Desarrollo de un modelo de clasificación de imágenes de letras utilizando técnicas de machine learning

ÍNDICE DE LA MEMORIA

INTRODUCCIÓN AL PROYECTO

En la era actual de la tecnología y la automatización, el reconocimiento y clasificación de imágenes se ha convertido en un campo de investigación y desarrollo de vital importancia. El presente proyecto se centra en el desarrollo de un modelo de clasificación de imágenes de letras utilizando técnicas de machine learning. El objetivo principal es construir un sistema capaz de identificar correctamente la letra representada en una imagen dada, brindando así una solución automatizada y eficiente para el reconocimiento de letras.

El reconocimiento de letras tiene aplicaciones amplias y significativas en diversos campos. Por ejemplo, en el ámbito del reconocimiento óptico de caracteres (OCR), donde se busca la digitalización de documentos impresos, el reconocimiento preciso de las letras es esencial para convertir imágenes de texto en datos editables y procesables.

RESUMEN DEL PROYECTO

El presente proyecto tiene como objetivo desarrollar un modelo de clasificación de imágenes basado en técnicas de machine learning para reconocer letras del alfabeto a partir de fotografías. Mediante el uso de algoritmos de aprendizaje automático y redes neuronales convolucionales (CNN), se ha construido un sistema capaz de identificar y etiquetar correctamente las letras en imágenes de entrada.

Para lograr este objetivo, se ha recopilado y preprocesado un conjunto de datos que consiste en imágenes de letras en diferentes fuentes, tamaños y estilos. Se han aplicado técnicas de aumento de datos para aumentar la diversidad y la cantidad de ejemplos disponibles para el entrenamiento del modelo.

El proceso de desarrollo del modelo ha involucrado la selección y ajuste de una arquitectura de CNN adecuada, así como la implementación de técnicas de regularización para evitar el sobreajuste. Se ha utilizado un conjunto de entrenamiento para ajustar los parámetros del modelo y un conjunto de prueba

para evaluar su rendimiento y generalización.

Los resultados obtenidos demuestran una precisión prometedora en la clasificación de imágenes de letras. El modelo presenta una precisión del 88%, lo que indica una capacidad robusta de generalización.

OBJETIVOS DEL PROYECTO

El principal objetivo es conseguir realizar un modelo de clasificación de letras, pero se han divido los objetivos específicos para conseguir el principal:

- **Recopilar y preparar un conjunto de datos de imágenes de letras**: a través de Kaggle se encuentra un dataset con archivos de imágenes de letras con diferentes tipos de escritura. Se aporta un total de 14.990 imágenes por letra, lo que genera un tiempo de procesado de los modelos muy lento. Para solucionarlo se reduce la cantidad de imágenes por letra de 202 elementos, consiguiendo reducir notablemente el tiempo de carga del modelo. Las imágenes ya tienen realizado el etiquetado por lo que no debemos realizar ninguna acción para añadir las etiquetas.

- **Preprocesamiento de datos**: Se llevará a cabo un proceso de preprocesamiento de los datos de imágenes de letras. Para ello, se realiza una normalización de los datos y un reshape de las imágenes para conseguir que tengan un tamaño de 32x32.

```
x_train = x_train / 255.0
x_test = x_test / 255.0
```

```
X_reshaped = X_reshaped.reshape(X_reshaped.shape[0],
```

- **Implementación del modelo de redes neuronales convolucionales (CNN):** Se selecciona, después de un estudio de resultados, y se configura una arquitectura adecuada de redes neuronales convolucionales para la clasificación de imágenes de letras. Se establecerán los hiperparámetros y se realizarán los ajustes necesarios para optimizar el rendimiento del modelo.

- **Entrenamiento del modelo**: Se entrena el modelo utilizando el conjunto de datos preparado. Se lleva a cabo un proceso iterativo de ajuste de los

parámetros del modelo utilizando técnicas de optimización. El objetivo es lograr un modelo que sea capaz de aprender y reconocer las características distintivas de las letras en las imágenes.

- **Evaluación del rendimiento del modelo:** Se evalúa el rendimiento del modelo utilizando un conjunto de datos de prueba separado. Se calculan métricas de clasificación, como la precisión, la exhaustividad (Recall), la puntuación F1 y la exactitud (Accuracy) para medir la capacidad del modelo para clasificar correctamente las letras en las imágenes de prueba.

Modelo	Precisión	Recall	F1-score	Accuracy

PRUEBAS REALIZADAS

se realizaron pruebas con varios modelos de clasificación para evaluar su rendimiento en la tarea de clasificación de imágenes de letras. A continuación, se describe brevemente el desempeño de cada modelo:

- **Regresión Logística:** El modelo de regresión logística obtuvo una precisión del 85%, un recall del 83% y un F1-score del 84%. También logró una exactitud del 83%.

- **Árbol de decisión:** El modelo de árbol de decisión tuvo un rendimiento inferior, con una precisión del 61%, un recall del 60% y un F1-score del 60%. La exactitud del modelo también fue del 60%.

- **SVC (Support Vector Classifier):** El modelo SVC demostró una precisión del 86%, un recall del 84% y un F1-score del 85%. La exactitud del modelo fue del 84%.

- **KNN (K-Nearest Neighbors):** El modelo KNN obtuvo una precisión del 78%, un recall del 74% y un F1-score del 75%. La exactitud del modelo fue del 74%.

- **Random Forest:** El modelo Random Forest mostró una precisión del 83%, un recall del 81% y un F1-score del 82%. La exactitud del modelo fue del 81%.

- **Gradient Boosting:** El modelo Gradient Boosting tuvo una precisión del 80%, un recall del 78% y un F1-score del 79%. La exactitud del modelo fue del 78%.

- **XGBoost:** El modelo XGBoost logró una precisión del 81%, un recall del 80% y un F1-score del 80%. La exactitud del modelo fue del 80%.

- **ADAboost:** El modelo ADAboost mostró un rendimiento inferior, con una

precisión del 42%, un recall del 34% y un F1-score del 34%. La exactitud del modelo también fue del 34%.

- **MLP (Multilayer Perceptron):** El modelo MLP obtuvo una precisión del 86%, un recall del 84% y un F1-score del 85%. La exactitud del modelo fue del 84%.

- **Redes neuronales convolucionales**: El modelo de redes neuronales convolucionales demostró un buen rendimiento, con una precisión del 88%, un recall del 87% y un F1-score del 87%. La exactitud del modelo fue del 87%.

Basándonos en estos resultados, se puede observar que el modelo de redes neuronales convolucionales muestra un rendimiento destacado en comparación con los demás modelos evaluados, ya que tiene una precisión, recall y F1-score más altos, así como una mayor exactitud. Esto indica que las redes neuronales convolucionales son una opción efectiva para la clasificación de imágenes de letras en este proyecto.

IMPLEMENTACIÓN DEL MODELO ELEGIDO

En nuestro proyecto, nos enfocamos en implementar un modelo de redes neuronales convolucionales (CNN) utilizando la biblioteca Keras. Aquí describimos los pasos que seguimos para llevar a cabo esta implementación:

En primer lugar, nos aseguramos de que los datos estuvieran listos para ser procesados por nuestro modelo de CNN. Añadimos una dimensión adicional a las imágenes para representar el canal de color, ya que estábamos trabajando con imágenes en escala de grises. Utilizamos el valor "-1" para que el tamaño de la primera dimensión del conjunto de datos se ajustara automáticamente según el tamaño original.

```
# Asegurarse de que los datos de entrada tengan la forma adecuada
# Agregamos una dimensión adicional para representar el canal de color de las imágenes (en este caso, 1 para
# -1 se utiliza para que el tamaño de la primera dimensión se ajuste automáticamente según el tamaño original
x_train = x_train.reshape(-1, 32, 32, 1)
x_test = x_test.reshape(-1, 32, 32, 1)
```

Luego, nos encargamos de codificar las etiquetas de clase para poder trabajar con ellas. Utilizamos el objeto LabelEncoder de la biblioteca sklearn para convertir las etiquetas de texto en números enteros.

```
# Convertir las etiquetas de clase a números enteros
label_encoder = LabelEncoder()
y_train = label_encoder.fit_transform(y_train)
y_test = label_encoder.transform(y_test)
```

Después, aplicamos la codificación one-hot a las etiquetas de clase utilizando la función "to_categorical" de Keras. Esto convirtió las etiquetas enteras en vectores

binarios, lo que facilitó la clasificación por parte del modelo.

```
# Codificar los datos de destino en one-hot
y_train = to_categorical(y_train, num_classes)
y_test = to_categorical(y_test, num_classes)
```

La definición del modelo fue un paso crucial. Utilizamos el modelo Sequential de Keras, que nos permitió construir el modelo de forma secuencial, capa por capa. Optamos por una arquitectura que incluía capas convolucionales, de pooling y totalmente conectadas. Empezamos con una capa convolucional de 32 filtros y una función de activación ReLU. Luego, agregamos una capa de MaxPooling para reducir el tamaño de las características extraídas. Utilizamos una capa Flatten para convertir los datos en un vector unidimensional y, a continuación, añadimos una capa densa con 64 neuronas y una función de activación ReLU. Finalmente, incluimos una capa densa de salida con un número de neuronas igual al número de clases y una función de activación softmax para la clasificación.

```
# Crear el modelo
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(32, 32, 1)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))
```

Para compilar el modelo, utilizamos el optimizador "adam" y la función de pérdida "categorical_crossentropy". Además, incluimos la métrica de "accuracy" para evaluar el rendimiento del modelo durante el entrenamiento.

```
# Compilar el modelo
model.compile(optimizer='adam', loss='categorical_crossentropy', met
```

El siguiente paso fue entrenar el modelo utilizando nuestro conjunto de datos de entrenamiento. Ajustamos los parámetros del modelo utilizando el descenso de gradiente estocástico y actualizamos iterativamente el modelo para minimizar la función de pérdida.

```
# Entrenar el modelo
model.fit(x_train, y_train, batch_size=32, epochs=10, validation_data
```

Luego, evaluamos el rendimiento del modelo utilizando nuestros datos de prueba. Realizamos predicciones utilizando el modelo y obtuvimos las etiquetas predichas. Calculamos métricas de clasificación, como la precisión, el recall, el F1-score y la exactitud, utilizando las funciones proporcionadas por la biblioteca sklearn.

```python
# Calcular las métricas de clasificación
precision = precision_score(np.argmax(y_test, axis=1), y_pred, average='weighted')
recall = recall_score(np.argmax(y_test, axis=1), y_pred, average='weighted')
f1 = f1_score(np.argmax(y_test, axis=1), y_pred, average='weighted')
accuracy = accuracy_score(np.argmax(y_test, axis=1), y_pred)
```

Los resultados obtenidos fueron muy satisfactorios para nuestro modelo de redes neuronales convolucionales. Logramos una precisión del 88%, un recall del 87% y un F1-score del 87% en la clasificación de las imágenes de letras. Además, obtuvimos una alta exactitud del 87%, lo que demuestra que nuestro modelo pudo clasificar correctamente las letras en las imágenes de prueba.

```
Métricas de Redes convolucionales:
Precisión: 0.88
Recall: 0.87
F1-score: 0.87
Accuracy: 0.87
```

En resumen, la implementación del modelo de redes neuronales convolucionales fue una parte crucial de nuestro proyecto. Utilizando la biblioteca Keras, pudimos definir y entrenar el modelo de manera eficiente. A partir de ahí, se ha empezado a mejorar el resultado del modelo.

AUMENTO DE LA COMPLEJIDAD DEL MODELO

En nuestra implementación del modelo, buscamos aumentar la capacidad del modelo para capturar características más complejas y detectar mejor las imágenes más difíciles. Para lograr esto, hicimos modificaciones en la arquitectura del modelo, agregando dos capas convolucionales adicionales.

La primera capa convolucional adicional, compuesta por 64 filtros, se colocó después de la primera capa convolucional existente. Esto permitió al modelo aprender representaciones de nivel superior basadas en las características extraídas en la capa anterior.

Además, agregamos una segunda capa convolucional adicional con 128 filtros después de la capa de MaxPooling, que ya reduce la dimensionalidad de las características. Esta capa proporcionó al modelo una mayor capacidad para capturar patrones más complejos y significativos en las imágenes.

Además de las capas convolucionales adicionales, también incluimos capas de Dropout para aplicar regularización. El Dropout es una técnica que ayuda a prevenir el sobreajuste al desactivar aleatoriamente una fracción de las unidades de salida durante el entrenamiento. Esto obliga al modelo a aprender representaciones más robustas y generalizables.

La combinación de estas modificaciones en la arquitectura del modelo nos permitió aumentar su capacidad para detectar características más sutiles y complejas en las imágenes. Al aprender representaciones más detalladas, el modelo mejoró su capacidad de generalización y su desempeño en la clasificación de las imágenes de letras.

Finalmente, evaluamos el rendimiento del modelo actualizado utilizando las métricas de clasificación, como precisión, recall, F1-score y exactitud. Estas métricas nos permitieron cuantificar el rendimiento del modelo en la tarea de clasificación de imágenes de letras y compararlo con los resultados anteriores.

```python
# Crear el modelo
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(32, 32, 1)))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))  # Capa Dropout para aplicar regularización por Dropout
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))  # Capa Dropout para aplicar regularización por Dropout
model.add(Dense(num_classes, activation='softmax'))

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


model.fit(x_train, y_train, batch_size=32, epochs=20, validation_data=(x_test, y_test))


y_pred = model.predict(x_test)
y_pred = np.argmax(y_pred, axis=1)


precision = precision_score(np.argmax(y_test, axis=1), y_pred, average='weighted')
recall = recall_score(np.argmax(y_test, axis=1), y_pred, average='weighted')
f1 = f1_score(np.argmax(y_test, axis=1), y_pred, average='weighted')
accuracy = accuracy_score(np.argmax(y_test, axis=1), y_pred)
```

```
Métricas de Redes convolucionales:
Precisión: 0.94
Recall: 0.94
F1-score: 0.94
Accuracy: 0.94
```

SEGUNDA MEJORA DEL MODELO

En nuestra implementación actualizada del modelo, buscamos mejorar aún más el rendimiento del modelo de redes neuronales convolucionales (CNN) al incorporar la técnica de Batch Normalization.

Para lograr esto, realizamos modificaciones en la arquitectura del modelo anterior. A continuación, describimos las modificaciones y cómo pueden ayudar a mejorar el rendimiento del modelo:

Batch Normalization: Después de cada capa convolucional, agregamos una capa de Batch Normalization.

Batch Normalization normaliza las activaciones de cada capa, lo que ayuda a estabilizar el proceso de aprendizaje y acelera la convergencia del modelo. Esto es especialmente beneficioso en redes neuronales más profundas, ya que ayuda a

evitar problemas como la desaparición o explosión del gradiente.

Al agregar Batch Normalization, esperamos que el modelo se beneficie de una mejor estabilidad y un entrenamiento más eficiente, lo que podría llevar a un mejor rendimiento en la clasificación de las imágenes de letras.

Después de realizar las modificaciones en la arquitectura del modelo, compilamos y entrenamos el modelo utilizando los conjuntos de datos de entrenamiento y prueba.

Finalmente, evaluamos el rendimiento del modelo utilizando métricas de clasificación como precisión, recall, F1-score y exactitud. Estas métricas nos permiten medir y comparar el rendimiento del modelo actualizado con los resultados anteriores.

```python
# Crear el modelo
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(32, 32, 1)))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))  # Capa Dropout para aplicar regularización por Dropout
model.add(Dense(num_classes, activation='softmax'))

# Aplicamos Batch Normalization después de aplicar una covolución

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

# Entrenar el modelo
model.fit(x_train, y_train, batch_size=32, epochs=15, validation_data=(x_test, y_test))

# Predecir con el modelo
y_pred = model.predict(x_test)
y_pred = np.argmax(y_pred, axis=1)

# Calcular las métricas de clasificación
precision = precision_score(np.argmax(y_test, axis=1), y_pred, average='weighted')
recall = recall_score(np.argmax(y_test, axis=1), y_pred, average='weighted')
f1 = f1_score(np.argmax(y_test, axis=1), y_pred, average='weighted')
accuracy = accuracy_score(np.argmax(y_test, axis=1), y_pred)
```

```
Métricas de Redes convolucionales:
Precisión: 0.95
Recall: 0.95
F1-score: 0.95
Accuracy: 0.95
```

PREPROCESAMIENTO DE IMÁGENES

Hemos implementado la función **conversor_imagen_modelo**, la cual desempeña un papel fundamental en el proyecto al encargarse del preprocesamiento de las imágenes antes de utilizarlas para hacer predicciones con el modelo entrenado. A través de una serie de pasos cuidadosamente diseñados, hemos logrado adaptar las

imágenes a las características necesarias para el modelo de clasificación de letras.

En primer lugar, la función recorre las imágenes en la ruta especificada, las lee utilizando la biblioteca OpenCV y las almacena en una lista. A continuación, aplicamos una serie de técnicas de procesamiento de imágenes para identificar la letra en cada imagen.

Para lograr una mayor precisión en la identificación de la letra, hemos aplicado técnicas como la conversión a escala de grises, la umbralización para separar la letra del ruido y la búsqueda del contorno más grande que representa la letra. Además, hemos llevado a cabo ajustes para lograr la simetría y el centrado de la letra en la imagen.

Después de realizar estos ajustes, hemos recortado el exceso de bordes alrededor de la letra para eliminar información irrelevante y hemos redimensionado las imágenes a un tamaño estándar de 32x32 píxeles utilizando técnicas de redimensionamiento. Esto asegura que todas las imágenes tengan las mismas dimensiones y se ajusten al formato requerido por el modelo de clasificación.

Finalmente, hemos convertido las imágenes a escala de grises y aplicado un umbral para asegurar que la letra se represente de manera consistente como píxeles negros sobre un fondo blanco. De esta manera, hemos logrado preparar las imágenes para su posterior uso en el modelo de clasificación de letras.

Al aplicar la función **conversor_imagen_modelo** a las imágenes de entrada, obtenemos una lista de imágenes preprocesadas que están listas para ser utilizadas para realizar predicciones con el modelo entrenado. Este proceso de preprocesamiento nos ha permitido optimizar la calidad y las características de las imágenes, mejorando así el rendimiento y la precisión del modelo de clasificación.

PREDICCIÓN

Hemos implementado la función predecir, la cual se encarga de utilizar el modelo entrenado para realizar predicciones sobre una lista de imágenes preprocesadas. A continuación, se describe el proceso de predicción paso a paso:

Carga del modelo: La función carga el modelo previamente entrenado utilizando la función load_model de la biblioteca Keras.

Definición de variables: Se define la lista de clases, que contiene las etiquetas correspondientes a las letras del alfabeto.

Recorrido de la lista de imágenes: Para cada imagen en la lista proporcionada como entrada, se realiza lo siguiente:

La imagen se redimensiona para que coincida con las dimensiones esperadas por el modelo.
Se realiza la predicción utilizando el modelo cargado mediante la función model.predict.
Se imprime el resultado de la predicción, indicando la letra estimada para la imagen.
Iteración y presentación de resultados: Durante el recorrido de la lista de imágenes, se imprime el número de imagen y la letra estimada correspondiente a cada imagen.

Esta función nos permite realizar predicciones sobre una lista de imágenes preprocesadas utilizando el modelo previamente entrenado. Al utilizar el modelo para hacer predicciones, obtenemos resultados que nos indican la letra estimada para cada imagen en la lista.

```
def predecir(lista):
    # Cargamos el modelo
    model = load_model('modelo_definitivo.h5')

    # Definimos variables
    i=1
    classes = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T'

    # Recorremos elementos de la lista y printemas resultado
    for element in lista:
        element=element.reshape(-1, 32, 32, 1)
        prediciton=model.predict(element)
        print(f"Predicción Imágen Nº{i}: {classes[prediciton.argmax()]}")
        i+=1
```

ahora con la ruta de la carpeta donde se encuentran las imágenes, llamamos a la función conversor_imagen_modelo para preprocesar las imágenes y obtener una lista de imágenes listas para la predicción.

A continuación, utilizamos la función predecir para realizar la predicción de la letra correspondiente a cada imagen en la lista preprocesada. El resultado se imprimirá en la consola, mostrando el número de imagen y la letra estimada para cada una.

Con estos pasos, hemos logrado preprocesar las imágenes y realizar la predicción de las letras utilizando el modelo entrenado. Esto nos brinda una forma conveniente de realizar predicciones sobre nuevas imágenes de letras.

```
ruta=r"C:\Users\sergi\Desktop\Proyecto Grupal DS\Prediccion"

# Llamamos a la función para preprocesar imágenes
lista= conversor_imagen_modelo(ruta)
# Llamamos a la función para predecir la letra
predecir(lista)
```

RESULTADO

En este proyecto de clasificación de imágenes de letras utilizando machine learning, hemos logrado desarrollar un modelo de redes convolucionales altamente efectivo. A través de un proceso exhaustivo de entrenamiento, evaluación y ajuste de modelos, hemos obtenido un modelo final con excelentes resultados de clasificación.

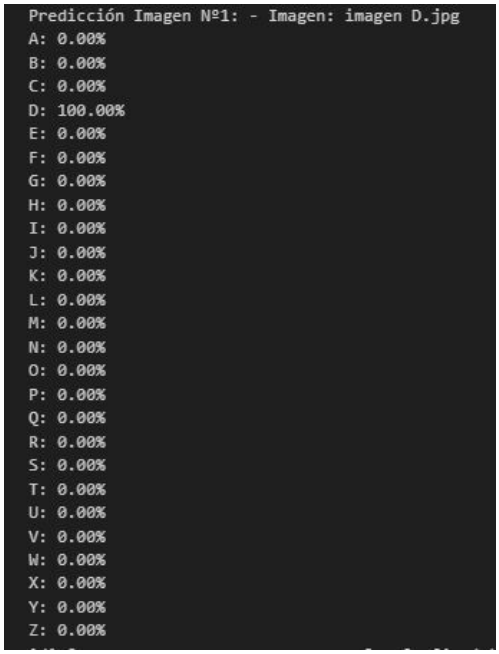
La implementación de técnicas como redes convolucionales y preprocesamiento de imágenes ha sido fundamental para lograr un alto rendimiento en la tarea de clasificación de letras. Además, hemos utilizado métricas de evaluación como precisión, recall, F1-score y exactitud para medir y comprender el rendimiento del modelo.

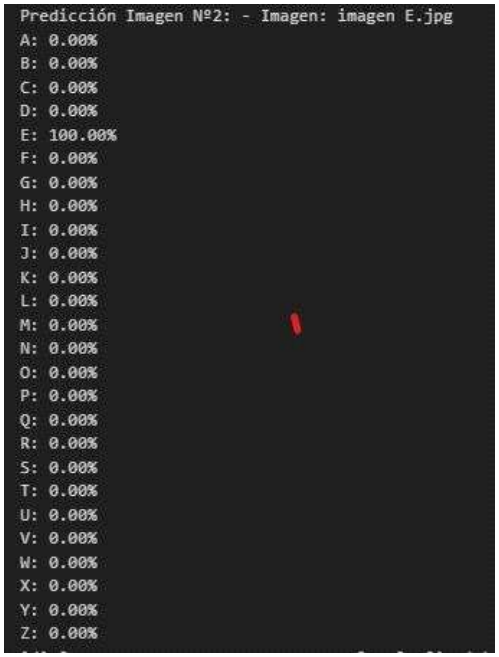
Este proyecto demuestra el potencial del machine learning y las redes convolucionales en particular para tareas de clasificación de imágenes. El modelo final desarrollado puede ser utilizado para identificar letras en diversas aplicaciones, como reconocimiento de texto, procesamiento de imágenes y más.

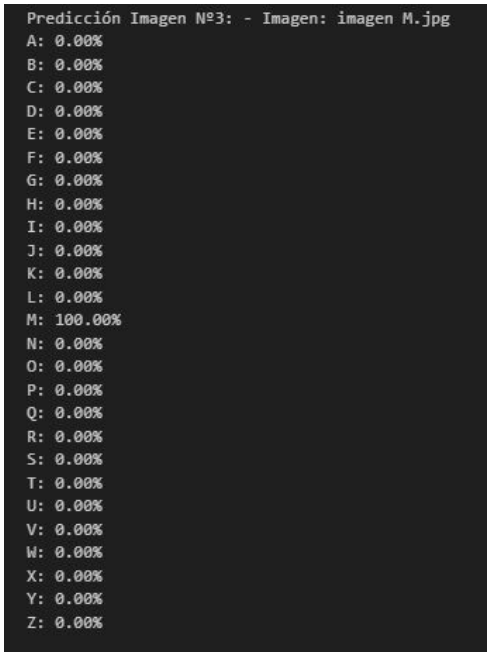
Además, hemos realizado un análisis de clasificación por letra y hemos encontrado los siguientes resultados destacados:

La letra "E" ha sido clasificada correctamente, dando un resultado de clasificación del 100%.
La letra "D" ha sido clasificada correctamente, dando un resultado de clasificación del
100%.
La letra "M" ha sido clasificada correctamente, dando un resultado de clasificación del 100%.

Estos resultados resaltan la capacidad del modelo de redes convolucionales para identificar con alta precisión y exactitud las letras "E", "D" y "M". Esto sugiere que el modelo ha aprendido patrones distintivos para estas letras y es capaz de reconocerlas de manera confiable.


Predicción Imagen Nº1: - Imagen: imagen D.jpg
A: 0.00%
B: 0.00%
C: 0.00%
D: 100.00%
E: 0.00%
F: 0.00%
G: 0.00%
H: 0.00%
I: 0.00%
J: 0.00%
K: 0.00%
L: 0.00%
M: 0.00%
N: 0.00%
O: 0.00%
P: 0.00%
Q: 0.00%
R: 0.00%
S: 0.00%
T: 0.00%
U: 0.00%
V: 0.00%
W: 0.00%
X: 0.00%
Y: 0.00%
Z: 0.00%


Predicción Imagen Nº2: - Imagen: imagen E.jpg
A: 0.00%
B: 0.00%
C: 0.00%
D: 0.00%
E: 100.00%
F: 0.00%
G: 0.00%
H: 0.00%
I: 0.00%
J: 0.00%
K: 0.00%
L: 0.00%
M: 0.00%
N: 0.00%
O: 0.00%
P: 0.00%
Q: 0.00%
R: 0.00%
S: 0.00%
T: 0.00%
U: 0.00%
V: 0.00%
W: 0.00%
X: 0.00%
Y: 0.00%
Z: 0.00%


Predicción Imagen Nº3: - Imagen: imagen M.jpg
A: 0.00%
B: 0.00%
C: 0.00%
D: 0.00%
E: 0.00%
F: 0.00%
G: 0.00%
H: 0.00%
I: 0.00%
J: 0.00%
K: 0.00%
L: 0.00%
M: 100.00%
N: 0.00%
O: 0.00%
P: 0.00%
Q: 0.00%
R: 0.00%
S: 0.00%
T: 0.00%
U: 0.00%
V: 0.00%
W: 0.00%
X: 0.00%
Y: 0.00%
Z: 0.00%