Trabajo práctico especial
Teoría de la información

Integrantes: Álvarez, Mauricio Ezequiel
(Maurialva.ma@gmail.com)
Barceló, Sofía
(SofiaBarcelo97@gmail.com)
ID Grupo: 2

1.1Resumen

En el trabajo práctico especial de teoría de la información del año 2020 se vio cómo se estudia el caso de un niño desaparecido, basándose en la serie “Stranger Things” en la cual hay un niño desaparecido y se debe ayudar a la policía a resolver caso con las herramientas que vimos en la materia.
Estas herramientas son diferentes algoritmos para analizar diferentes aspectos de las imágenes que nos provee la policía y llegar a una conclusión con cada uno de estos parámetros.

1.2. Introducción

Para el trabajo práctico especial y con el problema que ya nombramos en el resumen, se solucionaran los siguientes problemas:

1. Analizar las imágenes que se encuentran en la estación de policía con la foto de Will ordenándolas por parecido (utilizando factor de correlación como medida de similitud).
2. Estudiar las similitudes de de la foto original de Will, la foto más parecido que obtuvimos en el punto 1 y la foto que consiguió el policía de guardia, analizando las distribuciones de intensidades. Además, generar un histograma de cada una de estas imágenes, calcular la media y el desvío de cada una de estas distribuciones.
3. Implementar un algoritmo que permita codificar y decodificar una imagen mediante el método de Huffman. Con este algoritmo:
    A. Coprime la imagen Original.
    B. comprimir la imagen más parecida obtenida en el inciso 1, con el código de la imagen original.
    C. comprimir la imagen que trajo el policía con el código de Huffman de la imagen original.
    D. comprimir la imagen que trajo el policía, con el método implementado en este inciso, utilizando su código de decodificación, comparando el resultado con el del Inciso C.
    E. Comparar las tasas de compresión de los ejercicios A, B, C y D.

4. Diferentes canales de televisión, están teniendo un problema cuando transmiten la fotografía de Will. Utilizando la imagen original y las imágenes de salida, implementar:
    1. Un algoritmo que calcule la matriz de transición de cada canal.
    2. Un algoritmo que calcule el ruido de cada canal utilizando muestre computacional, generando un gráfico para estudiar la evolución del error y la convergencia.

3.Desarrollo

Enunciado 1

Para la resolución del ***enunciado 1*** usamos un método en la clase ***Calculador*** que calcula la correlación cruzada entre cada imagen con la imagen original y devuelve una lista de imágenes y sus respectivas correlaciones ordenada según las últimas de mayor a menor.

```
Public List<NodoImagen> ordenarimagenes ()
list<Nodoimagen> lista ; //Lista de las Imagenes
Imagen Original ; //Inicializa Imagen
Int [][] morig=orginal.getValoresMat ("Original")
for (i=0;i<5 ; i++){
        Imagen obtenida() ;
        int[][] mob =obtenida.getvaloresmat("Will"+i);
        NodoImagennodo=newNodoImagen(Obtenida,this.getcorrelacion(morig,mob,orginal.getancho(),original.getAncho()));
        //Calculo el valor de la correlación cruzada de las imagen
        Lista.add(nodo); //añado el nodo a la lista
}
Lista.sort(null); //Ordeno las imágenes por su correlacion
return Lista;
 }
```

Ilustración 1.1: *Pseudocódigo del método ordenarImagenes.*

Utilizando este algoritmo se llegó a que la imagen más similar es la imagen 1, lo cual quiere decir que es la que más zonas de valor similar tiene. Este ordenamiento se corresponde con lo observado por los integrantes del equipo ya que en las imágenes solo varían las sombras de la cara como se ve en la siguiente ilustración en la cual tenemos la foto ordenada por estas correlaciones de mayor a menor y la foto Original de Will.

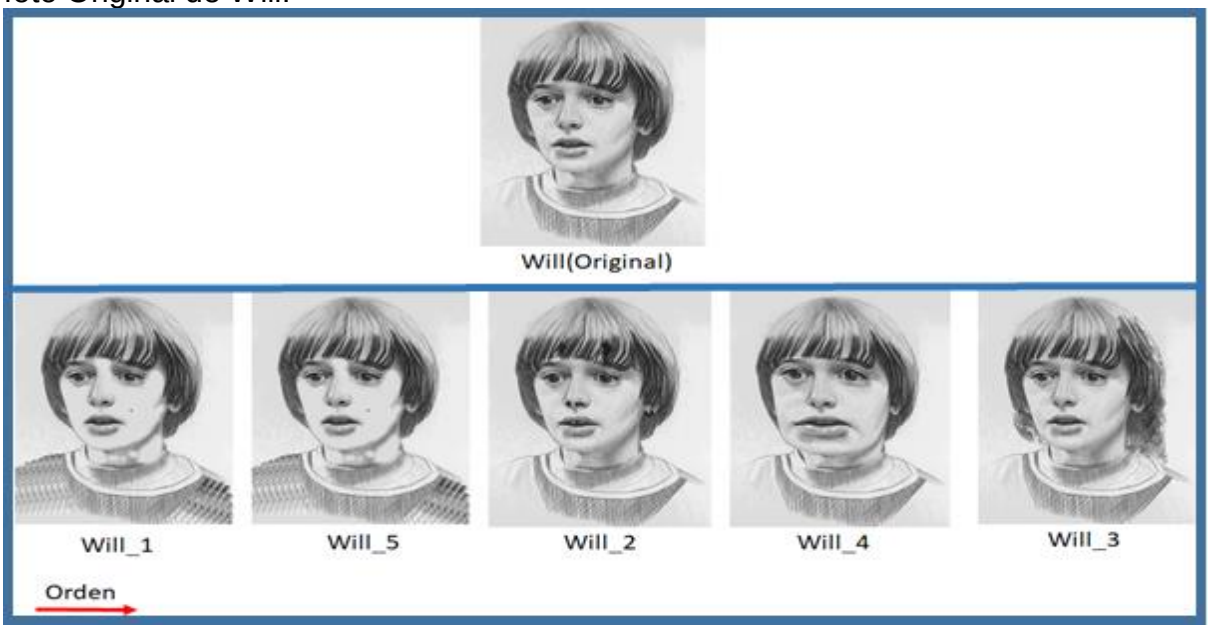

Ilustración 1.2: *Imagenes de Will ordenadas por factor de correlación.*

La imagen más similar después de la 1 es la 5 en la que hay más zonas distintas (lunar y ropa).

Además, podemos observar que la imagen 3 es la que más difiere en el algoritmo y esto se debe a que el fondo de la imagen original es blanco, mientras que el de la imagen 3 es muy oscuro. A pesar de esto, el niño de la imagen es muy parecido a Will y por esto la comparación del fondo es una deficiencia para el algoritmo en este caso.

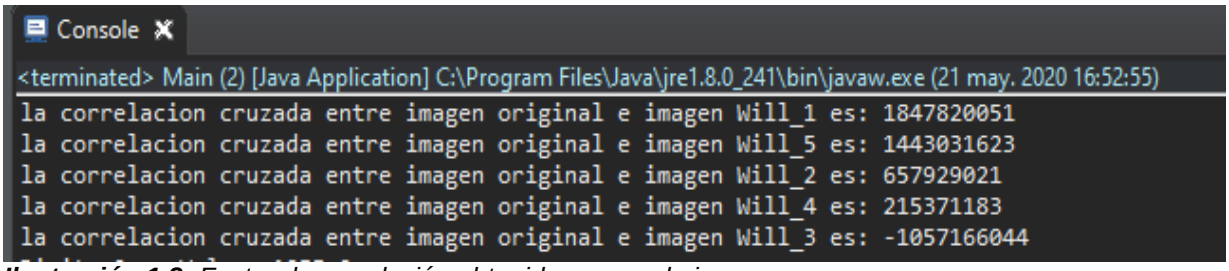

Ilustración 1.3: *Factor de correlación obtenido para cada imagen.*

Enunciado 2

Para la resolución del ***enunciado 2***. primero se cuantifican los valores de cada símbolo que fueron contados con el método ***getValoresMat*** , que devuelve los valores cada pixel de la imagen en una matriz. Con estos valores se calculó la media y el desvío estándar de cada una de las imágenes .

Las imágenes con las que se trabajó en este enunciado fueron la Imagen: Will _1 (la más parecida que por la correlatividad cruzada obtenidos en el ejercicio 1), una foto que tenía el oficial de la estación de policía y la foto original.

Para estas imágenes se obtuvieron los siguientes resultados e Histogramas:

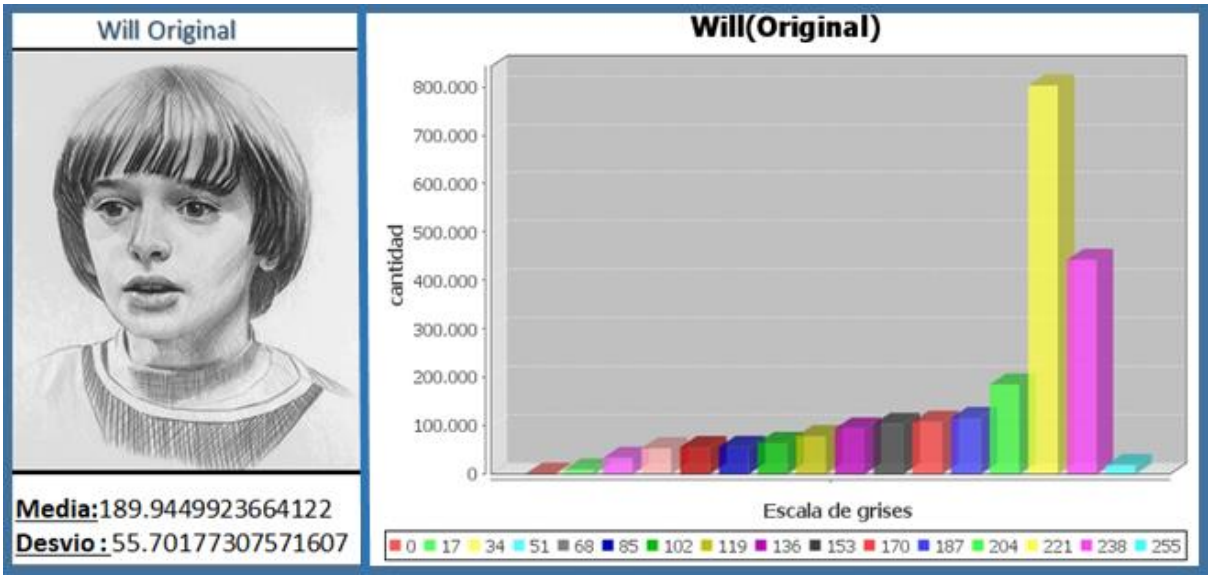

Ilustración 2.1: *Desvío, Media e Histograma de la imagen Original de Will.*

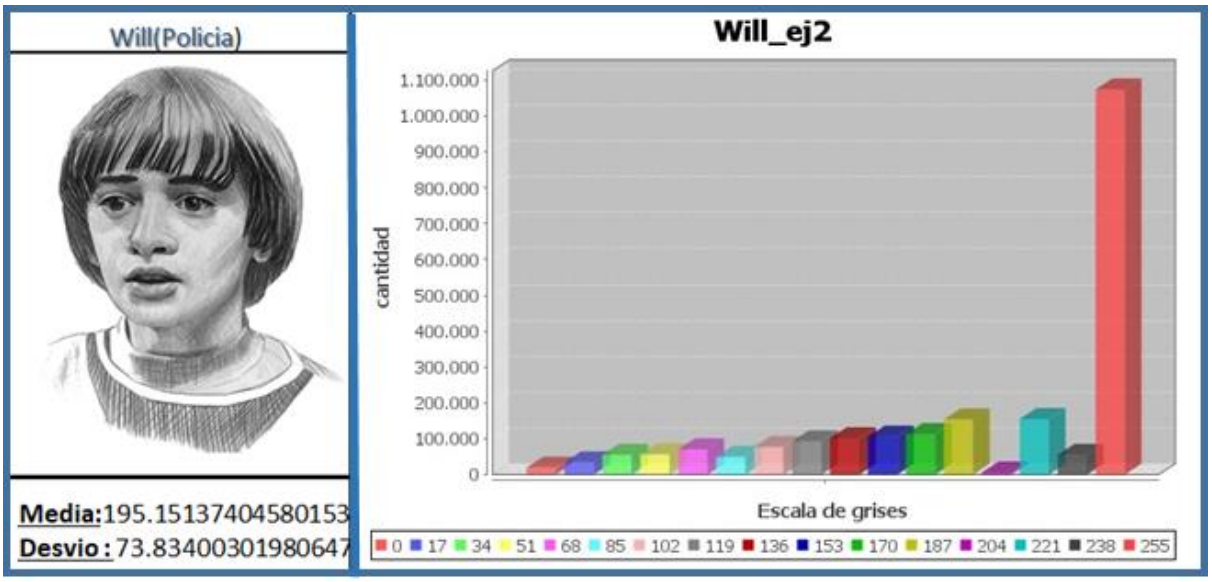

Ilustración 2.2: *Desvío, Media e Histograma de la imagen que tenía el policía("Will_ej2").*

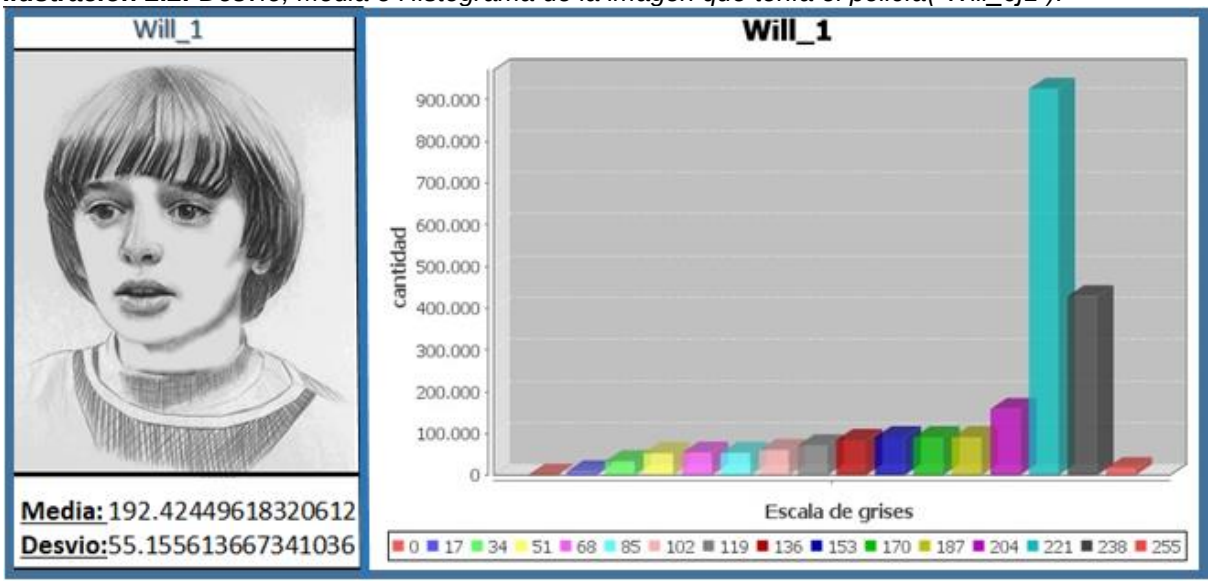

Ilustración 2.3: *Desvío, Media e Histograma de la imagen más parecida obtenida por el inciso 1 ("Wiil_1").*

A partir de estos resultados podemos deducir que el niño más parecido a Will es el de la imagen obtenida por el sistema. Esto se ve en que esta imagen y la original tienen mayor cantidad de los tonos de gris 221 y 238 mientras que la imagen aportada por el policía tiene muchísimos menos píxeles de estos colores y muchos más del tono 255. Además, en el resto de los tonos se puede observar también que los primeros dos histogramas se comportan de forma similar, mientras que en el tercero hay diferencias.

	Will Original	Will_ej2	Will_1
Media	189.94499236	195.151374045	192.42449618
Desvio	55.70177307	73.83400301	55.15561366

Tabla 2.4: *Desvío y Media de cada imagen*

Analizando los datos de la ***Tabla 2.4 vemos*** que las medias de cada imagen son similares, lo que nos dice que el promedio del valor de los símbolos que aparecen es similar en cada una de estas.
Ahora, si analizamos el desvío de cada una de las fotos, vemos que la imagen "Original" y la imagen "Will_1" tiene una dispersión de los símbolos menor que la imagen de "Will_ej2", ya que estas primeras tienen una diferencia del orden de los 55, en cambio, en la otra tiene una diferencia del orden de los 73, esto significa que hay más dispersión en los valores de los símbolos que aparecen en la foto "Will_ej2" que en las otras dos. Esto se puede observar gráficamente en los histogramas en los que se ve la cantidad de veces que aparece cada símbolo y sus valores, siendo visualmente más notorio que hay una diferencia entre estos símbolos.

Enunciado 3

En la resolución de este enunciado, creamos un algoritmo que calcula las probabilidades de ocurrencia de cada símbolo de cada imagen, y luego genera el árbol de Huffman semiestatico con el método ***generarArbol*** visto en la materia, correspondiente para esta Imagen, generando el código codificado para esta imagen y guardando esta codificación, las probabilidades de cada símbolo y los datos de la imagen en un archivo con el método ***codificar*** que se encuentran en la clase *Huffman.java*.
Desde este archivo, armamos el árbol con las probabilidades con el método ***decodificar,*** que, además, vuelve a generar la imagen y la guarda en la carpeta Sources. Teniendo el árbol listo para reconstruir la matriz, se guarda el código de decodificación que se utilizara más adelante para ver cómo se comporta con las otras imágenes.
En este enunciado, se nos pide comparar cómo se comportan las imágenes "Original", "Will_ej2" y "Will_1" con el código generado por la imagen original y "Will Ej2" con el código generado para su imagen, como se ve a continuación en la ***tabla 3.1***.
Las codificaciones que obtuvimos para cada uno de estos códigos fueron las siguientes:

Simbolo	Codigo Huffman Original	Probabilidad Imagen original	Codigo Huffman Policia	Probabilidad Imagen Policia
0	01100111	4.7373147732375E-4	0100110	0.008881903906600808
34	011000	0.015194881005837449	01101	0.02501930848675348
68	11000	0.025161652447238437	01000	0.03175348001796138
102	01011	0.029128423888639426	00011	0.034856757970363715
136	1111	0.04238796587337225	00001	0.04586124831612034
170	1101	0.048701841041760216	00101	0.050990121239335426
204	0100	0.08321239335428827	0100111	0.002772788504714863
238	10	0.19880422092501124	000000	0.024907049842837897
17	01100110	0.004271216883700045	010010	0.015535698248765133
51	11001	0.02414683430624158	01100	0.025796587337224966
85	01101	0.025297709923664122	000001	0.022533453075886844
119	01010	0.03527929950606197	00010	0.041772339470139204
153	1110	0.04666771441400988	0111	0.04777503367759318
187	0111	0.05222900763358779	0011	0.06918679838347552
221	00	0.360416255051639	0010	0.0699272563987427
255	0110010	0.008626852267624607	1	0.4824301751234845

Tabla 3.1 *Códigos de Huffman generados para la imagen Original y la imagen del policía(“Will_ej2”).*

Ahora con estos códigos realizamos un análisis, utilizando las tasas de compresión de los códigos de huffman que nombramos con anterioridad con las imágenes y obtuvimos los siguientes resultados:

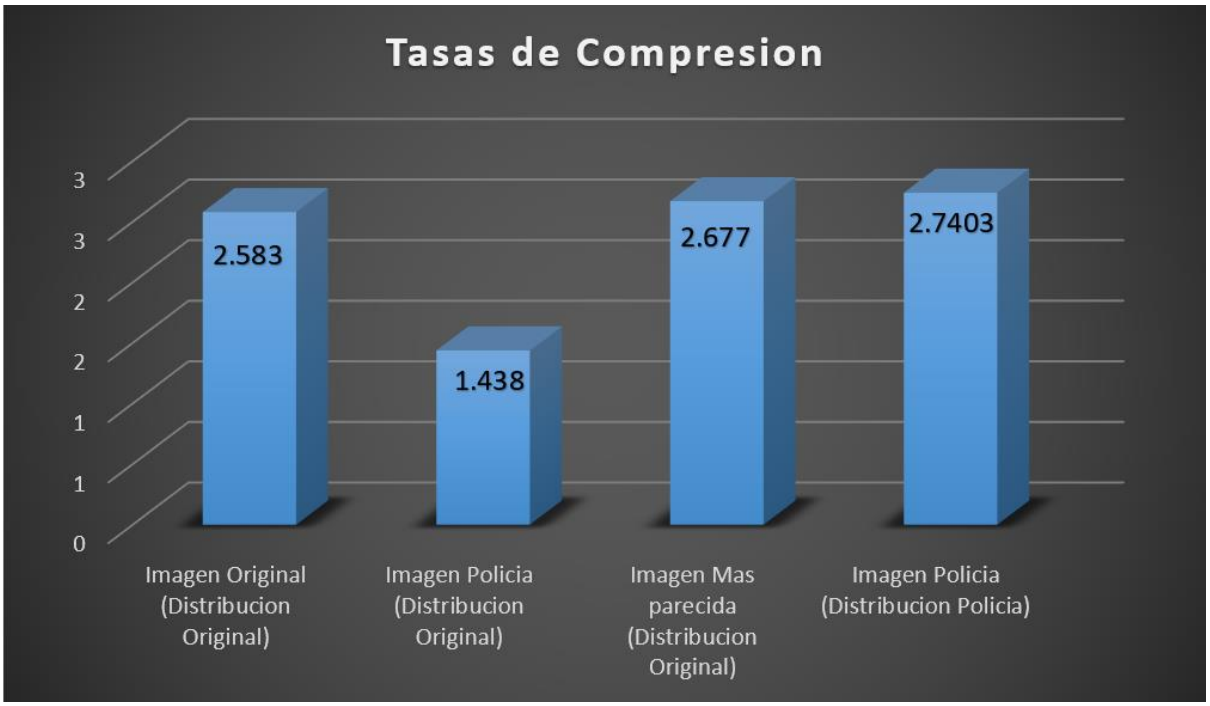

Gráfico 3.2 *Tasas de compresión de las imágenes.*

Como se ve en el gráfico, la imagen de la distribución original y la imagen más parecida, tienen buenos resultados en comparación con la imagen del policía con la distribución original, esto es debido a que el código de los símbolos que más se repiten en estas dos distribuciones son similares (ya que en el código de Huffman

los que tienen tamaño inferior, son los que símbolos que tienen las probabilidades más altas, lo que hace que el código generado sea más corto). En cambio, en la distribución del policía, esto no ocurre, y al no haber coincidencia en la la probabilidad de los símbolos, los símbolos que más se repiten no son los mismos, lo que hace que con ese árbol de Huffman el resultado sea más ineficiente que en los otros casos.
Cuando vemos la foto del policía con su distribución, vemos que ahora el resultado de esta compresión si es muy buena, dándonos un resultado óptimo.
ACLARACIÓN: Para sacar la tasa de compresión no utilizamos el tamaño original de la imagen, utilizamos, el alto por el ancho de la matriz, calculando un byte por cada dígito, ya que las fotos BMP utiliza codificación RLE, y queremos ver la tasa que tenemos con la imagen de tamaño original sin ningún tipo de compresión.

Enunciado 4
En este inciso se calculó la matriz de transición que describe a cada canal con el método ***matrizTrancision*** que se encuentra en la clase *Canal.java*. Luego, a partir de esta matriz de transición se calculó el ruido teórico para cada canal con el método ***destruido***, que luego será utilizado cuando comparemos estos resultados con los resultados obtenidos por muestreo computacional.
Los resultados teóricos obtenidos fueron:

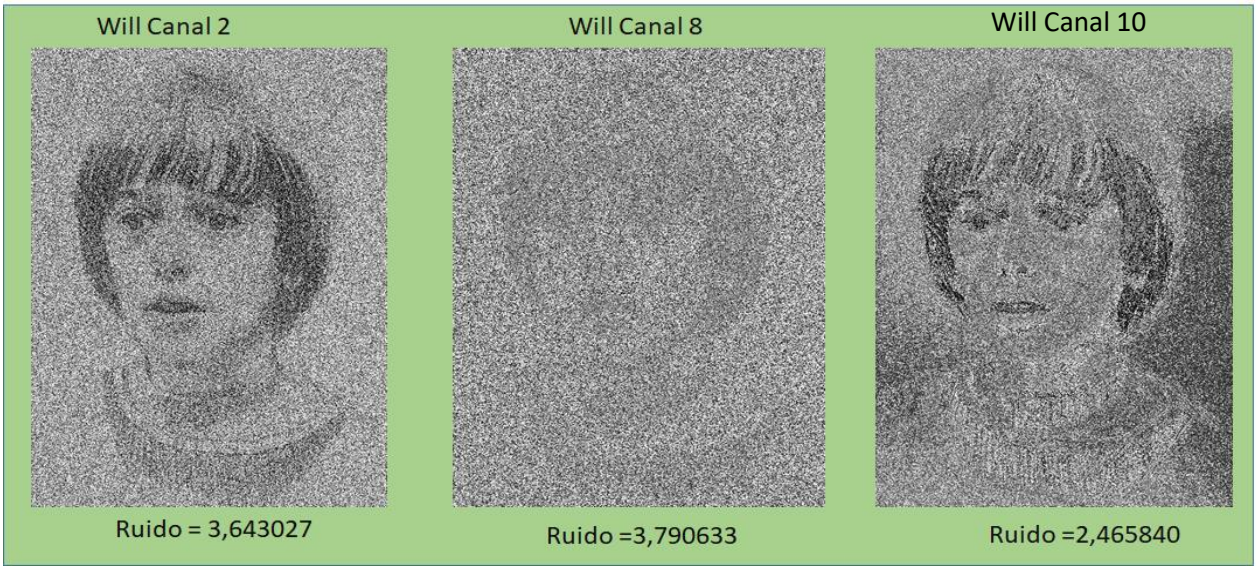

Gráfico 4.1 *Imágenes de los canales con el ruido calculado analíticamente.*

Para el segundo punto de este inciso, se calculó el ruido por muestreo computacional, con el método getRudiomuestreo que se muestra en el siguiente pseudocódigo y se obtuvieron los siguientes resultados:

```
el ruido por muestreo para el canal 2 es: 3.6058463406411 47
el ruido por muestreo para el canal 8 es: 3.7612345587773626
el ruido por muestreo para el canal 10 es: 2.4639872572496637
```

Ilustración 4.2 *Ruido calculado por muestreo computacional.*

Analizando la diferencia entre el ruido analítico y el ruido calculado por muestreo que calculamos con una épsilon igual a 0.001 vemos que el que más diferencia tiene en el ruido es el canal 2, con una diferencia de 0.03718 entre estos.

Este ruido se calculó con el método ***getRudiomuestreo***:

```
double getRuidoMuestreo( double epsilon){
int [] cant_X= int[Cant_Tonos];  //Arreglo para contar la cantidad de veces que aparece un simbolo
int [][] aciertos=Int[CantTonos][CantTonos]; //Matriz para sacar prob(y/x)
int tiradas =0; //Inicializo variables a utilizar
double[] evolución=double[501];
doublé ruido_ant=-1.0;
doublé ruido_act=0.0;
While (!Converge(ruido_act,ruido_ant,epsilon)|| tiradas<Min_pruebas){
//calculo las muestras de entrada y salida del canal de la proxima muestra
Int x=getX();
Int Y=getY(x);
// Actualizo variables tiradas , y los símbolos  en la matriz y el arreglo
tiradas++;
cant_x[x]++;
aciertos[x][y];
doublé tamp=0.0;
//Calculo del ruido actual
for (int i =0;i<Cant_Tonos;i++){
      temp=0.0;
     for  (int j=0;j<Cant_Tonos;j++){
            double prob= aciertos[i][j] / cant_x[i];
             if (prob > 0.0){
             temp+= prob * (-Math.log(prob) / Math.log(2)));
                 }
           }
            ruido_act+= cant_x[i])/((double)tiradas))*temp;
             }
            if(tiradas<=500)
           evolucion[tiradas]=Math.abs(ruido_act-ruido_ant);
  Graficos.getGraficoEvolucion(evolucion,canal.getNombre); //Genero el grafico error/Convergencia
 Return ruido_act;
}
```

Ilustración 4.3 *pseudocódigo de* ***getRudiomuestreo()****.*

Para cada canal se obtuvieron los siguientes gráficos de error por iteración de muestreo en el cual se ven los valores de las muestras desde la iteración N°1 hasta la iteración N°500 de la muestra con un error de épsilon de 0.001 :

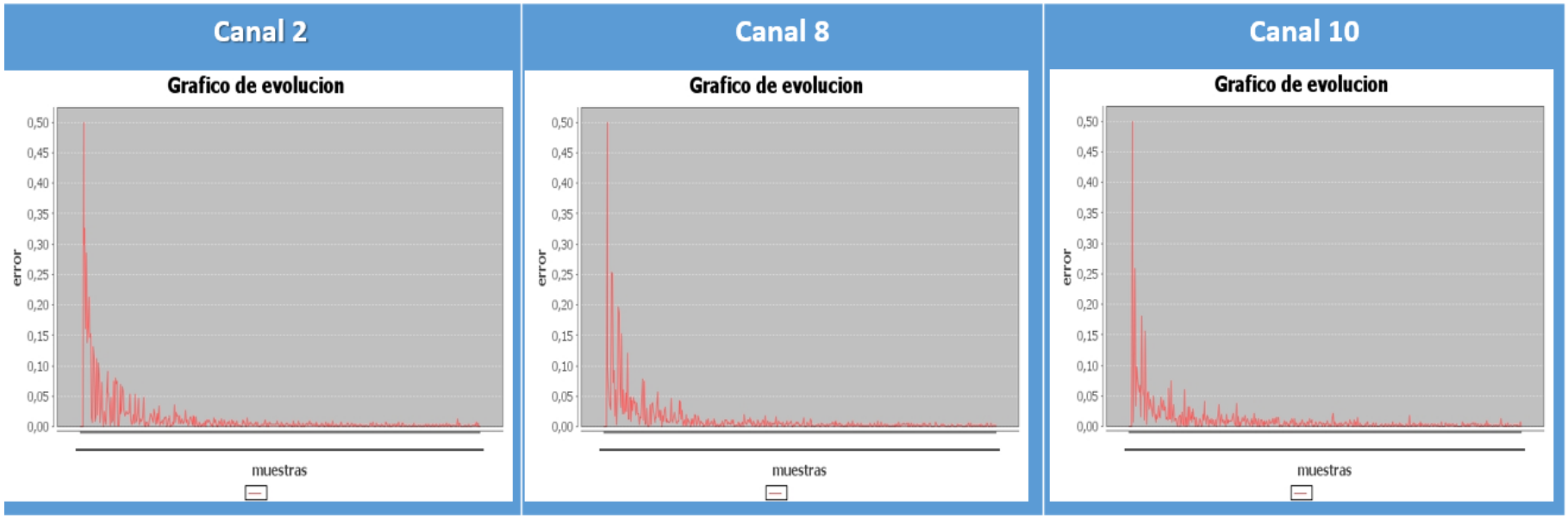

Ilustración 4.4 *Gráficos del error y la convergencia con el paso de la iteración de los canales.*

Como se ve en los gráficos al principio el error de convergencia del ruido en los 3 canales es de 0.50 aproximadamente, pero va disminuyendo hasta tener un error de 0.001793058 en la iteración 500 del canal N°2 que es hasta donde se muestra en estos gráficos. En los 3 canales el error del ruido se mueve de forma similar, aunque como se ve en los gráficos en el canal 10 posee fluctuaciones más volátiles, en comparación con los otros dos canales en los cuales no es tan volátil al principio. Luego de la Iteración 250 aproximadamente en los 3 gráficos, el error es similar pasando el error de 0.03 a partir de esta iteración.

4.Conclusion

Como resultado del trabajo practico especial de teoría de la información del año 2020 se concluye que para el problema a resolver los algoritmos que implementamos durante el transcurso del mismo, si bien cumplen con los objetivos propuestos por cada uno de los incisos, podrían ser mejores en términos de tiempo computacional, ya que la mayoría de estos posee una complejidad temporal de orden exponencial. Esto podría mejorarse utilizando más estructuras o paralelizando tareas que harían que se disminuya esta complejidad. Sin embargo, decidimos no realizar estos cambios ya que sería necesario un mayor tiempo de trabajo y no es la finalidad de este proyecto.

A partir del desarrollo de este proyecto se trabajó con diferentes herramientas de análisis de imágenes y fuentes de información, para calcular distintas métricas a partir de las mismas y la relación entre ellas. Esto nos dio un primer acercamiento a los algoritmos de compresión, codificación, decodificación, muestreo computacional y análisis de canales y a la gran cantidad de ámbitos en los que son aplicables.

Los resultados generales que obtuvimos en cada una de las aplicaciones fueron acordes a las expectativas que se tenían antes de empezar el proyecto. Visualizando las imágenes, teniendo en cuenta los aspectos de cada inciso, las salidas se correspondieron a lo que se esperó. Un ejemplo de estos, es que la tasa de compresión resulta mayor cuando la distribución de probabilidades usada para la codificación es la propia de la imagen que cuando es de otra. Por otra parte, el ruido obtenido mediante muestreo, se aproxima al ruido calculado analíticamente.