UNIVERSIDAD EUROPEA DEL ATLÁNTICO

GRADO EN

Ingeniería Informática

Inteligencia Artificial

TFA-OCR

Trabajo realizado por
Sara Aisha Patiño Solis

Profesor
Jose Manuel Breñosa

SANTANDER

Índice

Introducción

Problema a Afrontar

El reconocimiento óptico de caracteres (OCR) es una tecnología crucial con la creciente necesidad de digitalizar documentos físicos para almacenamiento, procesamiento y análisis electrónico. El problema principal que se aborda en este trabajo es la conversión automática de texto contenido en imágenes, tanto impresas como manuscritas, en texto digital editable. Este proceso no solo mejora la eficiencia en la gestión de documentos, sino que también facilita el acceso a la información y la integración de datos en sistemas digitales.

El OCR presenta varios desafíos, entre ellos:

- La variabilidad en los estilos de escritura, especialmente en textos manuscritos.
- La presencia de ruido en las imágenes, como manchas o borrosidad.
- La necesidad de reconocer caracteres en diferentes idiomas y tipografías.
- La clasificación correcta de caracteres con formas similares.

Propósito del Trabajo

El propósito de este trabajo es desarrollar un sistema de OCR eficiente para poder escanear imagen con caracteres digitales (tipografía de imprenta) y convertirla a texto digital. También deberá poder escanear imagen con caracteres manuales (tipografía de escritura manual) y convertirla a texto digital.

El objetivo final es no solo crear un sistema funcional, sino también adquirir un entendimiento profundo de las técnicas subyacentes, sus aplicaciones y limitaciones, así como las posibles áreas de mejora.

Lo que se Desea Aprender

A través de este proyecto, se pretende:

- Encontrar el mejor método para detectar las características de una imagen y poder encontrar el ideal que pueda reconocer los patrones para identificar letras.
- Aprender cómo preprocesar datos de imágenes para optimizar el rendimiento de las CNNs.
- Experimentar con la configuración de modelos de aprendizaje automático, ajustando hiperparámetros y evaluando su impacto en la precisión del modelo.
- Evaluar el rendimiento del sistema OCR y entender las limitaciones de las soluciones actuales, identificando áreas para futuras mejoras.

Cómo se ha afrontado el problema

El problema planteado consiste en desarrollar un software de Reconocimiento Óptico de Caracteres (OCR) capaz de convertir texto presente en imágenes, tanto de tipografía impresa como manuscrita, a texto digital. Al enfrentarme a esta tarea, decidí utilizar Redes Neuronales Convolucionales (CNNs) debido a su eficacia comprobada en la detección de patrones y características en datos visuales.

Qué son las Redes Neuronales Convolucionales (CNNs)

Las Redes Neuronales Convolucionales (CNNs) son un tipo de arquitectura de redes neuronales diseñadas específicamente para el procesamiento de datos con una estructura en cuadrícula, como imágenes. A diferencia de las redes neuronales tradicionales, las CNNs pueden capturar relaciones espaciales en los datos, lo que las hace ideales para tareas de visión por computadora.

Estructura de las CNNs

1. **Capas Convolucionales**: Estas capas aplican filtros (kernels) a la imagen de entrada para detectar patrones locales como bordes,

texturas y otras características relevantes. Cada filtro genera un mapa de características que representa la presencia de ese patrón en distintas partes de la imagen.

2. **Capas de Pooling**: Utilizadas para reducir la dimensionalidad de los mapas de características, las capas de pooling (como max-pooling) ayudan a reducir la cantidad de parámetros y cálculos en la red, y también ayudan a prevenir el sobreajuste.
3. **Capas Completamente Conectadas**: Estas capas actúan como un clasificador en la red, combinando las características extraídas por las capas convolucionales para producir la salida final, como la clasificación de una imagen.
4. **Funciones de Activación**: Las funciones como ReLU (Rectified Linear Unit) se utilizan para introducir no linealidad en la red, permitiendo que la red aprenda relaciones más complejas entre las entradas y las salidas.

Ventajas de las CNNs

- **Extracción Automática de Características**: A diferencia de los métodos tradicionales que requieren diseñar manualmente los extractores de características, las CNNs aprenden directamente de los datos de entrenamiento.
- **Invariancia Espacial**: Pueden reconocer patrones independientemente de su posición en la imagen.
- **Escalabilidad**: Las CNNs son capaces de manejar grandes cantidades de datos y complejas estructuras de entrada.

Por Qué Otros Métodos No Serían Tan Apropiados

Existen varios métodos alternativos para el reconocimiento de caracteres, como:

1. **Métodos Basados en Plantillas**: Estos métodos comparan la imagen de entrada con un conjunto de plantillas predefinidas. Aunque son

simples, su efectividad se reduce cuando se enfrentan a variaciones en la tipografía, el tamaño del texto o la orientación.

2. **Técnicas Basadas en Histogramas de Gradientes (HOG)**: Aunque HOG es eficaz para detectar bordes y texturas, requiere una segmentación precisa de los caracteres y puede fallar en presencia de ruido o deformaciones en la imagen.
3. **Redes Neuronales Feedforward Tradicionales**: Aunque pueden ser útiles, estas redes no capturan relaciones espaciales de manera eficiente, lo que limita su capacidad para manejar datos visuales complejos.
4. **Support Vector Machines (SVM)**: SVMs son eficaces para problemas de clasificación binaria con conjuntos de datos pequeños, pero su rendimiento decrece con grandes volúmenes de datos y estructuras de entrada complejas como imágenes.

Razones para utilizar CNNs

Estudiando para el examen final de Inteligencia Artificial, me di cuenta de que las CNNs son particularmente útiles para el reconocimiento de imágenes y patrones. A diferencia de otros métodos que simplemente analizan pixeles, las CNNs tienen la capacidad de identificar características más complejas dentro de las imágenes. Esta propiedad es crucial para el OCR, donde es necesario reconocer caracteres en diferentes estilos y configuraciones.

Al investigar más sobre las CNNs, entendí que su capacidad para extraer características jerárquicas (desde bordes simples hasta formas completas) las hace ideales para tareas como la detección de texto en imágenes. Además, la posibilidad de entrenar una red para reconocer caracteres específicos me permitió personalizar el modelo para las necesidades específicas del proyecto.

En resumen, la elección de las CNNs se basó en su eficacia para extraer características relevantes de las imágenes y su aplicabilidad directa al problema de OCR, donde es fundamental ir más allá de los simples pixeles y captar las estructuras subyacentes de los caracteres.

Solución Aportada

Lo primero que hice fue importar las librerías. A continuación explicaré para que sirve cada librería.

- google.colab.userdata: Esta librería es específica de Google Colab y se utiliza para interactuar con los datos de usuario en el entorno de Colab. Facilita la gestión de archivos y datos personalizados dentro del entorno de ejecución.
- **sklearn.model_selection**: Proporciona herramientas para dividir los datos en conjuntos de entrenamiento y prueba, así como para realizar validación cruzada, crucial para evaluar el rendimiento del modelo de manera robusta.
- **sklearn.metrics**: Ofrece funciones para evaluar el rendimiento del modelo, como la matriz de confusión y el puntaje F1, que ayudan a entender la precisión y el equilibrio del modelo entre precisión y exhaustividad.
- **os**: Este módulo estándar de Python permite interactuar con el sistema operativo, como manejar directorios y archivos, lo cual es útil para la gestión de datos y configuraciones en el proyecto
- **numpy**: Una librería fundamental para el cálculo numérico en Python. Facilita operaciones con matrices y grandes arreglos multidimensionales, esenciales en el procesamiento de datos y operaciones matemáticas.
- **pandas**: Ofrece estructuras de datos flexibles y potentes para la manipulación de datos tabulares, como DataFrames, facilitando la limpieza, transformación y análisis de datos.

- **tensorflow**: Una plataforma de aprendizaje automático de extremo a extremo. Es utilizada para construir y entrenar redes neuronales, incluida la creación de modelos CNN para OCR en este proyecto.
- **matplotlib.pyplot**: Una biblioteca de trazado en 2D que permite crear gráficos estáticos, animados e interactivos en Python. Es utilizada para visualizar datos y resultados del modelo.
- **seaborn**: Construida sobre matplotlib, ofrece una interfaz más atractiva y fácil de usar para crear visualizaciones estadísticas. Es útil para explorar y entender mejor los datos.
- **random**: Proporciona funciones para generar números aleatorios, lo cual es útil en la inicialización de parámetros y en la generación de muestras aleatorias para entrenamiento o validación.
- **PIL (Pillow)**: Una biblioteca para abrir, manipular y guardar imágenes en muchos formatos. Es crucial para el procesamiento de imágenes antes de alimentar a la red neuronal.
- **kaggle**: Esta línea instala la biblioteca de Kaggle, que permite descargar y gestionar datasets directamente desde Kaggle, una plataforma popular para proyectos de ciencia de datos.
- **google.colab**: Aunque ya está integrada en el entorno de Colab, esta línea asegura que la biblioteca se actualiza o reinstala para garantizar la compatibilidad y funcionalidad completa.

```
[ ]  from google.colab import userdata
     from sklearn import model_selection
     from sklearn.metrics import confusion_matrix, f1_score
     import os
     import numpy as np
     import pandas as pd
     import tensorflow as tf
     import matplotlib.pyplot as plt
     import seaborn as sns
     import random
     from PIL import Image

     !pip install kaggle
     !pip install google.colab
```

imágen 1

Después prepare las credenciales necesarias (clave de API y nombre de usuario) para que el programa pueda interactuar con la API de Kaggle de forma segura. Después de declarar las variables de entorno descargue el dataset de EMNIST.

Flujo general:

1. **Descargar**: El archivo emnist.zip se obtiene desde Kaggle.
2. **Crear una carpeta**: Se crea la carpeta EMNIST para organizar los datos.
3. **Descomprimir**: El archivo descargado se descomprime en la carpeta EMNIST.

Resultado esperado:

Después de ejecutar estos comandos:

- Habrá una nueva carpeta llamada **EMNIST** en tu directorio actual.
- Dentro de esta carpeta estarán los archivos del conjunto de datos **EMNIST**, listos para ser utilizados en análisis o entrenamiento de modelos.

imágen 2

El código carga los archivos emnist-letters-train.csv y emnist-letters-test.csv como DataFrames usando pd.read_csv(), luego muestra las dimensiones (número de filas y columnas) de cada conjunto con data.shape y test_data.shape, permitiendo inspeccionar y preparar los datos del conjunto **EMNIST** para su análisis.

```
[ ]  data = pd.read_csv('/content/EMNIST/emnist-letters-train.csv',header=None)
     test_data = pd.read_csv('/content/EMNIST/emnist-letters-test.csv',header=None)
     print('Filas y columnas de emnist-letters-train.csv: ',data.shape)
     print('Filas y columnas de emnist-letters-test.csv: ',test_data.shape)

     Filas y columnas de emnist-letters-train.csv:  (88800, 785)
     Filas y columnas de emnist-letters-test.csv:  (14800, 785)
```

imágen 3

Divide los datos de entrenamiento en conjuntos de entrenamiento (80%) y validación (20%) usando train_test_split, separando características (columnas excepto la primera) y etiquetas (primera columna). Además, extrae las características y etiquetas del conjunto de prueba. Finalmente, imprime el tamaño de cada conjunto (entrenamiento, validación y prueba) para verificar la distribución de las muestras.

```
[ ]  x_train, x_val, y_train, y_val = model_selection.train_test_split(data.values[:,1:], data.values[:,0], test_size=0.2, random_state = 42)
     x_test = test_data.values[:,1:] #(14800, 784)
     y_test = test_data.values[:,0]  #(14800, 1)
     print('Total del set de entrenamiento: ', x_train.shape[0])
     print('Total del set de validación: ', x_val.shape[0])
     print('Total del set de test : ', x_test.shape[0])

     Total del set de entrenamiento:  71040
     Total del set de validación:  17760
     Total del set de test :  14800
```

imágen 4

El código transforma las imágenes de entrada redimensionándolas a 28x28 píxeles, expandiendo dimensiones para modelos de aprendizaje profundo, volteándolas horizontalmente y rotándolas 90° en sentido horario mediante la función resize_and_rotate. Luego, con la función read_image_rot, visualiza un conjunto de imágenes transformadas en una cuadrícula de 2x5, mostrando las etiquetas correspondientes tanto en formato alfabético como numérico, junto con un título para verificar el preprocesamiento realizado.

```
[ ]  def resize_and_rotate(x):
       image = x.reshape(x.shape[0],28,28).astype('float32')
       image = np.expand_dims(image, axis = -1)
       image = np.fliplr(image)
       image = np.rot90(image, axes=(2,1))
       return image
```

```
[ ]  x_train = resize_and_rotate(x_train)
     x_val = resize_and_rotate(x_val)
     x_test = resize_and_rotate(x_test)
```

```
[ ]  def read_image_rot(data,label,name):
       fig, axes = plt.subplots(2,5, figsize=(10,5))
       for i, ax in enumerate(axes.flat):
         ax.imshow(data[i],cmap='gray')
         ax.set_title("%c,%d" %(chr(96+label[i]),label[i]))
         ax.set_xticklabels([])
         ax.set_yticklabels([])
       plt.suptitle(name)
```

imágen 5

El código selecciona una imagen aleatoria de cada clase en el conjunto de entrenamiento, mostrando 26 imágenes, una por cada clase, en una cuadrícula de 2 filas y 13 columnas. Cada imagen es etiquetada con su clase tanto en formato alfabético como numérico. Además, visualiza las primeras 10 imágenes de los conjuntos de validación y prueba usando la función read_image_rot, que muestra las imágenes junto con sus etiquetas en una cuadrícula, facilitando la comprobación visual del procesamiento de los datos.

```
[ ]  rand = []
     for i in np.unique(y_train):
       ind = np.where(y_train == i)[0]
       rand.append(random.choice(ind))

     fig, axes = plt.subplots(2,13, figsize = (30,5))
     for i, ax in enumerate(axes.flat):
       ax.imshow(x_train[rand[i]], cmap='gray')
       ax.set_title("%c,%d" %(chr(96+y_train[rand[i]]),y_train[rand[i]]))
       ax.set_xticklabels([])
       ax.set_yticklabels([])

     # Visualize data of Validation set and Test set
     read_image_rot(x_val[:10], y_val[:10],'Validation sets')
     read_image_rot(x_test[:10], y_test[:10],'Test sets')
```

imágen 6

El código convierte las etiquetas de los conjuntos de entrenamiento, validación y prueba en vectores de una sola categoría utilizando to_categorical de Keras, lo que las transforma en representaciones "one-hot" basadas en el número total de clases. Además, normaliza los valores de las imágenes dividiéndolos entre 255 para escalar los píxeles a un rango de 0 a 1, lo que mejora la eficiencia del entrenamiento en redes neuronales.

```
number_of_classes = len(np.unique(y_train))+1
y_train = tf.keras.utils.to_categorical(y_train, number_of_classes)
y_val = tf.keras.utils.to_categorical(y_val, number_of_classes)
y_test = tf.keras.utils.to_categorical(y_test, number_of_classes)
```

Definir cuales son tests y cuales entrenamiento

```
[ ]  x_train = x_train / 255.0
     x_val = x_val / 255.0
     x_test = x_test/ 255.0
```

imágen 7

El código define una red neuronal convolucional (CNN) en Keras para clasificar imágenes de 28x28 píxeles en escala de grises, con tres capas convolucionales que usan activación ReLU, seguidas de un agrupamiento máximo para reducir la dimensionalidad. Después, la salida se aplana y pasa a través de dos capas densas con activación ReLU, y finalmente, una capa de salida con activación softmax para clasificación multiclase. El modelo está

diseñado para clasificar las imágenes del conjunto EMNIST en un número de clases determinado por number_of_classes. En ésta parte del código es cuando definimos el modelo.

```
[ ]  model = tf.keras.models.Sequential([
         tf.keras.layers.Input(shape=(28,28,1)),
         tf.keras.layers.Conv2D(32, kernel_size = 3, activation = 'relu'),
         tf.keras.layers.Conv2D(64, kernel_size = 3, activation = 'relu'),
         tf.keras.layers.Conv2D(128, kernel_size = 3, activation = 'relu'),
         tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
         tf.keras.layers.Flatten(),
         tf.keras.layers.Dense(512, activation= 'relu'),
         tf.keras.layers.Dense(128, activation= 'relu'),
         tf.keras.layers.Dense(number_of_classes, activation= 'softmax'),
     ])
     print(model.summary())
```

imágen 8

En esta parte del código entrenamos el modelo. El código compila el modelo de la red neuronal utilizando la función de pérdida categorical_crossentropy para clasificación multiclase, el optimizador Adam con una tasa de aprendizaje de 0.001, y la métrica de precisión para evaluar el rendimiento del modelo. Luego, entrena el modelo con los datos de entrenamiento (x_train, y_train), validándolo en cada época con el conjunto de validación (x_val, y_val) durante 50 épocas, mostrando información detallada de cada época mediante verbose=1.

```
[ ]  model.compile(loss = 'categorical_crossentropy',
                   optimizer= tf.keras.optimizers.Adam(learning_rate = 0.001),
                   metrics = ['accuracy'])
     history = model.fit(x_train, y_train, validation_data=(x_val, y_val), epochs=50, verbose=1)
```

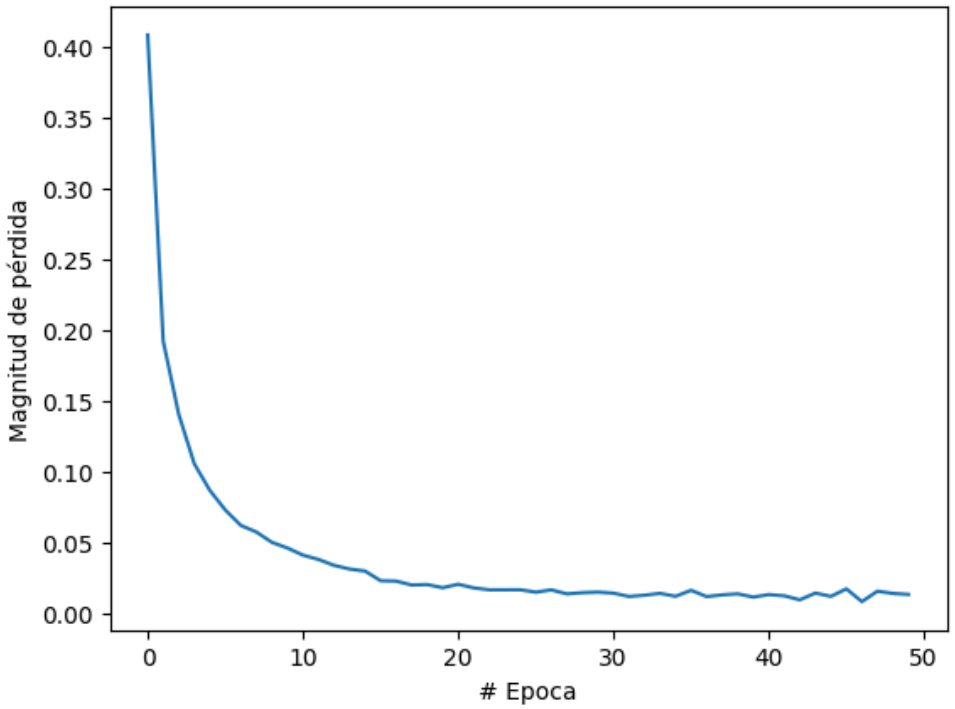
imágen 9

El código genera un gráfico que muestra la evolución de la pérdida durante el entrenamiento del modelo a lo largo de las épocas. Utiliza plt.xlabel y plt.ylabel para etiquetar los ejes del gráfico, donde el eje x representa el número de la época y el eje y muestra la magnitud de pérdida. Luego, plt.plot(history.history["loss"]) dibuja la curva de pérdida utilizando los valores

almacenados en el historial del entrenamiento (history.history["loss"]), permitiendo visualizar cómo la pérdida cambia con cada época.

```
plt.xlabel("# Epoca")
plt.ylabel("Magnitud de pérdida")
plt.plot(history.history["loss"])
```

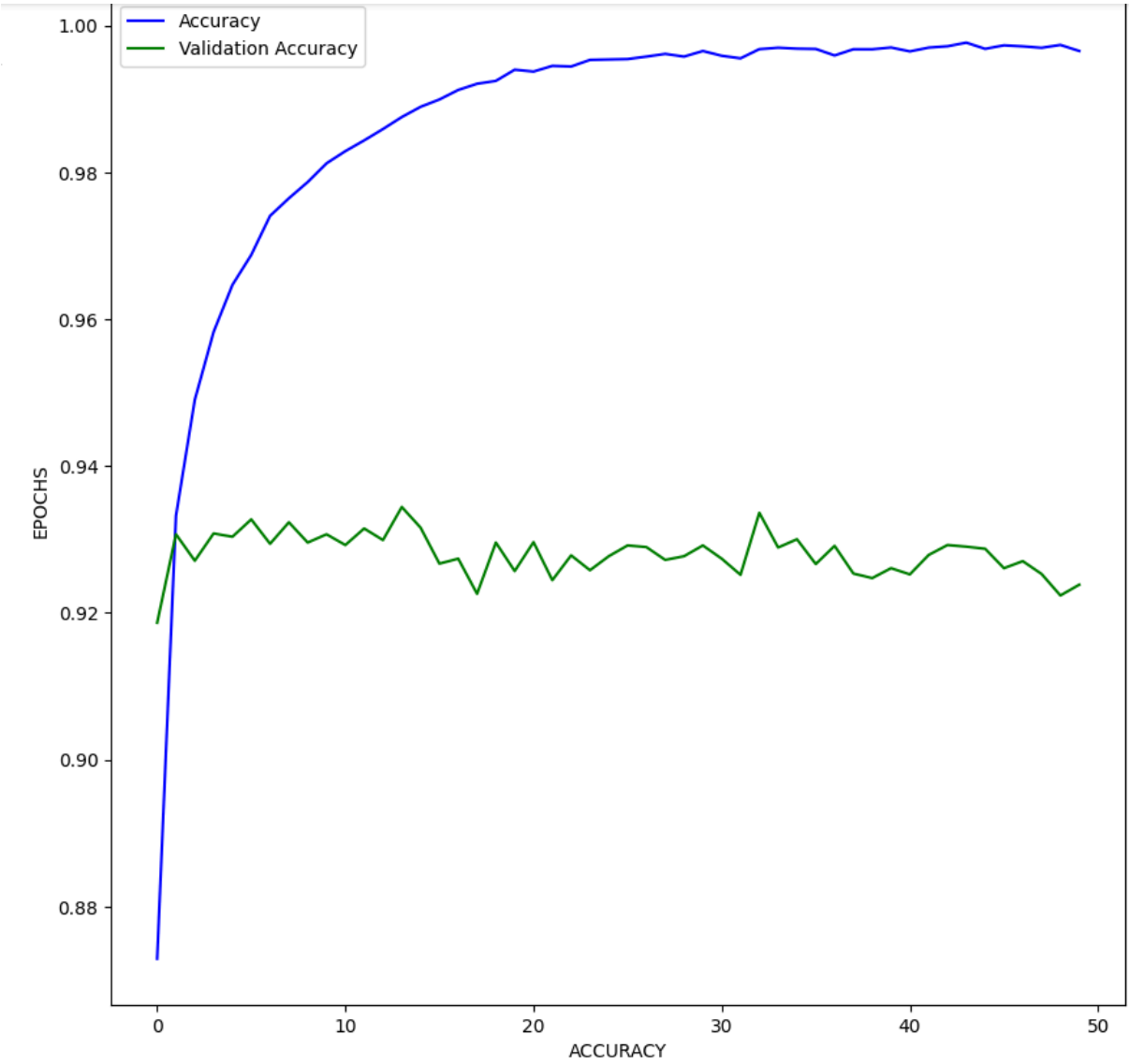
```
[<matplotlib.lines.Line2D at 0x781c146b8250>]
```



imágen 10

El código crea un gráfico de líneas para visualizar la evolución de la precisión del modelo durante el entrenamiento y su precisión en el conjunto de validación a lo largo de las épocas. Utiliza Seaborn para trazar dos curvas: una para la precisión en el conjunto de entrenamiento (accuracy) y otra para la precisión en el conjunto de validación (val_accuracy), con colores personalizados (azul y verde). El eje x representa las épocas, mientras que el eje y muestra los valores de precisión. Finalmente, se etiquetan los ejes y se incluye una leyenda para identificar ambas curvas, mostrando el rendimiento del modelo en ambas fases a lo largo del tiempo.

```
[15]
    # Suponiendo que ya tienes acc y val_acc
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    epoch = range(len(acc))

    plt.figure(figsize=(10, 10))
    sns.lineplot(x=epoch, y=acc, label='Accuracy', color='blue')  # Cambia 'blue' por el color deseado
    sns.lineplot(x=epoch, y=val_acc, label='Validation Accuracy', color='green')  # Cambia 'orange' por el color deseado
    plt.ylabel('EPOCHS')
    plt.xlabel('ACCURACY')
    plt.legend()
    plt.show()
```



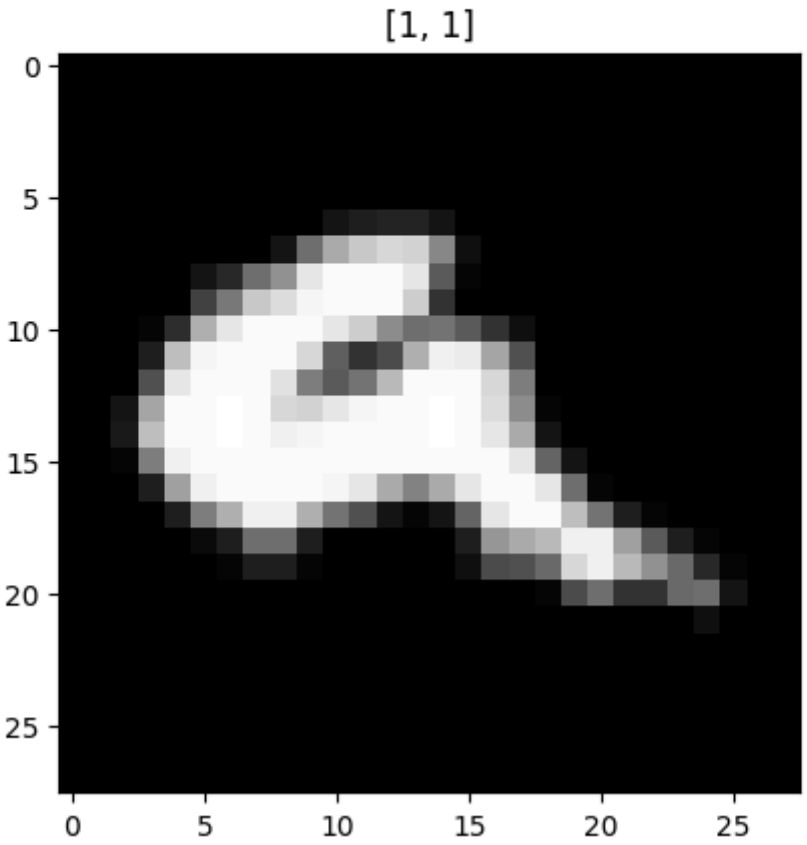
imágen 11

El código realiza una predicción sobre el primer elemento del conjunto de prueba (x_test) utilizando el modelo entrenado. Luego, visualiza la imagen correspondiente de x_test en escala de grises con plt.imshow. El título del gráfico muestra dos valores: el valor verdadero de la etiqueta de la imagen

(obtenido con np.argmax(y_test[0]), que indica la clase real) y el valor predicho por el modelo (obtenido con np.argmax(predict[0]), que indica la clase predicha). Finalmente, plt.show() muestra la imagen con su título. Este proceso permite verificar visualmente cómo el modelo predice una imagen específica del conjunto de prueba.

```
[16] #Test model with the first element of test set
     predict = model.predict(x_test)
     plt.imshow(x_test[0], cmap='gray')
     plt.title([np.argmax(y_test[0]), np.argmax(predict[0])])
     plt.show()
```

```
463/463 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step
```



imágen 12

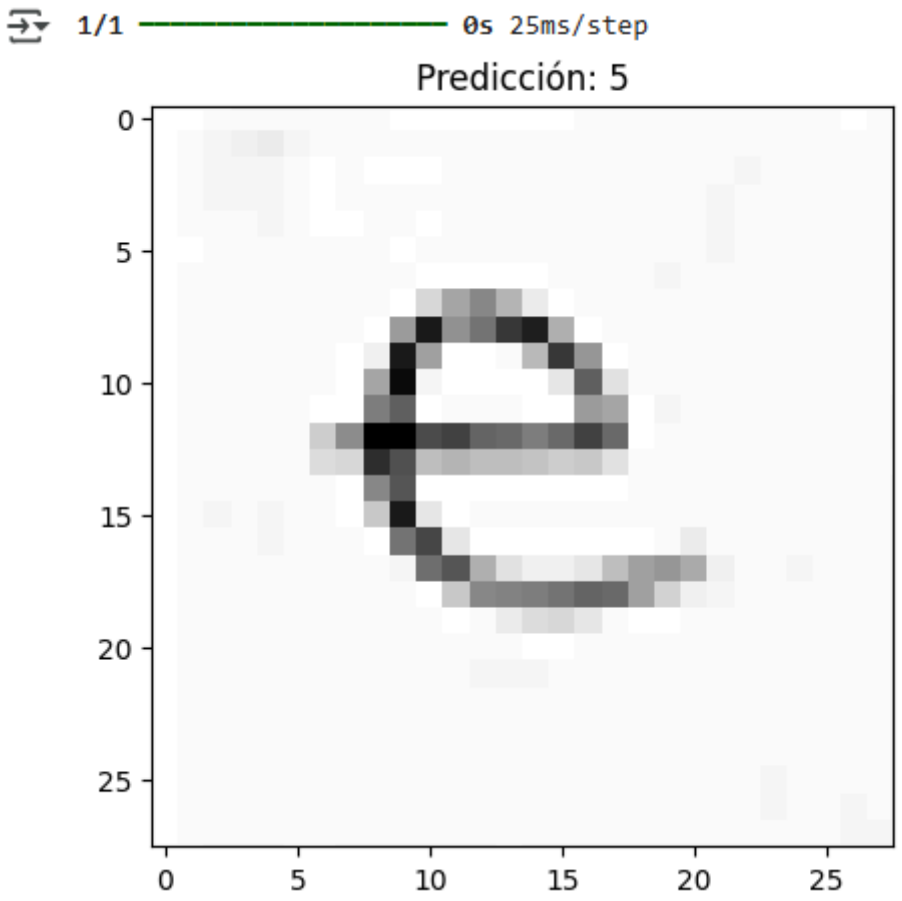
Podemos ver que el modelo predijo correctamente que es una a.

En la siguiente parte del código nos da la posibilidad de subir una imagen propia para que el modelo pueda realizar una predicción. Primero, la imagen se abre y convierte a escala de grises (convert('L')), luego se redimensiona a

28x28 píxeles para que coincida con el tamaño de entrada esperado por el modelo. A continuación, el modelo realiza la predicción utilizando model.predict(), y se obtiene la clase predicha. Finalmente, se muestra la imagen original junto con la clase predicha en el título del gráfico.

```python
uploaded_image_file = 'e.png'

# Cargar y predecir la imagen
image = Image.open(uploaded_image_file).convert('L')  # Convertir a escala de grises
image = image.resize((28, 28))  # Redimensionar a 28x28
image_array = np.array(image) / 255.0
image_array = image_array.reshape(1, 28, 28, 1)

prediction = model.predict(image_array)
predicted_class = np.argmax(prediction)

# Mostrar la imagen y la predicción
plt.imshow(image, cmap='gray')
plt.title(f'Predicción: {predicted_class}')
plt.show()
```

imágen 13



imágen 14

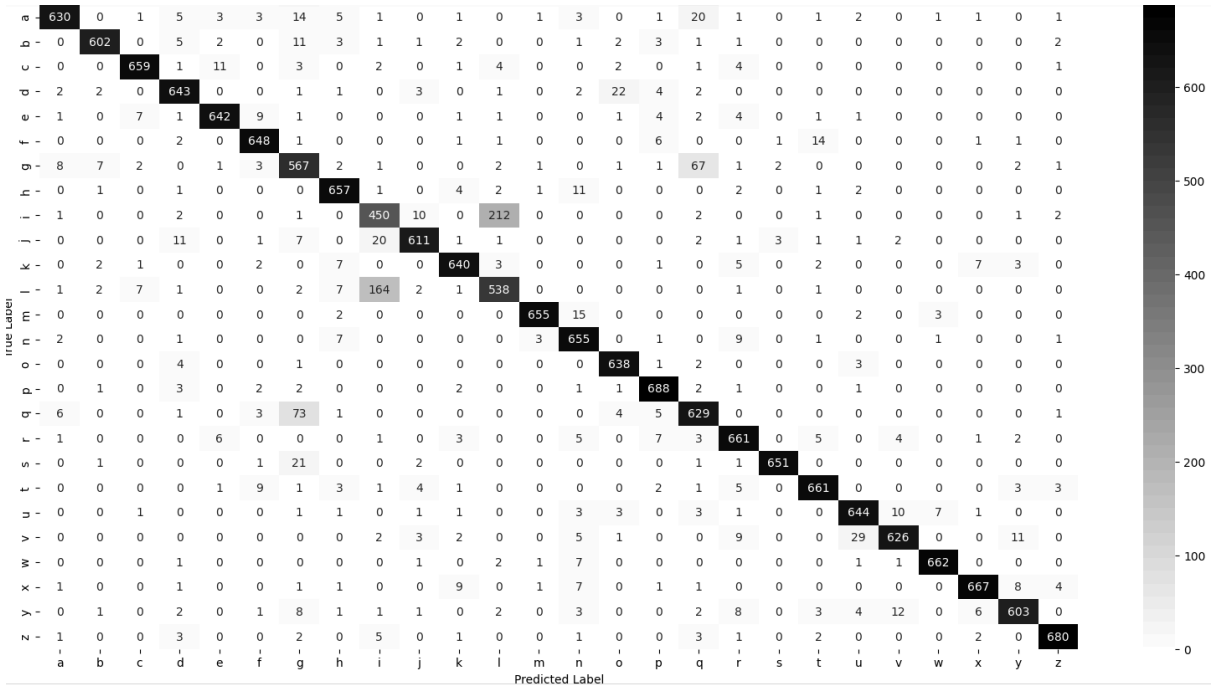
En este caso específico, el modelo hizo la predicción de manera correcta. Sin embargo he notado que el modelo tiene más errores al momento de utilizar las imágenes del dataset aunque sean preprocesadas por el código.

La última parte del código es la generación de la matriz de confusión. La matriz permite analizar que tanto se confundía el modelo en las predicciones, sobre todo, con qué letras se confundía más que otras. Esto permite ver que letras necesitan más entrenamiento.

Primero, convierte las etiquetas verdaderas y las predicciones del modelo en índices de clase. Luego, calcula la puntuación F1 para evaluar el equilibrio entre precisión y exhaustividad. Posteriormente, genera una matriz de confusión para comparar las predicciones con las etiquetas reales, y por último muestra la matriz generada.

```
[19] #Display confusion matrix to see the number of true predictions
     Y = np.argmax(y_val, axis = 1)
     P = np.argmax(model.predict(x_val, verbose = 0), axis = 1)
     f1 = f1_score(Y,P, average ='micro')
     print("F1 score: %0.2f%%" %f1)
     conf_matrix = confusion_matrix(Y, P)
     letter = []
     for i in range(1,27):
       letter.append(chr(96+i))
     plt.figure(figsize=(20, 10))
     sns.heatmap(conf_matrix, annot=True, cmap='binary', fmt= '.0f',xticklabels=letter, yticklabels=letter)
     plt.title('Confusion Matrix')
     plt.xlabel('Predicted Label')
     plt.ylabel('True Label')
     plt.show()
```

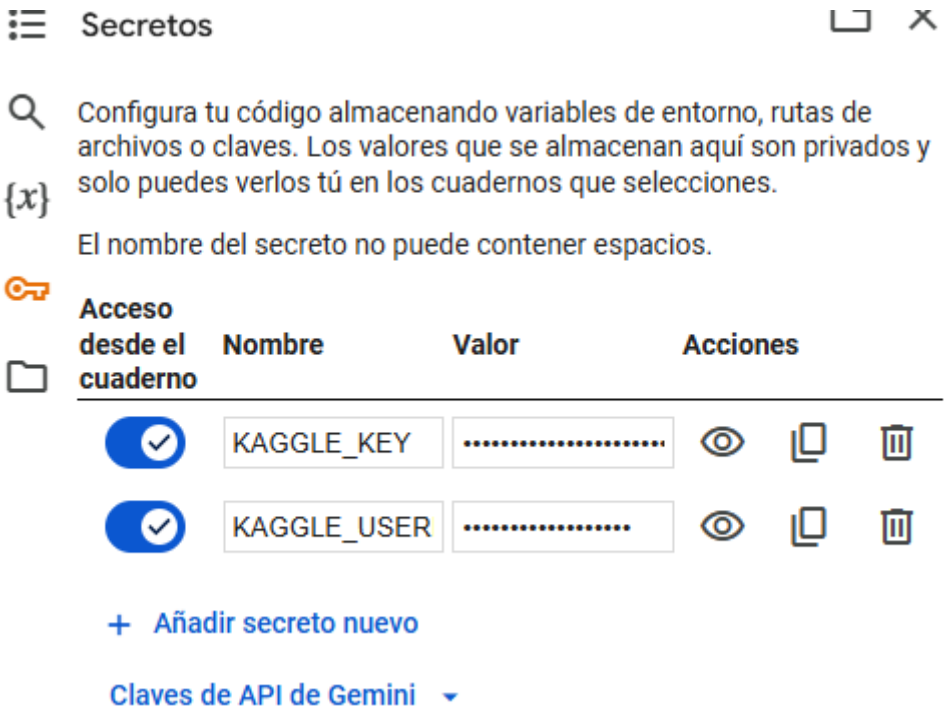
imágen 15



imágen 16

Cómo ejecutar el proyecto

Utilizando el entorno de Google Colab necesitarás agregar las variables de entorno en la parte de secretos (imágen 17) que tiene el icono de una llave.



imágen 17

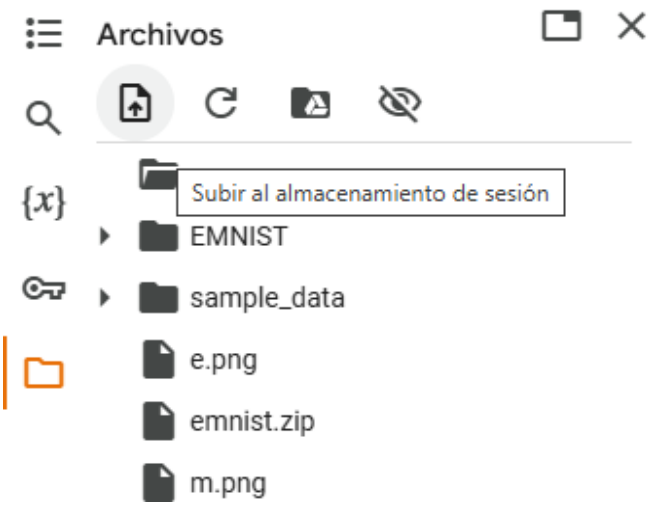
- **KAGGLE_KEY**: 84e8ed6d91d9c6bd7e56c77ac6232bb7
- **KAGGLE_USER:** saraaishapatiosols

Estas variables de entorno también se podrán encontrar en el archivo llamado Kaggle que se encuentra ajuntado a este proyecto. También se puede obtener valores propios registrando en la página de Kaggle: https://www.kaggle.com/.

Una vez que estén las variables de entorno, deberás subir por lo menos una imágen con la letra que desees predecir en los archivos de Google Colab donde dice 'Subir al almacenamiento de sesión' (imágen 19) y con el nombre de la imagen que importaste, deberás ajustar el código de la siguiente imagen para que coincida con el archivo que subiste (imágen 18).

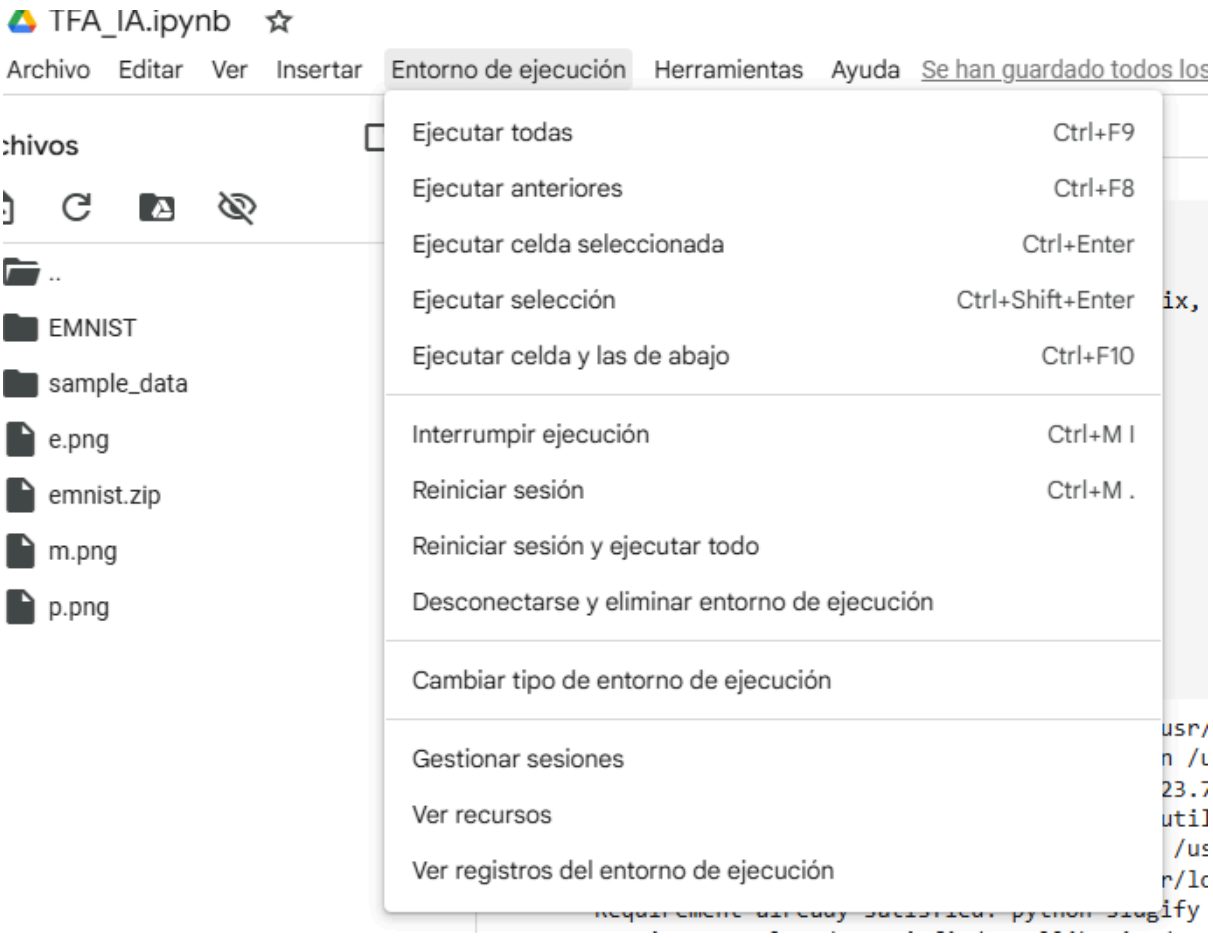
```
uploaded_image_file = 'e.png'

# Cargar y predecir la imagen
image = Image.open(uploaded_image_file).convert('L')  # Convertir a escala de grises
image = image.resize((28, 28))  # Redimensionar a 28x28
image_array = np.array(image) / 255.0
image_array = image_array.reshape(1, 28, 28, 1)

prediction = model.predict(image_array)
predicted_class = np.argmax(prediction)

# Mostrar la imagen y la predicción
plt.imshow(image, cmap='gray')
plt.title(f'Predicción: {predicted_class}')
plt.show()
```

imágen 18



imágen 19

Una vez completado ese paso, podrás ejecutar el proyecto en la parte que dice Entorno de ejecución y dar clic en ‘Ejecutar todas’ o simplemente presionar Ctrl+F9. La ejecución puede tardar varios minutos, sobre todo en la parte de entrenamiento del modelo. Una vez que esté ejecutado, se podrá cambiar la imagen que se desea predecir y sólo se tendrá que ejecutar la parte del código donde está la importación (imágen 18).



imágen 20

Conclusión

El desarrollo de este sistema de Reconocimiento Óptico de Caracteres (OCR) ha sido una experiencia muy valiosa, combinando teoría y práctica para resolver un problema real. A lo largo del proyecto, enfrenté varios retos, como reconocer diferentes estilos de escritura, manejar imágenes borrosas o con ruido, y tener que hacer mucha investigación por mi cuenta para realizar la parte práctica.

Las Redes Neuronales Convolucionales (CNNs) fueron una herramienta clave, ya que pueden identificar patrones complejos en las imágenes. Utilicé técnicas como el ajuste de datos antes de procesarlos, la normalización de imágenes y la conversión de etiquetas, lo que ayudó a mejorar la precisión del modelo.

Aunque obtuve buenos resultados, también identifique algunas áreas para mejorar. Por ejemplo, usar más datos y ajustar algunos parámetros del modelo podría aumentar su precisión. Además, incluir más variaciones en los datos de entrenamiento haría que el modelo sea más robusto y capaz de manejar diferentes tipos de imágenes.

Este proyecto no solo me permitió construir un sistema OCR funcional, sino que también me brindó la oportunidad de aprender y aplicar técnicas de inteligencia artificial en un contexto práctico. Las lecciones aprendidas me servirán para futuras mejoras y proyectos en este campo.