Pec2_redes_neuronales

December 6, 2023

PEC2: Actividad y Debate

Machine Learning

Demetrio Muñoz Alvarez

[ ]:
```
# Cargamos e instalamos las librerías necesarias para realizar la actividad:
!pip install tensorflow
import tensorflow as tf
import pandas as pd
import numpy as np
import seaborn as sns
import random
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report,␣
 ↪accuracy_score
from tensorflow.keras import layers, models
# Se eliminan los outputs para no sobrecargar el informe.
```

1 Cargar los datos (data3.csv y clase3.csv).

[4]:
```
file_data = 'data3.csv' # Establecemos la ruta  de los archivos a  importar.
file_class = 'class3.csv'
data3 = pd.read_csv(file_data) #  Impotamos los  conjuntos de datos usando la␣
 ↪librería "Pandas"
class3 = pd.read_csv(file_class)
class3 = class3.rename(columns = {'x': 'Class'})  # Renombramos la columna 'x'␣
 ↪del conjunto de datos "class3" a 'Class' para una mayorcom prension.
class3['Class'] = class3['Class'] - 1 # Restamos 1 a cada valor en la columna␣
 ↪'Class', transformándolos de '1 a 8' a '0 a 7'. Este paso lo realizamos para␣
 ↪procesos posteriores.
```

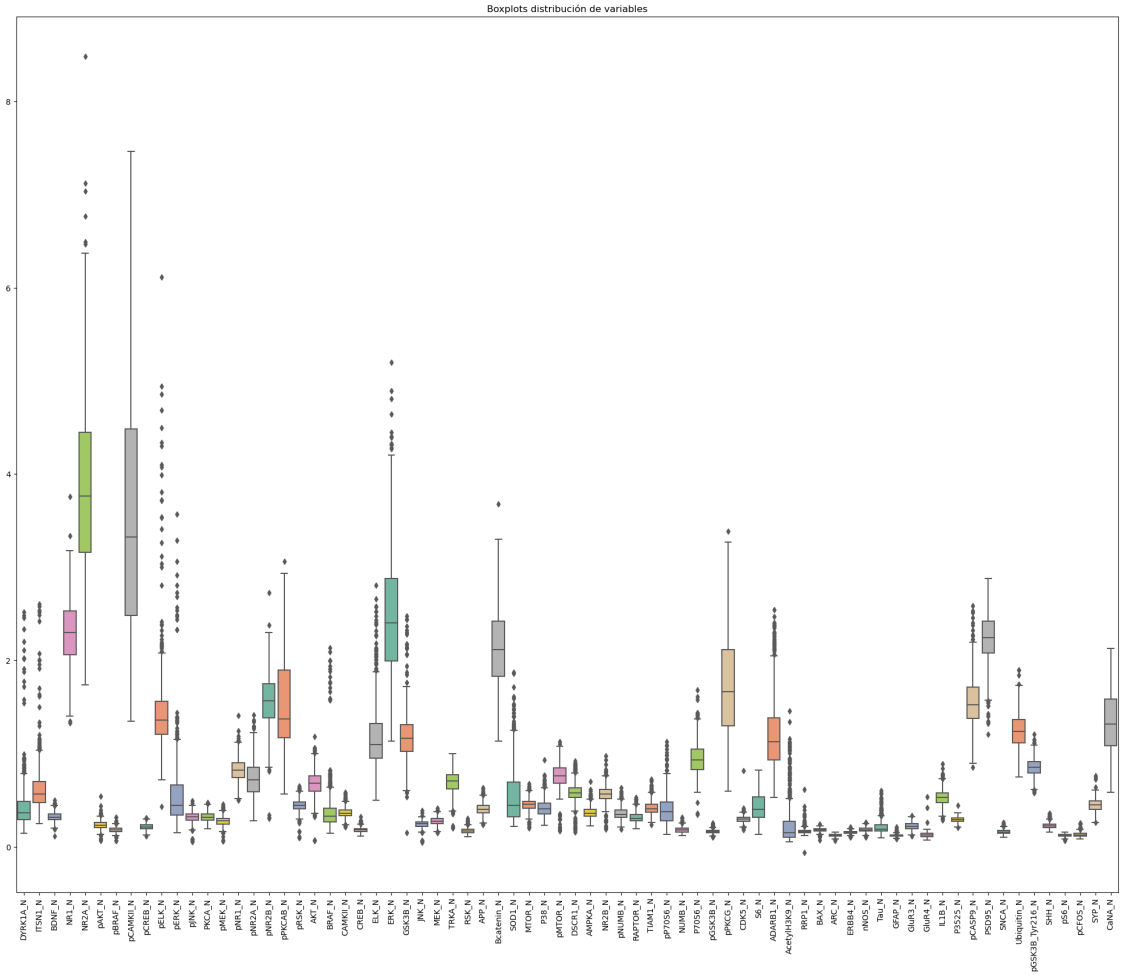
2 Realizar un estudio exploratorio de los datos con gráficos y tablas.

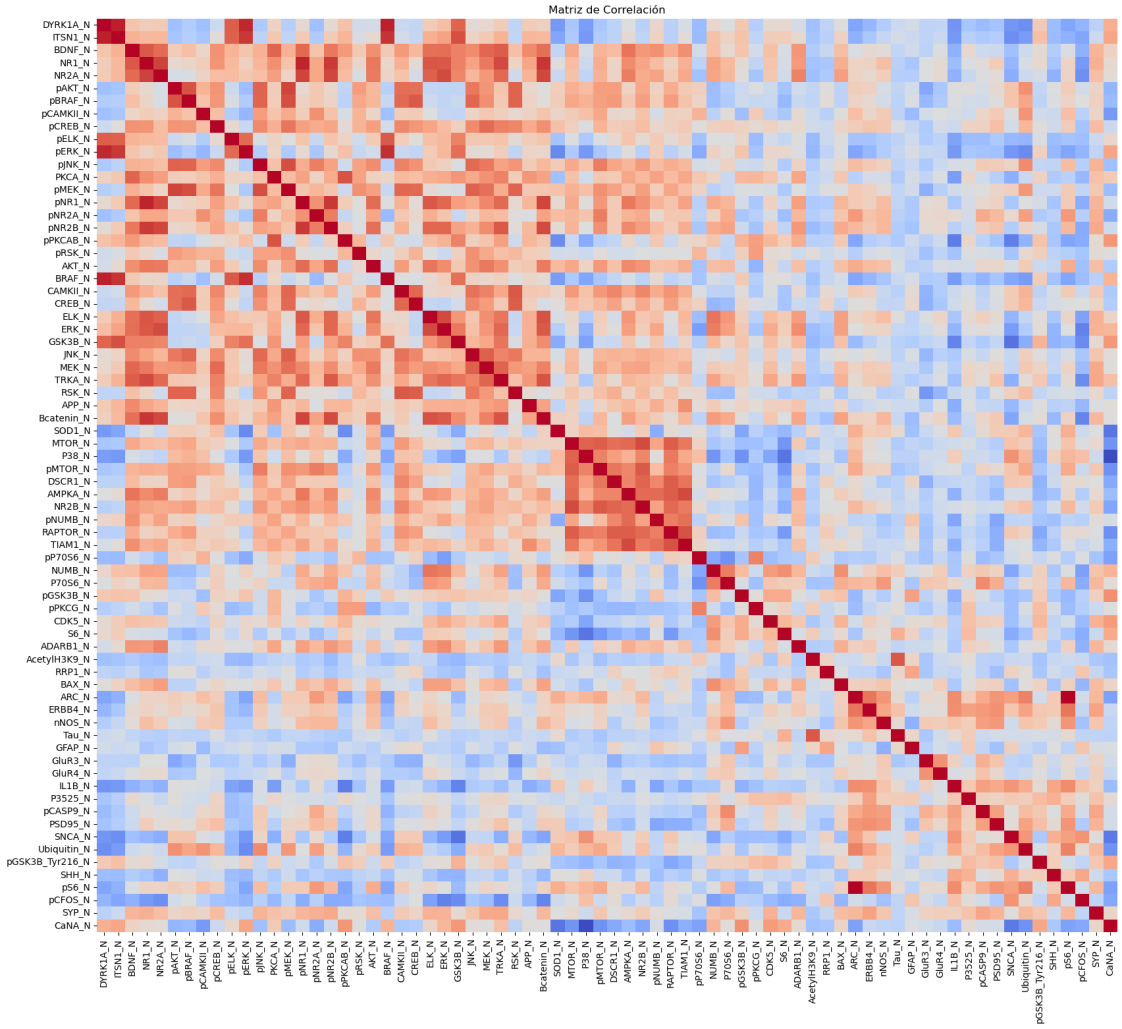
[6]:
```
# Para este apartado realizamos un analisis exploratorio de los datos del conjunto de datos 'data3' y 'class3':

# El conjunto de datos 'class3' solo encontramos el tipo de grupo o clase para la clasificación, tenemos 8 clases, categorizadas de 1 a 8.
print("Número de clases en 'class3':")
print(sorted(class3['Class'].unique())) # Para no tener problemas a la hora de implementar los modelos, hemos restamos una unidad a los valores de las clases, es decir, la clase 0 de nuestros datos corresponde a la clase 1 del enunciado de la PEC, así respectivamente con las demás clases:
# Mostramos las primeras entradas del conjunto de datos 'data3':
print("Data3:")
print(data3.head())
# Obtenemos un summary de las variables del conjunto de 'datos3':
print("\nData3 Summary:")
print(data3.describe())

# Exploramos si tenemos algún valor NA:
total_na = sum(data3.iloc[:, 1:].isnull().sum())
print(f"Número total de valores nulos: {total_na}") # No se encuentran valores nulos en el conjunto de datos de las variables.

# Valores atípicos de las variable:
data3_outliders = data3.drop('MouseID', axis=1)
# Calcular el rango intercuartílico (IQR) para cada columna
Q1 = data3_outliders.quantile(0.25)
Q3 = data3_outliders.quantile(0.75)
IQR = Q3 - Q1
# Identificamos los valores atípicos usando el 'IQR':
outliers = ((data3_outliders < (Q1 - 1.5 * IQR)) | (data3_outliders > (Q3 + 1.5 * IQR)))
total_outliers = outliers.sum().sum()
# Mostramos los valores atípicos
print(f'Número total de valores atípicos: {total_outliers}')
# Boxplot de la distribución:
plt.figure(figsize = (25, 20))
sns.boxplot(data = data3_outliders, palette = 'Set2')
plt.xticks(rotation = 90)
plt.title('Boxplots distribución de variables')
plt.show()

# Matriz de correlación:
data3_corr = data3.drop('MouseID', axis=1) # Exluimos 'MouseID' de la correalción.
```

```
correlation_mtr = data3_corr.corr()
# Heatmap:
plt.figure(figsize = (20, 18))
sns.heatmap(correlation_mtr, cmap = 'coolwarm', cbar = False)
plt.title('Matriz de Correlación')
plt.show()
```

```
Número de clases en 'class3':
[0, 1, 2, 3, 4, 5, 6, 7]
Data3:
  MouseID  DYRK1A_N   ITSN1_N    BDNF_N     NR1_N    NR2A_N    pAKT_N  \
0   309_1  0.503644  0.747193  0.430175  2.816329  5.990152  0.218830
1   309_2  0.514617  0.689064  0.411770  2.789514  5.685038  0.211636
2   309_3  0.509183  0.730247  0.418309  2.687201  5.622059  0.209011
3   309_4  0.442107  0.617076  0.358626  2.466947  4.979503  0.222886
4   309_5  0.434940  0.617430  0.358802  2.365785  4.718679  0.213106

    pBRAF_N  pCAMKII_N   pCREB_N  …  pCASP9_N   PSD95_N    SNCA_N  \
0  0.177565   2.373744  0.232224  …  1.603310  2.014875  0.108234
1  0.172817   2.292150  0.226972  …  1.671738  2.004605  0.109749
2  0.175722   2.283337  0.230247  …  1.663550  2.016831  0.108196
3  0.176463   2.152301  0.207004  …  1.484624  1.957233  0.119883
4  0.173627   2.134014  0.192158  …  1.534835  2.009109  0.119524

   Ubiquitin_N  pGSK3B_Tyr216_N     SHH_N     pS6_N   pCFOS_N     SYP_N  \
0     1.044979         0.831557  0.188852  0.106305  0.108336  0.427099
1     1.009883         0.849270  0.200404  0.106592  0.104315  0.441581
2     0.996848         0.846709  0.193685  0.108303  0.106219  0.435777
3     0.990225         0.833277  0.192112  0.103184  0.111262  0.391691
4     0.997775         0.878668  0.205604  0.104784  0.110694  0.434154

     CaNA_N
0  1.675652
1  1.743610
2  1.926427
3  1.700563
4  1.839730

[5 rows x 73 columns]

Data3 Summary:
          DYRK1A_N      ITSN1_N       BDNF_N        NR1_N       NR2A_N  \
count  1080.000000  1080.000000  1080.000000  1080.000000  1080.000000
mean      0.426375     0.617999     0.319440     2.298994     3.849333
std       0.249248     0.251873     0.049764     0.348348     0.937420
min       0.145327     0.245359     0.115181     1.330831     1.737540
25%       0.288163     0.473669     0.287650     2.059152     3.160287
```

```
50%       0.366540     0.566365     0.316703     2.298688     3.763306
75%       0.488204     0.699722     0.349149     2.530500     4.447601
max       2.516367     2.602662     0.497160     3.757641     8.482553

            pAKT_N      pBRAF_N    pCAMKII_N      pCREB_N       pELK_N  …  \
count  1080.000000  1080.000000  1080.000000  1080.000000  1080.000000  …
mean      0.233172     0.181881     3.534840     0.212702     1.430251  …
std       0.041577     0.027013     1.294136     0.032633     0.467202  …
min       0.063236     0.064043     1.343998     0.112812     0.429032  …
25%       0.205821     0.164619     2.479194     0.190828     1.206389  …
50%       0.231246     0.182472     3.325505     0.210681     1.356368  …
75%       0.257225     0.197226     4.480652     0.234642     1.562668  …
max       0.539050     0.317066     7.464070     0.306247     6.113347  …

          pCASP9_N      PSD95_N       SNCA_N  Ubiquitin_N  pGSK3B_Tyr216_N  \
count  1080.000000  1080.000000  1080.000000  1080.000000      1080.000000
mean      1.548348     2.235236     0.159821     1.239270         0.848767
std       0.248132     0.254395     0.024150     0.173580         0.094311
min       0.853176     1.206098     0.101233     0.750664         0.577397
25%       1.375598     2.079338     0.142838     1.116262         0.793739
50%       1.522693     2.242197     0.157549     1.236586         0.849858
75%       1.713087     2.420226     0.173303     1.363079         0.916173
max       2.586216     2.877873     0.257616     1.897202         1.204598

            SHH_N        pS6_N      pCFOS_N        SYP_N       CaNA_N
count  1080.000000  1080.000000  1080.000000  1080.000000  1080.000000
mean      0.226676     0.121521     0.130566     0.446073     1.337784
std       0.028989     0.014276     0.023618     0.066432     0.317126
min       0.155869     0.067254     0.085419     0.258626     0.586479
25%       0.206395     0.110839     0.113357     0.398082     1.081423
50%       0.224000     0.121626     0.126152     0.448459     1.317441
75%       0.241655     0.131955     0.143306     0.490773     1.585824
max       0.358289     0.158748     0.256529     0.759588     2.129791

[8 rows x 72 columns]
Número total de valores nulos: 0
Número total de valores atípicos: 1411
```


Boxplots distribución de variables
8
6
4
2
0
DYRK1A_N
ITSN1_N
BDNF_N
NR1_N
NR2A_N
pAKT_N
pBRAF_N
pCAMKII_N
pCREB_N
pELK_N
pERK_N
pJNK_N
PKCA_N
pMEK_N
pNR1_N
pNR2A_N
pNR2B_N
pPKCAB_N
pRSK_N
AKT_N
BRAF_N
CAMKII_N
CREB_N
ELK_N
ERK_N
GSK3B_N
JNK_N
MEK_N
TRKA_N
RSK_N
APP_N
Bcatenin_N
SOD1_N
MTOR_N
P38_N
pMTOR_N
DSCR1_N
AMPKA_N
NR2B_N
pNUMB_N
RAPTOR_N
TIAM1_N
pP70S6_N
NUMB_N
P70S6_N
pGSK3B_N
pPKCG_N
CDK5_N
S6_N
ADARB1_N
AcetylH3K9_N
RRP1_N
BAX_N
ARC_N
ERBB4_N
nNOS_N
Tau_N
GFAP_N
GluR3_N
GluR4_N
IL1B_N
P3525_N
pCASP9_N
PSD95_N
SNCA_N
Ubiquitin_N
pGSK3B_Tyr216_N
SHH_N
pS6_N
pCFOS_N
SYP_N
CaNA_N



Nuestros datos de estudio presentan dos conjuntos de datos. Uno muestra los valores de 8 clases ('class3') distintas con 1080 muestras, las cuales hemos numerado del 0 al 7. Esto difiere en una unidad menos de las clases presentadas en el enunciado de la PEC2 para evitar problemas en pasos relacionados con el entrenamiento de la red neuronal. Las 8 clases son:

Código PEC2	Descripción	Código Ejercicio
1	c-CS-s: control-context-shock-salino	0
2	c-CS-m: control-context-shock-memantina	1
3	c-SC-s: control-shock-context-salino	2
4	c-SC-m: control-shock-context-memantina	3
5	t-CS-s: trisómico-context-shock-salino	4
6	t-CS-m: trisómico-context-shock-memantina	5
7	t-SC-s: trisómico-shock-context-salino	6
8	t-SC-m: trisómico-shock-context-memantina	7

El otro conjunto de datos ('data3') muestra las mediciones de los niveles de expresión de proteínas/modificaciones de proteínas que produjeron señales detectables en la fracción nuclear de la corteza. Tenemos 72 variables y 1080 muestras de estudio. Este conjunto de datos no presenta valores faltantes o 'Na', pero observamos valores atípicos en sus mediciones, con un total de 1411 valores que se desvían. Para nuestro modelo, vamos a mantener estos valores, ya que pueden ser representativos del caso de estudio que estamos tratando. Aun así, en el siguiente paso, y debido a la naturaleza de los datos, se va a realizar una normalización de los mismos; esto no elimina los valores atípicos, pero puede mitigar su efecto.

Por último, las figuras presentadas muestran la distribución de las variables y cuáles de ellas presentan valores atípicos. También se presenta una matriz de correlación para ver las relaciones que tienen unas variables con otras.

3 Normalizar las expresiones con la transformación minmax.

[9]:
```
data3.drop('MouseID', axis = 1, inplace = True) # Eliminamos la columna␣
 ↪"MouseID", ya que no es necesaria para implementar el modelo.

norm = MinMaxScaler() # Establecemos la función MinMax.
data3_norm = pd.DataFrame(norm.fit_transform(data3), columns=data3.columns) #␣
 ↪Normalizamos los datos del conjunto "data3". Usamos la función␣
 ↪"MinMaxScaler" de la librería "sklearn.preprocessing". Con esto, todas las␣
 ↪variables están en un rango de valor de 0 a 1.

data_model_1 = pd.concat([class3, data3_norm], axis=1) # Combinamos ambos␣
 ↪conjuntos de datos para preparar los conjuntos de entrenemiento/prueba.
display(data_model_1.head()) # Comprobamos las primeras entradas del nuevo␣
 ↪conjunto para revisar la normalización y las clases.
```

	Class	DYRK1A_N	ITSN1_N	BDNF_N	NR1_N	NR2A_N	pAKT_N	\
0	1	0.151122	0.212885	0.824638	0.612119	0.630482	0.327006	
1	1	0.155750	0.188226	0.776455	0.601070	0.585247	0.311887	
2	1	0.153459	0.205696	0.793572	0.558911	0.575910	0.306369	
3	1	0.125169	0.157688	0.637326	0.468152	0.480646	0.335530	
4	1	0.122146	0.157838	0.637787	0.426467	0.441977	0.314976	

	pBRAF_N	pCAMKII_N	pCREB_N	…	pCASP9_N	PSD95_N	SNCA_N	\
0	0.448666	0.168257	0.617322	…	0.432843	0.483783	0.044770	
1	0.429899	0.154925	0.590173	…	0.472327	0.477640	0.054452	
2	0.441381	0.153485	0.607102	…	0.467603	0.484953	0.044526	
3	0.444307	0.132074	0.486945	…	0.364359	0.449304	0.119259	
4	0.433100	0.129086	0.410194	…	0.393332	0.480334	0.116965	

	Ubiquitin_N	pGSK3B_Tyr216_N	SHH_N	pS6_N	pCFOS_N	SYP_N	\
0	0.256699	0.405228	0.162941	0.426816	0.133930	0.336299	
1	0.226088	0.433471	0.220010	0.429952	0.110434	0.365208	
2	0.214719	0.429387	0.186816	0.448652	0.121560	0.353621	

```
3     0.208943          0.407971  0.179047  0.392700  0.151031  0.265619
4     0.215528          0.480342  0.245702  0.410187  0.147711  0.350381

     CaNA_N
0  0.705738
1  0.749771
2  0.868229
3  0.721879
4  0.812053

[5 rows x 73 columns]
```

4 Separar los datos en train (2/3) y test (1/3).

[11]:
```
# Establecemos semillas para reproducibilidad
random.seed(123)
np.random.seed(123)
tf.random.set_seed(123)

X = data_model_1.drop('Class', axis = 1) # Conjunto que contiene todas las␣
 ↪variables de interés para el modelo.
y = data_model_1['Class'] # Preparamos las etiquetas del modelo.

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 1/3,␣
 ↪random_state = 0) # Dividimos los datos en conjuntos de entrenamiento y␣
 ↪prueba. El conjunto de entrenamiento "X_train, y_train" contiene el 2/3 de␣
 ↪los datos, mientras que el conjunto de prueba "X_test, y_test" contiene el 1/
 ↪3.

# Mostramos el número de muestras de la separación:
print("Número de muestras de X_train:"  + str(len(X_train)))
print("Número de muestras de X_test:"  + str(len(X_test)))
```

```
Número de muestras de X_train:720
Número de muestras de X_test:360
```

5 Definir el modelo 1, que consiste en una red neuronal con una capa oculta densa de 35 nodos, con activación relu. Añadir un 20% de dropout. Proporcionar el summary del modelo y justificar el total de parámetros de cada capa.

[13]:
```
# Establecemos semillas para reproducibilidad
random.seed(123)
np.random.seed(123)
tf.random.set_seed(123)
```

```
# Definimos el modelo de red neuronal usando la biblioteca "Keras":
model_1 = models.Sequential() # Creamos un modelo secuencial en "Keras" para␣
 ↪preparar una arquitectura de red neuronal aplicando capas de forma␣
 ↪secuencial.
model_1.add(layers.Dense(35, activation = 'relu', input_shape = (X_train.
 ↪shape[1],))) # Agregamos una capa densa al modelo con 35 nodos. Establecemos␣
 ↪la activacion 'relu' para introducir no linealidades en el modelo.
model_1.add(layers.Dropout(0.2)) # Agregamos una capa de Dropout desactivando␣
 ↪aleatoriamente el 20% de las neuronas en cada paso de entrenamiento para␣
 ↪prevenir el sobreajuste.
model_1.add(layers.Dense(8, activation = 'softmax')) # Para la capa de salida,␣
 ↪establecemos la función de activación 'softmax' y una capa densa con 8␣
 ↪salidas, correspondientes a las clases iniciales.
model_1.summary() # Mostramos el resumen del modelo. Incluye información sobre␣
 ↪la arquitectura de la red neuronal, el número de parámetros entrenables en␣
 ↪cada capa, y el número total de parámetros en el modelo.
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 35)                2555

 dropout (Dropout)           (None, 35)                0

 dense_1 (Dense)             (None, 8)                 288

=================================================================
Total params: 2843 (11.11 KB)
Trainable params: 2843 (11.11 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

El resumen ('summary') del modelo 'model_1' nos muestra que tenemos un total de 2843 parámetros entrenables. La arquitectura del modelo presenta dos capas densas y una capa de 'Dropout'. El número total de parámetros se puede justificar de la siguiente forma:

- De la primera capa 'dense (Dense)', tenemos 35 nodos de salida conectados a cada variable ((X_train.shape[1],)=72), más 35 sesgos. Es igual a: 35 nodos $\times$ (X_train.shape[1],) $= 72 + 35 = 2555$ parámetros.
- De la capa 'dropout (Dropout)', no tenemos parámetros entrenables, lo que da como resultado 0 parámetros.
- De la última capa 'dense_1 (Dense)', obtenemos 8 nodos de salida (clases), el número de nodos de la capa anterior es 35, más 8 sesgos. Igual a 288 parámetros entrenables.

6 Ajustar el modelo 1 con un 20% de validación, mostrando la curva de aprendizaje de entrenamiento y validación con 50 épocas.

[16]:
```
# Establecemos semillas para reproducibilidad
random.seed(123)
np.random.seed(123)
tf.random.set_seed(123)

# Compilamos el model_1:
model_1.compile(optimizer = 'adam', # Establecemos el optimizador 'adam' de␣
 ↪forma predeterminada.
                loss = 'sparse_categorical_crossentropy', # Al manejar␣
 ↪etiquetas con valores enteros usamos 'sparse_categorical_crossentropy' como␣
 ↪fúncion de perdida.
                metrics = ['accuracy'] # Usamos 'accuracy' como metrica para␣
 ↪evaluar el rendimiento del modelo durante el entrenamiento.
)

# Entrenamos el model_1 y almacenamos la informacion en la variable "history":
history = model_1.fit(
    X_train, y_train,
    epochs = 50, # Establecemos las épocas de interacciones del entrenamiento a␣
 ↪'50'.
    validation_split = 0.2,  # 20% de los datos se usarán como datos de␣
 ↪validación.
    verbose = 2 # Nivel de detalle durante el entrenamiento.
)

# Curvas de aprendizaje de entrenamiento y validación:

# Estraemos las métricas de 'history' para la visualizacion del modelo:
train_loss = history.history['loss']
val_loss = history.history['val_loss']
train_acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
epochs = range(1, len(train_loss) + 1)

plt.figure(figsize = (12, 5))

# Gráfico pérdida:
plt.subplot(1, 2, 1)
plt.plot(epochs, train_loss, 'r', label = 'Pérdida de entrenamiento')
plt.plot(epochs, val_loss, 'b', label = 'Pérdida de validación')
plt.title('Entrenamiento y Validación: Pérdida (Model_1)')
plt.xlabel('Épocas')
```

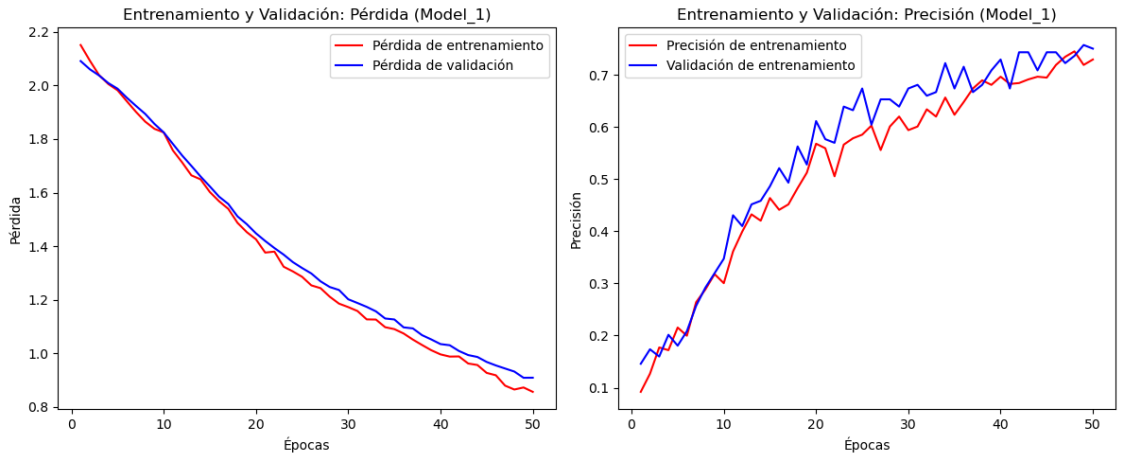
```
plt.ylabel('Pérdida')
plt.legend()
# Gráfico precisión:
plt.subplot(1, 2, 2)
plt.plot(epochs, train_acc, 'r', label = 'Precisión de entrenamiento')
plt.plot(epochs, val_acc, 'b', label = 'Validación de entrenamiento')
plt.title('Entrenamiento y Validación: Precisión (Model_1)')
plt.xlabel('Épocas')
plt.ylabel('Precisión')
plt.legend()

plt.tight_layout()
plt.show()
```

```
Epoch 1/50
18/18 - 4s - loss: 2.1507 - accuracy: 0.0920 - val_loss: 2.0907 - val_accuracy:
0.1458 - 4s/epoch - 204ms/step
Epoch 2/50
18/18 - 0s - loss: 2.0921 - accuracy: 0.1267 - val_loss: 2.0600 - val_accuracy:
0.1736 - 305ms/epoch - 17ms/step
Epoch 3/50
18/18 - 0s - loss: 2.0381 - accuracy: 0.1771 - val_loss: 2.0370 - val_accuracy:
0.1597 - 288ms/epoch - 16ms/step
Epoch 4/50
18/18 - 0s - loss: 2.0050 - accuracy: 0.1719 - val_loss: 2.0086 - val_accuracy:
0.2014 - 295ms/epoch - 16ms/step
Epoch 5/50
18/18 - 0s - loss: 1.9818 - accuracy: 0.2153 - val_loss: 1.9878 - val_accuracy:
0.1806 - 292ms/epoch - 16ms/step
Epoch 6/50
18/18 - 0s - loss: 1.9410 - accuracy: 0.1997 - val_loss: 1.9542 - val_accuracy:
0.2083 - 206ms/epoch - 11ms/step
Epoch 7/50
18/18 - 0s - loss: 1.9010 - accuracy: 0.2639 - val_loss: 1.9233 - val_accuracy:
0.2569 - 209ms/epoch - 12ms/step
Epoch 8/50
18/18 - 0s - loss: 1.8647 - accuracy: 0.2882 - val_loss: 1.8935 - val_accuracy:
0.2917 - 283ms/epoch - 16ms/step
Epoch 9/50
18/18 - 0s - loss: 1.8379 - accuracy: 0.3177 - val_loss: 1.8563 - val_accuracy:
0.3194 - 206ms/epoch - 11ms/step
Epoch 10/50
18/18 - 0s - loss: 1.8246 - accuracy: 0.3003 - val_loss: 1.8247 - val_accuracy:
0.3472 - 290ms/epoch - 16ms/step
Epoch 11/50
18/18 - 0s - loss: 1.7572 - accuracy: 0.3611 - val_loss: 1.7814 - val_accuracy:
0.4306 - 212ms/epoch - 12ms/step
```

```
Epoch 12/50
18/18 - 0s - loss: 1.7126 - accuracy: 0.3993 - val_loss: 1.7384 - val_accuracy:
0.4097 - 288ms/epoch - 16ms/step
Epoch 13/50
18/18 - 0s - loss: 1.6643 - accuracy: 0.4323 - val_loss: 1.6997 - val_accuracy:
0.4514 - 200ms/epoch - 11ms/step
Epoch 14/50
18/18 - 0s - loss: 1.6494 - accuracy: 0.4201 - val_loss: 1.6600 - val_accuracy:
0.4583 - 210ms/epoch - 12ms/step
Epoch 15/50
18/18 - 0s - loss: 1.6022 - accuracy: 0.4635 - val_loss: 1.6231 - val_accuracy:
0.4861 - 205ms/epoch - 11ms/step
Epoch 16/50
18/18 - 0s - loss: 1.5679 - accuracy: 0.4410 - val_loss: 1.5849 - val_accuracy:
0.5208 - 300ms/epoch - 17ms/step
Epoch 17/50
18/18 - 0s - loss: 1.5394 - accuracy: 0.4514 - val_loss: 1.5575 - val_accuracy:
0.4931 - 497ms/epoch - 28ms/step
Epoch 18/50
18/18 - 0s - loss: 1.4872 - accuracy: 0.4826 - val_loss: 1.5113 - val_accuracy:
0.5625 - 399ms/epoch - 22ms/step
Epoch 19/50
18/18 - 0s - loss: 1.4527 - accuracy: 0.5122 - val_loss: 1.4824 - val_accuracy:
0.5278 - 499ms/epoch - 28ms/step
Epoch 20/50
18/18 - 0s - loss: 1.4258 - accuracy: 0.5677 - val_loss: 1.4481 - val_accuracy:
0.6111 - 294ms/epoch - 16ms/step
Epoch 21/50
18/18 - 0s - loss: 1.3762 - accuracy: 0.5590 - val_loss: 1.4195 - val_accuracy:
0.5764 - 211ms/epoch - 12ms/step
Epoch 22/50
18/18 - 0s - loss: 1.3798 - accuracy: 0.5052 - val_loss: 1.3931 - val_accuracy:
0.5694 - 286ms/epoch - 16ms/step
Epoch 23/50
18/18 - 0s - loss: 1.3234 - accuracy: 0.5660 - val_loss: 1.3686 - val_accuracy:
0.6389 - 205ms/epoch - 11ms/step
Epoch 24/50
18/18 - 0s - loss: 1.3059 - accuracy: 0.5781 - val_loss: 1.3406 - val_accuracy:
0.6319 - 286ms/epoch - 16ms/step
Epoch 25/50
18/18 - 0s - loss: 1.2858 - accuracy: 0.5851 - val_loss: 1.3186 - val_accuracy:
0.6736 - 204ms/epoch - 11ms/step
Epoch 26/50
18/18 - 0s - loss: 1.2539 - accuracy: 0.6024 - val_loss: 1.2983 - val_accuracy:
0.6042 - 210ms/epoch - 12ms/step
Epoch 27/50
18/18 - 0s - loss: 1.2435 - accuracy: 0.5556 - val_loss: 1.2689 - val_accuracy:
0.6528 - 296ms/epoch - 16ms/step
```

```
Epoch 28/50
18/18 - 0s - loss: 1.2114 - accuracy: 0.6007 - val_loss: 1.2477 - val_accuracy:
0.6528 - 423ms/epoch - 23ms/step
Epoch 29/50
18/18 - 0s - loss: 1.1857 - accuracy: 0.6198 - val_loss: 1.2367 - val_accuracy:
0.6389 - 268ms/epoch - 15ms/step
Epoch 30/50
18/18 - 0s - loss: 1.1728 - accuracy: 0.5938 - val_loss: 1.2020 - val_accuracy:
0.6736 - 288ms/epoch - 16ms/step
Epoch 31/50
18/18 - 0s - loss: 1.1582 - accuracy: 0.6007 - val_loss: 1.1881 - val_accuracy:
0.6806 - 206ms/epoch - 11ms/step
Epoch 32/50
18/18 - 0s - loss: 1.1270 - accuracy: 0.6337 - val_loss: 1.1737 - val_accuracy:
0.6597 - 207ms/epoch - 11ms/step
Epoch 33/50
18/18 - 0s - loss: 1.1264 - accuracy: 0.6198 - val_loss: 1.1569 - val_accuracy:
0.6667 - 282ms/epoch - 16ms/step
Epoch 34/50
18/18 - 0s - loss: 1.0978 - accuracy: 0.6562 - val_loss: 1.1303 - val_accuracy:
0.7222 - 204ms/epoch - 11ms/step
Epoch 35/50
18/18 - 0s - loss: 1.0906 - accuracy: 0.6233 - val_loss: 1.1267 - val_accuracy:
0.6736 - 205ms/epoch - 11ms/step
Epoch 36/50
18/18 - 0s - loss: 1.0743 - accuracy: 0.6476 - val_loss: 1.0972 - val_accuracy:
0.7153 - 286ms/epoch - 16ms/step
Epoch 37/50
18/18 - 0s - loss: 1.0517 - accuracy: 0.6736 - val_loss: 1.0933 - val_accuracy:
0.6667 - 207ms/epoch - 12ms/step
Epoch 38/50
18/18 - 0s - loss: 1.0314 - accuracy: 0.6892 - val_loss: 1.0681 - val_accuracy:
0.6806 - 208ms/epoch - 12ms/step
Epoch 39/50
18/18 - 0s - loss: 1.0118 - accuracy: 0.6806 - val_loss: 1.0522 - val_accuracy:
0.7083 - 282ms/epoch - 16ms/step
Epoch 40/50
18/18 - 0s - loss: 0.9964 - accuracy: 0.6962 - val_loss: 1.0348 - val_accuracy:
0.7292 - 209ms/epoch - 12ms/step
Epoch 41/50
18/18 - 0s - loss: 0.9882 - accuracy: 0.6823 - val_loss: 1.0305 - val_accuracy:
0.6736 - 285ms/epoch - 16ms/step
Epoch 42/50
18/18 - 0s - loss: 0.9887 - accuracy: 0.6840 - val_loss: 1.0091 - val_accuracy:
0.7431 - 207ms/epoch - 12ms/step
Epoch 43/50
18/18 - 0s - loss: 0.9624 - accuracy: 0.6910 - val_loss: 0.9939 - val_accuracy:
0.7431 - 202ms/epoch - 11ms/step
```

```
Epoch 44/50
18/18 - 0s - loss: 0.9567 - accuracy: 0.6962 - val_loss: 0.9867 - val_accuracy:
0.7083 - 286ms/epoch - 16ms/step
Epoch 45/50
18/18 - 0s - loss: 0.9273 - accuracy: 0.6944 - val_loss: 0.9683 - val_accuracy:
0.7431 - 205ms/epoch - 11ms/step
Epoch 46/50
18/18 - 0s - loss: 0.9184 - accuracy: 0.7188 - val_loss: 0.9551 - val_accuracy:
0.7431 - 203ms/epoch - 11ms/step
Epoch 47/50
18/18 - 0s - loss: 0.8798 - accuracy: 0.7344 - val_loss: 0.9436 - val_accuracy:
0.7222 - 208ms/epoch - 12ms/step
Epoch 48/50
18/18 - 0s - loss: 0.8653 - accuracy: 0.7448 - val_loss: 0.9324 - val_accuracy:
0.7361 - 282ms/epoch - 16ms/step
Epoch 49/50
18/18 - 0s - loss: 0.8729 - accuracy: 0.7188 - val_loss: 0.9091 - val_accuracy:
0.7569 - 205ms/epoch - 11ms/step
Epoch 50/50
18/18 - 0s - loss: 0.8566 - accuracy: 0.7292 - val_loss: 0.9094 - val_accuracy:
0.7500 - 286ms/epoch - 16ms/step
```

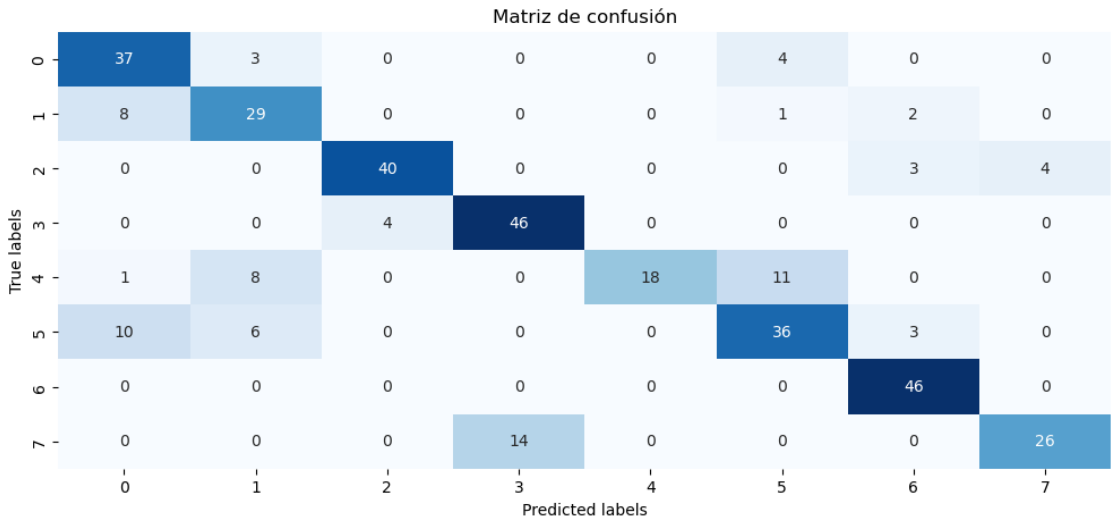


7 Obtener la tabla de clasificación errónea en test. Y las métricas usuales de evaluación.

[18]:
```
# Establecemos semillas para reproducibilidad
random.seed(123)
np.random.seed(123)
tf.random.set_seed(123)
```

```
# Predecimos las probabilidades de las clases del conjunto de prueba:
y_pred_probabilities = model_1.predict(X_test)

# Convertimos las probabilidades en clases (etiquetas):
y_pred_classes = np.argmax(y_pred_probabilities, axis=1)

# Calculamos la matriz de confusión:
confusion_mtx = confusion_matrix(y_test, y_pred_classes)

# Visualizamos la matriz de confusión
plt.figure(figsize = (12, 5))
sns.heatmap(confusion_mtx, annot = True, fmt = 'd', cmap = 'Blues', cbar =␣
 ↪False)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Matriz de confusión')
plt.show()

# Mostramos el informe de clasificación
classification_report_str = classification_report(y_test, y_pred_classes)
print("Informe de clasificación:\n", classification_report_str)
# Calculamos y mostramos la precisión general en el conjunto de prueba:
accuracy = accuracy_score(y_test, y_pred_classes)
print(f'Precisión en el conjunto de prueba: {accuracy * 100:.2f}%')
```

```
12/12 [==============================] - 0s 944us/step
```




```
Informe de clasificación:
              precision    recall  f1-score   support
```

```
              0       0.66      0.84      0.74        44
              1       0.63      0.72      0.67        40
              2       0.91      0.85      0.88        47
              3       0.77      0.92      0.84        50
              4       1.00      0.47      0.64        38
              5       0.69      0.65      0.67        55
              6       0.85      1.00      0.92        46
              7       0.87      0.65      0.74        40

    accuracy                           0.77       360
   macro avg       0.80      0.76      0.76       360
weighted avg       0.79      0.77      0.77       360
```

```
Precisión en el conjunto de prueba: 77.22%
```

Con la matriz de confusión (tabla de clasificación errónea) y el informe de clasificación, obtenemos las métricas del modelo para el conjunto de prueba. El modelo 'model_1' tiene una precisión global del 77,22%, lo que significa que solo el 22,78% de las muestras del conjunto de pruebas han sido clasificadas incorrectamente.

En general, encontramos un buen ajuste y parámetros para el modelo 'model_1'. Para la mayoría de las clases, observamos buenos números que suelen estar por encima del 80% para precisión, recall y F1-score. Aunque algunas clases, como la clase 0, 1 y 5 tienen una precisión menor al 80%, y la clase 4 y 5 muestran un recall de 47% y 65%, respectivamente. Por último, la clase 0, 4, 5 y 7 tienen un valor de F1-score menor al 80%. En otras palabras, la clase 5 parece ser la que peor se ajusta al modelo en comparación con todas las demás.

Cabe destacar que para interpretar las clases debemos sumarles una unidad para obtener su valor o etiqueta real, según el enunciado de la PEC. La clase 5 de nuestro modelo hace referencia a la clase 6: 't-CS-m: trisómico-context-shock-memantina'.

8 Definir el modelo 2, que consiste en una red neuronal con dos capas ocultas densas de 35 nodos y 15 nodos, con activación relu. Añadir un 20% de dropout en ambas capas. Proporcionar el summary del modelo y justificar el total de parámetros de cada capa.

[21]:
```
# Establecemos semillas para reproducibilidad
random.seed(123)
np.random.seed(123)
tf.random.set_seed(123)
# Definimos el modelo de red neuronal como en el apartado 5, en este caso␣
 ↪añadimos una capa adicional:
model_2 = models.Sequential()
model_2.add(layers.Dense(35, activation = 'relu', input_shape = (X_train.
 ↪shape[1],)))
model_2.add(layers.Dropout(0.2))
```

```
model_2.add(layers.Dense(15, activation = 'relu')) # Para este modelo se ha␣
 ↪añadido otra capa densa de 15 nodos.
model_2.add(layers.Dropout(0.2))
model_2.add(layers.Dense(8, activation = 'softmax'))
model_2.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_2 (Dense)             (None, 35)                2555

 dropout_1 (Dropout)         (None, 35)                0

 dense_3 (Dense)             (None, 15)                540

 dropout_2 (Dropout)         (None, 15)                0

 dense_4 (Dense)             (None, 8)                 128

=================================================================
Total params: 3223 (12.59 KB)
Trainable params: 3223 (12.59 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Al igual que en el modelo anterior, el resumen del model_2 nos muestra 3223 parámetros en total, de los cuales 3223 son parámetros entrenables. Al justificar, como en el apartado anterior, obtenemos:

- Capa 'dense_2 (Dense)': 35 nodos de salida $\times$ 72 variables + 35 sesgos = 2555 parámetros.
- Capa 'dropout_1 (Dropout)': Sin parámetros entrenables.
- Capa 'dense_3 (Dense)', capa añadida para el model_2: 15 nodos de salida $\times$ 35 nodos de la capa anterior + 15 sesgos = 540 parámetros.
- Capa 'dropout_2 (Dropout)': Sin parámetros entrenables.
- Capa 'dense_4 (Dense)': 8 nodos de salida (clases) $\times$ 15 nodos anteriores +8 sesgos = 128 parámetros.

9 Ajustar el modelo 2 con un 20% de validación, mostrando la curva de aprendizaje de entrenamiento y validación con 50 épocas.

[24]:
```
# Establecemos semillas para reproducibilidad
random.seed(123)
np.random.seed(123)
tf.random.set_seed(123)

# Volvemos a realizar los pasos del apartado 6, en este caso usando el model_2:
```

```
# Compilamos el model_2:
model_2.compile(optimizer = 'adam', loss = 'sparse_categorical_crossentropy',␣
 ↪metrics = ['accuracy'])

# Entrenamos el model_2 y almacenamos la informacion en la variable "history_2":
 ↪
history_2 = model_2.fit(
    X_train, y_train,
    epochs = 50,
    validation_split = 0.2,
    verbose = 2
)

# Estraemos las métricas de 'history_2' para la visualizacion del modelo:
train_loss_2 = history_2.history['loss']
val_loss_2 = history_2.history['val_loss']
train_acc_2 = history_2.history['accuracy']
val_acc_2 = history_2.history['val_accuracy']
epochs_2 = range(1, len(train_loss_2) + 1)

plt.figure(figsize = (12, 5))

# Gráfico pérdida:
plt.subplot(1, 2, 1)
plt.plot(epochs_2, train_loss_2, 'r', label = 'Pérdida de entrenamiento')
plt.plot(epochs_2, val_loss_2, 'b', label = 'Pérdida de validación')
plt.title('Entrenamiento y Validación: Pérdida (Model_2)')
plt.xlabel('Épocas')
plt.ylabel('Pérdida')
plt.legend()
# Gráfico precisión:
plt.subplot(1, 2, 2)
plt.plot(epochs_2, train_acc_2, 'r', label = 'Precisión de entrenamiento')
plt.plot(epochs_2, val_acc_2, 'b', label = 'Validación de entrenamiento')
plt.title('Entrenamiento y Validación: Precisión (Model_2)')
plt.xlabel('Épocas')
plt.ylabel('Precisión')
plt.legend()

plt.tight_layout()
plt.show()
```

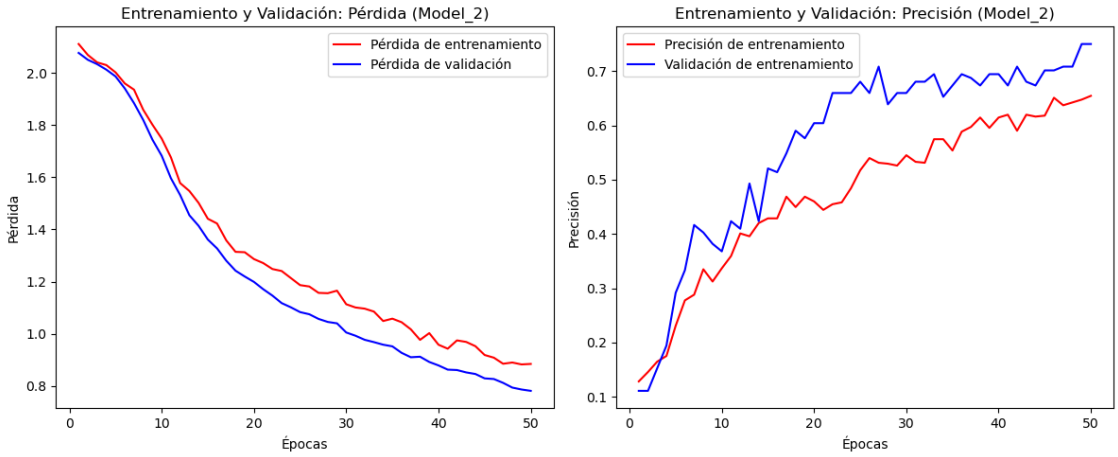
```
Epoch 1/50
18/18 - 3s - loss: 2.1101 - accuracy: 0.1285 - val_loss: 2.0761 - val_accuracy:
0.1111 - 3s/epoch - 156ms/step
Epoch 2/50
```

```
18/18 - 0s - loss: 2.0681 - accuracy: 0.1458 - val_loss: 2.0496 - val_accuracy:
0.1111 - 299ms/epoch - 17ms/step
Epoch 3/50
18/18 - 0s - loss: 2.0398 - accuracy: 0.1649 - val_loss: 2.0330 - val_accuracy:
0.1528 - 293ms/epoch - 16ms/step
Epoch 4/50
18/18 - 0s - loss: 2.0295 - accuracy: 0.1753 - val_loss: 2.0120 - val_accuracy:
0.1944 - 216ms/epoch - 12ms/step
Epoch 5/50
18/18 - 0s - loss: 2.0014 - accuracy: 0.2309 - val_loss: 1.9860 - val_accuracy:
0.2917 - 293ms/epoch - 16ms/step
Epoch 6/50
18/18 - 0s - loss: 1.9592 - accuracy: 0.2778 - val_loss: 1.9400 - val_accuracy:
0.3333 - 296ms/epoch - 16ms/step
Epoch 7/50
18/18 - 0s - loss: 1.9353 - accuracy: 0.2882 - val_loss: 1.8837 - val_accuracy:
0.4167 - 294ms/epoch - 16ms/step
Epoch 8/50
18/18 - 0s - loss: 1.8582 - accuracy: 0.3351 - val_loss: 1.8192 - val_accuracy:
0.4028 - 291ms/epoch - 16ms/step
Epoch 9/50
18/18 - 0s - loss: 1.8010 - accuracy: 0.3125 - val_loss: 1.7437 - val_accuracy:
0.3819 - 207ms/epoch - 12ms/step
Epoch 10/50
18/18 - 0s - loss: 1.7480 - accuracy: 0.3368 - val_loss: 1.6822 - val_accuracy:
0.3681 - 289ms/epoch - 16ms/step
Epoch 11/50
18/18 - 0s - loss: 1.6761 - accuracy: 0.3594 - val_loss: 1.5960 - val_accuracy:
0.4236 - 397ms/epoch - 22ms/step
Epoch 12/50
18/18 - 1s - loss: 1.5776 - accuracy: 0.4010 - val_loss: 1.5318 - val_accuracy:
0.4097 - 601ms/epoch - 33ms/step
Epoch 13/50
18/18 - 1s - loss: 1.5472 - accuracy: 0.3958 - val_loss: 1.4543 - val_accuracy:
0.4931 - 600ms/epoch - 33ms/step
Epoch 14/50
18/18 - 1s - loss: 1.5016 - accuracy: 0.4201 - val_loss: 1.4138 - val_accuracy:
0.4236 - 692ms/epoch - 38ms/step
Epoch 15/50
18/18 - 1s - loss: 1.4410 - accuracy: 0.4288 - val_loss: 1.3613 - val_accuracy:
0.5208 - 509ms/epoch - 28ms/step
Epoch 16/50
18/18 - 0s - loss: 1.4227 - accuracy: 0.4288 - val_loss: 1.3273 - val_accuracy:
0.5139 - 500ms/epoch - 28ms/step
Epoch 17/50
18/18 - 0s - loss: 1.3580 - accuracy: 0.4688 - val_loss: 1.2802 - val_accuracy:
0.5486 - 408ms/epoch - 23ms/step
Epoch 18/50
```

```
18/18 - 1s - loss: 1.3142 - accuracy: 0.4497 - val_loss: 1.2421 - val_accuracy:
0.5903 - 607ms/epoch - 34ms/step
Epoch 19/50
18/18 - 0s - loss: 1.3127 - accuracy: 0.4688 - val_loss: 1.2198 - val_accuracy:
0.5764 - 407ms/epoch - 23ms/step
Epoch 20/50
18/18 - 1s - loss: 1.2864 - accuracy: 0.4601 - val_loss: 1.1990 - val_accuracy:
0.6042 - 595ms/epoch - 33ms/step
Epoch 21/50
18/18 - 1s - loss: 1.2712 - accuracy: 0.4444 - val_loss: 1.1712 - val_accuracy:
0.6042 - 500ms/epoch - 28ms/step
Epoch 22/50
18/18 - 0s - loss: 1.2484 - accuracy: 0.4549 - val_loss: 1.1466 - val_accuracy:
0.6597 - 495ms/epoch - 28ms/step
Epoch 23/50
18/18 - 0s - loss: 1.2408 - accuracy: 0.4583 - val_loss: 1.1179 - val_accuracy:
0.6597 - 402ms/epoch - 22ms/step
Epoch 24/50
18/18 - 0s - loss: 1.2138 - accuracy: 0.4844 - val_loss: 1.1018 - val_accuracy:
0.6597 - 406ms/epoch - 23ms/step
Epoch 25/50
18/18 - 0s - loss: 1.1867 - accuracy: 0.5174 - val_loss: 1.0834 - val_accuracy:
0.6806 - 497ms/epoch - 28ms/step
Epoch 26/50
18/18 - 1s - loss: 1.1815 - accuracy: 0.5399 - val_loss: 1.0752 - val_accuracy:
0.6597 - 504ms/epoch - 28ms/step
Epoch 27/50
18/18 - 0s - loss: 1.1571 - accuracy: 0.5312 - val_loss: 1.0574 - val_accuracy:
0.7083 - 302ms/epoch - 17ms/step
Epoch 28/50
18/18 - 0s - loss: 1.1558 - accuracy: 0.5295 - val_loss: 1.0458 - val_accuracy:
0.6389 - 291ms/epoch - 16ms/step
Epoch 29/50
18/18 - 0s - loss: 1.1655 - accuracy: 0.5260 - val_loss: 1.0404 - val_accuracy:
0.6597 - 212ms/epoch - 12ms/step
Epoch 30/50
18/18 - 0s - loss: 1.1135 - accuracy: 0.5451 - val_loss: 1.0054 - val_accuracy:
0.6597 - 487ms/epoch - 27ms/step
Epoch 31/50
18/18 - 1s - loss: 1.1012 - accuracy: 0.5330 - val_loss: 0.9930 - val_accuracy:
0.6806 - 500ms/epoch - 28ms/step
Epoch 32/50
18/18 - 1s - loss: 1.0967 - accuracy: 0.5312 - val_loss: 0.9774 - val_accuracy:
0.6806 - 503ms/epoch - 28ms/step
Epoch 33/50
18/18 - 0s - loss: 1.0854 - accuracy: 0.5747 - val_loss: 0.9685 - val_accuracy:
0.6944 - 495ms/epoch - 27ms/step
Epoch 34/50
```

```
18/18 - 0s - loss: 1.0493 - accuracy: 0.5747 - val_loss: 0.9587 - val_accuracy:
0.6528 - 400ms/epoch - 22ms/step
Epoch 35/50
18/18 - 0s - loss: 1.0583 - accuracy: 0.5538 - val_loss: 0.9522 - val_accuracy:
0.6736 - 206ms/epoch - 11ms/step
Epoch 36/50
18/18 - 0s - loss: 1.0447 - accuracy: 0.5885 - val_loss: 0.9274 - val_accuracy:
0.6944 - 292ms/epoch - 16ms/step
Epoch 37/50
18/18 - 0s - loss: 1.0175 - accuracy: 0.5972 - val_loss: 0.9105 - val_accuracy:
0.6875 - 210ms/epoch - 12ms/step
Epoch 38/50
18/18 - 0s - loss: 0.9771 - accuracy: 0.6146 - val_loss: 0.9126 - val_accuracy:
0.6736 - 287ms/epoch - 16ms/step
Epoch 39/50
18/18 - 0s - loss: 1.0030 - accuracy: 0.5955 - val_loss: 0.8925 - val_accuracy:
0.6944 - 207ms/epoch - 11ms/step
Epoch 40/50
18/18 - 0s - loss: 0.9585 - accuracy: 0.6146 - val_loss: 0.8792 - val_accuracy:
0.6944 - 288ms/epoch - 16ms/step
Epoch 41/50
18/18 - 0s - loss: 0.9433 - accuracy: 0.6198 - val_loss: 0.8631 - val_accuracy:
0.6736 - 213ms/epoch - 12ms/step
Epoch 42/50
18/18 - 0s - loss: 0.9751 - accuracy: 0.5903 - val_loss: 0.8615 - val_accuracy:
0.7083 - 213ms/epoch - 12ms/step
Epoch 43/50
18/18 - 0s - loss: 0.9692 - accuracy: 0.6198 - val_loss: 0.8525 - val_accuracy:
0.6806 - 291ms/epoch - 16ms/step
Epoch 44/50
18/18 - 0s - loss: 0.9529 - accuracy: 0.6163 - val_loss: 0.8464 - val_accuracy:
0.6736 - 210ms/epoch - 12ms/step
Epoch 45/50
18/18 - 0s - loss: 0.9192 - accuracy: 0.6181 - val_loss: 0.8297 - val_accuracy:
0.7014 - 286ms/epoch - 16ms/step
Epoch 46/50
18/18 - 0s - loss: 0.9089 - accuracy: 0.6510 - val_loss: 0.8271 - val_accuracy:
0.7014 - 214ms/epoch - 12ms/step
Epoch 47/50
18/18 - 0s - loss: 0.8857 - accuracy: 0.6372 - val_loss: 0.8125 - val_accuracy:
0.7083 - 205ms/epoch - 11ms/step
Epoch 48/50
18/18 - 0s - loss: 0.8903 - accuracy: 0.6424 - val_loss: 0.7944 - val_accuracy:
0.7083 - 209ms/epoch - 12ms/step
Epoch 49/50
18/18 - 0s - loss: 0.8831 - accuracy: 0.6476 - val_loss: 0.7870 - val_accuracy:
0.7500 - 288ms/epoch - 16ms/step
Epoch 50/50
```

```
18/18 - 0s - loss: 0.8850 - accuracy: 0.6545 - val_loss: 0.7819 - val_accuracy:
0.7500 - 207ms/epoch - 11ms/step
```



10 Comparar en test, mediante las métricas de evaluación, los dos modelos.

[26]:
```python
# Métricas del modelo 1 en el conjunto de prueba:
eval_model_1 = model_1.evaluate(X_test, y_test)
loss_model_1, accuracy_model_1 = eval_model_1[0], eval_model_1[1]

# Métricas del modelo 2 en el conjunto de prueba:
eval_model_2 = model_2.evaluate(X_test, y_test)
loss_model_2, accuracy_model_2 = eval_model_2[0], eval_model_2[1]

# Imprimimos las métricas de evaluación
print("Métricas del Modelo 1 en el Conjunto de Prueba:")
print(f"Pérdida: {round(loss_model_1 * 100, 2)}%")
print(f"Precisión: {round(accuracy_model_1 * 100, 2)}%")

print("\nMétricas del Modelo 2 en el Conjunto de Prueba:")
print(f"Pérdida: {round(loss_model_2 * 100, 2)}%")
print(f"Precisión: {round(accuracy_model_2 * 100, 2)}%")

# Comparamos métricas adicionales:
y_pred_classes_1 = np.argmax(model_1.predict(X_test), axis=1)
y_pred_classes_2 = np.argmax(model_2.predict(X_test), axis=1)

# Matriz de confusión Model_1
confusion_mtx_1 = confusion_matrix(y_test, y_pred_classes_1)
print("\nMatriz de confusión del Modelo 1:")
```

```r
print(confusion_mtx_1)
print("\nOtras métricas del Modelo 1:")
print(classification_report(y_test, y_pred_classes_1))

# Matriz de confusión Model_2
confusion_mtx_2 = confusion_matrix(y_test, y_pred_classes_2)
print("\nMatriz de confusión del Modelo 2:")
print(confusion_mtx_2)
print("\nOtras métricas del Modelo 2:")
print(classification_report(y_test, y_pred_classes_2))
```

```
12/12 [==============================] - 0s 9ms/step - loss: 0.8537 - accuracy:
0.7722
12/12 [==============================] - 0s 8ms/step - loss: 0.7686 - accuracy:
0.7750
Métricas del Modelo 1 en el Conjunto de Prueba:
Pérdida: 85.37%
Precisión: 77.22%

Métricas del Modelo 2 en el Conjunto de Prueba:
Pérdida: 76.86%
Precisión: 77.5%
12/12 [==============================] - 0s 871us/step
12/12 [==============================] - 0s 875us/step

Matriz de confusión del Modelo 1:
[[37  3  0  0  0  4  0  0]
 [ 8 29  0  0  0  1  2  0]
 [ 0  0 40  0  0  0  3  4]
 [ 0  0  4 46  0  0  0  0]
 [ 1  8  0  0 18 11  0  0]
 [10  6  0  0  0 36  3  0]
 [ 0  0  0  0  0  0 46  0]
 [ 0  0  0 14  0  0  0 26]]

Otras métricas del Modelo 1:
              precision    recall  f1-score   support

           0       0.66      0.84      0.74        44
           1       0.63      0.72      0.67        40
           2       0.91      0.85      0.88        47
           3       0.77      0.92      0.84        50
           4       1.00      0.47      0.64        38
           5       0.69      0.65      0.67        55
           6       0.85      1.00      0.92        46
           7       0.87      0.65      0.74        40
```

```
    accuracy                           0.77       360
   macro avg       0.80      0.76      0.76       360
weighted avg       0.79      0.77      0.77       360


Matriz de confusión del Modelo 2:
[[25 13  0  0  2  4  0  0]
 [ 8 31  0  0  0  1  0  0]
 [ 0  0 39  0  0  0  1  7]
 [ 0  0  0 43  0  0  0  7]
 [ 0  2  0  0 23 13  0  0]
 [10  3  0  0  5 37  0  0]
 [ 0  0  0  0  0  0 46  0]
 [ 0  0  0  5  0  0  0 35]]

Otras métricas del Modelo 2:
              precision    recall  f1-score   support

           0       0.58      0.57      0.57        44
           1       0.63      0.78      0.70        40
           2       1.00      0.83      0.91        47
           3       0.90      0.86      0.88        50
           4       0.77      0.61      0.68        38
           5       0.67      0.67      0.67        55
           6       0.98      1.00      0.99        46
           7       0.71      0.88      0.79        40

    accuracy                           0.78       360
   macro avg       0.78      0.77      0.77       360
weighted avg       0.78      0.78      0.78       360
```

Modelo	Pérdida (%)	Precisión (%)
Model_1	85.37	77.22
Model_2	76.86	77.5

Ambos modelos presentan precisiones bastante similares en el conjunto de prueba, aunque se observa una diferencia la pérdida. La pérdida del Model_2 es menor, lo que indica que las predicciones del Model_2 están más cercanas a las etiquetas reales en comparación con el Model_1. Además, se pueden observar diferencias entre las clases de los modelos, ya que algunas clases muestran mejores métricas tanto en precisión como en recall. Por ejemplo, el Model_2 tiene un rendimiento mejor para las clases 2, 3, 6 y 7 en comparación con el Model_1.

En general, el Model_2, con una pérdida menor, parece ser una mejor opción al elegir un modelo. No obstante, es importante tener en cuenta las curvas de aprendizaje, donde el Model_1 parece indicar un mejor ajuste en comparación con el Model_2, especialmente en las gráficas de precisión de este último, lo que podría interpretarse como un posible sobreajuste.

En conclusión, ambos modelos son similares y escoger cualquiera de ellos parece ser una buena elección. Destacamos que el Model_2 es más complejo al tener una capa densa oculta adicional. Para asegurarnos de la elección del modelo, se deberían explorar otros factores como la complejidad del modelo, el tiempo de entrenamiento y la capacidad de generalización a futuros conjuntos de datos.