

Lenguaje de señas MNIST

Víctor Choza Merino y Adrián Turiel Charro

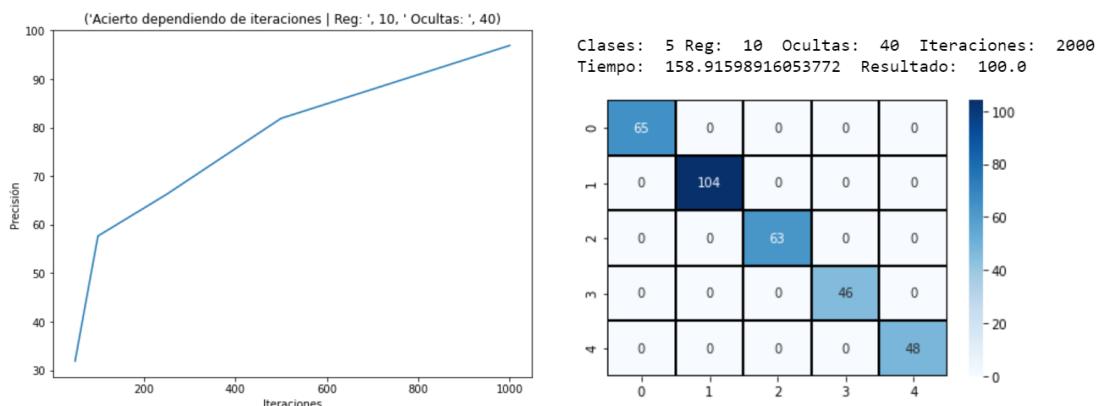
Nuestro proyecto es un sistema de aprendizaje automático cuyo fin es, dada una imagen que representa una letra del abecedario en lenguaje de señas, clasificar esa imagen con la letra que le corresponde, con la mayor precisión posible. El lenguaje de señas americano (ASL) es un lenguaje natural completo que tiene las mismas propiedades lingüísticas que los idiomas hablados.

Para esta clase de problemas se pueden aplicar diferentes técnicas de aprendizaje automático. Algunas de ellas: **redes neuronales**, **regresión logística**, **SVM** (vistas en clase) y **redes neuronales convolucionales o CNN** (fuera del temario).

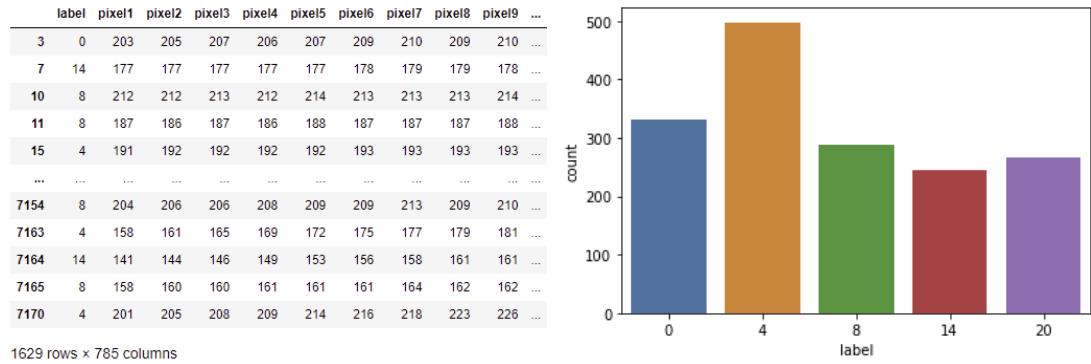
Para comprender mejor el conjunto de datos de nuestro proyecto, en total hay imágenes de 24 letras, y cada imagen está formada por 28x28 píxeles con valores de la escala de grises entre 0 y 255. El dataset que hemos escogido de Kaggle consta de un conjunto de datos de entrenamiento (27.455 imágenes) y de un conjunto de datos de test (7.172 imágenes).

Redes Neuronales (NN)

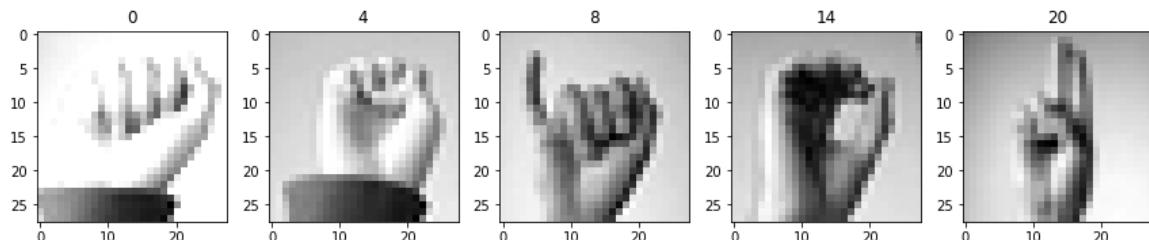
Tras realizar diversas pruebas de red neuronal usando ambos conjuntos de datos y las 24 clases, nos fijamos en que los resultados de precisión obtenidos eran muy bajos (4%) y el tiempo empleado para obtenerlos era bastante elevado. Por ello decidimos usar únicamente el conjunto de datos de test, que al seguir siendo muy voluminoso, lo dividimos en tres partes: entrenamiento, validación y test. De esta manera conseguimos que el tiempo empleado fuera mucho menor, aunque los resultados seguían sin ser realmente buenos. Finalmente realizamos el entrenamiento con las 5 vocales, de modo que pudimos obtener una precisión a la hora de identificar las letras mucho mayor, llegando incluso al 100% en algunas ocasiones.



Como se puede observar en las siguientes imágenes, hemos reducido el dataset a 1629 filas o imágenes. Y también podemos ver la frecuencia de aparición de cada una de las letras que hemos elegido, en este caso las vocales (A→0, E→4, I→8, O→14, U→20).



Dicho esto, la estructura de nuestra red neuronal estará compuesta por una capa de entrada de 784 nodos (28x28 píxeles), una única capa oculta con distintos números de nodos, y una capa de salida de 5 nodos (5 vocales). Para poder visualizar una letra o un conjunto de ellas usamos la función **pinta**, tal que, para las 5 vocales obtenemos:



En la parte de carga de datos, se crea un array con los números correspondientes a las letras que queremos que el sistema aprenda a identificar ("letters"). En este caso, como ya hemos dicho anteriormente, hemos usado las vocales (A→0, E→4, I→8, O→14, U→20), pero se pueden introducir cualquier conjunto de letras.

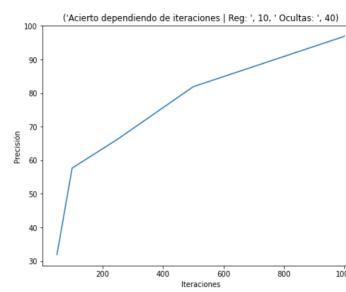
```
# Seleccionamos Las 5 Letras con Las que queremos trabajar, concretamente Las 5 vocales
letters = [0,4,8,14,20]
# Cargamos Los datos
data = get_data(letters)
# El conjunto de datos presenta La siguiente estructura
data
```

Respecto a las funciones, **get_data** y **convert_data** se usan en la fase de carga de datos para obtener únicamente los datos correspondientes a las letras que hemos introducido en **letters**, y para dividir estos datos en entrenamiento, validación y test. La función **weights** genera dos matrices de peso aleatorias con el tamaño adecuado para los datos a tratar. Debido a que estas matrices de pesos son aleatorias, los resultados que hemos obtenido han ido variando desde un 70% de precisión hasta un 100% aproximadamente. Ejecutaremos esta función **weights** justo después de obtener los datos y antes de realizar

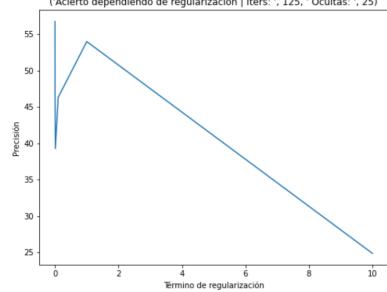
el entrenamiento para que al realizar el entrenamiento, las validaciones y el test siempre se usen las mismas matrices de pesos.

La función más importante sería **funcion**, la cual obtiene tres arrays de términos de regularización, números de iteraciones y nodos de la capa oculta; los datos de entrenamiento y de test/validación; y las matrices de pesos aleatorias que se han generado anteriormente. Esta función se encarga de adaptar las etiquetas de las "y" para llamar a la función **backprop** con **minimize** para cada una de las clases. Seguidamente, con los resultados obtenidos del **backprop**, se llama a la función **precision** para cada uno de los casos para obtener el porcentaje de acierto con el entrenamiento y los datos de validación/test. Como resultado, **funcion** muestra por pantalla los resultados de cada uno de los casos analizados, junto a un conjunto de gráficas dependiendo de los términos de regularización y nodos de la capa oculta, así como del número de iteraciones. Ejemplo:

```
Clases: 5 Reg: 10 Ocultas: 40 Iteraciones: 50
Tiempo: 3.706857919620993 Resultado: 31.901840490797547
Clases: 5 Reg: 10 Ocultas: 40 Iteraciones: 100
Tiempo: 7.304656744003296 Resultado: 57.668711656441715
Clases: 5 Reg: 10 Ocultas: 40 Iteraciones: 250
Tiempo: 18.515945196151733 Resultado: 66.25766871165644
Clases: 5 Reg: 10 Ocultas: 40 Iteraciones: 500
Tiempo: 40.074312686920168 Resultado: 81.90184049079755
Clases: 5 Reg: 10 Ocultas: 40 Iteraciones: 1000
Tiempo: 76.7988102867542 Resultado: 96.932515374231
```



```
Clases: 5 Reg: 0.0001 Ocultas: 25 Iteraciones: 125
Tiempo: 7.285073757171631 Resultado: 56.74846625766872
Clases: 5 Reg: 0.001 Ocultas: 25 Iteraciones: 125
Tiempo: 6.09544201538625 Resultado: 46.5932153771331
Clases: 5 Reg: 0.01 Ocultas: 25 Iteraciones: 125
Tiempo: 7.091835921972656 Resultado: 39.263803680981596
Clases: 5 Reg: 0.1 Ocultas: 25 Iteraciones: 125
Tiempo: 6.8711559772491455 Resultado: 46.31901840490797
Clases: 5 Reg: 1 Ocultas: 25 Iteraciones: 125
Tiempo: 6.973508837249756 Resultado: 53.987730861349694
Clases: 5 Reg: 10 Ocultas: 25 Iteraciones: 125
Tiempo: 6.842595310151967 Resultado: 24.846625766871167
Clases: 5 Reg: 0.0001 Ocultas: 40 Iteraciones: 125
Tiempo: 11.08013388749169 Resultado: 50.920245398775
Clases: 5 Reg: 0.001 Ocultas: 40 Iteraciones: 125
Tiempo: 9.088345850811768 Resultado: 47.23926368036809984
Clases: 5 Reg: 0.01 Ocultas: 40 Iteraciones: 125
Tiempo: 9.264087568359374 Resultado: 54.98797546012271
Clases: 5 Reg: 0.1 Ocultas: 40 Iteraciones: 125
Tiempo: 9.744845390319824 Resultado: 45.70552147239264
Clases: 5 Reg: 1 Ocultas: 40 Iteraciones: 125
Tiempo: 10.47926139831543 Resultado: 46.0122699386503086
```

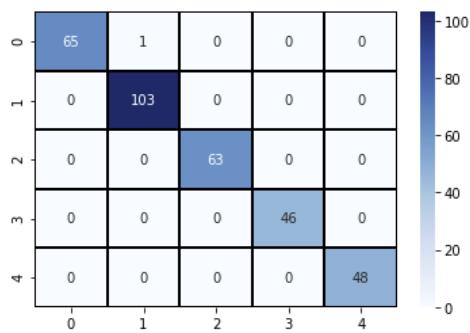


Nuestro objetivo es lograr una mayor precisión, por tanto, para implementar una red neuronal desde cero, tendremos que entrenar el conjunto de datos y analizar el resultado obtenido con un conjunto distinto, el conjunto de validación y así iremos ajustando los diferentes argumentos hasta encontrar la combinación óptima de ellos. Una vez los tengamos, los probaremos en un nuevo conjunto que no hayamos usado anteriormente, el conjunto de test, y así evitar el sobreaprendizaje.

Por ello, en la parte de entrenamiento y validación, manteniendo constante el número de iteraciones, llamamos a **funcion**, donde para cada argumento, como el término de regularización o el número de capas ocultas con el que queremos probar, le pasamos un vector con los distintos valores, y para cada combinación de ellos podremos ver cuánto ha tardado esa ejecución y la precisión resultante.

Después en la parte de Test, habiendo obtenido el término de regularización y número de capas ocultas adecuados, probaremos para distinto número de iteraciones, para ver a partir de qué iteración deja de ser eficaz la red ya sea porque disminuye la precisión resultante o porque deja de aumentar. Y finalmente en la parte de Óptimo, probaremos la red con los argumentos correspondientes para esa matriz de pesos aleatoria generada inicialmente.

Clases: 5 Reg: 0.1 Ocultas: 125 Iteraciones: 2000
Tiempo: 416.8347227573395 Resultado: 99.69325153374233



¡Rozando el éxito absoluto! Nuestra red neuronal ha clasificado perfectamente todas las imágenes (la diagonal principal) a excepción de una. El eje de ordenadas representa el valor real que representa la imagen, y el eje de abscisas la predicción. Ha confundido una E con una A, lo cual es entendible, debido a la gran similitud entre ambas imágenes, un puño cerrado donde solo cambia la posición del pulgar. El error se podría deber a una mala calidad de la imagen o unas tonalidades que dificultan su clasificación.

Support Vector Machines (SVM)

Aparte de la redes neuronales, hemos empleado el algoritmo de **Support vector machines** o **SVM**, un algoritmo de clasificación perteneciente también al aprendizaje supervisado. Este algoritmo es un clasificador discriminatorio definido formalmente por un hiperplano de separación. En otras palabras, dados los datos del entrenamiento etiquetados, el algoritmo genera un hiperplano y clasifica los nuevos ejemplos en varios espacios dimensionales, donde cada clase se encuentra en un lado. Este algoritmo nos permite utilizar las llamadas funciones kernel, que nos permiten resolver el problema trasladando los datos a un espacio donde el hiperplano solución es lineal, y por tanto más sencillo de obtener. Y una vez obtenida la solución se transforma al espacio original.



Para el entrenamiento, probamos para **kernels** diferentes (rbf y poly), varias combinaciones de los distintos argumentos: **C** y **gamma**. Estas combinaciones de parámetros las podemos obtener de forma muy sencilla gracias a la función de **GridSearchCV**, que realiza una

búsqueda exhaustiva de los parámetros especificados con validación cruzada. El parámetro C, para valores menores, aceptará un margen mayor, por tanto, una función de decisión más simple, a costa de menor precisión. Se comporta como un parámetro de regularización. Por otro lado, el parámetro gamma, define la influencia de un solo ejemplo de entrenamiento. A valores bajos influye más.

```

1 start_timef = time.time()
2 param_grid={'C':[0.1,1,10,100], 'gamma':[0.0001,0.001,0.1,1], 'kernel':['poly']}
3 svc3=svm.SVC(probability=False)
4 model3=GridSearchCV(svc3,param_grid,n_jobs=-1)
5 model3.fit(X_train,y_train.ravel())
6 print('Entrenamiento completado')
7 print ('Tiempo empleado: ', np.round((time.time() - start_timef),2), 'segundos')

```

Entrenamiento completado
Tiempo empleado: 1490.91 segundos

```

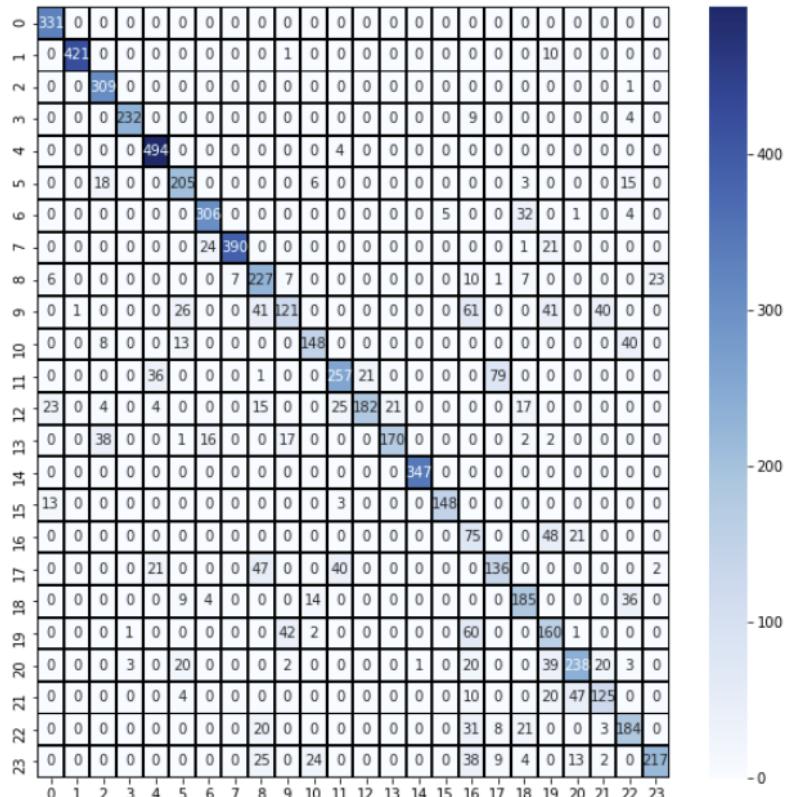
1 y_pred3=model3.predict(X_test)
2
3 print('Precisión: ', accuracy_score(y_pred3,y_test)*100, '%')

```

Precisión: 78.19297267150029 %

Como se puede ver en la foto anterior, finalmente acabamos eligiendo el kernel 'poly', con el que logramos mayor precisión con el dataset de train (100%), consiguiendo un total del 78% para la clasificación de las 24 clases y el dataset completo.

En la matriz de confusión podemos apreciar nuevamente para qué casos estamos clasificando erróneamente esa imagen y con qué clases la confundimos más.



Regresión Logística Multi-Clase

Otro algoritmo que hemos probado es **regresión lógistica** una técnica estadística multivariante que nos permite estimar la relación existente entre una variable dependiente no métrica y un conjunto de variables independientes métricas o no métricas. Para este algoritmo cabe destacar las funciones: *oneVsAll*, *coste*, *gradiente*, *sigmoide* y *funcion*.

La función **oneVsAll** nos permite entrenar varios clasificadores por regresión logística dado un lambda. Tras crear la matriz All_Thetas inicializada todo a ceros, llamaremos a la función **fmin_tnc**, donde para cada fila obtenemos el valor de los parámetros que minimizan la función de coste. Finalmente devuelve el resultado en forma de matriz.

La función principal es **funcion** donde llamamos al resto de funciones probando distintos valores para cada parámetro (el término de regularización), calculando para cada ejecución el tiempo empleado y la precisión obtenida. Finalmente nos devuelve la matriz de pesos de la ejecución que logró mayor precisión. Después observamos su matriz de confusión correspondiente.

Finalmente, en los 3 métodos de aprendizaje que hemos utilizado, hemos creado una función **solve**, a la cual se le pasa una matriz de imágenes y unos parámetros como los modelos y las matrices de peso, y esta te devuelve las letras a las que corresponden las imágenes que le hemos pasado. De esta manera podemos probar la eficacia de los programas con distintas imágenes.

Recientemente, hay muchos avances tecnológicos que han tenido lugar en el campo de la inteligencia artificial. Las redes neuronales convolucionales son una categoría de redes neuronales que han demostrado ser muy eficaces en áreas como la clasificación de imágenes. Su propósito principal es extraer características de la imagen de entrada. Es una herramienta muy potente debido a que es capaz de detectar características muy simples y componerlas en características más complejas hasta detectar lo que busca.

Finalmente, cabe destacar que respecto a los algoritmos probados, hemos obtenido el mejor resultado tanto en tiempo como en precisión con el algoritmo SVM.

Dataset Kaggle: <https://www.kaggle.com/datamunge/sign-language-mnist>

PRÁCTICA FINAL - VÍCTOR CHOZA MERINO - ADRIÁN TURIEL CHARRO

Lenguaje de señas MNIST

Redes Neuronales (NN)

<https://www.kaggle.com/datamunge/sign-language-mnist> (<https://www.kaggle.com/datamunge/sign-language-mnist>)



Librerías

In [9]:

```

1 import time
2 import scipy.io
3 import numpy as np
4 import pandas as pd
5 import seaborn as sns
6 from scipy.special import expit
7 import matplotlib.pyplot as plt
8 from scipy.optimize import minimize
9 from sklearn.metrics import confusion_matrix
10 from sklearn.model_selection import train_test_split

```

Funciones

In [10]:

```

1 def dibujarGraficas(regs, iterss, ocultas, resultados):
2     if (len(iterss)!=1):
3         if (len(regs)!=1 or len(ocultas)!=1):
4             print('Escoga un término de regularización y un número de nodos de la capa')
5         else:
6             string = 'Acierto dependiendo de iteraciones | Reg: ', 
7                     regs[0], ' Ocultas: ', ocultas[0]
8             plt.figure(figsize=(8, 6))
9             plt.xlabel("Iteraciones")
10            plt.ylabel("Precisión")
11            plt.title(string)
12            plt.plot(iterss, resultados)
13        else:
14            if (len(regs)!=1):
15                c = 0
16                for i, num_ocultas in enumerate(ocultas):
17                    res = resultados[c:(c+len(regs))]
18                    string = 'Acierto dependiendo de regularización | Iters: ', 
19                            iterss[0], ' Ocultas: ', num_ocultas
20                    plt.figure(figsize=(8, 6))
21                    plt.xlabel("Término de regularización")
22                    plt.ylabel("Precisión")
23                    plt.title(string)
24                    plt.plot(regs, res)
25                    c = c+len(regs)
26            if (len(ocultas)!=1):
27                for i, reg in enumerate(regs):
28                    indexes = np.arange(i, len(resultados), len(regs))
29                    res = resultados[indexes]
30                    string = 'Acierto dependiendo de la capa oculta | Iters: ', 
31                        iterss[0], ' Reg: ', reg
32                    plt.figure(figsize=(8, 6))
33                    plt.xlabel("Número de nodos de la capa oculta")
34                    plt.ylabel("Precisión")
35                    plt.title(string)
36                    plt.plot(ocultas, res)

```

In [11]:

```

1 def backprop (params_rn, num_entradas, num_ocultas, num_etiquetas, X, y, reg):
2     # backprop devuelve una tupla (coste, gradiente) con el coste y el gradiente de
3     # una red neuronal de tres capas , con num_entradas , num_ocultas nodos en la capa
4     # oculta y num_etiquetas nodos en la capa de salida. Si m es el n mero de ejemplos
5     # de entrenamiento, la dimensi n de 'X' es (m, num_entradas) y la de 'y' es
6     # (m, num_etiquetas)
7
8     Theta1 = np.reshape(params_rn[:num_ocultas * (num_entradas + 1)],
9                         (num_ocultas, (num_entradas + 1)))
10
11    Theta2 = np.reshape(params_rn[num_ocultas * (num_entradas + 1):],
12                         (num_etiquetas, (num_ocultas + 1)))
13
14    J=0
15    m = X.shape[0]
16
17    a1 = np.hstack((np.ones((m,1)), X))
18    a2 = expit(a1 @ Theta1.T)           # sigmoide
19    a2 = np.hstack((np.ones((m,1)), a2))
20    h = expit(a2 @ Theta2.T)           # sigmoide
21
22    for i in range(num_etiquetas):
23        H = h[:,i]
24        J = J + np.log(H).T @ -y[:,i] - (1-np.ravel(y[:,i])) @ np.log(1-H)
25
26    coste = J/m
27    coste_regularizado = coste + reg/(2*m) * (np.sum(Theta1[:,1:]**2)
28                                              + np.sum(Theta2[:,1:]**2))
29
30    Delta1 = np.zeros((Theta1.shape))
31    Delta2 = np.zeros((Theta2.shape))
32
33    for t in range(m):
34        a1t = a1[t, :] # (401,)
35        a2t = a2[t, :] # (26,)
36        ht = h[t, :] # (10,)
37        yt = y[t] # (10,)
38        d3t = ht - yt # (10, )
39        d2t = np.dot(Theta2.T, d3t) * (a2t * (1 - a2t)) # (26, )
40        Delta1 = Delta1 + np.dot(d2t[1:, np.newaxis], a1t[np.newaxis, :])
41        Delta2 = Delta2 + np.dot(d3t[:, np.newaxis], a2t[np.newaxis, :])
42
43    Delta1 = Delta1/m
44    Delta2 = Delta2/m
45
46    Delta1_reg = Delta1 + (reg/m) * np.hstack(
47        (np.zeros((Theta1.shape[0],1)),Theta1[:,1:]))
48    Delta2_reg = Delta2 + (reg/m) * np.hstack(
49        (np.zeros((Theta2.shape[0],1)),Theta2[:,1:]))
50
51    return (coste_regularizado, np.append(Delta1_reg,Delta2_reg))
52
53 def propagacion_hacia_adelante(Theta1, Theta2, X):
54     m = X.shape[0]
55     X = np.hstack((np.ones((m,1)),X))
56
57     a1 = expit(X @ Theta1.T)           # sigmoide
58     a1 = np.hstack((np.ones((m,1)), a1))
59     a2 = expit(a1 @ Theta2.T)           # sigmoide

```

```
60  
61     return a2
```

In [12]:

```
1 def porcentaje_acertado(X, y, H, pintaMatriz):  
2  
3     pos_max = np.argmax(H, axis=1)+1  
4     #for i in range(len(pos_max)):  
5     #    print(pos_max[i])  
6     #    print(y[i])  
7     suma = sum(pos_max[:,np.newaxis]==y)  
8  
9     if(pintaMatriz):  
10         pintarMatrizAciertos(pos_max, y.flatten())  
11  
12     return ((suma/np.shape(H)[0])*100)[0]  
13  
14 def pintarMatrizAciertos(vect1, vect2):  
15     cm = confusion_matrix(vect1,vect2)  
16     sns.heatmap(cm,cmap= "Blues",  
17                 plinecolor = 'black' , linewidth = 1 , annot = True, fmt='')
```

In [17]:

```

1 def get_data (letters_in):
2     data = pd.read_csv("archive/sign_mnist_test.csv")
3
4     query_beg = '(label == '
5     query = ') or (label == '.join(str(e) for e in letters_in)
6     query = query_beg+query+')'
7
8     return data.query(str(query))
9
10
11 def convert_data (data, num_etiquetas):
12     X = data.drop(['label'],axis=1)
13     y = data['label']
14     y = y.values.reshape(len(y),1)
15
16     X = X.reset_index(drop=True)
17
18     X = np.array(X)
19     y = np.array(y)
20
21 # Devolvemos los datos del archivo de datos test divididos en 60% train,
22 # 20% validación y 20% test
23 X_train, X_aux, y_train, y_aux = train_test_split(X, y,
24                                                 test_size=0.4,
25                                                 random_state=40)
26 X_val, X_test, y_val, y_test = train_test_split(X_aux,
27                                                 y_aux,
28                                                 test_size=0.5,
29                                                 random_state=40)
30
31 return X_train, X_val, X_test, y_train, y_val, y_test
32
33
34 def weights (num_entradas, num_ocultas, num_etiquetas):
35     INIT_EPSILON = 0.12
36
37     Rand_Theta1 = np.random.random(
38         (num_ocultas,num_entradas+1))*(2*INIT_EPSILON) - INIT_EPSILON
39     Rand_Theta2 = np.random.random(
40         (num_etiquetas,num_ocultas+1))*(2*INIT_EPSILON) - INIT_EPSILON
41
42 return Rand_Theta1, Rand_Theta2
43
44
45 def get_maxim (regs, ocultas, resultados):
46     maxim = np.where(resultados == max(resultados))[0][0]
47     c=0
48     for i, iters in enumerate(itersss):
49         for j, num_ocultas in enumerate(ocultas):
50             for k, reg in enumerate(regs):
51                 if (c == maxim):
52                     return num_ocultas, reg
53                 c=c+1
54
55
56 def pinta (X_test, y_test, letters_in):
57     fig,axe=plt.subplots(1,len(letters_in))
58     fig.set_size_inches(12, 12)
59     for i, letter in enumerate(letters_in):

```

```

60         axe[i].imshow(X_test[np.where(y_test==letter)[0][0]].reshape(28,28),cmap='gray')
61         axe[i].set_title(letter)
62     plt.tight_layout()
63
64
65 def minimiza (num_entradas, num_ocultas, num_etiquetas,
66                 X_train, Y_train, reg, iters, params_rn):
67     fmin = minimize(fun=backprop, x0=params_rn,
68                     args=(num_entradas, num_ocultas,
69                           num_etiquetas, X_train, Y_train, reg),
70                     method='TNC', jac=True,
71                     options={'maxiter': iters})
72
73     return fmin
74
75
76 def precision (result, num_ocultas, num_entradas, num_etiquetas,
77                 X_test, y_test, pintaMatriz):
78     theta1 = result[:((num_ocultas * (num_entradas + 1))].reshape(
79                     (num_ocultas, num_entradas + 1))
80     theta2 = result[-((num_ocultas + 1) * num_etiquetas):].reshape(
81                     (num_etiquetas, num_ocultas + 1))
82
83     H = propagacion_hacia_adelante(theta1, theta2, X_test)
84     return theta1, theta2, porcentaje_acertado(X_test, y_test, H, pintaMatriz)

```

In [18]:

```

1 def solve(X, theta1, theta2, letters):
2     letras = np.array(['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J',
3                         'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T',
4                         'U', 'V', 'W', 'X', 'Y', 'Z'])
5
6     X = np.array(X)
7     H = propagacion_hacia_adelante(theta1, theta2, X)
8     letras_i = np.argmax(H, axis=1)
9
10    return letras[np.array(letters)[letras_i]]

```

In [19]:

```

1 def funcion(X_train, y_train, X_test, y_test, letters, regs,
2             iterss, ocultas, Rand_Theta1, Rand_Theta2, pintaMatriz):
3     print('Tamaño de Train: ', len(y_train))
4     print('Tamaño de Test: ', len(y_test))
5     y_train_ok = np.zeros(y_train.shape)
6     y_test_ok = np.zeros(y_test.shape)
7     for i in range(len(letters)): # Cambia los labels de las y para entrenar
8         y_train_ok[np.where(y_train == letters[i])] = (i+1)
9         y_test_ok[np.where(y_test == letters[i])] = (i+1)
10
11    num_entradas = np.shape(X_train)[1] # pixeles
12    num_etiquetas = len(letters)
13
14    Y_train = np.zeros((X_train.shape[0],num_etiquetas))      # Convertimos y en vector
15    for i in range(1,num_etiquetas+1):
16        Y_train[:,i-1][:,np.newaxis] = np.where(y_train_ok==i,1,0)
17
18    resultados = np.zeros(len(regs)*len(iterss)*len(ocultas))
19    c = 0
20    start_timef = time.time()
21    for i, iters in enumerate(iterss):
22        for j, num_ocultas in enumerate(ocultas):
23            for k, reg in enumerate(regs):
24                start_time = time.time()
25                print('-----')
26                print ('Clases: ', num_etiquetas, '| Reg: ',reg, '| Ocultas: ',
27                      num_ocultas, '| Iteraciones: ', iters)
28                params_rn = np.append(
29                    Rand_Theta1[:num_ocultas,:].flatten(),Rand_Theta2[:num_etiquetas,:(
30                        num_ocultas+1)].flatten())
31
32                fmin = minimiza(num_entradas, num_ocultas, num_etiquetas,
33                                X_train, Y_train, reg, iters, params_rn)
34
35                theta1, theta2, resultado = precision(fmin.x, num_ocultas,
36                                            num_entradas, num_etiquetas,
37                                            X_test, y_test_ok, pintaMatriz)
38                resultados[c] = resultado
39                print ('Tiempo: ', np.round((time.time() - start_time),2),
40                      'segundos | Precisión: ', resultado, '%')
41                c = c+1
42
43                print('-----')
44                print ('Tiempo total empleado: ',
45                      np.round((time.time() - start_timef),2), 'segundos')
46
47                if (len(resultados) > 1):
48                    return resultados
49                else:
50                    return theta1, theta2, resultados

```

Cargando datos

In [20]:

```

1 # Seleccionamos Las 5 letras con Las que queremos trabajar, concretamente Las 5 vocales
2 letters = [0,4,8,14,20]
3 # Cargamos los datos
4 data = get_data(letters)
5 # El conjunto de datos presenta la siguiente estructura
6 data

```

Out[20]:

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	p
3	0	203	205	207	206	207	209	210	209	210	...	154	
7	14	177	177	177	177	177	178	179	179	178	...	232	
10	8	212	212	213	212	214	213	213	213	214	...	213	
11	8	187	186	187	186	188	187	187	187	188	...	187	
15	4	191	192	192	192	192	193	193	193	193	...	81	
...	
7154	8	204	206	206	208	209	209	213	209	210	...	233	
7163	4	158	161	165	169	172	175	177	179	181	...	78	
7164	14	141	144	146	149	153	156	158	161	161	...	199	
7165	8	158	160	160	161	161	161	164	162	162	...	177	
7170	4	201	205	208	209	214	216	218	223	226	...	112	

1629 rows × 785 columns

In [21]:

```

1 # Inicializamos aleatoriamente la matriz de pesos
2 Rand_Theta1, Rand_Theta2 = weights(np.shape(data)[1]-1, np.shape(data)[1], 24)

```

In [22]:

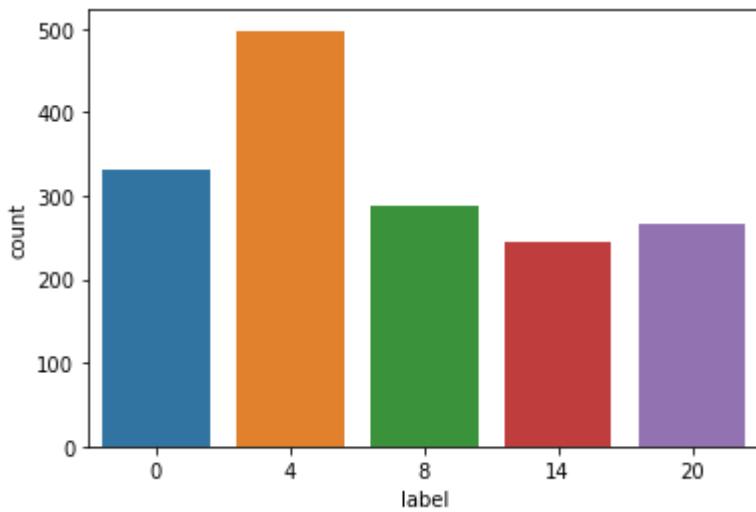
```

1 # Mostramos la frecuencia de aparición de cada letra (en este caso las vocales)
2 sns.countplot(data['label'])

```

Out[22]:

<matplotlib.axes._subplots.AxesSubplot at 0x21b21614310>

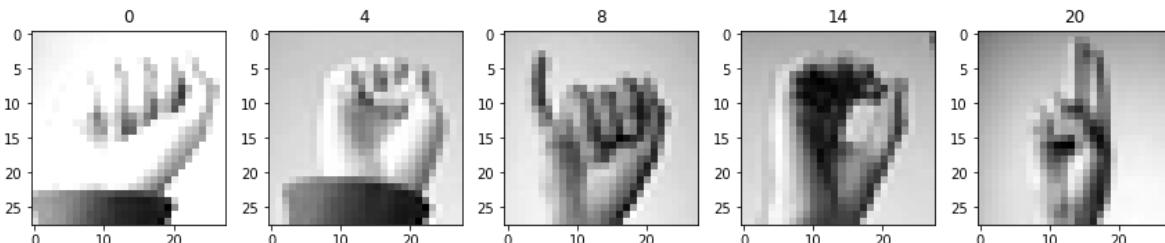


In [16]:

```

1 # Dividimos el conjunto de datos en varios subconjuntos
2 X_train, X_val, X_test, y_train, y_val, y_test = convert_data(data, len(letters))
3
4 # Dibujamos como se verian las vocales
5 pinta(X_train, y_train, letters)

```



Entrenamiento y validación

In [17]:

```

1 regs = [0.0001,0.001,0.01,0.1,1,10]
2 iterss = [125]
3 ocultas = [25,40,50,100,125]
4
5 resultados = funcion(X_train, y_train, X_val, y_val, letters,
6                         regs, iterss, ocultas, Rand_Theta1, Rand_Theta2, 0)
7 dibujarGraficas(regs, iterss, ocultas, resultados)

```

Tamaño de Train: 977

Tamaño de Test: 326

Clases: 5 | Reg: 0.0001 | Ocultas: 25 | Iteraciones: 125

Tiempo: 6.84 segundos | Precisión: 42.63803680981595 %

Clases: 5 | Reg: 0.001 | Ocultas: 25 | Iteraciones: 125

Tiempo: 6.35 segundos | Precisión: 37.11656441717792 %

Clases: 5 | Reg: 0.01 | Ocultas: 25 | Iteraciones: 125

Tiempo: 6.97 segundos | Precisión: 36.19631901840491 %

Clases: 5 | Reg: 0.1 | Ocultas: 25 | Iteraciones: 125

Tiempo: 7.07 segundos | Precisión: 36.809815950920246 %

Clases: 5 | Reg: 1 | Ocultas: 25 | Iteraciones: 125

Tiempo: 7.12 segundos | Precisión: 47.54601226993865 %

Clases: 5 | Reg: 10 | Ocultas: 25 | Iteraciones: 125

Tiempo: 7.11 segundos | Precisión: 43.558282208588956 %

Clases: 5 | Reg: 0.0001 | Ocultas: 40 | Iteraciones: 125

Tiempo: 9.06 segundos | Precisión: 36.19631901840491 %

Clases: 5 | Reg: 0.001 | Ocultas: 40 | Iteraciones: 125

Tiempo: 9.27 segundos | Precisión: 36.19631901840491 %

Clases: 5 | Reg: 0.01 | Ocultas: 40 | Iteraciones: 125

Tiempo: 9.5 segundos | Precisión: 36.19631901840491 %

Clases: 5 | Reg: 0.1 | Ocultas: 40 | Iteraciones: 125

Tiempo: 10.2 segundos | Precisión: 34.96932515337423 %

Clases: 5 | Reg: 1 | Ocultas: 40 | Iteraciones: 125

Tiempo: 8.86 segundos | Precisión: 41.104294478527606 %

Clases: 5 | Reg: 10 | Ocultas: 40 | Iteraciones: 125

Tiempo: 9.21 segundos | Precisión: 43.558282208588956 %

Clases: 5 | Reg: 0.0001 | Ocultas: 50 | Iteraciones: 125

Tiempo: 10.47 segundos | Precisión: 44.171779141104295 %

Clases: 5 | Reg: 0.001 | Ocultas: 50 | Iteraciones: 125

Tiempo: 10.28 segundos | Precisión: 41.717791411042946 %

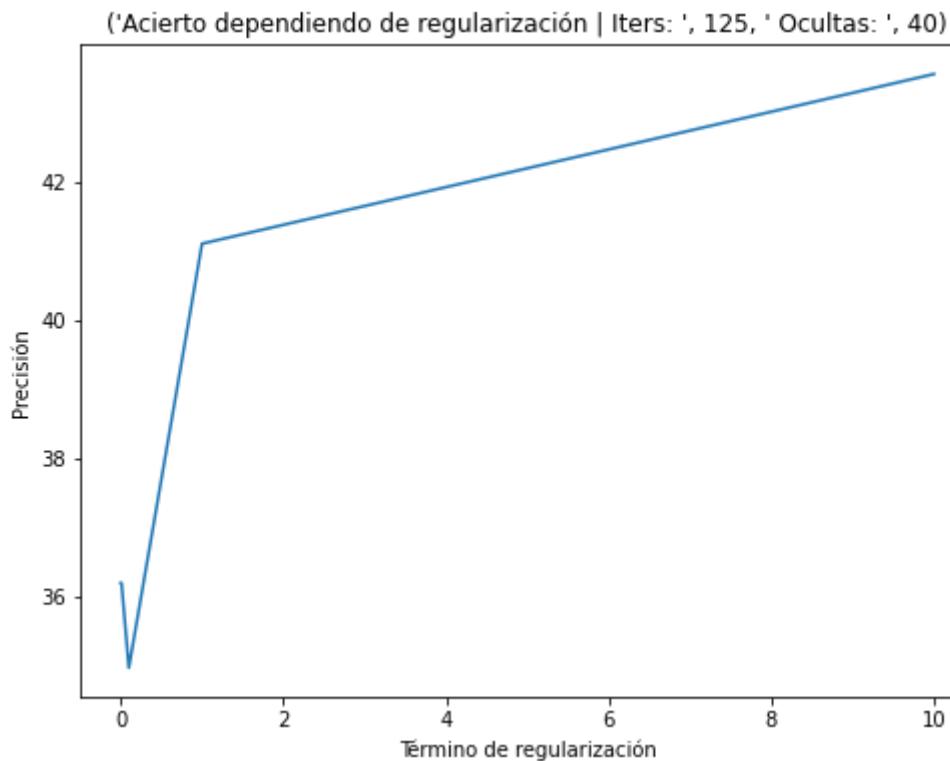
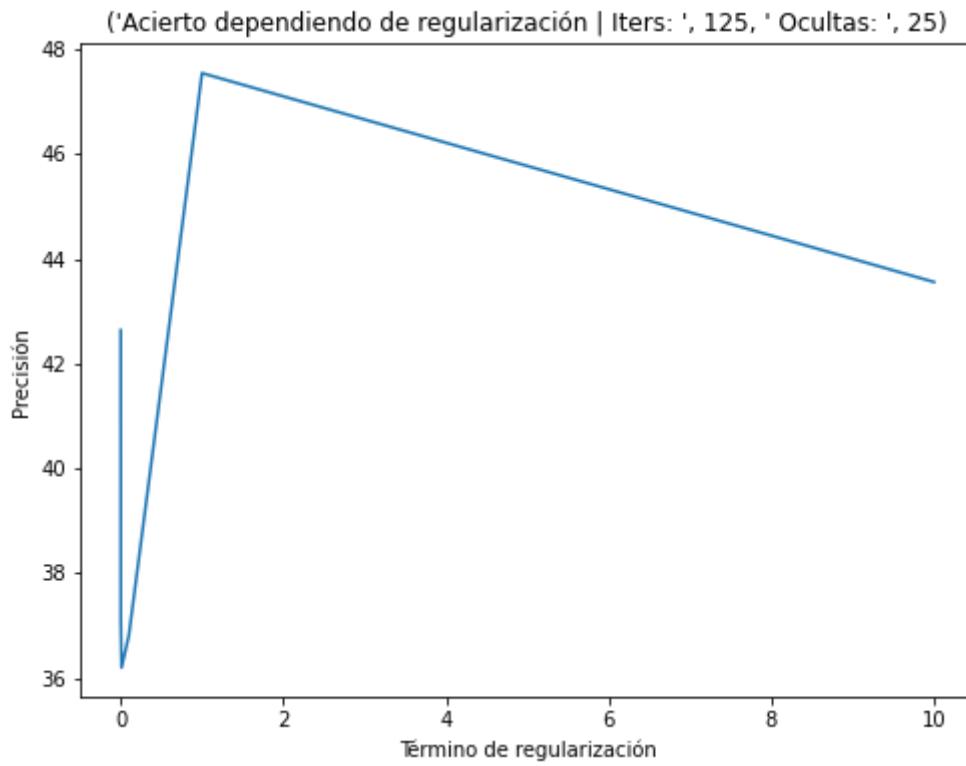
Clases: 5 | Reg: 0.01 | Ocultas: 50 | Iteraciones: 125

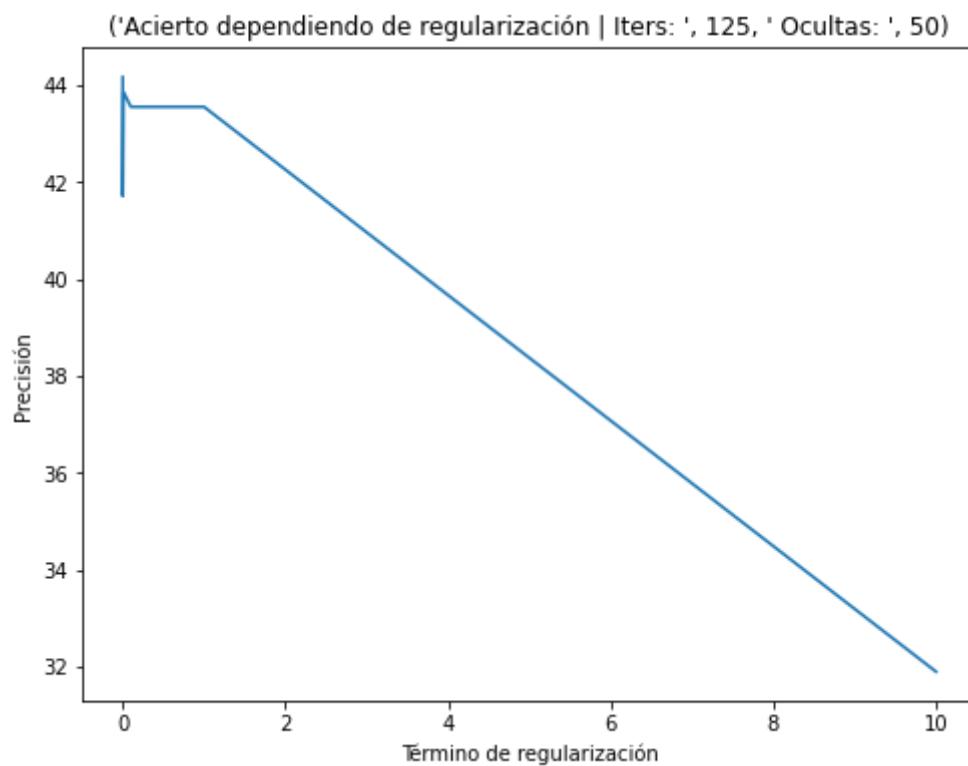
Tiempo: 11.71 segundos | Precisión: 43.86503067484663 %

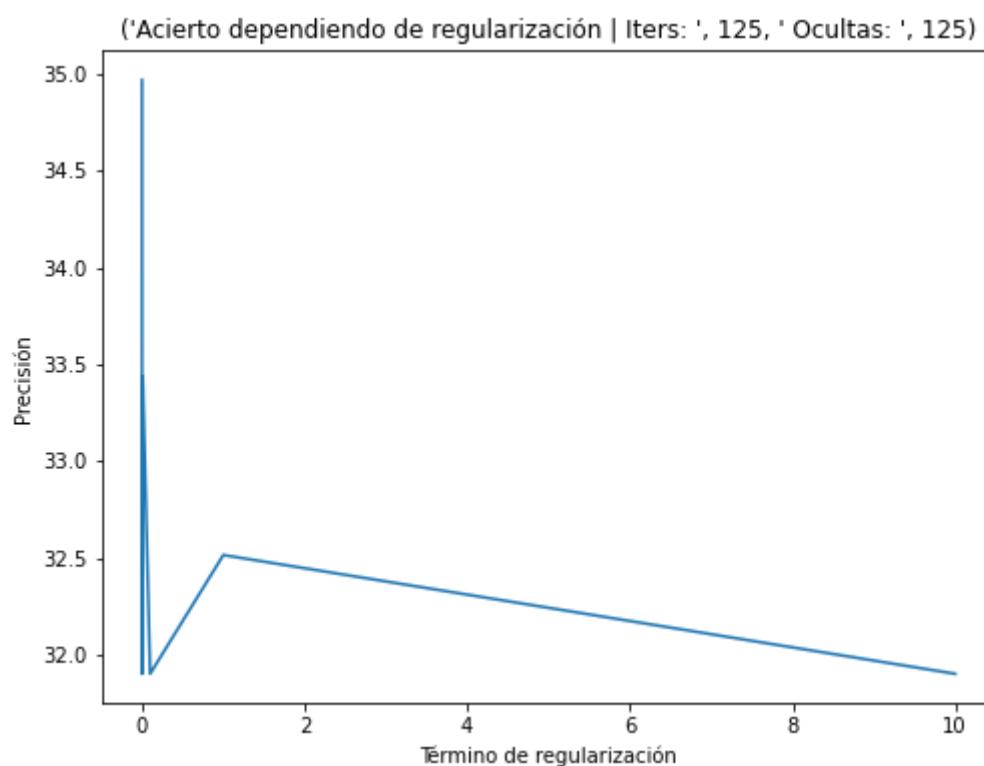
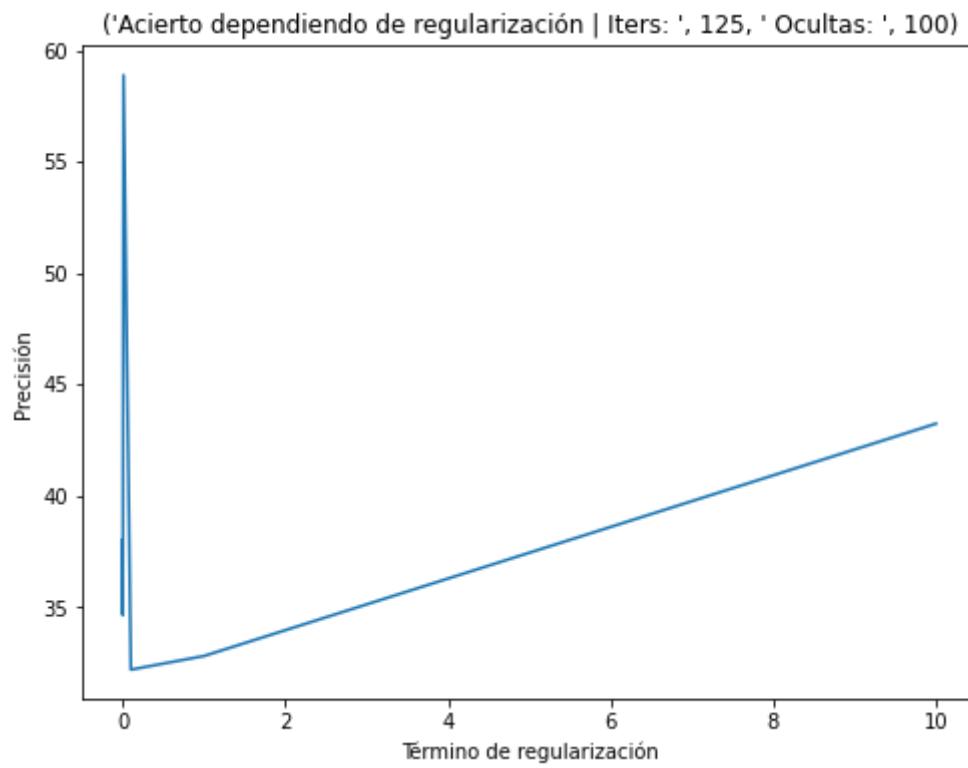
Clases: 5 | Reg: 0.1 | Ocultas: 50 | Iteraciones: 125

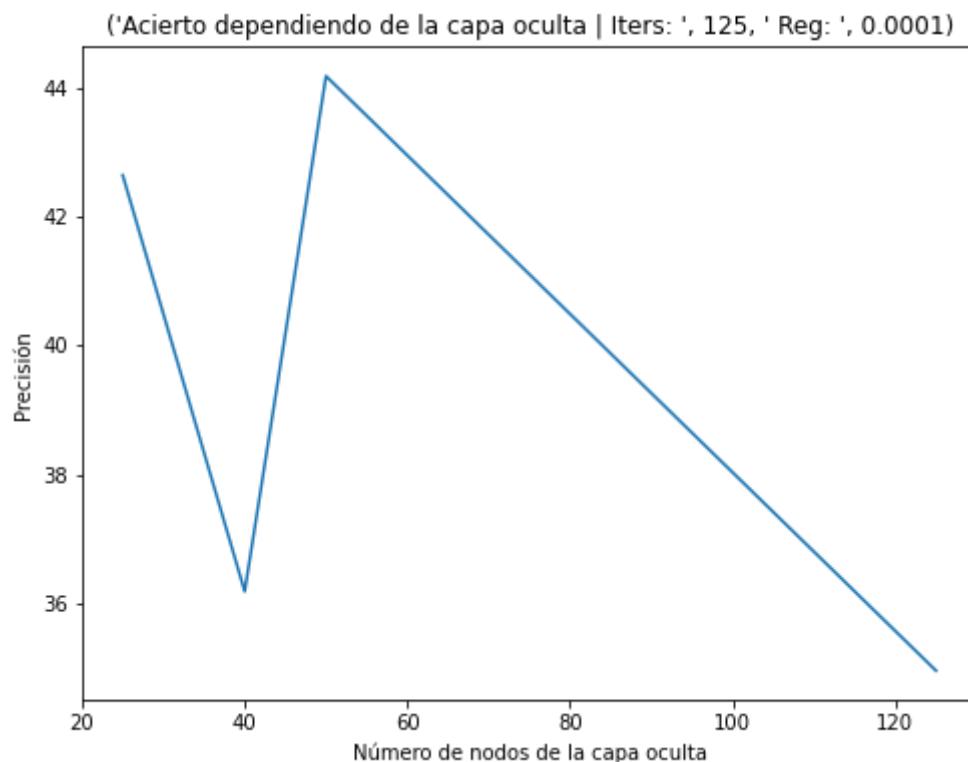
Tiempo: 12.88 segundos | Precisión: 43.558282208588956 %

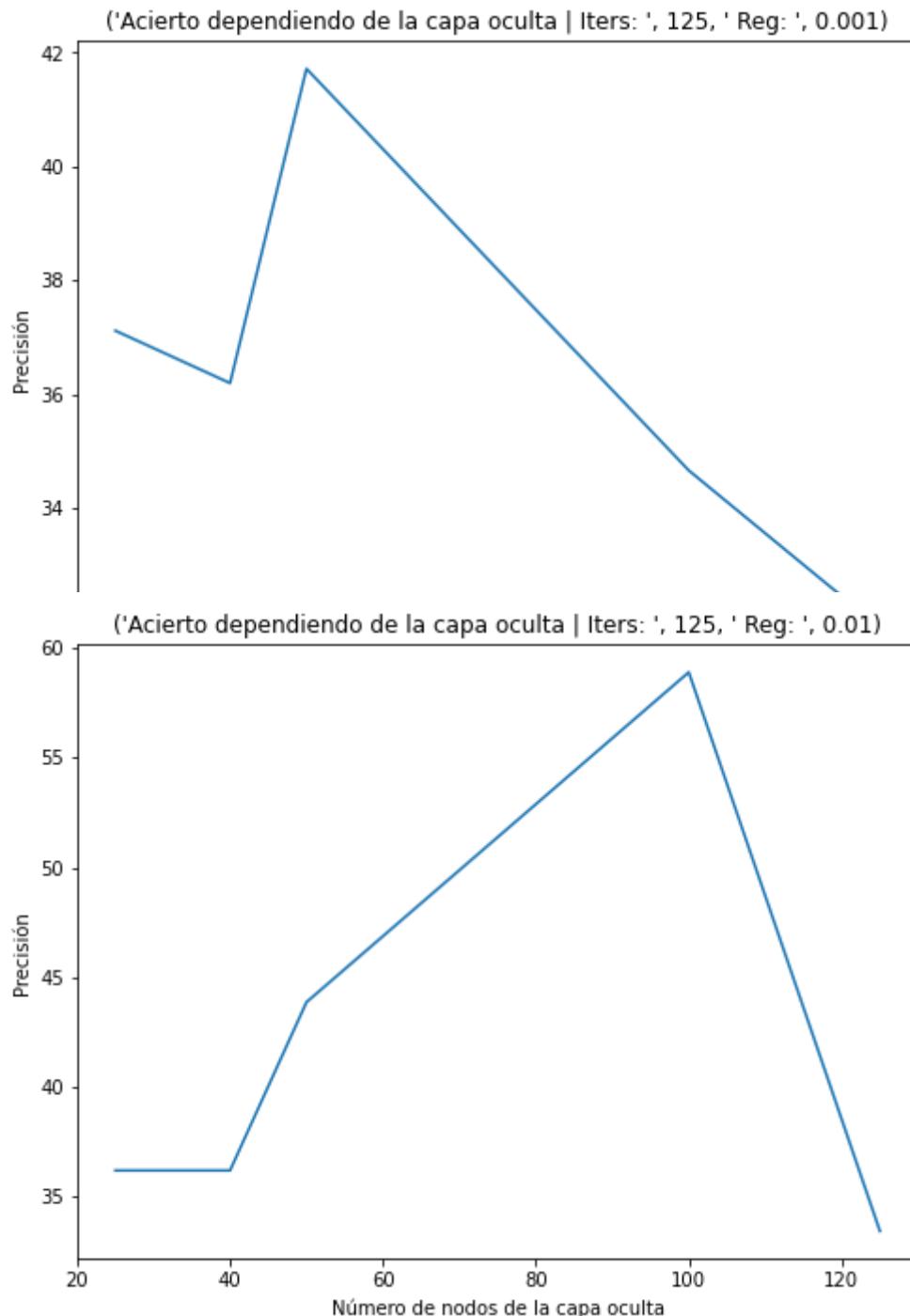
```
-----  
Clases: 5 | Reg: 1 | Ocultas: 50 | Iteraciones: 125  
Tiempo: 10.69 segundos | Precisión: 43.558282208588956 %  
-----  
Clases: 5 | Reg: 10 | Ocultas: 50 | Iteraciones: 125  
Tiempo: 10.19 segundos | Precisión: 31.901840490797547 %  
-----  
Clases: 5 | Reg: 0.0001 | Ocultas: 100 | Iteraciones: 125  
Tiempo: 24.65 segundos | Precisión: 38.036809815950924 %  
-----  
Clases: 5 | Reg: 0.001 | Ocultas: 100 | Iteraciones: 125  
Tiempo: 41.93 segundos | Precisión: 34.66257668711656 %  
-----  
Clases: 5 | Reg: 0.01 | Ocultas: 100 | Iteraciones: 125  
Tiempo: 54.67 segundos | Precisión: 58.895705521472394 %  
-----  
Clases: 5 | Reg: 0.1 | Ocultas: 100 | Iteraciones: 125  
Tiempo: 49.19 segundos | Precisión: 32.20858895705521 %  
-----  
Clases: 5 | Reg: 1 | Ocultas: 100 | Iteraciones: 125  
Tiempo: 57.86 segundos | Precisión: 32.82208588957055 %  
-----  
Clases: 5 | Reg: 10 | Ocultas: 100 | Iteraciones: 125  
Tiempo: 44.62 segundos | Precisión: 43.25153374233129 %  
-----  
Clases: 5 | Reg: 0.0001 | Ocultas: 125 | Iteraciones: 125  
Tiempo: 38.0 segundos | Precisión: 34.96932515337423 %  
-----  
Clases: 5 | Reg: 0.001 | Ocultas: 125 | Iteraciones: 125  
Tiempo: 25.37 segundos | Precisión: 31.901840490797547 %  
-----  
Clases: 5 | Reg: 0.01 | Ocultas: 125 | Iteraciones: 125  
Tiempo: 22.84 segundos | Precisión: 33.43558282208589 %  
-----  
Clases: 5 | Reg: 0.1 | Ocultas: 125 | Iteraciones: 125  
Tiempo: 22.85 segundos | Precisión: 31.901840490797547 %  
-----  
Clases: 5 | Reg: 1 | Ocultas: 125 | Iteraciones: 125  
Tiempo: 22.7 segundos | Precisión: 32.515337423312886 %  
-----  
Clases: 5 | Reg: 10 | Ocultas: 125 | Iteraciones: 125  
Tiempo: 25.62 segundos | Precisión: 31.901840490797547 %  
-----  
Tiempo total empleado: 594.07 segundos
```



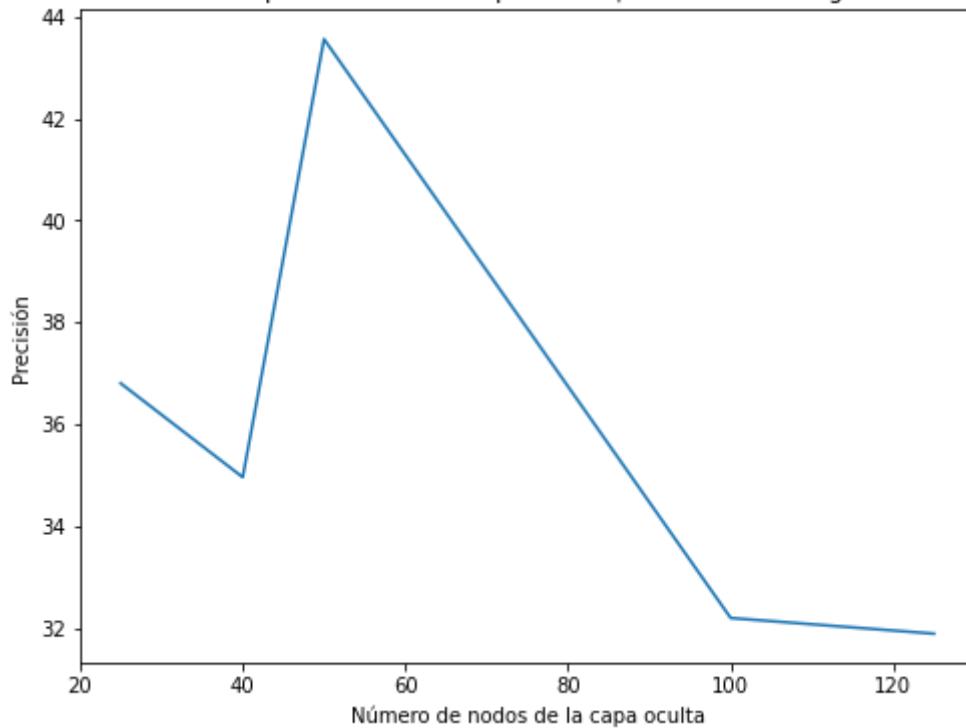




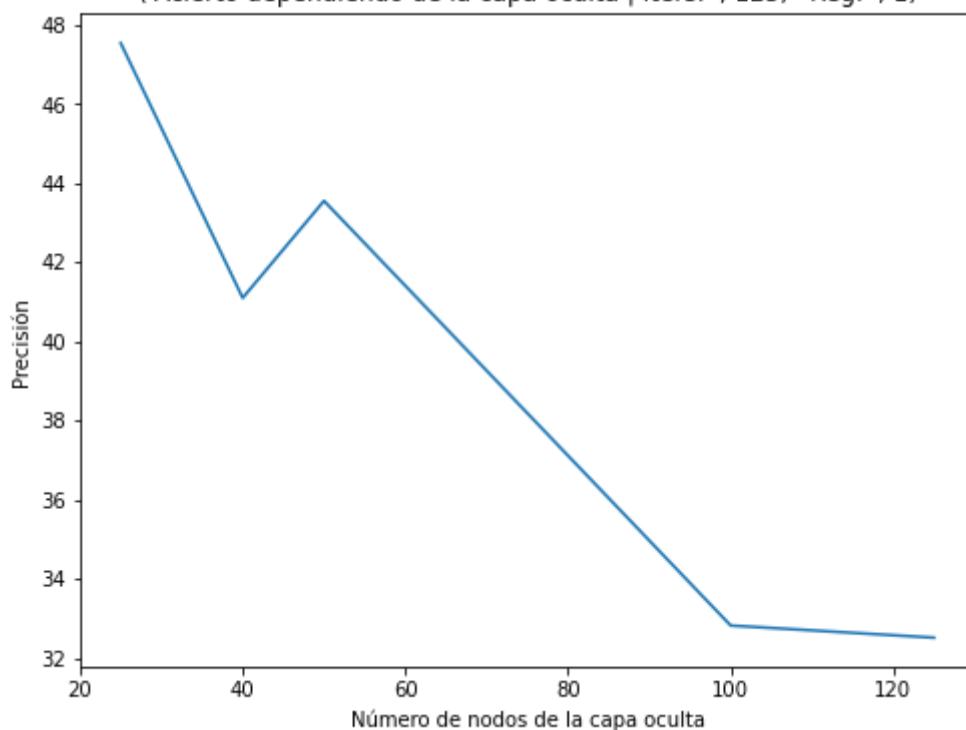


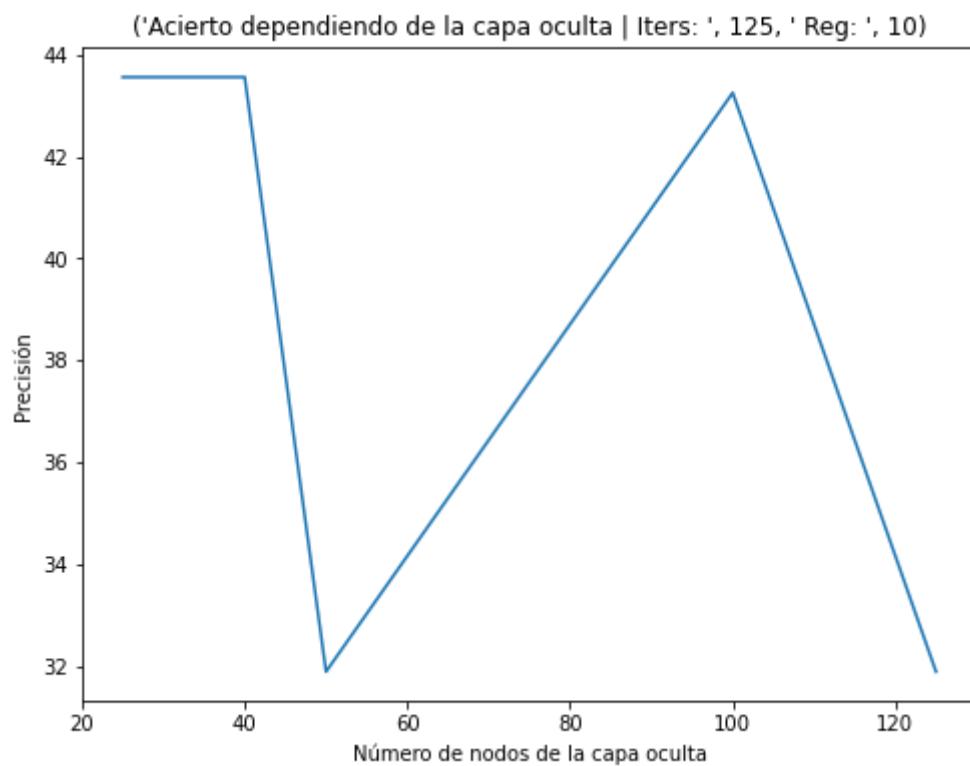


('Acierto dependiendo de la capa oculta | Iters: ', 125, ' Reg: ', 0.1)



('Acierto dependiendo de la capa oculta | Iters: ', 125, ' Reg: ', 1)





Test

In [18]:

```

1 max_ocultas, max_regs = get_maxim(regs, ocultas, resultados)
2 iterss_optim = [50, 100, 250, 500, 1000]
3
4 resultados_optim = funcion(X_train, y_train, X_test, y_test,
5                             letters, [max_regs], iterss_optim,
6                             [max_ocultas], Rand_Theta1, Rand_Theta2, 0)
7 dibujarGraficas([max_regs], iterss_optim, [max_ocultas], resultados_optim)

```

Tamaño de Train: 977

Tamaño de Test: 326

Clases: 5 | Reg: 0.01 | Ocultas: 100 | Iteraciones: 50

Tiempo: 11.77 segundos | Precisión: 31.901840490797547 %

Clases: 5 | Reg: 0.01 | Ocultas: 100 | Iteraciones: 100

Tiempo: 13.9 segundos | Precisión: 37.423312883435585 %

Clases: 5 | Reg: 0.01 | Ocultas: 100 | Iteraciones: 250

Tiempo: 35.5 segundos | Precisión: 61.34969325153374 %

Clases: 5 | Reg: 0.01 | Ocultas: 100 | Iteraciones: 500

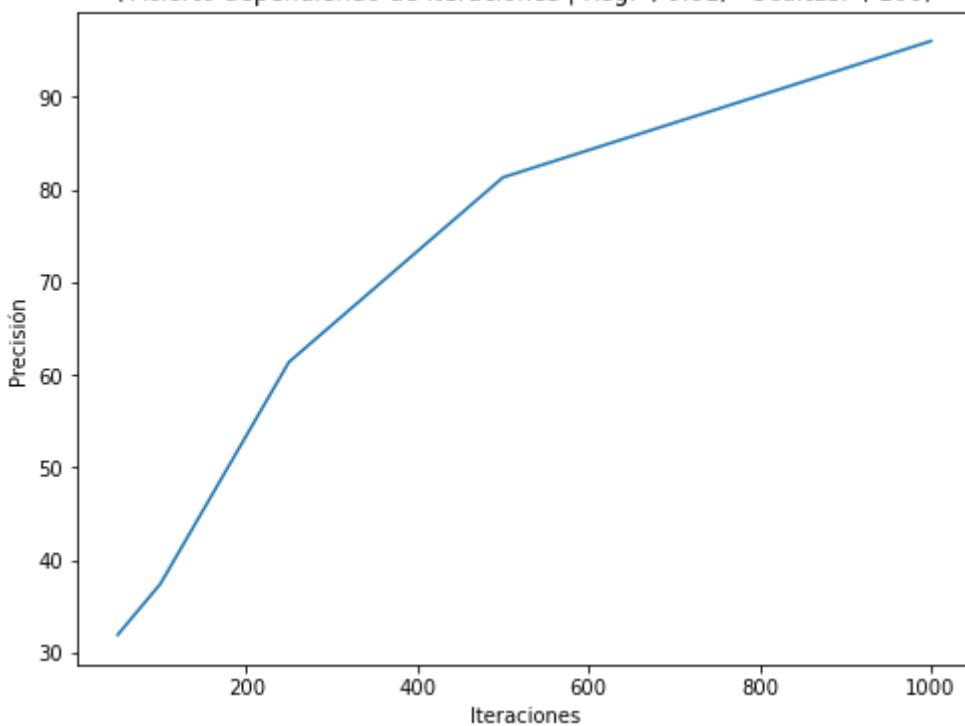
Tiempo: 70.83 segundos | Precisión: 81.28834355828221 %

Clases: 5 | Reg: 0.01 | Ocultas: 100 | Iteraciones: 1000

Tiempo: 147.09 segundos | Precisión: 96.0122699386503 %

Tiempo total empleado: 279.09 segundos

('Acierto dependiendo de iteraciones | Reg: ', 0.01, ' Ocultas: ', 100)



Óptimo

In [19]:

```
1 max_ocultas, max_regs = get_maxim(regs, ocultas, resultados)
2 iterss_optim = [2000]
3
4 theta1, theta2, resultados_optim = funcion(X_train, y_train,
5                               X_test, y_test, letters,
6                               [max_regs], iterss_optim,
7                               [max_ocultas], Rand_Theta1,
8                               Rand_Theta2, 1)
```

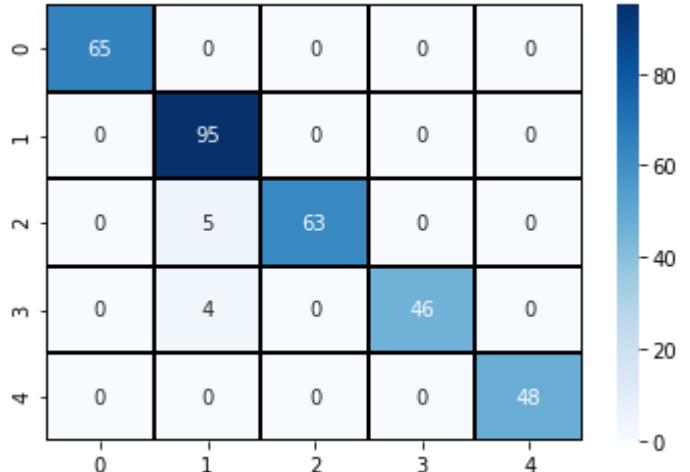
Tamaño de Train: 977

Tamaño de Test: 326

Clases: 5 | Reg: 0.01 | Ocultas: 100 | Iteraciones: 2000

Tiempo: 341.76 segundos | Precisión: 97.23926380368098 %

Tiempo total empleado: 341.76 segundos



Prueba de identificación de imágenes

In [20]:

```
1 Xs = [X_test[0], X_test[1], X_test[2]]  
2 letras = solve(Xs, theta1, theta2, letters)  
3 print('Predicción:', letras)  
4 print('Real:',y_test[0],y_test[1],y_test[2])
```

Predicción: ['E' 'U' 'U']

Real: [4] [20] [20]

Regresión Logística

Librerías

In [1]:

```

1 import time
2 import scipy.io
3 import numpy as np
4 import pandas as pd
5 import seaborn as sns
6 import scipy.optimize as opt
7 import matplotlib.pyplot as plt
8 from sklearn.metrics import accuracy_score
9 from sklearn.metrics import confusion_matrix
10 from sklearn.model_selection import train_test_split

```

Funciones

In [2]:

```

1 def sigmoide(z): # g(z)
2     return (1 / (1 + np.exp(-z)))

```

In [3]:

```

1 def gradiente(Thetas, X, Y, lambdaaa):
2     H = sigmoide(X @ Thetas) # Hipótesis
3
4     return (1/len(X)) * (X.T @ (H-np.ravel(Y))) + (lambdaaa/len(X)) * Thetas

```

In [4]:

```

1 def coste(Thetas, X, Y, lambdaaa):
2     H = sigmoide(X @ Thetas) # Hipótesis
3
4     return (-1/len(X)) * (np.log(H).T @ Y + (1-np.ravel(Y)) @ np.log(1-H+1e-6)) +
5             (lambdaaa/(2*len(X))) * sum(Thetas[1:]**2)

```

In [5]:

```

1 def porcentaje_acertado(X, y, H):
2
3     pos_max = np.argmax(H, axis=1)+1
4     suma = sum(pos_max[:, np.newaxis]==y)
5
6     return ((suma/np.shape(H)[0])*100)[0]

```

In [6]:

```

1 def pintarMatrizAciertos(y_real, y_pred):
2     pos_max = np.argmax(y_pred, axis=1)+1
3     y_pred_ok = pos_max[:,np.newaxis]
4     cm = confusion_matrix(y_real,y_pred_ok)
5     fig, ax = plt.subplots(figsize=(10,10))
6     sns.heatmap(cm,cmap= "Blues", linecolor = 'black' , linewidth = 1 ,
7                  annot = True, fmt='',ax=ax)

```

In [7]:

```

1 def oneVsAll (X, y, num_etiquetas , reg) : # reg = Lambda
2     #oneVsAll entrena varios clasificadores por regresión Logística con término
3     #de regularización 'reg' y devuelve el resultado en una matriz, donde
4     #la fila i-ésima corresponde al clasificador de la etiqueta i-ésima
5
6     Thetas = np.zeros(np.shape(X)[1])
7     All_Thetas = np.zeros([num_etiquetas, np.shape(Thetas)[0]])
8
9     for i in range(num_etiquetas):
10         y_add = (y == (i+1)) * 1      # vector de booleanos
11         result = opt.fmin_tnc (func=coste , x0=Thetas ,
12                             fprime=gradiente , args=(X, y_add, reg))
13         All_Thetas[i, :] = result[0]
14
15     return All_Thetas

```

In [8]:

```

1 def solve(X, Thetas):
2     letras = np.array(['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I',
3                        'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R',
4                        'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z'])
5
6     X = np.array(X)
7     m = np.shape(X)[0]
8     X1s = np.hstack([np.ones([m,1]), X])
9     H = sigmoide(X1s @ Thetas.T)
10    letras_i = np.argmax(H, axis=1)+1
11
12    return letras[letras_i]

```

In [9]:

```

1 def funcion(X_train, X_test, y_train, y_test, num_etiquetas, reg_vec):
2     start_timef = time.time()
3     m_train = np.shape(X_train)[0]
4     X1s_train = np.hstack([np.ones([m_train, 1]), X_train])
5
6     m_test = np.shape(X_test)[0]
7     X1s_test = np.hstack([np.ones([m_test, 1]), X_test])
8
9     maxim = 0
10
11    for i, reg_i in enumerate(reg_vec):
12        print('-----')
13        start_time = time.time()
14        All_Thetas = oneVsAll(X1s_train, y_train, num_etiquetas, reg_i)
15
16        resultado = porcentaje_acertado(X1s_test, y_test, sigmoide(
17            X1s_test @ All_Thetas.T))
18
19        if (resultado > maxim):
20            All_Thetas_maxim = All_Thetas
21
22        print('Entrenamiento completado con reg: ', reg_i)
23        print ('Tiempo empleado: ', np.round(
24            (time.time() - start_time),2), 'segundos')
25        print("Precisión:", resultado, '%')
26        print('-----')
27        print ('Tiempo total: ', np.round(
28            (time.time() - start_timef),2), 'segundos')
29
30    return All_Thetas_maxim

```

Cargando datos

In [10]:

```

1 data_train = pd.read_csv("archive/sign_mnist_train.csv")
2 data_test = pd.read_csv("archive/sign_mnist_test.csv")

```

In [11]:

```

1 X_train = data_train.drop(['label'],axis=1)
2 y_train = data_train['label']
3 X_test = data_test.drop(['label'],axis=1)
4 y_test = data_test['label']
5
6 y_train = y_train.values.reshape(len(y_train),1)
7 y_test = y_test.values.reshape(len(y_test),1)
8
9 X_train = X_train.reset_index(drop=True)
10 X_test = X_test.reset_index(drop=True)
11
12 X_train = np.array(X_train)
13 y_train = np.array(y_train)
14 X_test = np.array(X_test)
15 y_test = np.array(y_test)

```

Sólo dataset de Test

In [12]:

```
1 X_train1, X_test1, y_train1, y_test1 = train_test_split(  
2     X_test, y_test, test_size=0.3, random_state=50)  
3 print('Tamaño de Train: ', len(y_train1))  
4 print('Tamaño de Test: ', len(y_test1))  
5 All_Thetas1 = funcion(X_train1, X_test1, y_train1,  
6                         y_test1, 24, [0.0001, 0.001, 0.01, 0.1, 1, 10])
```

Tamaño de Train: 5020

Tamaño de Test: 2152

Entrenamiento completado con reg: 0.0001

Tiempo empleado: 20.84 segundos

Precisión: 95.30669144981412 %

Entrenamiento completado con reg: 0.001

Tiempo empleado: 24.04 segundos

Precisión: 95.30669144981412 %

Entrenamiento completado con reg: 0.01

Tiempo empleado: 21.97 segundos

Precisión: 95.30669144981412 %

Entrenamiento completado con reg: 0.1

Tiempo empleado: 17.07 segundos

Precisión: 95.30669144981412 %

Entrenamiento completado con reg: 0

Tiempo empleado: 18.44 segundos

Precisión: 95.30669144981412 %

Entrenamiento completado con reg: 1

Tiempo empleado: 20.88 segundos

Precisión: 95.30669144981412 %

Entrenamiento completado con reg: 10

Tiempo empleado: 22.81 segundos

Precisión: 95.30669144981412 %

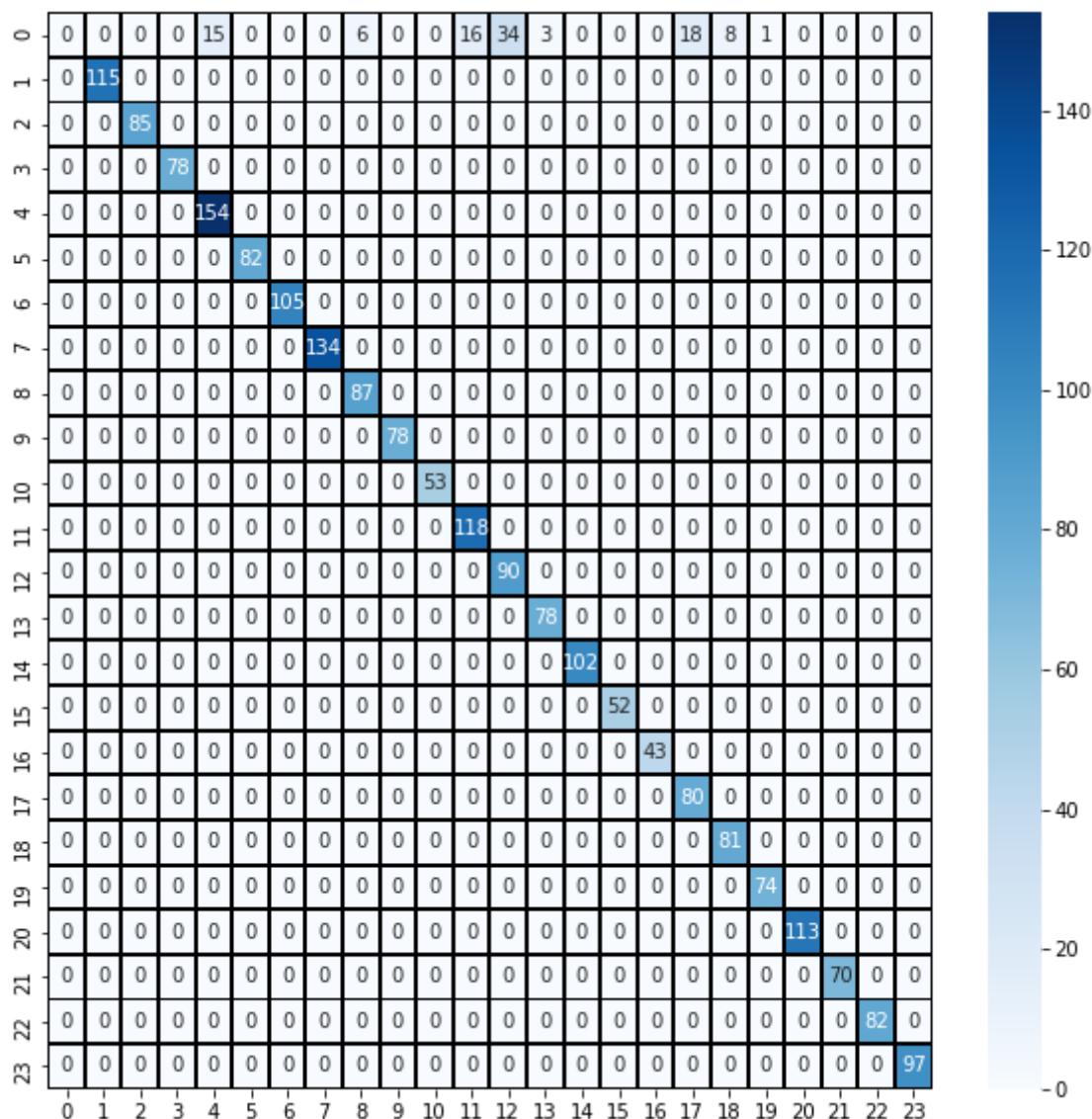
Tiempo total: 146.08 segundos

In [13]:

```

1 m_test = np.shape(X_test1)[0]
2 X1s_test = np.hstack([np.ones([m_test,1]), X_test1])
3 pintarMatrizAciertos(y_test1, sigmoide(X1s_test @ All_Thetas1.T))

```



Sólo dataset de Train

In [14]:

```
1 X_train2, X_test2, y_train2, y_test2 = train_test_split(  
2     X_train, y_train, test_size=0.3, random_state=50)  
3 print('Tamaño de Train: ', len(y_train2))  
4 print('Tamaño de Test: ', len(y_test2))  
5 All_Thetas2 = funcion(X_train2, X_test2  
6                         , y_train2, 24, [0.0001,0.001,0.01,0.1,0,1,10])
```

Tamaño de Train: 19218

Tamaño de Test: 8237

Entrenamiento completado con reg: 0.0001

Tiempo empleado: 168.54 segundos

Precisión: 95.64161709360204 %

Entrenamiento completado con reg: 0.001

Tiempo empleado: 170.78 segundos

Precisión: 95.64161709360204 %

Entrenamiento completado con reg: 0.01

Tiempo empleado: 169.05 segundos

Precisión: 95.64161709360204 %

Entrenamiento completado con reg: 0.1

Tiempo empleado: 168.15 segundos

Precisión: 95.64161709360204 %

Entrenamiento completado con reg: 0

Tiempo empleado: 159.21 segundos

Precisión: 95.64161709360204 %

Entrenamiento completado con reg: 1

Tiempo empleado: 206.35 segundos

Precisión: 95.65375743595969 %

Entrenamiento completado con reg: 10

Tiempo empleado: 228.46 segundos

Precisión: 95.65375743595969 %

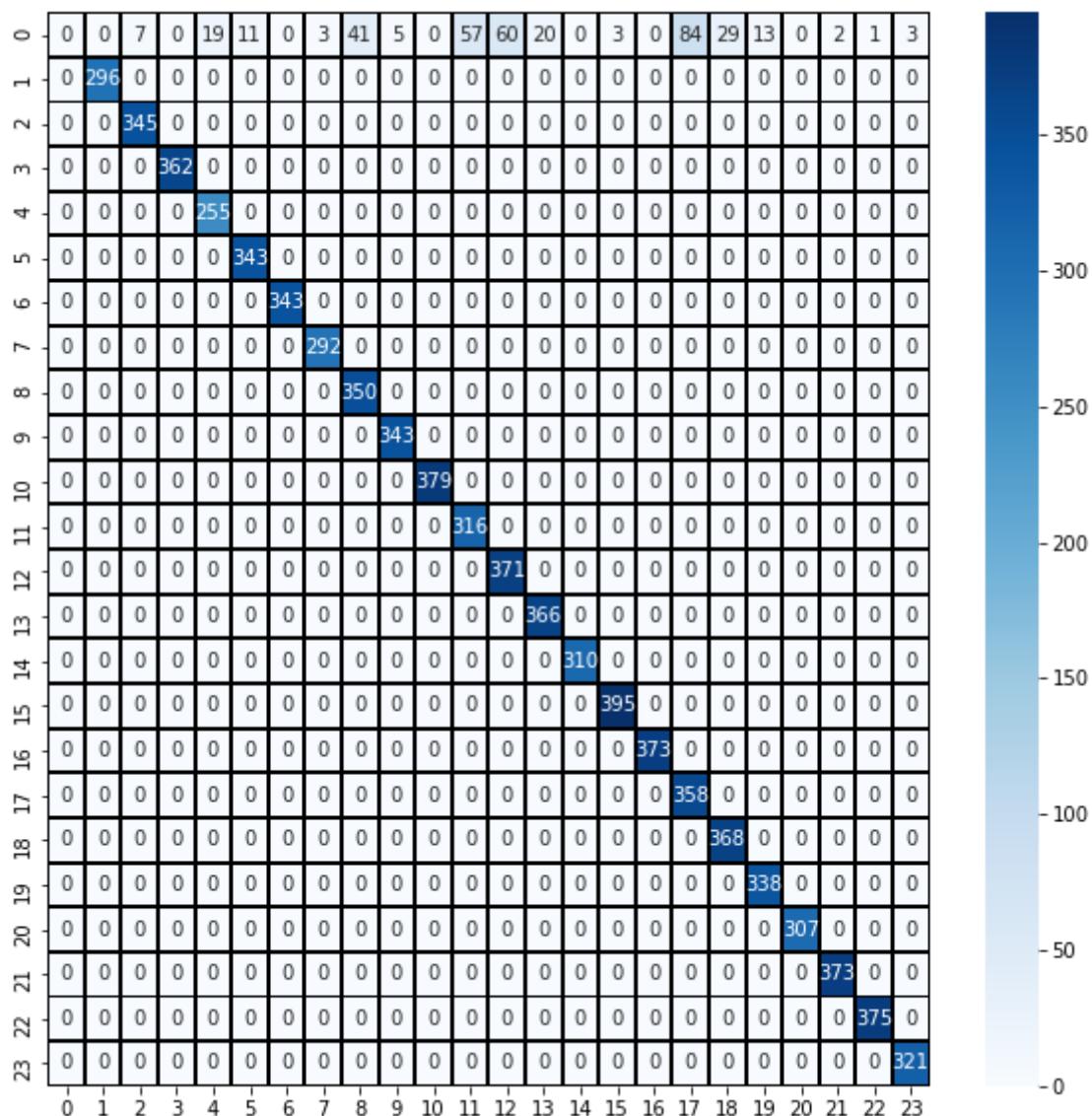
Tiempo total: 1270.65 segundos

In [15]:

```

1 m_test = np.shape(X_test2)[0]
2 X2s_test = np.hstack([np.ones([m_test,1]), X_test2])
3 pintarMatrizAciertos(y_test2, sigmoide(X2s_test @ All_Thetas2.T))

```



Dataset completo

In [16]:

```
1 print('Tamaño de Train: ', len(y_train))
2 print('Tamaño de Test: ', len(y_test))
3 All_Thetas3 = funcion(X_train, X_test,
4                         y_train, y_test, 24, [0.0001, 0.001, 0.01, 0.1, 0, 1, 10])
```

Tamaño de Train: 27455

Tamaño de Test: 7172

Entrenamiento completado con reg: 0.0001

Tiempo empleado: 231.32 segundos

Precisión: 55.27049637479086 %

Entrenamiento completado con reg: 0.001

Tiempo empleado: 208.37 segundos

Precisión: 55.828220858895705 %

Entrenamiento completado con reg: 0.01

Tiempo empleado: 221.1 segundos

Precisión: 55.68878973786949 %

Entrenamiento completado con reg: 0.1

Tiempo empleado: 220.29 segundos

Precisión: 55.96765197992192 %

Entrenamiento completado con reg: 0

Tiempo empleado: 204.52 segundos

Precisión: 55.409927495817065 %

Entrenamiento completado con reg: 1

Tiempo empleado: 220.4 segundos

Precisión: 56.28834355828221 %

Entrenamiento completado con reg: 10

Tiempo empleado: 268.27 segundos

Precisión: 56.52537646402676 %

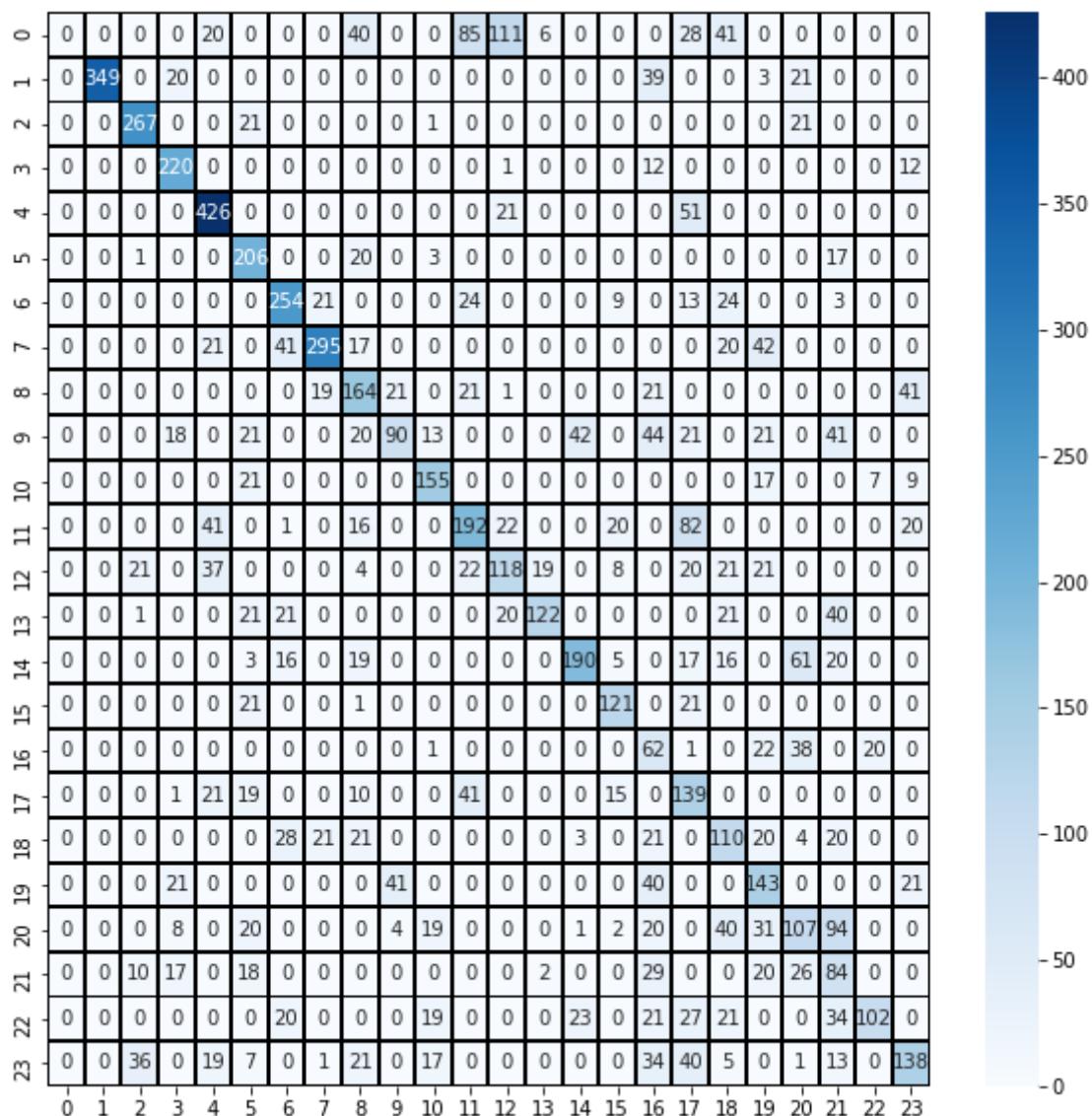
Tiempo total: 1574.36 segundos

In [17]:

```

1 m_test = np.shape(X_test)[0]
2 X3s_test = np.hstack([np.ones([m_test,1]), X_test])
3 pintarMatrizAciertos(y_test, sigmoide(X3s_test @ All_Thetas3.T))

```



Prueba de identificación de imágenes

In [18]:

```
1 Xs = [X_test[0], X_test[1], X_test[2]]
2 letras = solve(Xs, All_Thetas1)
3 print('Predicción:', letras)
4 print('Real:',y_test[0],y_test[1],y_test[2])
```

Predicción: ['G' 'F' 'K']

Real: [6] [5] [10]

Support Vector Machine (SVM)

Librerías

In [27]:

```

1 import time
2 import scipy.io
3 import numpy as np
4 import pandas as pd
5 from sklearn.metrics import accuracy_score
6 from sklearn.model_selection import train_test_split
7 from sklearn import svm
8 from sklearn.model_selection import GridSearchCV
9 from sklearn.metrics import confusion_matrix
10 import matplotlib.pyplot as plt
11 import seaborn as sns

```

Funciones

In [31]:

```

1 def pintarMatrizAciertos(y_real, y_pred):
2     pos_max = np.argmax(y_pred, axis=1)+1
3     cm = confusion_matrix(y_real,y_pred)
4     fig, ax = plt.subplots(figsize=(10,10))
5     sns.heatmap(cm,cmap= "Blues", linecolor = 'black' ,
6                  linewidth = 1 ,annot = True, fmt=' ',ax=ax)

```

In [17]:

```

1 def solve(X, model):
2     letras = np.array(['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J',
3                        'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T',
4                        'U', 'V', 'W', 'X', 'Y', 'Z'])
5
6     X = np.array(X)
7     y_pred = model.predict(X)
8
9     return letras[y_pred]

```

In [18]:

```

1 def funcion(X, y, C_vec, gamma_vec, kernel_vec):
2     start_timef = time.time()
3     param_grid={'C':C_vec,'gamma':gamma_vec,'kernel':kernel_vec}
4     svc=svm.SVC(probability=False)
5     model=GridSearchCV(svc,param_grid,n_jobs=-1)
6     model.fit(X,y.ravel())
7
8     print('Entrenamiento completado')
9     print ('Tiempo empleado: ', np.round((time.time() - start_timef),2),
10           'segundos')
11
12 return model

```

Cargando datos

In [19]:

```

1 data_train = pd.read_csv("archive/sign_mnist_train.csv")
2 data_test = pd.read_csv("archive/sign_mnist_test.csv")

```

In [21]:

```

1 X_train = data_train.drop(['label'],axis=1)
2 y_train = data_train['label']
3 X_test = data_test.drop(['label'],axis=1)
4 y_test = data_test['label']
5
6 y_train = y_train.values.reshape(len(y_train),1)
7 y_test = y_test.values.reshape(len(y_test),1)
8
9 X_train = X_train.reset_index(drop=True)
10 X_test = X_test.reset_index(drop=True)
11
12 X_train = np.array(X_train)
13 y_train = np.array(y_train)
14 X_test = np.array(X_test)
15 y_test = np.array(y_test)

```

Sólo dataset de test

In [22]:

```

1 X_train1, X_test1, y_train1, y_test1 = train_test_split(X_test, y_test,
2                                         test_size=0.3, random_state=50)
3 print('Tamaño de Train: ', len(y_train1))
4 print('Tamaño de Test: ', len(y_test1))

```

Tamaño de Train: 5020

Tamaño de Test: 2152

In [34]:

```

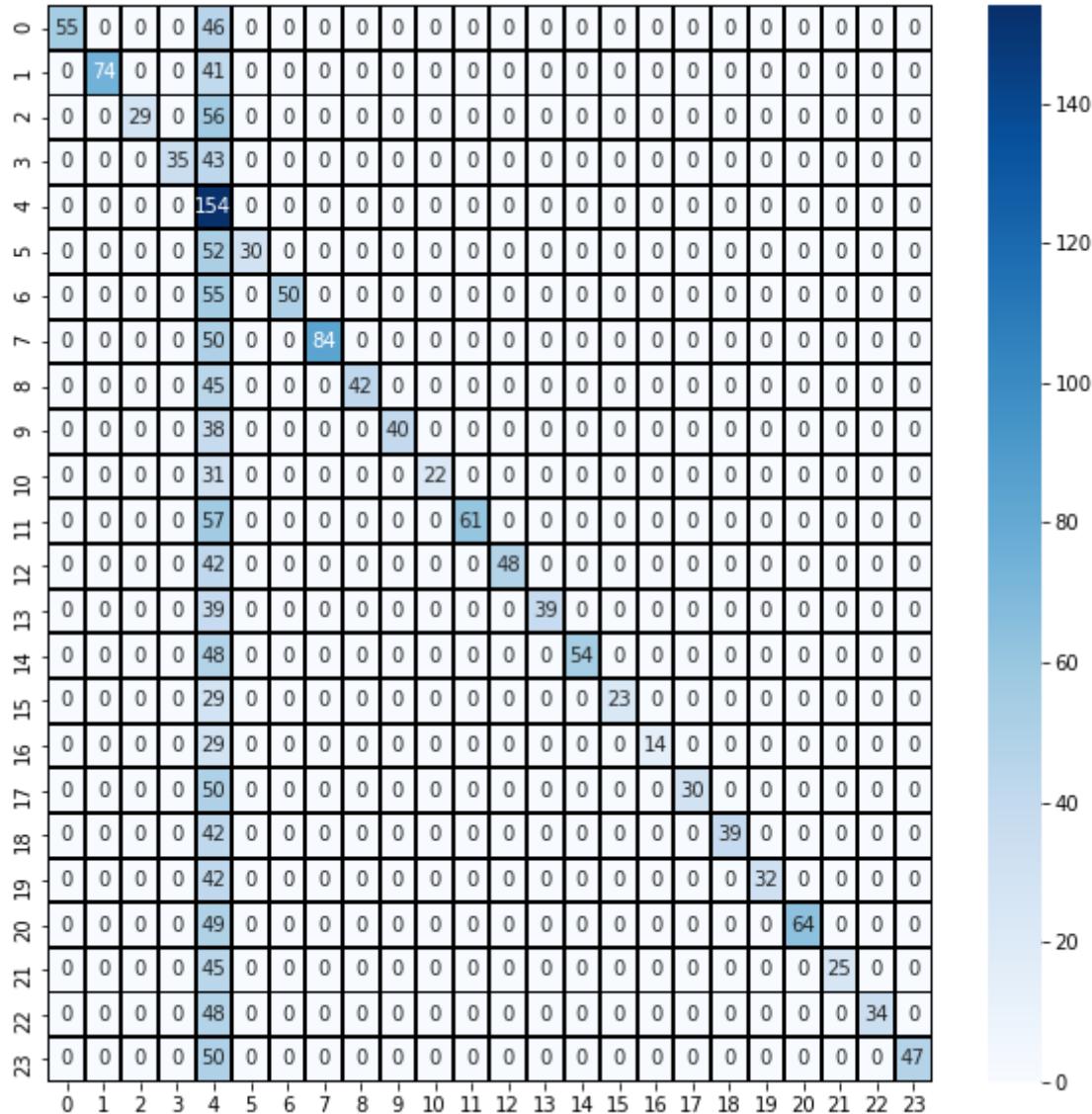
1 model0 = funcion(X_train1, y_train1, [0.1,1,10,100], [0.0001,0.001,0.1,1], ['rbf'])
2 print('Precisión: ', accuracy_score(model0.predict(X_test1),y_test1)*100, '%')
3 pintarMatrizAciertos(y_test1, model0.predict(X_test1).reshape(len(X_test1),1))

```

Entrenamiento completado

Tiempo empleado: 1085.5 segundos

Precisión: 52.276951672862445 %



In [33]:

```

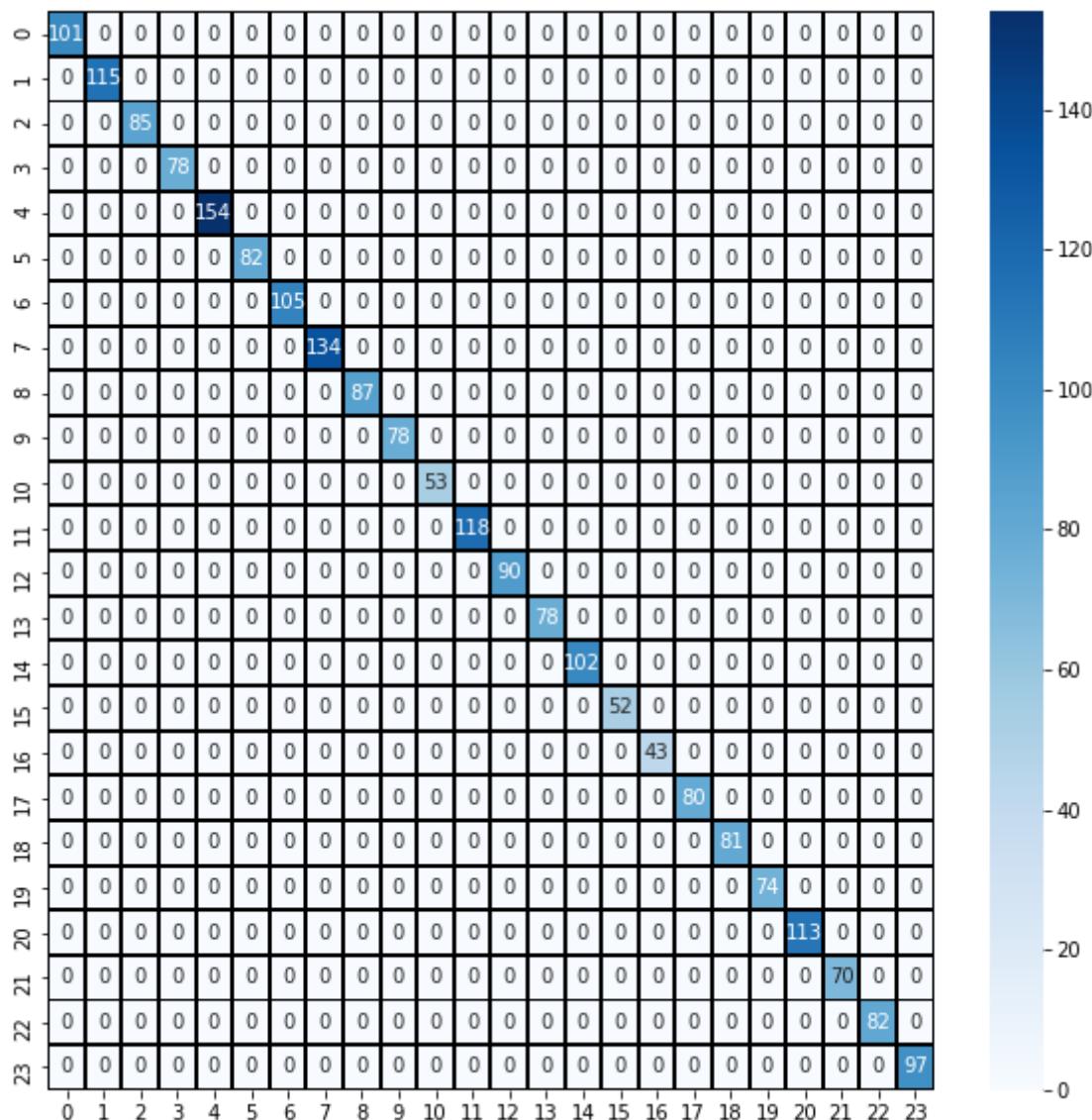
1 model1 = funcion(X_train1, y_train1, [0.1,1,10,100], [0.0001,0.001,0.1,1], ['poly'])
2 print('Precisión: ', accuracy_score(model1.predict(X_test1),y_test1)*100, '%')
3 pintarMatrizAciertos(y_test1, model1.predict(X_test1).reshape(len(X_test1),1))

```

Entrenamiento completado

Tiempo empleado: 106.53 segundos

Precisión: 100.0 %



Sólo dataset de train

In [35]:

```

1 X_train2, X_test2, y_train2, y_test2 = train_test_split(X_train,
2                                         y_train,
3                                         test_size=0.3,
4                                         random_state=50)
5 print('Tamaño de Train: ', len(y_train2))
6 print('Tamaño de Test: ', len(y_test2))

```

Tamaño de Train: 19218

Tamaño de Test: 8237

In [36]:

```

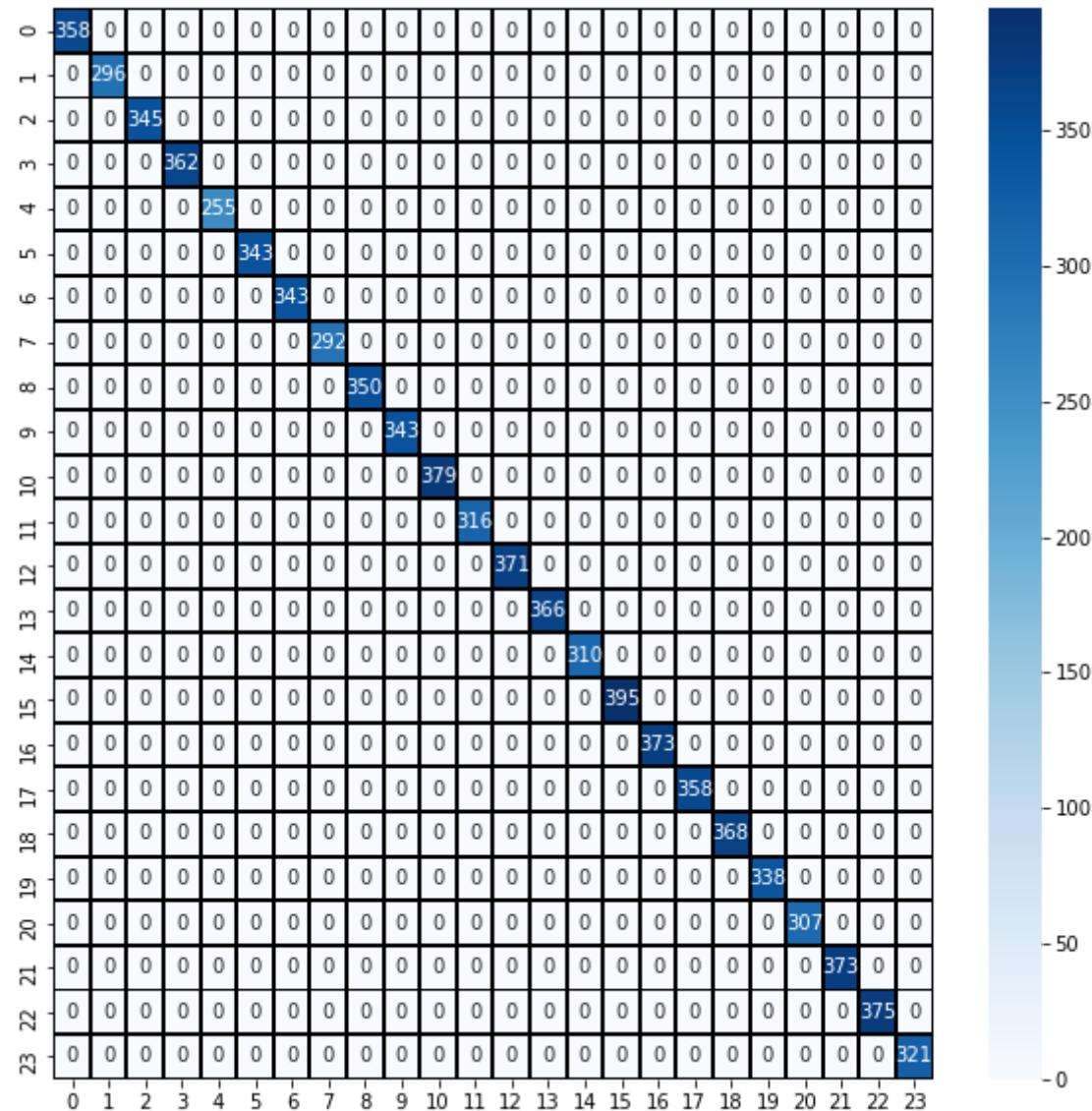
1 model2 = funcion(X_train2, y_train2, [0.1,1,10,100], [0.0001,0.001,0.1,1], ['poly'])
2 print('Precisión: ', accuracy_score(model2.predict(X_test2),y_test2)*100, '%')
3 pintarMatrizAciertos(y_test2, model2.predict(X_test2).reshape(len(X_test2),1))

```

Entrenamiento completado

Tiempo empleado: 1046.08 segundos

Precisión: 100.0 %



Dataset completo

In [37]:

```
1 print('Tamaño de Train: ', len(y_train))
2 print('Tamaño de Test: ', len(y_test))
```

Tamaño de Train: 27455

Tamaño de Test: 7172

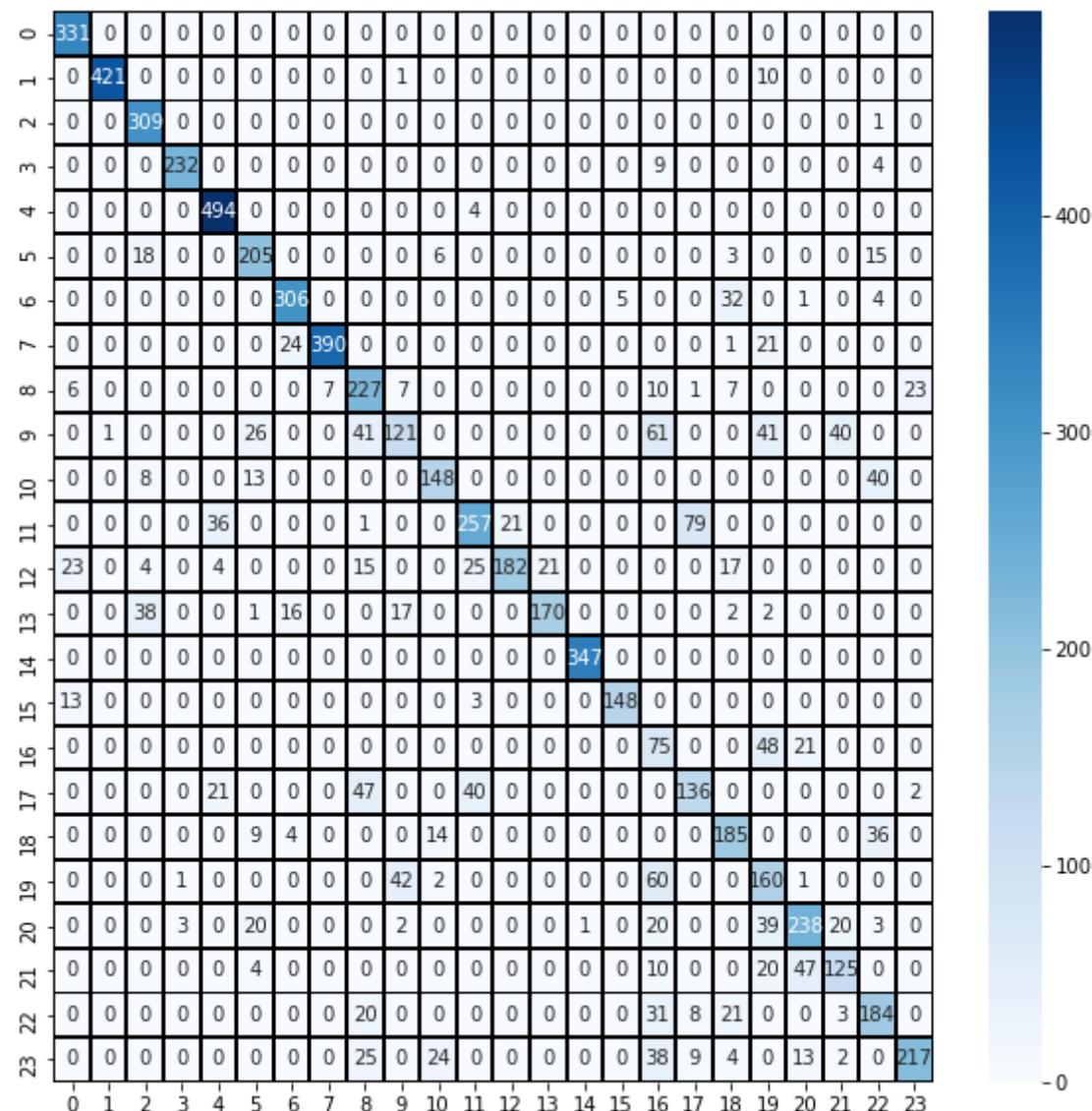
In [38]:

```
1 model3 = funcion(X_train, y_train, [0.1,1,10,100], [0.0001,0.001,0.1,1], ['poly'])
2 print('Precisión: ', accuracy_score(model3.predict(X_test),y_test)*100, '%')
3 pintarMatrizAciertos(y_test, model3.predict(X_test).reshape(len(X_test),1))
```

Entrenamiento completado

Tiempo empleado: 1472.89 segundos

Precisión: 78.19297267150029 %



Prueba de identificación de imágenes

In [39]:

```
1 Xs = [X_test[0], X_test[1], X_test[2]]  
2 letras = solve(Xs, model1)  
3 print('Predicción:', letras)  
4 print('Real:', y_test[0], y_test[1], y_test[2])
```

Predicción: ['G' 'F' 'K']

Real: [6] [5] [10]