Tarea 1: Esteganografía

Benjamín del Pino B.

Agosto 2019

Resumen

Usando la esteganografía se logra poder ocultar texto dentro de imágenes en escala de grises. Esto se logra guardando en los bits menos significativos de los píxeles de la imagen los bits de los caracteres del mensaje que se quiere ocultar. Los resultados demuestran que es posible ocultar mensajes en las imágenes y que a un bajo numero de bits significativos como 1 o 2 la imagen resultante no presenta mayores cambios comparada a la imagen original, sin embargo mientras crece ese valor la imagen resultante presenta notorios cambios. También fue posible decodificar un mensaje oculto en una imagen ya codificada, esto se logra sabiendo cuantos son los bits menos significativos con los que esta se codificó, los cuales se encontraban también ocultos en la imagen.

Introducción

Se desea poder ocultar texto en una imagen y de manera inversa poder descifrar mensajes ocultos en imágenes ya codificadas. En este sentido podemos usar la esteganografía para llevar a cabo nuestro propósito, esta practica trata la aplicación de técnicas que permiten ocultar datos (texto, imágenes, vídeos, etc.) dentro de un archivo de modo que no se perciba el hecho. Es decir, pretende ocultar mensajes dentro de otros objetos y de esta forma establecer un canal encubierto de comunicación, de modo que el propio acto de la comunicación pase inadvertido para observadores que tienen acceso a ese canal. Existen múltiples técnicas según el tipo de archivo en el cual se quiera ocultar un mensaje, en nuestro caso particular de imágenes el método

mas utilizado es el **LSB (Least Significant Bit)**, ya que un archivo de imagen es un archivo que muestra diferentes colores e intensidades de luz en sus píxeles, los cuales pueden ser alterados para ocultar el mensaje deseado y así las modificaciones serán imperceptibles para el ojo humano respecto a la imagen original.

Desarrollo

El uso del programa se muestra en la figura 1. Los argumentos `--text` y `--nbits` solo se usan cuando se usa `--encode`. En este modo se procesa el mensaje que se quiere ocultar en la imagen para luego hacer lo mismo con la imagen a la que se le van a modificar los píxeles. Se comienza por cambiar el formato a los caracteres del mensaje a formato ASCII, para luego convertirlos a números binarios y agregarlos a una lista, del mismo modo se convierten los números enteros de la matriz de la imagen a binario para luego codificar el mensaje. Para llevar a cabo la codificación se dividen uno a uno los números binarios que están en la lista por el numero de bits que fue ingresado para formar los substrings que serán añadidos a los píxeles de la imagen. Luego se procede a modificar los píxeles de la imagen agregando dichos substrings a los bits menos significativos píxel por píxel, obteniendo así una nueva matriz que será la imagen con el mensaje oculto. Para dejar un registro del numero de bits usados se modifican los últimos 4 bits del píxel ubicado en la esquina inferior derecha de la matriz de la imagen, dejando 0001 si el numero de bits fue 1, 0010 si fue 2, 0100 si fue 4 o 1000 si fue 8. De la misma forma se guarda la cantidad de píxeles que se van a utilizar en la posición inmediatamente anterior en donde se guardó el numero de bits menos significativos, este numero será usado en el modo `--decode`. Por ultimo se convierten todos los píxeles de la matriz de la imagen que estaban en binario a entero para poder guardar así la nueva imagen que contiene el mensaje oculto. Por otro lado si se quiere decodificar el mensaje de una imagen (modo `--decode`), se comienza por identificar el numero de bits utilizados en la codificación y la cantidad de píxeles de la imagen que se van a usar. Estos píxeles se agregan a una lista, y se extraen de ellos los bits menos significativos y se van agrupando de a 8 bits con lo que se consiguen uno a uno los caracteres del mensaje oculto. Para terminar el proceso, se convierten los números binarios de la lista a enteros y luego se hace la inversa de la conversión de caracter a ASCII que se llevo a cabo en la codificación para así obtener los caracteres

del mensaje, el cual se imprime en pantalla.

```
usage: tarea_1.py [-h] [--encode | --decode] [--image IMAGE] [--text TEXT]
                  [--nbits {1,2,4,8}]

CC5508 - Procesamiento y Analisis de Imagenes.Tarea 1: Estenografía

optional arguments:
  -h, --help          show this help message and exit
  --encode            indica que la tarea es codificar un texto de entrada
                      dentro de una imagen.
  --decode            indica que la tarea es decodificar una imagen, mostrando
                      el texto oculto.
  --image IMAGE       indica la imagen que sera codificada/decodificada.
  --text TEXT         indica el archivo que contiene el texto a codificar.
  --nbits {1,2,4,8}   indica el numero de bits menos significativos a ser
                      usados.
```

Figure 1: uso del programa en la terminal

Resultados Experimentales y Discusión

De la experimentación se puede desprender que a un numero de bits menos significativos bajo (1 o 2) la imagen resultante comparada a la imagen original no presenta cambios observables, esto se puede apreciar comparando las figuras 3 y 5, que tienen un numero de bits de 1 y 2 respectivamente, con la figura 2 que muestra la imagen original. También podemos observar que al aplicar zoom la imagen no presenta incoherencias con la imagen original, esto lo podemos ver en las figuras 2 y 4 que presentan los mismos números de bits que las figuras 3 y 5 respectivamente. Por el contrario cuando el valor de los bits menos significativos supera a 2 la imagen se empieza a tornar algo extraña, esto lo podemos apreciar no tan notoriamente en la figura 7, pero si lo podemos apreciar en la figura 8 al hacer zoom, en donde se nota que la imagen se vuelve algo difusa. Ya en las figuras 9 y 10 el cambio es rotundo, la imagen obtenida es completamente distinta a la original y se nota que está corrompida. Esto se debe a que mientras menor es el numero de bits menos significativos menor va a ser el cambio que presenten los píxeles de la imagen original con la imagen resultante (ver figuras 11 y 12).

Figure 2: imagen original

Figure 3: imagen codificada con un numero de bits igual a 1

Figure 4: imagen codificada con un numero de bits igual a 1 con zoom aplicado

Figure 5: imagen codificada con un numero de bits igual a 2

Figure 6: imagen codificada con un numero de bits igual a 2 con zoom aplicado

Figure 7: imagen codificada con un numero de bits igual a 4

Figure 8: imagen codificada con un numero de bits igual a 4 con zoom aplicado

Figure 9: imagen codificada con un numero de bits igual a 8

Figure 10: imagen codificada con un numero de bits igual a 8 con zoom aplicado

```
Imagen Original:
[[155 155 155 ... 164 148 120]
 [155 155 155 ... 164 148 120]
 [155 155 155 ... 164 148 120]
 ...
 [ 35  35  43 ...  95  91  88]
 [ 37  37  48 ...  94  95  98]
 [ 37  37  48 ...  94  95  98]]


Imagen Codificada:
[[154 155 154 ... 164 148 121]
 [154 155 155 ... 164 148 121]
 [154 155 155 ... 164 148 120]
 ...
 [ 34  35  43 ...  95  90  88]
 [ 36  37  49 ...  94  94  98]
 [ 36  37  49 ...  95  94  99]]
```

Figure 11: comparación de imágenes usando `nbit = 1`

```
Imagen Original:
[[155 155 155 ... 164 148 120]
 [155 155 155 ... 164 148 120]
 [155 155 155 ... 164 148 120]
 ...
 [ 35  35  43 ...  95  91  88]
 [ 37  37  48 ...  94  95  98]
 [ 37  37  48 ...  94  95  98]]


Imagen Codificada:
[[ 76 111 114 ... 103  32 101]
 [108 105 116 ... 108  97  98]
 [111 114 101 ... 112 116 117]
 ...
 [ 35  35  43 ...  95  91  88]
 [ 37  37  48 ...  94  95  98]
 [ 37  37  48 ...  94  95  98]]
```

Figure 12: comparación de imágenes usando `nbit = 8`

Conclusiones

Quedó demostrado que es posible codificar texto en imágenes de manera que no se perciba a simple vista siempre y cuando se utilicen valores ya sea 1 o 2 de bits menos significativos ya que vimos que si se usaban valores mayores a estos la imagen resultante es distinta que la original, lo cual estaría violando el principio fundamental de la esteganografía. Lamentablemente mi implementación solo puede recibir imágenes en blanco y negro, pero no por eso no puedo hacer una apreciación de lo que podría suceder si se recibe una imagen a color, en ese caso la cantidad de bytes se triplicaría lo cual ayudaría a ocultar de una mejor manera los caracteres del mensaje. También mi implementación se limita a recibir valores determinados de bits pero en general mientras aumenta el numero de bits menos significativos la imagen resultante es notoriamente distinta a la original. El texto a codificar no debe ser tan largo, por lo tanto se asume que los textos que recibe son lo suficientemente cortos para ser codificados. El problema mas grande que se me presentó en el desarrollo de la tarea fue el manejo de los bits, es por esto que fueron strings los que me ayudaron a hacer las transformaciones ya que me manejo bastante bien con ellos, sin embargo para próximas tareas es necesario mejorar este aspecto. El poco conocimiento de la librearía scikit-image también fue un obstáculo ya que al no estar familiarizado con esta no sabia muy bien que hacían las funciones o no sabia muy bien si existían funciones que me ayudaran a resolver la tarea mas fácilmente, es por esto que mi programa no es el mas óptimo pero funciona al menos. Por ultimo el programa al ser secuencial tuvo muchas dificultades para ser debugeado, es por eso que para próximas entregas se deberá poner mas esfuerzo en la metodología de diseño del programa para no perder tanto tiempo resolviendo errores.