CORPORACION UNIVERSITARIA AUTÒNOMA DEL CAUCA

CLASIFICADOR DE DIGITOS PROPIOS

ENTREGABLE Nº6 TALLER

CRISTIAN CAMILO MARTINEZ CORDOBA

ELECTIVA II

INGENIERÌA ELECTRONICA

MAYO 2025

DESCRIPCION DEL DATASET.

El Dataset utilizado en este proyecto consiste en imágenes de dígitos manuscritos del 0 al 9, organizadas en carpetas por clase. Cada carpeta contiene imágenes correspondientes a un único dígito, permitiendo una clasificación supervisada. Las imágenes se encuentran en formato .jpg y presentan variaciones en escritura. Durante el pre procesamiento, todas las imágenes fueron convertidas a escala de grises para reducir la dimensión y el costo computacional, sin perder la información relevante de forma. Posteriormente, se redimensionaron a un tamaño uniforme de 32x32 píxeles para facilitar su uso como entrada en redes neuronales convolucionales (CNN). Finalmente, los valores de píxeles fueron normalizados dividiendo por 255, lo que lleva todos los valores al rango [0.0, 1.0], mejorando la estabilidad y velocidad del entrenamiento.

El conjunto total de datos fue dividido en dos subconjuntos: 80% para entrenamiento y 20% para prueba, asegurando una distribución equilibrada de clases en ambos conjuntos. Esto permite entrenar la red neuronal y luego evaluar su rendimiento en datos no vistos, garantizando una medición justa de su capacidad de generalización.

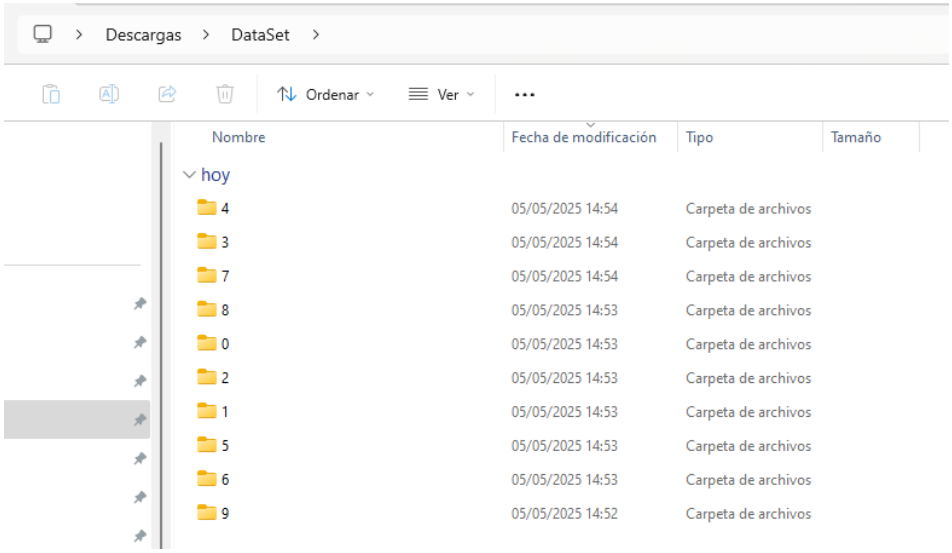

Fig 1. Dataset Implementado.

ARQUITECTURA DE LA CNN.

La red neuronal convolucional (CNN) utilizada en este proyecto fue diseñada para clasificar imágenes de dígitos manuscritos en una de las diez clases posibles, correspondientes a los números del 0 al 9. La red toma como entrada imágenes en escala de grises de tamaño 32 x 32 píxeles, por lo que su forma de entrada es de 32 x 32 x 1. La arquitectura comienza con una primera capa convolucional compuesta por 32 filtros de tamaño 3 x 3, seguida de una capa de normalización por lotes que mejora la estabilidad del entrenamiento. Posteriormente, se aplica una capa de agrupamiento máximo con tamaño 2 x 2 para reducir la dimensión espacial, y una capa de Dropout con una tasa del 25% para prevenir el sobreajuste.

A continuación, se incluye una segunda capa convolucional con 64 filtros de tamaño 3 x 3, también acompañada por Batch Normalization, MaxPooling de 2x2 y Dropout con una tasa del 25%. La salida de estas capas convolucionales se aplana mediante una capa Flatten para convertirse en un vector unidimensional, el cual se conecta a una capa densa oculta de 128 neuronas, seguida nuevamente de Batch Normalization y Dropout con un 50% de desactivación. Finalmente, la capa de salida contiene 10 neuronas, una por cada clase, y utiliza una función de

activación para obtener probabilidades asociadas a cada categoría. El modelo fue compilado utilizando el optimizador Adam, la función de pérdida categórica cruzada y la métrica de precisión accuracy. Esta arquitectura es lo suficientemente profunda para extraer características relevantes de las imágenes, pero también incluye mecanismos para evitar el sobreajuste, como la normalización y el dropout.

RESULTADOS Y ANALISIS.

Luego del entrenamiento del modelo de red neuronal convolucional (CNN), se evaluó su desempeño sobre el conjunto de prueba. Como se puede observar en la matriz de confusión, el modelo presenta un desempeño deficiente, ya que clasifica incorrectamente la mayoría de las imágenes. En este caso particular, todos los dígitos fueron clasificados como la clase "3", lo que indica que la red no logró aprender una representación adecuada de las diferentes clases.

Esta conclusión se refuerza con las gráficas de precisión (accuracy) y pérdida (loss) por época. En la curva de precisión, la precisión del conjunto de validación permanece en 0% durante todas las épocas, mientras que la precisión de entrenamiento muestra una ligera mejora, alcanzando un máximo de apenas 20%. Por otro lado, la pérdida de validación se mantiene alta y casi constante, lo que evidencia que el modelo no está generalizando correctamente y probablemente esté estancado desde el inicio del entrenamiento.

Este comportamiento puede deberse a diversas causas, tales como: un dataset demasiado pequeño, imágenes con poca variabilidad o mala calidad, arquitectura del modelo aún insuficiente para capturar las características necesarias o parámetros no óptimos o necesidad de más o menos épocas de entrenamiento. Como resultado, el modelo no es actualmente útil para realizar predicciones confiables. Será necesario revisar la calidad del dataset, realizar un mejor pre procesamiento, aumentar la cantidad de datos y posiblemente ajustar la arquitectura del modelo para mejorar el rendimiento.

CONCLUSIONES Y MEJORAS.

En esta práctica se implementó una red neuronal convolucional (CNN) para el reconocimiento automático de dígitos utilizando un conjunto de imágenes etiquetadas. El modelo se entrenó con una arquitectura compuesta por dos capas convolucionales, normalización por lotes, max pooling, capas densas y funciones de activación ReLU y softmax. También se aplicaron técnicas de pre procesamiento como conversión a escala de grises, redimensionamiento, normalización y aumento de datos. Sin embargo, los resultados obtenidos fueron insatisfactorios. La precisión del modelo en el conjunto de validación fue prácticamente nula, y la matriz de confusión mostró que el modelo clasificó la mayoría de las imágenes en una sola clase. Esto indica que la red no logró aprender patrones representativos ni diferenciar adecuadamente entre las distintas clases.

A partir de estos resultados, se proponen las siguientes mejoras:

- Aumentar el tamaño del dataset.
- Modificar el equilibrio entre clases.
- Mejorar la calidad de las imágenes.
- Ajustar la arquitectura del modelo.
- Probar más épocas y ajustes.

En conclusión, si bien se logró implementar la arquitectura base y cumplir con los requisitos técnicos del proyecto, el desempeño actual del modelo evidencia que aún se requieren mejoras significativas para lograr un sistema eficaz de reconocimiento de dígitos.

CODIGO IMPLEMENTADO.

```
      import numpy as np
import os
import cv2
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,
Dropout, BatchNormalization
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import confusion_matrix
import seaborn as sns

# Ruta del dataset
ruta_dataset = r"C:\Users\Darcy Sorany\Downloads\DataSet"

# Listas para almacenar imágenes y etiquetas
imagenes = []
etiquetas = []

# Cargar las imágenes de cada dígito (0-9)
for etiqueta in range(10):
    carpeta_digito = os.path.join(ruta_dataset, str(etiqueta))
    for imagen_nombre in os.listdir(carpeta_digito):
        if imagen_nombre.endswith(".jpg"):  # Solo cargar JPEG if
imagen_nombre.lower().endswith((".jpg", ".jpeg", ".png")):

            imagen = cv2.imread(os.path.join(carpeta_digito,
imagen_nombre))
            if imagen is not None:  # Verificar que la imagen se cargó
correctamente
                imagen_gris = cv2.cvtColor(imagen, cv2.COLOR_BGR2GRAY)
                imagen_redimensionada = cv2.resize(imagen_gris, (32,
32))  # Tamaño 32x32
                imagen_normalizada = imagen_redimensionada / 255.0  #
Normalizar
```

```
                imagenes.append(imagen_normalizada)
                etiquetas.append(etiqueta)

# Convertir listas a arrays de NumPy
imagenes = np.array(imagenes)
etiquetas = np.array(etiquetas)

# División en entrenamiento y prueba (80%-20%)
imagenes_train, imagenes_test, etiquetas_train, etiquetas_test =
train_test_split(
    imagenes, etiquetas, test_size=0.2, random_state=42)

# Expandir dimensiones para que tengan canal (32,32,1)
imagenes_train = np.expand_dims(imagenes_train, axis=-1)
imagenes_test = np.expand_dims(imagenes_test, axis=-1)

# Etiquetas en one-hot encoding
etiquetas_train = to_categorical(etiquetas_train, 10)
etiquetas_test = to_categorical(etiquetas_test, 10)

# Aumento de datos
datagen = ImageDataGenerator(
    rotation_range=10,
    zoom_range=0.1,
    width_shift_range=0.1,
    height_shift_range=0.1)

datagen.fit(imagenes_train)

# Crear el modelo CNN mejorado
modelo = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 1)),
    BatchNormalization(),
    MaxPooling2D((2, 2)),
    Dropout(0.25),

    Conv2D(64, (3, 3), activation='relu'),
    BatchNormalization(),
    MaxPooling2D((2, 2)),
    Dropout(0.25),

    Flatten(),
    Dense(128, activation='relu'),
    BatchNormalization(),
    Dropout(0.5),
    Dense(10, activation='softmax')
])

# Compilar modelo
```

```
optimizer = Adam(learning_rate=0.001)
modelo.compile(optimizer=optimizer,
               loss='categorical_crossentropy',
               metrics=['accuracy'])

# Callbacks
callbacks = [
    EarlyStopping(patience=5, monitor='val_loss',
restore_best_weights=True),
    ModelCheckpoint('best_model.h5', monitor='val_accuracy',
save_best_only=True)
]

# Entrenar modelo
history = modelo.fit(datagen.flow(imagenes_train, etiquetas_train,
batch_size=32),
                     epochs=10,
                     validation_data=(imagenes_test, etiquetas_test),
                     callbacks=callbacks)

# Cargar el mejor modelo
modelo = tf.keras.models.load_model('best_model.h5')

# Evaluación del modelo
test_loss, test_acc = modelo.evaluate(imagenes_test, etiquetas_test,
verbose=0)
print(f"\nPrecisión en prueba: {test_acc:.4f}")
print(f"Pérdida en prueba: {test_loss:.4f}")

# --- MATRIZ DE CONFUSIÓN (REQUISITO) ---
y_pred = np.argmax(modelo.predict(imagenes_test), axis=1)
y_true = np.argmax(etiquetas_test, axis=1)

cm = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicho')
plt.ylabel('Real')
plt.title('Matriz de Confusión')
plt.show()

# Visualización de resultados
def plot_results(history):
    plt.figure(figsize=(12, 4))

    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Train Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation
Accuracy')
```

```
    plt.title('Accuracy over epochs')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend()

    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.title('Loss over epochs')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend()

    plt.tight_layout()
    plt.show()

plot_results(history)

# Visualización de ejemplos
def plot_examples(images, true_labels, pred_labels, num_examples=4,
title=""):
    indices = np.random.choice(range(len(images)), num_examples)
    plt.figure(figsize=(10, 4))
    plt.suptitle(title)
    for i, idx in enumerate(indices):
        plt.subplot(1, num_examples, i+1)
        plt.imshow(images[idx].reshape(32, 32), cmap='gray')
        plt.title(f'Real: {true_labels[idx]}\nPred: {pred_labels[idx]}')
        plt.axis('off')
    plt.tight_layout()
    plt.show()

# Ejemplos correctos
correct_idx = np.where(y_pred == y_true)[0]
plot_examples(imagenes_test[correct_idx],
            y_true[correct_idx],
            y_pred[correct_idx],
            title="Ejemplos correctamente clasificados")

# Ejemplos incorrectos
incorrect_idx = np.where(y_pred != y_true)[0]
plot_examples(imagenes_test[incorrect_idx],
            y_true[incorrect_idx],
            y_pred[incorrect_idx],
            title="Ejemplos mal clasificados")
```

EVIDENCIA DE LOS RESULTADOS DEL CODIGO.

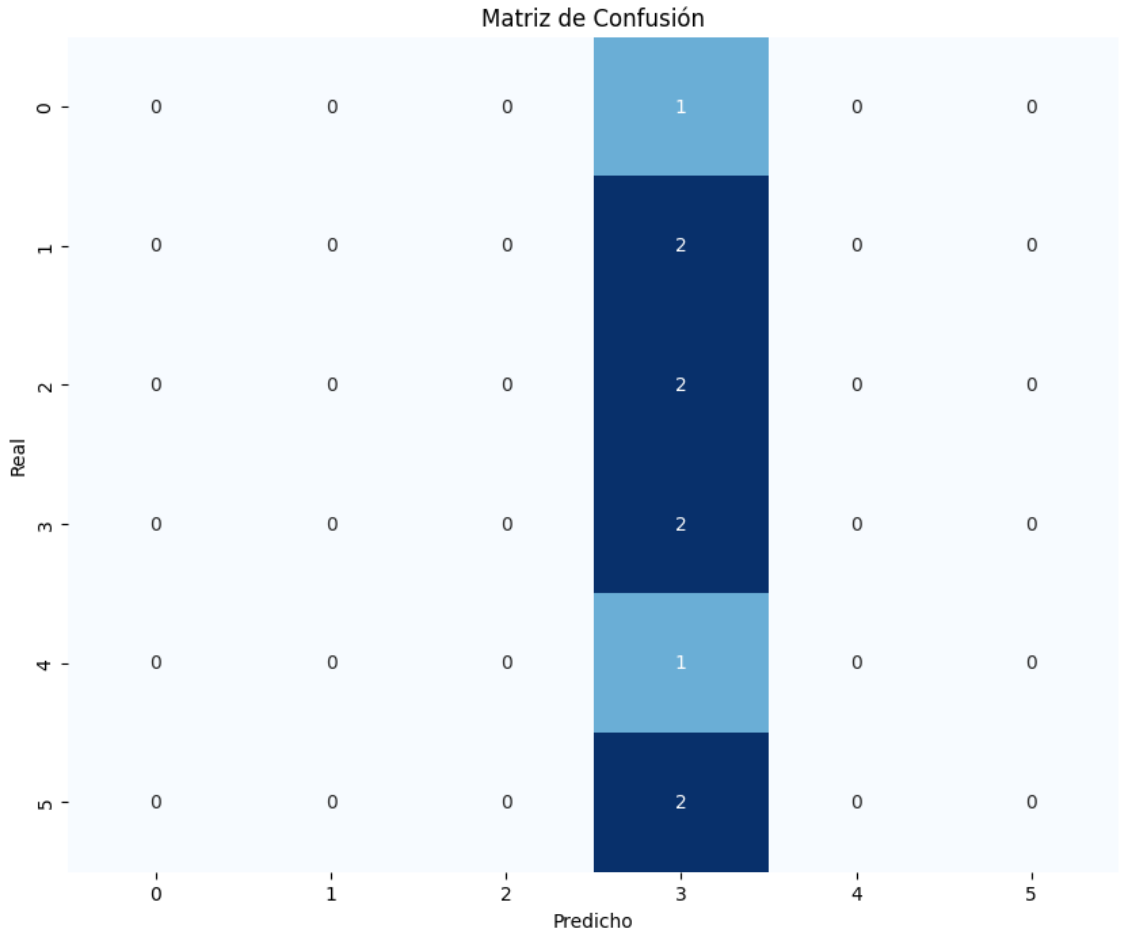

Fig 2. Matriz de Confusión del Sistema.

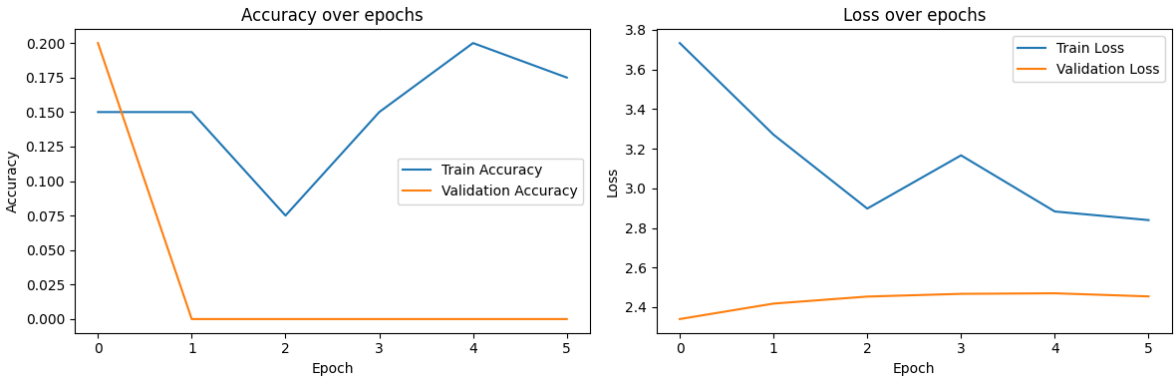

Fig 3. Diagramas de Aciertos y Perdidas del Sistema.

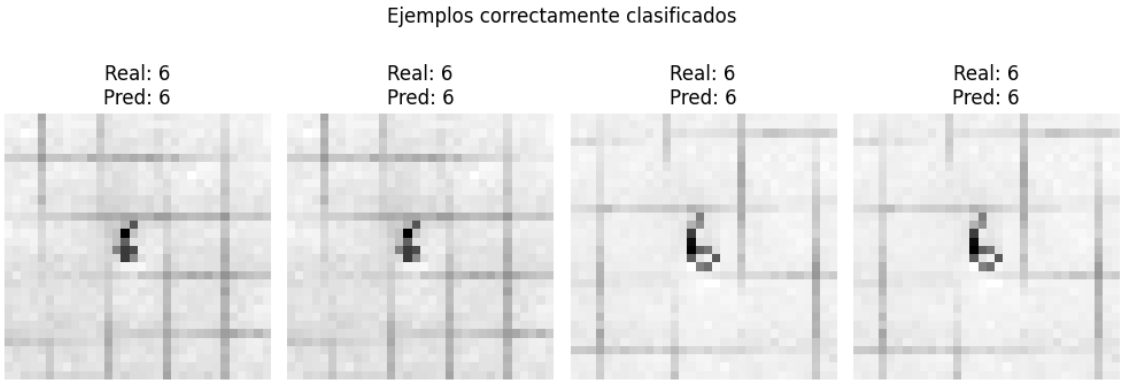

Fig 4. Ejemplos Clasificados Correctamente.

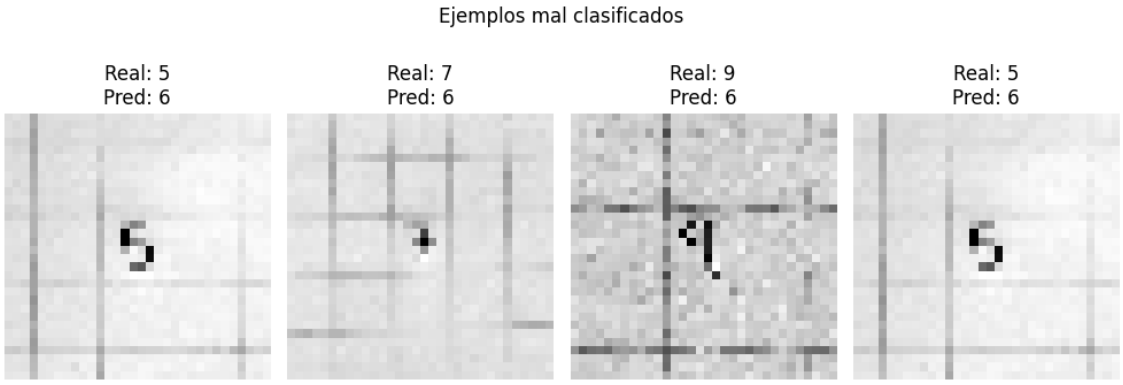

Fig 5. Ejemplos Clasificados Incorrectamente.

REFERENCIAS.

https://gamma.app/docs/Ele-II-Cap-3-Topicos-AvanzadosCNN-43ngl1jf9kinxns?mode=doc