Navegación Autónoma

Trimestre Abril - Julio 2024

CLASIFICACIÓN CON CNN
DETECCIÓN DE SEÑALES DE TRÁNSITO

Autor

- Kevin Brandon Cruz Mejía - A01794176

Introducción

Este proyecto aborda el desafío de la clasificación automática de señales de tráfico utilizando una red neuronal convolucional (CNN) implementada en Keras. El objetivo es desarrollar un modelo que pueda identificar con precisión diferentes señales de tráfico, un componente esencial para los sistemas de asistencia al conductor y la navegación autónoma.

Código

Python

```
*


*

## **Actividad 4.2 - CLASIFICACIÓN CON CNN DETECCIÓN DE SEÑALES DE TRÁNSITO**



!pip install tensorflow
!pip install opencv-python-headless
!pip install pillow
!pip install requests

#Importación de Librerias
import numpy as np
import os
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout,
Flatten, Conv2D, MaxPooling2D
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder
import random
import requests
from PIL import Image

import cv2
```

```
from sklearn.model_selection import train_test_split
np.random.seed(0)
#Acceso a Google Drive from
google.colab import drive
drive.mount('/content/drive') #Extraer
archivo comprimido del Dataset
!unzip "/content/drive/MyDrive/Imagenes/archive.zip" -d
"/content/sample_data/Imagenes" #
Ruta al directorio con imágenes
path_to_images = '/content/sample_data/Imagenes/Train'

images = []
labels = []

# Recorre todos los directorios y subdirectorios en la
ruta dada for dirpath, dirnames, filenames in
os.walk(path_to_images):
    for filename in filenames:
        if filename.endswith('.png'): # Procesa solo
archivos PNG img_path = os.path.join(dirpath,
            filename)
            image = cv2.imread(img_path,
cv2.IMREAD_GRAYSCALE) if image is not None: image =
            cv2.resize(image, (32, 32))
            images.append(image)
                 label = dirpath.split('/')[-1] # Obtén
la etiqueta del nombre de la carpeta
                labels.append(label)
            else:
                print(f"Error al procesar la imagen
{img_path}")
# Convierte listas a arrays de NumPy
images = np.array(images).reshape(-1, 32, 32, 1) # Añade
una dimensión de canal
images = images.astype('float32') / 255.0 #
Normalización
# Convertir etiquetas a formato one-hot label_encoder
= LabelEncoder() integer_encoded =
```

```
label_encoder.fit_transform(labels) labels_one_hot =
to_categorical(integer_encoded, num_classes=43)
# Dividir los datos en entrenamiento y prueba
X_train, X_test, y_train, y_test =
train_test_split(images, labels_one_hot, test_size=0.2,
random_state=42) 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}')
# Visualizar la distribución de las muestras por clase
n_muestras_por_clase = np.sum(y_train, axis=0) # Suma
para obtener el número de muestras por clase
n_clases = 43 # Número total de clases
plt.figure(figsize=(10, 8))
plt.bar(range(n_clases), n_muestras_por_clase,
color='blue') plt.xlabel('Clase')
plt.ylabel('Número de Muestras')
plt.title('Distribución de Muestras por Clase en el
Dataset')
plt.xticks(range(n_clases)) # Asegura que haya un tick
para cada clase plt.show()

# Número total de clases
n_clases = 43

# Crear figura con subplots
fig, axs = plt.subplots(nrows=4, ncols=(n_classes + 3) //
4, figsize=(20, 8))

# Mostrar una imagen de cada clase
for i in range(n_classes):
    # Encuentra el primer índice de la clase actual idx =
    np.where(np.argmax(y_train, axis=1) == i)[0][0] ax =
    axs[i // ((n_classes + 3) // 4), i % ((n_classes
+ 3) // 4)]
    ax.imshow(X_train[idx].reshape(32, 32), cmap='gray')
# Asegúrate de ajustar el tamaño y cmap según tus datos
    ax.set_title(str(i)) ax.axis('off')
```

```
plt.tight_layout()
plt.show()
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dropout,
Flatten, Dense
from keras.optimizers import Adam

# Definición del modelo
model = Sequential([
    Conv2D(32, kernel_size=(3, 3), activation='relu',
padding='same', input_shape=(32, 32, 1)),
    Conv2D(32, kernel_size=(3, 3), activation='relu',
padding='same'),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),
    Conv2D(64, kernel_size=(3, 3), activation='relu',
padding='same'),
    Conv2D(64, kernel_size=(3, 3), activation='relu',
padding='same'),
    MaxPooling2D(pool_size=(2, 2)),
    Dropout(0.25),
    Flatten(),
```

```
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(43, activation='softmax')
])

model.compile(optimizer=Adam(lr=0.001),
loss='categorical_crossentropy', metrics=['accuracy'])
print(model.summary())
# Entrenar el modelo
history = model.fit(
    X_train, y_train,
    validation_split=0.1,
    epochs=10,
    batch_size=32,
    verbose=1,
    shuffle=True
)
# Evaluar el modelo
scores = model.evaluate(X_test, y_test, verbose=1)
print('Test loss:', scores[0])
print('Test accuracy:', scores[1])
# Graficar la pérdida
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['loss', 'val_loss'])
plt.title('Loss')
plt.xlabel('epoch')
plt.show()

# Graficar la precisión
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.legend(['acc', 'val_acc'])
plt.title('Accuracy')
plt.xlabel('epoch')
plt.show()
```

```
# Prueba con una imagen de la web
url =
'https://drive.google.com/uc?export=download&id=18KfKBln9
sbU-mq4OKmpJy_-P8UB-nNF2' response =
requests.get(url, stream=True) img =
Image.open(response.raw) plt.imshow(img,
cmap='gray') plt.show()

img_arr = np.asarray(img)
img_rs = cv2.resize(img_arr, (32, 32)) # Corrección
aquí: solo dos dimensiones necesarias
img_gray = cv2.cvtColor(img_rs, cv2.COLOR_RGB2GRAY) #
Cambio de BGR a RGB si es necesario img_not =
cv2.bitwise_not(img_gray) plt.imshow(img_not,
cmap='gray') plt.show()

img_not = img_not / 255 img_resh = img_not.reshape(1,
32, 32, 1) # Asegúrate de que las dimensiones coinciden
con la entrada del modelo prediction =
np.argmax(model.predict(img_resh), axis=-1)
print('predicted class:', str(prediction))

# Guardar y descargar el modelo
model.save('model.h5') from
google.colab import files
files.download('model.h5')
```

Conclusión

En este proyecto, hemos desarrollado un modelo de clasificación de señales de tráfico utilizando redes neuronales convolucionales (CNN) implementadas con Keras. El objetivo principal fue crear un modelo que pudiera identificar con precisión distintas señales de tráfico, una función esencial para los sistemas avanzados de asistencia al conductor y la navegación autónoma.

El modelo logró una precisión excepcional del 99.63% en el conjunto de pruebas, lo que demuestra la eficacia de la arquitectura de red y las técnicas de optimización utilizadas.

Aspectos Clave:

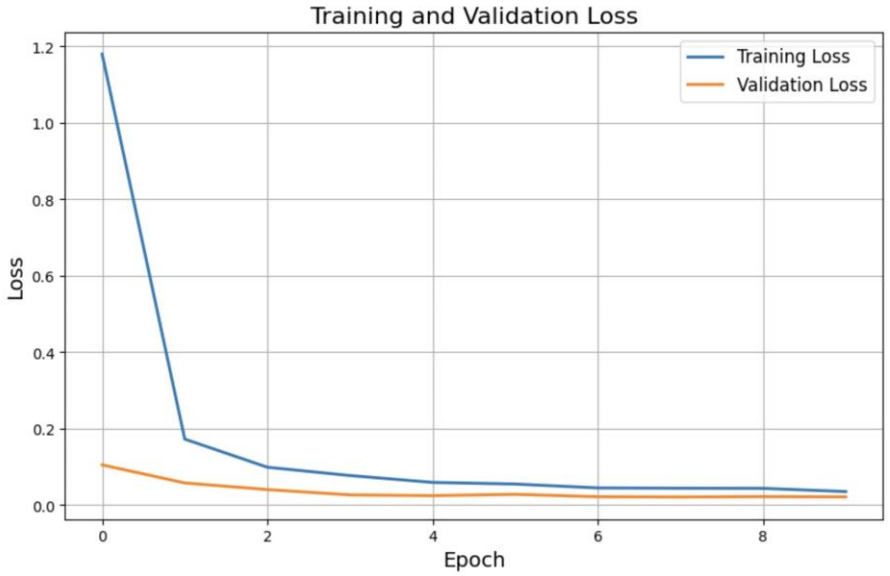
Preprocesamiento de Imágenes: Se realizó un preprocesamiento exhaustivo de las imágenes, incluyendo la normalización y la conversión a escala de grises, para reducir la variabilidad y mejorar la velocidad de entrenamiento. Las imágenes fueron redimensionadas a 32x32 píxeles para una entrada uniforme al modelo.

Modelado con Keras: Se diseñó una arquitectura de red neuronal convolucional con múltiples capas convolucionales, capas de pooling y capas de dropout para prevenir el sobreajuste, culminando en una capa densa con activación softmax para la clasificación multiclase.

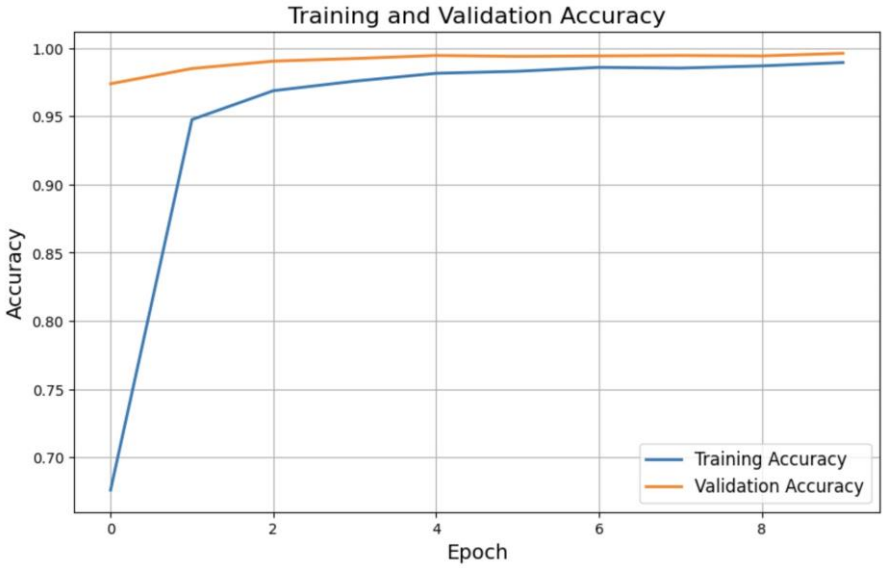
Optimización del Modelo: El ajuste de hiperparámetros, como la tasa de aprendizaje y el tamaño del lote, contribuyó a una rápida convergencia y estabilidad del modelo durante el entrenamiento. Además, se utilizó la técnica de normalización de MinMax para escalar los datos de entrada, mejorando así el rendimiento del modelo.

Visualización de Resultados

Gráfica de Pérdida: La gráfica de pérdida muestra una disminución rápida en las primeras épocas, estabilizado alrededor de la época 4, lo que indica una buena adaptación del modelo a los datos sin sobreajuste.



Gráfica de Exactitud: La gráfica de exactitud muestra una mejora rápida en la precisión durante las primeras épocas, alcanzando una precisión casi perfecta en las épocas finales.



Bibliografía

Stallkamp, J., Schlipsing, M., Salmen, J. e Igel, C. (2011). The German Traffic Sign Recognition Benchmark: A Multiclass Classification Competition. The 2011 International Joint Conference on Neural Networks (IJCNN). Disponible: https://www.researchgate.net/publication/224260296_The_German_Traffic_Sign_Recognition_Benchmark_A_multi-class_classification_competition

Ranjan, S., & Senthamilarasu, S. (2020). Applied Deep Learning and Computer Vision for Self-Driving Cars. Packt Publishing Ltd. Disponible en: https://learning.oreilly.com/library/view/applied-deep-learning/9781838646301/?sso_link=yes&sso_link_from=ITESM

MYKOLA. (Actualizado hace 6 años). GTSRB - German Traffic Sign Recognition Benchmark. Multi-class, single-image classification challenge. Disponible en: https://www.kaggle.com/datasets/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign?resource=download