PREGUNTA 1 Cargar los archivos

In [ ]:
```
import pandas as pd

# Cargar los datos de expresión y etiquetas
expression_data = pd.read_csv('D:\TCGA-PANCAN-HiSeq-801x20531\expression.csv')
labels_data = pd.read_csv('D:\TCGA-PANCAN-HiSeq-801x20531\labels.csv')



print(labels_data)
```

```
              X Class
0      sample_0  PRAD
1      sample_1  LUAD
2      sample_2  PRAD
3      sample_3  PRAD
4      sample_4  BRCA
..          ...   ...
796  sample_796  BRCA
797  sample_797  LUAD
798  sample_798  COAD
799  sample_799  PRAD
800  sample_800  PRAD

[801 rows x 2 columns]
```

PREGUNTA 2 Selección de genes

In [ ]:
```
import numpy as np

variances = expression_data.var(axis=0)
threshold = np.percentile(variances, 80)
selected_genes = variances[variances >= threshold].index
filtered_expression_data = expression_data[selected_genes]

print(selected_genes)
print(filtered_expression_data)
```

```
Index(['gene_18', 'gene_21', 'gene_26', 'gene_28', 'gene_30', 'gene_31',
       'gene_33', 'gene_39', 'gene_44', 'gene_45',
       ...
       'gene_20445', 'gene_20466', 'gene_20472', 'gene_20476', 'gene_20484',
       'gene_20490', 'gene_20500', 'gene_20504', 'gene_20524', 'gene_20529'],
      dtype='object', length=4107)
      gene_18   gene_21   gene_26   gene_28   gene_30   gene_31    gene_33  \
0    0.591871  0.000000  7.215116  6.620204  0.000000  4.063658   4.747656
1    0.000000  0.000000  9.949812  1.174790  0.000000  0.000000   1.327170
2    1.683023  3.660427  6.096650  7.680507  0.000000  0.452595   0.000000
3    1.267356  0.000000  3.913761  6.469165  0.000000  1.267356   7.742714
4    0.889707  0.000000  1.942120  5.861429  0.000000  0.649386   2.612801
..        ...       ...       ...       ...       ...       ...        ...
796  0.496922  1.955573  0.000000  6.139531  0.000000  0.000000   0.865919
797  0.000000  0.000000  9.169589  3.480317  0.000000  3.844888   2.245891
798  1.002595  5.179822  1.325099  4.362533  6.831738  7.452291   5.574986
799  0.000000  0.000000  3.450934  5.514419  0.581881  1.316320   2.451778
800  0.000000  4.709103  3.485246  6.164110  0.748375  2.831715  10.319525

      gene_39   gene_44   gene_45  ...  gene_20445  gene_20466  gene_20472  \
0    0.000000  1.010279  6.962850  ...    2.015391    4.377061    1.010279
1    0.811142  2.651224  6.698691  ...    6.266046    2.530820    0.587845
2    0.452595  0.000000  7.592345  ...    3.677147    6.172313    0.000000
3    0.434882  1.931418  8.890292  ...    1.637239    6.318335    0.000000
4    0.649386  0.000000  4.443070  ...    0.649386    6.775999    5.370398
..        ...       ...       ...  ...         ...         ...         ...
796  0.000000  1.159435  4.611202  ...    3.495810    7.520336    4.536476
797  0.000000  0.000000  7.580251  ...    3.547845    6.547101    0.618051
798  6.299794  1.002595  6.941834  ...    8.599548    6.825366    4.705480
799  0.000000  0.000000  7.694602  ...    3.898170    4.035615    1.800703
800  2.525919  2.137340  8.015053  ...    6.647696    6.848310    0.748375

     gene_20476  gene_20484  gene_20490  gene_20500  gene_20504  gene_20524  \
0      9.175285    2.717803    0.000000    5.902800    2.602077    7.220030
1      0.000000    4.516185    4.008227    1.004394    0.811142    6.256586
2      8.834108    6.535353    2.337254    0.000000    5.014445    5.401607
3      7.529188    4.466457    0.434882    5.075383    6.233192    8.942805
4      7.647883    0.889707    2.045093    3.954001    4.586531    7.181162
..          ...         ...         ...         ...         ...         ...
796    8.486940    0.496922    3.070784    3.823709    1.611739    4.484415
797    8.207756    0.618051    0.000000    0.000000    3.173351    6.555327
798    8.089895    1.588469    2.326135    0.000000    4.092394    3.589763
799    7.605413    5.745713    0.000000    1.800703    5.861179    4.745888
800    8.148944    3.195174    2.137340    5.860772    6.217814    9.139459

     gene_20529
0      5.286759
1      2.094168
2      1.683023
3      3.292001
4      5.110372
..          ...
796    8.819269
797    9.659081
798    4.677458
799    5.718751
800    4.550716

[801 rows x 4107 columns]
```

```
C:\Users\diego\AppData\Local\Temp\ipykernel_2236\2896967688.py:3: FutureWarnin
g: The default value of numeric_only in DataFrame.var is deprecated. In a futur
e version, it will default to False. In addition, specifying 'numeric_only=Non
e' is deprecated. Select only valid columns or specify the value of numeric_onl
y to silence this warning.
  variances = expression_data.var(axis=0)
```

PREGUNTA 3 Normalizar las expresiones con la transformación minmax.

In [ ]:
```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
normalized_expression_data = pd.DataFrame(scaler.fit_transform(filtered_expressi
print(normalized_expression_data)
```

```
      gene_18   gene_21   gene_26   gene_28   gene_30   gene_31   gene_33  \
0    0.054492  0.000000  0.543929  0.645785  0.000000  0.426832  0.356653
1    0.000000  0.000000  0.750091  0.114598  0.000000  0.000000  0.099700
2    0.154951  0.470410  0.459611  0.749215  0.000000  0.047539  0.000000
3    0.116682  0.000000  0.295048  0.631051  0.000000  0.133119  0.581648
4    0.081913  0.000000  0.146411  0.571768  0.000000  0.068209  0.196279
..        ...       ...       ...       ...       ...       ...       ...
796  0.045750  0.251316  0.000000  0.598896  0.000000  0.000000  0.065050
797  0.000000  0.000000  0.691272  0.339496  0.000000  0.403853  0.168716
798  0.092306  0.665672  0.099896  0.425554  0.685723  0.782762  0.418804
799  0.000000  0.000000  0.260157  0.537918  0.058405  0.138262  0.184183
800  0.000000  0.605178  0.262744  0.601294  0.075117  0.297433  0.775224

      gene_39   gene_44   gene_45  ...  gene_20445  gene_20466  gene_20472  \
0    0.000000  0.086837  0.553104  ...    0.215858    0.463935    0.133042
1    0.095402  0.227883  0.525729  ...    0.671123    0.268247    0.077412
2    0.053232  0.000000  0.618338  ...    0.393840    0.654217    0.000000
3    0.051148  0.166013  0.752842  ...    0.175356    0.669695    0.000000
4    0.076377  0.000000  0.291981  ...    0.069552    0.718203    0.707219
..        ...       ...       ...  ...         ...         ...         ...
796  0.000000  0.099658  0.309405  ...    0.374418    0.797097    0.597401
797  0.000000  0.000000  0.617084  ...    0.379991    0.693942    0.081390
798  0.740948  0.086177  0.550926  ...    0.921052    0.723436    0.619657
799  0.000000  0.000000  0.628934  ...    0.417512    0.427744    0.237132
800  0.297085  0.183713  0.662142  ...    0.711999    0.725868    0.098552

     gene_20476  gene_20484  gene_20490  gene_20500  gene_20504  gene_20524  \
0      0.847687    0.306397    0.000000    0.653184    0.326209    0.784147
1      0.000000    0.509141    0.357826    0.111143    0.101689    0.679510
2      0.816167    0.736776    0.208654    0.000000    0.628635    0.586653
3      0.695608    0.503535    0.038823    0.561625    0.781424    0.971253
4      0.706574    0.100303    0.182571    0.437536    0.574990    0.779926
..          ...         ...         ...         ...         ...         ...
796    0.784093    0.056022    0.274138    0.423119    0.202056    0.487040
797    0.758299    0.069677    0.000000    0.000000    0.397827    0.711956
798    0.747410    0.179079    0.207661    0.000000    0.513043    0.389874
799    0.702650    0.647755    0.000000    0.199260    0.734786    0.515437
800    0.752866    0.360215    0.190807    0.648533    0.779496    0.992611

     gene_20529
0      0.442221
1      0.141369
2      0.102625
3      0.254246
4      0.425599
..          ...
796    0.775104
797    0.854243
798    0.384804
799    0.482929
800    0.372860

[801 rows x 4107 columns]
```

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

In [ ]:
```
from sklearn.model_selection import train_test_split

labels = labels_data['Class'].values
X_train, X_test, y_train, y_test = train_test_split(normalized_expression_data,
```

```
print('xtrain shape: ', X_train.shape)
print('ytrain shape: ', y_train.shape)
print('xtest shape: ', X_test.shape)
print('ytest shape: ', y_test.shape)
```

```
xtrain shape:  (534, 4107)
ytrain shape:  (534,)
xtest shape:  (267, 4107)
ytest shape:  (267,)
```

PREGUNTA 5 Definir el modelo 1, que consiste en una red neuronal con una capa oculta densa de 100 nodos, con activación relu. Añadir un 30% de dropout. Proporcionar el summary del modelo y justificar el total de parámetros de cada capa.

In [ ]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout

model = Sequential([
    Dense(100, activation='relu', input_dim=4107),
    Dropout(0.3),
    Dense(50, activation='relu'),
    Dropout(0.3),
    Dense(len(labels_data['Class'].unique()), activation='softmax')
])

model.summary()
```

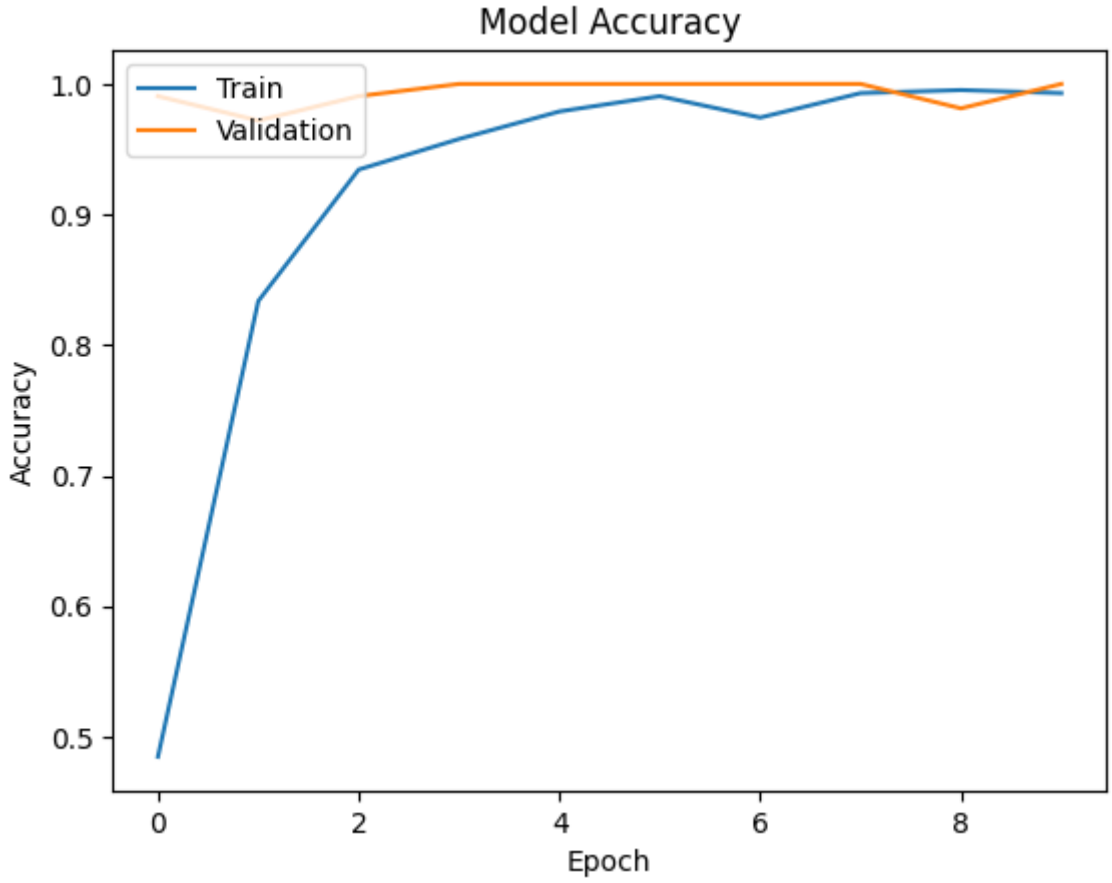
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 100)               410800

 dropout (Dropout)           (None, 100)               0

 dense_1 (Dense)             (None, 50)                5050

 dropout_1 (Dropout)         (None, 50)                0

 dense_2 (Dense)             (None, 5)                 255

=================================================================
Total params: 416,105
Trainable params: 416,105
Non-trainable params: 0
_________________________________________________________________
```

PREGUNTA 6 Ajustar el modelo 1 con un 20% de validación, mostrando la curva de aprendizaje de entrenamiento y validación con 10 epocas.

In [ ]:
```
from sklearn.preprocessing import LabelEncoder
from keras.optimizers import Adam
from sklearn.preprocessing import LabelEncoder
from keras.utils import to_categorical
```

```
le = LabelEncoder()
y_train = to_categorical(le.fit_transform(y_train))
y_test = to_categorical(le.transform(y_test))
# Compile model with binary crossentropy loss function
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy']

history = model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split

import matplotlib.pyplot as plt

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```

```
Epoch 1/10
14/14 [==============================] - 2s 28ms/step - loss: 0.4957 - accurac
y: 0.4848 - val_loss: 0.1933 - val_accuracy: 0.9907
Epoch 2/10
14/14 [==============================] - 0s 10ms/step - loss: 0.2291 - accurac
y: 0.8337 - val_loss: 0.0689 - val_accuracy: 0.9720
Epoch 3/10
14/14 [==============================] - 0s 10ms/step - loss: 0.1182 - accurac
y: 0.9344 - val_loss: 0.0399 - val_accuracy: 0.9907
Epoch 4/10
14/14 [==============================] - 0s 10ms/step - loss: 0.0909 - accurac
y: 0.9578 - val_loss: 0.0236 - val_accuracy: 1.0000
Epoch 5/10
14/14 [==============================] - 0s 11ms/step - loss: 0.0681 - accurac
y: 0.9789 - val_loss: 0.0145 - val_accuracy: 1.0000
Epoch 6/10
14/14 [==============================] - 0s 10ms/step - loss: 0.0456 - accurac
y: 0.9906 - val_loss: 0.0127 - val_accuracy: 1.0000
Epoch 7/10
14/14 [==============================] - 0s 10ms/step - loss: 0.0469 - accurac
y: 0.9742 - val_loss: 0.0097 - val_accuracy: 1.0000
Epoch 8/10
14/14 [==============================] - 0s 11ms/step - loss: 0.0328 - accurac
y: 0.9930 - val_loss: 0.0078 - val_accuracy: 1.0000
Epoch 9/10
14/14 [==============================] - 0s 11ms/step - loss: 0.0287 - accurac
y: 0.9953 - val_loss: 0.0125 - val_accuracy: 0.9813
Epoch 10/10
14/14 [==============================] - 0s 10ms/step - loss: 0.0279 - accurac
y: 0.9930 - val_loss: 0.0079 - val_accuracy: 1.0000
```



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

In [ ]:
```
y_pred = model.predict(X_test)
y_test_labels = np.argmax(y_test, axis=1)
y_pred_labels = np.argmax(y_pred, axis=1)
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test_labels, y_pred_labels)
print(cm)
```

```
9/9 [==============================] - 0s 2ms/step
[[102   0   0   0   0]
 [  0  29   0   0   0]
 [  0   0  47   0   0]
 [  0   0   0  44   0]
 [  0   0   0   0  45]]
```

In [ ]:
```
from sklearn.metrics import classification_report

report = classification_report(y_test_labels, y_pred_labels)
print(report)
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	102
1	1.00	1.00	1.00	29
2	1.00	1.00	1.00	47
3	1.00	1.00	1.00	44
4	1.00	1.00	1.00	45
accuracy			1.00	267
macro avg	1.00	1.00	1.00	267
weighted avg	1.00	1.00	1.00	267

In [ ]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_sc

accuracy = accuracy_score(y_test_labels, y_pred_labels)
precision = precision_score(y_test_labels, y_pred_labels, average='weighted')
recall = recall_score(y_test_labels, y_pred_labels, average='weighted')
f1 = f1_score(y_test_labels, y_pred_labels, average='weighted')

print('Accuracy:', accuracy)
print('Precision:', precision)
print('Recall:', recall)
print('F1 score:', f1)
```

```
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1 score: 1.0
```

PREGUNTA 8 Definir el modelo 2, que consiste en una red neuronal con dos capas ocultas densas de 100 nodos y 10 nodos, con activación relu. Añadir un 30% de dropout en ambas capas. Proporcionar el summary del modelo y justificar el total de parámetros de cada capa.

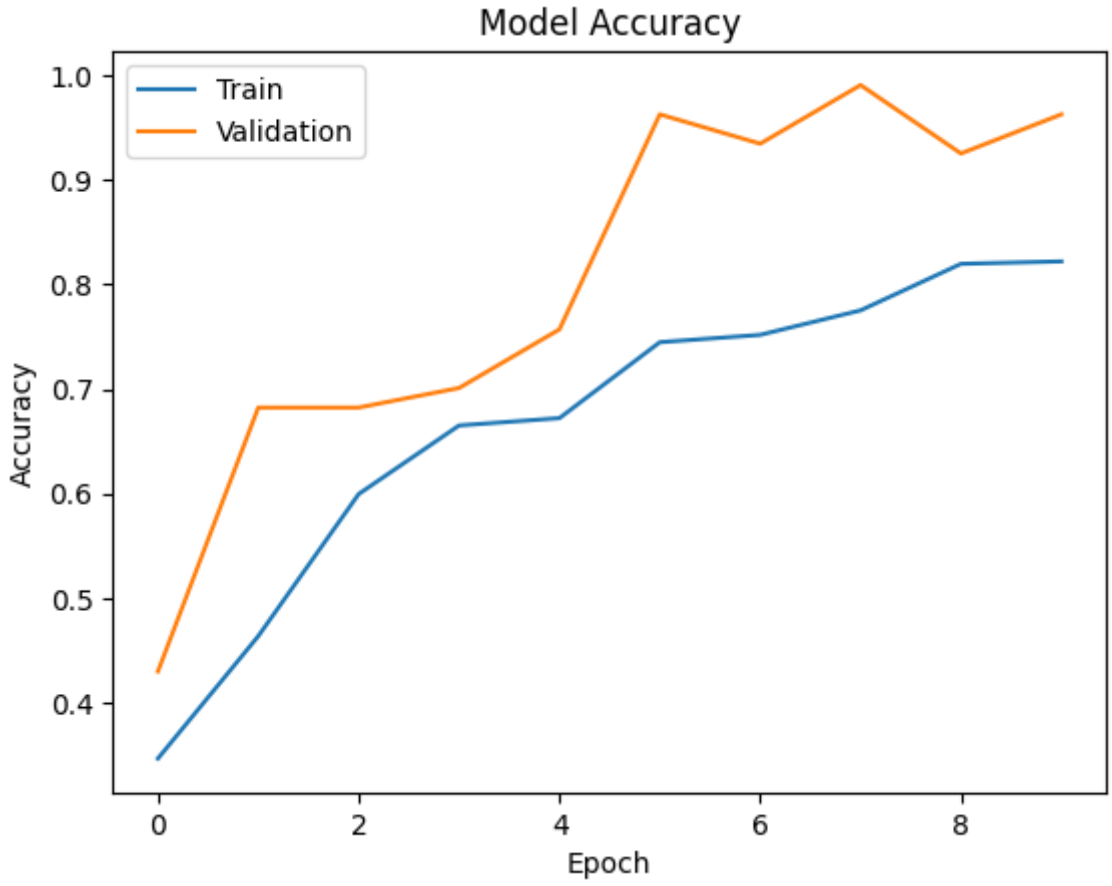
In [ ]:
```
model2 = Sequential([
    Dense(100, activation='relu', input_dim=4107),
    Dropout(0.3),
    Dense(10, activation='relu'),
    Dropout(0.3),
    Dense(len(labels_data['Class'].unique()), activation='softmax')
])

model2.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_6 (Dense)             (None, 100)               410800

 dropout_4 (Dropout)         (None, 100)               0

 dense_7 (Dense)             (None, 10)                1010

 dropout_5 (Dropout)         (None, 10)                0

 dense_8 (Dense)             (None, 5)                 55

=================================================================
Total params: 411,865
Trainable params: 411,865
Non-trainable params: 0
_________________________________________________________________
```

PREGUNTA 9 Ajustar el modelo 2 con un 20% de validación, mostrando la curva de aprendizaje de entrenamiento y validación con 10 epocas.

In [ ]:
```
model2.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'
history2 = model2.fit(X_train, y_train, epochs=10, batch_size=32, validation_spl

plt.plot(history2.history['accuracy'])
plt.plot(history2.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```

```
Epoch 1/10
14/14 [==============================] - 1s 22ms/step - loss: 0.6265 - accurac
y: 0.3466 - val_loss: 0.4909 - val_accuracy: 0.4299
Epoch 2/10
14/14 [==============================] - 0s 9ms/step - loss: 0.4876 - accuracy:
0.4637 - val_loss: 0.3882 - val_accuracy: 0.6822
Epoch 3/10
14/14 [==============================] - 0s 10ms/step - loss: 0.3939 - accurac
y: 0.5995 - val_loss: 0.3478 - val_accuracy: 0.6822
Epoch 4/10
14/14 [==============================] - 0s 10ms/step - loss: 0.3602 - accurac
y: 0.6651 - val_loss: 0.3185 - val_accuracy: 0.7009
Epoch 5/10
14/14 [==============================] - 0s 10ms/step - loss: 0.3412 - accurac
y: 0.6721 - val_loss: 0.3133 - val_accuracy: 0.7570
Epoch 6/10
14/14 [==============================] - 0s 10ms/step - loss: 0.3089 - accurac
y: 0.7447 - val_loss: 0.2901 - val_accuracy: 0.9626
Epoch 7/10
14/14 [==============================] - 0s 10ms/step - loss: 0.3068 - accurac
y: 0.7518 - val_loss: 0.2706 - val_accuracy: 0.9346
Epoch 8/10
14/14 [==============================] - 0s 10ms/step - loss: 0.2900 - accurac
y: 0.7752 - val_loss: 0.2623 - val_accuracy: 0.9907
Epoch 9/10
14/14 [==============================] - 0s 10ms/step - loss: 0.2689 - accurac
y: 0.8197 - val_loss: 0.2289 - val_accuracy: 0.9252
Epoch 10/10
14/14 [==============================] - 0s 10ms/step - loss: 0.2604 - accurac
y: 0.8220 - val_loss: 0.1988 - val_accuracy: 0.9626
```



In [ ]:
```
y_pred2 = model2.predict(X_test)
y_test_labels = np.argmax(y_test, axis=1)
```

```
y_pred_labels2 = np.argmax(y_pred2, axis=1)

cm2 = confusion_matrix(y_test_labels, y_pred_labels2)
print(cm2)

report2 = classification_report(y_test_labels, y_pred_labels2)
print(report2)

accuracy2 = accuracy_score(y_test_labels, y_pred_labels2)
precision2 = precision_score(y_test_labels, y_pred_labels2, average='weighted')
recall2 = recall_score(y_test_labels, y_pred_labels2, average='weighted')
f1_2 = f1_score(y_test_labels, y_pred_labels2, average='weighted')

print('Accuracy:', accuracy2)
print('Precision:', precision2)
print('Recall:', recall2)
print('F1 score:', f1_2)
```

```
9/9 [==============================] - 0s 2ms/step
[[102   0   0   0   0]
 [  0  29   0   0   0]
 [  5   0  42   0   0]
 [  0   0   0  44   0]
 [  0   0   0   0  45]]
              precision    recall  f1-score   support

           0       0.95      1.00      0.98       102
           1       1.00      1.00      1.00        29
           2       1.00      0.89      0.94        47
           3       1.00      1.00      1.00        44
           4       1.00      1.00      1.00        45

    accuracy                           0.98       267
   macro avg       0.99      0.98      0.98       267
weighted avg       0.98      0.98      0.98       267

Accuracy: 0.9812734082397003
Precision: 0.9821484826210227
Recall: 0.9812734082397003
F1 score: 0.9809713826750486
```

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

Los dos modelos parecen tener un rendimiento muy bueno en la tarea de clasificación, con una precisión, recall y f1-score cercanos a 1 en todas las clases. Sin embargo, el modelo 1 parece tener un rendimiento ligeramente mejor que el modelo 2, ya que tiene una precisión, recall y f1-score perfectos en todas las clases, mientras que el modelo 2 tiene una precisión ligeramente inferior en la clase 2 y una precisión perfecta en las demás clases. Además, el modelo 1 tiene una precisión promedio, recall y f1-score de 1, mientras que el modelo 2 tiene una precisión promedio, recall y f1-score de 0.98. En términos de accuracy, el modelo 1 tiene una precisión perfecta, mientras que el modelo 2 tiene una precisión de 0.981. En general, ambos modelos parecen ser muy buenos, pero el modelo 1 parece tener un rendimiento ligeramente mejor en esta tarea de clasificación específica.