Tarea 1 CC5508: Esteganografía

Javier Morales

4 de Octubre del 2020

1 Resumen

En el siguiente informe se estudia la estenografía en el ámbito de las imágenes, es decir, técnicas que permiten ocultar una imagen dentro de otra tratando de no alterar demasiado la original. Se escriben dos programas, uno para ocultar la imagen y otro para obtenerla de vuelta, y se procede a estudiar la capacidad de ocultamiento de distintos inputs. Se concluye que es posible alterar alrededor de un tercio de los bits menos significativos de una imagen sin exponer la ocultada, cambiar más bits que estos resulta en un solapamiento de ambas imágenes.

2 Introducción

La esteganografía se refiere al conjunto de técnicas que buscan esconder información dentro de distintos medios. Su nombre deriva del griego combinando las palabras *steganós* (oculto o cubierto) con *graphia* (escritura). Las primeras técnicas consistían del uso de propiedades físicas y químicas, como el uso de colores y lentes especiales o tinta invisible.

Hoy en día el foco se encuentra en el ámbito digital, tratando de esconder información en archivos o señales electromagnéticas. Debido a esto el siguiente informe busca ser una introducción al manejo de imágenes a nivel de datos, manipulando los bits que componen los pixeles y viendo como esto afecta a la imagen original.

3 Desarrollo

3.1 `hide.py`

El programa `hide.py` se encarga de ocultar una imagen dentro de otra y se ejecuta con la siguiente linea:

```
python3 hide.py -imagenA [path to imageA] -imagenB [path to imageB]
```

De esta forma `imagenA` es la imagen en donde se va a ocultar la `imagenB`. Este programa se puede dividir en varios pasos, en donde se encarga de compatibilizar distintas combinaciones de inputs. Luego de la ejecución el programa muestra la imagen A con B ocultada dentro y guarda el archivo **hidden.png** que contiene el resultado.

3.1.1 Igualamiento de dimensiones

Para permitir una mayor cantidad inputs, y para permitir imágenes en gris y a color se realiza un igualamiento de dimensiones. Esto se divide en dos reglas: si una imagen es a color y la otra en gris, la imagen en gris se transforma en una a color triplicando la imagen en los canales de color; las primeras dos dimensiones del output siempre son las de la imagen A, si la imagen B es más grande se recorta y si es más pequeña se rellenan los valores faltantes con ceros.

3.1.2 Validaciones

Para determinar si es posible ocultar la imagen B dentro de la imagen A se calculan los bits máximos ocupados por B y se comparan con la "capacidad de ocultamiento" de A, en donde este último es un valor que se busca experimentalmente. Si los bits máximos ocupados por B son mayores a este valor el programa lanza un error.

3.1.3 Operaciones de bits

Una vez se tienen dos imágenes del mismo tamaño y los bits máximos ocupados por B, que llamaremos `bits_hidden`, se realiza un simple operación de bits: se borran los `bits_hidden` bits menos significativos de A y se realiza un `or` lógico entre ambas imágenes. Esta es la imagen resultante.

3.1.4 Guardado de número de bits ocultados

Para poder saber cuantos bits fueron escondidos en la imagen resultante se guarda el valor `bits_hidden` dentro de uno de los pixeles del resultado. Si la imagen es en gris se hace en la posición (0,0) y si es a color en (0,0,0).

3.1.5 Parámetro `-b`

Para facilitar los experimentos se agrega un parámetro opcional `-b` que corresponde a un numero entre 1 y 8. Este valor indica que se deben insertar los `b` bits más significativos de la imagen B, de esta forma se pueden hacer varias pruebas con la misma imagen. La única validación que se hace en este caso es que la imagen B tenga por lo menos un valor con más de `b` bits, para evitar insertar ceros innecesarios en A.

3.2 `unhide.py`

El programa `unhide.py` recupera la imagen ocultada previamente, y se ejecuta con la siguiente linea:

```
python3 unhide.py [path to image]
```

Luego de la ejecución el programa muestra la imagen recuperada y guarda el archivo **unhidden.png** que contiene el resultado.

3.2.1 Obtención de número de bits ocultados

Lo primero es obtener el valor `hidden_bits` desde la posición (0,0) o (0,0,0) según corresponda. No se realiza ninguna validación de este valor, se asume que se recibe el archivo de `hidden.py` como input.

3.2.2 Operaciones de bits

Simplemente se hace un shift de 8 menos `hidden_bits` bits hacia la izquierda, dejando los bits ocultados como los bits más significativos, de esta forma el gamma del resultado es más alto y es más fácil ver imágenes con valores bajos.

4 Resultados y discusión

El primer experimento consiste ocultar solo dos bits de una imagen dentro de otra, puesto que esconder solo un bit es muy poca información de una imagen. En el resultado 1 la imagen a la izquierda corresponde a la imagen A que contiene a la imagen B, esta tiene buena definición de colores y no muestra contornos ni figuras de la imagen escondida. La imagen a la derecha es la imagen escondida B, y es fácil notar que con solo 2 bits se tienen pocos colores y contornos pixelados.

(a) Resultado de `hide.py`

(b) Resultado de `unhide.py`

Resultado 1: Imagen brillante con 2 bits de imagen oscura escondidos

Como la imagen A todavía parece tener capacidad de ocultamiento se decide aumentar la cantidad de bits escondidos a 4, y en el resultado 2 se ve que la imagen A pierde calidad de colores pero todavía no muestra figuras de B. Por otro lado la calidad de la imagen escondida aumenta considerablemente.

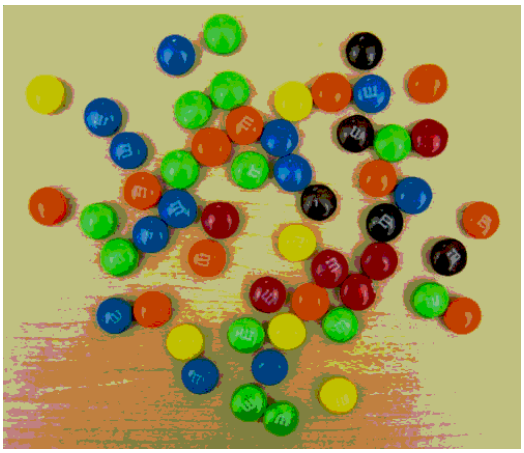
(a) Resultado de `hide.py`

(b) Resultado de `unhide.py`

Resultado 2: Imagen brillante con 4 bits de imagen oscura escondidos

Como en los experimentos anteriores se tiene una imagen brillante y con varios colores vs. una oscura con pocos colores se decide invertir estas imágenes. De nuevo se realiza el experimento con dos bits y en el resultado 3 nuevamente la imagen A no muestra mayores cambios, pero la imagen B escondida es de muy mala calidad en cuanto a colores.

(a) Resultado de `hide.py`



(b) Resultado de `unhide.py`

Resultado 3: Imagen oscura con 2 bits de imagen brillante escondidos

De la misma forma, como la imagen A todavía parece tener más capacidad, se aumentan los bits escondidos a 4. En el resultado 4 la imagen A muestra claramente figuras de la imagen B, especialmente en el fondo oscuro de la imagen. La imagen B en cambio mejora bastante sus colores y contornos.

(a) Resultado de `hide.py`

(b) Resultado de `unhide.py`

Resultado 4: Imagen oscura con 4 bits de imagen brillante escondidos

En los siguientes experimentos se incluyen imágenes en escala de grises, en el resultado 5 se esconden 3 bits de una imagen gris en una a color y en el resultado 6 se invierten las imágenes. En ambos casos la imagen A no muestra contornos de B ni pierde calidad de colores, mientras que la imagen B presenta una perdida de colores, aunque menos notoria que solo usando 2 bits.

(a) Resultado de `hide.py`

(b) Resultado de `unhide.py`

Resultado 5: Imagen de color con 3 bits de imagen gris escondidos

(a) Resultado de `hide.py`

(b) Resultado de `unhide.py`

Resultado 6: Imagen gris con 3 bits de imagen de color escondidos

5 Conclusiones

La capacidad de ocultamiento de una imagen parece estar relacionada a la relación entre el brillo entre la imagen que oculta y la ocultada. Una imagen brillante puede ocultar mas bits de una imagen más oscura, y al contrario una imagen oscura puede ocultar menos bits de una imagen más brillante.

En cualquier caso ocultar más de la mitad de los bits significa que la imagen escondida se comienza a mostrar, utilizar de 3 de 8 bits para ocultar parece tener buenos resultados en la mayoría de los casos y por lo tanto el programa utiliza la formula `ceiling(bits_A * 3 / 8)` para determinar la capacidad de ocultamiento. Estos resultados se mantienen si se mezclan imágenes grises con imágenes a color.