ACTIVIDAD 2: REDES NEURONALES CONVOLUCIONALES

En esta actividad, vamos a trabajar con Convolutional Neural Networks para resolver un problema de clasificación de imágenes. En particular, vamos a clasificar imágenes de personajes de la conocida serie de los Simpsons.

Como las CNN profundas son un tipo de modelo bastante avanzado y computacionalmente costoso, se recomienda hacer la práctica en Google Colaboratory con soporte para GPUs. En este enlace se explica cómo activar un entorno con GPUs. *Nota: para leer las imágenes y estandarizarlas al mismo tamaño se usa la librería opencv. Esta líbrería está ya instalada en el entorno de Colab, pero si trabajáis de manera local tendréis que instalarla.*

El dataset a utilizar consiste en imágenes de personajes de los Simpsons extraídas directamente de capítulos de la serie. Este dataset ha sido recopilado por Alexandre Attia y es más complejo que el dataset de Fashion MNIST que hemos utilizado hasta ahora. Aparte de tener más clases (vamos a utilizar los 18 personajes con más imágenes), los personajes pueden aparecer en distintas poses, en distintas posiciones de la imagen o con otros personajes en pantalla (si bien el personaje a clasificar siempre aparece en la posición predominante).

El dataset de training puede ser descargado desde aquí:

Training data (~500MB)

Por otro lado, el dataset de test puede ser descargado de aquí:

Test data (~10MB)

Antes de empezar la práctica, se recomienda descargar las imágenes y echarlas un vistazo.

Carga de los datos

In [1]:
```
import cv2
import os
import numpy as np
import keras
from tensorflow import keras
import pandas as pd
import matplotlib.pyplot as plt
import glob
import tensorflow as tf
```

In [2]:
```
print(tf.__version__)
```

```
2.10.1
```

In [3]:
```
print("Num GPUs Avaible: ", len(tf.config.experimental.list_physical_devices('GPU')))
```

```
Num GPUs Avaible:  1
```

In [4]:
```
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout
from tensorflow.keras.models import Sequential
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from keras.losses import categorical_crossentropy
import keras_preprocessing
from keras_preprocessing import image
from keras_preprocessing.image import ImageDataGenerator
from keras.models import load_model
from skimage.color import rgb2gray
import matplotlib.pyplot as plt
import time
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import ConfusionMatrixDisplay
plt.rcParams["figure.figsize"] = (20,6)
```

In [5]:
```
# Esta variable contiene un mapeo de número de clase a personaje.
# Utilizamos sólo los 18 personajes del dataset que tienen más imágenes.
MAP_CHARACTERS = {
    0: 'abraham_grampa_simpson', 1: 'apu_nahasapeemapetilon', 2: 'bart_simpson',
    3: 'charles_montgomery_burns', 4: 'chief_wiggum', 5: 'comic_book_guy', 6: 'edna_krabappel',
    7: 'homer_simpson', 8: 'kent_brockman', 9: 'krusty_the_clown', 10: 'lisa_simpson',
    11: 'marge_simpson', 12: 'milhouse_van_houten', 13: 'moe_szyslak',
    14: 'ned_flanders', 15: 'nelson_muntz', 16: 'principal_skinner', 17: 'sideshow_bob'
}

# Vamos a standarizar todas las imágenes a tamaño 64x64
IMG_SIZE = 60
```

In [6]:
```
def load_train_set(dirname, map_characters, verbose=True):
    """Esta función carga los datos de training en imágenes.

    Como las imágenes tienen tamaños distintas, utilizamos la librería opencv
    para hacer un resize y adaptarlas todas a tamaño IMG_SIZE x IMG_SIZE.

    Args:
        dirname: directorio completo del que leer los datos
        map_characters: variable de mapeo entre labels y personajes
        verbose: si es True, muestra información de las imágenes cargadas

    Returns:
        X, y: X es un array con todas las imágenes cargadas con tamaño
                IMG_SIZE x IMG_SIZE
              y es un array con las labels de correspondientes a cada imagen
    """
    X_train = []
    y_train = []
    for label, character in map_characters.items():
        files = os.listdir(os.path.join(dirname, character))
        images = [file for file in files if file.endswith("jpg")]
        if verbose:
          print("Leyendo {} imágenes encontradas de {}".format(len(images), character))
        for image_name in images:
            image = cv2.imread(os.path.join(dirname, character, image_name))
            X_train.append(cv2.resize(image,(IMG_SIZE, IMG_SIZE)))
            y_train.append(label)
    return np.array(X_train), np.array(y_train)
```

In [7]:
```
def load_test_set(dirname, map_characters, verbose=True):
    """Esta función funciona de manera equivalente a la función load_train_set
    pero cargando los datos de test."""
    X_test = []
    y_test = []
    reverse_dict = {v: k for k, v in map_characters.items()}
    for filename in glob.glob(dirname + '/*.*'):
        char_name = "_".join(filename.split('\\')[-1].split('_')[:-1])
        if char_name in reverse_dict:
            image = cv2.imread(filename)
            image = cv2.resize(image, (IMG_SIZE, IMG_SIZE))
            X_test.append(image)
```

```
            y_test.append(reverse_dict[char_name])
    if verbose:
        print("Leídas {} imágenes de test".format(len(X_test)))
    return np.array(X_test), np.array(y_test)
```

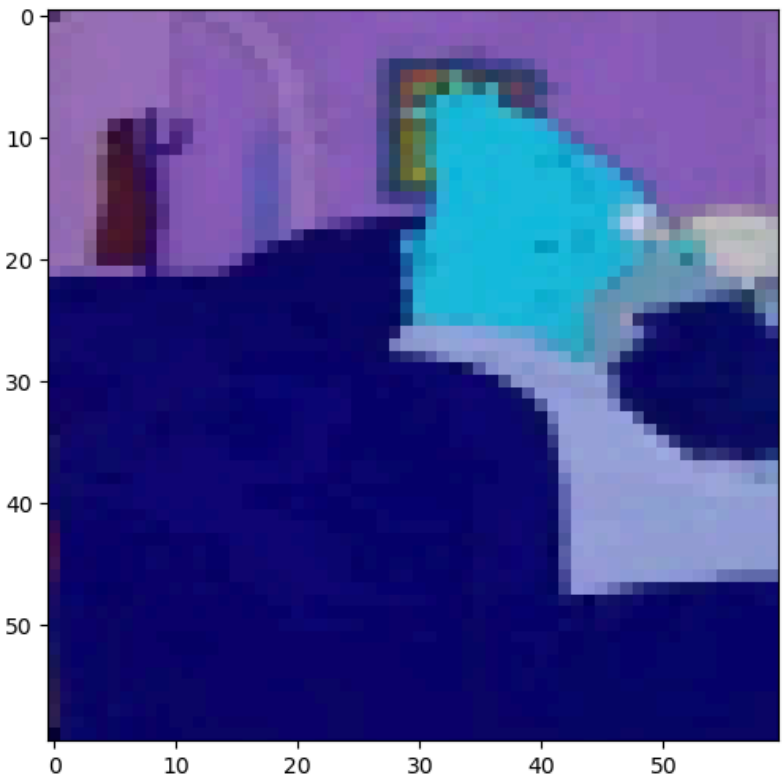
In [8]:
```
# Cargamos los datos. Si no estás trabajando en colab, cambia los paths por
# los de los ficheros donde hayas descargado los datos.
# DATASET_TRAIN_PATH_COLAB = "/root/.keras/datasets/simpsons"
DATASET_TRAIN_PATH_COLAB = "./simpsons_train/simpsons"
# DATASET_TEST_PATH_COLAB = "/root/.keras/datasets/simpsons_testset"
DATASET_TEST_PATH_COLAB = "./simpsons_test/simpsons_testset"

X, y = load_train_set(DATASET_TRAIN_PATH_COLAB, MAP_CHARACTERS)
X_t, y_t = load_test_set(DATASET_TEST_PATH_COLAB, MAP_CHARACTERS)
```

```
Leyendo 913 imágenes encontradas de abraham_grampa_simpson
Leyendo 623 imágenes encontradas de apu_nahasapeemapetilon
Leyendo 1342 imágenes encontradas de bart_simpson
Leyendo 1193 imágenes encontradas de charles_montgomery_burns
Leyendo 986 imágenes encontradas de chief_wiggum
Leyendo 469 imágenes encontradas de comic_book_guy
Leyendo 457 imágenes encontradas de edna_krabappel
Leyendo 2246 imágenes encontradas de homer_simpson
Leyendo 498 imágenes encontradas de kent_brockman
Leyendo 1206 imágenes encontradas de krusty_the_clown
Leyendo 1354 imágenes encontradas de lisa_simpson
Leyendo 1291 imágenes encontradas de marge_simpson
Leyendo 1079 imágenes encontradas de milhouse_van_houten
Leyendo 1452 imágenes encontradas de moe_szyslak
Leyendo 1454 imágenes encontradas de ned_flanders
Leyendo 358 imágenes encontradas de nelson_muntz
Leyendo 1194 imágenes encontradas de principal_skinner
Leyendo 877 imágenes encontradas de sideshow_bob
Leídas 890 imágenes de test
```

In [9]:
```
# Vamos a barajar aleatoriamente los datos. Esto es importante ya que si no
# lo hacemos y, por ejemplo, cogemos el 20% de los datos finales como validation
# set, estaremos utilizando solo un pequeño número de personajes, ya que
# las imágenes se leen secuencialmente personaje a personaje.
perm = np.random.permutation(len(X))
X, y = X[perm], y[perm]
```

In [10]:
```
plt.imshow(X_t[2])
```

Out[10]:
```
<matplotlib.image.AxesImage at 0x28cdc22e130>
```



In [11]:
```
plt.imshow(np.flip(X_t[2], axis=-1) )
```

Out[11]:
```
<matplotlib.image.AxesImage at 0x28cd9248130>
```

Ejercicio

Utilizando Convolutional Neural Networks con Keras, entrenar un clasificador que sea capaz de reconocer personajes en imágenes de los Simpsons con una accuracy en el dataset de test de, al menos, **85%**. Redactar un informe analizando varias de las alternativas probadas y los resultados obtenidos.

A continuación se detallan una serie de aspectos para ser analizados en vuestro informe:

- Análisis de los datos a utilizar.
- Análisis de resultados, obtención de métricas de *precision* y *recall* por clase y análisis de qué clases obtienen mejores o peores resultados.
- Análisis visual de los errores de la red. ¿Qué tipo de imágenes o qué personajes dan más problemas a nuestro modelo?
- Comparación de modelos CNNs con un modelo de Fully Connected para este problema.
- Utilización de distintas arquitecturas CNNs, comentando aspectos como su profundidad, hiperparámetros utilizados, optimizador, uso de técnicas de regularización, *batch normalization*, etc.
- Utilización de *data augmentation*. Esto puede conseguirse con la clase ImageDataGenerator de Keras.

Notas:

- Recuerda partir los datos en training/validation para tener una buena estimación de los valores que nuestro modelo tendrá en los datos de test, así como comprobar que no estamos cayendo en overfitting. Una posible partición puede ser 80 / 20.
- No es necesario mostrar en el notebook las trazas de entrenamiento de todos los modelos entrenados, si bien una buena idea seria guardar gráficas de esos entrenamientos para el análisis. Sin embargo, **se debe mostrar el entrenamiento completo del mejor modelo obtenido y la evaluación de los datos de test con este modelo**.
- Las imágenes **no están normalizadas**. Hay que normalizarlas como hemos hecho en trabajos anteriores.
- El test set del problema tiene imágenes un poco más "fáciles", por lo que es posible encontrarse con métricas en el test set bastante mejores que en el training set.

Definición de funciones

In [12]:
```
from collections import Counter

def get_dataFrame(y):
    category_counts = Counter(y)
    data = []
    class_ids = {category: idx for idx, category in enumerate(MAP_CHARACTERS)}
    for category, count in category_counts.items():
        class_id = class_ids.get(category, -1)
        class_name = MAP_CHARACTERS.get(category, 'Unknown')
        data.append([class_id, class_name, count])

    # Crear el DataFrame
    df = pd.DataFrame(data, columns=['Class ID', 'Class Name', 'Number of Images'])

    # Ordenar el DataFrame por ID de clase (opcional)
    df = df.sort_values(by='Number of Images',ascending=False)
    return df

def display_images(set_img,y_label,num_rows,num_cols):
    fig,axes=plt.subplots(num_rows,num_cols,figsize=(6,6))
    axes=axes.ravel()
    for i in range(len(set_img)):
        plt.tight_layout()
        img=set_img[i]
        ax=plt.subplot(num_rows,num_cols,i+1)
        ax.title.set_text(str(MAP_CHARACTERS[y_label[i]]))
        plt.imshow(np.flip(img, axis=-1))
        plt.axis('off')
```

In [13]:
```
#Función para graficar resultados del performance del modelo
def PlotPerformance(acc, val_acc, loss, val_loss, name_exp):
    accuracy = acc
    val_accuracy = val_acc
    loss_model = loss
    val_loss_model = val_loss
    Accname = f'{name_exp} Accuracy'
    Lossname = f'{name_exp} Loss'

    epochs = range(len (accuracy))

    #------------------------------------------------
    # Accuracy
    #------------------------------------------------

    plt.plot(epochs, accuracy,'b',label='Training accuracy')
    plt.plot(epochs, val_accuracy,'r',label='Validation accuracy')
    plt.title(Accname)
    plt.legend()
    plt.figure()

    #------------------------------------------------
    # Loss
    #------------------------------------------------

    plt.plot(epochs, loss_model, 'b', label='Training Loss')
    plt.plot(epochs, val_loss_model, 'r', label='Validation Loss')
    plt.title(Lossname)
    plt.legend()
    plt.show()
```

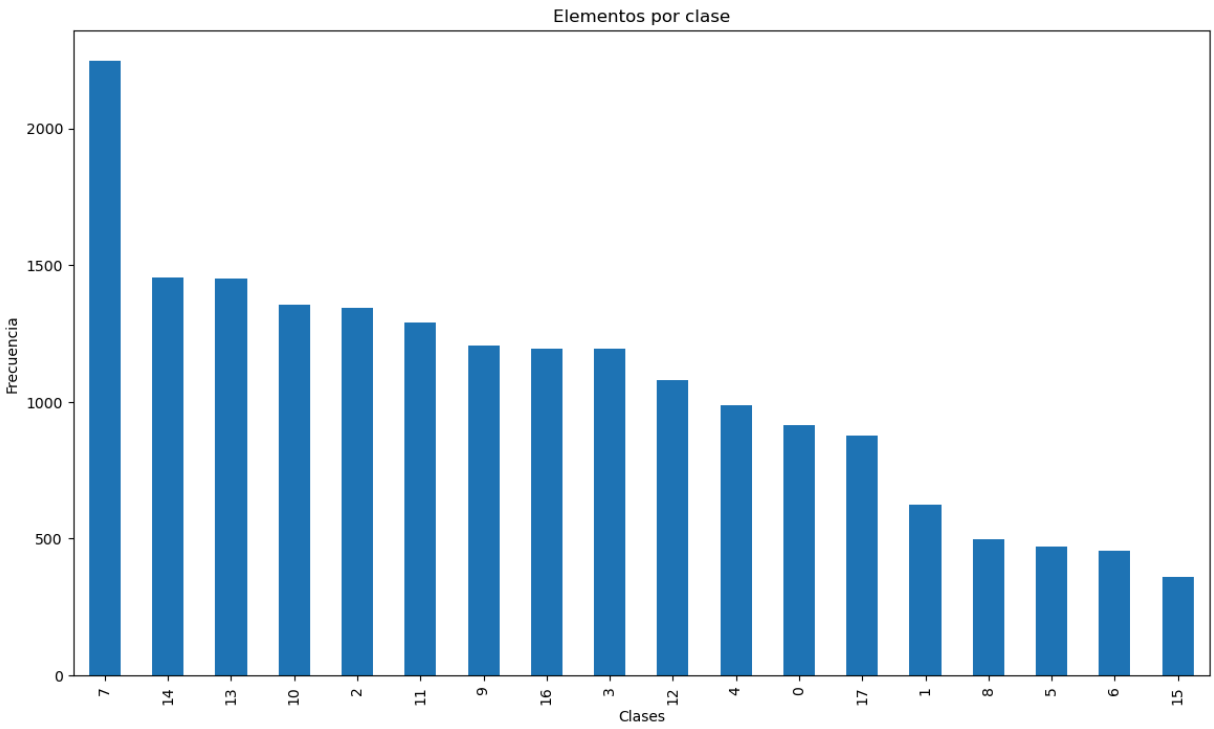
Análisis de los datos a utilizar

Verificación de las clases

In [14]:
```
import random
idx_sample=random.sample(range(0,100),18)
display_images(X[idx_sample],y[idx_sample],6,3)
```

In [15]:
```
ax=pd.Series(list(y)).value_counts().plot(kind='bar',figsize=(14,8),title='Elementos por clase')
ax.set_xlabel('Clases')
ax.set_ylabel('Frecuencia')
```

Out[15]: Text(0, 0.5, 'Frecuencia')



In [16]:
```
print(MAP_CHARACTERS[15])
```

```
nelson_muntz
```

In [17]:
```
from collections import Counter
category_counts = Counter(y)
data = []
class_ids = {category: idx for idx, category in enumerate(MAP_CHARACTERS)}
```

```
for category, count in category_counts.items():
    class_id = class_ids.get(category, -1)
    class_name = MAP_CHARACTERS.get(category, 'Unknown')
    data.append([class_id, class_name, count])

# Crear el DataFrame
df = pd.DataFrame(data, columns=['Class ID', 'Class Name', 'Number of Images'])

# Ordenar el DataFrame por ID de clase (opcional)
df = df.sort_values(by='Number of Images',ascending=False)
df
```

Out[17]:

	Class ID	Class Name	Number of Images
0	7	homer_simpson	2246
6	14	ned_flanders	1454
2	13	moe_szyslak	1452
12	10	lisa_simpson	1354
7	2	bart_simpson	1342
3	11	marge_simpson	1291
14	9	krusty_the_clown	1206
8	16	principal_skinner	1194
5	3	charles_montgomery_burns	1193
10	12	milhouse_van_houten	1079
1	4	chief_wiggum	986
9	0	abraham_grampa_simpson	913
11	17	sideshow_bob	877
13	1	apu_nahasapeemapetilon	623
4	8	kent_brockman	498
16	5	comic_book_guy	469
17	6	edna_krabappel	457
15	15	nelson_muntz	358

Balanceo de datos

Para este caso vemos que hay una gran variedad de datos, por lo que queremos disminuir esa cantidad y vamos a aumentar las imágenes que sean inferiores a 1k

Uso de Data augmentation

In [18]:

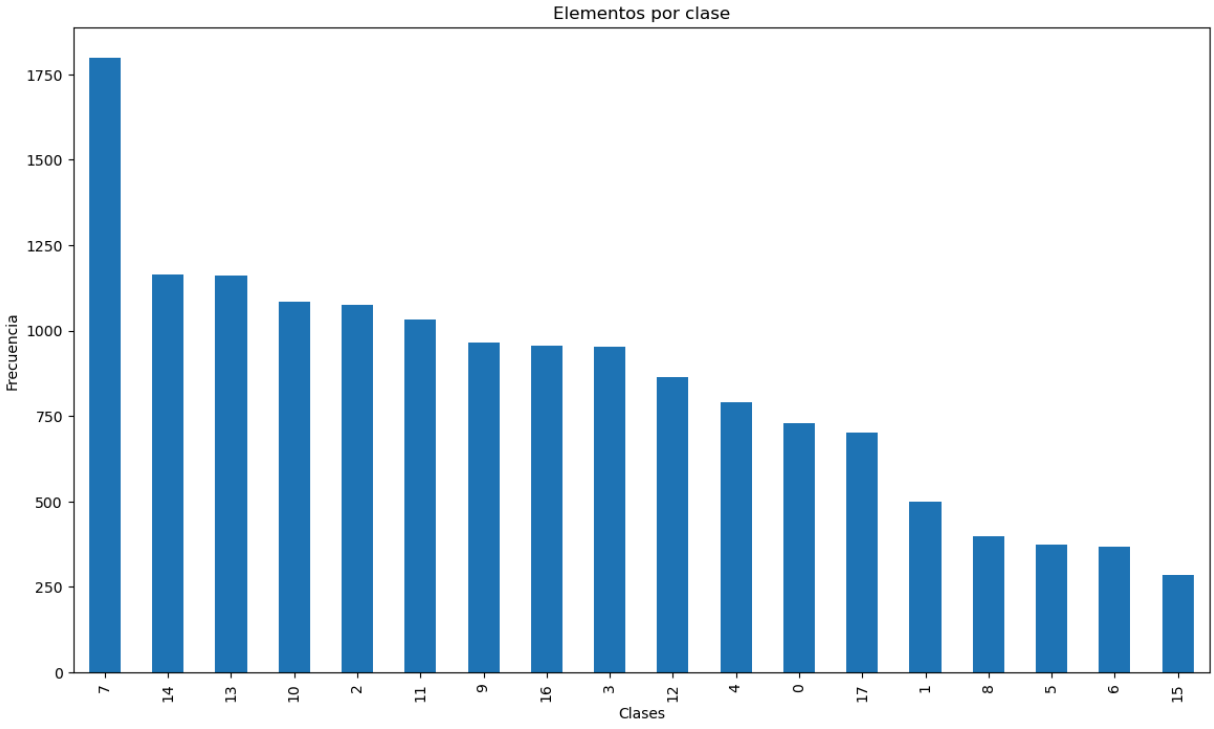
```
datagen=ImageDataGenerator(
    rotation_range=20,
     width_shift_range=0.2,      # Desplazar las imágenes horizontalmente hasta un 20%
    height_shift_range=0.2,     # Desplazar las imágenes verticalmente hasta un 20%
    shear_range=0.2,            # Aplicar operaciones de cizalladura con un ángulo de corte de hasta 20 gr
    zoom_range=0.2,             # Hacer zoom a las imágenes hasta un 20%
    horizontal_flip=True,       # Invertir las imágenes horizontalmente
    fill_mode='nearest'
)

def apply_blur(image):
    return cv2.GaussianBlur(image, (5, 5), 0)

def apply_color_shift(image):
    image_hsv = cv2.cvtColor(image, cv2.COLOR_RGB2HSV)
    image_hsv[:, :, 0] = (image_hsv[:, :, 0].astype(int) + np.random.randint(-10, 10)) % 180
    image_rgb = cv2.cvtColor(image_hsv, cv2.COLOR_HSV2RGB)
```

```
    return image_rgb

def augment_images(X, y, target_class, num_augmented):
    target_images = X[np.array(y) == target_class]
    augmented_images = []

    while len(augmented_images) < num_augmented:
        for image in target_images:
            if len(augmented_images) >= num_augmented:
                break
            image = np.expand_dims(image, 0)
            # Generar un lote de imágenes aumentadas
            augmented_batch = next(datagen.flow(image, batch_size=num_augmented))
            for augmented_image in augmented_batch:
                augmented_image = augmented_image.astype(np.uint8)
                # Aplicar desenfoque
                augmented_image = apply_blur(augmented_image)
                # Aplicar cambio de color
                augmented_image = apply_color_shift(augmented_image)
                augmented_images.append(augmented_image)
                if len(augmented_images) >= num_augmented:
                    break
    return augmented_images
```

In [19]:
```
### dividimos la data
from sklearn.model_selection import train_test_split
X_train,X_valid,y_train,y_valid=train_test_split(X,y,stratify=y,test_size=0.2)
```

In [20]:
```
ax=pd.Series(list(y_train)).value_counts().plot(kind='bar',figsize=(14,8),title='Elementos por clase')
ax.set_xlabel('Clases')
ax.set_ylabel('Frecuencia')
```

Out[20]: Text(0, 0.5, 'Frecuencia')



In [21]:
```
df=get_dataFrame(y_train)
df
```

Out[21]:

	Class ID	Class Name	Number of Images
7	7	homer_simpson	1797
10	14	ned_flanders	1163
5	13	moe_szyslak	1162
2	10	lisa_simpson	1083
11	2	bart_simpson	1074
14	11	marge_simpson	1033
3	9	krusty_the_clown	965
6	16	principal_skinner	955
13	3	charles_montgomery_burns	954
0	12	milhouse_van_houten	863
16	4	chief_wiggum	789
1	0	abraham_grampa_simpson	730
8	17	sideshow_bob	702
9	1	apu_nahasapeemapetilon	498
12	8	kent_brockman	398
17	5	comic_book_guy	375
4	6	edna_krabappel	366
15	15	nelson_muntz	286

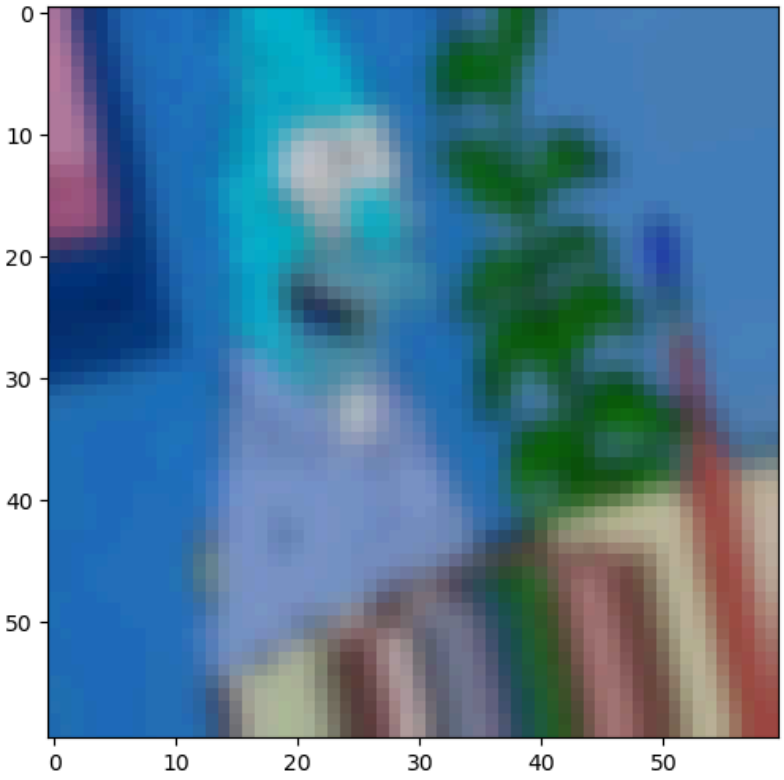
Tomamos como prioridad las que tienen alrededor de 750 imagenes, las que tenga menos de esa cantidad las vamos a aumentar, **Trabajando solo con el grupo de entrenamiento**

In [22]:
```
augmented_images=[]
augmented_labels=[]
category_counts = Counter(y_train)
THRESHOLD=750
for category in MAP_CHARACTERS:
    count=category_counts[category]
    if count < THRESHOLD:
        num_augmented = THRESHOLD - count
        print('Aumentando', num_augmented, 'imágenes para la categoría:', category)
        augmented_imgs = augment_images(X_train, y_train, category, num_augmented)
        assert len(augmented_imgs) == num_augmented, f"Error: se esperaban {num_augmented} imágenes aument
        augmented_images.extend(augmented_imgs)
        augmented_labels.extend([category] * num_augmented)

# X_augmented=np.concatenate((X,np.array(augmented_images)))


augmented_images = np.array(augmented_images)
augmented_labels = np.array(augmented_labels)
```

```
Aumentando 20 imágenes para la categoría: 0
Aumentando 252 imágenes para la categoría: 1
Aumentando 375 imágenes para la categoría: 5
Aumentando 384 imágenes para la categoría: 6
Aumentando 352 imágenes para la categoría: 8
Aumentando 464 imágenes para la categoría: 15
Aumentando 48 imágenes para la categoría: 17
```

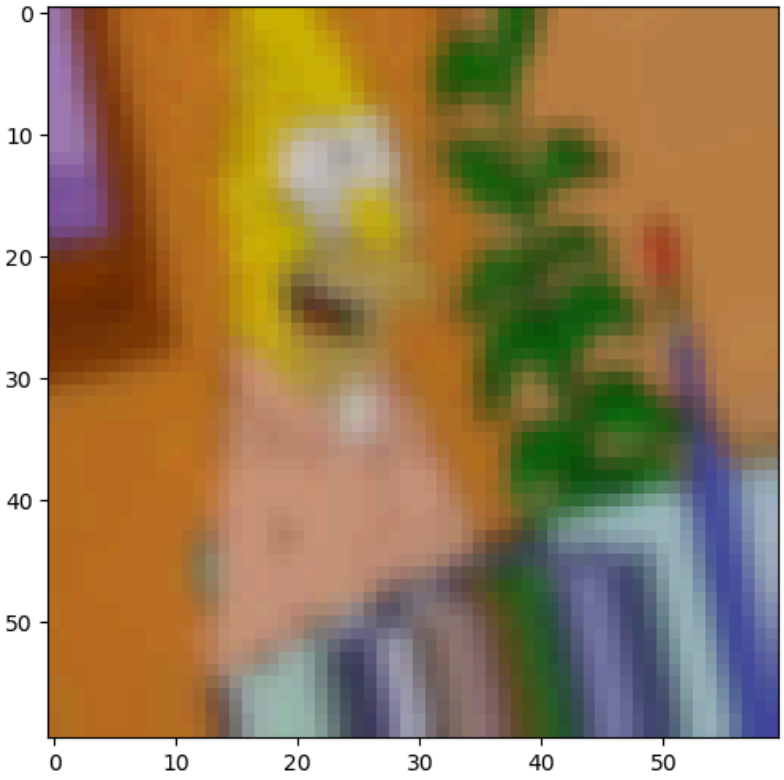
In [23]:
```
plt.imshow(augmented_images[2])
```

Out[23]:
```
<matplotlib.image.AxesImage at 0x28ce4cf5f70>
```



In [24]:
```
plt.imshow(np.flip(augmented_images[2], axis=-1) )
```

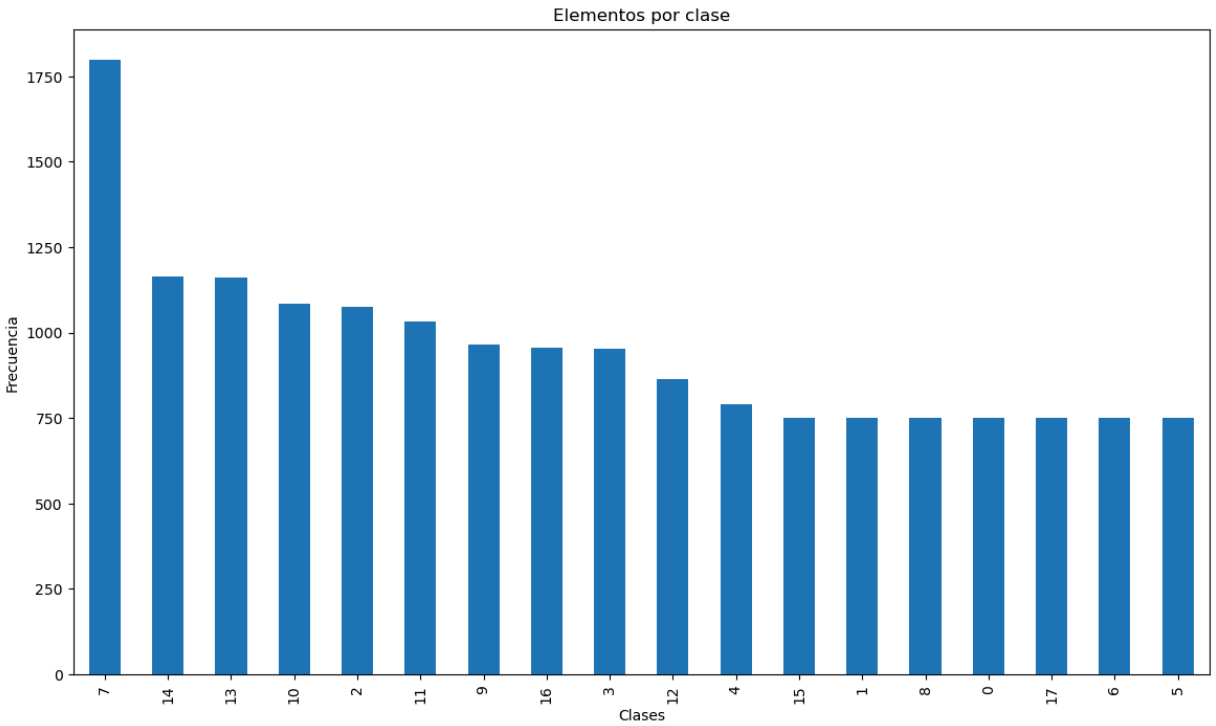
Out[24]:
```
<matplotlib.image.AxesImage at 0x28ce4d992e0>
```



In [25]:
```
X_augmented=np.concatenate((X_train,np.array(augmented_images)))
y_augmented = np.concatenate((y_train, augmented_labels))
```

In [26]:
```
ax=pd.Series(list(y_augmented)).value_counts().plot(kind='bar',figsize=(14,8),title='Elementos por clase')
ax.set_xlabel('Clases')
ax.set_ylabel('Frecuencia')
```

Out[26]:
```
Text(0, 0.5, 'Frecuencia')
```



In [27]:
```
df=get_dataFrame(y_augmented)
df
```

Out[27]:

	Class ID	Class Name	Number of Images
7	7	homer_simpson	1797
10	14	ned_flanders	1163
5	13	moe_szyslak	1162
2	10	lisa_simpson	1083
11	2	bart_simpson	1074
14	11	marge_simpson	1033
3	9	krusty_the_clown	965
6	16	principal_skinner	955
13	3	charles_montgomery_burns	954
0	12	milhouse_van_houten	863
16	4	chief_wiggum	789
15	15	nelson_muntz	750
9	1	apu_nahasapeemapetilon	750
12	8	kent_brockman	750
1	0	abraham_grampa_simpson	750
8	17	sideshow_bob	750
4	6	edna_krabappel	750
17	5	comic_book_guy	750

Serialización

Para mantener los datos y no cargar nuevamente el modelo vamos a crear el objeto de las variables serializadas

In [28]:
```
import pickle
with open('X_augmented.pkl','wb') as f:
    pickle.dump(X_augmented,f)
```

```
with open('X_valid.pkl','wb') as f:
    pickle.dump(X_valid,f)
```

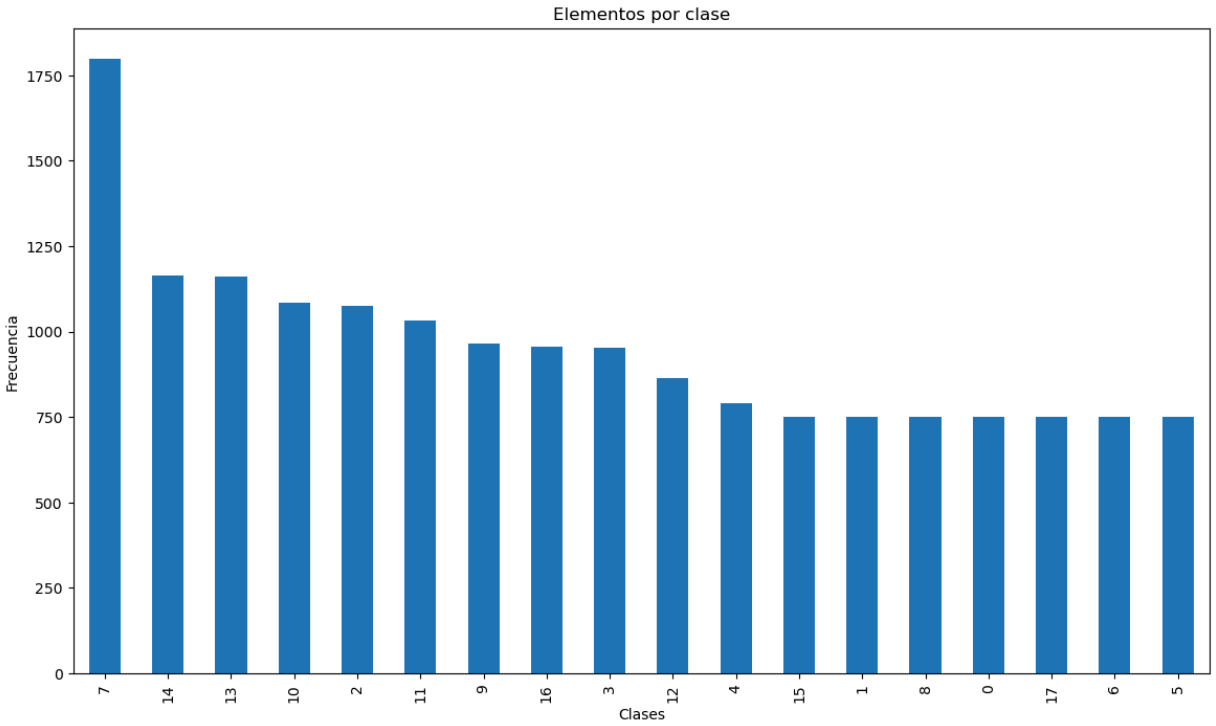
In [29]:
```
with open('y_augmented.pkl','wb') as f:
    pickle.dump(y_augmented,f)

with open('y_valid.pkl','wb') as f:
    pickle.dump(y_valid,f)
```

Carga de modelo serializado

In [30]:
```
import pickle
with open('X_augmented.pkl', 'rb') as f:
    X_augmented = pickle.load(f)

with open('y_augmented.pkl', 'rb') as f:
    y_augmented = pickle.load(f)
```

In [31]:
```
###probamos los datos
ax=pd.Series(list(y_augmented)).value_counts().plot(kind='bar',figsize=(14,8),title='Elementos por clase')
ax.set_xlabel('Clases')
ax.set_ylabel('Frecuencia')
```

Out[31]: Text(0, 0.5, 'Frecuencia')



Normalizamos los datos

In [32]:
```
X_agumented=X_augmented/255.0
```

Convertimos etiquetas

In [33]:
```
y_augmented_labels=to_categorical(y_augmented)
```

Preparación del modelo principal

In [34]:
```
modelo_Marco = Sequential([
    # La capa de entrada es una imagen de 60x60 pixeles en tres canales R,G,B
    Conv2D(32, 3 , activation='relu', input_shape= (60, 60, 3)),
    MaxPooling2D(2),
    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D((2,2)),
    Conv2D(128, (3,3), activation='relu'),
```

```
    MaxPooling2D(2,2),
    Conv2D(256, (3,3), activation='relu'),
    MaxPooling2D(2,2),
    Dropout(0.3),
    # Capa Flatten
    Flatten(),
    Dense(256, activation='relu'),
    #usamos 18 porque son esa la cantidad de clases y softmax ya que es multiclase
    Dense(18, activation='softmax')
])
```

In [35]:
```
modelo_Marco.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 58, 58, 32)        896

 max_pooling2d (MaxPooling2D  (None, 29, 29, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 27, 27, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 13, 13, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 11, 11, 128)       73856

 max_pooling2d_2 (MaxPooling  (None, 5, 5, 128)        0
 2D)

 conv2d_3 (Conv2D)           (None, 3, 3, 256)         295168

 max_pooling2d_3 (MaxPooling  (None, 1, 1, 256)        0
 2D)

 dropout (Dropout)           (None, 1, 1, 256)         0

 flatten (Flatten)           (None, 256)               0

 dense (Dense)               (None, 256)               65792

 dense_1 (Dense)             (None, 18)                4626

=================================================================
Total params: 458,834
Trainable params: 458,834
Non-trainable params: 0
_________________________________________________________________
```

In [36]:
```
from tensorflow.keras.optimizers import RMSprop
```

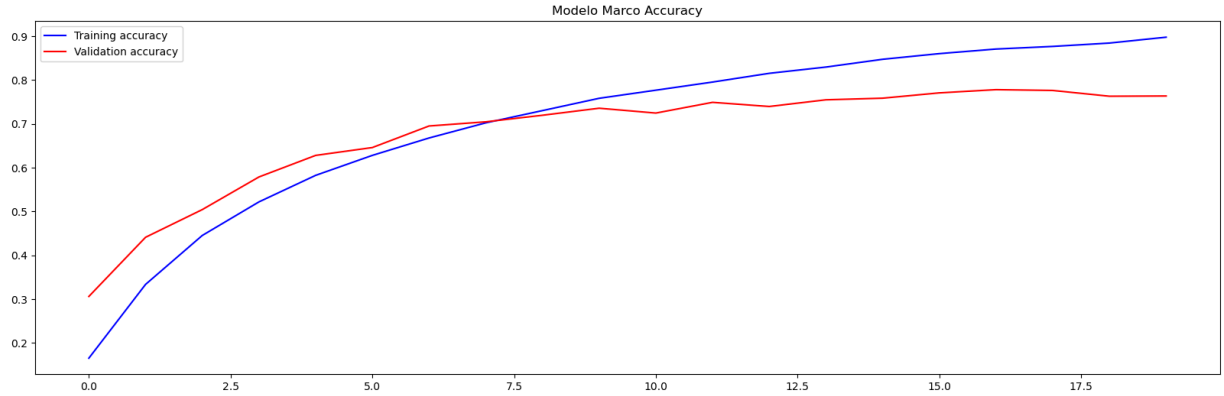
In [37]:
```
modelo_Marco.compile(optimizer=RMSprop(learning_rate=0.0001),loss='sparse_categorical_crossentropy',metric
```

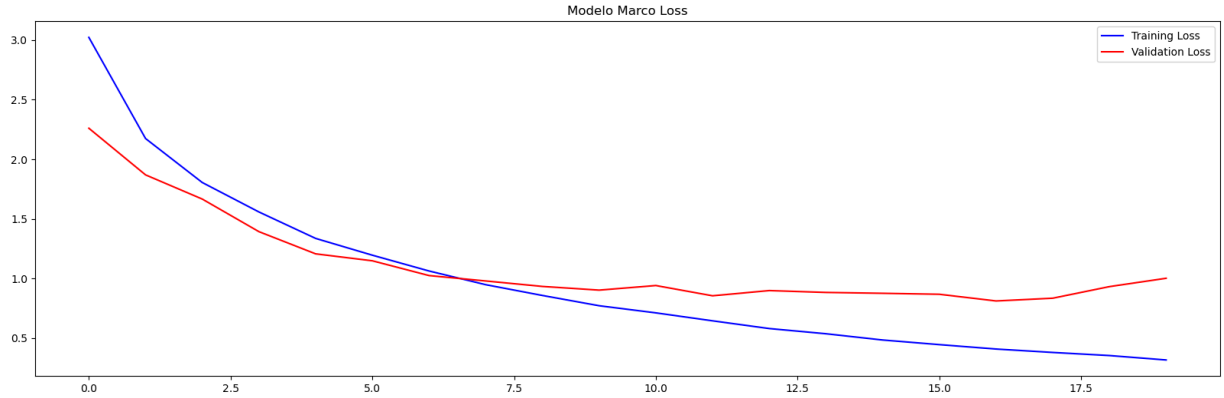
In [38]:
```
history = modelo_Marco.fit(X_augmented, y_augmented,
                    validation_data=(X_valid,y_valid),
                    epochs=20,
                    batch_size=32,
                    verbose=2

                    )
```

```
Epoch 1/20
534/534 - 11s - loss: 3.0231 - accuracy: 0.1653 - val_loss: 2.2607 - val_accuracy: 0.3061 - 11s/epoch - 21m
s/step
Epoch 2/20
534/534 - 4s - loss: 2.1743 - accuracy: 0.3336 - val_loss: 1.8694 - val_accuracy: 0.4412 - 4s/epoch - 8ms/s
tep
Epoch 3/20
534/534 - 4s - loss: 1.8054 - accuracy: 0.4453 - val_loss: 1.6665 - val_accuracy: 0.5043 - 4s/epoch - 8ms/s
tep
Epoch 4/20
534/534 - 4s - loss: 1.5581 - accuracy: 0.5219 - val_loss: 1.3929 - val_accuracy: 0.5788 - 4s/epoch - 8ms/s
tep
Epoch 5/20
534/534 - 4s - loss: 1.3363 - accuracy: 0.5825 - val_loss: 1.2067 - val_accuracy: 0.6281 - 4s/epoch - 8ms/s
tep
Epoch 6/20
534/534 - 4s - loss: 1.1955 - accuracy: 0.6281 - val_loss: 1.1484 - val_accuracy: 0.6460 - 4s/epoch - 8ms/s
tep
Epoch 7/20
534/534 - 4s - loss: 1.0629 - accuracy: 0.6678 - val_loss: 1.0245 - val_accuracy: 0.6952 - 4s/epoch - 8ms/s
tep
Epoch 8/20
534/534 - 4s - loss: 0.9479 - accuracy: 0.7023 - val_loss: 0.9791 - val_accuracy: 0.7047 - 4s/epoch - 8ms/s
tep
Epoch 9/20
534/534 - 4s - loss: 0.8575 - accuracy: 0.7303 - val_loss: 0.9333 - val_accuracy: 0.7197 - 4s/epoch - 8ms/s
tep
Epoch 10/20
534/534 - 4s - loss: 0.7713 - accuracy: 0.7584 - val_loss: 0.9023 - val_accuracy: 0.7357 - 4s/epoch - 8ms/s
tep
Epoch 11/20
534/534 - 4s - loss: 0.7115 - accuracy: 0.7769 - val_loss: 0.9408 - val_accuracy: 0.7247 - 4s/epoch - 8ms/s
tep
Epoch 12/20
534/534 - 4s - loss: 0.6448 - accuracy: 0.7954 - val_loss: 0.8547 - val_accuracy: 0.7491 - 4s/epoch - 8ms/s
tep
Epoch 13/20
534/534 - 4s - loss: 0.5799 - accuracy: 0.8154 - val_loss: 0.8986 - val_accuracy: 0.7397 - 4s/epoch - 8ms/s
tep
Epoch 14/20
534/534 - 4s - loss: 0.5363 - accuracy: 0.8296 - val_loss: 0.8829 - val_accuracy: 0.7549 - 4s/epoch - 8ms/s
tep
Epoch 15/20
534/534 - 4s - loss: 0.4838 - accuracy: 0.8474 - val_loss: 0.8758 - val_accuracy: 0.7586 - 4s/epoch - 8ms/s
tep
Epoch 16/20
534/534 - 4s - loss: 0.4450 - accuracy: 0.8603 - val_loss: 0.8676 - val_accuracy: 0.7707 - 4s/epoch - 8ms/s
tep
Epoch 17/20
534/534 - 4s - loss: 0.4081 - accuracy: 0.8708 - val_loss: 0.8114 - val_accuracy: 0.7781 - 4s/epoch - 8ms/s
tep
Epoch 18/20
534/534 - 4s - loss: 0.3793 - accuracy: 0.8768 - val_loss: 0.8345 - val_accuracy: 0.7763 - 4s/epoch - 8ms/s
tep
Epoch 19/20
534/534 - 4s - loss: 0.3533 - accuracy: 0.8844 - val_loss: 0.9318 - val_accuracy: 0.7631 - 4s/epoch - 8ms/s
tep
Epoch 20/20
534/534 - 4s - loss: 0.3162 - accuracy: 0.8978 - val_loss: 1.0018 - val_accuracy: 0.7636 - 4s/epoch - 8ms/s
tep
```

Inicio del análisis del modelo

In [39]:
```
PlotPerformance(history.history['accuracy'],
                history.history['val_accuracy'],
                history.history['loss'],
                history.history['val_loss'],
                'Modelo Marco')
```





Análisis de gráficos

Se puede apreciar que el Training loss y el validation loss bajan juntas, como curvas suaves, esto es una señal positiva. Indica que el modelo está aprendiendo bien y que la capacidad de generalización del modelo es buena. No hay evidencia significativa de sobreajuste (overfitting) o subajuste (underfitting) aunque en el gráfico, en las últimas épocas se comienza a separar. A pesar de esto, las predicciones tienen un 90% de precisión.

El training accuracy y el validation accuracy van creciendo juntas, puede interpretarse como una señal positiva sobre el rendimiento y la capacidad de generalización del modelo. El hecho de que ambas precisiones crezcan juntas sugiere que el modelo está aprendiendo correctamente los patrones en los datos de entrenamiento y es capaz de aplicar ese conocimiento de manera efectiva a los datos no vistos (conjunto de validación). El crecimiento conjunto de ambas precisiones indica que el modelo no solo está ajustándose bien a los datos de entrenamiento, sino que también está generalizando bien. Esto significa que el modelo está siendo capaz de identificar patrones subyacentes en los datos en lugar de memorizar ejemplos específicos del conjunto de entrenamiento. Al final del gráfico, en las últimas épocas se ve que comienzan a separarse, lo cual pudo generar sobreajuste, sin embargo al detener el entrenamiento, se evitó, por lo que se ha logrado una buena precisión del modelo.

In [40]:
```
modelo_Marco.save('modelo_Marco2.h5', save_format='h5')
# model.save('number_model', save_format='tf')
# model.save('number_model.keras')
```

Evaluación del modelo, superando el 89% de accuracy

In [41]:
```
modelo_Marco.evaluate(X_t, y_t)
```

```
28/28 [==============================] - 0s 14ms/step - loss: 0.3794 - accuracy: 0.9011
```

Out[41]: [0.3793521821498871, 0.901123583316803]

Matriz de confusión para análisis visual

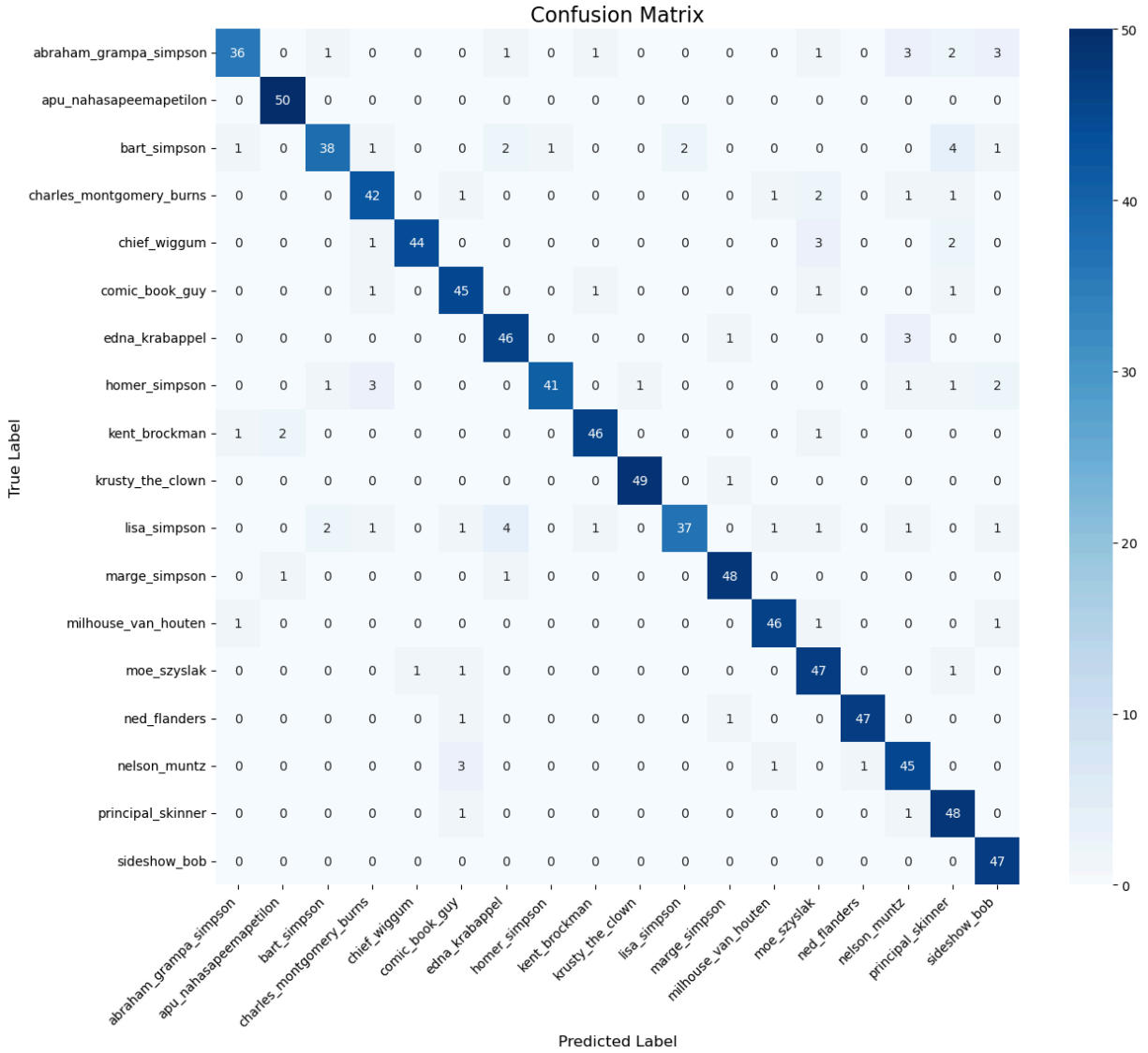
In [42]:
```
import numpy as np

class_label = ['abraham_grampa_simpson', 'apu_nahasapeemapetilon','bart_simpson',
    'charles_montgomery_burns', 'chief_wiggum', 'comic_book_guy', 'edna_krabappel',
```

```
    'homer_simpson', 'kent_brockman', 'krusty_the_clown',  'lisa_simpson',
    'marge_simpson',  'milhouse_van_houten',  'moe_szyslak',
    'ned_flanders', 'nelson_muntz',  'principal_skinner',  'sideshow_bob']

#Ejercicio realizar las predicciones con X_test
class_= modelo_Marco.predict(X_t)
classes = np.argmax(class_, axis = 1)
```

```
28/28 [==============================] - 0s 11ms/step
```

In [43]:
```
# Calculo de la matriz de confusion con tensorflow
confusion_mtx = tf.math.confusion_matrix(y_t, classes)
```

In [44]:
```
import seaborn as sns
import matplotlib.pyplot as plt
# Plot de la matriz de confusión
plt.figure(figsize=(14, 12))
c = sns.heatmap(confusion_mtx, annot=True, fmt='g', cmap=plt.cm.Blues, xticklabels=class_label, yticklabel

# Rotar las etiquetas de los ejes para mejorar la legibilidad
plt.xticks(rotation=45, ha='right', fontsize=10)
plt.yticks(rotation=0, fontsize=10)

# Títulos y etiquetas
plt.xlabel('Predicted Label', fontsize=12)
plt.ylabel('True Label', fontsize=12)
plt.title('Confusion Matrix', fontsize=16)

# Mostrar el gráfico
plt.show()
```



Análisis visual

Análisis de Precisión por Clase

La precisión por clase se puede determinar observando la diagonal principal de la matriz de confusión.

- Abraham Grampa Simpson:

  Verdaderamente clasificados: 36

  Falsamente clasificados: 1 (como Bart Simpson), 1 (como Comic Book Guy), 1 (como Edna Krabappel), 1 (como Kent Brockman), 1 (como Marge Simpson), 3 (como Moe Szyslak), 2 (como Nelson Muntz), 3 (como Sideshow Bob)

- Apu Nahasapeemapetilon:

  Verdaderamente clasificados: 50

  Falsamente clasificados: No hay clasificaciones falsas registradas.

- Bart Simpson:

  Verdaderamente clasificados: 38

  Falsamente clasificados: 1 (como Abraham Grampa Simpson), 2 (como Kent Brockman), 4 (como Lisa Simpson)

- Charles Montgomery Burns:

  Verdaderamente clasificados: 42

  Falsamente clasificados: 1 (como Bart Simpson), 1 (como Krusty the Clown), 2 (como Moe Szyslak), 1 (como Nelson Muntz)

- Chief Wiggum:

  Verdaderamente clasificados: 44

  Falsamente clasificados: 3 (como Milhouse Van Houten)

- Comic Book Guy:

  Verdaderamente clasificados: 45

  Falsamente clasificados: 1 (como Abraham Grampa Simpson), 1 (como Charles Montgomery Burns), 3 (como Moe Szyslak)

- Edna Krabappel:

  Verdaderamente clasificados: 46

  Falsamente clasificados: 3 (como Marge Simpson)

- Homer Simpson:

  Verdaderamente clasificados: 41

  Falsamente clasificados: 3 (como Bart Simpson), 1 (como Kent Brockman), 1 (como Lisa Simpson), 1 (como Marge Simpson), 2 (como Nelson Muntz)

- Kent Brockman:

  Verdaderamente clasificados: 46

  Falsamente clasificados: 1 (como Charles Montgomery Burns), 1 (como Lisa Simpson), 1 (como Moe Szyslak), 2 (como Nelson Muntz)

- Krusty the Clown:

  Verdaderamente clasificados: 49

Falsamente clasificados: 1 (como Charles Montgomery Burns)

- Lisa Simpson:

  Verdaderamente clasificados: 37

  Falsamente clasificados: 2 (como Bart Simpson), 4 (como Kent Brockman), 1 (como Krusty the Clown), 1 (como Marge Simpson), 1 (como Milhouse Van Houten)

- Marge Simpson:

  Verdaderamente clasificados: 48

  Falsamente clasificados: 1 (como Charles Montgomery Burns), 1 (como Homer Simpson), 1 (como Nelson Muntz)

- Milhouse Van Houten:

  Verdaderamente clasificados: 46

  Falsamente clasificados: 1 (como Chief Wiggum), 1 (como Comic Book Guy)

- Moe Szyslak:

  Verdaderamente clasificados: 47

  Falsamente clasificados: 1 (como Charles Montgomery Burns)

- Ned Flanders:

  Verdaderamente clasificados: 48

  Falsamente clasificados: No hay clasificaciones falsas registradas.

- Nelson Muntz:

  Verdaderamente clasificados: 45

  Falsamente clasificados: 3 (como Edna Krabappel), 1 (como Homer Simpson)

- Principal Skinner:

  Verdaderamente clasificados: 48

  Falsamente clasificados: No hay clasificaciones falsas registradas.

- Sideshow Bob:

  Verdaderamente clasificados: 47

  Falsamente clasificados: No hay clasificaciones falsas registradas.

A continuación el grupo infiere lo sguiente:

- Altas Tasas de Acierto: Las clases como Apu Nahasapeemapetilon, Ned Flanders, y Principal Skinner tienen una alta precisión con 50, 48 y 48 clasificaciones correctas respectivamente y sin clasificaciones incorrectas.
- Mejoras Necesarias: Algunas clases como Abraham Grampa Simpson, Bart Simpson, y Homer Simpson tienen varias clasificaciones incorrectas, lo que indica que el modelo podría necesitar ajustes adicionales o más datos para mejorar la precisión para estas clases.
- Confusión entre Clases: Hay ciertas clases que tienden a ser confundidas con otras, como Bart Simpson siendo confundido con Lisa Simpson y viceversa. Esto podría ser debido a características visuales similares o insuficiencia de datos de entrenamiento específicos para estas clases.

Para calcular los valores verdaderos positivos (TP), falsos positivos (FP), falsos negativos (FN), verdaderos negativos (TN), y el Classification Error Rate (CER) para cada clase, podemos seguir los siguientes pasos:

- Verdaderos Positivos (TP): Son los elementos en la diagonal principal de la matriz de confusión.
- Falsos Positivos (FP): Son los elementos de la columna de la clase, excluyendo la diagonal.
- Falsos Negativos (FN): Son los elementos de la fila de la clase, excluyendo la diagonal.
- Verdaderos Negativos (TN): Son todos los elementos de la matriz, excluyendo la fila y la columna de la clase.
- Classification Error Rate (CER):

Classification Error Rate

Es una métrica que reporta la clasificación errónea del modelo, es decir, es una métrica opuesta al *accuracy*.

$$Classification\ Error\ Rate = \frac{FP + FN}{TP + FP + TN + FN}$$

In [3]:
```
# Pasamos la matriz en un arreglo
confusion_mtx = np.array([
    [36, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 3, 0, 2, 0, 3],
    [0, 50, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [1, 0, 38, 0, 0, 0, 0, 0, 2, 0, 4, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 42, 0, 0, 0, 0, 0, 1, 0, 0, 0, 2, 0, 1, 0, 0],
    [0, 0, 0, 0, 44, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0],
    [1, 0, 0, 0, 0, 45, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 46, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0],
    [0, 0, 1, 0, 0, 0, 0, 41, 1, 0, 1, 1, 0, 0, 0, 2, 0, 0],
    [1, 0, 2, 0, 0, 0, 0, 0, 46, 0, 1, 0, 0, 0, 0, 2, 0, 0],
    [0, 0, 0, 1, 0, 0, 0, 0, 0, 49, 0, 0, 0, 1, 0, 0, 0, 0],
    [0, 0, 2, 0, 0, 0, 0, 0, 0, 1, 37, 1, 1, 0, 0, 0, 1, 0],
    [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 48, 0, 0, 0, 1, 0, 0],
    [0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 46, 0, 0, 0, 0, 0],
    [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 47, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 48, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 3, 0, 0, 1, 0, 45, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 48, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 47]
])

# Inicializamos los contadores para cada clase
results = []

for i, label in enumerate(class_label):
    TP = confusion_mtx[i, i]
    FP = np.sum(confusion_mtx[:, i]) - TP
    FN = np.sum(confusion_mtx[i, :]) - TP
    TN = np.sum(confusion_mtx) - (TP + FP + FN)
    CER = (FP + FN) / (TP + FP + FN + TN)

    results.append([label, TP, FP, FN, TN, CER])

# Convertimos los resultados a un DataFrame
df_results = pd.DataFrame(results, columns=['Clase', 'TP', 'FP', 'FN', 'TN', 'CER'])


print(df_results)

# Identificamos las clases con mayores problemas
df_results_sorted = df_results.sort_values(by='CER', ascending=False)
print("\nClases con mayor Classification Error Rate:")
print(df_results_sorted.head(5))
```

```
                      Clase  TP  FP  FN   TN       CER
0     abraham_grampa_simpson  36   4  11  819  0.017241
1     apu_nahasapeemapetilon  50   0   0  820  0.000000
2               bart_simpson  38   6   7  819  0.014943
3   charles_montgomery_burns  42   2   4  822  0.006897
4               chief_wiggum  44   1   3  822  0.004598
5             comic_book_guy  45   0   4  821  0.004598
6             edna_krabappel  46   1   3  820  0.004598
7              homer_simpson  41   2   6  821  0.009195
8              kent_brockman  46   4   6  814  0.011494
9           krusty_the_clown  49   2   2  817  0.004598
10              lisa_simpson  37  13   6  814  0.021839
11             marge_simpson  48   2   3  817  0.005747
12       milhouse_van_houten  46   4   3  817  0.008046
13               moe_szyslak  47  10   2  811  0.013793
14             ned_flanders  48   0   0  822  0.000000
15             nelson_muntz  45  11   5  809  0.018391
16         principal_skinner  48   2   1  819  0.003448
17              sideshow_bob  47   3   1  819  0.004598

Clases con mayor Classification Error Rate:
                     Clase  TP  FP  FN   TN       CER
10            lisa_simpson  37  13   6  814  0.021839
15            nelson_muntz  45  11   5  809  0.018391
0   abraham_grampa_simpson  36   4  11  819  0.017241
2             bart_simpson  38   6   7  819  0.014943
13             moe_szyslak  47  10   2  811  0.013793
```

Estos resultados sugieren que las clases "abraham_grampa_simpson", "homer_simpson", y "lisa_simpson" presentan los mayores problemas en términos de clasificación y podrían beneficiarse de mejoras en el modelo o más datos de entrenamiento específicos para esas clases.

En las siguientes imágenes se pueden ver algunas predicciones, mostrando que por ejemplo en el abuelo Abraham se cometen ciertos errores de predicción, apoyando al análisis. True 0 significa la primera clase, que es el abuelo Abraham

In [50]:
```
# Tamaño de la figura
w = 9
h = 9

# Número de columnas y filas del subplot
cols = 3
rows = 5

# Creación de la figura
fig = plt.figure(figsize=(w, h), constrained_layout=True)

for i in range(1, cols*rows+1):
  img = X_t[i]
  ax = fig.add_subplot(rows, cols, i)
  plt.imshow(img, cmap="gray")
  ax.title.set_text('True ' + str(y_t[i])+' - Pred '+str(class_label[classes[i]]))
plt.show()
```

Análisis de datos: precision y recall

In [45]:
```
from sklearn.metrics import classification_report
print(classification_report(y_t, classes, target_names=class_label))
```

	precision	recall	f1-score	support
abraham_grampa_simpson	0.92	0.75	0.83	48
apu_nahasapeemapetilon	0.94	1.00	0.97	50
bart_simpson	0.90	0.76	0.83	50
charles_montgomery_burns	0.86	0.88	0.87	48
chief_wiggum	0.98	0.88	0.93	50
comic_book_guy	0.85	0.92	0.88	49
edna_krabappel	0.85	0.92	0.88	50
homer_simpson	0.98	0.82	0.89	50
kent_brockman	0.94	0.92	0.93	50
krusty_the_clown	0.98	0.98	0.98	50
lisa_simpson	0.95	0.74	0.83	50
marge_simpson	0.94	0.96	0.95	50
milhouse_van_houten	0.94	0.94	0.94	49
moe_szyslak	0.82	0.94	0.88	50
ned_flanders	0.98	0.96	0.97	49
nelson_muntz	0.82	0.90	0.86	50
principal_skinner	0.80	0.96	0.87	50
sideshow_bob	0.85	1.00	0.92	47
accuracy			0.90	890
macro avg	0.91	0.90	0.90	890
weighted avg	0.91	0.90	0.90	890

El análisis de los valores de precision y recall, junto con el f1-score, permitió al grupo evaluar el rendimiento del modelo. Aquí está el análisis detallado de los resultados:

Hay que recordar que Precision mide la proporción de verdaderos positivos entre el total de predicciones positivas. Alta precisión significa que pocos ejemplos clasificados como positivos son realmente negativos.

El Recall mide la proporción de verdaderos positivos entre el total de ejemplos que realmente son positivos. Alto recall significa que el modelo detecta la mayoría de los ejemplos positivos.

Y el F1-Score es la media armónica de la precisión y el recall. Es útil cuando se necesita un balance entre precisión y recall.

Análisis por Clase

- abraham_grampa_simpson:

  Precision: 0.92 Recall: 0.75 F1-Score: 0.83

  Análisis: Alta precisión pero relativamente bajo recall. El modelo identifica bien los positivos, pero pierde algunos positivos reales.

- apu_nahasapeemapetilon:

  Precision: 0.94 Recall: 1.00 F1-Score: 0.97

  Análisis: Excelente rendimiento con alta precisión y recall perfecto. El modelo identifica casi todos los positivos correctamente.

- bart_simpson:

  Precision: 0.90 Recall: 0.76 F1-Score: 0.83

  Análisis: Alta precisión pero bajo recall. Similar a "abraham_grampa_simpson", el modelo pierde algunos positivos reales.

- charles_montgomery_burns:

  Precision: 0.86 Recall: 0.88 F1-Score: 0.87

  Análisis: Buen balance entre precisión y recall. El modelo tiene un rendimiento sólido para esta clase.

- chief_wiggum:

  Precision: 0.98 Recall: 0.88 F1-Score: 0.93

Análisis: Muy alta precisión y buen recall. El modelo es confiable para esta clase, aunque podría mejorar ligeramente en recall.

- comic_book_guy:

  Precision: 0.85 Recall: 0.92 F1-Score: 0.88

  Análisis: Buena precisión y recall alto. El modelo identifica la mayoría de los positivos correctamente.

- edna_krabappel:

  Precision: 0.85 Recall: 0.92 F1-Score: 0.88

  Análisis: Similar a "comic_book_guy". Buen rendimiento con una ligera mejora posible en precisión.

- homer_simpson:

  Precision: 0.98 Recall: 0.82 F1-Score: 0.89

  Análisis: Muy alta precisión pero un recall menor. El modelo identifica bien los positivos, pero pierde algunos casos reales.

- kent_brockman:

  Precision: 0.94 Recall: 0.92 F1-Score: 0.93

  Análisis: Buen balance entre precisión y recall. Rendimiento sólido y confiable.

- krusty_the_clown:

  Precision: 0.98 Recall: 0.98 F1-Score: 0.98

  Análisis: Excelente rendimiento con alta precisión y recall. El modelo identifica casi todos los positivos correctamente.

- lisa_simpson:

  Precision: 0.95 Recall: 0.74 F1-Score: 0.83

  Análisis: Muy alta precisión pero bajo recall. Similar a "abraham_grampa_simpson", el modelo pierde algunos positivos reales.

- marge_simpson:

  Precision: 0.94 Recall: 0.96 F1-Score: 0.95

  Análisis: Muy buen rendimiento con alta precisión y recall. El modelo es confiable para esta clase.

- milhouse_van_houten:

  Precision: 0.94 Recall: 0.94 F1-Score: 0.94

  Análisis: Muy buen rendimiento con alta precisión y recall. El modelo es confiable para esta clase.

- moe_szyslak:

  Precision: 0.82 Recall: 0.94 F1-Score: 0.88

  Análisis: Buena precisión y alto recall. El modelo identifica la mayoría de los positivos correctamente.

- ned_flanders:

  Precision: 0.98 Recall: 0.96 F1-Score: 0.97

  Análisis: Excelente rendimiento con alta precisión y recall. El modelo identifica casi todos los positivos correctamente.

- nelson_muntz:

  Precision: 0.82 Recall: 0.90 F1-Score: 0.86

Análisis: Buena precisión y alto recall. El modelo identifica la mayoría de los positivos correctamente.

- principal_skinner:

  Precision: 0.80 Recall: 0.96 F1-Score: 0.87

  Análisis: Baja precisión pero alto recall. El modelo detecta la mayoría de los positivos, pero también genera más falsos positivos.

- sideshow_bob:

  Precision: 0.85 Recall: 1.00 F1-Score: 0.92

  Análisis: Alta precisión y recall perfecto. El modelo identifica todos los positivos correctamente.

- Promedios

Accuracy: 0.90

El modelo tiene una precisión global del 90%, lo cual es muy bueno.

Conclusiones

- El modelo tiene un rendimiento sólido en general con una precisión y recall global altos.
- Clases como apu_nahasapeemapetilon, krusty_the_clown, ned_flanders y sideshow_bob tienen un rendimiento excelente con alta precisión y recall.
- Clases como abraham_grampa_simpson, bart_simpson y lisa_simpson tienen un recall más bajo, lo que indica que el modelo no está detectando algunos positivos reales en estas clases.
- Es importante mejorar el recall en las clases con bajos valores de recall sin comprometer demasiado la precisión. Mejorar el recall para las clases con problemas puede implicar ajustar el modelo, recopilar más datos de entrenamiento para esas clases específicas o aplicar técnicas de balanceo de datos.

In [ ]: