



DATA SCIENCE PROJECT

CONTENT

- 01** CONJUNTO DE DATOS
- 02** PROCESAMIENTO DE DATOS
- 03** TARGET Y MISSING VALUES
- 04** MODELOS
- 05** RESULTADOS
- 06** TECNICAS
- 07** RESULTADO FINAL

DATOS:

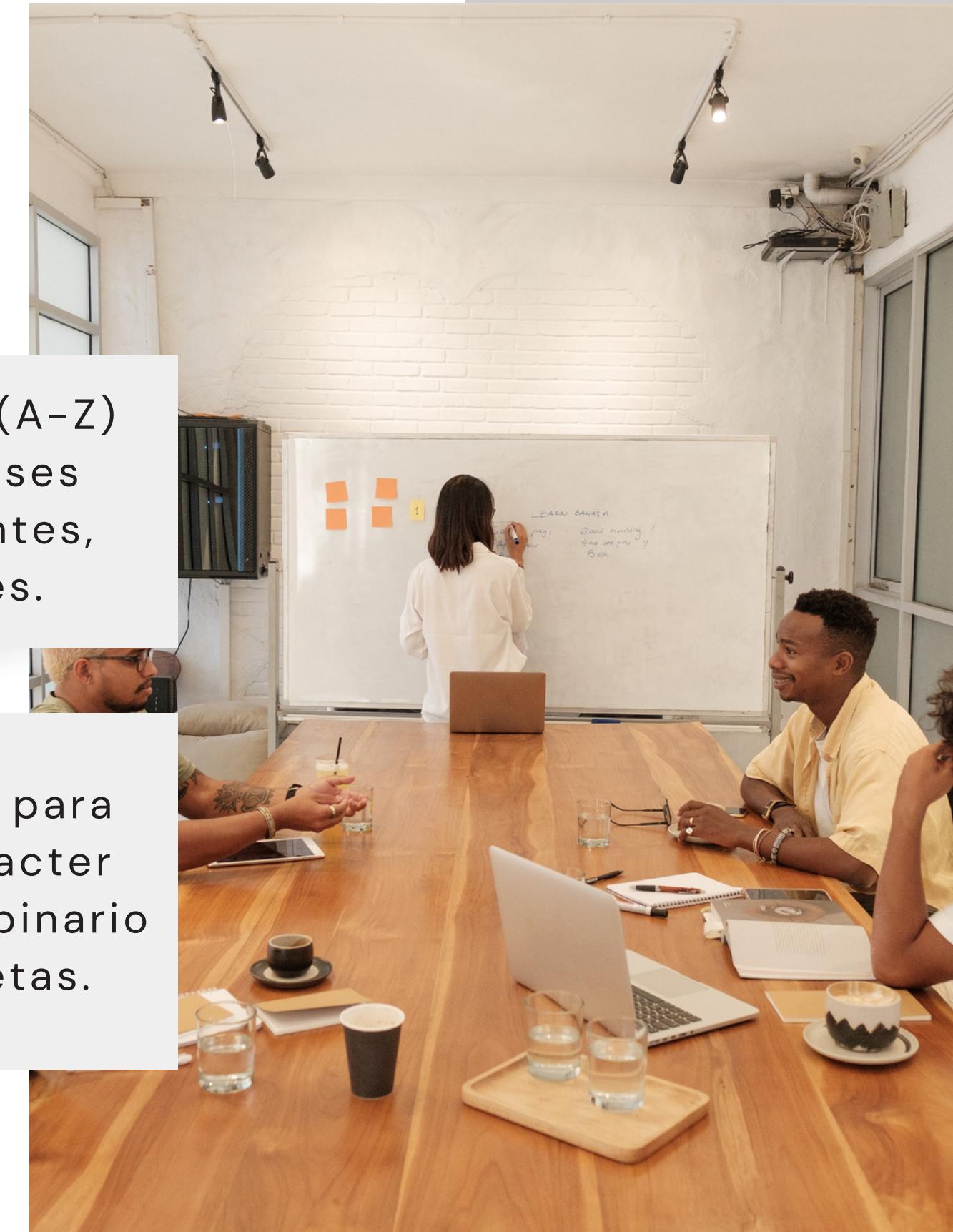
Conjunto de datos de fuentes de caracteres alfabéticos.



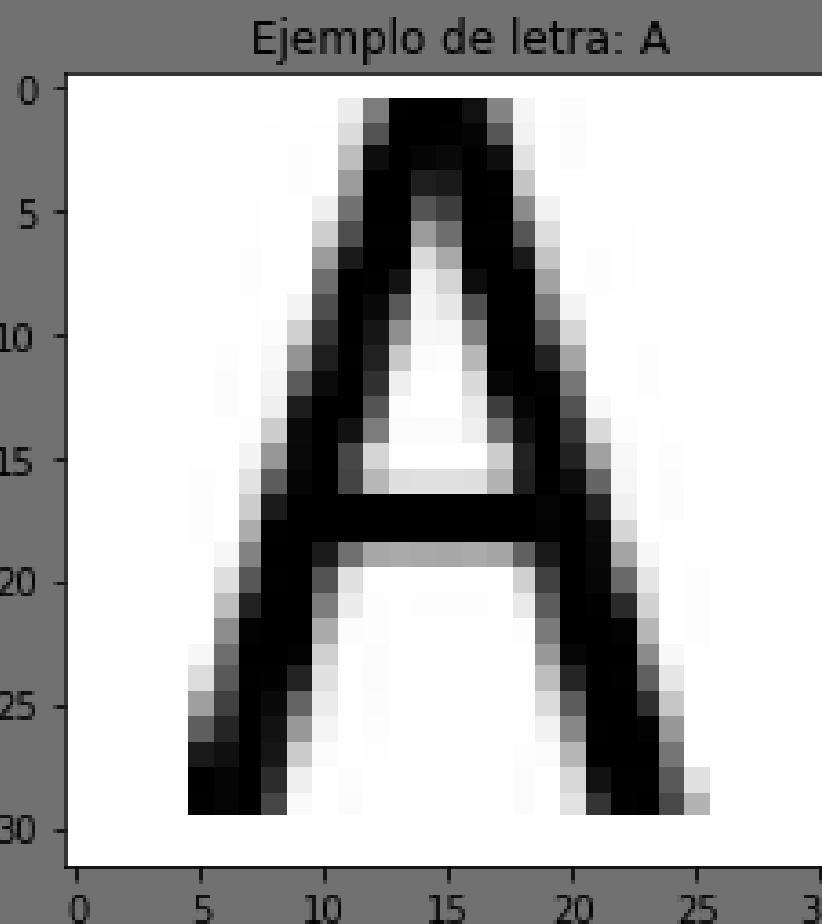
El conjunto de datos contiene 26 carpetas (A-Z) de caracteres alfabéticos en escala de grises renderizados utilizando más de 14900 fuentes, cada una con un tamaño de 32×32 píxeles.



También se proporciona un archivo binario numpy para facilitar la carga de datos. El kernel "Loading Character Dataset" contiene una demo para cargar el archivo binario numpy en matrices de píxeles de imagen y etiquetas.

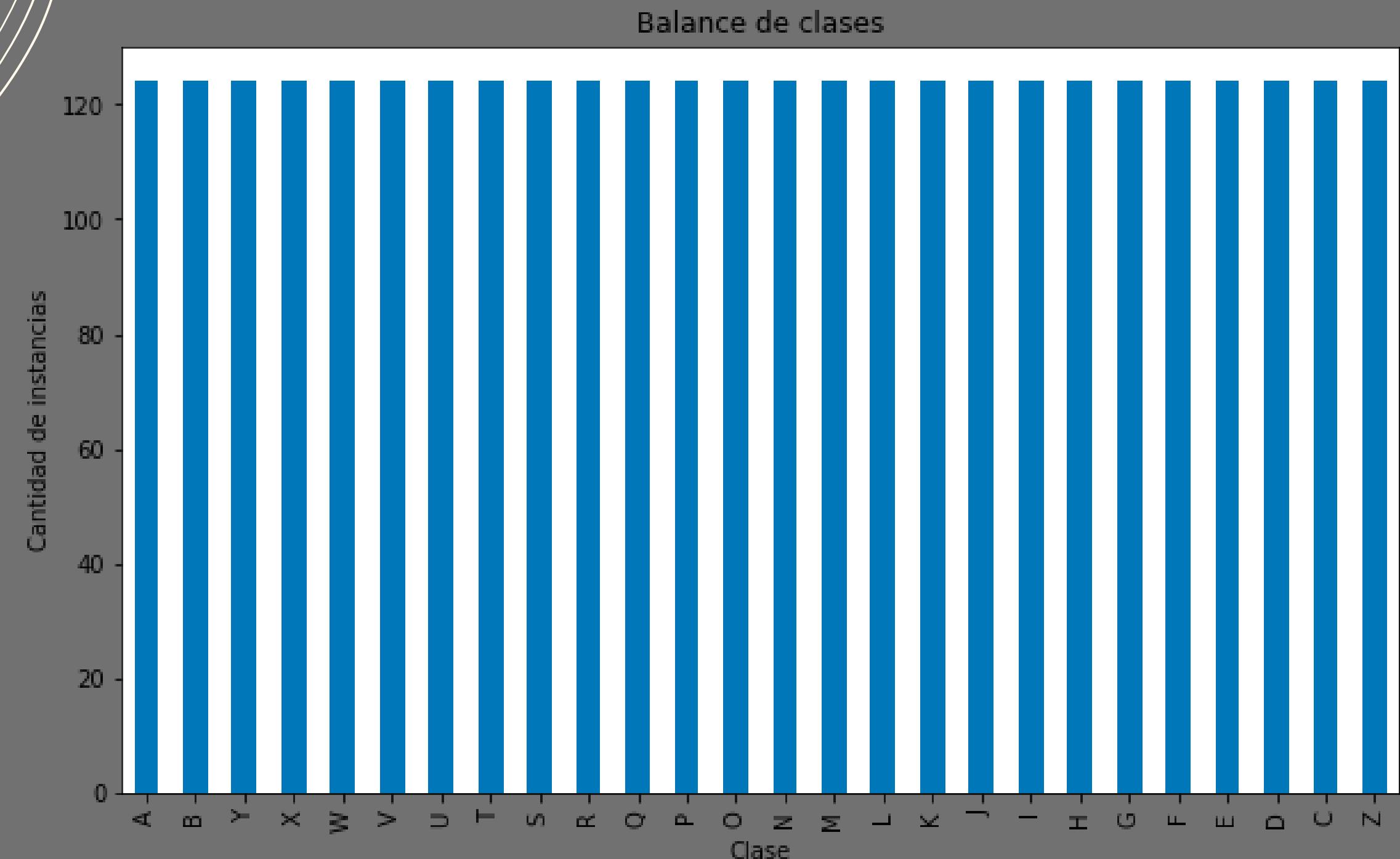


PROCESAMIENTO DE DATOS



DIMENSIONES DE X_RESHAPE: (3224, 1024)

TARGET



MISSING VALUES: Series([], dtype: int64)



MODELOS

REGRESIÓN LOGÍSTICA

Métricas Regresión Logística

Precisión: 0.85

Recall: 0.83

F1-score: 0.84

Accuracy: 0.83

DECISION TREE

Métricas Decision Tree

Precisión: 0.61

Recall: 0.60

F1-score: 0.60

Accuracy: 0.60

SVC

Métricas SVC:

Precisión: 0.86

Recall: 0.84

F1-score: 0.85

Accuracy: 0.84

KNN

Métricas KNN

Precisión: 0.78

Recall: 0.74

F1-score: 0.75

Accuracy: 0.74

ESAMBLE

Métricas de clasificación

Precisión: 0.80
Recall: 0.78
F1-score: 0.79
Accuracy: 0.78

Métricas Random Forest

Precisión: 0.83
Recall: 0.81
F1-score: 0.82
Accuracy: 0.81

Métricas XGboost

Precisión: 0.81
Recall: 0.80
F1-score: 0.80
Accuracy: 0.80

Métricas ADAboost

Precisión: 0.42
Recall: 0.34
F1-score: 0.34
Accuracy: 0.34

REDES CONVOLUCIONALES

Métricas de Redes convolucionales

Precisión: 0.88

Recall: 0.87

F1-score: 0.87

Accuracy: 0.87

DEEP LEARNING

Métricas MLP

Precisión: 0.86

Recall: 0.84

F1-score: 0.85

Accuracy: 0.84

RESULTADOS

Modelo	Precisión	Recall	F1-score	Accuracy
Regresión Logística	0.85	0.83	0.84	0.83
Decision Tree	0.61	0.60	0.60	0.60
SVC	0.86	0.84	0.85	0.84
KNN	0.78	0.74	0.75	0.74
Random Forest	0.83	0.81	0.82	0.81
Gradient Boosting	0.80	0.78	0.79	0.78
XGboost	0.81	0.80	0.80	0.80
ADABOOST	0.42	0.34	0.34	0.34
MLP	0.86	0.84	0.85	0.84
Redes convolucionales	0.88	0.87	0.87	0.87





TECNICAS

DROP OUT

Métricas de Redes Convolucionales con Dropout

Precisión: 0.89

Recall: 0.88

F1-score: 0.88

Accuracy: 0.88

AUMENTANDO LA COMPLEJIDAD DEL MODELO

Métricas de Redes Convolucionales

Precisión: 0.94

Recall: 0.93

F1-score: 0.94

Accuracy: 0.93

BATCH NORMALIZATION

Métricas Batch Normalization

Precisión: 0.95

Recall: 0.95

F1-score: 0.95

Accuracy: 0.95

DROP OUT

Métricas de Redes Convolucionales con Dropout

Precisión: 0.89

Recall: 0.88

F1-score: 0.88

Accuracy: 0.88

Drop out

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
import numpy as np
from keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder

# Asegurarse de que los datos de entrada tengan la forma adecuada
# Agregamos una dimensión adicional para representar el canal de color de las imágenes (en este caso, 1 para imágenes en escala de grises).
# -1 se utiliza para que el tamaño de la primera dimensión se ajuste automáticamente según el tamaño original del conjunto de datos
x_train = x_train.reshape(-1, 32, 32, 1)
x_test = x_test.reshape(-1, 32, 32, 1)

# Obtener el número de clases
num_classes = len(np.unique(y_train))

# Convertir las etiquetas de clase a números enteros
label_encoder = LabelEncoder()
y_train = label_encoder.fit_transform(y_train)
y_test = label_encoder.transform(y_test)

# Codificar los datos de destino en one-hot
y_train = to_categorical(y_train, num_classes)
y_test = to_categorical(y_test, num_classes)

# Crear el modelo
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(32, 32, 1)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25)) # Capa Dropout para aplicar regularización por Dropout
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5)) # Capa Dropout para aplicar regularización por Dropout
model.add(Dense(num_classes, activation='softmax'))

# Se añaden dos capas Dropout después de las capas convolucionales y densas que desactivan un porcentaje d
# de las neuronas en el entrenamiento

# Compilar el modelo
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Entrenar el modelo
model.fit(x_train, y_train, batch_size=32, epochs=10, validation_data=(x_test, y_test))

# Predecir con el modelo
y_pred = model.predict(x_test)
y_pred = np.argmax(y_pred, axis=1)

# Calcular las métricas de clasificación
precision = precision_score(np.argmax(y_test, axis=1), y_pred, average='weighted')
recall = recall_score(np.argmax(y_test, axis=1), y_pred, average='weighted')
f1 = f1_score(np.argmax(y_test, axis=1), y_pred, average='weighted')
accuracy = accuracy_score(np.argmax(y_test, axis=1), y_pred)

# Imprimir las métricas de clasificación
print("Métricas de Redes convolucionales con Dropout:")
print("Precisión: {:.2f}".format(precision))
print("Recall: {:.2f}".format(recall))
print("F1-score: {:.2f}".format(f1))
print("Accuracy: {:.2f}".format(accuracy))
```

AUMENTANDO LA COMPLEJIDAD DEL MODELO

Métricas de Redes Convolucionales

Precisión: 0.94

Recall: 0.93

F1-score: 0.94

Accuracy: 0.93

Aumentando la complejidad del modelo

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
import numpy as np
from keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder

# Asegurarse de que los datos de entrada tengan la forma adecuada
# Agregamos una dimensión adicional para representar el canal de color de las imágenes (en este caso, 1 para imágenes en escala de grises).
# -1 se utiliza para que el tamaño de la primera dimensión se ajuste automáticamente según el tamaño original del conjunto de datos
x_train = x_train.reshape(-1, 32, 32, 1)
x_test = x_test.reshape(-1, 32, 32, 1)

# Obtener el número de clases
num_classes = len(np.unique(y_train))

# Convertir las etiquetas de clase a números enteros
label_encoder = LabelEncoder()
y_train = label_encoder.fit_transform(y_train)
y_test = label_encoder.transform(y_test)

# Codificar los datos de destino en one-hot
y_train = to_categorical(y_train, num_classes)
y_test = to_categorical(y_test, num_classes)

# Crear el modelo
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(32, 32, 1)))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25)) # Capa Dropout para aplicar regularización por Dropout
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5)) # Capa Dropout para aplicar regularización por Dropout
model.add(Dense(num_classes, activation='softmax'))

# Compilar el modelo
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

model.fit(x_train, y_train, batch_size=32, epochs=20, validation_data=(x_test, y_test))

y_pred = model.predict(x_test)
y_pred = np.argmax(y_pred, axis=1)

precision = precision_score(np.argmax(y_test, axis=1), y_pred, average='weighted')
recall = recall_score(np.argmax(y_test, axis=1), y_pred, average='weighted')
f1 = f1_score(np.argmax(y_test, axis=1), y_pred, average='weighted')
accuracy = accuracy_score(np.argmax(y_test, axis=1), y_pred)

# Imprimir las métricas de clasificación
print("Métricas de Redes convolucionales:")
print("Precisión: {:.2f}".format(precision))
print("Recall: {:.2f}".format(recall))
print("F1-score: {:.2f}".format(f1))
print("Accuracy: {:.2f}".format(accuracy))

# Se intenta aumentar la complejidad del modelo y en este caso se agregan dos capas convolucionales (64 filtros y 128 filtros)
# Para que aumente la capacidad del modelo de detectar mejor aquellas imágenes más difíciles.
```

BATCH NORMALIZATION

Métricas Batch Normalization

Precisión: 0.95
Recall: 0.95
F1-score: 0.95
Accuracy: 0.95

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, BatchNormalization
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
import numpy as np
from keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder

# Asegurarse de que los datos de entrada tengan la forma adecuada
# Agregamos una dimensión adicional para representar el canal de color de las imágenes (en este caso, 1 para imágenes en escala de grises).
# -1 se utiliza para que el tamaño de la primera dimensión se ajuste automáticamente según el tamaño original del conjunto de datos
x_train = x_train.reshape(-1, 32, 32, 1)
x_test = x_test.reshape(-1, 32, 32, 1)

# Obtener el número de clases
num_classes = len(np.unique(y_train))

# Convertir las etiquetas de clase a números enteros
label_encoder = LabelEncoder()
y_train = label_encoder.fit_transform(y_train)
y_test = label_encoder.transform(y_test)

# Codificar los datos de destino en one-hot
y_train = to_categorical(y_train, num_classes)
y_test = to_categorical(y_test, num_classes)

# Crear el modelo
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(32, 32, 1)))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5)) # Capa Dropout para aplicar regularización por Dropout
model.add(Dense(num_classes, activation='softmax'))

# Aplicamos Batch Normalization después de aplicar una covolución

# Compilar el modelo
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Entrenar el modelo
model.fit(x_train, y_train, batch_size=32, epochs=15, validation_data=(x_test, y_test))

# Predecir con el modelo
y_pred = model.predict(x_test)
y_pred = np.argmax(y_pred, axis=1)

# Calcular las métricas de clasificación
precision = precision_score(np.argmax(y_test, axis=1), y_pred, average='weighted')
recall = recall_score(np.argmax(y_test, axis=1), y_pred, average='weighted')
f1 = f1_score(np.argmax(y_test, axis=1), y_pred, average='weighted')
accuracy = accuracy_score(np.argmax(y_test, axis=1), y_pred)

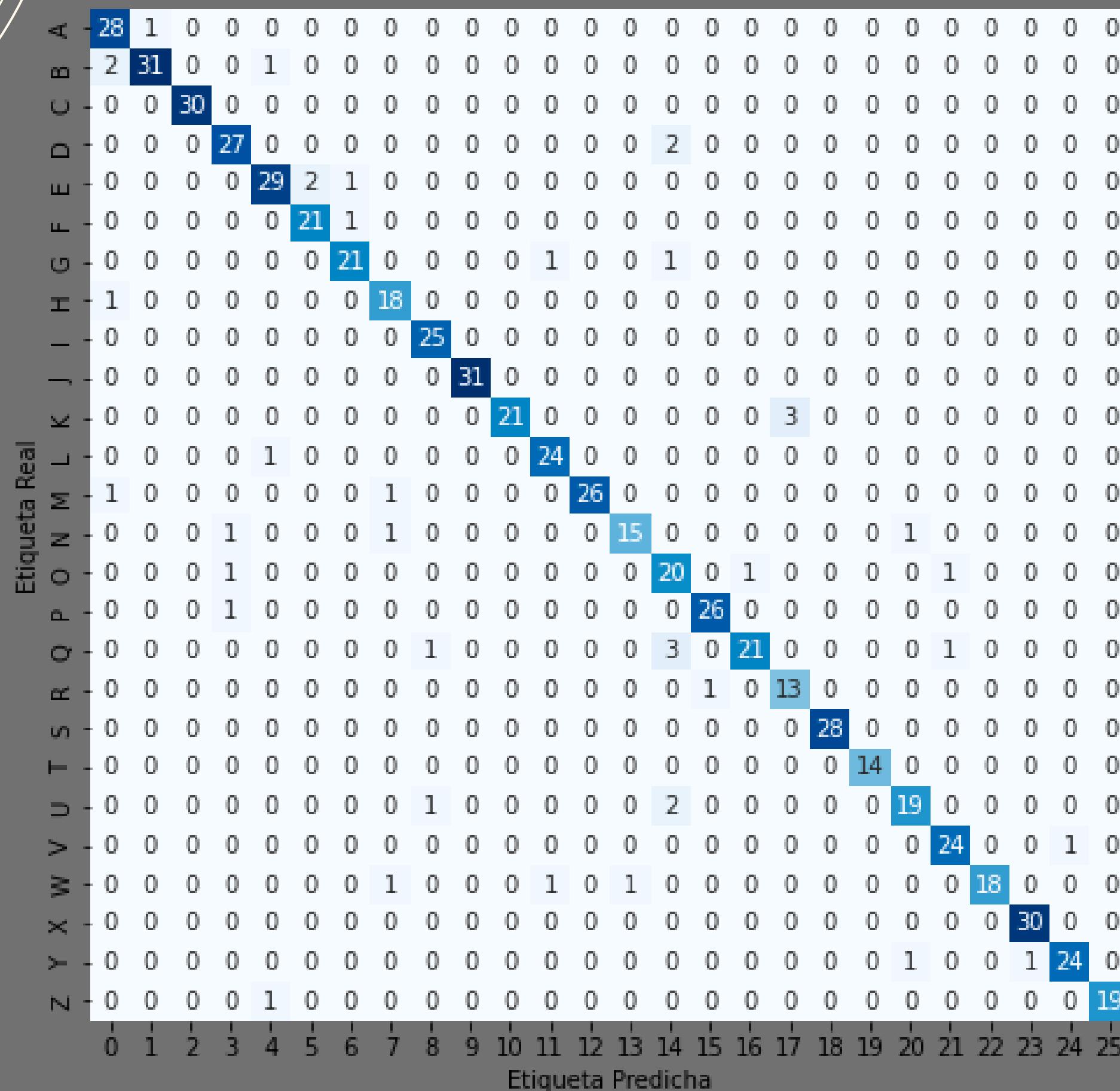
# Imprimir las métricas de clasificación
print("Métricas de Redes convolucionales:")
print("Precisión: {:.2f}".format(precision))
print("Recall: {:.2f}".format(recall))
print("F1-score: {:.2f}".format(f1))
print("Accuracy: {:.2f}".format(accuracy))

# Añadimos dos capas de Batch Normalization, después de las capas convolucionales, aumentando así ligeramente la complejidad.
```

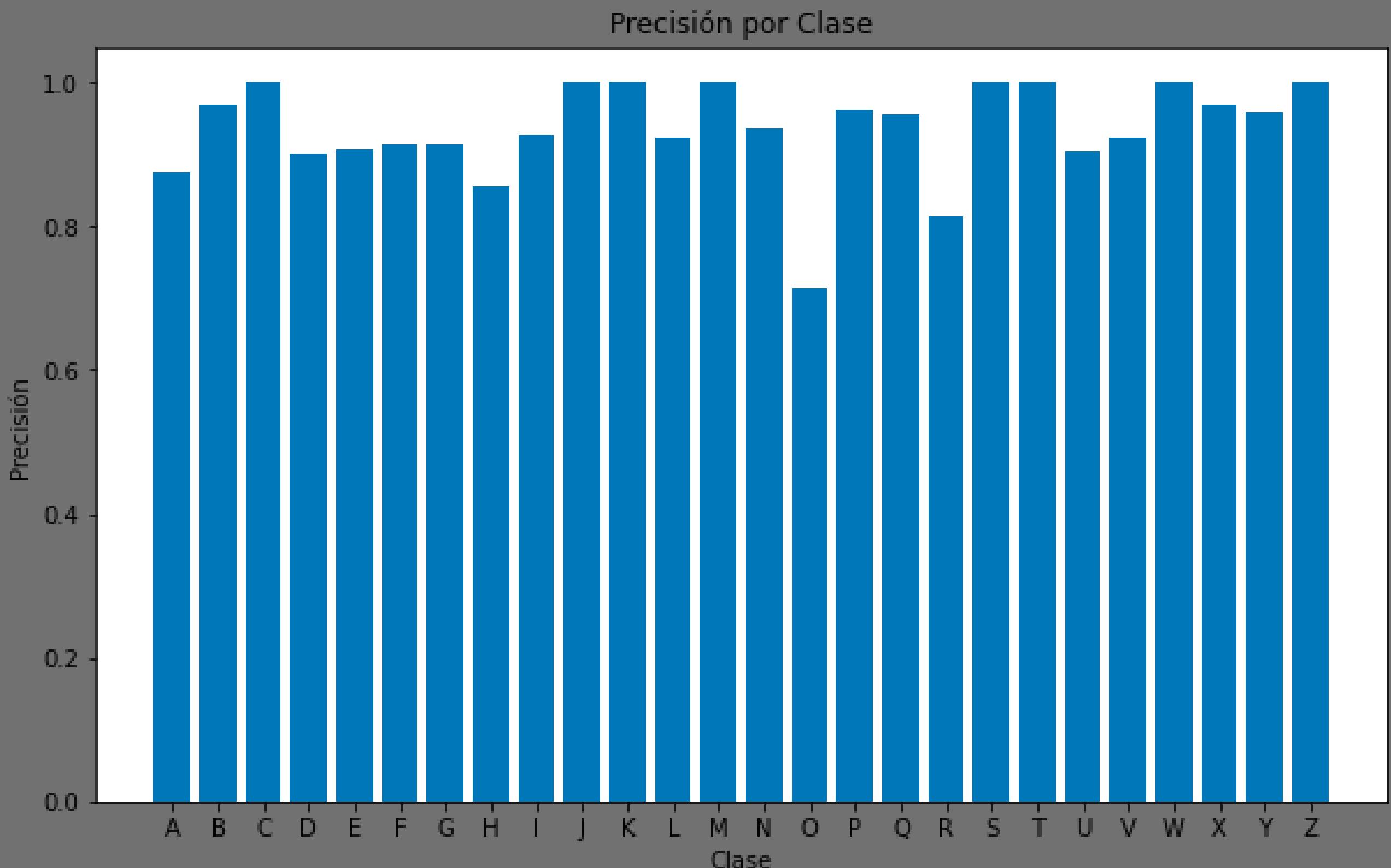
```
# Crear el modelo
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(32, 32, 1)))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5)) # Capa Dropout para aplicar regularización por Dropout
model.add(Dense(num_classes, activation='softmax'))
```

MATRIZ DE CONFUSIÓN

Matriz de Confusión



PRESICIÓN POR CLASE



PREPROCESADO DE IMÁGENES



IMAGEN ORIGINAL

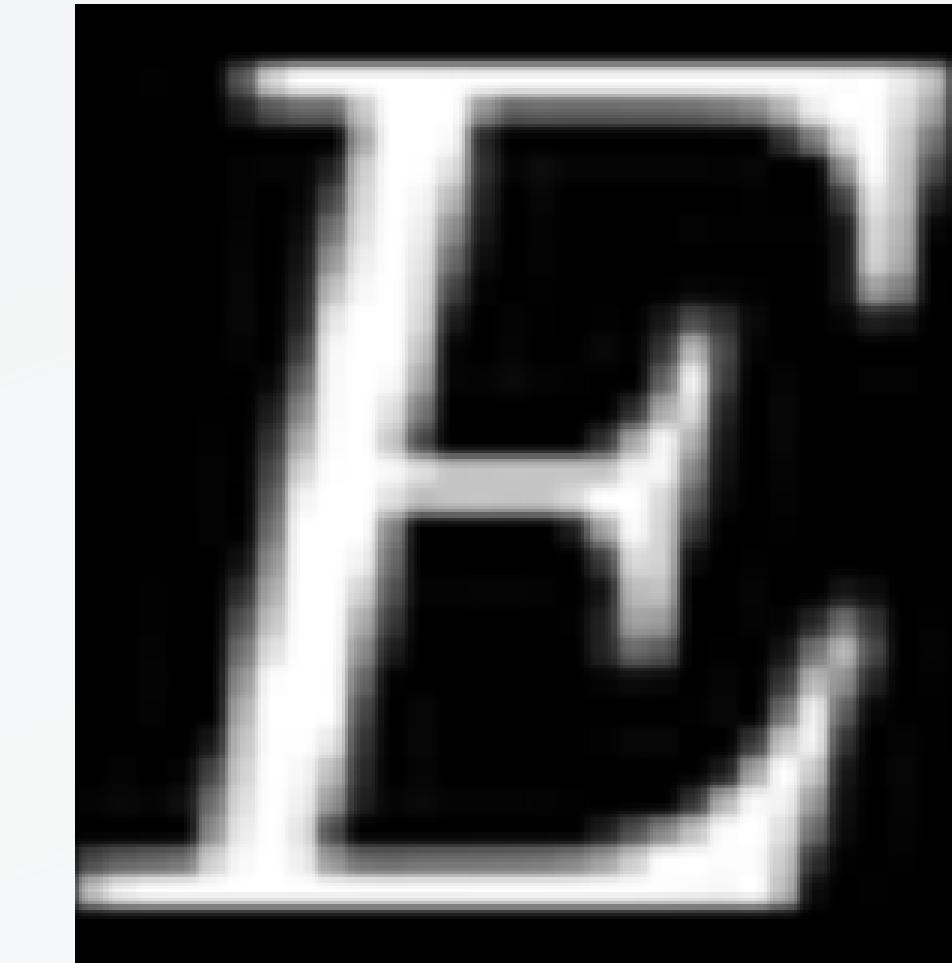
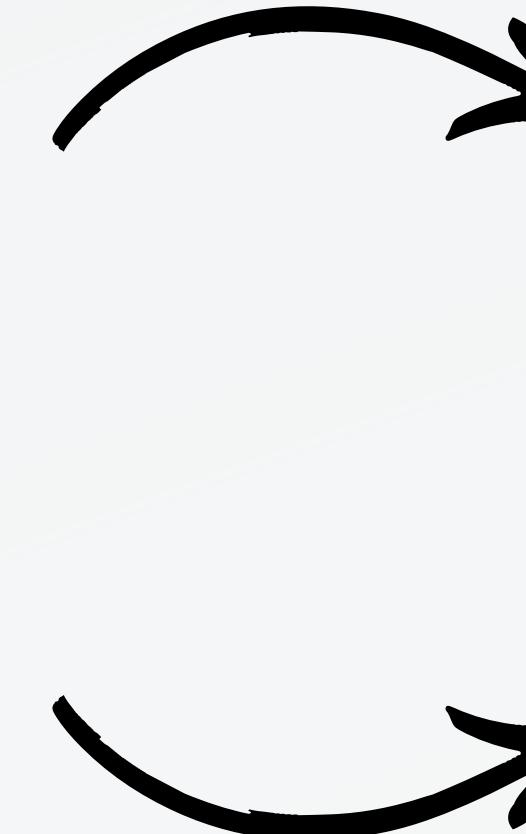


IMAGEN ENTRENAMIENTO

PRERREQUISITOS PARA EL CORRECTO PROCESADO DE IMÁGENES

- En la imagen solo puede aparecer una letra.
- El fondo debe de ser de un color homogéneo.
- Debe existir un claro contraste entre el fondo y la letra.
- Puede aparecer algo de ruido, pero nunca más grande que la letra.

01



02



03



04



05



06

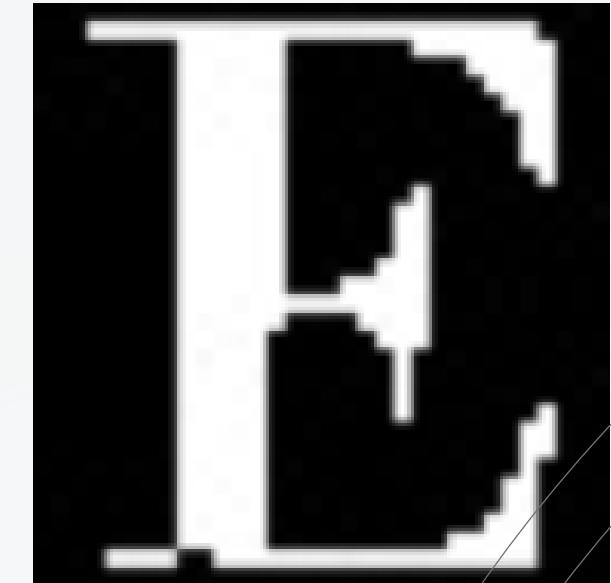


IMAGEN ORIGINAL

DETECCIÓN OBJETO

- Detección contorno más grande.
- Creación de rectángulo que envuelva el contorno.

ENCUADRE Y SIMETRÍA

- Centrar imagen según posición del rectángulo.
- Recortar lado más largo.

RECORTE BORDES

- Cálculo de escala de conversión.
- Recortar bordes sobrantes.

REDIMENSIONAR

- Resize a la nueva dimensión.

CAMBIO COLOR

- Convertir a blanco y negro.

E



Predicción Imagen №2: - Imagen: imagen E.jpg

A: 0.00%
B: 0.00%
C: 0.00%
D: 0.00%
E: 100.00%
F: 0.00%
G: 0.00%
H: 0.00%
I: 0.00%
J: 0.00%
K: 0.00%
L: 0.00%
M: 0.00%
N: 0.00%
O: 0.00%
P: 0.00%
Q: 0.00%
R: 0.00%
S: 0.00%
T: 0.00%
U: 0.00%
V: 0.00%
W: 0.00%
X: 0.00%
Y: 0.00%
Z: 0.00%

D



Predicción Imagen №1: - Imagen: imagen D.jpg

A: 0.00%
B: 0.00%
C: 0.00%
D: 100.00%
E: 0.00%
F: 0.00%
G: 0.00%
H: 0.00%
I: 0.00%
J: 0.00%
K: 0.00%
L: 0.00%
M: 0.00%
N: 0.00%
O: 0.00%
P: 0.00%
Q: 0.00%
R: 0.00%
S: 0.00%
T: 0.00%
U: 0.00%
V: 0.00%
W: 0.00%
X: 0.00%
Y: 0.00%
Z: 0.00%

M



Predicción Imagen №3: - Imagen: imagen M.jpg

A: 0.00%
B: 0.00%
C: 0.00%
D: 0.00%
E: 0.00%
F: 0.00%
G: 0.00%
H: 0.00%
I: 0.00%
J: 0.00%
K: 0.00%
L: 0.00%
M: 100.00%
N: 0.00%
O: 0.00%
P: 0.00%
Q: 0.00%
R: 0.00%
S: 0.00%
T: 0.00%
U: 0.00%
V: 0.00%
W: 0.00%
X: 0.00%
Y: 0.00%
Z: 0.00%

**THANK'S FOR
LISTENING**

