Tecnológico de Monterrey

Navegación Autónoma

EJERCICIO DE
DETECCIÓN CON KERAS

Autor

- Kevin Brandon Cruz Mejía - A01794176

Código

```
## **Actividad 4.1 - Ejercicio de clasificación con Keras**


#Importacion de Librerias:

import matplotlib.image as mpimg
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from skimage.feature import hog
from sklearn.metrics import classification_report,
accuracy_score
from sklearn.model_selection import train_test_split
from skimage import color, io
import os
import glob

### Se utiliza el dataset de Kaggle que ya incluye dos
datasets propiamente separados: Uno con unicamente imagenes de
personas, y otro con imagenes de lugares y objetos:

Dataset - "Person" photos:
https://www.kaggle.com/datasets/constantinwerner/human-
detection-dataset
Dataset - "Person" photos:
https://paperswithcode.com/dataset/inria-person

Dataset - "Street" photos:
https://www.kaggle.com/datasets/mikhailma/house-rooms-streets-
image-dataset


#Se cargan las direcciones de los archivos de cada uno de los
datasets
#1-2 = Human/Peaton
```

```
human = glob.glob("/Users/fer/Downloads/human detection
dataset/2/*.jpg")

#0 = No human/No peaton

no_human = glob.glob("/Users/fer/Downloads/human detection
dataset/00/*.jpg")

#Se muestran las dimensiones de cada dataset:

print("Tamaño del dataset para peatones:", len(human))
print("Tamaño del dataset sin peatones:", len(no_human))

## Se extraen las caracteristicas de los HOG para ambos
datasets:

### *1*. Peatones

human_hog_accum = [] #Se crea la lisga para los features de
los humanos o peatones
target_size = (128, 64) #Se define un tamaño para la toma de
las imagenes


for img_path in human:
    #Se carga a color
    img_color = cv2.imread(img_path)

    # Redimensiona la imagen al tamaño objetivo
    img_color = cv2.resize(img_color, target_size)
    #Se convierte a escala de grises
    img_gray = cv2.cvtColor(img_color, cv2.COLOR_BGR2GRAY)
    human_hog_feature, human_hog_img = hog(img_gray,
orientations=11, pixels_per_cell=(16, 16),
                                           cells_per_block=(2,
2), transform_sqrt=False, visualize=True,

feature_vector=True)
```

```
    human_hog_accum.append(human_hog_feature)

#Creamos un arreglo usando Numpy:
X_human = np.vstack(human_hog_accum).astype(np.float64)
y_human = np.ones(len(X_human))

print("Tamaño de los vectores: ", X_human.shape)
print("Cantidad de 'Unos': ", y_human.shape)
print()

## 2. No peatones

no_human_hog_accum = []  #Se crea la lista para los features
de los no_humanos o no peatones
target_size = (128, 64)  #Se define un tamaño para la toma de
las imagenes

for img_path in no_human:
    #Se carga la imagen a color
    img_color = cv2.imread(img_path)
    #Se reajusta su tamaño
    img_color = cv2.resize(img_color, target_size)
    #Se convierte a escala de grises
    img_gray = cv2.cvtColor(img_color,cv2.COLOR_BGR2GRAY)

    no_human_hog_feature, human_hog_img = hog(img_gray,
orientations=11, pixels_per_cell=(16, 16),
                                          cells_per_block=(2,
2), transform_sqrt=False, visualize=True,

                                        feature_vector=True)

    no_human_hog_accum.append(no_human_hog_feature)

X_no_human = np.vstack(no_human_hog_accum).astype(np.float64)
y_no_human = np.zeros(len(X_no_human))

print("Tamaño de los vectores: ", X_no_human.shape)
```

```
print("Cantidad de 'Zeros': ", y_no_human.shape)
print()

X = np.vstack((X_human,X_no_human))
X.shape

y = np.hstack((y_human,y_no_human))
y.shape

## Se generan los datos de prueba y entrenamiento:



from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size = 0.3, random_state = 42)

# Se escala el set de entrenamiento
scalar = MinMaxScaler()
scalar.fit(X_train)
X_train_scaled = scalar.transform(X_train)

# Se escala el set de prueba
scalar = MinMaxScaler()
scalar.fit(X_test)
X_test_scaled = scalar.transform(X_test)

print("Tamaño de X de entrenamiento:", X_train.shape)
print("Tamaño de X de prueba:", X_test.shape)

# Se grafican los datos escalados
sns.scatterplot(x=X_train_scaled[ :, 0], y=X_train_scaled[:,
1], hue=y_train

### Se cargan las bibliotecas necesarias para definir la red
neuronal:
```

```
## LIBRERIAS

import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import Adam

### Una capa de varias entradas y dos capas ocultas, y una
neurona de salida:

#Modelo1
model = Sequential()
model.add(Dense(128, activation='relu', input_dim=924))
model.add(Dense(64, activation='relu'))
model.add(Dense(1, activation='sigmoid'))  # Capa de salida
para clasificación binaria

#Compilación del modelo:
model.compile(optimizer=Adam(learning_rate=0.01),
loss='binary_crossentropy',  metrics=['accuracy'])

hist = model.fit(x=x_train_scaled, y=y_train, verbose=2,
batch_size=50,epochs=100, shuffle= 'true')
hist = model.fit(x=x_train, y=y_train, epochs=100,
batch_size=60, validation_data=(X_test, y_test), verbose=2)

# resumen del modelo
model.summary()

# Exactitud del modelo
plt.plot(hist.history['accuracy'])
plt.legend(['Exactitud'])
plt.ylabel('Exactitud')
plt.xlabel('Epoca')

# Función de pérdida
plt.plot(hist.history['loss'])
```

```
plt.legend(['Error'])
plt.title('Error')
plt.xlabel('Epoca')

# #Matriz de Confusión

from sklearn.metrics import classification_report,
confusion_matrix

# Predecir los datos de prueba
y_pred = model.predict(X_test)
y_pred = np.round(y_pred).astype(int)  # Redondear las
predicciones y convertir a enteros

# Calcular la matriz de confusión
conf_matrix = confusion_matrix(y_test, y_pred

sns.heatmap(conf_matrix, annot=True, fmt = "d")

### Reporte de clasificación

print(classification_report(y_test,y_pred))

# FIN DEL CÓDIGO #
```

□

Conclusión

En este proyecto, desarrollamos un modelo de clasificación para la detección de peatones utilizando Keras y técnicas de extracción de características con HOG (Histogram of Oriented Gradients). Los resultados obtenidos fueron sobresalientes, alcanzando una precisión del 99%. El análisis detallado de la matriz de confusión reveló que nuestro modelo clasificó correctamente 81 de 82 imágenes sin peatones y 89 de 90 imágenes con peatones. Solo hubo 1 falso positivo y 1 falso negativo, lo que demuestra una alta capacidad de generalización del modelo.

El proceso de preprocesamiento incluyó la conversión de imágenes a escala de grises y la redimensión a un tamaño uniforme, lo cual facilitó la extracción de características robustas. La red neuronal diseñada consta de dos capas ocultas con activación ReLU y una capa de salida con activación *sigmoid* para la clasificación binaria. El entrenamiento se realizó con

una tasa de aprendizaje optimizada y un tamaño de batch adecuado, lo que contribuyó a la rápida convergencia del modelo.

La evaluación del modelo se realizó utilizando un conjunto de datos de prueba separado, donde se observaron métricas de precisión, recall y F1-score cercanas al 99%. La gráfica de exactitud a lo largo de las épocas mostró una rápida estabilización, indicando un entrenamiento eficiente y una buena adaptación a los datos.

Estos resultados demuestran que la combinación de técnicas de visión por computadora y redes neuronales profundas es efectiva para la detección de peatones en imágenes. Además, el uso de herramientas como Keras facilita la implementación y optimización de modelos complejos, permitiendo obtener resultados de alta calidad en aplicaciones prácticas de navegación autónoma. Este enfoque puede ser expandido y mejorado para su uso en sistemas de conducción autónoma y otras aplicaciones de seguridad vial.

Aspectos Clave:

1. Preprocesamiento de Imágenes: Utilizamos HOG para extraer características robustas y significativas de las imágenes, lo que mejoró la capacidad del modelo para diferenciar entre peatones y no peatones. Las imágenes fueron convertidas a escala de grises y redimensionadas a un tamaño uniforme para estandarizar la entrada al modelo.

2. Modelado con Keras: Diseñamos un modelo de red neuronal con dos capas ocultas, cada una con activación ReLU, y una capa de salida con activación *sigmoid* para la clasificación binaria. Esta arquitectura demostró ser eficaz para manejar el conjunto de datos y proporcionar predicciones precisas.

3. Optimización del Modelo:Ajustamos hiperparámetros clave como la tasa de aprendizaje y el tamaño del batch, lo que contribuyó a una rápida convergencia y estabilidad del modelo durante el entrenamiento. También empleamos la técnica de escalado MinMax para normalizar los datos de entrada, mejorando así el rendimiento del modelo.

4. Resultados:Nuestro modelo alcanzó una precisión del 99%, validada por una matriz de confusión que mostró solo 1 falso positivo y 1 falso negativo. Esto indica una excelente capacidad de clasificación y generalización.

5. Evaluación del Rendimiento:Utilizamos métricas de precisión, recall y F1-score para evaluar el rendimiento del modelo. La alta precisión y consistencia en estas métricas aseguran que el modelo funciona bien tanto en el conjunto de entrenamiento como en el de prueba, proporcionando confianza en su uso para aplicaciones prácticas de detección de peatones.

6. Visualización de Resultados:Las gráficas de dispersión de los datos escalados y las curvas de aprendizaje mostraron una rápida mejora y estabilización de la precisión, reflejando un proceso de entrenamiento eficiente. La matriz de confusión y el reporte

de clasificación proporcionaron una visión detallada de la capacidad predictiva del modelo.

Bibliografía

Venturi, L., & Korda, K. (2020). Hands-On Vision and Behavior For Self-Driving Cars. Packt Publishing. Disponible en: O'Reilly Learning. Licencia: Copyright © 2020 Packt Publishing.

Ranjan, S., & Senthamilarasu, S. (2020). Applied Deep Learning and Computer Vision for Self-Driving Cars. Packt Publishing Ltd. Disponible en: O'Reilly Learning. Licencia: Copyright © 2020 Packt Publishing.

Konstantin, V. (n.d.). *Human Detection Dataset*. Kaggle. Disponible en: https://www.kaggle.com/datasets/constantinwerner/human-detection-dataset.

Dalal, N., & Triggs, B. (2005). *INRIA Person Dataset*. Papers with Code. Disponible en: https://paperswithcode.com/dataset/inria-person.

Mazurov, M. (n.d.). *House, Rooms & Streets Image Dataset*. Kaggle. Disponible en: https://www.kaggle.com/datasets/mikhailma/house-rooms-streets-image-dataset.