APÉNDICE A

EXPLICACIÓN DE PROGRAMAS

Este Apéndice tiene por objetivo la explicación de los programas que se realizaron para el presente trabajo. Todos los comandos y funciones, así como su estructuración son propios del ambiente de Matlab.

Programa para calcular la transformada de distancia

En las siguientes líneas se describe un posible programa para calcular la transformada de distancia de una imagen de 64x64 que se ha leído previamente (*image*). Primero se debe definir una función cuyo nombre sea el mismo del programa y su argumento sea la imagen que se ha leído previamente y a continuación una variable global (para que pueda ser usada por otros programas) donde se guardará su transformada de distancia:

```
function DistanceTransform(image)
global dt
```

donde *function* es el comando de Matlab para indicar una nueva función y *global* se usa para establecer una variable global. En seguida se procede a detectar el contorno de la imagen con el método *canny*:

```
dt=edge(image, 'canny');
```

por comodidad guardamos el contorno en la variable que tenemos (*dt*). *Edge()* es la función que trae Matlab en su *Image Processing Toolbox* para detectar contornos, y los argumentos necesarios son el nombre de la imagen y el método de detección de contorno preferido. Este método nos da como resultado una matriz binaria de 64x64 con 1's en los píxeles que definen a la imagen y 0´s en los demás. Como queremos que estos valores sean al revés, simplemente aplicamos el operador binario NOT (~):

```
dt=~dt;
```

y como se quiere que los píxeles que no forman parte del contorno tengan un valor grande -255-, procedemos a multiplicar la matriz por este número:

```
dt=255*dt;
```

donde obviamente sólo valdrán 255 aquellos píxeles que anteriormente valían uno y los demás seguirán valiendo 0. Aquí automáticamente la matriz deja de ser binaria. Ya preparada la matriz según el algoritmo, ahora se procede a calcular la transformada de distancia haciendo el barrido de ambos operadores en toda la matriz, excluyendo el cálculo de las distancias de los píxeles de las orillas para facilitar el proceso:

```
for i=2: 63,
   for j=2: 63,
      neighbors=[dt(i-1,j-1)+1.4142,dt(i-1,j)+1,dt(i-1,j+1)+1.4142,dt(i,j-
```

```
1)+1,dt(i,j),dt(i+1,j-1)+1.4142];
      dt(i,j)=min(neighbors);
   end
end
for i=63:-1:2,
   for j=63:-1:2,
      neighbors=[dt(i-1,j+1)+1.4142,dt(i,j),dt(i,j+1)+1,dt(i+1,j-
1)+1.4142,dt(i+1,j)+1,dt(i+1,j+1)+1.4142];
      dt(i,j)=min(neighbors);
   end
end
```

donde los primeros *for* anidados son para hacer el barrido del primer operador y los siguientes para hacer el barrido del segundo operador. Aquí en la variable *neighbors* se guardan las distancias de los píxeles tomados en cuenta –por cada operador- correspondientes a la métrica euclidiana y en seguida el píxel actual se iguala al valor mínimo de *neighbors* con la función *min()*, esto obedeciendo al algoritmo de la transformada de distancia descrito en el capítulo 2. La transformada de distancia de la imagen *image* queda alojada en la variable global *dt*.

Para hacer uso de este programa, se guarda con extensión .m (al igual que todos los programas que se usarán) en una carpeta con destino conocido por Matlab y se ejecuta en la ventana principal o desde otro programa con el comando:

DistanceTransform(image);

donde *image* es una imagen *grayscale* de 64x64 que se leyó previamente.

Programa Para Entrenar el Perceptrón Multicapa

En seguida se describe una buena opción para un programa que obtenga los *patterns* necesarios de la transformada de distancia alojada en la variable global *dt* y luego definir la red neuronal. Primeramente se define una función cuyo nombre sea el mismo del programa y su argumento sea la transformada de distancia. Esta función se iguala a una variable, que será el nombre con el que se guardará la red neuronal ya entrenada:

function net=NNDT(dt)
global dt

donde *net* es el nombre deseado. Es menester mencionar que esta función es diferente a la del programa de la transformada de distancia al tener la opción de guardarla en una variable con el nombre que se le indique.

Antes de definir la red en Matlab, se debe contar con los patterns de entrenamiento. Como la imagen es de 64x64 píxeles, existen 4096 distancias de donde se deben tomar los *patterns*. Entre más *patterns*, mejor será la generalización que ofrezca la red. Para este trabajo en específico se decidió ir tomando un píxel si y otro no, en cada renglón desde el 2 al 63, y sólo de las columnas 2 a la 63, como si se fueran tomando en cuenta sólo los

cuadros de un solo color en un tablero de ajedrez de 62x62. Esto nos da como resultado 1922 *patterns* con los cuales se logra una aproximación muy eficiente de la transformada de distancia. El código para escoger los *patterns* es el siguiente:

```
inputs=zeros(2,1922);
outputs=zeros(1,1922);
for i=2: 63,
   c=mod(i,2);
   for j=1: 31,
      inputs(1,j+(i-2)*31)=i;
      inputs(2,j+(i-2)*31)=(2+(j-1)*2+c);
      outputs(j+(i-2)*31)=dt(i,2+(j-1)*2+c);
   end
end
```

donde *inputs* y *outputs* son las entradas y salidas respectivamente que definen a los *patterns* y al principio se inicializan a ceros. *Inputs* es un arreglo de 2 renglones para que cada columna contenga el par (x, y) ya que en Matlab las entradas de una red deben estar definidas en una columna, así se presentarán las 1922 columnas que contienen las entradas con su respectiva salida dentro del vector *outputs*. La variable *c* es para descubrir si se trata de un renglón par o impar mediante la función *mod()* (módulo) y así decidir si se empieza en una columna par o impar. Los valores del vector *outputs* se toman directamente de la variable *dt*.

Ahora si ya se puede definir la red y es con el siguiente código:

```
net=newff(minmax(inputs),[9,25,1],{'tansig','tansig','purelin'},'trainlm');
net.trainParam.show = 10;
net.trainParam.epochs = 2000;
net.trainParam.goal = 5e-2;
net.trainParam.mu_max = 1000000000000;
net = init(net) ;
net=train(net,inputs,outputs) ;
```

donde en la primera línea el comando *newff* indica que se trata de una red neuronal multicapa, *minmax(inputs)* saca el rango de valores que las entradas pueden tomar, *[9,25,1]* indica el número de neuronas que contiene cada capa, *tansig* se refiere a la función de activación *hyperbolic tangent sigmoid* y *purelin* a la función *linear*, y *trainlm* se refiere al algoritmo de entrenamiento Levenberg-Marquardt. Aquí al no indicar la ecuación de actualización de pesos deseada y la función de minimización del error que se quiere utilizar, Matlab por *default* empleará la *generalizad delta rule* (*gdm*) y el *Mean Squared Error* (*MSE*) respectivamente. Las siguientes líneas hacen referencia a objetos dentro del programa que trae Matlab en su *Neural Network Toolbox*, y son parámetros de la red: *show* es el número que indica cada cuantas iteraciones (*epochs*) se quiere que Matlab grafique el progreso del MSE, *epochs* es el número de iteraciones máximo que se quiere que el entrenamiento realice, *goal* es el MSE al que se quiere llegar con el entrenamiento, y *mu_max* es la μ máxima permitida del algoritmo Levenberg-Marquardt cuyo valor debe ser muy grande ya que crece muy rápido para hacer veloz el entrenamiento. Es recomendable

siempre inicializar los pesos de una red antes de su entrenamiento y Matlab lo hace por nosotros con la función *isk()* cuyo argumento debe ser el nombre de la red. En al última línea se procede al entrenamiento de la red haciendo uso de la función *train()* que debe tener por argumentos el nombre de la red, el nombre del arreglo de entradas y el nombre del arreglo de salidas que conforman los *patterns*.

Programa que agrupa el cálculo de la transformada de distancia y el entrenamiento del MLP

A continuación se enlista el código que nos facilita el uso de los programas desarrollados anteriormente para calcular la transformada de distancia de una imagen, y luego "enseñársela" al perceptrón multicapa:

```
function l=LearnImage
image=input('Enter name of the image file to be learned: ','s');
global dt
tic;
image=imread(['C:\Path\' image '.jpg']);
DistanceTransform(image);
   isk('Training…')
l=NNDT(dt);
toc;
```

que se trata de una función de nombre *LearnImage* sin argumentos de entrada, cuyo

resultado se guardará en la variable *l* que corresponde a la variable con la que se llame a la función en la ventana principal de Matlab. En el programa anterior primero se solicita el nombre (*string)* de la imagen sin extensión que será aprendida; el argumento *'s'* de la función *input()* indica que se pide una entrada de tipo *string* (cadena). Aquí también se declara la variable global *dt* puesto que será usada. En la quinta línea se lee la imagen solicitada y se guarda en la variable *image*. Luego se llama a la función *DistanceTransform()* dándole como argumento la imagen recién leída, y por último se llama a la función *NNDT()* dándole como argumento la transformada de distancia calculada y se guarda el perceptrón multicapa entrenado en la variable solicitada *l*. Los comandos *tic* y *toc* son de gran ayuda ya que calculan el tiempo que se tarda el proceso que existe entre ellos y lo despliega en segundos al finalizar el programa. La función *isk()* sirve para desplegar comentarios dentro de la ejecución de un programa.

Programa Para el Reconocimiento de Imágenes

La imagen desconocida se parecerá más (y por lo tanto será asociada) a la imagen de la base de datos con la que tenga menor distancia, es decir, con la imagen que presente el menor valor en el promedio de las distancias que muestran los píxeles de las transformadas con respecto a los píxeles que conforman el contorno de la imagen desconocida. Ya que se haya leído la imagen a reconocer, se puede proseguir como sigue:

```
image=edge(image, 'canny');
image=~image;
x=0;
```

```
inputs=zeros(2,1000);
   for i=1: 64,
      for j=1: 64,
         if(image(i,j)==0)
            x=x+1;
            inputs(1,x)=i;
            inputs(2,x)=j;
         end
      end
   end
```

donde primero se saca el contorno de la imagen leída *image*, luego se invierten los valores de los píxeles con la operación NOT (~) para que valgan 0 los que pertenecen al contorno, luego se define un arreglo *inputs* de dos renglones donde se guardarán los pares (x, y) que correspondan a los píxeles del contorno y suponiendo que son menos de 1000 se define lugar (columnas) sólo para esa cantidad de coordenadas, y en los *for* anidados se buscan los pares (x, y) de los píxeles que definen el contorno, así como también se va haciendo el conteo de éstos para definir bien el tamaño de *inputs* que será tomado en cuenta. En el arreglo *inputs* el primer renglón corresponde a las coordenadas x y el segundo renglón a las coordenadas y, así, cada columna define un par de coordenadas correspondientes a la ubicación de los píxeles con valor 0 que conforman el contorno. Matlab tiene la forma peculiar de etiquetar los elementos de un arreglo bidimensional en orden de arriba hacia abajo y de izquierda a derecha, es decir el orden lo definen las columnas empezando en la primera columna, después del último elemento de esta columna se sigue con el primer

elemento de la segunda columna, y así sucesivamente. Así, si se quiere tomar en cuenta los primeros 3 pares de coordenadas en *inputs*, es de la forma siguiente:

inputs([1:2:6;2:2:6]);

donde 1:2:6 significa *renglón1-de 2 en 2-hasta 6*, ya que como las columnas sólo constan de dos elementos, si se quiere seguir la numeración de los elementos por renglón se tendrán que ir contando de dos en dos; en el renglón 1 los números de elemento son impares y en el renglón 2 son pares. De igual forma 2:2:6 significa *renglón2-de 2 en 2-hasta 6*. En el ejemplo anterior se cuenta hasta 6 debido a que es el número total de elementos necesarios para conformar los tres pares de coordenadas, 3 x (renglón 1) y 3 y (renglón 2).

Ya que se tiene detectado el contorno a reconocer, entonces se prosigue de la siguiente forma:

```
d1=sim(Object1,inputs([1:2:x*2;2:2:x*2]));
   d2=sim(Object2,inputs([1:2:x*2;2:2:x*2]));
   d3=sim(Object3,inputs([1:2:x*2;2:2:x*2]));
   d4=sim(Object4,inputs([1:2:x*2;2:2:x*2]));
   d5=sim(Object5,inputs([1:2:x*2;2:2:x*2]));
% etc…
   d1=mean(d1);d2=mean(d2);d3=mean(d3);d4=mean(d4);d5=mean(d5); %...
   l=min([d1,d2,d3,d4,d5]); %...
   if (l==d1)
```

```
    disp('image is an Object1')
  elseif (l==d2)
    disp('image is an Object2')
  elseif (l==d3)
    disp('image is an Object3')
  elseif (l==d4)
    disp('image is an Object4')
  else disp('image is an Object5')
%etc…
  end
end
```

donde se simulan las redes de la base de datos con las entradas *inputs* que se acaban de obtener de la imagen desconocida con la función *sim()* y se guardan los resultados en las variables *d1, d2, d3, d4, d5, etc*. La función *sim()* requiere los argumentos que son el nombre de la red a simular, y las entradas con las que se quiere simular. Aquí se supone que se tienen 5 redes entrenadas con la transformada de distancia de 5 imágenes, cada una representando un objeto y cuyos nombres son los que se muestran en el código anterior (*Object1, Object2, etc*). Del vector *inputs* sólo se toman en cuenta el número de elementos en donde se guardan las coordenadas de los píxeles que definen cada contorno, dado por 2*x. En seguida se procede a sacar el promedio de las distancias (salidas) obtenidas para cada red con la función *mean()* que trae Matlab y se guardan en las mismas variables por simplicidad. Luego se calcula el menor valor de los promedios obtenidos, que correspondería a la transformada de distancia de la imagen con la que más tiene parecido la

imagen desconocida por presentar menor diferencia en valor (distancia) entre los píxeles de su contorno y las salidas arrojadas por la red neuronal que aproxima la transformada de distancia. Por último con el condicional *if* se busca el promedio de distancias con menor valor y se asocia la imagen desconocida a este mostrando un anuncio donde se especifica el objeto que se reconoce según la base de datos actual.

El código para un programa que lea imágenes de corrido puede ser el siguiente:

```
tic;
for n=1:70,
   image=imread(['C:\Path\images\image' int2str(n) '.jpg']);
   image=edge(image, 'canny');
   image=~image;
   x=0;
   inputs=zeros(2,1000);
   for i=1: 64,
      for j=1: 64,
         if(image(i,j)==0)
            x=x+1;
            inputs(1,x)=i;
            inputs(2,x)=j;
         end
      end
   end
```

```
   d1=sim(Object1,inputs([1:2:x*2;2:2:x*2]));
   d2=sim(Object2,inputs([1:2:x*2;2:2:x*2]));
   d3=sim(Object3,inputs([1:2:x*2;2:2:x*2]));
   d4=sim(Object4,inputs([1:2:x*2;2:2:x*2]));
   d5=sim(Object5,inputs([1:2:x*2;2:2:x*2]));
  %...
   d1=mean(d1);d2=mean(d2);d3=mean(d3);d4=mean(d4);d5=mean(d5);
   l=min([d1,d2,d3,d4,d5]);
   if (l==d1)
      disp(['image' int2str(n) '.jpg is an Object1'])
   elseif (l==d2)
      disp(['image' int2str(n) '.jpg is an Object2'])
   elseif (l==d3)
      disp(['image' int2str(n) '.jpg is an Object3'])
   elseif (l==d4)
      disp(['image' int2str(n) '.jpg is an Object4'])
   else disp(['image' int2str(n) '.jpg is an Object5'])
   end
end
toc;
disp('Recalling terminated')
```

donde la función *int2str()* convierte su argumento en una cadena y en el código anterior se usó para concatenar el número de imagen que se está leyendo.

Si se quiere reconocer una imagen en específico se puede desarrollar un programa como el que se muestra a continuación y que puede ser más práctico:

```
v=1;
while v>0
   v=input('Enter number of image: ');
   if v<=0
      break
   end
   disp(' ');
   tic;
   image=imread(['C:\matlab\work\thesis\images\image' int2str(v) '.jpg']);
   image=edge(image, 'canny');
   image=~image;
   x=0;
   inputs=zeros(2,1000);
   for i=1: 64,
      for j=1: 64,
         if(image(i,j)==0)
            x=x+1;
            inputs(1,x)=i;
            inputs(2,x)=j;
         end
      end
```

```
    end
    da=sim(LetterA,inputs([1 :2 :x*2 ;2 :2 :x*2])) ;
    de=sim(LetterE,inputs([1 :2 :x*2 ;2 :2 :x*2])) ;
    di=sim(LetterI,inputs([1 :2 :x*2 ;2 :2 :x*2])) ;
    do=sim(LetterO,inputs([1:2:x*2;2:2:x*2]));
    du=sim(LetterU,inputs([1 :2 :x*2 ;2 :2 :x*2])) ;
    da=mean(da) ;de=mean(de) ;di=mean(di) ;do=mean(do) ;du=mean(du) ;
    l=min([da,de,di,do,du]) ;
    if (l==da)
       disp(['image' int2str(v) '.jpg is an A'])
    elseif (l==de)
       disp(['image' int2str(v) '.jpg is an E'])
    elseif (l==di)
       disp(['image' int2str(v) '.jpg is an I'])
    elseif (l==do)
       disp(['image' int2str(v) '.jpg is an O'])
    else disp(['image' int2str(v) '.jpg is an U'])
    end
    toc;
end
disp('Recalling terminated')
```

que básicamente es igual al anterior pero aquí se reconoce el número de imagen proporcionado, finalizando su ejecución cuando se pida el reconocimiento de una imagen

que no existe o con número menor a 0. En este programa se obtienen tiempos de reconocimiento más específicos de cada imagen.