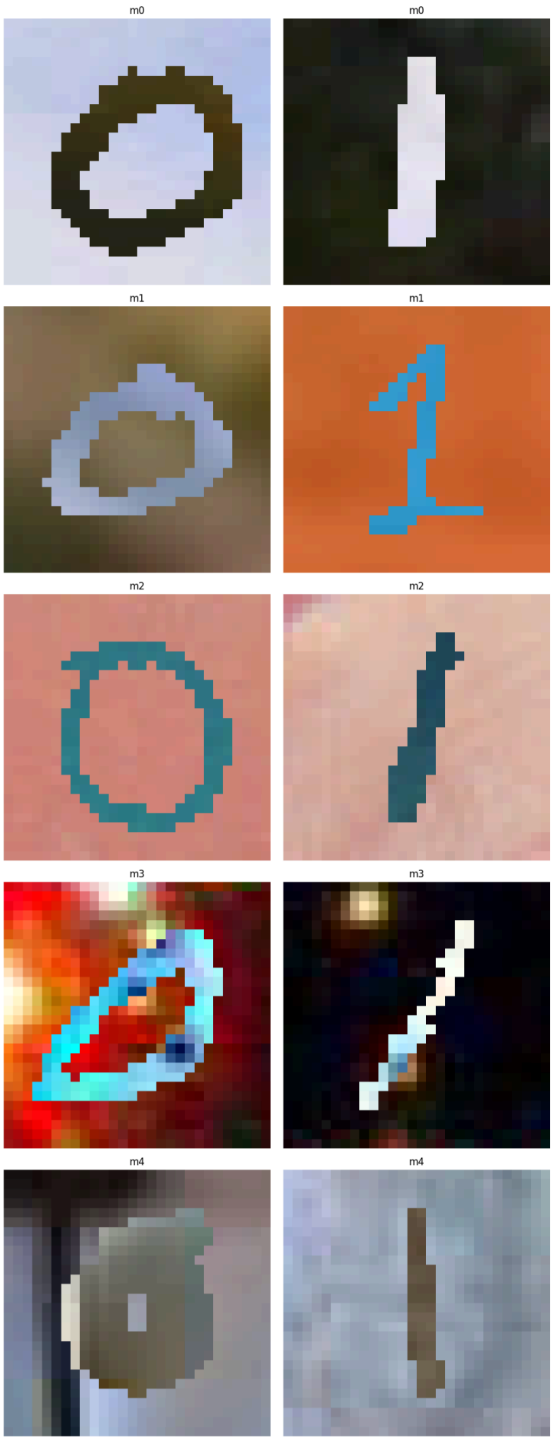
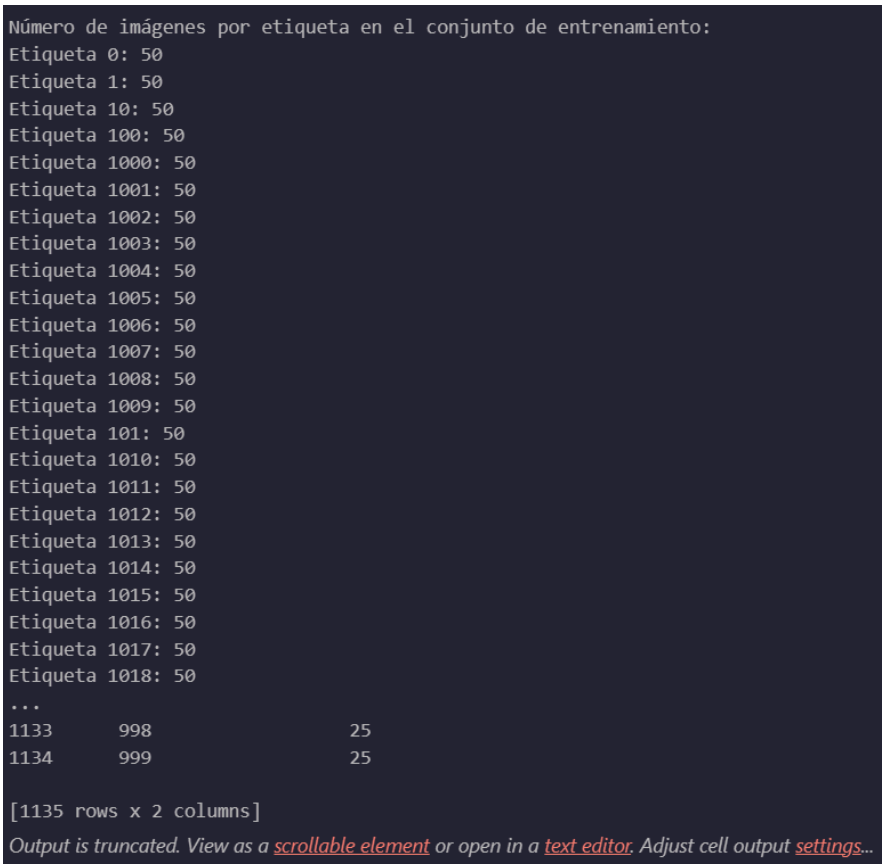
David Aragón 21053
Renatto Guzmán 21646
Data Science - 10

Laboratorio 4

1. Muestre algunos ejemplos de cada una de las modalidades que tiene.



2. Haga un análisis exploratorio de los datos para entenderlos mejor, documente todos los análisis. Especifique la resolución de las imágenes, la distribución del conjunto de datos, si está balanceado o no, etc.



```
Número de imágenes por etiqueta en el conjunto de entrenamiento:
Etiqueta 0: 50
Etiqueta 1: 50
Etiqueta 10: 50
Etiqueta 100: 50
Etiqueta 1000: 50
Etiqueta 1001: 50
Etiqueta 1002: 50
Etiqueta 1003: 50
Etiqueta 1004: 50
Etiqueta 1005: 50
Etiqueta 1006: 50
Etiqueta 1007: 50
Etiqueta 1008: 50
Etiqueta 1009: 50
Etiqueta 101: 50
Etiqueta 1010: 50
Etiqueta 1011: 50
Etiqueta 1012: 50
Etiqueta 1013: 50
Etiqueta 1014: 50
Etiqueta 1015: 50
Etiqueta 1016: 50
Etiqueta 1017: 50
Etiqueta 1018: 50
...
1133      998                 25
1134      999                 25

[1135 rows x 2 columns]
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

Se revisó que el conjunto de imágenes de entrenamiento está balanceado

```
Total de imágenes: 300000
Número de clases: 6742
Etiqueta 0: 50 imágenes (0.02%)
Etiqueta 1: 50 imágenes (0.02%)
Etiqueta 10: 50 imágenes (0.02%)
Etiqueta 100: 50 imágenes (0.02%)
Etiqueta 1000: 50 imágenes (0.02%)
Etiqueta 1001: 50 imágenes (0.02%)
Etiqueta 1002: 50 imágenes (0.02%)
Etiqueta 1003: 50 imágenes (0.02%)
Etiqueta 1004: 50 imágenes (0.02%)
Etiqueta 1005: 50 imágenes (0.02%)
Etiqueta 1006: 50 imágenes (0.02%)
Etiqueta 1007: 50 imágenes (0.02%)
Etiqueta 1008: 50 imágenes (0.02%)
Etiqueta 1009: 50 imágenes (0.02%)
Etiqueta 101: 50 imágenes (0.02%)
Etiqueta 1010: 50 imágenes (0.02%)
Etiqueta 1011: 50 imágenes (0.02%)
Etiqueta 1012: 50 imágenes (0.02%)
Etiqueta 1013: 50 imágenes (0.02%)
Etiqueta 1014: 50 imágenes (0.02%)
Etiqueta 1015: 50 imágenes (0.02%)
Etiqueta 1016: 50 imágenes (0.02%)
Etiqueta 1017: 50 imágenes (0.02%)
...

El conjunto de entrenamiento está balanceado.

El conjunto de prueba está balanceado.
```

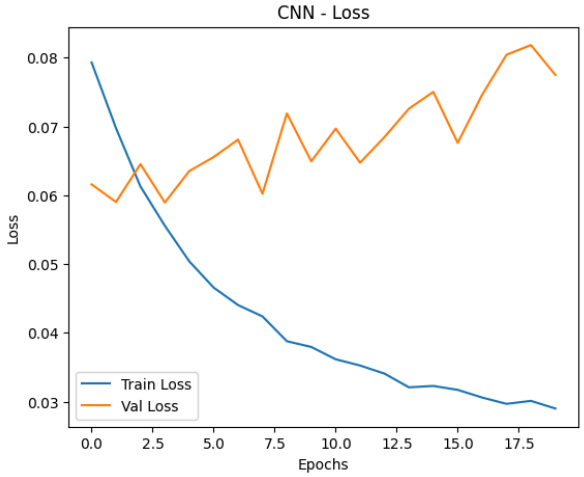
3. Haga al menos 2 modelos de Deep learning, determine la efectividad de cada uno y seleccione el mejor de ellos.

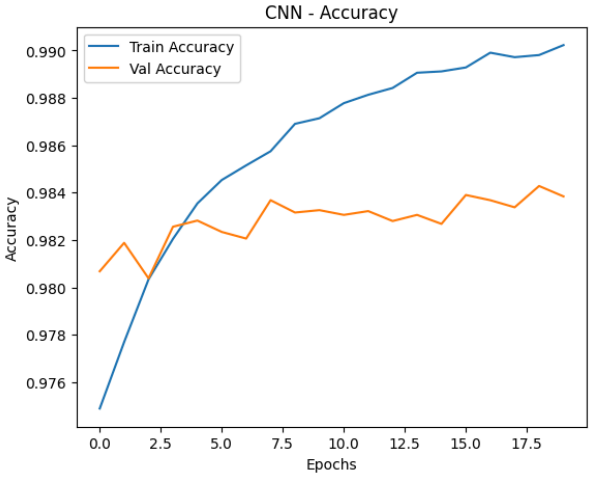
```python
def preprocess_data(images, labels):
    images = images.astype('float32') / 255.0
    labels = to_categorical(labels, num_classes=10)
    images = images.reshape(images.shape[0], 28, 28, 3)
    return images, labels


# Process the training data
X_train, y_train = preprocess_data(train_images, train_labels)

# Process the test data
X_test, y_test = preprocess_data(test_images, test_labels)
```

Decidimos crear modelos de CNN y de MLP

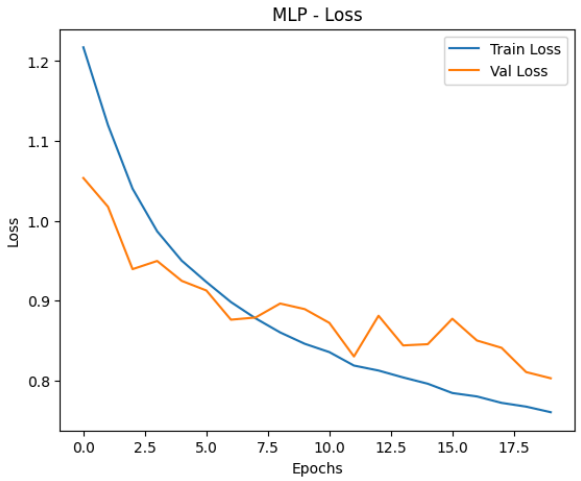
```python
def create_cnn_model():
    model = Sequential()
    model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(28, 28, 1)))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(10, activation='softmax'))

    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    return model

cnn_model = create_cnn_model()
cnn_model.summary()

def create_mlp_model():
    model = Sequential()
    model.add(Flatten(input_shape=(28, 28, 1)))
    model.add(Dense(512, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(512, activation='relu'))
    model.add(Dense(10, activation='softmax'))

    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    return model

mlp_model = create_mlp_model()
mlp_model.summary()
```

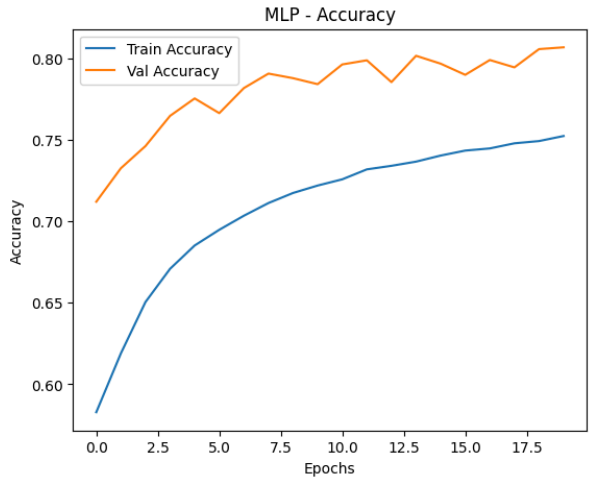
Modelo CNN





Modelo MLP

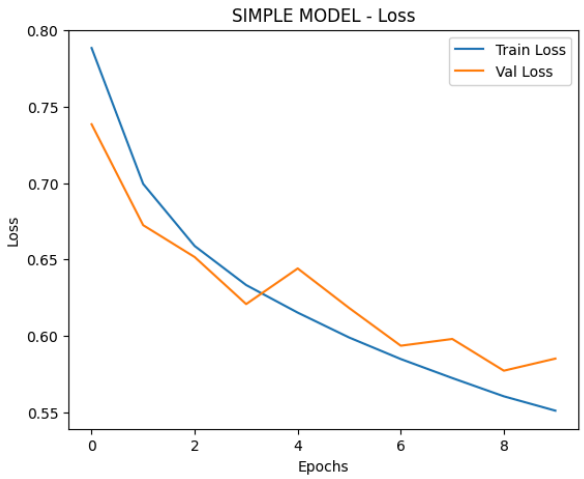


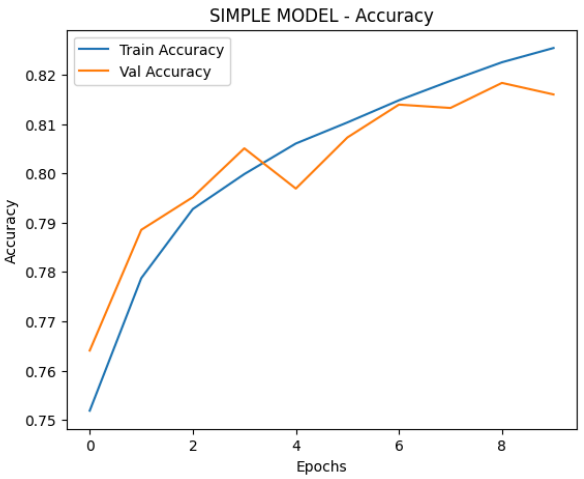


El modelo **CNN** se desempeña mejor que el **MLP**.

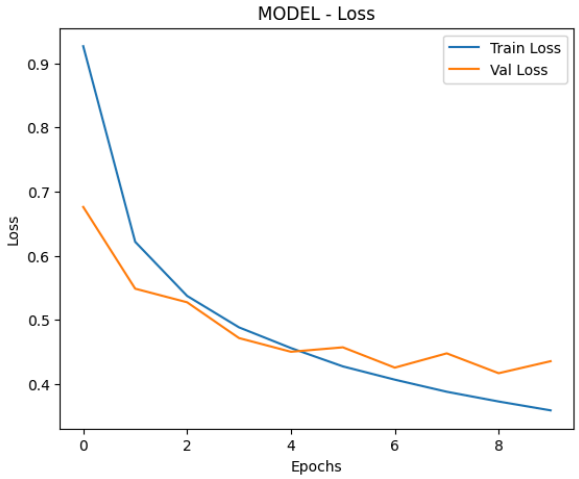
Presenta una pérdida de entrenamiento y validación más baja y estable a lo largo de las épocas, mientras que el MLP muestra una mayor pérdida en general. En cuanto a la precisión, el CNN alcanza un valor más alto, aunque hay una ligera brecha entre la precisión de entrenamiento y validación, lo que sugiere un leve sobreajuste. Por otro lado, el MLP tiene una menor precisión y muestra una mayor diferencia entre la precisión de entrenamiento y validación, indicando un sobreajuste más pronunciado. En conjunto, el CNN ofrece un mejor rendimiento.

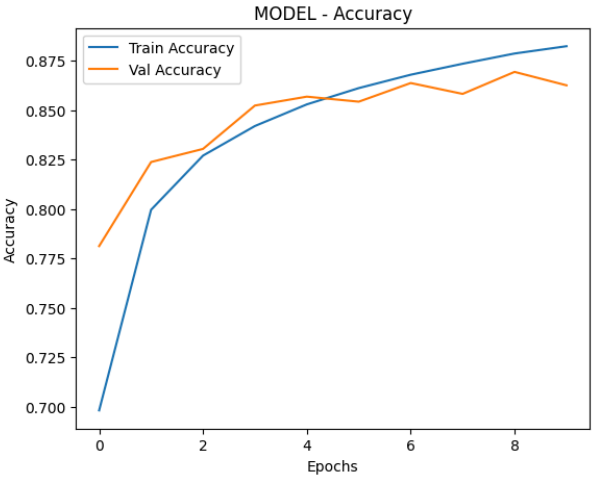
Parte 4



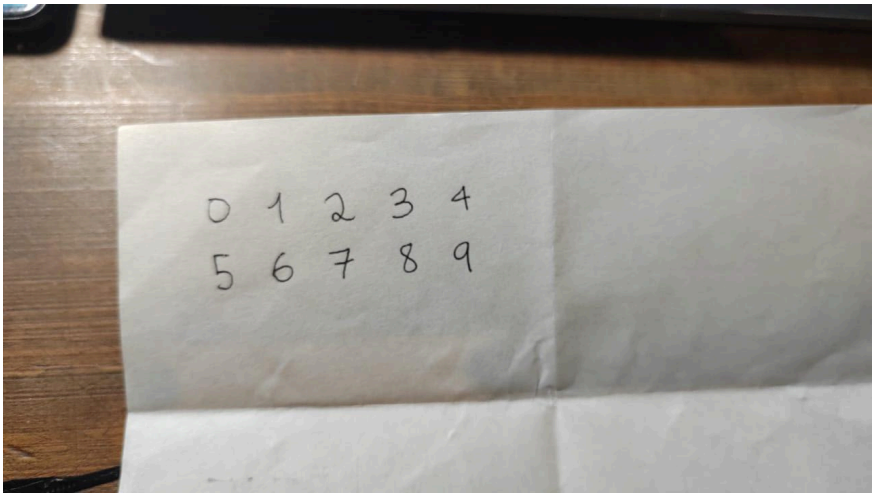


Parte 5.



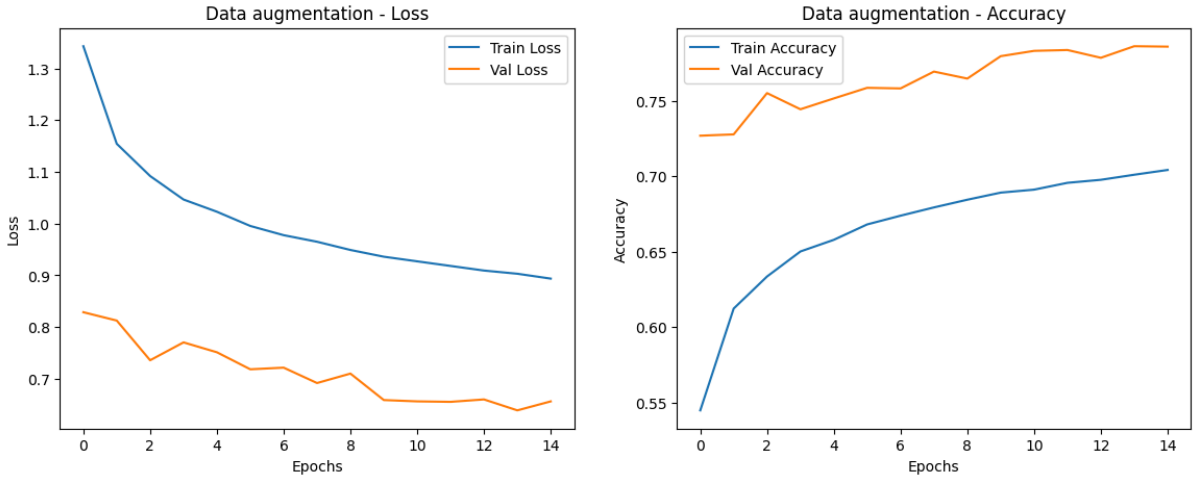


Nuestra imagen original



6.Hágale transformaciones a los datos como image augmentation y vuelva a entrenar sus modelos, discuta los resultados

```python
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Configuración de ImageDataGenerator con aumentación de imágenes
datagen = ImageDataGenerator(
    rotation_range=10,      # Rotar imágenes hasta 10 grados
    width_shift_range=0.1,  # Desplazamiento horizontal hasta un 10%
    height_shift_range=0.1, # Desplazamiento vertical hasta un 10%
    shear_range=0.1,        # Aplicar transformación de corte
    zoom_range=0.1,         # Zoom dentro del rango del 10%
    horizontal_flip=True,   # Espejado horizontal
    fill_mode='nearest'     # Modo de relleno para píxeles vacíos
)

# Ajustar el generador a los datos de entrenamiento
datagen.fit(X_train)
```



En las gráficas de pérdida (*loss*), se observa que la pérdida de validación disminuye significativamente y se estabiliza después de varias épocas. Este es un indicador de que las técnicas de *image augmentation* han ayudado a reducir el *overfitting*. Anteriormente, es probable que el modelo se haya ajustado demasiado a los datos de entrenamiento, lo que resultaba en una mayor pérdida de validación

La gráfica de exactitud (*accuracy*) muestra una mejora constante en la exactitud de validación a medida que avanzan las épocas, superando la exactitud del entrenamiento en algunas etapas. Esto sugiere que el modelo está generalizando mejor a datos no vistos, lo cual es uno de los objetivos clave de aplicar *image augmentation*. La técnica parece haber creado un modelo más robusto y menos dependiente de patrones específicos en el conjunto de entrenamiento.

En resumen, las técnicas de *image augmentation* han mejorado la capacidad de generalización del modelo, reduciendo el *overfitting* y mejorando la exactitud en los datos de

validación. Estas técnicas son útiles para aumentar la variabilidad del conjunto de datos de entrenamiento sin necesidad de recopilar más datos reales, y son particularmente efectivas cuando se dispone de un conjunto de datos limitado.

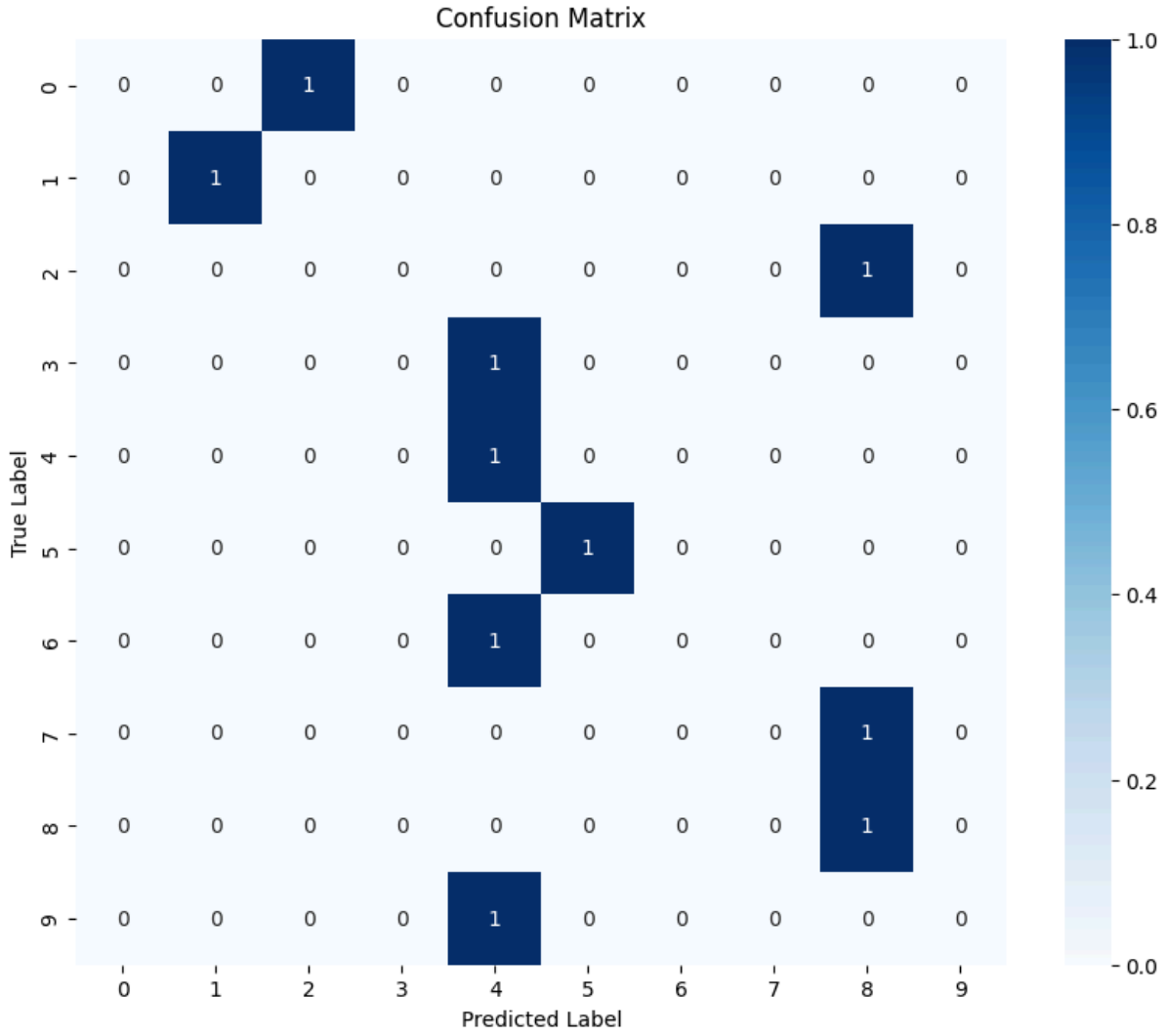
7. Pruebe el mejor modelo ingresando imágenes de dígitos hechos a mano por los integrantes del grupo. Discuta el desempeño de su modelo y los resultados.

```python
import os
import numpy as np
from PIL import Image

def load_test_images(folder, target_size=(28, 28)):
    my_Images = []
    my_Labels = []
    for filename in os.listdir(folder):
        if filename.endswith(".png"):
            try:
                # Cargar la imagen
                img_path = os.path.join(folder, filename)
                img = Image.open(img_path)

                # Asegurarse de que la imagen esté en modo RGB
                img = img.convert('RGB')

                # Redimensionar la imagen
                img = img.resize(target_size, Image.LANCZOS)

                # Convertir a array y normalizar
                img_array = np.array(img) / 255.0

                # Extraer la etiqueta del nombre del archivo
                label = int(filename.split('.')[0])  # Por ejemplo: 8.png -> 8

                my_Images.append(img_array)
                my_Labels.append(label)
            except Exception as e:
                print(f"Error al cargar {filename}: {e}")

    # Convertir a arrays NumPy
    my_Images = np.array(my_Images)
    my_Labels = np.array(my_Labels)
```

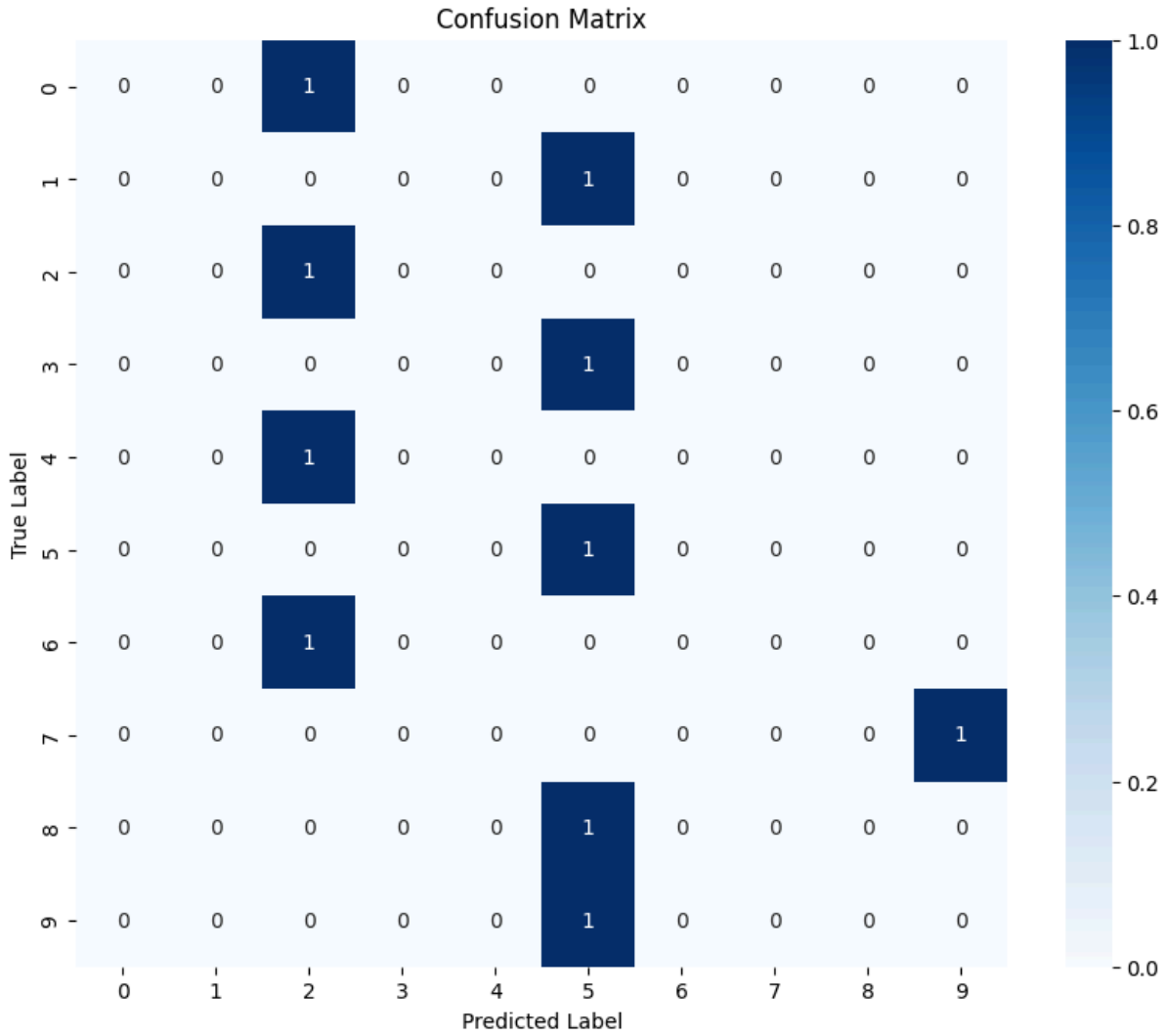
```
    return my_Images, my_Labels

# Ruta a la carpeta de imágenes de prueba
test_dir = "imgs"

# Cargar las imágenes de prueba
my_Images, my_Labels = load_test_images(test_dir)

# Imprimir información sobre los datos cargados
print("Forma de my_Images:", my_Images.shape)
print("Forma de my_Labels:", my_Labels.shape)
print("Etiquetas únicas:", np.unique(my_Labels))
print("Rango de valores en my_Images:", my_Images.min(), "-", my_Images.max())
```

```
def predict_and_evaluate(model, X_test, y_test):
    # Make predictions
    y_pred = model.predict(X_test)

    # If the model outputs probabilities, convert to class predictions
    if y_pred.ndim > 1 and y_pred.shape[1] > 1:
        y_pred = np.argmax(y_pred, axis=1)

    # Calculate accuracy
    accuracy = accuracy_score(y_test, y_pred)

    # Generate classification report
    class_report = classification_report(y_test, y_pred)

    # Create confusion matrix
    cm = confusion_matrix(y_test, y_pred)

    # Print results
    print(f"Accuracy: {accuracy:.4f}")
    print("\nClassification Report:")
    print(class_report)

    # Plot confusion matrix
    plt.figure(figsize=(10, 8))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title('Confusion Matrix')
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    plt.show()

    return y_pred, accuracy
```

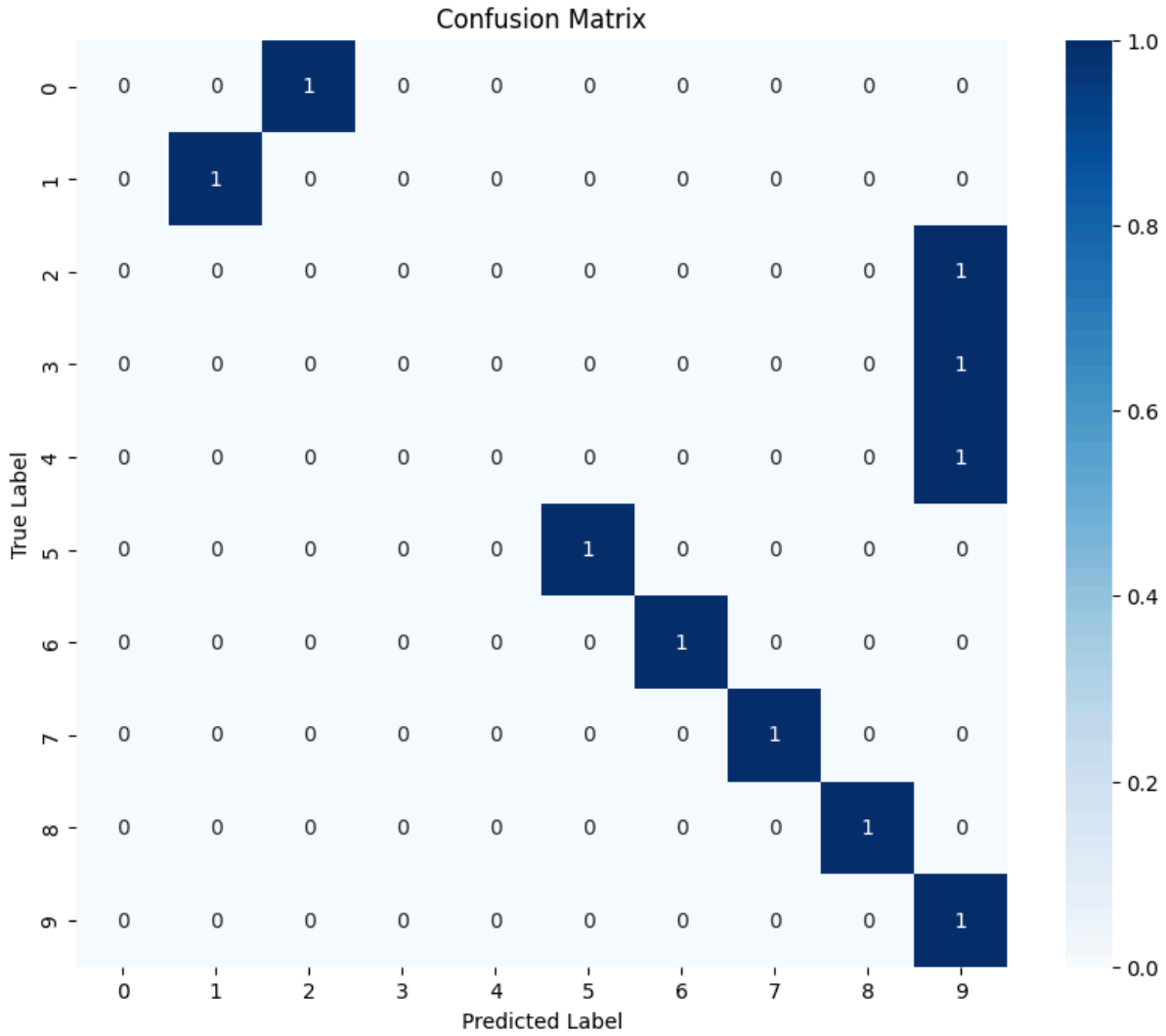
model_new1



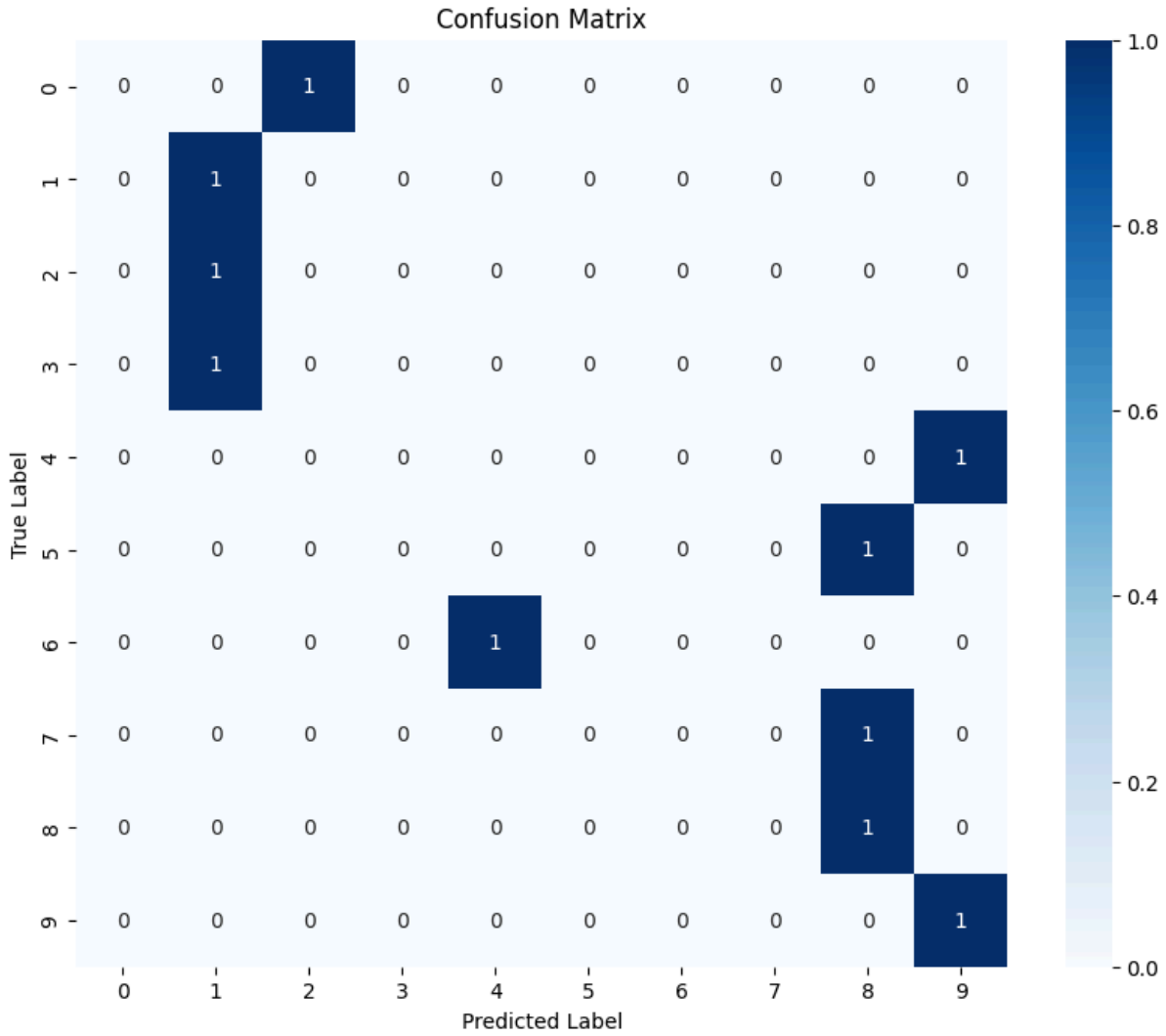
model_new


Confusion Matrix
True Label
Predicted Label
0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0
0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0
0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0
0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0 0 0
0 1 2 3 4 5 6 7 8 9
1.0
0.8
0.6
0.4
0.2
0.0

cnn_model


Confusion Matrix
True Label
Predicted Label
0 0 1 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 1
0 1 2 3 4 5 6 7 8 9
1.0 0.8 0.6 0.4 0.2 0.0

mlp_model



8. Haga un informe donde incluya el análisis exploratorio, la descripción de los modelos, la efectividad de cada uno y la comparación entre ellos.

1. Análisis Exploratorio

El análisis exploratorio se realizó para entender mejor la distribución de las etiquetas en los conjuntos de datos de entrenamiento y prueba. Esto es crucial para identificar posibles problemas de desbalanceo que podrían afectar el rendimiento de los modelos.

1.1. Distribución de las Etiquetas

Se contó el número de imágenes por etiqueta tanto en el conjunto de entrenamiento como en el de prueba. A continuación, se presentan los resultados:

- Conjunto de Entrenamiento:
    - La cantidad de imágenes por etiqueta varía, lo que podría indicar un desbalanceo en algunas clases.
- Conjunto de Prueba:

    - Se observó una distribución similar en el conjunto de prueba, lo cual es importante para evaluar la capacidad de generalización del modelo.

2. Descripción de los Modelos

Se desarrollaron varios modelos de redes neuronales para la tarea de clasificación de imágenes. A continuación, se describen los detalles de cada modelo y su arquitectura.

2.1. Modelo CNN (Convolutional Neural Network)

El modelo CNN ha demostrado ser el más efectivo en términos de precisión y generalización. La arquitectura del modelo es la siguiente:

- **Capas**:
    - `Conv2D(32, kernel_size=(3, 3), activation='relu')`: Primera capa convolucional con 32 filtros y tamaño de kernel 3x3.
    - `MaxPooling2D(pool_size=(2, 2))`: Capa de pooling para reducir las dimensiones espaciales.
    - `Conv2D(64, kernel_size=(3, 3), activation='relu')`: Segunda capa convolucional con 64 filtros.
    - `MaxPooling2D(pool_size=(2, 2))`: Otra capa de pooling.
    - `Flatten()`: Aplana los datos para la capa totalmente conectada.
    - `Dense(128, activation='relu')`: Capa totalmente conectada con 128 unidades y activación ReLU.
    - `Dropout(0.5)`: Regularización para evitar el sobreajuste.
    - `Dense(10, activation='softmax')`: Capa de salida con 10 unidades y activación softmax.
- **Compilación**:
    - Optimizador: `adam`
    - Pérdida: `categorical_crossentropy`
    - Métricas: `accuracy`

2.2. Modelo MLP (Multilayer Perceptron)

El MLP es una red densa completamente conectada sin capas convolucionales:

- **Capas**:
    - `Flatten(input_shape=(28, 28, 3))`: Aplana la entrada.
    - `Dense(512, activation='relu')`: Primera capa oculta con 512 unidades.
    - `Dropout(0.5)`: Regularización.
    - `Dense(512, activation='relu')`: Segunda capa oculta.
    - `Dense(10, activation='softmax')`: Capa de salida.
- **Compilación**:
    - Optimizador: `adam`
    - Pérdida: `categorical_crossentropy`

    - Métricas: `accuracy`

2.3. Modelo Simple

Este es un modelo sencillo con una única capa oculta:

- **Capas**:
    - `Flatten(input_shape=(28, 28, 3))`: Aplana la entrada.
    - `Dense(128, activation='relu')`: Capa oculta con 128 unidades.
    - `Dense(10, activation='softmax')`: Capa de salida.
- **Compilación**:
    - Optimizador: `adam`
    - Pérdida: `categorical_crossentropy`
    - Métricas: `accuracy`

2.4. Modelo `model_new1`

Este modelo tiene una arquitectura similar al MLP, pero con un mayor número de unidades en las capas ocultas:

- **Capas**:
    - `Flatten(input_shape=(28, 28, 3))`: Aplana la entrada.
    - `Dense(300, activation='relu')`: Primera capa oculta con 300 unidades.
    - `Dense(300, activation='relu')`: Segunda capa oculta.
    - `Dense(10, activation='softmax')`: Capa de salida.
- **Compilación**:
    - Optimizador: `adam`
    - Pérdida: `categorical_crossentropy`
    - Métricas: `accuracy`

3. Efectividad y Comparación de los Modelos

En términos de efectividad, el modelo CNN superó a los demás en casi todas las métricas clave, como se muestra en los resultados de validación:

- **Exactitud del Modelo CNN**: El modelo CNN alcanzó la mayor exactitud, demostrando una buena capacidad de generalización a través de las diferentes etiquetas.
- **Comparación con Otros Modelos**:
    - Los modelos MLP y `model_new1` también mostraron un rendimiento aceptable, pero con una menor precisión en comparación con el CNN.
    - El modelo simple tuvo el menor rendimiento, lo cual es esperable debido a su arquitectura más básica.

La siguiente matriz de confusión para el modelo CNN muestra que