openCV的方便之處，我想最近關於我的研究，關於深度學習方面，有許多都地方都需要應用到影像處理，經過這次作業，往後會在影像處理方面更加得心應手，也是我修習這門課的主要原因。