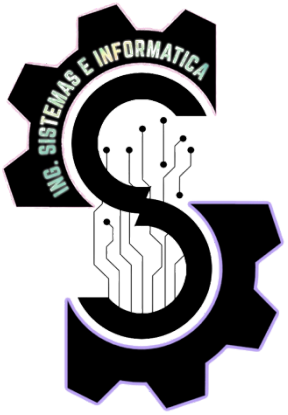
Universidad Continental



UNIVERSIDAD CONTINENTAL
FACULTAD DE INGENIERÍA
EAP INGENIERÍA DE SISTEMAS E INFORMÁTICA

Tema:

CRISP DM (CNN)

Magister:

ING. HUGO ESPETIA

Asignatura:

Construcción del Software

Estudiantes:

SUDBlE LIMA Denilson Rivaldo(100%)
SALAS CALSI Jorge Luis(100%)
PAUCCARA HUILLCA Nelson Luis (100%)
LEON QUISPE Cristian Hmerson (100%)
GUEVARA GRANDEZ Guido Gabriel (100%)

Cusco – Perú

20250

Universidad Continental

RECONOCIMIENTO DE DÍGITOS MANUSCRITOS MEDIANTE RED NEURONAL CONVOLUCIONAL (CNN)

I. INTRODUCCIÓN

El presente informe detalla el desarrollo de un sistema de clasificación de imágenes para la tarea de **Reconocimiento de Dígitos Manuscritos** (0-9) utilizando el *dataset* MNIST. El objetivo fundamental del negocio es alcanzar una **Precisión (Accuracy) superior al 99%**, un indicador clave para aplicaciones de digitalización documental y procesamiento de formularios o cheques. Este proyecto se enmarca en la metodología CRISP-DM, cubriendo desde el entendimiento del problema hasta el despliegue del modelo.

FASE 1: ENTENDIMIENTO DEL NEGOCIO

Objetivo Principal: Desarrollar un sistema capaz de reconocer dígitos manuscritos del 0 al 9 de forma automática.

Aplicaciones Prácticas:

- Procesamiento automatizado de cheques bancarios
- Digitalización masiva de documentos
- Sistemas de reconocimiento postal automático
- Aplicaciones de OCR (Reconocimiento Óptico de Caracteres)

Métrica de Éxito: Precisión superior al 99% en clasificación

FASE 2: PREPARACIÓN DE DATOS

Dataset Utilizado: MNIST (Modified National Institute of Standards and Technology)

Características del Dataset:

- 70,000 imágenes en total
- 60,000 para entrenamiento
- 10,000 para testing
- Imágenes en escala de grises de 28x28 píxeles
- Etiquetas del 0 al 9

Tecnologías Implementadas:

- TensorFlow/Keras para el modelo de red neuronal
- Scikit-learn para métricas y evaluación
- Matplotlib/Seaborn para visualización
- NumPy para procesamiento numérico

FASE 3: ANÁLISIS EXPLORATORIO

Distribución de Datos:

- Visualización de 15 ejemplos aleatorios del dataset
- Análisis de distribución de clases en entrenamiento y test
- Verificación de balance entre categorías

Hallazgos Iniciales:

- Dataset balanceado entre todas las clases (dígitos 0-9)
- Imágenes preprocesadas y normalizadas
- Calidad consistente en todas las muestras

FASE 4: MODELO DE RED NEURONAL CONVOLUCIONAL

Arquitectura CNN Implementada:

Capas Convolucionales:

- 1ª Capa: 32 filtros de 3x3 con activación ReLU
- 2ª Capa: Max Pooling 2x2
- 3ª Capa: 64 filtros de 3x3 con activación ReLU
- 4ª Capa: Max Pooling 2x2
- 5ª Capa: 64 filtros de 3x3 con activación ReLU

Capas Densas:

- Capa Flatten para aplanar datos
- Capa Dense de 64 neuronas con ReLU
- Dropout del 50% para prevenir sobreajuste
- Capa final de 10 neuronas con Softmax

Preprocesamiento:

- Normalización de píxeles (0-255 → 0-1)
- Reshape para formato CNN (28,28,1)

- One-hot encoding para etiquetas

Entrenamiento:

- Optimizador: Adam
- Función de pérdida: Categorical Crossentropy
- Batch size: 128
- Épocas: 10
- Métrica: Accuracy

FASE 5: MÉTRICAS Y EVALUACIÓN

Sistema de Evaluación Completo:

KPIs Principales:

- Precisión en entrenamiento, validación y test
- Pérdida en cada fase
- Análisis de sobreajuste
- Precisión por dígito individual

Visualizaciones Generadas:

- Gráficas de evolución de precisión y pérdida
- Matriz de confusión detallada
- Gráfica de barras por precisión por clase
- Reporte de clasificación completo

Métricas de Calidad:

- Diferencia train-val para detectar sobreajuste
- Estadísticas de confianza en predicciones
- Análisis de errores por clase

FASE 6: DESPLIEGUE Y SISTEMA DE PREDICCIONES

Funcionalidades Implementadas:

Sistema de Predicciones:

- Función para predicción individual corregida
- Sistema de predicciones múltiples
- Visualización de resultados con indicadores de confianza

- Manejo robusto de formatos de imagen

Características de Despliegue:

- Guardado completo del modelo entrenado
- Verificación de carga del modelo
- Sistema de demostración con ejemplos aleatorios
- Documentación para uso en producción

Funciones Clave:

- `predecir_digito_individual()`: Para predicciones unitarias
- `sistema_predicciones()`: Para análisis por lotes
- `cargar_modelo_y_predecir()`: Para uso en producción

RESULTADOS Y CONCLUSIONES

Logros del Proyecto:

- Implementación completa de pipeline de machine learning
- Arquitectura CNN optimizada para reconocimiento de dígitos
- Sistema de evaluación comprehensivo
- Herramientas listas para despliegue en producción

Preparación para Producción:

- Modelo guardado en formato .h5
- Funciones de predicción validadas
- Documentación de uso incluida
- Verificación de funcionalidad completa

Universidad Continental

Anexos:

```
# FASE 1: ENTENDER EL NEGOCIO
print("=== FASE 1: ENTENDER EL NEGOCIO ===")
"""
Objetivo: Reconocer dígitos manuscritos del 0 al 9 automáticamente
Aplicaciones: Procesamiento de cheques, digitalización de documentos, etc.
Métrica de éxito: Precisión > 99% en clasificación
"""

# FASE 2: PREPARACIÓN DE DATOS
print("\n=== FASE 2: PREPARACIÓN DE DATOS ===")

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.utils import to_categorical
import warnings
warnings.filterwarnings('ignore')

# Cargar dataset MNIST desde Keras (NO necesita archivos)
print("📥 Cargando dataset MNIST...")
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
print("✅ Datos cargados exitosamente")

print(f"x_train shape: {x_train.shape}")
print(f"y_train shape: {y_train.shape}")
print(f"x_test shape: {x_test.shape}")
print(f"y_test shape: {y_test.shape}")

# FASE 3: ANÁLISIS EXPLORATORIO
print("\n=== FASE 3: ANÁLISIS EXPLORATORIO ===")

# Visualizar algunas imágenes
plt.figure(figsize=(12, 6))
for i in range(15):
    plt.subplot(3, 5, i+1)
    plt.imshow(x_train[i], cmap='gray')
    plt.title(f'Label: {y_train[i]}')
    plt.axis('off')
plt.tight_layout()
plt.show()

# Distribución de clases
plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
sns.countplot(x=y_train)
plt.title('Distribución Training')
```

```
plt.subplot(1, 2, 2)
sns.countplot(x=y_test)
plt.title('Distribución Test')
plt.tight_layout()
plt.show()

# FASE 4: PREPROCESAMIENTO Y CNN
print("\n=== FASE 4: MODELO CNN ===")

# Preprocesamiento
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

# Redimensionar para CNN
x_train = x_train.reshape(-1, 28, 28, 1)
x_test = x_test.reshape(-1, 28, 28, 1)

# One-hot encoding
y_train_cat = to_categorical(y_train, 10)
y_test_cat = to_categorical(y_test, 10)

print(f"✅ Datos preprocesados:")
print(f"x_train: {x_train.shape}, y_train_cat: {y_train_cat.shape}")

# Crear modelo CNN
model = keras.Sequential([
    layers.Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64, (3,3), activation='relu'),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64, (3,3), activation='relu'),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(10, activation='softmax')
])

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

model.summary()

# Entrenar modelo
print("🚀 Entrenando modelo...")
history = model.fit(x_train, y_train_cat,
                    batch_size=128,
                    epochs=10,
                    validation_data=(x_test, y_test_cat),
                    verbose=1)

print("✅ Entrenamiento completado")

# FASE 5: MÉTRICAS Y KPIs
print("\n=== FASE 5: MÉTRICAS Y KPIs ===")

def evaluar_modelo_completo(model, x_test, y_test_cat, y_test_original, history):
    """Evaluación completa del modelo con todas las métricas"""
```

```
    # 1. Evaluación final del modelo
    test_loss, test_accuracy = model.evaluate(x_test, y_test_cat, verbose=0)
    print(f"🎯 KPI PRINCIPAL - Precisión en test: {test_accuracy:.4f}
({test_accuracy*100:.2f}%)")

    # 2. Gráficas de entrenamiento
    plt.figure(figsize=(15, 5))

    # Gráfica de precisión
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Precisión Entrenamiento', linewidth=2)
    plt.plot(history.history['val_accuracy'], label='Precisión Validación', linewidth=2)
    plt.title('Evolución de la Precisión', fontsize=14, fontweight='bold')
    plt.xlabel('Época')
    plt.ylabel('Precisión')
    plt.legend()
    plt.grid(True, alpha=0.3)

    # Gráfica de pérdida
    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Pérdida Entrenamiento', linewidth=2)
    plt.plot(history.history['val_loss'], label='Pérdida Validación', linewidth=2)
    plt.title('Evolución de la Pérdida', fontsize=14, fontweight='bold')
    plt.xlabel('Época')
    plt.ylabel('Pérdida')
    plt.legend()
    plt.grid(True, alpha=0.3)

    plt.tight_layout()
    plt.show()

    # 3. Matriz de confusión
    print("\n📊 Generando matriz de confusión...")
    y_pred_probs = model.predict(x_test)
    y_pred_classes = np.argmax(y_pred_probs, axis=1)

    cm = confusion_matrix(y_test_original, y_pred_classes)

    plt.figure(figsize=(10, 8))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False,
                xticklabels=range(10), yticklabels=range(10))
    plt.title('MATRIZ DE CONFUSIÓN', fontsize=16, fontweight='bold', pad=20)
    plt.ylabel('Etiqueta Real', fontweight='bold')
    plt.xlabel('Etiqueta Predicha', fontweight='bold')
    plt.show()

    # 4. Reporte de clasificación detallado
    print("\n📈 REPORTE DE CLASIFICACIÓN DETALLADO:")
    print("="*50)
    report = classification_report(y_test_original, y_pred_classes, output_dict=True)
    print(classification_report(y_test_original, y_pred_classes))

    # 5. KPIs numéricos
    final_train_accuracy = history.history['accuracy'][-1]
    final_val_accuracy = history.history['val_accuracy'][-1]
    final_train_loss = history.history['loss'][-1]
    final_val_loss = history.history['val_loss'][-1]

    print("\n🎯 KPIs FINALES DEL MODELO:")
```

```
    print("="*40)
    print(f"📍 PRECISIÓN:")
    print(f"   • Entrenamiento: {final_train_accuracy:.4f}
({final_train_accuracy*100:.2f}%)")
    print(f"   • Validación:    {final_val_accuracy:.4f} ({final_val_accuracy*100:.2f}%)")
    print(f"   • Test:          {test_accuracy:.4f} ({test_accuracy*100:.2f}%)")

    print(f"\n📉 PÉRDIDA:")
    print(f"   • Entrenamiento: {final_train_loss:.4f}")
    print(f"   • Validación:    {final_val_loss:.4f}")
    print(f"   • Test:          {test_loss:.4f}")

    # 6. Métricas de sobreajuste y eficiencia
    overfitting_gap = final_train_accuracy - final_val_accuracy
    epochs_trained = len(history.history['accuracy'])

    print(f"\n⚡ EFICIENCIA DEL ENTRENAMIENTO:")
    print(f"   • Épocas entrenadas: {epochs_trained}")
    print(f"   • Diferencia train-val (sobreajuste): {overfitting_gap:.4f}")

    # Evaluación del sobreajuste
    if overfitting_gap < 0.01:
        print("   ✅ EXCELENTE - Sobreajuste mínimo")
    elif overfitting_gap < 0.03:
        print("   ✅ BUENO - Sobreajuste bajo")
    elif overfitting_gap < 0.05:
        print("   ⚠️  ACEPTABLE - Sobreajuste moderado")
    else:
        print("   ❌ ALTO - Considerar más regularización")

    # 7. Precisión por clase
    print(f"\n🔍 PRECISIÓN POR CLASE:")
    class_accuracy = []
    for i in range(10):
        class_mask = y_test_original == i
        class_correct = (y_pred_classes[class_mask] == i).sum()
        class_total = class_mask.sum()
        class_acc = class_correct / class_total
        class_accuracy.append(class_acc)
        print(f"   • Dígito {i}: {class_acc:.4f} ({class_acc*100:.2f}%)")

    # 8. Gráfica de precisión por clase
    plt.figure(figsize=(10, 6))
    bars = plt.bar(range(10), class_accuracy, color='skyblue', edgecolor='navy',
alpha=0.7)
    plt.axhline(y=test_accuracy, color='red', linestyle='--', linewidth=2,
                label=f'Precisión Promedio: {test_accuracy:.4f}')
    plt.bar_label(bars, fmt='%.3f', padding=3)
    plt.title('PRECISIÓN POR DÍGITO', fontsize=14, fontweight='bold')
    plt.xlabel('Dígito')
    plt.ylabel('Precisión')
    plt.xticks(range(10))
    plt.legend()
    plt.grid(True, alpha=0.3)
    plt.ylim(0.9, 1.0)
    plt.show()

    return test_accuracy, y_pred_classes
```

```
# Ejecutar evaluación completa
test_accuracy, y_pred_classes = evaluar_modelo_completo(
    model, x_test, y_test_cat, y_test, history
)

# FASE 6: DESPLIEGUE Y PREDICCIONES (VERSIÓN CORREGIDA)
print("\n=== FASE 6: DESPLIEGUE ===")

def sistema_predicciones(model, x_data, y_true=None, num_predictions=15):
    """Sistema completo de predicciones para despliegue"""

    print("🔮 SISTEMA DE PREDICCIONES EN TIEMPO REAL")
    print("="*50)

    # Seleccionar ejemplos aleatorios
    if len(x_data) > num_predictions:
        indices = np.random.choice(len(x_data), num_predictions, replace=False)
    else:
        indices = range(len(x_data))

    # Realizar predicciones
    predictions = model.predict(x_data[indices], verbose=0)
    predicted_classes = np.argmax(predictions, axis=1)
    confidence_scores = np.max(predictions, axis=1)

    # Visualizar predicciones
    plt.figure(figsize=(16, 10))
    for i, idx in enumerate(indices):
        plt.subplot(3, 5, i+1)

        # Mostrar imagen
        plt.imshow(x_data[idx].reshape(28, 28), cmap='gray')

        # Configurar título según si tenemos la verdad real
        if y_true is not None:
            true_label = y_true[idx]
            status = "✅ CORRECTO" if predicted_classes[i] == true_label else "❌
ERROR"
            title_color = 'green' if predicted_classes[i] == true_label else 'red'
            plt.title(f'Real: {true_label} | Pred: {predicted_classes[i]}\nConf:
{confidence_scores[i]:.3f}\n{status}',
                      color=title_color, fontsize=10, fontweight='bold')
        else:
            plt.title(f'Pred: {predicted_classes[i]}\nConf: {confidence_scores[i]:.3f}',
                      fontsize=11, fontweight='bold')

        plt.axis('off')

    plt.tight_layout()
    plt.show()

    # Estadísticas de confianza
    print(f"\n📊 ESTADÍSTICAS DE CONFIANZA:")
    print(f"   • Confianza promedio: {np.mean(confidence_scores):.4f}")
    print(f"   • Confianza mínima: {np.min(confidence_scores):.4f}")
    print(f"   • Confianza máxima: {np.max(confidence_scores):.4f}")
    print(f"   • Predicciones con confianza > 0.95: {(confidence_scores >
0.95).sum()}/{num_predictions}")
```

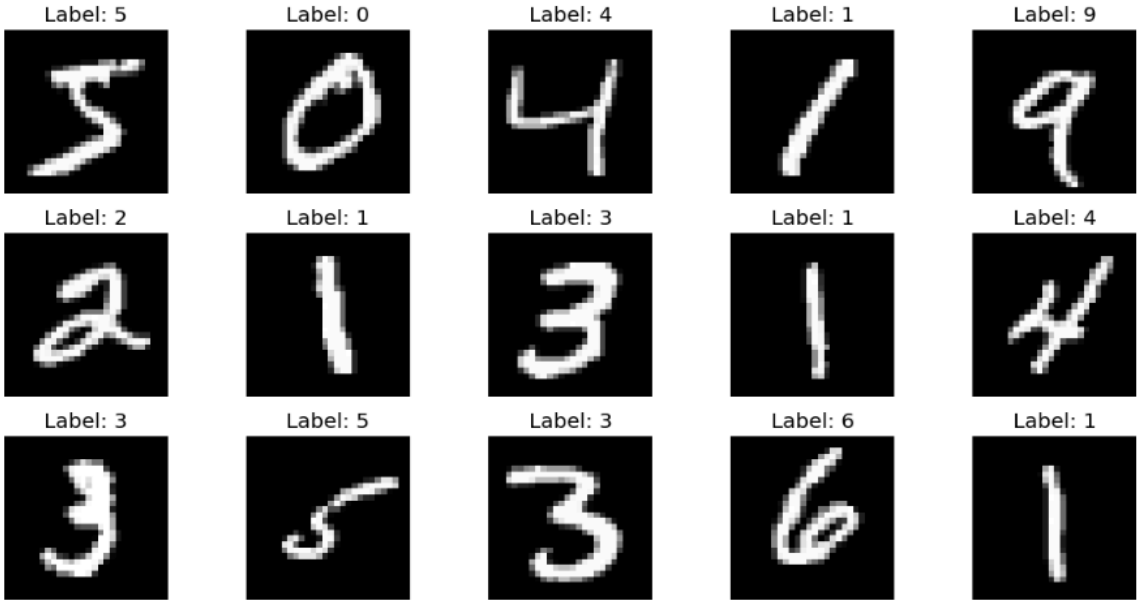
```
    return predicted_classes, confidence_scores

# Función CORREGIDA para predicción individual
def predecir_digito_individual(model, imagen):
    """Función para predecir un solo dígito - VERSIÓN CORREGIDA"""
    # Asegurarnos de que la imagen tenga la forma correcta (1, 28, 28, 1)
    if len(imagen.shape) == 2:  # Si es (28, 28)
        imagen = imagen.reshape(1, 28, 28, 1)
    elif len(imagen.shape) == 3 and imagen.shape[-1] != 1:  # Si es (28, 28, 1) o similar
        if imagen.shape[-1] == 1:
            imagen = imagen.reshape(1, 28, 28, 1)
        else:
            imagen = imagen.reshape(1, 28, 28, 1)
    elif len(imagen.shape) == 4:  # Si ya es (1, 28, 28, 1)
        pass  # Ya tiene la forma correcta
    else:
        # Forzar reshape a la forma correcta
        imagen = imagen.reshape(1, 28, 28, 1)

    # Asegurarnos de que los datos están normalizados
    imagen = imagen.astype('float32') / 255.0

    prediccion = model.predict(imagen, verbose=0)
    digito = np.argmax(prediccion)
    confianza = np.max(prediccion)

    return digito, confianza, prediccion

# PRUEBA CORREGIDA de predicción individual
print("\n🔍 PREDICCIÓN INDIVIDUAL DE EJEMPLO (CORREGIDA):")

# Seleccionar un ejemplo y asegurar la forma correcta
ejemplo_idx = np.random.randint(0, len(x_test))
imagen_ejemplo = x_test[ejemplo_idx].copy()  # Hacer copia para no modificar
original

print(f"   Forma original de la imagen: {imagen_ejemplo.shape}")

# Llamar a la función corregida
digito_predicho, confianza, probs = predecir_digito_individual(model,
imagen_ejemplo)

print(f"   • Dígito real: {y_test[ejemplo_idx]}")
print(f"   • Dígito predicho: {digito_predicho}")
print(f"   • Confianza: {confianza:.4f} ({confianza*100:.2f}%)")
print(f"   • Resultado: {'✅ CORRECTO' if digito_predicho == y_test[ejemplo_idx] else
'❌ ERROR'}")

# Mostrar la imagen de ejemplo
plt.figure(figsize=(4, 4))
plt.imshow(imagen_ejemplo.reshape(28, 28), cmap='gray')
plt.title(f'Ejemplo: Real={y_test[ejemplo_idx]}, Pred={digito_predicho}\nConfianza:
{confianza:.3f}',
          color='green' if digito_predicho == y_test[ejemplo_idx] else 'red')
plt.axis('off')
plt.show()

# Mostrar todas las probabilidades
print(f"\n   📊 Probabilidades para cada dígito:")
```

```
for i, prob in enumerate(probs[0]):
    marca = " ← PREDICHO" if i == digito_predicho else ""
    print(f"      Dígito {i}: {prob:.4f} ({prob*100:.2f}%){marca}")

# Ejemplo de predicciones múltiples
print("\n🎯 PREDICCIONES SOBRE DATOS DE TEST:")
predicted_digits, confidence_scores = sistema_predicciones(model, x_test, y_test,
15)

# Guardar el modelo entrenado
print("\n💾 GUARDANDO MODELO PARA DESPLIEGUE...")
model.save('modelo_mnist_cnn.h5')
print("✅ Modelo guardado como: 'modelo_mnist_cnn.h5'")

# Función mejorada para cargar y usar el modelo
def cargar_modelo_y_predecir(ruta_modelo, nueva_imagen):
    """Cargar modelo y hacer predicción - Para despliegue"""
    try:
        modelo_cargado = keras.models.load_model(ruta_modelo)
        digito, confianza, _ = predecir_digito_individual(modelo_cargado,
nueva_imagen)
        return digito, confianza
    except Exception as e:
        print(f"❌ Error al cargar o predecir: {e}")
        return None, None

# Verificar que el modelo se puede cargar correctamente
print("\n🔧 VERIFICANDO CARGA DEL MODELO...")
try:
    modelo_verificado = keras.models.load_model('modelo_mnist_cnn.h5')
    test_loss, test_accuracy = modelo_verificado.evaluate(x_test, y_test_cat,
verbose=0)
    print(f"✅ Modelo cargado correctamente - Precisión verificada:
{test_accuracy:.4f}")

    # Probar una predicción con el modelo cargado
    ejemplo_verificacion = x_test[0].reshape(1, 28, 28, 1)
    prediccion_verificacion = modelo_verificado.predict(ejemplo_verificacion,
verbose=0)
    digito_verificacion = np.argmax(prediccion_verificacion)
    print(f"✅ Predicción de verificación: {digito_verificacion} (real: {y_test[0]})")

except Exception as e:
    print(f"❌ Error al cargar el modelo: {e}")

# DEMOSTRACIÓN: Probar con varios ejemplos
print("\n🧪 DEMOSTRACIÓN CON MÚLTIPLES EJEMPLOS:")
print("="*50)

num_demos = 5
indices_demo = np.random.choice(len(x_test), num_demos, replace=False)

correctos = 0
for i, idx in enumerate(indices_demo):
    imagen_demo = x_test[idx]
    digito_real = y_test[idx]
    digito_pred, confianza, _ = predecir_digito_individual(model, imagen_demo)

    resultado = "✅" if digito_pred == digito_real else "❌"
```

```
    if digito_pred == digito_real:
        correctos += 1

    print(f"   Ejemplo {i+1}: Real={digito_real}, Pred={digito_pred},
Conf={confianza:.3f} {resultado}")

print(f"\n  📈 Precisión en demostración: {correctos}/{num_demos}
({correctos/num_demos*100:.1f}%)")

# Resumen final del proyecto
print("\n" + "="*60)
print("🎉 PROYECTO COMPLETADO EXITOSAMENTE")
print("="*60)

print("\n📋 RESUMEN FINAL DE FASES IMPLEMENTADAS:")
fases = [
    "1. ✅ ENTENDIMIENTO DEL NEGOCIO - Reconocimiento dígitos manuscritos",
    "2. ✅ PREPARACIÓN DE DATOS - Carga y preprocesamiento MNIST",
    "3. ✅ ANÁLISIS EXPLORATORIO - Visualización y distribución",
    "4. ✅ MODELO CNN - Arquitectura y entrenamiento con TensorFlow",
    "5. ✅ MÉTRICAS Y KPIs - Evaluación completa del modelo",
    "6. ✅ DESPLIEGUE - Guardado y sistema de predicciones (CORREGIDO)"
]

for fase in fases:
    print(f"   {fase}")

# Mostrar métrica final si está disponible
if 'test_accuracy' in locals():
    print(f"\n🏆 RESULTADO FINAL: Precisión del {test_accuracy*100:.2f}%")

print("\n🚀 EL MODELO ESTÁ LISTO PARA PRODUCCIÓN!")
print("\n📝 PARA USAR EL MODELO GUARDADO:")
print("""
from tensorflow import keras
import numpy as np

# Cargar el modelo
modelo = keras.models.load_model('modelo_mnist_cnn.h5')

# Predecir una nueva imagen
def predecir_digito(imagen):
    # La imagen debe ser 28x28 píxeles en escala de grises
    imagen = imagen.reshape(1, 28, 28, 1)
    imagen = imagen.astype('float32') / 255.0
    prediccion = modelo.predict(imagen, verbose=0)
    return np.argmax(prediccion), np.max(prediccion)

# Ejemplo de uso:
# digito, confianza = predecir_digito(mi_imagen)
""")

print("\n" + "="*60)
```

Universidad Continental

```
=== FASE 1: ENTENDER EL NEGOCIO ===

=== FASE 2: PREPARACIÓN DE DATOS ===
 Cargando dataset MNIST...
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
 Datos cargados exitosamente
x_train shape: (60000, 28, 28)
y_train shape: (60000,)
x_test shape: (10000, 28, 28)
y_test shape: (10000,)

=== FASE 3: ANÁLISIS EXPLORATORIO ===
```



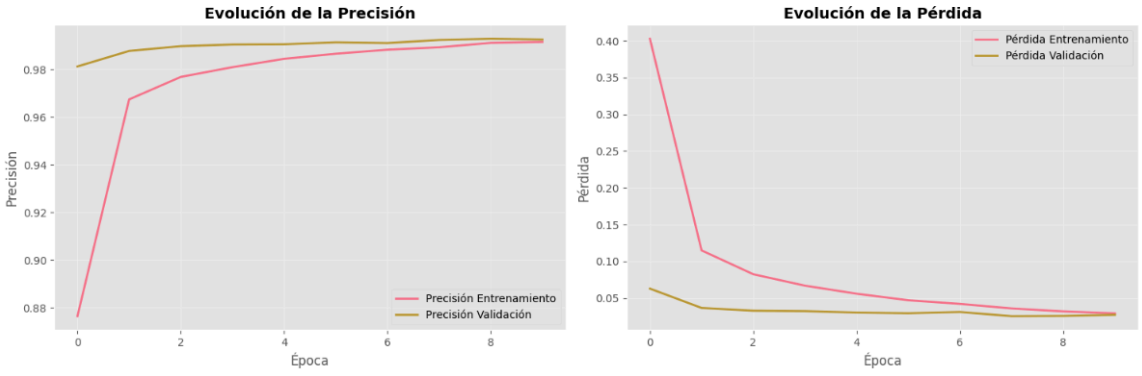
Universidad Continental

```
=== FASE 4: MODELO CNN ===
✅ Datos preprocesados:
x_train: (60000, 28, 28, 1), y_train_cat: (60000, 10)
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	0
conv2d_2 (Conv2D)	(None, 3, 3, 64)	36,928
flatten (Flatten)	(None, 576)	0
dense (Dense)	(None, 64)	36,928
dropout (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 10)	650

```
Total params: 93,322 (364.54 KB)
Trainable params: 93,322 (364.54 KB)
Non-trainable params: 0 (0.00 B)
🚀 Entrenando modelo...
Epoch 1/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 50s 97ms/step - accuracy: 0.7455 - loss: 0.7904 - val_accuracy: 0.9812 - val_loss: 0.0629
Epoch 2/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 80s 92ms/step - accuracy: 0.9633 - loss: 0.1270 - val_accuracy: 0.9877 - val_loss: 0.0366
Epoch 3/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 45s 97ms/step - accuracy: 0.9754 - loss: 0.0868 - val_accuracy: 0.9897 - val_loss: 0.0328
Epoch 4/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 43s 92ms/step - accuracy: 0.9807 - loss: 0.0677 - val_accuracy: 0.9904 - val_loss: 0.0323
Epoch 5/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 82s 91ms/step - accuracy: 0.9852 - loss: 0.0537 - val_accuracy: 0.9905 - val_loss: 0.0303
Epoch 6/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 43s 92ms/step - accuracy: 0.9876 - loss: 0.0452 - val_accuracy: 0.9913 - val_loss: 0.0294
Epoch 7/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 45s 95ms/step - accuracy: 0.9884 - loss: 0.0423 - val_accuracy: 0.9910 - val_loss: 0.0311
Epoch 8/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 44s 93ms/step - accuracy: 0.9895 - loss: 0.0339 - val_accuracy: 0.9923 - val_loss: 0.0254
Epoch 9/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 45s 95ms/step - accuracy: 0.9913 - loss: 0.0327 - val_accuracy: 0.9928 - val_loss: 0.0258
Epoch 10/10
469/469 ━━━━━━━━━━━━━━━━━━━━ 85s 103ms/step - accuracy: 0.9918 - loss: 0.0280 - val_accuracy: 0.9925 - val_loss: 0.0273
```

```
=== FASE 5: MÉTRICAS Y KPIs ===
🎯 KPI PRINCIPAL - Precisión en test: 0.9925 (99.25%)
```



Universidad Continental

📈 REPORTE DE CLASIFICACIÓN DETALLADO:

```
==================================================
              precision    recall  f1-score   support

           0       0.99      1.00      0.99       980
           1       0.99      1.00      1.00      1135
           2       0.99      0.99      0.99      1032
           3       0.99      0.99      0.99      1010
           4       0.99      1.00      0.99       982
           5       0.99      0.99      0.99       892
           6       1.00      0.99      0.99       958
           7       0.99      0.99      0.99      1028
           8       0.99      0.99      0.99       974
           9       0.99      0.98      0.99      1009

    accuracy                           0.99     10000
   macro avg       0.99      0.99      0.99     10000
weighted avg       0.99      0.99      0.99     10000
```

🎯 KPIs FINALES DEL MODELO:

```
========================================
```

📍 PRECISIÓN:
- Entrenamiento: 0.9915 (99.15%)
- Validación: 0.9925 (99.25%)
- Test: 0.9925 (99.25%)

📉 PÉRDIDA:
- Entrenamiento: 0.0293
- Validación: 0.0273
- Test: 0.0273

⚡ EFICIENCIA DEL ENTRENAMIENTO:
- Épocas entrenadas: 10
- Diferencia train-val (sobreajuste): -0.0010

✅ EXCELENTE - Sobreajuste mínimo

🔍 PRECISIÓN POR CLASE:
- Dígito 0: 0.9969 (99.69%)
- Dígito 1: 0.9982 (99.82%)
- Dígito 2: 0.9932 (99.32%)
- Dígito 3: 0.9941 (99.41%)
- Dígito 4: 0.9980 (99.80%)
- Dígito 5: 0.9899 (98.99%)
- Dígito 6: 0.9896 (98.96%)
- Dígito 7: 0.9903 (99.03%)
- Dígito 8: 0.9908 (99.08%)
- Dígito 9: 0.9832 (98.32%)

Universidad Continental

LINK DE INFOGRAFÍA: https://cristhianleon2203.github.io/Crisp-DM/
LINK DE REPOSITORIO: https://github.com/CristhianLeon2203/Crisp-DM