



Facultate : Automatica si Calculatoare

Specializare: Calculatoare si Tehnologia Informatiei

Disciplina : Procesare de Imagini

Proiect : Recunoasterea semnelor de circulatie

Student : Herman Felician-Nicu

Seria : B

Grupa : 30239

## Cuprins

1. Prezentarea Temei .....	3
2.Functionalitati .....	3
3.Detalii de implementare .....	3
3.1 Structura proiectului .....	3
3.2 Functii si algoritmi.....	4
3.3 Structuri de date si biblioteci .....	5
4.Manual de utilizare .....	6
5.Concluzii si dezvoltari ulterioare.....	7
6.Referinte .....	7

## 1. Prezentarea Temei

Proiectul ne introduce in zona de machine learning, fiind format din doua mari parti, una de antrenare a unei retea neuronale iar cealalta de testare a acelei retele pe baza unor imagini de test.

Pentru respectiva antrenare avem mai multe foldere cu imagini pe baza carora modelul o sa invete sa recunoasca semnele de circulatie. In final proiectul o sa fie capabil de clasificarea diferitelor imagini primite de la utilizatori.

## 2.Functionalitati

Ca si prima parte, am reusit sa implementez un model antrenat sa recunoasca semnele de circulatie, acesta fiind salvat in folderul proiectului cu numele de „traffic\_classifier”. Mai apoi, in partea a doua, modelul a fost incarcat si un mesaj cu denumirea semnului din imaginea incarcata de utilizatori o sa fie vizibil.

In momentul in care utilizatorul trebuie sa incarce o imagine, o sa fie deschis un fișier dialog care permite navigarea prin folderul cu imaginile de test si selectarea pe rand a cate unei imagini pana cand programul este oprit.

## 3.Detalii de implementare

### 3.1 Structura proiectului

Proiectul este structurat in doua surse de tipul python, una in care antrenam modelul iar cealalta in care il testam. Am ales sa creez doua surse si sa la impart astfel deoarece antrenarea modelului dureaza destul de mult in momentul rularii(cateva minute bune, depinde de numarul de epochs) iar acest lucru este destul de inconfortabil. De aceea modelul este antrenat o singura data, fiind creat si salvat in folderul radacina, mai apoi fiind nevoie numai sa il incarcam si sa testam.

Deasemenea, in folderul proiectului avem si toate datele(imaginile) pe baza carora o sa lucram. In folderul Train se gasesc 43 de subfoldere, fiecare continand diverse imagini cu acel semn de circulatie in diverse conditii(luminozitate diferita, anumite rotatii sau scalari). Pe langa acel folder, avem si cel de Test unde sunt mai multe imagini disponibile utilizatorilor pentru a fi incarcate spre testare. Numele semnelor sunt salvate in fisierul excel „signnames” care o sa fie incarcat in prima sursa python fiind citite pentru utilizare ulterioara.

In final, o sa apara si modelul salvat sub forma .h5, cu numele de „traffic\_classifier.h5”, model pe care o sa il incarcam si o sa il testam.

### 3.2 Functii si algoritmi

Pentru inceput, am implementat un algoritm de importare a imaginilor din clasele cu datele de train, fiecare imagine fiind redimensionata pentru a avea imagini de aceeaasi dimensiune pe tot parcursul proiectului. Apoi imaginile sunt convertite in array(necesare CNN-ului), deasemenea arrayul cu numele claselor sunt memorate pentru a putea recunoaste semnul respectiv.

Pe langa resize-ul facut in momentul importarii, imaginilor li se mai aplica niste filtre, precum egalizarea histogramelor(pentru aceeaasi luminozitate), transformarea acestora din canalele de culoare RGB la GraYscale si normalizarea acelor valori.

Apoi, liniile sunt ingrosate, incremendat datele de train, validare si test, cu 1. Augumentarea imaginilor, sau marirea datelor de train este reprezentata de generarea unor imagini pe baza unei imagini, acestea avand diferite lucruri modificate, precum zoom diferit, rotatie in oarecare parte sau fiind distorsionata asupra unei axe.

La baza modelului se gaseste „Convolutional Neural Networks”, care functioneaza fenomenal in lucrul cu clasificarea imaginilor. Acesta functioneaza prin folosirea unor „neuroni” capabili sa fie invatati, acestia primind anumite inputuri pe baza imaginilor de train care sunt prelucrate ca in final sa poate oferi ca si raspuns un anumit output. Similaritatile realizate intre imaginile de train si cele de test sunt punctul forte a acestui model, mai apoi folosind si recunoasterea de obiecte. Dupa primirea imaginii de input, aceasta este trecute prin mai multe filtre, in final avand un rezultat, care in cazul nostru o sa fie una dintre cele 43 de clase cu semne.

Modelul Sequential, cel pe care il folosim noi, cuprinde lucrul cu mai multe layere liniare, provenite de la o imagine si returnand un raspuns. Pe rand se adauga diferite filtre, inainte ca modelul sa le compileze pe toate acestea.

Functia Flatten() redimensioneaza shapeurile cu datele colectate din imagine, pentru a putea compila modelul pe seturi de date de aceleasi dimensiun.

In final, modelul este antrenat pe baza CNN si in cele din urma salvat, ca mai apoi sa fie incarcat pentru recunoasterea semnelor.

In partea de testare, am implementat o functie care incarca modelul din folderul fisierului si il pregateste pentru testare. Deasemenea functiile de prelucrare a imaginilor sunt implementate si aici, fiind nevoie de aplicarea acestora pe imaginea incarcata de catre utilizator inainte de clasificarea acesteia.

Functia care ne returneaza numele semnului pe baza clasei din care face parte este implementat ca si „getName”, primeste un numar(de la 0 la 42) si afiseaza semnul care e prezent in clasa cu numarul respectiv.

Ca si concluzie, am implementat inca doua functii, una de selectare a pathului, adica a locatiei imaginii pe care sa o clasificam, iar cea de clasificare care aplica functiile implementate, o clasifica si afiseaza numele semnului.

Intre clasificarea a oricaror doua imagini, programul o sa astepte aproximativ 4 secunde pana o sa fie disponibila selectarea altei imagini, totul fiind inchis in momentul opririi rularii programului principal.

Ca si acuratete a modelului, in momentul in care acesta este compilat, dupa 10 pasi, se ajunge destul de aproape de 100%, atingand un procent de aproape 97%. In primul pas modelul este eficient in proportie de 38,54%, apoi evoluand la fiecare pas, destul de mult la inceput, al doilea pas ajungand la 87,46%.

```
785/785 [=====] - 51s 23ms/step - loss: 2.2628 - accuracy: 0.3854 - val_loss: 0.1836 - val_accuracy: 0.9477
Epoch 2/10
785/785 [=====] - 21s 27ms/step - loss: 0.4055 - accuracy: 0.8746 - val_loss: 0.1271 - val_accuracy: 0.9619
Epoch 3/10
785/785 [=====] - 25s 32ms/step - loss: 0.2612 - accuracy: 0.9194 - val_loss: 0.0804 - val_accuracy: 0.9790
Epoch 4/10
785/785 [=====] - 26s 33ms/step - loss: 0.2118 - accuracy: 0.9338 - val_loss: 0.0574 - val_accuracy: 0.9879
Epoch 5/10
785/785 [=====] - 26s 33ms/step - loss: 0.1675 - accuracy: 0.9496 - val_loss: 0.0800 - val_accuracy: 0.9824
Epoch 6/10
785/785 [=====] - 26s 34ms/step - loss: 0.1448 - accuracy: 0.9555 - val_loss: 0.0409 - val_accuracy: 0.9879
Epoch 7/10
785/785 [=====] - 26s 33ms/step - loss: 0.1315 - accuracy: 0.9595 - val_loss: 0.0339 - val_accuracy: 0.9899
Epoch 8/10
785/785 [=====] - 25s 32ms/step - loss: 0.1139 - accuracy: 0.9644 - val_loss: 0.0351 - val_accuracy: 0.9912
Epoch 9/10
785/785 [=====] - 25s 32ms/step - loss: 0.1065 - accuracy: 0.9665 - val_loss: 0.0260 - val_accuracy: 0.9921
Epoch 10/10
785/785 [=====] - 25s 32ms/step - loss: 0.1113 - accuracy: 0.9669 - val_loss: 0.0338 - val_accuracy: 0.9912
```

### 3.3 Structuri de date si biblioteci


Ca si biblioteci principale, am folosit „cv2”, care cuprinde majoritatea functiilor din OpenCV pentru lucrul cu imagini, diferite transformari sau operatii fiind disponibile. Deasemenea, pentru importarea imaginilor avem nevoie de „os” care e o biblioteca legata de sistemul de operare unde ne este permisa listarea diferitelor directoare din anumite foldere.

Tensorflow fiind cea mai importanta biblioteca, de aceasta fiind nevoie pentru a apela toate functiile legate de model, adaugarea filtrelor la acesta, compilarea si antrenarea acestuia. Din biblioteca tensorflow am important si keras, cele doua functionand foarte bine impreuna. Pentru buna functionare a acestora am avut nevoie de Python 3.8 si de CUDA, care este o aplicatie prin care modelul si antrenarea au loc pe procesorul din placa video, pe GPU, nu pe CPU.

Pentru partea de testare, am importat filedialog din biblioteca tkinter ca sa fie posibila selectarea unei imagini din folderul de test.

Pentru versiunile tuturor bibliotecilor, o sa incarc o poza cu tot ce am instalat in interpretorul proiectului.

Python interpreter:

 Project Default (Python 3.8 (PROIECTPI) (2)) C:\Users\My-Pc\Desktop\PROIECTPI\venv\Scripts\pyth

```
(venv) C:\Users\My-Pc\Desktop\PROIECTPI>pip list
```

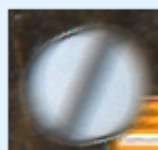
Package	Version		Version
absl-py	0.12.0	protobuf	3.17.0
astunparse	1.6.3	pyasn1	0.4.8
cachetools	4.2.2	pyasn1-modules	0.2.8
certifi	2020.12.5	python-dateutil	2.8.1
chardet	4.0.0	pytz	2021.1
flatbuffers	1.12	requests	2.25.1
gast	0.4.0	requests-oauthlib	1.3.0
google-auth	1.30.0	rsa	4.7.2
google-auth-oauthlib	0.4.4	scikit-learn	0.24.2
google-pasta	0.2.0	scipy	1.6.3
grpcio	1.34.1	setuptools	56.2.0
h5py	3.1.0	six	1.15.0
idna	2.10	sklearn	0.0
joblib	1.0.1	tensorboard	2.5.0
keras-nightly	2.5.0.dev2021032900	tensorboard-data-server	0.6.1
Keras-Preprocessing	1.1.2	tensorboard-plugin-wit	1.8.0
Markdown	3.3.4	tensorflow	2.5.0
numpy	1.19.5	tensorflow-estimator	2.5.0
oauthlib	3.1.0	termcolor	1.1.0
opencv-python	4.5.2.52	threadpoolctl	2.1.0
opt-einsum	3.3.0	typing-extensions	3.7.4.3
pandas	1.2.4	urllib3	1.26.4
pip	21.1.1	Werkzeug	2.0.0
		wheel	0.36.2
		wrapt	1.12.1

## 4.Manual de utilizare

Din punct de vedere al modelului, acesta este deja antrenat si salvat, sursa de test incarcandu-l si fiind pregatita pentru a-l testa. Utilizatorul trebuie sa ruleze aceasta sursa, mai apoi urmand sa efectueze selectia unei imagini pentru a o testa. O sa prezint in continuare modul de utilizare, incarcand cinci imagini iar in final afisand rezultatele.



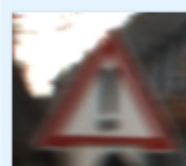
00121



00306



00764



00331



08689

```
Speed Limit (100 km/h)
End of all speed and passing limits
General Caution
Priority road
Keep right

Process finished with exit code 0
```

Dupa cum este vizibil, indiferent de claritatea, dimensiunea sau luminozitatea imaginilor de test, modelul functioneaza perfect fiind capabil de a recunoaste ce doreste utilizatorul afisand numele semnului respectiv.

## 5.Concluzii si dezvoltari ulterioare

Initial, datele pentru train si test au fost gasite destul de usor, fiind disponibile online in gama larga, acest subiect este foarte dezbatut si dezvoltat in zilele noastre, masinile autonome devenind din ce in ce mai utilizate si apreciate.

Ca si dificultati, reusirea rezolvarii compatibilitatii dintre python, tensorflow, keras si cuda a fost cea mai dificila din punctul meu de vedere, acest lucru durand cateva zile in care am instalat diferite pachete, am urmarit diferite tutoriale si am descarcat o multitudine de aplicatii.

Mai apoi, modelarea CNN-unului pentru a obtine o acuratete destul de mare a reprezentat o piedica in implementarea proiectului.

Ca si dezvoltari ulterioare proiectul poate sa fie capabil de recunoasterea tuturor semnelor din o imagine oarecare si afisarea lor in ordinea prioritatilor sau in orindea aparitiilor in parcurgerea imaginii.

## 6.Referinte

<https://www.hindawi.com/journals/mpe/2015/250461/>

<https://data-flair.training/blogs/python-project-traffic-signs-recognition/>

<https://www.pyimagesearch.com/2019/11/04/traffic-sign-classification-with-keras-and-deep-learning/>

<https://www.youtube.com/watch?v=SWaYRyi0TTs&t=63s>

<https://stackoverflow.com/questions/67549661/importerror-cannot-import-name-layernormalization-from-tensorflow-python-ker/67667525#67667525>

<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6767627/>