Traffic Signs Recognition

Berinde Amalia, IA, sg1

I. Introducere

Obiectivul acestui proiect este construirea unui model de rețea neuronală convoluțională capabil sa clasifice semnele de circulatie din imagini in categoria corespunzatoare lor.

Pentru realizarea proiectului se foloseste setul de date GTSRB- German Traffic Sign Recognition Benchmark disponibil pe Kaggle. Setul de date contine peste 50.000 de imagini cu diferite semne de circulatie care sunt clasificate in 43 de directoare. Pe langa setul de date, proiectul se foloeste si de biblioteci precum: TensorFlow, Keras si scikit-learn.

Exemple de imagini din setul de date :

II. Incarcarea setului de date

Prima etapa a proiectului presupune preluarea imaginilor si a etichetelor(labels) acestora din setul de date prin parcurgerea celor 43 de directoare si stocarea in liste. In cazul imaginilor, inainte de adaugarea in lista, se face o redimensionare a acestora. Dupa stocarea in liste, acestea sunt convertite in numpy arrays.

III. Pregatirea setului de date pentru antrenare

Inainte de a construi modelul retelei neuronale, se foloseste metoda *train_test_split()* din biblioteca scikit-learn pentru a impartii datele in setul de antrenare si cel de testare. In cazul acestui proiect 80% din date vor fi folosite pentru antrenare si 20% pentru testare.

IV. Construirea modelului

Primul pas al acestei etape este declararea modelului drept model *Sequential(),* deoarece aceasta retea neuronala are mai multe straturi.

In continuare, adaugam modelului:

- Straturile de convolutie cu activare ReLU. Un strat de convolutie realizeaza o serie de operatii de filtrare liniara pe matricea de la intrarea in strat.
- Max pooling pentru a reduce dimensiunea spatiala. Operatia de pooling inlocuieste valoarea unei zone din imagine (feature map) cu o statistica a acelei zone. Functia de max-pooing inlocuieste valoarea dintr-o zona bine definita cu maximul acelei zone, rezultand un feture map mai mic.
- Dropout pentru a anula contributia unor neuroni catre urmatorul strat. Structurile de dropout sunt importante in antrenamentul CNN-urilor, deoarece previn supraadaptarea (overfitting) datelor de antrenament. Daca nu avem dropout, primul lot de date din antrenament influenteaza invatarea intr-o maniera disproportionat de mare
- Flatten converteste matricea bi-dimensionala de neuroni intr-un vector care este pasat stratului complet conectat.
- Dense adauga stratul complet conectat la reteaua neuronala.

Secventa de cod corespunzatoare construirii modelului este urmatoarea:

```
model = Sequential()


model.add(Conv2D(filters=32, kernel_size=(5, 5), activation='relu', input_shape=x_train.shape[1:]))
model.add(Conv2D(filters=32, kernel_size=(5, 5), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))


model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))


model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(rate=0.5))
model.add(Dense(43, activation='softmax'))

model.summary()
```

V. Antrenarea retelei neuronale

Dupa construirea modelului urmeaza compilarea acestuia folosind optimizatorul ‘adam’, functia de loss ‘categorical_crossentropy’ si pentru metrics 'accuracy'

Pentru a efectua antrenarea propriu-zisa se foloseste model.fit(), caruia ii sunt pasati parametri corespunzatori.

```
#Compilation of the model
model.compile(optimizer='adam' ,loss='categorical_crossentropy' ,metrics=['accuracy'])
```

✓ 0.4s

```
history = model.fit(x_train, y_train, batch_size=32,
                    epochs=30, validation_data=(x_test, y_test))
```

Epochs reprezinta numarul de iterari pentru care reteaua neuronala ruleaza un antrenament.

VI. Testare

Dupa antrenarea retelei neuronale testam acuratetea pe directorul Test al setului de date folosind metodele predict si accuracy_score din biblioteca sklearn.

```
from sklearn.metrics import accuracy_score
import pandas as pd

y_test = pd.read_csv(r'C:\Users\user\Desktop\Traffic_Sign_Recognition\Test.csv')

labels = y_test["ClassId"].values
imgs = y_test["Path"].values

data = []

for img in imgs:
    image = Image.open(img)
    image = image.resize((30, 30))
    data.append(np.array(image))

X_test = np.array(data)

pred = np.argmax(model.predict(X_test), axis=1)

#Accuracy with the test data
print(accuracy_score(labels, pred))
```

✓ 12.8s

```
395/395 [==============================] - 7s 19ms/step
0.9554235946159937
```

Pentru a verifica daca predictiile facute sunt corecte am creat functia test_on_img() care primeste ca input o imagine . Folosind metoda model.predict() si cu ajutorul unui dictionar care asociaza eticheta imaginii cu stringul corepunzator, verificam predictia modelului.

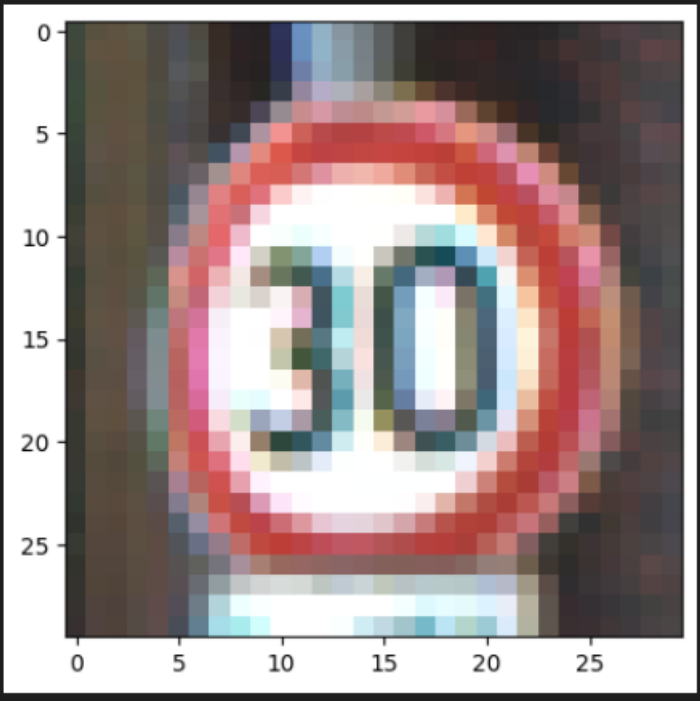
```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
def test_on_img(img):
    data=[]
    image = Image.open(img)
    image = image.resize((30,30))
    data.append(np.array(image))
    x_test=np.array(data)
    y_pred = model.predict(x_test)
    classes_x = np.argmax(y_pred, axis=1)
    return image,classes_x
```

✓ 0.4s

```
plot, prediction = test_on_img(r'C:\Users\user\Desktop\Traffic_Sign_Recognition\Test\00001.png')
s = [str(i) for i in prediction]
a = int("".join(s))
print("Predicted traffic sign is: ", classes[a])
plt.imshow(plot)
plt.show()
```

✓ 0.3s

```
1/1 [==============================] - 0s 119ms/step
Predicted traffic sign is:  Speed limit (30km/h)
```



Referinte bibliografice:

-Axel Thevenot (2020), ‘Conv2d: Finally Understand What Happens in the Forward Pass’ [1]

-Machine Learning pentru Aplicatii Vizuale - Retele Convolutionale [2]

-Baeldung (2022), ‘How ReLU and Dropout Layers Work in CNNs’ [3]

- Traffic-Signs-Recognition-using-CNN-Keras [4]

-Mykola, GTSRB - German Traffic Sign Recognition Benchmark [5]

- Derrick Mwiti (2018), ‘Convolutional Neural Networks: An Intro Tutorial’ [6]