Clasificador de Dígitos Propios

Alex Javier Ramirez Pérez

Ulises Ortega Revelo

Corporación Universitaria Autónoma del Cauca

Ingeniería Electrónica

Electiva II

Cristhian Alejandro Cañar Muñoz

4 de mayo de 2025

Introducción

El problema de la clasificación de dígitos manuscritos es una tarea fundamental en el campo de la visión por computadora. Implica la capacidad de un sistema para reconocer y categorizar automáticamente dígitos numéricos escritos a mano. Esta tarea tiene aplicaciones en diversos campos como: El reconocimiento de códigos postales, el procesamiento de cheques bancarios y la lectura de formularios.

Las Redes Neuronales Convolucionales (CNN) han demostrado ser particularmente efectivas para tareas de visión por computadora, incluyendo la clasificación de imágenes. Su arquitectura especializada les permite aprender representaciones jerárquicas de las características visuales, lo que las hace muy adecuadas para reconocer patrones complejos en los datos de píxeles.

Este informe documenta el proceso, los resultados y el análisis de un modelo CNN entrenado para clasificar dígitos manuscritos. El objetivo es proporcionar una visión general completa del desarrollo del modelo, su rendimiento y sus posibles áreas de mejora.

Descripción del Dataset

- **Nombre:** dataset
- **Origen:** Creado por Alex Javier Ramirez Pérez
- **Características:**
    - **Numero total de ejemplos:** 100 ejemplos
    - **Numero de clases:** 10 (0-9)
    - **Tamaño de las imágenes:** 3060x4080 píxeles
    - **Formato de las imágenes:** JPG

Se tomaron 10 fotografías de cada número del 0 al 9, sobre un fondo blanco, las dimensiones se mantienen en su versión original, y el preprocesamiento para los datos de **training** y **validation** se realizo en el código de Python. Cada 10 imágenes del respectivo número esta guardado en una subcarpeta con el nombre de este.

Arquitectura de la CNN

La red consta de las siguientes capas:

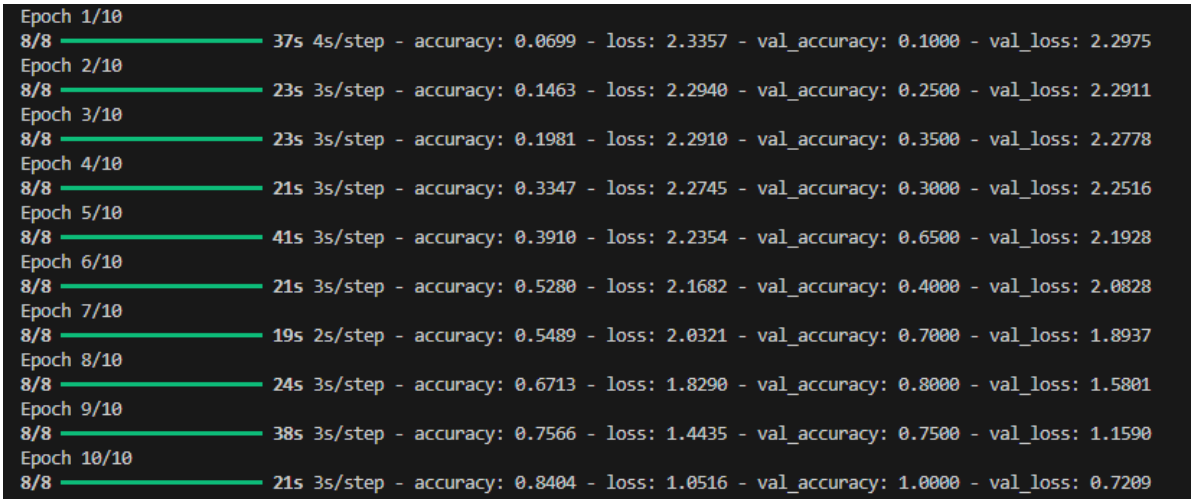
- **Capa de entrada:** La forma de entrada es (28,28,1). Esto significa que la red espera imágenes con tamaño de 28x28 pixeles con 1 canal (escala de grises)
- **Primera capa convolucional:** Esta capa tiene 28 filtros de tamaño 3x3, con una función de activación Relu**.**
- **Primera capa de max pooling:** (2,2). Esta capa reduce el tamaño de la salida de la anterior capa en un factor de 2x2.
- **Segunda capa convolucional:** Esta capa tiene 64 filtros de tamaño 3x3, con una función de activación Relu.
- **Segunda capa de max pooling:** (2,2). Esta capa reduce el tamaño de la salida de la anterior capa en un factor de 2x2.
- **Capa flatten:** Esta capa aplana la salida de las capas de max pooling en un vector 1D.
- **Primera capa densa:** Esta capa tiene 128 unidades y utiliza la función de activación Relu.
- **Capa de salida (densa):** Esta capa tiene 10 unidades (una para cada digito) y utiliza la función de activación softmax para producir distribución de probabilidad sobre las 10 clases.

Resultados y análisis:

- **Proceso de imagen y entrenamiento:**
    - Se redimensionaron las imágenes a un tamaño de 28x28 pixeles y se transformó a escala de grises.
    - Del conjunto de datos se tomaron 20% para validación y 80% para entrenamiento
    - Se normalizaron los valores de los pixeles (0-1)

- Se utilizaron 10 épocas para entrenamiento con un tamaño de lote de 10.

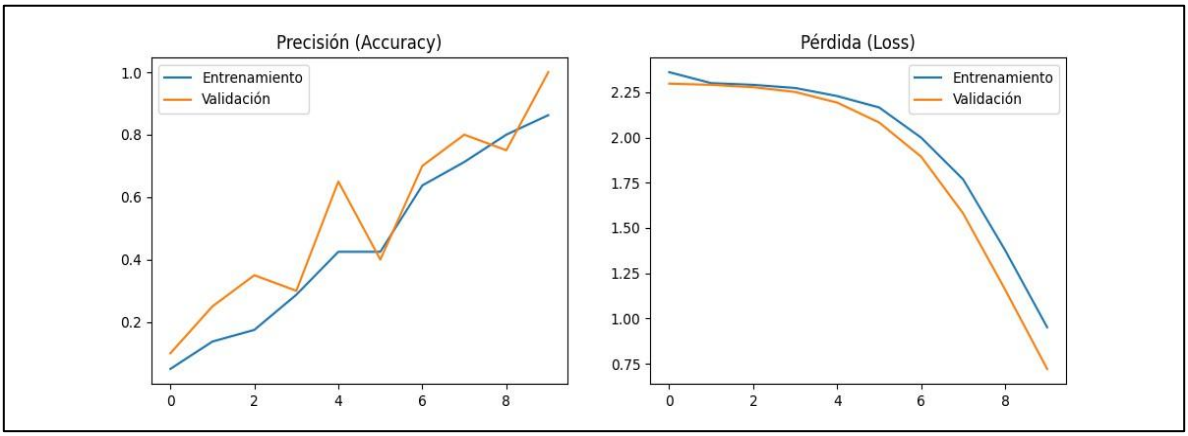
- **Resultado de entrenamiento**



El entrenamiento se completó en las 10 épocas. En cada una se registraron accuracy, loss, val_accuracy y val_loss. Accuracy y loss indican el rendimiento promedio del modelo por lote en cada época: accuracy es el porcentaje de clasificaciones correctas y loss es el error. Se observa que accuracy aumenta y loss disminuye, lo que sugiere un aprendizaje adecuado de los manuscritos. El accuracy final de 0.8404 indica un 84% de acierto en el entrenamiento.

Los valores de val_accuracy y val_loss indican la capacidad del modelo para generalizar a datos no vistos, mostrando un rendimiento similar al del conjunto de entrenamiento. El val_accuracy alcanzó un valor de 1.000, lo que sugiere un 100% de acierto en el conjunto de validación. Si bien este resultado es positivo, un acierto perfecto en la validación es inusual. Es importante considerar que podría también indicar un sobreajuste al conjunto de validación.

- **Grafica de precisión y perdida**

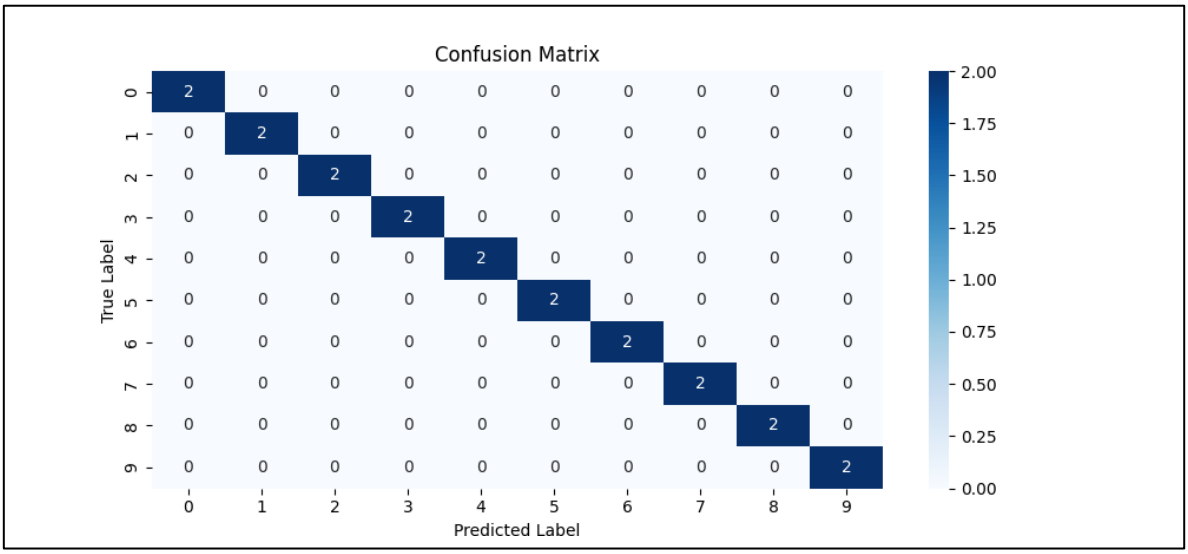


En estas gráficas se analiza la precisión y la pérdida durante el entrenamiento, en relación con los valores de val_accuracy y val_loss.

En la gráfica de precisión, se observan picos en la curva de validación. Estos altibajos sugieren que la red podría estar aprendiendo características muy específicas de las imágenes de entrenamiento, que a veces se ajustan bien a la validación y otras no, lo que indica una posible inestabilidad en la generalización.

En la gráfica de pérdida, se aprecia un descenso gradual y uniforme tanto en el entrenamiento como en la validación, lo que indica un aprendizaje adecuado. Aunque al inicio hay una ligera estabilización, la pérdida disminuye posteriormente de forma constante.

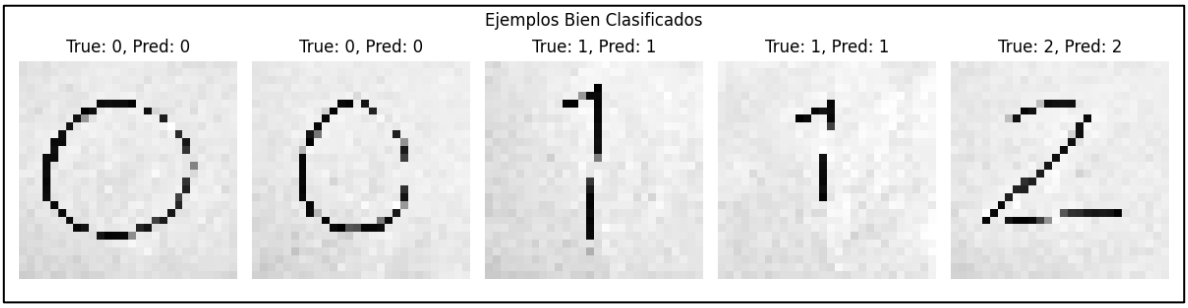
- **Matriz de confusión**



Una matriz de confusión es una herramienta que se utiliza en el aprendizaje automático para evaluar el rendimiento de un modelo de clasificación. Permite visualizar qué tan bien el modelo está clasificando las instancias en diferentes categorías, comparando las clases reales de las instancias con las clases predichas por el modelo.

En esta matriz, la diagonal con valores de 2 indica que el modelo ha predicho correctamente 2 instancias de cada clase (equivalente al 20% de validación). Es decir, para cada categoría, el modelo acertó en sus 2 predicciones. Si hubiera un valor diferente de 0 fuera de la diagonal, eso indicaría que el modelo cometió errores al confundir una clase con otra. Sin embargo, en este caso, no hay tales errores.

- **Ejemplos bien y mal clasificados**



En esta imagen se visualizan las predicciones exactas del modelo, clasificando así los valores 0,1 y 2 correctamente.

No se visualiza los ejemplos mal clasificados porque el modelo no tuvo ningún error en la clasificación de ningún digito.

- **Predicción del modelo**

Se le paso la imagen de un numero 7 al modelo por medio del uso de la plataforma de streamlit, el cual predijo correctamente el valor, con una confianza de 49.16% sobre los demás valores, no fue el mejor porcentaje, pero lo acertó válidamente.

Conclusiones y mejoras

El modelo CNN demostró ser capaz de aprender a clasificar dígitos manuscritos, como se evidencia en el aumento general de la precisión, la disminución de la pérdida durante el entrenamiento, y la predicción realizada.

Existe una indicación de posible sobreajuste, ya que la precisión de validación se estabiliza y la precisión de entrenamiento continúa mejorando hacia el final del entrenamiento. Esto sugiere que el modelo podría estar comenzando a memorizar los datos de entrenamiento en lugar de generalizar a datos no vistos.

Una de las mejoras es aumentar el tamaño y la diversidad del conjunto de entrenamiento aplicando transformaciones a las imágenes, como rotaciones, traslaciones, cambios de escala, etc. Esto puede ayudar al modelo a aprender a ser más robusto a las variaciones en los datos, ya que los del dataset que use eran muy pocos para que el modelo pudiera aprender bastantes características.

También se podría modificar la CNN, Considerando añadir más capas convolucionales o aumentar el número de filtros en las capas existentes para permitir que el modelo aprenda características más complejas.

Código

- Importar las librerías necesarias para el desarrollo del código

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import os
from sklearn.metrics import confusion_matrix
import seaborn as sns
```

- Se generalizan los valores de los pixeles y se toma el 20% del dataset como validación

```
datagen = ImageDataGenerator(
    rescale=1./255,
    validation_split=0.2
)
```

- Se crean dos generadores, uno de entrenamiento y otro de validación, a cada uno se le dice que el tamaño de las imágenes los transforme a 28x28 pixeles, el tamaño del lote de 10

```
train_generator = datagen.flow_from_directory(
    directorio_dataset,
    target_size=(28, 28),
    batch_size=10,
    class_mode='categorical',
    color_mode='grayscale',
    subset='training'
)

validation_generator = datagen.flow_from_directory(
    directorio_dataset,
    target_size=(28, 28),
    batch_size=10,
    class_mode='categorical',
    color_mode='grayscale',
    subset='validation',
    shuffle=False
)
```

- Se crean las clases para el dataset

```
clases = train_generator.class_indices
print("Etiquetas de clase:", clases)
class_labels = list(clases.keys())
```

- Se crea la CNN y se entrena utilizando 10 épocas y los valores generados de entrenamiento y validación

```
print("Creando el modelo CNN...")
model = Sequential([
    Conv2D(28, (3,3), activation='relu', input_shape=(28,28,1)),
    MaxPooling2D((2,2)),
    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D((2,2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])

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

print("Entrenando el modelo...")
history = model.fit(
    train_generator,
    epochs=10,
    validation_data=validation_generator
)
```

- Se grafican los resultados en el entrenamiento como el accuracy, loss, val_accuracy y val_loss, en precisión y validación.

```
print("Graficando resultados...")
fig2, ax2 = plt.subplots(1, 2, figsize=(12,4))

ax2[0].plot(history.history['accuracy'], label='Entrenamiento')
ax2[0].plot(history.history['val_accuracy'], label='Validación')
ax2[0].set_title('Precisión (Accuracy)')
ax2[0].legend()

ax2[1].plot(history.history['loss'], label='Entrenamiento')
ax2[1].plot(history.history['val_loss'], label='Validación')
ax2[1].set_title('Pérdida (Loss)')
ax2[1].legend()

plt.show()
```

- Se dan los valores generales del modelo terminado, también, se crea la matriz de confusión y los valores a usar.

```
print("Evaluando el modelo en el conjunto de validación...")
loss, acc = model.evaluate(validation_generator, verbose=0)
print(f"\nPrecisión en Validación: {acc:.4f}")
print(f"Pérdida en Validación: {loss:.4f}")

print("Calculando y visualizando la Matriz de Confusión...")
y_pred_probs = model.predict(validation_generator)
y_pred = np.argmax(y_pred_probs, axis=1)
y_true = validation_generator.classes

cm = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=class_labels, yticklabels=class_labels)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```

- Se identifica los ejemplos bien y mal clasificados.

```
print("Identificando ejemplos bien y mal clasificados...")
num_ejemplos_a_mostrar = 5
bien_clasificados_indices = []
mal_clasificados_indices = []

x_val_batch, y_val_batch = next(validation_generator)
y_val_true = np.argmax(y_val_batch, axis=1)
y_val_pred = np.argmax(model.predict(x_val_batch), axis=1)

for i in range(len(y_val_true)):
    if y_val_true[i] == y_val_pred[i] and len(bien_clasificados_indices) < num_ejemplos_a_mostrar:
        bien_clasificados_indices.append(i)
    elif y_val_true[i] != y_val_pred[i] and len(mal_clasificados_indices) < num_ejemplos_a_mostrar:
        mal_clasificados_indices.append(i)
```

- Se grafican esos ejemplos, dependiendo los valores que arrojo el modelo y la identificación, y por ultimo se exporta el modelo para hacer pruebas

```
if bien_clasificados_indices:
    plt.figure(figsize=(12, 3))
    for i, index in enumerate(bien_clasificados_indices):
        plt.subplot(1, num_ejemplos_a_mostrar, i + 1)
        plt.imshow(x_val_batch[index].squeeze(), cmap='gray')
        plt.title(f"True: {y_val_true[index]}, Pred: {y_val_pred[index]}")
        plt.axis('off')
    plt.suptitle("Ejemplos Bien Clasificados")
    plt.tight_layout()
    plt.show()

if mal_clasificados_indices:
    plt.figure(figsize=(12, 3))
    for i, index in enumerate(mal_clasificados_indices):
        plt.subplot(1, num_ejemplos_a_mostrar, i + 1)
        plt.imshow(x_val_batch[index].squeeze(), cmap='gray')
        plt.title(f"True: {y_val_true[index]}, Pred: {y_val_pred[index]}")
        plt.axis('off')
    plt.suptitle("Ejemplos Mal Clasificados")
    plt.tight_layout()
    plt.show()

model.save('modelo_cnn_numeros_propios.keras')
```