In [1]:
```
from tensorflow.keras.datasets import mnist
import math, time
import matplotlib.pyplot as plt
import numpy as np
#!pip install seaborn
import seaborn as sns
%matplotlib inline
import pandas as pd
from sklearn.model_selection import train_test_split
import pickle
from PIL import Image

from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from imblearn.metrics import specificity_score
from matplotlib import*
from matplotlib.cm import register_cmap
import matplotlib.pyplot as plt

import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D
#from keras.layers.normalization import BatchNormalization
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ReduceLROnPlateau
from keras.models import model_from_json
from keras.models import load_model

from sklearn.svm import SVC #SVR para regresión
from sklearn.metrics import classification_report
from keras import models
from keras.layers import BatchNormalization, MaxPool2D, GlobalMaxPool2D
#Arquitecturas de Transfer learning. Puedes configurar parámetros específicos de ca
from tensorflow.keras.applications.vgg16 import VGG16
#from keras.applications.resnet50 import ResNet50
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.applications.xception import Xception
from keras.applications.inception_resnet_v2 import InceptionResNetV2

import tensorflow as tf
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Input, Flatten, Dense, concatenate, Dropout
from tensorflow.keras.applications import VGG16, Xception
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.layers import Conv2D, MaxPooling2D
import os
from skimage import io
from sklearn.model_selection import train_test_split
import os
import cv2
import numpy as np
import os
import cv2
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report
from sklearn.preprocessing import LabelEncoder
```

```
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix, recall_score,
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, BatchNormalization, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.utils import to_categorical

print('Módulos importados')
```

```
D:\anaconda3\Lib\site-packages\paramiko\transport.py:219: CryptographyDeprecationW
arning: Blowfish has been deprecated
  "class": algorithms.Blowfish,
Módulos importados
```

In [2]:

```
# Funciones para guardar y cargar objetos pickle
def guardarObjeto(pipeline,nombreArchivo):
    print("Guardando Objeto en Archivo")
    with open(nombreArchivo+'.pickle', 'wb') as handle:
        pickle.dump(pipeline, handle, protocol=pickle.HIGHEST_PROTOCOL)
        print("Objeto Guardado en Archivo")
def cargarObjeto(nombreArchivo):
    with open(nombreArchivo+'.pickle', 'rb') as handle:
        pipeline = pickle.load(handle)
        print("Objeto Cargado desde Archivo")
    return pipeline
# Funciones para guardar y cargar la Red Neuronal (Arquitectura y Pesos)
def guardarNN(model,nombreArchivo):
    print("Guardando Red Neuronal en Archivo")
    model.save(nombreArchivo+'.h5')
    print("Red Neuronal Guardada en Archivo")

def cargarNN(nombreArchivo):
    model = load_model(nombreArchivo+'.h5')
    print("Red Neuronal Cargada desde Archivo")
    return model

# Función para medir la calidad de modelos
def obtenerResultados(y_test, y_pred):
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='macro')
    recall = recall_score(y_test, y_pred, average='macro')
    f1 = f1_score(y_test, y_pred, average='macro')
    specificity = specificity_score(y_test, y_pred, average='macro')

    accuracy=str(round(accuracy, 4))
    precision=str(round(precision, 4))
    recall=str(round(recall, 4))
    f1=str(round(f1, 4))
    specificity=str(round(specificity, 4))

    print("Accuracy:", accuracy)
    print("Precision:", precision)
    print("Recall o Sensitivity:", recall)
    print("F1-Score:", f1)
    print("Specificity:", specificity)
```

```
    return accuracy, precision, recall, f1, specificity
print('Funciones para guardar y cargar modelos personalizados')
```

```
Funciones para guardar y cargar modelos personalizados
```

In [3]:
```
#Importar dataset
from keras.datasets import cifar100
# Cargar el conjunto de datos CIFAR-100
(x_train, y_train), (x_test, y_test) = cifar100.load_data(label_mode='fine')
```

In [4]:
```
import random

# Obtener las categorías únicas y convertirlas a una lista
categorias_unicas = list(set(y_train.flatten()))

# Seleccionar 5 categorías al azar
categorias_al_azar = random.sample(categorias_unicas, 5)

# Inicializar un diccionario para contar los datos en cada categoría
conteo_por_categoria = {categoria: 0 for categoria in categorias_al_azar}

# Contar los datos en cada categoría
for etiqueta in y_train.flatten():
    if etiqueta in categorias_al_azar:
        conteo_por_categoria[etiqueta] += 1

# Mostrar el conteo por categoría
print("Conteo de datos por categoría:")
for categoria, conteo in conteo_por_categoria.items():
    print(f"{categoria}: {conteo} datos")
```

```
Conteo de datos por categoría:
94: 500 datos
54: 500 datos
70: 500 datos
75: 500 datos
9: 500 datos
```

In [5]:
```
# Obtener nombres de las etiquetas
label_names = [
    'apple', 'aquarium_fish', 'baby', 'bear', 'beaver', 'bed', 'bee', 'beetle', 'bi
    'bowl', 'boy', 'bridge', 'bus', 'butterfly', 'camel', 'can', 'castle', 'caterpi
    'chair', 'chimpanzee', 'clock', 'cloud', 'cockroach', 'couch', 'crab', 'crocodi
    'dolphin', 'elephant', 'flatfish', 'forest', 'fox', 'girl', 'hamster', 'house',
    'lamp', 'lawn_mower', 'leopard', 'lion', 'lizard', 'lobster', 'man', 'maple_tre
    'mouse', 'mushroom', 'oak_tree', 'orange', 'orchid', 'otter', 'palm_tree', 'pea
    'plain', 'plate', 'poppy', 'porcupine', 'possum', 'rabbit', 'raccoon', 'ray', '
    'rose', 'sea', 'seal', 'shark', 'shrew', 'skunk', 'skyscraper', 'snail', 'snake
    'squirrel', 'streetcar', 'sunflower', 'sweet_pepper', 'table', 'tank', 'telepho
    'train', 'trout', 'tulip', 'turtle', 'wardrobe', 'whale', 'willow_tree', 'wolf'
]

# Seleccionar aleatoriamente algunas imágenes para visualizar
num_images_to_display = 5
random_indices = random.sample(range(len(x_train)), num_images_to_display)

# Mostrar las imágenes seleccionadas
plt.figure(figsize=(15, 5))
for i, idx in enumerate(random_indices):
    plt.subplot(1, num_images_to_display, i + 1)
    plt.imshow(x_train[idx])
    plt.title(label_names[y_train[idx][0]])
    plt.axis('off')
```

```
plt.show()
```

In [6]:

```
import numpy as np
from sklearn.model_selection import train_test_split

# Fijar la semilla para reproducibilidad
np.random.seed(42)

# Realizar muestreo para obtener un total de 5000 datos en el conjunto de entrenami
x_train_sample, _, y_train_sample, _ = train_test_split(x_train, y_train, train_siz
x_test_sample, _, y_test_sample, _ = train_test_split(x_test, y_test, train_size=35

# Obtener las categorías únicas y convertirlas a una lista
categorias_unicas_train = list(set(y_train_sample.flatten()))

# Inicializar un diccionario para contar los datos en cada categoría en el conjunto
conteo_por_categoria_train = {categoria: 0 for categoria in categorias_unicas_train

# Contar los datos en cada categoría en el conjunto de entrenamiento
for etiqueta in y_train_sample.flatten():
    conteo_por_categoria_train[etiqueta] += 1

# Mostrar el conteo por categoría en el conjunto de entrenamiento
print("Conteo de datos por categoría en el conjunto de entrenamiento:")
for categoria, conteo in conteo_por_categoria_train.items():
    print(f"{label_names[categoria]}: {conteo} datos")

# Obtener las categorías únicas en el conjunto de prueba y convertirlas a una lista
categorias_unicas_test = list(set(y_test_sample.flatten()))

# Inicializar un diccionario para contar los datos en cada categoría en el conjunto
conteo_por_categoria_test = {categoria: 0 for categoria in categorias_unicas_test}

# Contar los datos en cada categoría en el conjunto de prueba
for etiqueta in y_test_sample.flatten():
    conteo_por_categoria_test[etiqueta] += 1

# Mostrar el conteo por categoría en el conjunto de prueba
print("\nConteo de datos por categoría en el conjunto de prueba:")
for categoria, conteo in conteo_por_categoria_test.items():
    print(f"{label_names[categoria]}: {conteo} datos")
```

```
Conteo de datos por categoría en el conjunto de entrenamiento:
apple: 175 datos
aquarium_fish: 175 datos
baby: 175 datos
bear: 175 datos
beaver: 175 datos
bed: 175 datos
bee: 175 datos
beetle: 175 datos
bicycle: 175 datos
bottle: 175 datos
bowl: 175 datos
boy: 175 datos
bridge: 175 datos
bus: 175 datos
butterfly: 175 datos
camel: 175 datos
can: 175 datos
castle: 175 datos
caterpillar: 175 datos
cattle: 175 datos
chair: 175 datos
chimpanzee: 175 datos
clock: 175 datos
cloud: 175 datos
cockroach: 175 datos
couch: 175 datos
crab: 175 datos
crocodile: 175 datos
cup: 175 datos
dinosaur: 175 datos
dolphin: 175 datos
elephant: 175 datos
flatfish: 175 datos
forest: 175 datos
fox: 175 datos
girl: 175 datos
hamster: 175 datos
house: 175 datos
kangaroo: 175 datos
keyboard: 175 datos
lamp: 175 datos
lawn_mower: 175 datos
leopard: 175 datos
lion: 175 datos
lizard: 175 datos
lobster: 175 datos
man: 175 datos
maple_tree: 175 datos
motorcycle: 175 datos
mountain: 175 datos
mouse: 175 datos
mushroom: 175 datos
oak_tree: 175 datos
orange: 175 datos
orchid: 175 datos
otter: 175 datos
palm_tree: 175 datos
pear: 175 datos
pickup_truck: 175 datos
pine_tree: 175 datos
plain: 175 datos
plate: 175 datos
poppy: 175 datos
```

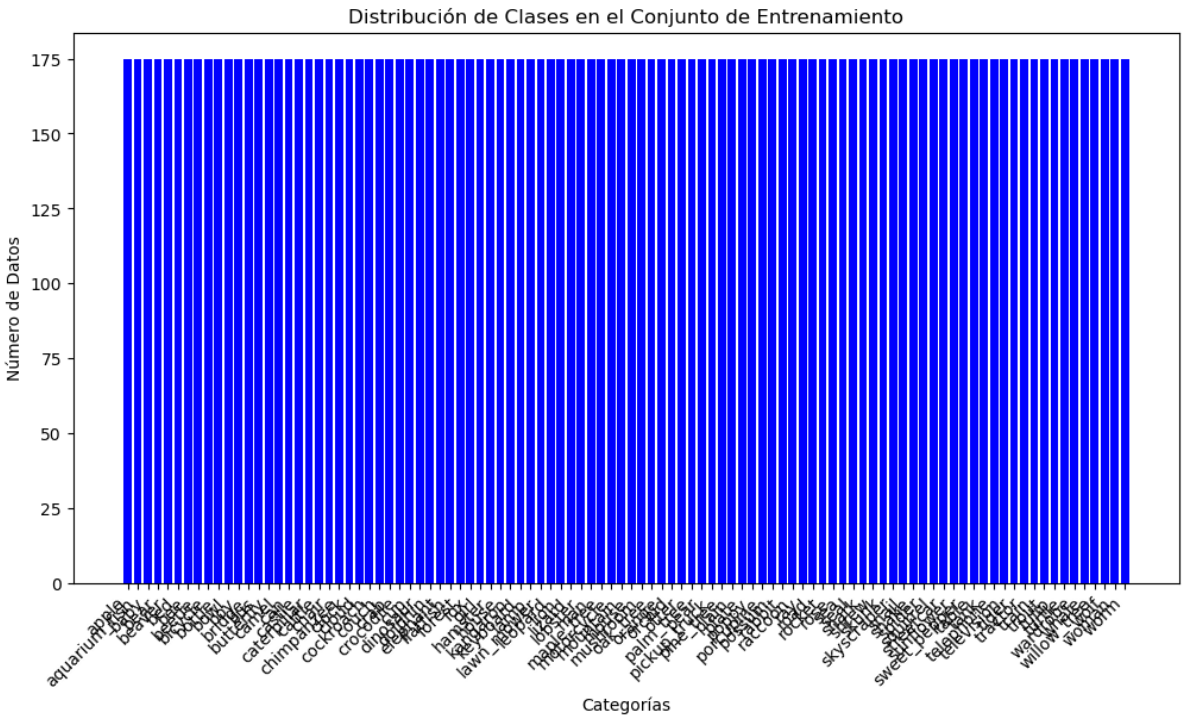
```
porcupine: 175 datos
possum: 175 datos
rabbit: 175 datos
raccoon: 175 datos
ray: 175 datos
road: 175 datos
rocket: 175 datos
rose: 175 datos
sea: 175 datos
seal: 175 datos
shark: 175 datos
shrew: 175 datos
skunk: 175 datos
skyscraper: 175 datos
snail: 175 datos
snake: 175 datos
spider: 175 datos
squirrel: 175 datos
streetcar: 175 datos
sunflower: 175 datos
sweet_pepper: 175 datos
table: 175 datos
tank: 175 datos
telephone: 175 datos
television: 175 datos
tiger: 175 datos
tractor: 175 datos
train: 175 datos
trout: 175 datos
tulip: 175 datos
turtle: 175 datos
wardrobe: 175 datos
whale: 175 datos
willow_tree: 175 datos
wolf: 175 datos
woman: 175 datos
worm: 175 datos

Conteo de datos por categoría en el conjunto de prueba:
apple: 35 datos
aquarium_fish: 35 datos
baby: 35 datos
bear: 35 datos
beaver: 35 datos
bed: 35 datos
bee: 35 datos
beetle: 35 datos
bicycle: 35 datos
bottle: 35 datos
bowl: 35 datos
boy: 35 datos
bridge: 35 datos
bus: 35 datos
butterfly: 35 datos
camel: 35 datos
can: 35 datos
castle: 35 datos
caterpillar: 35 datos
cattle: 35 datos
chair: 35 datos
chimpanzee: 35 datos
clock: 35 datos
cloud: 35 datos
cockroach: 35 datos
```

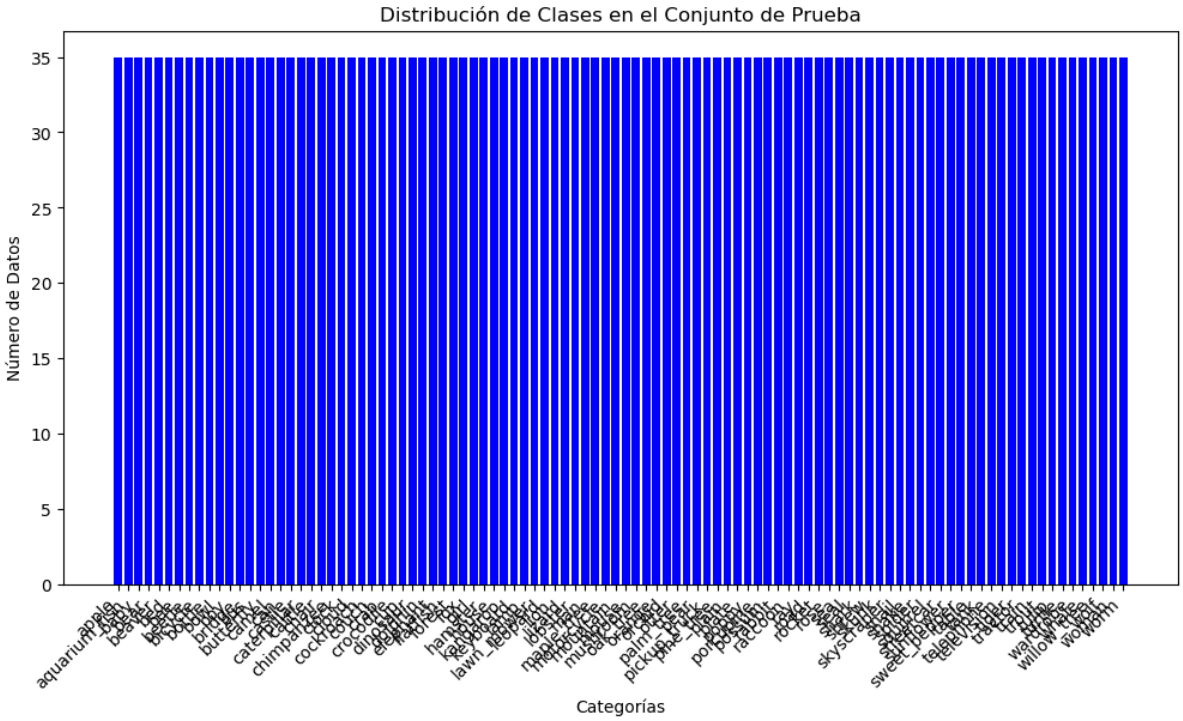
```
couch: 35 datos
crab: 35 datos
crocodile: 35 datos
cup: 35 datos
dinosaur: 35 datos
dolphin: 35 datos
elephant: 35 datos
flatfish: 35 datos
forest: 35 datos
fox: 35 datos
girl: 35 datos
hamster: 35 datos
house: 35 datos
kangaroo: 35 datos
keyboard: 35 datos
lamp: 35 datos
lawn_mower: 35 datos
leopard: 35 datos
lion: 35 datos
lizard: 35 datos
lobster: 35 datos
man: 35 datos
maple_tree: 35 datos
motorcycle: 35 datos
mountain: 35 datos
mouse: 35 datos
mushroom: 35 datos
oak_tree: 35 datos
orange: 35 datos
orchid: 35 datos
otter: 35 datos
palm_tree: 35 datos
pear: 35 datos
pickup_truck: 35 datos
pine_tree: 35 datos
plain: 35 datos
plate: 35 datos
poppy: 35 datos
porcupine: 35 datos
possum: 35 datos
rabbit: 35 datos
raccoon: 35 datos
ray: 35 datos
road: 35 datos
rocket: 35 datos
rose: 35 datos
sea: 35 datos
seal: 35 datos
shark: 35 datos
shrew: 35 datos
skunk: 35 datos
skyscraper: 35 datos
snail: 35 datos
snake: 35 datos
spider: 35 datos
squirrel: 35 datos
streetcar: 35 datos
sunflower: 35 datos
sweet_pepper: 35 datos
table: 35 datos
tank: 35 datos
telephone: 35 datos
television: 35 datos
tiger: 35 datos
```

```
tractor: 35 datos
train: 35 datos
trout: 35 datos
tulip: 35 datos
turtle: 35 datos
wardrobe: 35 datos
whale: 35 datos
willow_tree: 35 datos
wolf: 35 datos
woman: 35 datos
worm: 35 datos
```

In [7]:

```python
import matplotlib.pyplot as plt

# Función para visualizar la distribución de clases
def plot_class_distribution(y, dataset_name):
    unique_classes, counts = np.unique(y, return_counts=True)
    class_labels = [label_names[c] for c in unique_classes]

    plt.figure(figsize=(12, 6))
    plt.bar(class_labels, counts, color='blue')
    plt.title(f'Distribución de Clases en el Conjunto de {dataset_name}')
    plt.xlabel('Categorías')
    plt.ylabel('Número de Datos')
    plt.xticks(rotation=45, ha="right")
    plt.show()

# Visualizar la distribución de clases en el conjunto de entrenamiento
plot_class_distribution(y_train_sample.flatten(), 'Entrenamiento')

# Visualizar la distribución de clases en el conjunto de prueba
plot_class_distribution(y_test_sample.flatten(), 'Prueba')
```





In [8]:
```
# Normalizar X_train y X_test
X_train_normalized = x_train_sample.astype('float32') / 255.0
X_test_normalized = x_test_sample.astype('float32') / 255.0

# Verificar las dimensiones antes y después de la normalización
print("Dimensiones de X_train antes de la normalización:", x_train_sample.shape)
print("Dimensiones de X_test antes de la normalización:", x_test_sample.shape)

print("\nNormalizando...")

print("Dimensiones de X_train después de la normalización:", X_train_normalized.sha
print("Dimensiones de X_test después de la normalización:", X_test_normalized.shape
```

```
Dimensiones de X_train antes de la normalización: (17500, 32, 32, 3)
Dimensiones de X_test antes de la normalización: (3500, 32, 32, 3)

Normalizando...
Dimensiones de X_train después de la normalización: (17500, 32, 32, 3)
Dimensiones de X_test después de la normalización: (3500, 32, 32, 3)
```

In [9]:
```
x_train_sample.shape[1:]
```

Out[9]:
```
(32, 32, 3)
```

In [10]:
```
print('SVM Classifier with gamma = 0.1; Kernel = Polynomial')
classifierSVM = SVC(gamma=0.1, kernel='poly', C=1.0, verbose=True)#random_state = 0
#kernels: "linear", "poly", "rbf" y "sigmoid"
#help(SVC)
#Para regresión: sv_regressor = SVR(kernel='linear', C=1.0, epsilon=0.1)
```

```
SVM Classifier with gamma = 0.1; Kernel = Polynomial
```

In [13]:
```
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Codificar las etiquetas categóricas en valores numéricos
```

```
le = LabelEncoder()
Y = y_train.flatten()
Y_encoded = le.fit_transform(Y)

# Reshape para que cada imagen sea un vector unidimensional
X_flatten = x_train.reshape((x_train.shape[0], -1))

# Dividir el conjunto de datos en conjuntos de entrenamiento y prueba
X_train, X_test, Y_train, Y_test = train_test_split(X_flatten, Y_encoded, test_size

# Crear y entrenar el clasificador SVM
classifierSVM = SVC(kernel='linear', C=1.0)
classifierSVM.fit(X_train, Y_train)

# Realizar predicciones en el conjunto de prueba
Y_pred = classifierSVM.predict(X_test)

# Evaluar el rendimiento del clasificador
accuracy = accuracy_score(Y_test, Y_pred)
conf_matrix = confusion_matrix(Y_test, Y_pred)
classification_rep = classification_report(Y_test, Y_pred)

# Imprimir métricas de rendimiento
print(f'Accuracy: {accuracy:.2f}')
print(f'Confusion Matrix:\n{conf_matrix}')
print(f'Classification Report:\n{classification_rep}')
```

```
Accuracy: 0.16
Confusion Matrix:
[[36  2  4 ...  0  1  0]
 [ 2 20  1 ...  2  1  0]
 [ 3  1 10 ...  2  1  0]
 ...
 [ 0  4  2 ...  7  3  1]
 [ 3  1  4 ...  1 12  0]
 [ 0  1  0 ...  1  0  4]]
Classification Report:
```

	precision	recall	f1-score	support
0	0.21	0.34	0.26	107
1	0.14	0.19	0.16	107
2	0.08	0.11	0.09	88
3	0.05	0.07	0.06	103
4	0.08	0.10	0.09	93
5	0.07	0.09	0.08	105
6	0.07	0.09	0.08	93
7	0.09	0.12	0.11	96
8	0.21	0.14	0.17	87
9	0.18	0.19	0.19	88
10	0.11	0.10	0.10	100
11	0.04	0.04	0.04	100
12	0.10	0.13	0.12	99
13	0.08	0.07	0.07	86
14	0.13	0.13	0.13	100
15	0.05	0.07	0.06	87
16	0.15	0.11	0.13	116
17	0.20	0.30	0.24	91
18	0.11	0.12	0.11	113
19	0.14	0.12	0.13	104
20	0.32	0.39	0.35	107
21	0.16	0.22	0.19	113
22	0.19	0.12	0.15	107
23	0.21	0.34	0.26	99
24	0.30	0.59	0.39	104
25	0.05	0.04	0.04	106
26	0.04	0.02	0.03	104
27	0.06	0.05	0.05	105
28	0.14	0.17	0.15	92
29	0.08	0.05	0.06	99
30	0.30	0.37	0.33	109
31	0.11	0.10	0.11	96
32	0.07	0.04	0.05	94
33	0.09	0.11	0.10	104
34	0.08	0.09	0.08	82
35	0.09	0.09	0.09	98
36	0.25	0.27	0.26	107
37	0.16	0.14	0.15	100
38	0.19	0.14	0.16	109
39	0.11	0.06	0.08	103
40	0.10	0.10	0.10	96
41	0.33	0.31	0.32	104
42	0.04	0.02	0.02	105
43	0.15	0.13	0.14	110
44	0.07	0.05	0.06	87
45	0.01	0.01	0.01	106
46	0.09	0.09	0.09	99
47	0.18	0.28	0.22	103
48	0.18	0.14	0.16	98
49	0.18	0.24	0.21	104
50	0.05	0.03	0.04	105
51	0.12	0.14	0.13	102

52	0.41	0.53	0.46	110
53	0.32	0.45	0.38	95
54	0.21	0.26	0.23	89
55	0.05	0.03	0.04	94
56	0.23	0.23	0.23	105
57	0.16	0.21	0.18	107
58	0.11	0.11	0.11	91
59	0.19	0.19	0.19	106
60	0.32	0.49	0.38	92
61	0.23	0.25	0.24	114
62	0.16	0.17	0.17	104
63	0.10	0.10	0.10	103
64	0.10	0.09	0.09	104
65	0.07	0.06	0.07	99
66	0.02	0.01	0.01	103
67	0.24	0.32	0.27	101
68	0.29	0.42	0.35	93
69	0.20	0.21	0.21	94
70	0.16	0.15	0.15	109
71	0.28	0.31	0.29	95
72	0.03	0.02	0.02	96
73	0.23	0.29	0.25	105
74	0.05	0.04	0.04	114
75	0.12	0.18	0.14	101
76	0.22	0.23	0.23	104
77	0.03	0.02	0.02	103
78	0.00	0.00	0.00	89
79	0.14	0.08	0.10	97
80	0.07	0.06	0.06	88
81	0.09	0.07	0.08	86
82	0.27	0.24	0.25	97
83	0.14	0.13	0.13	110
84	0.06	0.03	0.04	89
85	0.20	0.18	0.19	109
86	0.23	0.20	0.21	101
87	0.14	0.18	0.16	99
88	0.11	0.07	0.08	86
89	0.17	0.14	0.15	101
90	0.08	0.06	0.07	98
91	0.26	0.26	0.26	88
92	0.13	0.11	0.12	116
93	0.05	0.03	0.04	98
94	0.39	0.43	0.41	102
95	0.15	0.18	0.16	90
96	0.21	0.22	0.22	105
97	0.07	0.07	0.07	95
98	0.11	0.10	0.11	117
99	0.12	0.05	0.07	88
accuracy			0.16	10000
macro avg	0.15	0.16	0.15	10000
weighted avg	0.15	0.16	0.15	10000

In [15]:
```
from sklearn.metrics import precision_score, recall_score, f1_score

# Evaluar el rendimiento del clasificador
accuracy = accuracy_score(Y_test, Y_pred)
precision = precision_score(Y_test, Y_pred, average='weighted')
recall = recall_score(Y_test, Y_pred, average='weighted')
f1 = f1_score(Y_test, Y_pred, average='weighted')

# Confusion Matrix y Classification Report
conf_matrix = confusion_matrix(Y_test, Y_pred)
```

```
classification_rep = classification_report(Y_test, Y_pred)

# Imprimir métricas de rendimiento
print(f'Accuracy: {accuracy:.2f}')
print(f'Precision: {precision:.2f}')
print(f'Recall: {recall:.2f}')
print(f'F1-Score: {f1:.2f}')

print(f'Confusion Matrix:\n{conf_matrix}')
print(f'Classification Report:\n{classification_rep}')
```

```
Accuracy: 0.16
Precision: 0.15
Recall: 0.16
F1-Score: 0.15
Confusion Matrix:
[[36  2  4 ...  0  1  0]
 [ 2 20  1 ...  2  1  0]
 [ 3  1 10 ...  2  1  0]
 ...
 [ 0  4  2 ...  7  3  1]
 [ 3  1  4 ...  1 12  0]
 [ 0  1  0 ...  1  0  4]]
Classification Report:
```

	precision	recall	f1-score	support
0	0.21	0.34	0.26	107
1	0.14	0.19	0.16	107
2	0.08	0.11	0.09	88
3	0.05	0.07	0.06	103
4	0.08	0.10	0.09	93
5	0.07	0.09	0.08	105
6	0.07	0.09	0.08	93
7	0.09	0.12	0.11	96
8	0.21	0.14	0.17	87
9	0.18	0.19	0.19	88
10	0.11	0.10	0.10	100
11	0.04	0.04	0.04	100
12	0.10	0.13	0.12	99
13	0.08	0.07	0.07	86
14	0.13	0.13	0.13	100
15	0.05	0.07	0.06	87
16	0.15	0.11	0.13	116
17	0.20	0.30	0.24	91
18	0.11	0.12	0.11	113
19	0.14	0.12	0.13	104
20	0.32	0.39	0.35	107
21	0.16	0.22	0.19	113
22	0.19	0.12	0.15	107
23	0.21	0.34	0.26	99
24	0.30	0.59	0.39	104
25	0.05	0.04	0.04	106
26	0.04	0.02	0.03	104
27	0.06	0.05	0.05	105
28	0.14	0.17	0.15	92
29	0.08	0.05	0.06	99
30	0.30	0.37	0.33	109
31	0.11	0.10	0.11	96
32	0.07	0.04	0.05	94
33	0.09	0.11	0.10	104
34	0.08	0.09	0.08	82
35	0.09	0.09	0.09	98
36	0.25	0.27	0.26	107
37	0.16	0.14	0.15	100
38	0.19	0.14	0.16	109
39	0.11	0.06	0.08	103
40	0.10	0.10	0.10	96
41	0.33	0.31	0.32	104
42	0.04	0.02	0.02	105
43	0.15	0.13	0.14	110
44	0.07	0.05	0.06	87
45	0.01	0.01	0.01	106
46	0.09	0.09	0.09	99
47	0.18	0.28	0.22	103
48	0.18	0.14	0.16	98

49	0.18	0.24	0.21	104
50	0.05	0.03	0.04	105
51	0.12	0.14	0.13	102
52	0.41	0.53	0.46	110
53	0.32	0.45	0.38	95
54	0.21	0.26	0.23	89
55	0.05	0.03	0.04	94
56	0.23	0.23	0.23	105
57	0.16	0.21	0.18	107
58	0.11	0.11	0.11	91
59	0.19	0.19	0.19	106
60	0.32	0.49	0.38	92
61	0.23	0.25	0.24	114
62	0.16	0.17	0.17	104
63	0.10	0.10	0.10	103
64	0.10	0.09	0.09	104
65	0.07	0.06	0.07	99
66	0.02	0.01	0.01	103
67	0.24	0.32	0.27	101
68	0.29	0.42	0.35	93
69	0.20	0.21	0.21	94
70	0.16	0.15	0.15	109
71	0.28	0.31	0.29	95
72	0.03	0.02	0.02	96
73	0.23	0.29	0.25	105
74	0.05	0.04	0.04	114
75	0.12	0.18	0.14	101
76	0.22	0.23	0.23	104
77	0.03	0.02	0.02	103
78	0.00	0.00	0.00	89
79	0.14	0.08	0.10	97
80	0.07	0.06	0.06	88
81	0.09	0.07	0.08	86
82	0.27	0.24	0.25	97
83	0.14	0.13	0.13	110
84	0.06	0.03	0.04	89
85	0.20	0.18	0.19	109
86	0.23	0.20	0.21	101
87	0.14	0.18	0.16	99
88	0.11	0.07	0.08	86
89	0.17	0.14	0.15	101
90	0.08	0.06	0.07	98
91	0.26	0.26	0.26	88
92	0.13	0.11	0.12	116
93	0.05	0.03	0.04	98
94	0.39	0.43	0.41	102
95	0.15	0.18	0.16	90
96	0.21	0.22	0.22	105
97	0.07	0.07	0.07	95
98	0.11	0.10	0.11	117
99	0.12	0.05	0.07	88
accuracy			0.16	10000
macro avg	0.15	0.16	0.15	10000
weighted avg	0.15	0.16	0.15	10000

In [14]:
```
guardarObjeto(classifierSVM,'CIFAR100classififierSVM')
```

```
Guardando Objeto en Archivo
Objeto Guardado en Archivo
```