**CHAPTER** **12:** **OCR** **CON** **EL Self-ORGANIZING** **MAP**

**­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­­\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

* ¿Qué es OCR?
* Recortar una imagen
* Disminución de una imagen
* Formación de la red neuronal para reconocer personajes
* Recordar personajes
* Una aplicación OCR de "grado comercial"

En el capítulo anterior, aprendió a construir un mapa autoorganizado (SOM). Usted aprendió que un SOM se puede utilizar para clasificar muestras en varios grupos. En este capítulo, examinaremos una solicitud som específica; aplicaremos un SOM al reconocimiento óptico de caracteres (OCR).

Los programas OCR son capaces de leer texto impreso. Esto puede ser texto escaneado de un documento o texto escrito a mano dibujado en un dispositivo portátil, como un personal - asistente digital (PDA). Los programas OCR se utilizan ampliamente en muchas industrias. Uno de los mayores usuarios de sistemas OCR es el Servicio Postal de los Estados Unidos.

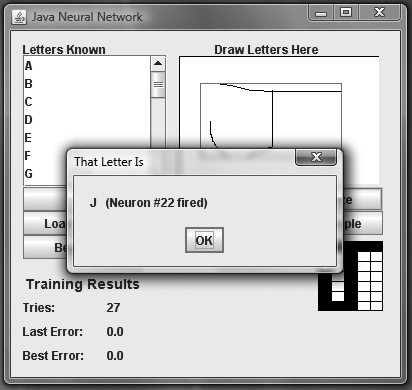
En las décadas de 1970 y 1980, el Servicio Postal de los Estados Unidos tenía muchas máquinas de clasificación de cartas (LSM). Estas máquinas eran tripuladas por empleados humanos, que llevaron los códigos postales de 60 letras por minuto. Los clasificadores de cartas humanos han sido reemplazados por máquinas informatizadas de clasificación de cartas. Estas nuevas máquinas de clasificación de cartas se basan en la tecnología OCR; escanean las letras entrantes, leen sus códigos postales y las dirigen a su destinatario correcto.

En este capítulo, se presentará un programa para demostrar cómo se puede entrenar un mapa autoorganizado para reconocer la escritura humana. No crearemos un programa que pueda escanear páginas de texto; más bien, este programa leerá caracteres individuales a medida que sean dibujados por el usuario. Esta función es similar a las técnicas de reconocimiento de escritura a mano empleoados por muchos PDA.

# La aplicación OCR

Cuando se inicia la aplicación OCR, muestra una interfaz GUI simple. A través de esta interfaz, el usuario puede entrenar y utilizar la red neuronal. La interfaz GUI se muestra en el cuadro 12.1.

### Figura 12.1: La aplicación OCR.



El programa no está inmediatamente listo para reconocer letras al iniciarse. Primero debe ser entrenado usando letras que realmente han sido dibujadas. Los archivos de entrenamiento se almacenan en el mismo directorio que la aplicación OCR. El nombre de la muestra de formación es "sample.dat".

Si descargó el archivo "sample.dat" de Heaton Research, verá que contiene muestras de escritura a mano que he producido. Si utiliza este archivo para entrenar el programa y luego intenta reconocer su propia escritura a mano, es posible que no experimente los resultados que lograría con un archivo de formación basado en su propia escritura a mano. La creación de una muestra basada en su propia escritura se cubrirá en la siguiente sección. Por ahora, nos centraremos en cómo el programa reconoce la escritura a mano utilizando el archivo de ejemplo proporcionado.

Debería comenzar haciendo clic en el botón "Cargar" de la aplicación OCR. A continuación, el programa intentará cargar el archivo de entrenamiento. Se debe mostrar un pequeño cuadro de mensaje que indique que el archivo se cargó correctamente. Una vez cargado el archivo, el programa mostrará todas las letras para las que se entrenará. El archivo de formación proporcionado sólo contiene entradas para las 26 letras mayúsculas del alfabeto latino.

Ahora que las letras han sido cargadas, la red neuronal debe ser entrenada. Al hacer clic en el botón "Entrenar", la aplicación comenzará el proceso de entrenamiento. El proceso de entrenamiento puede tomar en cualquier lugar de unos segundos a varios minutos, dependiendo de la velocidad de su ordenador. Una vez completado el entrenamiento, se mostrará un pequeño cuadro de mensaje.

**Uso del** **programa** **ejemplo** **para** **reconocer** **letras**

Ahora que se ha cargado el conjunto de entrenamiento y se ha entrenado la red neuronal, está listo para reconocer caracteres. La interfaz de usuario hace que este proceso sea muy fácil. Simplemente dibuje el carácter que desea que el programa reconozca en la gran región rectangular que contiene la instrucción "Dibujar letras aquí". Una vez que haya dibujado una letra, puede seleccionar varias opciones diferentes.

Las letras que dibujas se reducen antes de que se reconozcan, lo que significa que la imagen se asigna a una pequeña cuadrícula de cinco píxeles de ancho y siete píxeles de alto. La ventaja de reducir a tal tamaño es doble. En primer lugar, la imagen de menor resolución requiere menos neuronas de entrada para el procesamiento que una imagen de tamaño completo. En segundo lugar, al reducir todo al mismo tamaño, se neutraliza el tamaño de un carácter; no importa si dibujas un carácter grande o un carácter pequeño. Si hace clic en el botón "Downsample", puede ver la versión de su letra. Al hacer clic en el botón "Borrar", se borrarán las regiones de dibujo y desampled.

Observará que se dibuja un cuadro alrededor de su letra cuando se muestra la versión de muestreo descendente. Esta es una caja de recortes. El propósito del cuadro de recorte es eliminar cualquier espacio en blanco no esencial en la imagen. Esto también tiene el efecto deseable de eliminar la necesidad de que el programa considere la posición de una letra. Se puede dibujar una letra en el centro de la región de dibujo, cerca de la parte superior o en alguna otra ubicación, y el programa seguirá reconociéndolo.

Cuando esté listo para reconocer una letra, debe hacer clic en "Reconocer" pero en la tonelada. Esto hará que la aplicación rebamplee la letra y, a continuación, intente reconocerla mediante el mapa autoorganizándose. El proceso exacto para reducir el muestreo de una imagen se discutirá en la siguiente sección. A continuación, el patrón se presenta al mapa autoorganizado y se selecciona la neurona ganadora.

Si recuerdas del capítulo 11, un mapa autoorganizado tiene varias neuronas de salida. Se selecciona una neurona de salida como el ganador para cada patrón de entrada. El mapa autoorganizado utilizado por este programa de muestra tiene 26 neuronas de salida para que coincidan con las 26 letras en el conjunto de muestras. El programa responderá a la carta en la que entras diciéndote a ambos qué neurona disparó, y qué letra cree que has dibujado. Al igualar mi propia escritura a mano, he encontrado que este programa generalmente alcanza una tasa de éxito de aproximadamente de 80-90%. Si usted está teniendo problemas para conseguir que el programa reconozca sus letras, asegúrese de que está escribiendo letras capital claras. También puede intentar entrenar la red neuronal para reconocer su propia escritura a mano, según lo cubierto en la siguiente sección.

## Capacitación del Programa de Muestra para Reconocer Letras

Usted puede encontrar que el programa no reconoce su escritura a mano, así como usted piensa que debería. Esto puede deberse a que el programa fue entrenado usando mi escritura, que puede no ser representativo de la escritura a mano de toda la población. (Mis maestros de primaria seguramente argumentarían que este es realmente el caso.) En esta sección, le explicaré cómo puede entrenar la red usando su propia escritura a mano.

Hay dos enfoques entre los que puedes elegir: puedes empezar desde un conjunto de entrenamiento en blanco e introducir las 26 letras tú mismo, o puedes comenzar con mi conjunto de entrenamiento y reemplazar letras individuales. Este último es un buen enfoque si la red está reconociendo la mayoría de sus personajes, y fallando en sólo un pequeño conjunto. En este caso, usted puede simplemente volver a entrenar la red neuronal para las letras que el programa está luchando para reconocer.

Para eliminar una letra que el conjunto de entrenamiento ya ha enumerado, debe seleccionar esa letra y pulsar el botón "Eliminar" en la aplicación OCR. Tenga en cuenta que este es el botón "Eliminar" de la GUI y no el botón eliminar en el teclado de su computadora.

Para añadir una nueva letra al conjunto de entrenamiento, debe dibujar la letra en el área de entrada del dibujo. Una vez dibujada la carta, haga clic en el botón "Agregar". Se le pedirá la carta real que acaba de dibujar. El carácter que escriba en respuesta a este mensaje se le mostrará cuando la aplicación OCR reconozca la letra.

Una vez completado el conjunto de entrenamiento, debe guardarlo. Esto se logra haciendo clic en el botón "Guardar" de la aplicación. Esto guardará el conjunto de entrenamiento en el archivo "ejemplo. dat". Si ya tiene un archivo denominado "sample.dat", se sobrescribirá; por lo tanto, es importante hacer una copia de su archivo de entrenamiento anterior si desea conservarlo. Si sale de la aplicación OCR sin guardar sus datos de entrenamiento, se perderá. Al iniciar la aplicación OCR de nuevo, puede hacer clic en "Cargar" para recuperar los datos que almacenó en el archivo "sample.dat".

En las dos secciones anteriores aprendió a usar la aplicación OCR. Como ha visto, el programa es experto en reconocer los caracteres que ha introducido y demuestra un buen uso del mapa autoorganizándose.

# Implementación del Programa OCR

Ahora veremos cómo se implementó el programa OCR. Hay varias clases que componen la aplicación OCR. El propósito de cada clase de la aplicación se resume en el Cuadro 12.1.

### Tabla 12.1: Clases para la aplicación OCR

|  |  |
| --- | --- |
| **clase** | **propósito** |
| Entry | El área de dibujo a través de la cual el usuario introduce letras. |
| OCR | El marco principal; esta clase inicia la aplicación OCR. |
| Sample | Se utiliza para mostrar una imagen de muestreo descendente. |
| SampleData | Se utiliza para sostener una imagen des amplificado. |

Ahora examinaremos cada sección del programa. Comenzaremos examinando cómo un usuario dibuja una imagen.

**Imágenes** de dibujo

Aunque no está directamente relacionado con las redes neuronales, el proceso por el cual un usuario es capaz de dibujar caracteres es una parte importante de la aplicación OCR. Examinaremos el proceso en esta sección. El código de este proceso está contenido en el archivo sample.java y se puede ver en el listado 12.1.

### Lista 12.1: Imágenes de dibujo (muestra.java)

paquete com.heatonresearch.book.introneuralnet.ch12.ocr;

importar java.awt.Color; importar java.awt.Graphics;

importar javax.swing. JPanel;

/\*\*

* Capítulo 12: OCR y el mapa de autoorganización

\*

* Ejemplo: elemento GUI que muestra los datos muestreados.

\*

* @author Jeff Heaton
* @version 2,1

\*/

muestra de clase pública extiende JPanel {

/\*\*

* + Id. de serie para esta clase.

\*/

serie larga final estática privadaVersionUID = 2250441617163548592L;

/\*\*

\* Los datos de la imagen.

\*/

Datos SampleData;

/\*\*

* El constructor.

\*

* @param ancho
* El ancho de la imagen desamplificado
* @param altura
* La altura de la imagen amplificado

\*/

Muestra (ancho final int, altura int final) {

this.data = new SampleData(' ', width, height);

}

/\*\*

* El objeto de datos de imagen.

\*

* @return el objeto de datos de imagen.

\*/

SampleData getData(){ devuelve this.data;

}

/\*\*

* @param g
* Muestre la imagen atenuada.

\*/ @Override

pintura vacía pública ( gráficos finales g) { if (this.data == null) {

retorno;

}

int x, y;

final int vcell = getHeight() / this.data.getHeight(); final int hcell = getWidth() / this.data.getWidth();

g.setColor(Color.white);

g.fillRect(0, 0, getWidth(), getHeight());

g.setColor(Color.black);

para (y = 0; y < esto. data.getHeight(); y++) { g.drawLine(0, y \* vcell, getWidth(), y \* vcell);

}

para (x = 0; x < esto. data.getWidth(); x++) { g.drawLine(x \* hcell,0, x \* hcell, getHeight());

}

para (y = 0; Y < esto. data.getHeight(); y++) {

para (x = 0; x < esto. data.getWidth(); x++) { if (this.data.getData(x, y)) {

g.fillRect(x \* hcell, y

\* vcell, hcell, vcell);

}

}

}

g.setColor(Color.black);

g.drawRect(0, 0, getWidth() - 1, getHeight() - 1);

}

/\*\*

* Asigne un nuevo objeto de datos de imagen.

\*

* datos @param
* El objeto de datos de imagen.

\*/

void setData(datos sampledata finales) { this.data = data;

}

}

La clase **Sample** define una serie de propiedades. Las variables se describen en

Tabla 12.2.

### Tabla 12.2: Variables para la aplicación OCR

|  |  |
| --- | --- |
| **variable** | **propósito** |
| downSampleBottom | La parte inferior de la región de cultivo; utilizado durante el downsampling. |
| downSampleLeft | El lado izquierdo de la región de cultivo; utilizado durante el downsampling. |
| downSampleRight | El lado derecho de la región de cultivo; utilizado durante el downsampling. |
| downSampleTop | El lado superior de la región de cultivo; utilizado durante el downsampling. |
| entryGraphics | Objeto gráfico que permite al usuario dibujar en la imagen que corresponde al área de dibujo. |
| entryImage | La imagen que contiene el carácter que el usuario está dibujando. |
| lastX | La última coordenada x en la que el usuario estaba dibujando. |
| lastY | La última coordenada y en la que el usuario estaba dibujando. |
| pixelMap | Mapa numérico de píxeles que en realidad se reducirá. Esto se toma directamente de la entryImage. |
| ratioX | La relación de downsample para la dimensión x. |
| ratioY | La relación de downsample para la dimensión y. |
| sample | El objeto que contendrá la imagen atenuada. |

La mayor parte del dibujo real se controla mediante processMouseMotionEvent. Se dibuja una línea cuando un usuario arrastra el ratón desde la última posición del ratón hasta la posición actual del ratón. El ratón se mueve más rápido de lo que el programa puede aceptar nuevos valores; por lo tanto, al dibujar lalíneae, cubrimos píxeles perdidos lo mejor que podemos. La línea se dibuja en la imagen fuera de la pantalla y, a continuación, se actualiza a la pantalla del usuario. Esto se logra con las siguientes líneas de código.

entryGraphics.setColor(Color.black); entryGraphics.drawLine(lastX,lastY,e.getX(),e.getY()); getGraphics(). drawImage(entryImage,0,0,this);

lastX = e.getX(); lastY = e.getY();

Este método se llama repetidamente a medida que se ejecuta el programa y lo que el usuario está dibujando se guarda en la imagen fuera de pantalla. En la siguiente sección, aprenderá a reducir la tamaño de una imagen. A continuación, verá que se accede a la imagen fuera de pantalla de esta sección como una matriz de enteros, lo que permite que los datos de imagen sean directamente manipu- lated.

# Reducir el muestreo de la imagen

Cada vez que se dibuja una carta para el entrenamiento o el reconocimiento, debe ser downsampled. En esta sección, examinaremos el proceso mediante el cual se logra la remuestreo. Sin embargo, antes de discutir el proceso de desampling, debemos discutir cómo se almacenan las imágenes remuestradas.

## Almacenamiento de imágenes desamplificados

Las imágenes remuestradas se almacenan en la clase **SampleData.** Los **datos de ejemplo**

clase se muestra en el Listado 12.2.

### Listado 12.2: Datos de imagen atenuados (SampleData.java)

paquete com.heatonresearch.book.introneuralnet.ch12.ocr;

/\*\*

* Capítulo 12: OCR y el mapa de autoorganización

\*

* SampleData: Contiene datos muestreados que se usarán para entrenar
* la red neuronal.

\*

* @author Jeff Heaton
* @version 2,1

\*/

clase pública SampleData implementa Comparable<SampleData>, Clonable {

/\*\*

* + Los datos desamplificados como una cuadrícula de booleanos.

\*/

cuadrícula booleana protegida[][];

/\*\*

* + La carta.

\*/

carta char protegida;

/\*\*

* + El constructor

\*

* + @param carta
  + ¿Qué letra es esta?
  + @param ancho
  + La anchura
  + @param altura
  + La altura

\*/

sampledatapúblico(letra char final, ancho final int, altura int final) {

this.grid = nuevo booleano[width][height]; this.letter = letra;

}

/\*\*

* + Despeje la imagen desalada

\*/

vacío público claro() {

para (int x = 0; x < this.grid.length; x++) {

para (int y = 0; y < this.grid[0].length; y++) { this.grid[x][y] = false;

}

}

}

/\*\*

* + Cree una copia de este ejemplo

\*

* + @return Una copia de este ejemplo

\*/ @Override

clonación de objetos públicos()

{

final SampleData obj = new SampleData( este.letter, getWidth(),

getHeight());

para (int y = 0; y < getHeight(); y++) {

para (int x = 0; x < getWidth(); x++) { obj.setData(x, y, getData(x, y));

}

}

retorno obj;

}

/\*\*

* + Compare este ejemplo con otro, utilizado para ordenar.

\*

* + @param o
  + El objeto contra el que se compara.
  + @return Igual que String.compareTo

\*/

public int compareTo(final SampleData o) { final SampleData obj = o;

if (this.getLetter() > obj.getLetter()) { devolver 1;

} else {

devolución -1;

}

}

/\*\*

* + Obtenga un píxel del ejemplo.

\*

* + @param x
  + La coordenada x
  + @param y
  + La coordenada y
  + @return El píxel solicitado

\*/

valor booleano público getData(final int x, final int y) { devolver this.grid[x][y];

}

/\*\*

* + Obtenga la altura de la imagen muestreada hacia abajo.

\*

* + @return La altura de la imagen atenuada.

\*/

público int getHeight() {

devolver this.grid[0].length;

}

/\*\*

* + Obtenga la letra que representa este ejemplo.

\*

* + @return la letra que representa este ejemplo.

\*/

public char getLetter() { devolver this.letter;

}

/\*\*

* + Obtenga el ancho de la imagen atenuada.

\*

* + @return El ancho de la imagen desamplada

\*/

público int getWidth() {

devolver this.grid.length;

}

/\*\*

* + Establezca un píxel de datos de ejemplo.

\*

* + @param x
  + La coordenada x
  + @param y
  + La coordenada y
  + @param v
  + El valor a establecer

\*/

public void setData(final int x, final int y, final boolean v)

{

this.grid[x][y] = v;

}

/\*\*

* + Establezca la letra que representa este ejemplo.

\*

* + @param carta
  + La letra que representa este ejemplo.

\*/

public void setLetter(carta char final) { this.letter = letter;

}

/\*\*

* + Convierta este ejemplo en una cadena.

\*

* + @return Sólo devuelve la letra de que esta muestra es
  + asignado a.

\*/ @Override

public String toString() { devolver "" + this.letter;

}

}

Como puede ver, esta clase representa una cuadrícula de 5 X 7. Todas las imágenes amplificados son

almacenado en esta clase. La clase **SampleData** incluye métodos para establecer y ac- cesar los datos asociados a la cuadrícula de muestreo descendente. La clase **SampleData** también contiene un método denominado clone que se usa para crear duplicados exactos de una imagen.

## Negando el tamaño y la posición

Como se mencionó anteriormente, todas las imágenes se reducen antes de ser utilizadas. Este facili- tates el procesamiento de imágenes por la red neuronal, ya que el tamaño y la posición no tienen que ser considerados. Esto es particularmente important, ya que el área de dibujo es lo suficientemente grande como para permitir a un usuario dibujar letras de diferentes tamaños. La disminución del muestreo da como resultado im-ages de tamaño consistente. En esta sección, explicaré cómo se hace esto.

Cuando dibujas una imagen, lo primero que hace el programa es dibujar un cuadro alrededor de los límites de tu letra. Esto permite al programa eliminar todo el espacio en blanco. El proceso se realiza dentro del método **downsample** de la clase **Entry.** A medida que dibuja un carácter, el carácter también se dibuja en la variable de instancia **entryImage** del objeto **Entry.** Para recortar esta imagen, y eventualmente reducirla, debemos agarrar su patrón de bits. Esto se hace usando la clase **PixelGrabber,** como se muestra aquí:

final int w = this.entryImage.getWidth(esto); final int h = this.entryImage.getHeight(esto);

último PixelGrabber grabber = nuevo PixelGrabber(entryImage,

0,0,w,h,true);

grabber.grabPixels();

pixelMap = (int[])grabber.getPixels();

Después de que el programa procese este código, la variable **pixelMap,** que es un ar- ray de tipos de datos **int,** contiene el patrón de bits de la imagen. El siguiente paso es recortar la imagen y eliminar cualquier espacio en blanco. El recorte se logra arrastrando cuatro lines imaginarios, uno desde la parte superior, otro desde la izquierda, otro desde la parte inferior y otro desde el lado derecho de la imagen. Estas líneas se detienen tan pronto como se encuentran con un píxel que contiene parte de la imagen. A continuación, las cuatro líneas se ajustan a los bordes exteriores de la imagen. Los métodos **hLineClear** y **vLineClear** aceptan un parámetro que indica la línea que se va a escanear y devuelven **true** si esa línea está clara. El programa funciona llamando a **hLineClear** and **vLineClear** hasta que cruzan los bordes exteriores de la imagen. El método de línea horizontal (**hLineClear**) se muestra aquí:

booleano protegido hLineClear(int y)

{

final int w = this.entryImage.getWidth(esto); for ( int i=0;i<w;i++ ) {

si ( este.pixelMap[(y\*w)+i] !=-1 ) devuelve false;

}

volver verdadero;

}

Como puede ver, el método de línea horizontal acepta una coordenada **y** que especifica la línea horizontal que se va a comprobar. A continuación, el programa recorre en bucle cada coordenada **x** de esta fila, comprobando si hay valores de píxel. El valor de -1 indica blanco y se omite. El **método findBounds** utiliza **hLineClear** y **vLineClear** para calcular las cuatro aristas. El beginning de este método se muestra aquí:

búsqueda de vacío protegidoBounds(int w,int h)

{

línea superior

for ( int y =0;y<h;y++ ) { if ( ! hLineClear(y) ) { this.downSampleTop=y;

descanso;

}

}

resultados finales

for ( int y=h-1;y>=0;y-- ) {

if ( ! hLineClear(y) ) { this.downSampleBottom=y; descanso;

}

}

Para calcular la línea superior del rectángulo de recorte, el programa comienza en 0 y se ajusta a la parte inferior de la imagen. La primera línea no clara encontrada se establece como la parte superior del rectángulo de recorte. El mismo proceso, se lleva a cabo al revés para disuadir- minar la parte inferior de la imagen. Los procesos para determinar los límites izquierdo y derecho se llevan a cabo de la misma manera.

## Realización del Downsampling

Una vez que el recorte ha tenido lugar, la imagen debe ser downsampled. Esto implica reducir la imagen a una resolución de 5 X 7. Para entender cómo reducir una imagen a 5 X 7, comience por pensar en una cuadrícula imaginaria que se dibuja encima de la imagen de alta resolución. La cuadrícula divide la imagen en regiones, cinco a través y siete hacia abajo. Si se rellena cualquier píxel de una región, también se rellena el píxel correspondiente de la im-age de 5 X 7. La mayor parte del trabajo realizado por este proceso se realiza dentro del método **downSampleRegion,** como se muestra aquí:

booleano protegido downSampleRegion(final int x, final int y) { final int w = this.entryImage.getWidth(this);

final int startX = (int) (this.downSampleLeft + (x

\* this.ratioX));

final int startY = (int) (this.downSampleTop + (y \* this.ratioY));

final int endX = (int) (startX + this.ratioX); final int endY = (int) (startY + this.ratioY);

para (int yy = startY; yy <= endY; yy++) { para (int xx = startX; xx <= endX; xx++) {

final int loc = xx + (yy \* w);

if (this.pixelMap[loc] != -1) { return true;

}

}

}

devolver falso;

}

El método **downSampleRegion** acepta el número de la región que se va a calcular. A continuación, se calculan las coordenadas **x** e **y** iniciales y finales. El método **downSampleLeft** se utiliza para calcular la primera coordenada **x** para el re-gion especificado. Este es el lado izquierdo del rectángulo de recorte. A continuación, **x** se multiplica por **ratioX**, que es la relación utilizada para indicar el número de píxeles que componen cada región. Nos permite determinar dónde colocar **startX**. La posición de inicio **y, startY,** se calcula de la misma manera. A continuación, el programa recorre cada **x** e **y** de la región especificada. Si se rellena incluso un píxel de la región, el método devuelve **true**. Se llama al método **downSampleRegion** para cada región de la imagen. El resultado final es una copia reducida de la imagen, almacenada en la clase **SampleData.** La clase es una clase contenedora que contiene una matriz de 5 X 7 de valores booleanos. Es este struc- ture que forma la entrada para el entrenamiento y el reconocimiento de personajes.

# Uso del mapa autoorganizado

El patrón de caracteres desamplificado que dibuja el usuario ahora se alimenta a las neuronas de entrada del mapa autoorganizado. Hay una neurona de entrada para cada píxel de la imagen desamplificado. Debido a que la imagen atenuada es una cuadrícula de 5 X 7, hay 35 neuronas de entrada.

La red neuronal comunica qué letra cree que el usuario dibujó a través de las neuronas de salida. El número de neuronas de salida siempre coincide con el número de muestras de letras únicas proporcionadas. Desde que se proporcionaron 26 letras en la muestra, hay 26 neuronas de salida. Si este programa se modificara para admitir múltiples muestras por letra individual, todavía habría 26 neuronas de salida.

Además de la entrada y salida neurons, también hay conexiones entre las neuronas individuales. Estas conexiones no son todas iguales. A cada conexión se le asigna un peso. Los pesos son en última instancia los únicos factores que determinan lo que la red generará para un patrón de entrada determinado. Con el fin de determinar el número total de con-

nections, usted debe multiplicar el número de neuronas de entrada por el number de las neuronas desalida. Una red neuronal con 26 neuronas de salida y 35 neuronas de entrada tendrá un total de 910 pesos de conexión. El proceso de entrenamiento se dedica a encontrar los valores correctos para estos pesos.

El proceso de reconocimiento comienza cuando el usuario dibuja un carácter y, a continuación, hace clic en el botón "Reconocer". En primer lugar, la letra se reduce a una imagen de 5 X 7. Esta imagen desamplada debe copiarse de su matriz de 2 dimensiones a una matriz de dobles que se alimentarán a las neuronas de entrada, como se ve aquí:

este entry.downSample();

doble entrada final[] = nuevo doble[5\*7]; int idx=0;

final SampleData ds = esto. sample.getData(); para ( int y =0;y<ds.getHeight();y++ ) {

for ( int x = 0;x<ds.getWidth();x++ ) { input[idx++] = ds.getData(x,y)?. 5:-.5;

}

}

El código anterior realiza la conversión. Las neuronas requieren entrada de punto flotante; allí-

fore, el programa utiliza el valor de 0.5 para representar un píxel negro y -0.5 para volver a pre- envió un píxel blanco. La matriz de 5 X 7 de 35 valores se alimenta a las neuronas de entrada. Esto se logra pasando la matriz de entrada al método ganador de la red neuronal. Este método identificará cuál de las 35 neuronas ganó, y almacenará esta información en el mejor entero.

final int best = net.winner ( entrada , normfac , sintetizador ) ; mapa char final[] = mapNeurons();

JOptionPane.showMessageDialog(esto", " + map[best] + " (Neuron #"

+ mejor + "despedido)", "Esa carta es", JOptionPane.PLAIN\_MESSAGE);

Conocer la neurona ganadora no es muy útil, porque no te muestra qué letra fue realmente reconocida. Para determinar qué neurona está asociada con cada letra, la red debe ser alimentada cada letra para ver qué neurona gana. Para los ex-amplios, si usted fuera a alimentar la imagen de entrenamiento de "J" en la red neuronal, y la neurona #4 fue devuelta como la ganadora, usted sabría que la neurona #4 es la neurona que fue entrenada para reconocer el patrón de J. Este proceso se realiza llamando al método **mapNeurons** . El método **mapNeurons** devuelve una matriz de caracteres. El índice de cada elemento de matriz corresponde al número de neurona que reconoce el carácter en particular.

## Formación de la Red Neuronal

El aprendizaje es el proceso de selección de una matriz de peso de neuronas que reconocerá correctamente los patrones de entrada. Un mapa autoorganizado aprende evaluando y optimizando constantemente su matriz de peso. Para ello, se debe establecer una matriz de peso inicial. Esto se logra seleccionando números aleatorios. Esta matriz de peso probablemente hará un mal trabajo de reconocer letras, pero proporcionará un punto de partida.

Una vez creada la matriz de peso aleatoria inicial, el entrenamiento puede comenzar. En primer lugar, se evalúa la matriz de peso para determinar su nivel de error actual. El error es disuadido- extraído mediante la evaluación de lo bien que las entradas de entrenamiento (las letras que creó) se asignan a las neuronas de salida. El error se calcula mediante el método **evaluateErrors** de la clase **KohonenNetwork.** Cuando el nivel de error es bajo, digamos por debajo del 10%, el proceso está completo.

El proceso de entrenamiento comienza cuando el usuario hace clic en el botón "Iniciar entrenamiento". Esto comienza el entrenamiento y se calcula el número de neuronas de entrada y salida. En primer lugar, el número de neuronas de entrada se determina a partir del tamaño de la imagen remuestrada. Puesto que la altura es siete y el ancho es cinco para este ejemplo, el número de neurons de entradaes 35. El número de neuronas de salida coincide con el número de caracteres que se ha dado el programa.

Esta parte del programa se puede modificar si desea entrenarlo con más de una muestra por letra. Por ejemplo, si desea utilizar 4 muestras por letter, tendrá que asegurarse de que el recuento de neuronas de salida sigue siendo 26, aunque se proporcionarán 104 muestras de entrada para el entrenamiento, 4 para cada una de las 26 letras.

El entrenamiento se ejecuta en un subproceso en segundo plano como método **de ejecución** Java. La firma para el método de ejecución se muestra aquí:

ejecución pública del vacío( )

En primer lugar, calculamos el número de neuronas de entrada necesarias. Este es el producto de la altura y anchura de la imagen desamplificado.

probar {

entrada int finalNeuron = OCR. DOWNSAMPLE\_HEIGHT

\* OCR. DOWNSAMPLE\_ANCHURA;

salida finalNeuron = this.letterListModel.size();

A continuación, se asigna el conjunto de entrenamiento. Se trata de una matriz de 2 dimensiones con filas iguales al número de elementos de entrenamiento, que en este ejemplo son las 26 letters de la apuestaalfa. El número de columnas es igual al número de neuronas de entrada.

doble conjunto final[][] = nuevo doble[this.letterListModel.size()] [inputNeuron];

A continuación, recorremos todas las muestras de cartas.

para (int t = 0; t < this.letterListModel.size(); t++) { int idx = 0;

Se obtienen los datos de muestra de una carta.

final SampleData ds = (SampleData) this.letterListModel

. getElementAt(t);

Los datos se transfieren a la red y el booleano **true** y **false** se transforman en 0,5 y -0,5.

para (int y = 0; y < ds.getHeight(); y++) { para (int x = 0; x < ds.getWidth(); x++) {

set[t][idx++] = ds.getData(x, y) ? . 5 : -.5;

}

}

}

Se han creado los datos de entrenamiento, por lo que ahora se crea el nuevo objeto **SelfOrganizingMap.** Este objeto utilizará la normalización multiplicativa, que se discutió en el capítulo 11.

this.net = nuevo SelfOrganizingMap(inputNeuron, outputNeuron, NormalizationType.MULTIPLICATIVE);

Ahora se crea un objeto **TrainSelfOrganizingMap** para entrenar el mapa auto-orga- nizing recién creado. Este entrenador utiliza el método de entrenamiento sustractivo, que también se discutió en el capítulo 11.

tren TrainSelfOrganizingMap final = nuevo TrainSelfOrganizingMap( this.net, set,LearningMethod.SUBTRACTIVE,0.5);

Se realiza un seguimiento del número de **intentos** de entrenamiento.

int tries = 1;

El número de intentos y la información de error se actualizan en la ventana para cada iteración de entrenamiento.

hacer {

train.iteration();

actualización (tries++, train.getTotalError(), train.getBestEr- ror());

} mientras (( train.getTotalError() > MAX\_ERROR) && ! this.halt);

this.halt = true;

update(tries, train.getTotalError(), train.getBestError());

} catch (excepción final e) {

Cuando el entrenamiento se equivocao ha alcanzado un nivel aceptable, se muestran los num-bers de entrenamiento final.

e.printStackTrace();

JOptionPane.showMessageDialog(esto, "Error: " + e, "Entrenamiento", JOptionPane.ERROR\_MESSAGE);

}

}

La red neuronal ya está lista para usarse.

# Más allá de este ejemplo

El programa presentado aquí sólo es capaz de reconocer letras individuales, una a la vez. Además, los datos de muestra proporcionados sólo incluyen soporte para las letras mayúsculas de la apuesta alfalatina. No hay nada en este programa que le impida utilizar caracteres en mayúsculas y minúsculas, así como dígitos. Si entrena el programa para dos conjuntos de 26 letras cada uno y 10 dígitos, el programa requerirá 62 conjuntos de entrenamiento.

Puede tener problemas rápidamente con un escenario de este tipo. El programa tendrá dificultades para diferenciar entre una "o" minúscula, una "O" en mayúsculas y el dígito cero (0). El problema de discernir entre ellos no puede ser manejado por la red neuronal. En su lugar, tendrá que examinar el contexto en el que aparecen las letras y dígitos.

Se agregarán muchas capas de complejidad si el programa se expande para procesar una página completa de escritura a la vez. Incluso si la página es sólo texto, será necesario que el programa determine dónde comienza y termina cada línea. Además, los espacios between letras tendrán que ser localizados para que los personajes individuales se pueden alimentar al mapa autoorganizándose para su procesamiento.

Si la imagen que se está escaneando no es texto puro, entonces el trabajo se vuelve aún más com- plex. Será necesario que el programa escanee alrededor de los bordes del texto y los gráficos. Algunas líneas pueden estar en diferentes fuentes, y por lo tanto son de diferentes tamaños. Todas estas cuestiones tendrán que ser consideradas para extender este programa a una aplicación de grado comercial OCR.

Otra limitación de este programa de ejemplo es que solo se puede definir un dibujo por carácter. Es posible que desee utilizar tres muestras de escritura a mano diferentes para una letra, en lugar de solo una. Las clases de red neuronal subyacentes admitirán fácilmente esta característica. Este cambio se puede implementar agregando algunas clases más a la interfaz de usuario. Para ello, tendrá que modificar el programa para aceptar más datos de entrenamiento que el número de neuronas de salida.

Como se puede ver, hay muchas consideraciones que tendrán que hacerse para expandir esta aplicación en una aplicación de grado comercial. Además, usted no será capaz de utilizar una sola red neuronal. Es probable que several diferentes tipos de obras de red neuronal serán necesarios para llevar a cabo las tareas mencionadas.

# Resumen del capítulo

Este capítulo presentó una aplicación práctica del mapa autoorganizado. Se le presentó el concepto de OCR y los usos de esta tecnología. El ejemplo presentado imita las capacidades OCR de un PDA. Los caracteres se introducen cuando un usuario se basa en un cuadro de altaresolución gh. Desafortunadamente, esta resolución es demasiado alta para ser presentada directamente a la red neuronal. Para resolver este problema, utilizamos las técnicas de recorte y reducción para transformar la imagen en una segunda imagen que tiene una ución de resolución mucho menor.

Una vez introducida la imagen, debe recortarse. El recorte es el proceso por el cual se elimina el espacio en blanco adicional. El programa calcula automáticamente el tamaño de cualquier espacio en blanco alrededor de la imagen. A continuación, se traza un rectángulo alrededor del contorno be interpolar la imagen y el espacio en blanco. El uso de cultivos tiene el efecto de eliminar la dependencia po-sition. No importa dónde se dibuje la carta, ya que el cultivo elimi- nates todas las áreas no esenciales.

Una vez recortada la imagen, debe ser downsampled. El desampling es el proceso por el cual una imagen de alta resolución se transforma en una im-age de menor resolución. Una imagen de alta resolución se reduce al dividirla en una serie de re-gions que son iguales al número de píxeles en la imagen desamplada. A cada píxel de la imagen atenuada se le asigna el color medio de la región correspondiente en la imagen de alta resolución. A continuación, la imagen desamplada resultante se alimentaal proceso de entrada o recogimiento de la red neuronal.

El siguiente capítulo presentará otra aplicación de redes neuronales, bots. Los bots son programas informáticos que pueden acceder a sitios web y realizar tareas automatizadas. Al hacerlo, un bot puede encontrar una amplia serie de datos. Además, es posible que estos datos no estén con un formato uniforme. Una red neuronal es ideal para su uso en la comprensión de estos datos.

# vocabulario

Downsample

Reconocimiento óptico de caracteres (OCR)

# Preguntas para revisión

1. Describir cómo se reduce una imagen.
2. ¿Por qué es necesario reducir el amplificador?
3. ¿Cómo se relacionan las dimensiones de la imagen atenuada con las neuronas de entrada del mapa autoorganizado?
4. ¿Cuáles serían las ventajas y desventajas de usar un mayor número de neuronas de entrada para el mapa autoorganizable utilizado para OCR en este capítulo?
5. La aplicación OCR en este capítulo dibujó cuatro líneas imaginarias sobre el carácter dibujado. Estas líneas formaron un cuadrado que encajaba exactamente alrededor del carácter que dibujó el usuario. ¿Cuál es el propósito de las líneas imaginarias