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
```

Una tercera versión calcula los promedios.....

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. Resultados

3.1. Bilineal

3.1.1. 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

Cuadro 1: 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

Cuadro 2: 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

Cuadro 3: 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

Cuadro 4: 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

Cuadro 5: 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

Cuadro 6: 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

Cuadro 7: 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

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

3.1.2. 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

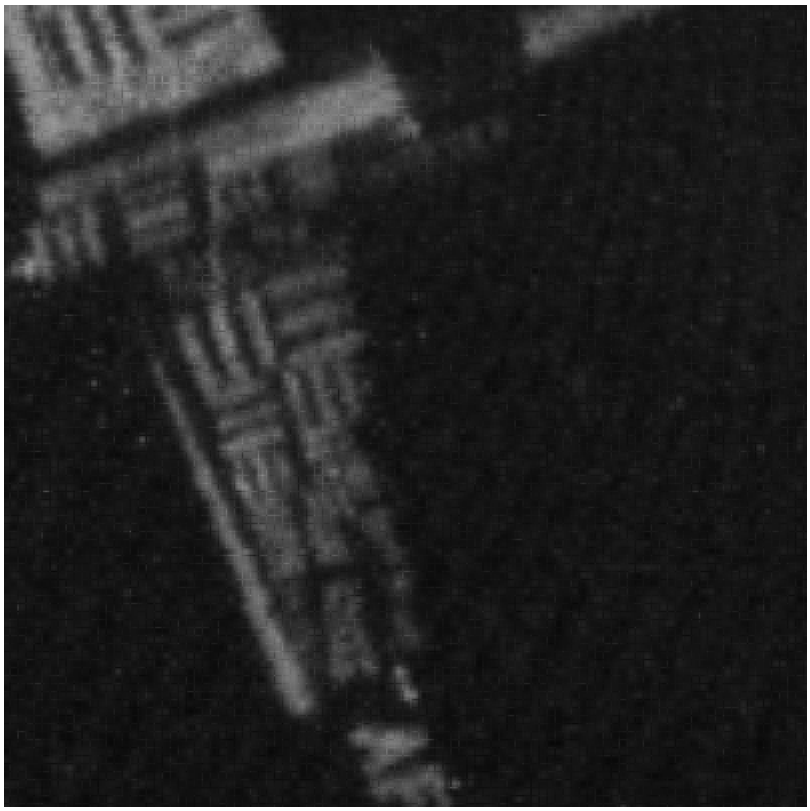

Figura 16: Fracción Imagen 2 - Bilineal por bloques

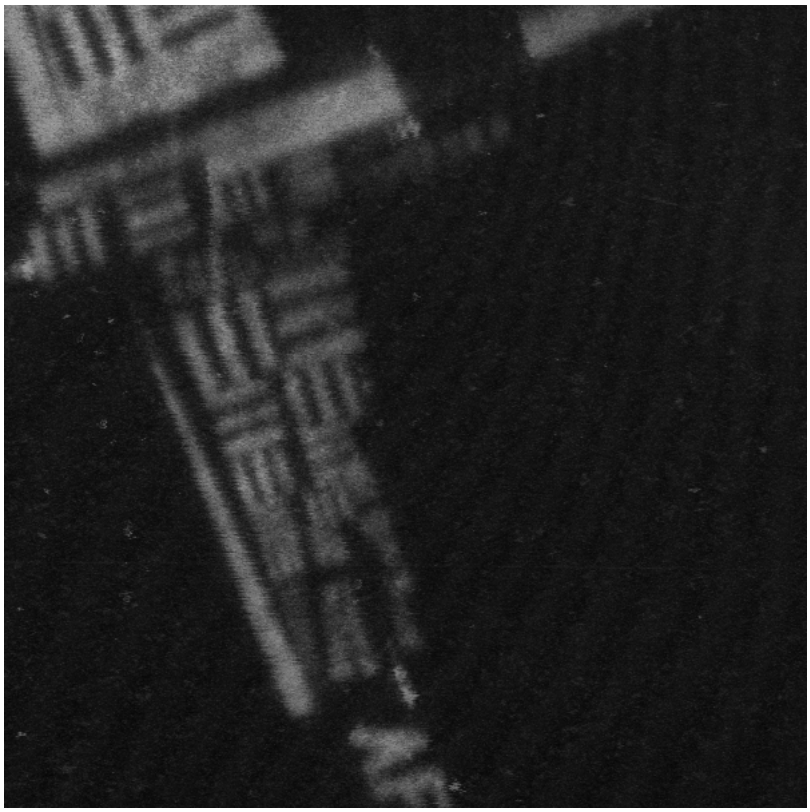

Figura 17: Fracción Imagen 2 - Orginal

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.

4. Discusion

4.1. 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