DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico III

Marche un telebeam Don Niembraaaaaa...

Métodos Numéricos
Primer Cuatrimestre - 2015

Integrante	LU	Correo electrónico

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

Índice

Resumen

En este trabajo se utilizaran distintas técnicas para obtener un re-escalamiento de imágenes. Se utilizara vecino más cercano, interpolación de polinomios bilineal, splines cúbicos, y distintas variantes de los métodos anteriormente mencionados. Se implementaran algoritmos para los mismos, dando la posibilidad de re-escalar las imágenes en distintos tamaños(siempre mayor al original). Se llevara a cabo una experimentación con su respectivo análisis. Como las imágenes obtenidas, no contienen integramente información original, se utilizaran las métricas de Error Cuadrático Medio (ECM) y Peak to Signal Noise Ratio (PSNR) para estudiar en forma cuantitativa la calidad de las mismas. También se considerara la calidad subjetiva, y el tiempo de computo.

Palabras Clave: re-escalamiento imágenes, interpolación, ECM, PSNR

1. Intoducción

En el presente trabajo se nos plantea como objetivo el re-escalamiento de imágenes, específicamente ampliarlas.

Se trabajara con imágenes en escala de grises por lo que dada una imagen de mxn, contendrá mxn pixels cada uno con un valor entre 0-255.

Para ampliar las imágenes, a partir de un valor $k \in \mathbb{N}_{>0}$ insertaremos entre la fila i y la fila i +1, k filas con i = 1, ..., m -1, y de manera análoga para las columnas. De esta forma obtendremos una nueva imagen con $(m-1)*k+m$ filas y $(n-1)*k+n$ columnas.

El problema que ahora se genera es ¿que valores asignar a los pixels de las columnas y filas agregadas?. Para esto, emplearemos distintos criterios de asignación de valor a partir de ciertos métodos.

En primera instancia consideraremos el método de vecino mas cercano, para esto el valor de un píxel nuevo sera igual a aquel cuya distancia a otro píxel, en una vecindad definida, sea la mínima. En particular la vencidad tomada será de aquellos cuatro valores más cercano con respecto a los pixels originales (pixels de la imagen sin ampliar, $imagen\ original$).

Otro método empleado fue mediante la interpolación de polinomios, esta consiste en que dada una terna de puntos (x_0, y_0), (x_1, y_1), ... (x_n, y_n) se busca obtener un polinomio P que los interpole es decir, que verifique $p(x_0) = y_0$, $p(x_1) = y_1$, ... $p(x_n) = y_n$. En general, la interpolación de una serie de puntos es usada para aproximar una función continua en un cierto intervalo. Dado que siempre existe un polinomio interpolador para $n+1$ puntos, de grado a lo sumo n que los interpole[1] , una de la forma de obtenerlo es mediante el método de interpolación de Lagrange, el cual se basa en construir primero los polinomios $L_{n,k}$ definidos como se indica en la ecuación 1.

$$L_{n,k} = \prod_{\substack{i=0 \\ i \neq k}}^{n} \frac{(x - x_i)}{(x_k - x_i)} \tag{1}$$

El polinomio de grado a lo sumo n que interpola los $n+1$ puntos se construye según la ecuación 2.

$$P(x) = \sum_{k=0}^{n} y_k L_{n,k}(x) \tag{2}$$

Por lo que el segundo método empleado consiste en usar interpolación bilineal entre dos puntos (x_0, y_0) y (x_1, y_1), en nuestro caso entre dos pixels, por lo que el polinomio interpolador de Lagrange sera de grado a lo sumo uno es decir, será una recta que pasa por dos puntos. En la ecuación 3

$$P(x) = L_{1,0}(x)y_0 + L_{1,1}(x)y_1 = \frac{x - x_1}{x_0 - x_1}y_0 + \frac{x - x_0}{x_1 - x_0}y_1 \tag{3}$$

o equivalentemente obtenemos una formula mas clara para el mismo, donde además podemos distinguir la pendiente y la ordenada al origen.

$$P(x) = \frac{y_1 - y_0}{x_1 - x_0}(x - x_0) + y_0 \tag{4}$$

Ses $f(x_i) = y_i$, en nuestro caso el valor de f depende de dos puntos, por lo que el valor del píxel p_{ij} se obtendrá extendiendo la ecuación 4 a:

$$p(i,j) = \frac{f(i,j_1) - f(i,j_0)}{j_1 - j_0} + f(i,j_0) \tag{5}$$

Otro de los métodos utilizados es el de splines cúbicos. Dada una función f definida en $[a,b]$ y un conjunto de puntos $a = x_0 < x_1 < \ldots < x_n = b$ un trazador cúbico S para f es una función tal que en cada subintervalo $[x_j, x_{j+1}]$ con $j = 0, 1, \ldots, n-1$ $S_j(x)$ es un polinomio cubico, y verifica:

- $S(x_j) = f(x_j)\ \forall\ j = 0, 1, \ldots, n$
- $S_{j+1}(x_{j+1}) = S_j(x_{j+1})\ \forall\ j = 0, 1, \ldots, n-2$

- $S'_{j+1}(x_{j+1}) = S'_j(x_{j+1}) \; \forall \; j = 0, 1, \ldots, n-2$
- $S''_{j+1}(x_{j+1}) = S''_j(x_{j+1}) \; \forall \; j = 0, 1, \ldots, n-2$
- Y una de las siguientes condiciones
  - $S''(x_0) = S''(x_n) = 0 \; \forall \; j = 0, 1, \ldots, n-2$ (condición natural)
  - $S'(x_0) = f'(x_0) y S'(x_n) = f'(x_n)$ (condición sujeta)

Escribiendo a los S_j en la forma $S_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3$, y planteando las condiciones anteriormente mencionadas se puede obtener un sistema lineal de $n+1$ ecuaciones y $n+1$ incógnitas, donde estas son los c_j. En el caso de la condición natural, que fue el utilizado en este trabajo práctico, mas específicamente se obtiene:

$$A = \begin{bmatrix} 1 & 0 & 0 & \cdots & \cdots & \cdots & 0 \\ h_0 & 2(h_0+h_1) & h_1 & \ddots & & & \vdots \\ 0 & h_1 & 2(h_1+h_2) & h_2 & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & & \ddots & \ddots & \ddots & 0 \\ \vdots & & & & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} \\ 0 & \cdots & \cdots & \cdots & 0 & 0 & 1 \end{bmatrix}$$

$$b = \begin{bmatrix} 0 \\ \frac{3}{h_1}(a_2 - a_1) - \frac{3}{h_0}(a_1 - a_0) \\ \vdots \\ \frac{3}{h_{n-1}}(a_n - a_{n-1}) - \frac{3}{h_{n-2}}(a_{n-1} - a_{n-2}) \\ 0 \end{bmatrix}; \quad c = \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix}$$

con $h_j = x_{j+1} - x_j$.

donde A es una matriz estrictamente diagonal dominante. Esto último implica que la matriz es invertible y por lo tanto el sistema tiene solución única. Una vez determinado c, se pueden obtener los a_j,b_j y d_j.

A partir de los resultados obtenidos en cada método buscaremos introducir alguna modificación en los mismos con el fin de obtener alguna mejora temporal y/o cualitativa. La forma en que se medirá la cálidad de la imagen obtenida será a través de el *Error Cuadrático Medio* (ECM) y *Peak to Signal Noise Ratio* (PSNR) los mismos se definen como:

$$\texttt{ECM}(I, \bar{I}) = \frac{1}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} |I_{ij} - \bar{I}_{ij}|^2 \tag{6}$$

y

$$\texttt{PSNR}(I, \bar{I}) = 10 \log_{10} \left(\frac{255^2}{\texttt{ECM}(I, \bar{I})} \right). \tag{7}$$

Con I e $\bar{I}$ la imagen original y la ampliada respectivamente, de dimensiones mxn. Como para útilizar esta métrica es necesario que las imágenes tengan igual dimensiones, aquellas con las que trabajaremos serán reducidas y luego ampliadas, con los métodos con los que trabajemos, a su tamaño original.

2. Desarollo

En este trabajo practico se aplicaran distintos métodos para re-escalar una imagen, es decir obtener una imagen ïgual", pero con una cantidad de pixeles mayor. Para esto en todos los casos, se ejecutara desde el programa en C++ un script de matlab que dada una imagen en cualquier formato, obtenga un archivo".csv" con la matriz que representa esa imagen convertida a escala de grises. Luego se utilizara el mismo para aplicarle los métodos para re-escalarla, obteniendo un archivo ".csv" con la imagen final, y nuevamente se llamara a un script de matlab para obtener una imagen en formato TIFF en blanco y negro.

Lo primero que se aplicara a la matriz de la imagen de entrada en escala de grises es aumentar su tamaño según un parámetro $k \in \mathbb{N}_{>0}$ que indica la cantidad de filas y columnas que seran insertadas entre cada par de puntos consecutivos, tal como se puede ver en la figura 5. Estas nuevas filas y columnas seran rellenadas provisoriamente con -1.

1	2	3
4	5	6
7	8	9

(a) Imagen original

1	-1	-1	2	-1	-1	3
-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1
4	-1	-1	5	-1	-1	6
-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1
7	-1	-1	8	-1	-1	9

(b) Imagen expandida

Figura 1: Expansión de una imagen para un k de 3.

Luego se aplicaran distintos métodos para rellenar la imagen.

2.1. Vecino mas cercano

Se llevaron a cabo tres versiones de este método. La original consiste en recorrer la matriz expandida sustituyendo en cada posición los -1 por el valor de la matriz original mas cercano. Se utiliza una función auxiliar que para cada posición nos devuelve el vecino mas cercano. Hay que notar que se puede dar el caso de que halla dos vecinos mas cercanos, en este caso el algoritmo implementado tomara alguno de ellos.

Una segunda versión considera no solo los valores originales como los mas cercanos, sino que en cada paso considera también los valores que ya fueron completados. Para esto es importante el orden en que es completada la matriz, para este trabajo es completada por filas de izquierda a derecha, de arriba hacia abajo.

En este caso el vecino mas cercano resulta ser el elemento de la izquierda o el de arriba, es por esto que el algoritmo se simplifica bastante. Para evitar el aglomeramiento de un número particular, es decir que un mismo número se repita en toda una fila, y en las siguientes de abajo, se decidió que en las columnas múltiplos de $k+1$ se tome como mas cercano al elemento de arriba, y en los demás casos al de la izquierda.

En el siguiente fragmento podemos encontrar el seudo-código del algoritmo efectivamente implementado.

Algoritmo 1 `vecinoMasCercano(expandida, k)`

```
1: for i ← [0 : cantidad_filas) do
2:     for j ← [0 : cantidad_columnas) do
3:         if expandida[i][j] == −1 then
4:             if j mod (k + 1) == 0 then
5:                 expandida[i][j] ← expandida[i − 1][j]
6:             else
7:                 expandida[i][j] ← expandida[i][j − 1]
8:             end if
9:         end if
10:    end for
11: end for
```

También se implemento una tercera versión como una variación de la anterior, es decir completando la matriz de izquierda a derecha de arriba hacia abajo, y considerar como posibles mas cercanos no solo a los originales, sino también a los elementos completados hasta el momento, pero que en lugar de decidir con algún criterio cual elemento tomar como mas cercano, tomamos un promedio de todos ellos.

2.2. Splines

Implementamos dos variantes distintas del método de interpolación por medio de splines: la primera es procesar la imagen de a bloques de un tamaño fijo, la segunda es ir variando el tamaño del bloque de acuerdo a ciertos criterios. En los dos casos vamos a trabajar con bloques de píxeles de la imagen expandida (inicialmente con valores -1 en los píxeles nuevos), calculando los splines correspondientes y pudiendo así hallar los valores de los píxeles nuevos. Empezaremos explicando la primera implementación.

Una vez definido el bloque de la imagen con el que se va a trabajar, la idea va a ser recorrer las *filas* del bloque que poseen píxeles de la imagen original (es decir aquellas filas que no todos sus píxeles tienen el valor -1). Supongamos que trabajamos con el siguiente bloque:

P_{00}	-1	-1	P_{03}
-1	-1	-1	-1
-1	-1	-1	-1
P_{30}	-1	-1	P_{33}

(a) Bloque

Figura 2: Bloque de imagen expandida con los valores de los píxeles originales. Indexamos desde 0.

Donde P_{00}, P_{03}, P_{30} y P_{33} son píxeles de la imagen original, cuyos valores ya conocemos. Lo que vamos a hacer es empezar tomando la primera fila (que no todos sus píxeles son -1) y calcularemos el spline correspondiente a los puntos $(0, P_{00})$ y $(3, P_{03})$ ya que 0 y 3 son las coordenadas de los píxeles originales en esta fila y P_{00} y P_{03} sus respectivos valores. Una vez que tenemos dicho spline, podemos utilizarlo para hallar los valores del segundo y tercer píxel de esta fila, luego de hacer esto la primera fila ya tendrá valores válidos en sus píxeles. Repetimos el mismo proceso pero ahora para la última fila y ahora con los puntos $(0, P_{30})$ y $(3, P_{33})$ (observemos que si tuvieramos un bloque de tamaño más grande,

tendríamos que repetir este proceso tantas veces como filas no agregadas existan). Para hallar los splines definimos los intervalos según las coordenadas de los píxeles conocidos (en este caso como hay solo dos entonces va a haber un solo intervalo) y utilizamos el algoritmo descripto en [1] para hallar el polinomio cúbico por cada intervalo. Nuestra imagen expandida quedaría de la siguiente manera:

P_{00}	P_{01}	P_{02}	P_{03}
-1	-1	-1	-1
-1	-1	-1	-1
P_{30}	P_{31}	P_{32}	P_{33}

(a) Bloque

Figura 3: Bloque de imagen expandida luego de hacer interpolación en la primera y última fila.

Ahora solo quedaría calcular los valores de los píxeles de las filas agregadas. Si observar con atención la Figura 3 vemos que podemos usar la misma idea pero ahora trabajando con las columnas. Es decir, al principio vimos que la primera y última filas tenían píxeles originales en los extremos que pudimos usar para interpolar y hallar los valores de los píxeles nuevos. Ahora podemos hacer exactamente lo mismo, pero generando un spline por columna y así pudiendo calcular los valores de los píxeles intermedios.
A continuación mostramos graficamente la secuencia de pasos que realiza el algoritmo para calcular los píxeles restantes.

P_{00}	P_{01}	P_{02}	P_{03}
P_{10}	-1	-1	-1
P_{20}	-1	-1	-1
P_{30}	P_{31}	P_{32}	P_{33}

(a) Spline para columna 1.

P_{00}	P_{01}	P_{02}	P_{03}
P_{10}	P_{11}	-1	-1
P_{20}	P_{21}	-1	-1
P_{30}	P_{31}	P_{32}	P_{33}

(b) Spline para columna 2.

P_{00}	P_{01}	P_{02}	P_{03}
P_{10}	P_{11}	P_{12}	-1
P_{20}	P_{21}	P_{22}	-1
P_{30}	P_{31}	P_{32}	P_{33}

(c) Spline para columna 3.

P_{00}	P_{01}	P_{02}	P_{03}
P_{10}	P_{11}	P_{12}	P_{13}
P_{20}	P_{21}	P_{22}	P_{23}
P_{30}	P_{31}	P_{32}	P_{33}

(d) Spline para columna 4.

Figura 4

En resumen lo que hacemos entonces es lo siguiente:
Sea B el bloque de la imagen expandida, T es el tamaño de dicho bloque y k la cantidad de píxeles a agregar entre cada par de píxeles originales, entonces

1. Para las filas $F(B)_0$, $F(B)_{k+1}$, $F(B)_{2(k+1)}$, ..., $F(B)_{T-1}$:
   a) Calcular S_0, S_1, ..., S_{T-1} utilizando los valores de los píxeles originales de cada una de las filas
   b) Utilizar S_0 para reemplazar los píxeles con -1 de la primera fila, S_1 para los de la fila $k+1$, ..., S_{T-1} para los de la fila $T-1$.
2. Para las columnas $C(B)_0$, $C(B)_1$, $C(B)_2$, ..., $C(B)_{T-1}$:
   a) Calcular S'_0, S'_1, ..., S'_{T-1} utilizando los valores de los píxeles originales de cada una de las columnas (algunos van a ser los originales de la imagen, otros los que calculamos en el paso anterior)

b) Utilizar S'_0 para reemplazar los píxeles con -1 de la primera columna, S'_1 para los de la segunda, ..., S'_{T-1} para los de la $T-1$.

Un algoritmo equivalente podría ser primero trabajar con las columnas 0, $k+1$, $2(k+1)$, ..., $T-1$. Calcular los respectivos splines, utilizarlos para hallar los valores de los píxeles intermedios de cada una de estas columnas y luego trabajar con las filas 0, 1, 2, ..., $T-1$. Esto lo podemos hacer debido a que entre cada par de píxeles originales contiguos se agrega siempre la misma cantidad de píxeles nuevos, independientemente si los originales estan en la misma fila o en la misma columna.

Ahora que ya sabemos como trabaja el algoritmo con un bloque particular de la imagen expandida, vamos a ver que hacemos cuando tenemos la imagen entera. Si tuvieramos una imagen de 512x512 por ejemplo y quisiéramos utilizar interpolación por splines para agrandarla dado un cierto k, como ya tenemos el algoritmo que procesa bloques de la imagen, una opción podría ser tomar un bloque de 512x512 que comience en el primer píxel de la primera fila y aplicar dicho algoritmo. De esta manera, al finalizar vamos a tener la imagen correctamente (sin píxeles cuyo valor sea -1) expandida. Sin embargo, quizás tenemos una imagen donde existe algún patrón definido en el cambio de tonalidad. Es decir, quizás podemos dividir la imagen en distintas partes del mismo tamaño y, al observar estas partes nos damos cuenta que podríamos expandir una independientemente de la otra, para hacer esto la idea es simplemente ir moviendonos sobre los bloques de la imagen expandida e ir aplicando el algoritmo sobre cada uno de ellos. Veamoslo con el ejemplo de la Figura 1 b) y tomemos un tamaño de bloque igual a 4 con respecto a la imagen expandida (los bloques siempre son cuadrados):

1	-1	-1	2	-1	-1	3
-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1
4	-1	-1	5	-1	-1	6
-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1
7	-1	-1	8	-1	-1	9

(a) Imagen expandida

1	-1	-1	2
-1	-1	-1	-1
-1	-1	-1	-1
4	-1	-1	5

(b) Primer bloque

2	-1	-1	3
-1	-1	-1	-1
-1	-1	-1	-1
5	-1	-1	6

(c) Segundo bloque

4	-1	-1	5
-1	-1	-1	-1
-1	-1	-1	-1
7	-1	-1	8

(d) Tercer bloque

5	-1	-1	6
-1	-1	-1	-1
-1	-1	-1	-1
8	-1	-1	9

(e) Cuarto bloque

Figura 5: Imagen expandida partida en bloques de tamaño 4.

Al observar las imágenes, nos puede llamar la atención el hecho de que cuando dos bloques están uno seguido del otro, la primera columna del segundo es la última del primero, al igual que cuando dos

bloques están uno debajo del otro, la primera fila del que está abajo es la última del que está arriba. Uno esperaría que los bloques sean disjuntos, sin embargo, si así fuera entonces no podríamos definir correctamente los polinomios cúbicos al construir los splines porque por ejemplo el segundo bloque de la Figura 5 comenzaría con una columna de píxeles donde todos valen -1, con lo cual el extremo izquierdo correspondiente a este intervalo no estaría definido. Esto nos sugiere que además, es necesario que la última columna de cada bloque tenga valores de los píxeles originales, es decir sea un múltiplo de $k + 1$.

Una dificultad con la que nos podemos encontrar es que, al ir moviendonos en bloques del tamaño fijado sobre la imagen expandida, suceda que nos pasemos de las dimensiones de dicha imagen. Cuando eso suceda lo que va a hacer el algoritmo es simplemente "retroceder" cierta cantidad de píxeles para formar un bloque del tamaño esperado. Más especificamente cuando hablamos de retroceder píxeles nos referimos a que si estamos queriendo procesar un bloque que supera la cantidad de columnas de la imagen expandida, entonces nos movemos horizontalmente en la imagen tantos píxeles como la diferencia entre la última columna del bloque y el ancho de la imagen. Si nos estamos pasando en la cantidad de filas de la imagen entonces nos movemos verticalmente tantos píxeles como la diferencia entre la última fila del bloque y el alto de la imagen.

2.3. Splines 2:

Para el caso anterior el tamaño de los bloques era fijo, si teníamos dos puntos conocidos $P_{i,j}$ y $P_{i,j+1}$, dentro del bloque con el que trabajamos, los valores intermedios serian calculados creando un spline por fila para estos puntos. Pero, ¿Que sucede si el valor de ambos difieren de manera considerable? Esto podría implicar que uno es muy claro y el otro no luego, al momento de ampliar la imagen aquellos nuevos pixels que se encuentren entre los $P_{i,j}$ y $P_{i,j+1}$ se verán afectados por estos dos, pudiendo tener un valor intermedio a estos.

Supongamos que ambos puntos representaban dos objetos distintos o se trataba de sus de bordes, entonces en la imagen ampliada nos gustaría que aun siga existiendo esta distinción, por ejemplo si se pasa de un color blanco a otro negro no nos gustaría que los pixels intermedio queden en alguna otra tonalidad. Con los splines cúbicos no siempre se puede evitar esto, aun cuando el tamaño del bloque no sea el de toda la imagen deberíamos tomar bloque muy pequeños para poder evitarlo. Por lo que para solucionar esto propusimos que el tamaño de un bloque, sea al comenzar, desde el primer punto conocido de la fila $P_{i,0}$ con la que se trabaja, hasta el punto anterior a un $P_{i,t}$ talque $\mid f(P_{i,0}) - f(P_{i,t}) \mid \leqslant a$ (siendo f la función que retorna el valor de un pixel p) con a un valor determinado. Luego, con la misma idea el bloque se tomara desde $P_{i,t}$ hasta un $P_{i,t+r}$ o hasta el final de la fila. Notemos que al igual que en el caso anterior una vez que se completan las filas podemos aplicar la misma idea a las columnas y así obtener completar los valores.

Pero entonces al utilizar spline cúbicos para interpolar por ejemplo desde $P_{i,0}$ a $P_{i,t-(k+1)}$ y otro para $P_{i,t-(k+1)}$ a $P_{i,t+r}$ los valores entre $P_{i,t-(k+1)}$ y $P_{i,t}$ aun no tienen ningún valor asignado, como queremos que en la imagen ampliada estos sean exactamente igual a $P_{i,t-(k+1)}$ o a $P_{i,t}$ y no un valor intermedio entre estos. Optaremos por completar a los mismos siguiendo la idea de vecino mas cercano definiendo como vecindad a los valores originales mas cercano.

Para este caso el tamaño de los bloques a tomar esta ligado a que valor de a utilizar, notemos que si usamos valores muy chicos, es decir permitimos una diferencia mínima, posiblemente terminemos aplicando el método de vecino mas cercano en casi toda la imagen y perdamos la aplicación de spline. Mientas que por el contrario, si permitimos diferencias muy grandes entonces caeríamos en la implementación anterior (solo splines). Para comprobar si esto sucede experimentaremos con varios valores y veremos si existe algún valor o rango de valores óptimos el cual hace reducir el ECM y si estas variaciones influyen o no en el tiempo de computo.

2.4. Interpolación Bilineal

La interpolación bilineal intuye que el valor correcto de cada pixel es un promedio de sus valores más cercanos, a priori, este método tendría que ser un poco más completo que `vecino mas cercano`
En primera instacia, dentro del contexto del presente trabajo, a la interpolacíon bilineal la consideramos como una técnica que consiste en rellenar los píxeles utilizando interpolaciones lineales entre píxeles consecutivos de la imagen original, primero completando aquellas posiciones correspondientes a filas, es decir, completando de a k filas y luego sobre la matriz resultante, completando aquellas posiciones correspondientes a columnas.
Por ejemplo tomando un fracción de 2x2 de una imagen, obtenemos lo siquiente:

P_{00}	-1	-1	P_{03}
-1	-1	-1	-1
-1	-1	-1	-1
P_{30}	-1	-1	P_{33}

(a) Bloque

Figura 6: Fracción de imagen expandida con los valores de los píxeles originales. Indexamos desde 0.

Luego interpolamos linealmente por filas, de a k filas,obteniendo:

P_{00}	P_{01}	P_{02}	P_{03}
-1	-1	-1	-1
-1	-1	-1	-1
P_{30}	P_{31}	P_{32}	P_{33}

(a) Bloque

Figura 7: Fracción de imagen expandida luego de interpolar por filas, de a k filas.

Finalmente, a la matriz resultante la interpolamos linealmente por columnas, todas las columnas, obteniendo:

P_{00}	P_{01}	P_{02}	P_{03}
P_{10}	P_{11}	P_{12}	P_{31}
P_{20}	P_{21}	P_{22}	P_{32}
P_{30}	P_{31}	P_{32}	P_{33}

(a) Bloque

Figura 8: Fracción de imagen expandida luego de interpolar por columnas, todas las columnas.

El polinomio de grado 1 para interpolar linealmente, lo definimos de la siguiente manera, suponiendo X_1 y X_2 pixeles originales, y X un pixel a rellenar:

$$f(X) = f(X_1) + \frac{f(X_2) - f(X_1)}{(X_2 - X_1)}(X - X_1) \tag{8}$$

Luego, ideamos distintas variantes de este método.
En primer lugar, pensamos que pasaría si en vez de interpolar con píxeles originales consecutivos, interpolemos ignorando un pixel, es decir, sin utilizar toda la información original de la imagen. Este razonamiento se produjo al pensar que pasaría en las imagenes donde no haya tanta variación de colores(en este caso, variaciones de tonos de grises).
Como la interpretación del método de interpolación bilineal es la denominación de cada pixel como un promedio de sus valores más cercanos, también pensamos que sería interesante interpolar por diagonales, en los sectores de la imagen que lo permitiera. Estos sectores quedan limitados en el centro de la imagen ya que en las esquinas no hay la cantidad minima de pixeles para interpolar(2 o mas pixeles). Ante esto, procedemos a usar el metodo bilineal original, para completar la imagen.
Por último también verificaremos que sucedería al interpolar bilinealmente de a bloques, es decir tomando fracciones de la imagen, en donde en las puntas estan los píxeles originales, y los demás serán calculados en base a estos 4 pixeles, siguiendo la siguiente ecuación:
Sean $Q_{11} = (X_1, Y_1)$, $Q_{12} = (X_1, Y_2)$, $Q_{21} = (X_2, Y_1)$, $Q_{22} = (X_2, Y_2)$, $R_1 = (X, Y_1)$, $R_2 = (X, Y_2)$ y $P = (X, Y)$, donde Q_{11}, Q_{12}, Q_{21} y Q_{22} son los 4 píxeles originales, P el pixel a rellenar y R_1, R_2 las proyecciones ortogonales de P a las rectas Y_1 y Y_2 respectivamente:

$$f(R_1) = \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21}) \tag{9}$$

$$f(R_2) = \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22}) \tag{10}$$

$$f(P) = \frac{y_2 - y}{y_2 - y_1} f(R_1) + \frac{y - y_1}{y_2 - y_1} f(R_2) \tag{11}$$

Q_{11}	R_1	-1	Q_{21}
-1	P	-1	-1
-1	-1	-1	-1
Q_{12}	R_2	-1	Q_{22}

(a) Bloque

Figura 9: Fracción de imagen expandida donde se muestra el ejemplo de interpolacion bilineal por bloques.

Ante las distintas variantes del método en estudio, promovemos que en cuanto a la variante de ignorar un pixel en el método de interpolación bilineal, si lo usamos para imágenes donde no haya grandes cambios de colores seguidos, el mismo debe devolvernos similares resultados (Objetivos y subjetivos) que las otras 3 variantes del metodo. Y para imagenes donde haya gran cantidad de cambios de color la variante debería dar peor que las otras 3.
En cuanto a las variantes de interpolar por diagonales y por bloques, deben dar similares resultados (Objetivos y subjetivos) que el método original, sin importa que imagen usamos, debido a que en la implementación se utilizan cuentas similares. Aunque, ya que en la variante original se realizan menos casteos de las variables, esperamos que el resultado sea levemente mejor en calidad objetiva que el de la variante por bloques.

Se dispondrá de una serie de experimentaciones sobre el siguiente conjunto de 10 imagenes:

- Las imágenes `imagen1.tiff`, `imagen2.tiff` y `imagen3.tiff` fueron elegidas por ser medianas y no poseer tanto cambio de colores muy marcados.
- Las imágenes `imagen4.tiff` y `imagen7.tiff` fueron elegidas por ser medianas y por tener personas en las mismas.
- Las imágenes `imagen5.tiff` y `imagen6.tiff` fueron elegidas por ser medianas y por tener muchas variaciones de colores.
- La imagen `imagen8.tiff` fue elegida por ser grande y por tener personas en la misma.
- La imagen `imagen9.tiff` fue elegida por ser grande y por tener muchas variaciones de colores.
- La imagen `imagen9.tiff` fue elegida por ser grande y no poseer tanto cambio de colores muy marcados.

Las imágenes utilizadas para este experimento se pueden encontrar en...
Las mismas fueron reducidas con el script de MATLab `reducirImagen.m`, para luego comparar lo que devuelve el método con la imagen original.
La experimentación empezará aplicando zoom con k = 2 a todas las imágenes, y luego empezaremos a variar el k para las últimas 3 imágenes, ya que al ser grande, hay mayor cantidad de k válidos. Sin perdida de generalidad, usaremos imágenes cuadradas y valores de k, talque $(anchoDeImagen + k)/(k + 1)$ de como resultado un número entero. Esto se asume para no trabajar con casos bordes de acuerdo a la esquematización del zoom dada por la catedra.

3. Experimentación

Para poder tener una idea de como se comportan las distintas implementaciones, hemos seleccionado un conjunto de imágenes (cada una con su particularidad que será explicada brevemente en la sección correspondiente) por cada método. De esta manera podemos ver, para ese conjunto, cual es la mejor variante en términos tanto cuantitavos (PSNR) como cualitativos (analizando visualmente como quedan las imágenes, artifacts y demás). Si bien esto nos da información sobre las variantes de cada método, se trata de una experimentación bastante aislada en el sentido de que permite comparar variantes de un mismo método y nunca relaciona un método con otro. Es por eso que decidimos, una vez terminada esta etapa, realizar una nueva experimentación seleccionando un nuevo conjunto de imágenes y probando con ellas la mejor variante de cada método, de esta manera podemos tener un candidato a la mejor implementación y vemos como se comportan distintos métodos.

El tiempo de cómputo está presentado en segundos y fue tomado sin tener en cuenta las operaciones realizadas en MATLAB y las operaciones de escritura en archivos, es solo el procesamiento realizado por los algoritmos.

3.1. Vecino Mas Cercano

Siguiendo los pasos explicados en el desarrollo se aplicaron los métodos de vecino mas cercano para un conjunto de 10 imágenes y un k fijo de 3, y se obtuvieron los siguientes resultados para la metrica PSNR:

Imagen	Pixeles	Original	Ignorando	Diagonal	Bloques
1	100x100	32.6180	30.0720	31.7492	32.6217
2	q	32.0807	31.3422	31.8774	32.0938
3	e	34.4520	32.9934	34.4501	34.4509
4	e	27.7731	23.6071	26.8081	27.7777
5	f	21.9590	19.7860	21.7686	21.9598
6	e	20.9380	19.7715	20.7258	20.9392
7	d	29.9870	26.6496	29.3290	29.9945
8	r	37.0023	31.9828	36.2431	37.0397
9	x	24.0090	21.4330	23.6539	24.0111
10	x	34.9894	31.8309	34.9878	34.9894

Tabla 1: Tabla de PSNR de las 10 imágenes ejecutadas en las 4 variantes del método de interpolación bilineal

Para estudiar la diferencia entre aplicar los métodos a dos imágenes distintas, pero de igual cantidad de pixeles, se hicieron pruebas para distintos valores de k sobre dos ejemplares.

Con la imagen monaLisa.tiff y confites.tiff, ambas de 257x257 pixeles, se obtuvieron los siguientes resultados:

k	Original	Dinámico	Promedio
1	28.73	28.73	30.70
3	24.90	22.02	21.32
7	20.86	18.22	17.19

Tabla 2: Resultados de PSNR de la imagen confites.tiff ejecutadas en las variantes del método de vecino mas cercano.

k	Original	Dinámico	Promedio
1	30.64	30.65	32.75
3	27.24	25.18	24.05
7	23.94	21.37	18.64

Tabla 3: Resultados de PSNR de la imagen monaLisa.tiff ejecutadas en las variantes del método de vecino mas cercano.

También para imágenes de distintas dimensiones en pixeles, se hicienron estudios de la métrica de PSNR.

Para la imágen de palabras.tiff de 641x361 pixeles se obtuvieron los siguientes resultados de PSNR para los distintos valores de k:

k	Original	Dinámico	Promedio
1	19.97	19.96	22.30
3	16.83	15.16	16.43
4	16.04	14.27	15.62
7	14.47	12.96	14.54
9	13.83	12.65	14.10

Tabla 4: Resultados de PSNR de la imagen palabras.tiff ejecutadas en las variantes del método de vecino mas cercano.

Estudiando cualitativamente a las imágenes puede notarse a simple vista la diferencia de calidad de las imágenes a las que se le aplica el método. En las siguientes figuras podemos ver como se pasa de una imagen legible a una ilegible variando el k.

Para la imagen de globo.tiff de 1425x1425, se obtuvieron los siguientes resultados para PSNR:

k	Original	Dinámico	Promedio
1	35.64	35.64	37.62
3	31.92	29.61	28.55
7	28.32	25.87	22.22
15	25.42	22.90	15.74

Tabla 5: Resultados de PSNR de la imagen globo.tiff ejecutadas en las variantes del método de vecino mas cercano.

Podemos ver que la de k15 de promedios... Cualitativamente hasta para k 3 la es indistinguible al ojo la calidad de las imágenes, y a partir de k 7, ya comienza a ser mas apreciable.

Para la imagen de tigre.tiff de 2821x1861 se obtuvieron los siguientes resultados para PSNR:

k	Original	Dinámico	Promedio
1	29.49	29.48	31.72
2	28.44	25.92	26.65
3	26.20	24.22	24.18
4	25.35	23.14	22.55
9	22.47	20.46	18.06
11	21.82	19.87	16.75
14	21.04	19.08	14.82
59	16.92	15.62	10.84

Tabla 6: Resultados de PSNR de la imagen tigre.tiff ejecutadas en las variantes del método de vecino mas cercano.

Para la imagen de venus.tiff de 4096x4096 se obtuvieron los siguientes resultados para PSNR:

k	Original	Dinámico	Promedio
1	33.39	33.39	35.42
3	29.98	28.15	28.42
7	27.16	25.39	23.50
15	24.98	23.33	16.26
31	23.18	21.40	8.93
63	21.42	19.53	8.58
127	19.66	17.71	8.51
255	17.27	15.78	8.50

Tabla 7: Resultados de PSNR de la imagen venus.tiff ejecutadas en las variantes del método de vecino mas cercano.

3.2. Bilineal

3.2.1. Resultados Objetivos

Los siguentes resultados fueron obtenidos mediante la experimentación de las variantes del método de interpolación bilineal, sobre el conjunto de 10 imágenes detallado en la sección correspondiente de Desarrollo.

Con k=2, obtuvimos los siguiente resultados de PSNR:

Imagenes	Original	Ignorando	Diagonal	Bloques
1	32.6180	30.0720	31.7492	32.6217
2	32.0807	31.3422	31.8774	32.0938
3	34.4520	32.9934	34.4501	34.4509
4	27.7731	23.6071	26.8081	27.7777
5	21.9590	19.7860	21.7686	21.9598
6	20.9380	19.7715	20.7258	20.9392
7	29.9870	26.6496	29.3290	29.9945
8	37.0023	31.9828	36.2431	37.0397
9	24.0090	21.4330	23.6539	24.0111
10	34.9894	31.8309	34.9878	34.9894

Tabla 8: Tabla de PSNR de las 10 imágenes ejecutadas en las 4 variantes del método de interpolación bilineal

Con k=2, obtuvimos los siguiente resultados de tiempo(en segundos):

Imagenes	Original	Ignorando	Diagonal	Bloques
1	5.60	4.96	4.92	5.08
2	5.90	5.93	5.91	5.96
3	4.96	4.92	4.93	4.94
4	6.44	6.00	5.95	6.00
5	5.91	5.92	5.90	5.92
6	4.93	4.98	4.93	4.95
7	4.92	4.93	4.91	4.94
8	18.21	18.52	17.75	18.90
9	24.35	24.44	23.82	24.55
10	27.07	27.11	26.52	27.43

Tabla 9: Tabla de tiempos de las 10 imágenes ejecutadas en las 4 variantes del método de interpolación bilineal

Variando el k, con las imágenes `imagen8.tiff`, `imagen9.tiff` y `imagen10.tiff` se obtuvieron los siguientes resultados:
Con la imagen `imagen8.tiff`:

k	Original	Ignorando	Diagonal	Bloques
3	34.1484	29.3108	33.3469	34.1729
5	30.6667	26.8999	30.1220	30.6836
7	28.3968	25.3233	27.9253	28.4112

Tabla 10: Tabla de PSNR de la imagen8 ejecutadas en las 4 variantes del método de interpolación bilineal, variando el k en 3, 5 y 7.

k	Original	Ignorando	Diagonal	Bloques
3	17.73	17.27	16.60	17.47
5	16.37	16.25	15.67	16.78
7	15.92	16.06	15.70	16.65

Tabla 11: Tabla de tiempos de la imagen8 ejecutadas en las 4 variantes del método de interpolación bilineal, variando el k en 3, 5 y 7.

Con la imagen `imagen9.tiff`:

k	Original	Ignorando	Diagonal	Bloques
3	22.4157	19.8676	22.1082	22.4174
5	20.4671	18.0751	20.1985	20.4688
8	18.7939	16.5599	18.5421	18.7957

Tabla 12: Tabla de PSNR de la imagen9 ejecutadas en las 4 variantes del método de interpolación bilineal, variando el k en 3, 5 y 8.

k	Original	Ignorando	Diagonal	Bloques
3	22.93	23.26	22.55	23.81
5	22.04	21.95	21.58	22.14
8	21.83	21.15	19.93	21.55

Tabla 13: Tabla de tiempos de la imagen9 ejecutadas en las 4 variantes del método de interpolación bilineal, variando el k en 3, 5 y 8.

Con la imagen `imagen10.tiff`:

k	Original	Ignorando	Diagonal	Bloques
1	38.2838	34.9337	38.2821	38.2838
5	30.9075	27.6721	30.9051	30.9075
8	27.4966	25.3160	27.4906	27.4967

Tabla 14: Tabla de PSNR de la imagen10 ejecutadas en las 4 variantes del método de interpolación bilineal, variando el k en 1, 5 y 8.

k	Original	Ignorando	Diagonal	Bloques
1	33.77	33.33	32.35	33.02
5	23.48	23.47	22.77	24.43
8	23.42	23.55	21.65	23.75

Tabla 15: Tabla de tiempos de la imagen10 ejecutadas en las 4 variantes del método de interpolación bilineal, variando el k en 1, 5 y 8.

3.2.2. Resultados Subjetivos

Con k=2, en las variantes del método original, por diagonales y por bloques se pudo observar un cierto efecto de desenfoque de la imagen en donde los contornos de los objetos no estaban bien definidos. Esto se ve en todas las imagenes excepto en la imagen5, que se nota muy despixelada, seguramente debido a la cantidad de colores y letras que presenta la misma. La siguiente comparación muestra lo recientemente enunciado.

Figura 10: Fracción Imagen 4 - Bilineal por bloques

Figura 11: Fracción Imagen 4 - Orginal

Figura 12: Fracción Imagen 5 - Bilineal por bloques

Figura 13: Fracción Imagen 5 - Orginal

Tambien es notable que el método con la variante de ignorar un pixel, en las imagenes que varia el color seguido, deja más detalles especiales de lo que esperabamos, como una imagen despixelada, o con franjas blancas. En cambio en las que no varía el color seguido, deja imagenes aceptables. En la siguiente comparación se observa lo recientemente detallado.

Figura 14: Fracción Imagen 4 - Bilineal ignorando un pixel

Figura 15: Fracción Imagen 4 - Orginal

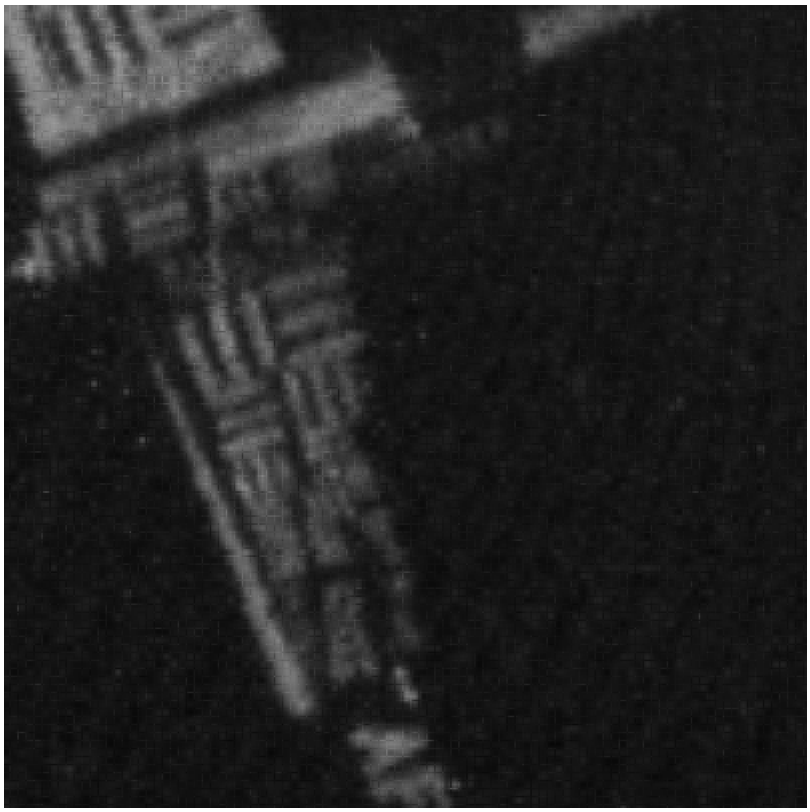

Figura 16: Fracción Imagen 2 - Bilineal por bloques

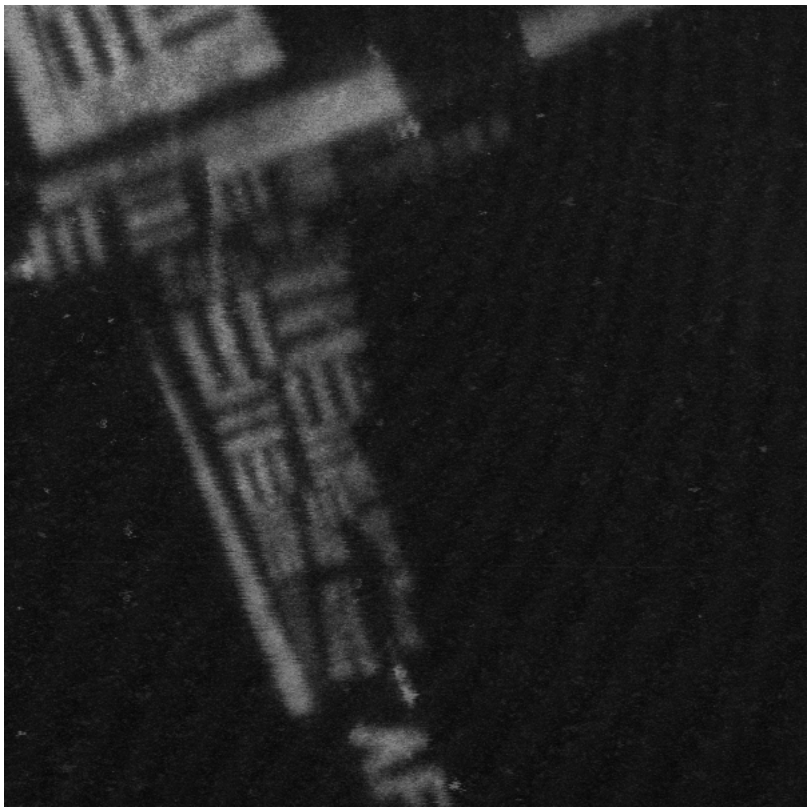

Figura 17: Fracción Imagen 2 - Orginal

Comparando las variantes original, por diagonal y por bloques sobre cada una de las imagen, no se

noto diferencias visibles al sentido humano comparando cada una de ellas, es decir comparando la misma foto pero con diferentes variantes.

3.3. Splines

Como mencionamos anteriormente en el desarrollo, implementamos dos variantes distintas de este método y por lo tanto experimentamos con ambas variantes. Las imágenes seleccionadas se encuentran en la carpeta Experimentación/Splines y a continuación explicaremos qué particularidad tiene cada imagen junto con los k que vamos a trabajar.

Tanque: esta imagen fue elegida para ver como afecta en los resultados el hecho de tener todo un terreno relativamente uniforme junto con un objeto más o menos mediano (en este caso el tanque), de tonalidad similar. Al igual que Cuadrícula y Grises, las dimensiones de esta imagen son 511x511 y trabajaremos con $k = 1, 2$ y 4.

Cuadrícula: nos pareció interesante incluir esta imagen por el hecho de contener, en su mayor parte, cuadrados y así poder ver el comportamiento del algoritmo de splines por bloques de tamaño fijo. Además como está compuesta básicamente por líneas, podemos ver que tan tolerante es al k, es decir, cuanta información se pierde.

Grises: al igual que Cuadrícula, esta imagen se puede dividir pero en rectángulos y es interesante ver si el algoritmo por bloques fijos puede funcionar relativamente bien, ya que los bloques que usa deben ser necesariamente cuadrados. Además, dado que existe un cambio de tonalidad por cada rectángulo adyacente, esto puede significar una experimentación interesante en la variante de bloques de tamaño variable porque sirve para evaluar que tan bien puede identificar ese cambio de tonalidad.

Lago:

Auto y Rosa: elegimos estas imágenes por el nivel de detalle que contienen. El auto sobre todo, tiene una cantidad importante de iluminación y reflejos, cosa que ninguna otra imagen hasta ahora tenía. Las dimensiones de Auto (4096x4096) nos permiten trabajar con varios valores de k que serán 2, 4, 6, 8 y 12 mientras que con Rosa (2047x2047) vamos a usar $k = 1, 2, 5$.

Piano: esta imagen fue elegida para ver como se comportan los algoritmos tanto con el reflejo de las teclas (que está bien delimitado) como con los bordes de las mismas. Sus dimensiones son 1549x1549 y trabajaremos con $k = 1, 2, 3$.

Ahora que ya explicamos los aspectos que van a compartir las experiencias de ambas variantes, pasamos a hablar específicamente de la variante del método que utiliza bloques de tamaño fijo. La idea va a ser tratar de ver las diferencias en las imágenes al tomar bloques de un tamaño u otro, eligiendo estos tamaños en base a la imagen con la cual vamos a estar trabajando. El enfoque que adoptaremos en general va a ser el siguiente: dada una imagen y un k, aplicar el método una vez con un bloque del tamaño de la imagen y otra vez con bloques de cierto tamaño, que será decidido según ciertos aspectos de la imagen. Por ejemplo, si la imagen es relativamente parecida en todas partes (porque se trata de un terreno por ejemplo) salvo por un area delimitada, el tamaño del bloque va a ser el tamaño de este area. De esta manera, estamos separando de alguna manera a la parte que difiere en la imagen de las partes que son parecidas. Decimos en general porque en algunos casos no se va a cumplir que la imagen posea esta uniformidad y ahí va a ser cuando cambiemos un poco este enfoque. En casos donde la imagen posee muchos detalles y es muy cambiante, vamos a tomar bloques de tamaño chico bajo la hipótesis de que como es probable que de un bloque a otro la imagen cambie significativamente, al tomar bloques reducidos vamos a estar descartando información sobre otra parte de la imagen que no aporta al area que queremos procesar, porque justamente en esa otra parte la imagen pudo haber cambiado mucho.

Comencemos analizando los resultados obtenidos para las imágenes de 511x511. Para la imagen Tan-

que.tiff, utilizamos bloques de un tamaño que aproxima a la mitad del tanque (esto es debido a que el tanque es rectangular y los bloques deben ser cuadrados por como está diseñado el algoritmo). Haciendo un análisis cualitativo de nuestras imágenes, comenzando por las que se procesaron con $k = 1$, podemos observar que entre la que trabaja con un bloque del tamaño entero de la imagen (256x256) y la que trabaja con bloques de 50x50 practicamente no hay diferencias que se puedan percibir fácilmente de manera visual. Si tratamos de ver muy detenidamente las diferencias entre ambas, podemos encontrar que, en la parte más oscura del tanque (más o menos por la mitad izquierda) la imagen procesada con bloques de 256 remarca un poco más el blanco del borde. Sin embargo, como ya dijimos, es practicamente imperceptible. Lo que sí se puede percibir notoriamente es la diferencia entre cualquiera de estas imágenes y la imagen original. Lo que más llama la atención son los detalles del tanque que se pierden totalmente, por ejemplo las lineas negras que tiene en la parte de arriba que se borronean. En la parte del cañón se produce un despixelado bastante fuerte, probablemente mayor al que se produce en las otras partes del tanque como por ejemplo en la de abajo donde están las ruedas. Una de las razones posibles que podrían explicar esto es que, justamente, los píxeles del cañón están aislados del resto del tanque y sus vecinos más cercanos corresponden a los píxeles del terreno, entonces por más que tomemos un bloque grande de 256x256 o bloques del tamaño de la mitad del tanque, seguramente en el/los bloque/s correspondiente/s al cañón del tanque van a estar tanto los píxeles del cañón como los del terreno y, seguramente van a predominar estos últimos. Obsevermos que en las lineas negras que mencionamos al principio sucede algo similar, se pierde mucha de esa información debido a que las líneas son muy finas y por lo tanto, en el bloque que tomemos que contengan estas líneas, van a predominar los píxeles grises. Todo esto sumado a que cuando realizamos la reducción del tamaño de la imagen ya se pierde una cantidad de información significativa y las partes donde están los detalles se ven más afectadas.

Para $k = 2$ y $k = 4$ lo que sucede es que las diferencias que notamos para $k = 1$ tanto entre las imágenes con zoom como entre alguna de ellas y la original aumentan bastante. En el caso particular de $k = 2$ ahora quizás es más fácil identificar diferencias entre la imagen aumentada con un bloque de 171 y la procesada con bloques de 30 sobre todo en la parte de las ruedas del tanque. Sin embargo, lo más relevante es que ahora la parte del cañón ya se comienza a confundir con el terreno. Y si pasamos a $k = 4$, para ambas imágenes, el cañón ya se perdió por completo.

En resumen, pareciera que las diferencias entre las imágenes procesadas con bloques relativamente chicos en relación al tamaño de las imágenes y las que se procesan con un bloque del tamaño de la imagen, van aumentando de a poco a medida que aumentamos el k. Si bien procesarlas de una u otra manera no evita (al menos en este caso) los problemas más relevantes (como el despixelado o que se confunda totalmente una parte de la imagen con el resto de los píxeles la rodean), existen algunas pequeñas diferencias como por ejemplo en las ruedas del tanque. Dado que estas diferencias no son visualmente muy relevantes, no podemos estar seguros de si contribuyen a una reconstrucción mejor o peor de la imagen ya que son diferencias muy pequeñas. Para resolver esto vamos a cuantificar un poco las cosas, a continuación mostramos el PSNR de cada imagen junto con el tiempo de cómputo que llevó procesarla:

Tanque			
K	Bloque	PSNR	T. de cómputo
1	256	29.9208	0.216126
1	50	29.9234	0.138024
2	171	27.5585	0.148907
2	30	27.5685	0.086371
4	103	25.4209	0.093326
4	20	25.4398	0.079059

Podemos ver como el análisis cualitativo que habíamos hecho sobre las imágenes, concuerda bastante con los números presentados en la tabla. Para empezar, vemos como efectivamente a medida que aumentamos el k el PSNR decrece y eso significa que el error entre la imagen esperada y la resultante es mayor.

También es cierto que las diferencias entre las imágenes procesadas con un bloque y las que procesamos con varios bloques aumentan a medida que aumenta el k ya que para $k = 1$ la diferencia es de 0.0026, para $k = 2$ de 0.01 y para $k = 4$ de 0.0189. Lo interesante de esto es que dado que estas diferencias son muy chicas, no eramos capaces de decidir si estaban ayudando a reconstruir mejor o peor la imagen porque visualmente era muy dificil. Ahora podemos ver que, en los tres casos el PSNR es mayor cuando trabajamos la imagen con bloques del tamaño de la mitad del tanque.

En cuanto al tiempo de cómputo, se observa que si bien todos los tiempos medidos son bastante chicos, al trabajar con las imágenes reducidas de mayor tamaño (k chico), el tiempo es mayor. Si bien trabajar con cualquier valor de k va a resultar en una imagen expandida del mismo tamaño, lo que sucede al trabajar con k pequeños es que los splines que se van a ir generando van a tener más información, con lo cual van a demandar un costo computacional más grande. Esto se puede ver también por la diferencia de tiempos que existe, para un mismo k, cuando se trabaja con bloques de tamaño chico y cuando se trabaja con un bloque grande. La tabla muestra que, en el primer caso, los tiempos son menores, lo que significa que concuerda con lo que dijimos recién, ya que al trabajar con bloques pequeños, si bien vamos a generar más splines, van a tener menos información. Por lo tanto una posible hipótesis en cuanto al tiempo de cómputo es que generar cierta cantidad de splines con muchos puntos es más costoso que generar quizás, una mayor cantidad pero con menos puntos cada uno.

Pasemos ahora a analizar la imagen Grises y veamos si sucede algo parecido a lo que pasó con la anterior. Para esta imagen, al igual que la anterior, vamos a trabajar con bloques de un tamaño cercano al tamaño de los rectángulos (además del caso de trabajar con un solo bloque que siempre lo usamos). En este caso vuelve a pasar que las diferencias entre las imágenes que trabajan con muchos bloques y las que trabajan con un solo bloque aumentan a medida que aumenta el k. Comenzando por $k = 1$, podemos ver que el tercer y cuarto rectángulos de la primera columna (en los otros es dificil de ver esto) difieren levemente entre la imagen 256 y 38: en el primer caso pareciera estar cubierto por líneas horizontales y en el extremo derecho también por líneas verticales, mientras que en el segundo caso pasa algo parecido pero en la parte de arriba de esos rectángulos no existen esas líneas. En principio es dificil tratar de pensar un posible motivo por el cual sucede esto ya que solo es apreciable en algunas partes de la imagen y no en todas. En cuanto a las diferencias con la imagen original, podemos observar que existe una leve variación entre el tamaño de los rectángulos: en la original los rectángulos de la columna izquierda y derecha parecieran ser un tanto más chicos que los de las imágenes resultantes y el del medio un poco más grande. Para $k = 2$ la diferencia en los tamaños de los rectángulos entre las imágenes resultantes y la imagen original es significativamente más notoria al igual que el cuadriculado que observamos antes en algunos rectángulos, que ahora se puede observar en otros como los que se encuentran más cercanos al centro de la imagen. Finalmente para $k = 4$ podemos ver que los dos defectos que nombramos antes se potenciaron bastante al igual que las diferencias entre las imágenes procesadas. A grandes rasgos se puede notar que la mayor parte de los rectángulos de la imagen 103 posee una especie de cuadriculado mientras que los de la imagen 38 solo líneas, en los extremos de ambos casos varía levemente este patrón. Vemos que al igual que nos pasó con la imagen Tanque, si bien existen diferencias entre la imagen procesada con un bloque solo y la que procesada con varios bloques, ambas comparten un defecto mayor que es el cambio en el tamaño de los rectángulos que es lo más apreciable a la vista. A continuación presentamos la tabla con los resultados sobre el tiempo de cómputo y el PSNR:

Grises			
K	Bloque	PSNR	T. de cómputo
1	256	37.3803	0.213046
1	50	37.4554	0.12934
2	171	34.0313	0.142121
2	25	34.0659	0.104159
4	103	31.7862	0.093746
4	15	31.8087	0.072207

Si bien el procesamiento por bloques sigue siendo superior al procesamiento con un solo bloque en términos de PSNR, al igual que con la imagen Tanque, aca la diferencia entre el PSNR de una y de otra va disminuyendo a medida que aumenta el k, a diferencia del caso anterior. Visualmente esto tiene algún sentido ya que, tanto en un caso o en otro a medida que el k aumentaba el rayado y el cuadrículado se hacían cada vez más visibles.

Con respecto al tiempo de cómputo, no tiene mucho sentido analizarlo mucho ya que era bastante esperable que nos de muy parecido al que tomamos con Tanque, porque estamos trabajando con una imagen de las mismas dimensiones y varios de los tamaños de los bloques son los mismos y otros son muy parecidos.

Ahora vamos a presentar el análisis correspondiente a la última imagen de 511, Cuadrícula, que representa un caso bastante patológico. Decimos esto porque al aumentar el k de 1 a 2 la cantidad de información que se pierde es muy grande, lo cual es bastante lógico porque la imagen está compuesta básicamente por líneas y estas líneas contienen una cantidad de píxeles bastante reducida. Sin embargo, nos pareció una instancia interesante debido a que como el algoritmo trabaja con bloques necesariamente cuadrados y salvo por algunos pocos sectores, esta imagen es practicamente un cuadriculado, ahora ya no es necesario tomar el tamaño del bloque como la mitad de algún sector (como sucedía en las imágenes anteriores). Para $k = 1$ lo que podemos observar si miramos la imagen que fue procesada tomando bloques del tamaño del cuadrado y la que fue procesada con un solo bloque del tamaño total de la imagen, es que esta última remarca las líneas. Si miramos en detalle, además de las líneas también remarca algunas partes de los números (se puede ver fácilmente en la parte inferior de los números 2). Si bien ambas variantes reconstruyen la parte de las líneas correctamente (con sus diferencias, pero podemos decir que es visualmente aceptable), el mayor problema viene con los números. Los números del eje vertical se entienden bastante pero los del eje horizontal parecen haber perdido una cantidad de píxeles importante. Probando con otros tamaños de bloque más grandes (por ejemplo cercanos al tamaño de uno de estos números del eje horizontal o tomar bloques más chicos que uno de los cuadrados de la imagen) se puede ver que la calidad de estos números no mejora, con lo cual empezamos a pensar que no es un problema de la elección del tamaño del bloque, si no de la cantidad de información perdida a la hora de reducir la imagen.

Para los casos con $k = 2$ y $k = 4$, las imágenes resultantes difieren mucho de la original porque la cantidad de información que se pierde al reducirlas es muy significativa, por el tipo de imagen que representan, esto se puede observar fácilmente al reducir la imagen original con esos valores de k y ver que queda algo realmente muy distinto a lo que uno esperaría, lo cual no pasa con $k = 1$ y es por eso que el método funciona mejor. Entonces podemos concluir que la diferencia que existe entre la imagen original y las procesadas con $k = 2$ y $k = 4$ no es una consecuencia del método utilizado si no del tipo de imagen, muy sensible a reducciones.
A continuación, presentamos la tabla correspondiente a esta imagen:

Cuadrícula			
K	Bloque	PSNR	T. de cómputo
1	256	11.3605	0.220708
1	9	14.1944	0.269276
2	171	9.2938	0.142723
2	17	10.0905	0.112803
4	103	8.4355	0.093314
4	17	9.0206	0.071696

Vemos como impacta el hecho de usar bloques del tamaño de los cuadrados para $k = 1$ (que, por lo que explicamos anteriormente sobre las reducciones, es el caso que más vale la pena analizar). Si bien visualmente la única diferencia significativa que existe entre estas imágenes es que una remarca más las líneas que la otra, éste fue el caso en el cual se pudo apreciar la mayor diferencia entre procesar por bloques y hacerlo solo con un bloque. Concluimos que esto sucedió justamente porque la imagen está dividida en cuadrados y los bloques con los que trabaja nuestro algoritmo deben ser cuadrados también.

En este caso, los tiempos medidos nos pueden estar diciendo algo más ya que vemos que el tiempo para $k = 1$ y un tamaño de bloque de 9 píxeles, es levemente mayor que el tiempo para un bloque de 256 y el mismo k. Esto contradice lo que veníamos pensando por los datos tomados de las imágenes anteriores, sin embargo, como se tratan de tiempos muy cortos y la diferencia no es demasiado grande, puede que sea un caso particular que no debe ser tenido en cuenta. Seguiremos analizando este aspecto en las próximas imágenes y probablemente nos digan algo más ya que al ser de dimensiones más grandes, los tiempos van a ser mayores.

Ahora pasaremos a analizar las imágenes más grandes, empezando por Lago. Dado que en esta imagen podemos identificar tres sectores que son el lago, el cielo y el terreno y son sectores relativamente grandes los tres (junto con la imagen en general, que también es grande), vamos a trabajar con tamaños de bloques bastante variables y no tan chicos como los de las imágenes anteriores. Además, para cambiar un poco, vamos a dejarlos fijos cuando varíe el k. Estos bloques van a ser de 50, 100 y 200.

Visualmente lo primero que podemos decir es que esta vez sí son realmente imperceptibles las diferencias entre las imágenes procesadas con un tramaño de bloque u otro, incluso para los k más grandes. Más aún, lo que sucede es que si bien las diferencias entre las imagenes resultantes y la imagen original existen y son apreciables, de alguna manera no afectan la calidad de la imagen para los primeros valores de k. Esto último se debe a lo que dijimos inicialmente: estamos trabajando con una imagen donde existen tres secciones bastante grandes y que además, los bordes no están bien definidos (es decir, no hay una linea bien definida que divida una sección de la otra) entonces si existe alguna diferencia en los bordes, visualmente no va a ser muy perceptible a menos que sea muy grande. porque en la imagen inicial el borde ya es irregular. A medida que se aumenta el k lo que notamos es que estas diferencias en los arboles se hacen más significativas y ya en $k = 5$ y $k = 7$ el cambio es bastante grande, incluso en el terreno se nota que la calidad es mala.
Dado que probamos el método con varios valores de k y varios tamaños de bloque, para presentar mejor la información usaremos una tabla por cada valor de k:

Lago k = 1		
Tamaño de bloque	PSNR	Tiempo de cómputo
50	43.8854	4.08434
100	43.9019	3.55443
200	43.9061	3.27132
1525	43.9126	7.16835

Lago k = 2		
Tamaño de bloque	PSNR	Tiempo de cómputo
50	38.9604	2.34028
100	38.9562	2.16114
200	38.9574	2.20069
1017	38.9569	4.48417

Lago k = 3		
Tamaño de bloque	PSNR	Tiempo de cómputo
50	36.8325	1.70029
100	36.8275	1.49301
200	36.8242	1.38925
763	36.8231	3.36569

Lago k = 5		
Tamaño de bloque	PSNR	Tiempo de cómputo
50	35.2486	1.21937
100	35.2415	1.23198
200	35.2409	1.12401
509	35.2398	2.38993

Lago k = 7		
Tamaño de bloque	PSNR	Tiempo de cómputo
50	34.5746	0.908093
100	34.5554	0.822251
200	34.5525	0.781545
382	34.5534	1.97297

En todas las tablas se cumple que el tiempo de cómputo correspondiente a procesar la imagen con un solo bloque grande, es mayor que utilizando bloques de tamaños más pequeños. Esta diferencia se acentúa cuando la imagen de la cual partimos es más grande y, a medida que vamos tomando k mayores la diferencia se va haciendo más chica. En conclusión, la diferencia es notoria, lo que sucede es que cuando trabajamos con imágenes chicas el tiempo de cómputo ya de por sí es chico y es por eso que tanto trabajar con un bloque grande como trabajar con bloques chicos va a demandar un costo computacional bajo. También podemos ver que esta diferencia es relevante cuando la diferencia del tamaño de bloque es necesariamente grande ya que para los tamaños 50, 100 y 200 que utilizamos, en $k = 1, 3$ y 7 nos dió un tiempo de cómputo mayor para 50 que para 100 y mayor para 100 que para 200.

En cuanto a los PSNR observados, no se puede concluir nada sobre si es mejor utilizar un tamaño de bloque u otro ya que son todos muy similares y, por ejemplo si miramos para $k = 1$, se registra un aumento a medida que aumenta el tamaño de bloque pero para $k = 3, 5, 7$ pasa exactamente lo contrario. $k = 2$ es el caso que nos incita a pensar que, para este tipo de imágenes, poco influye en el PSNR un tamaño de bloque u otro, ya que a diferencia de los demás k, no hay un crecimiento o decrecimiento estricto del PSNR según el tamaño del bloque.

Pasemos ahora a analizar la imagen Piano. Este es otro ejemplo para el cuál visualmente el método funciona muy bien. Las diferencias entre imágenes procesadas con distintos tamaños de bloque son, al igual que en el caso anterior, imperceptibles. Además de que, la única diferencia realmente notoria entre la imagen original y las resultantes, es el borde de las teclas negras (sobre todo de la tercera), el cual se despixelea un poco. Esta diferencia aumenta con el k. Una posible razón por la cual sucede esto es por la iluminación que tiene la imagen, el borde de la tercera tecla negra esta bien definido en la imagen original pero al ser blanco por el reflejo de la luz y tener de un lado negro y del otro gris, como el spline va a tener en cuenta toda esta información se va a terminar mezclando de alguna manera y al estar bien definido va a resultar notorio. Es por eso que no pasa tanto con las otras teclas donde el borde es más difuso. La tabla para esta imagen es la siguiente:

Piano			
K	Tamaño de bloque	PSNR	Tiempo de cómputo
1	200	47.8004	0.849895
1	775	47.8024	1.87312
2	100	44.8576	0.60743
2	517	44.8571	1.17809
3	50	42.8922	0.428638
3	388	42.8862	0.884801

Como podemos ver, no queda claro aca tampoco cuál es el tamaño de bloque a utilizar ya que, dependiendo el k a veces tomar un tamaño de bloque relativamente mediano (en comparación al tamaño de la

imagen) da mejor que procesar la imagen como un solo bloque mientras otras veces pasa lo contrario. En relación al tiempo de cómputo, se mantiene nuestra teoría de que procesando la imagen con un solo bloque grande es más costoso que con muchos de tamaño menor por la cantidad de información necesaria para construir cada spline.

Analicemos ahora la imagen Rosa. Vamos a estar trabajando con un tamaño de bloque bastante chico en relación al tamaño de la imagen, debido a que ésta contiene numerosos detalles y por lo tanto de un bloque a otro puede cambiar significativamente. Los k van a ser 1, 2 y 5. En este caso nuevamente las diferencias entre imágenes procesadas con un tamaño de bloque u otro no son relevantes. En cuanto a las diferencia con la imagen original, éstas son realmente apreciables recién para $k = 5$, entre ellas podemos encontrar la forma de algunas gotas y, en mayor medida, los bordes. El despixelado que se produce en los bordes es notable por el mismo motivo que en la imagen Piano, al tener gotas que tienen una tonalidad más clara que los borde, sombras que son mucho más oscuras o mismo el reflejo de la luz, se mezclan los tonos y, cuando el borde está bien definido se genera el despixelado. Si observamos la parte inferior izquierda, en los bordes no se produce despixelado o por lo menos no se puede ver, porque ya vienen difusos. La tabla para esta imagen es la siguiente:

Rosa			
K	Tamaño de bloque	PSNR	Tiempo de cómputo
1	40	43.871	2.01534
1	1024	43.9082	3.19154
2	20	39.0773	1.50334
2	683	39.1616	2.03111
5	10	32.1426	0.967969
5	342	32.2075	1.07513

Al parecer nuestra hipótesis de que el método podía funcionar mejor tomando la imagen de a pequeños bloques por tener muchos detalles, no era cierta. Esto se ve en los números dado que para los tres valores de k utilizados, obtenemos un PSNR mayor cuando procesamos la imagen con un solo bloque del tamaño de la misma. También se probó con valores de tamaño de bloque mucho más reducidos como 5x5 y aún así seguía dando mayor PSNR procesando la imagen con un solo bloque, aunque al igual que sucede con estos resultados, las diferencias de PSNR no fueron demasiado grandes.

Finalmente analizamos nuestra última imagen para Splines: Auto. Dado que el tamaño de esta imagen es el más grande, vamos a trabajar con más valores de k, que serán los siguientes: 2, 4, 6, 8 y 12. Dado que esta imagen tiene muchísimos detalles y, considerando los resultados obtenidos con la imagen Rosa, esta vez decidimos tomar un tamaño de bloque más. De esta manera apuntamos a tener un tamaño chico, otro mediano-grande y el que siempre usamos del tamaño de la imagen, para poder ver si vuelve a suceder que gana el tamaño del bloque mayor o fue un resultado particular de la imagen anterior. Los tamaños de los bloques los vamos a mostrar explicitamente cuando veamos la tabla ya que varían según el k.

Una vez más, sucede que las diferencias entre las imágenes procesadas tanto con bloques chicos como medianos y un único bloque se llegan a apreciar bien recién para $k = 12$. Las diferencias con la imagen original hasta $k = 6$ se podría decir que son aceptables ya que se observan más que nada en las partes donde la imagen refleja la luz y esto genera un efecto de como si se hubiera movido de alguna manera este reflejo. Para $k = 8$ y $k = 12$ ya podemos ver que los bordes se empiezan a despixelar bastante, es fácilmente apreciable en los del parabrisas, sobre todo en el que está sobre el reflejo de la luz y hace que se mezcle la parte clara con la oscura.
Para poder ver si efectivamente sucede lo mismo que con la imagen Rosa, con respecto a los bloques y el

PSNR, a continuación mostramos las tablas correspondientes a esta imagen:

Auto k = 2		
Tamaño de bloque	PSNR	Tiempo de ejecución
50	38.3178	4.11688
300	38.3146	3.50407
1366	38.317	8.08269

Auto k = 4		
Tamaño de bloque	PSNR	Tiempo de ejecución
40	34.0937	2.63596
250	34.0984	2.44113
820	34.0993	4.97891

Auto k = 6		
Tamaño de bloque	PSNR	Tiempo de ejecución
30	31.806	2.07926
200	31.8033	1.51972
586	31.8026	3.8546

Auto k = 8		
Tamaño de bloque	PSNR	Tiempo de ejecución
20	30.2226	1.93031
150	30.2372	1.83401
456	30.2389	3.22826

Auto k = 12		
Tamaño de bloque	PSNR	Tiempo de ejecución
10	28.2243	2.01117
100	28.2178	1.44431
316	28.2196	2.69884

Al igual que sucedió con la imagen Rosa, fueron pocos los casos en donde tomar bloques de tamaño pequeño significó una mejor en el PSNR, exactamente sucedió en $k = 2, 6$. Las diferencias entre el tiempo de cómputo procesando la imagen con un bloque grande y procesándola con muchos de tamaño menor sigue siendo significativa y creciente cuando disminuye el k.

Como conclusión de toda esta extensa experimentación, podemos decir que cuando tenemos una imagen que podemos dividir en bloques cuadrados de igual tamaño (como lo fue Cuadrícula), conviene utilizar bloques precisamente de ese tamaño. Ahora bien si nuestra imagen es grande y sabemos poco sobre ella, a nivel PSNR da lo mismo si utilizamos muchos bloques o un bloque grande para procesarla, sin embargo es preferible utilizar bloques relativamente chicos porque, según lo observado en la experimentación, el tiempo de cómputo va a ser menor. En resumen, utilizar bloques de tamaño chico parece ser la mejor alternativa.

3.4. Tiempo de computo

Luego de hacer pruebas con distintas imágenes nos dimos cuenta que el tiempo de computo de cada método depende solo de la cantidad de pixeles de la imágen en cuestión, lo cual es esperable ya que los algoritmos desarrollados no tienen saltos condicionales según el contenido de los pixeles. En la figura 3.4 encontramos una comparación entre el tiempo de computo de cada método según la cantidad de pixeles de la imágen.

Podemos ver que uno es mas rapido que el otro, sin embargo el otro tiene mas calidad

4. Discusion

4.1. Vecino Mas Cercano

4.2. Metodo Bilineal

Con las imágenes donde no había cambios tan seguidos de colores, se pudo verificar que la variante de ignorar un pixel, retorno un valor similar de PSNR, con diferencias de 1 0 2 puntos en relación a las otras variantes. Nunca se obtuvo mejor valor de PSNR. En cuanto con las imágenes que si tenian cambios seguidos de color, el valor de PSNR dio muy por debajo de lo que dieron las otras variantes, llegando a diferencias de entre 4 y 6 puntos. Ademas se encontraron artifacts muy notorios en las imágenes como los expuestos en la sección de resultados.
Tambien los resultados mostraron que la variante original y la de por bloques, casi devolvieron similares valores de PSNR, con minimas diferencias, siempre obteniendo mayor valor la variante por bloques. Suponiamos que iban a dar similares pero que la variante original iba a dar siempre mejor que por bloques debido a que en la implementación se realizan menos casteos de variables y eso podria haber arrastrado errores de precisión.
En cuanto a los valores de PSNR de la variante de diagonales, siempre obtuvimos valores con diferencias de a lo sumo 1 punto, lo que marcaría que las imágenes elegidas posiblemente no tengan una continuidad diagonal de color en todos los sectores.

Para destacar es que los resultados arrojaron que a medida que el k aumenta, si la imagen posee bastantes cambios abruptos de colores, se obtendrán imagenes con menos calidad objetiva y subjetiva.
Para finalizar con el estudio de este metodo, concluimos que la mejor variante en cuanto tiempo, calidad objetiva y calidad subjetiva es la variante por bloques.

5. Conclusiones

Referencias

[1] Burden R. L., Faires J.D.- Numerical Analysis