

**CAIET DE PROIECT LA
INFORMATICĂ**

SELF DRIVING CAR

Elev: Moiceanu Andrei
Profesor îndrumător: Hăbuc
Ştefăniță



SELF DRIVING CAR

CUPRINS

- Intro: Motivul alegerii temei
- Capitolul I: Limbajul de programare Python
 - I.1:Scurt istoric al limbajului Python
 - I.2:De ce Python?
 - I.3:Sintaxa python
 - I.4:Tipuri de date
 - I.5:Biblioteci
- Capitolul II: Rețeaua Artificială de Neuroni
 - II.1:Ce este o rețea de neuroni și la ce ajută
 - II.2:Modalități de a reprezenta o rețea de neuroni
 - II.3Tipuri de rețele de neuroni
 - II.4:Realizarea calculelor într-o rețea de neuroni(Algoritmul Feed-Foward)
 - II.5:Reducerea erorilor (Algoritmul Gradient Descent)
 - II.6:Optimizarea rețelei (Algoritmul Back-Propagation)

SELF DRIVING CAR

CUPRINS

- Capitolul III:Realizarea și implementarea rețelei neuronale
 - III.1:Google Colab
 - III.2:Preluarea datelor
 - III.3:Procesarea datelor
 - III.4:Modelarea, analizarea și optimizarea rețelei de neuroni
 - III.5:Implementarea rețelei intr-un sistem:Server-Client
- Capitolul IV: Descrierea aplicației:
 - IV.1:Despre simulatorul Udacity
 - IV.2:Cerințe de sistem
 - IV.3:Cum se instalează programul
 - IV.4:Cum se utilizează programul
 - IV.5:Folosirea rețelei neuronale in proiecte de viitor
- Capitolul V: Bibliografie

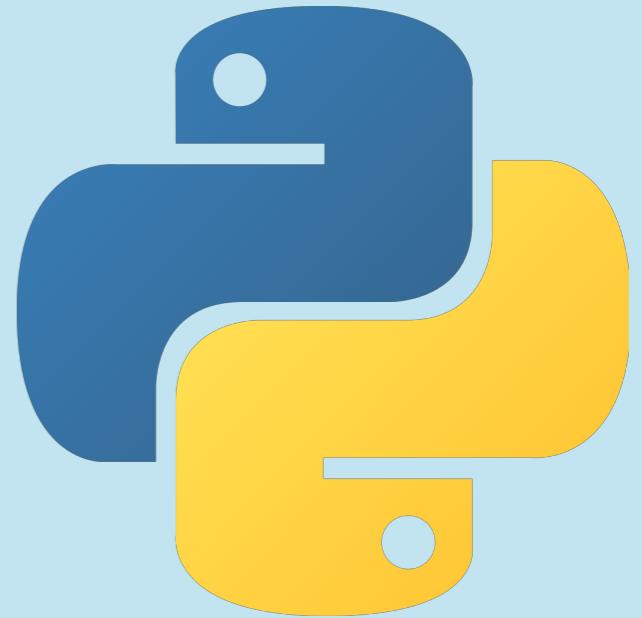
Introducere

MOTIVAȚIA ALEGERII TEMEI

Am ales acestă temă deoarece se ocupă de aria mea favorită de studiu din domeniul informaticii (Inteligenta Artificială) și aplicația este una interesantă și de actualitate.Vedem cum firme precum Tesla sau Google dezvoltă vehicule autonome dar și cum producători clasici de autovehicule doresc să implementeze sisteme semi-autonome pe vehiculele lor, companii mari precum BMW, Grupul Volkswagen, Mercedes-Benz sau Renault.

Acest domeniu este un domeniu căutat, interesant și plin de provocări.Am dorit ca prin acest proiect să capăt cunoștiințe noi,să rămân cu ceva important și să mă pot mândri cu el când îl voi prezenta într-un viitor CV.De asemenea acest proiect oferă continuitate mai sunt multe lucruri de făcut, probleme de rezolvat și provocări de înfruntat până când voi ajunge să obțin rezultatul final, un autovehicul complet autonom.Acest lucru poate fi inceputul unei cariere, fiind nevoie de programatori în acest domeniu pentru a face traficul mult mai sigur astfel de sisteme dorind să fie implementate pe mijloace de transport în comun de companii precum Uber.

CAPITOLUL I: LIMBAJUL DE PROGRAMARE PYTHON



1: Scurt istoric al limbajului Python

Python este un limbaj de programare dinamic multi-paradigmă, creat în 1989 de programatorul olandez [Guido van Rossum](#). Van Rossum este și în ziua de astăzi un lider al comunității de dezvoltatori de software care lucrează la perfectionarea limbajul Python și implementarea de bază a acestuia, CPython, scrisă în C. Python este un limbaj multifuncțional folosit de exemplu de către companii ca [Google](#) sau [Yahoo!](#) pentru programarea aplicațiilor web, însă există și o serie de aplicații științifice sau de divertisment programate parțial sau în întregime în Python. Popularitatea în creștere, dar și puterea limbajului de programare Python au dus la adoptarea sa ca limbaj principal de dezvoltare de către programatori specializați și chiar și la predarea limbajului în unele medii universitare. Din aceleași motive, multe sisteme bazate pe Unix, inclusiv [Linux](#), [BSD](#) și [Mac OS X](#) includ din start interpretatorul CPython.

Python pune accentul pe curățenia și simplitatea codului, iar sintaxa sa le permite dezvoltatorilor să exprime unele idei programatice într-o manieră mai clară și mai concisă decât în alte limbaje de programare ca C. În ceea ce privește paradigma de programare, Python poate servi ca limbaj pentru software de tipul object-oriented, dar permite și

programarea imperativă, funcțională sau procedurală. Sistemul de tipizare este dinamic iar administrarea memoriei decurge automat prin intermediul unui serviciu „gunoier” (*garbage collector*). Alt avantaj al limbajului este existența unei ample biblioteci standard de metode.

Implementarea de referință a Python este scrisă în C și poartă deci numele de *CPython*. Această implementare este software liber și este administrată de fundația *Python Software Foundation*.

2:De ce python?

Am ales acest limbaj de programare datorită versatilității sale,suportului mare pe care îl are dar și sintaxei simple

care face posibilă rezolvarea unei probleme în mai puține linii de cod.Ca orice limbaj Python vine cu avantajele și dezavantajele sale.Puteți observa aceste lucruri in tabelul de mai jos.

Avantajele și dezavantajele limbajului de programare Python	
Avantaje	Dezavantaje
Versatilitate	Viteză redusă de compilare
Compatibilitate multi-platformă	Incapacitatea de a lucra direct cu memoria
Suport tehnic mare	Nevoia de a avea un interpretator instalat
Ușor	

SINTAXA PYTHON

Ca orice limbaj de programare Python are o sintaxă. Dorind să simplifice sintaxă, limbajul Python are un mod de abordare puțin diferit față de alte limbaje care se aseamănă cu C.

Acoladele "{}" sunt înlocuite cu două puncte ":" și secvența dintr-un bloc de comandă se identează cu un spațiu tab(spațiu generat prin apăsarea tastei tab).

Blocurile condiționale își păstrează sintaxa familiară:

If conditie :

comanda

While conditie:

comanda

```
def rutinaSaptamanala(zi):
    zile_de_munca = ['luni','marti','miercuri','joi','vineri']
    if zi in zile_de_munca:
        trezeste_te_la_sapte()
        return 0
    else:
        trezeste_te_la_zece()
        return 0
rutinaSaptamanala("Duminica")
```

Structura repetitivă pentru(for este modificată)

C.

```
#include<stdio.h>

char zile[7][10]={"Luni","Marti","Miercuri","Joi","Vineri","Sambata"};
for(int i=0;i<7;i++){
    printf("%s",zile[i]);
}
```

Python

```
zile = ['luni','marti','miercuri','joi','vineri','sambata','duminica']
for zi in zile:
    print zi
```

Dorind să fie mai ușor de învățat Python adoptă o formă mai intuitivă a structurii for. Renunțând la abstractizarea pe care o oferă limbajul C.

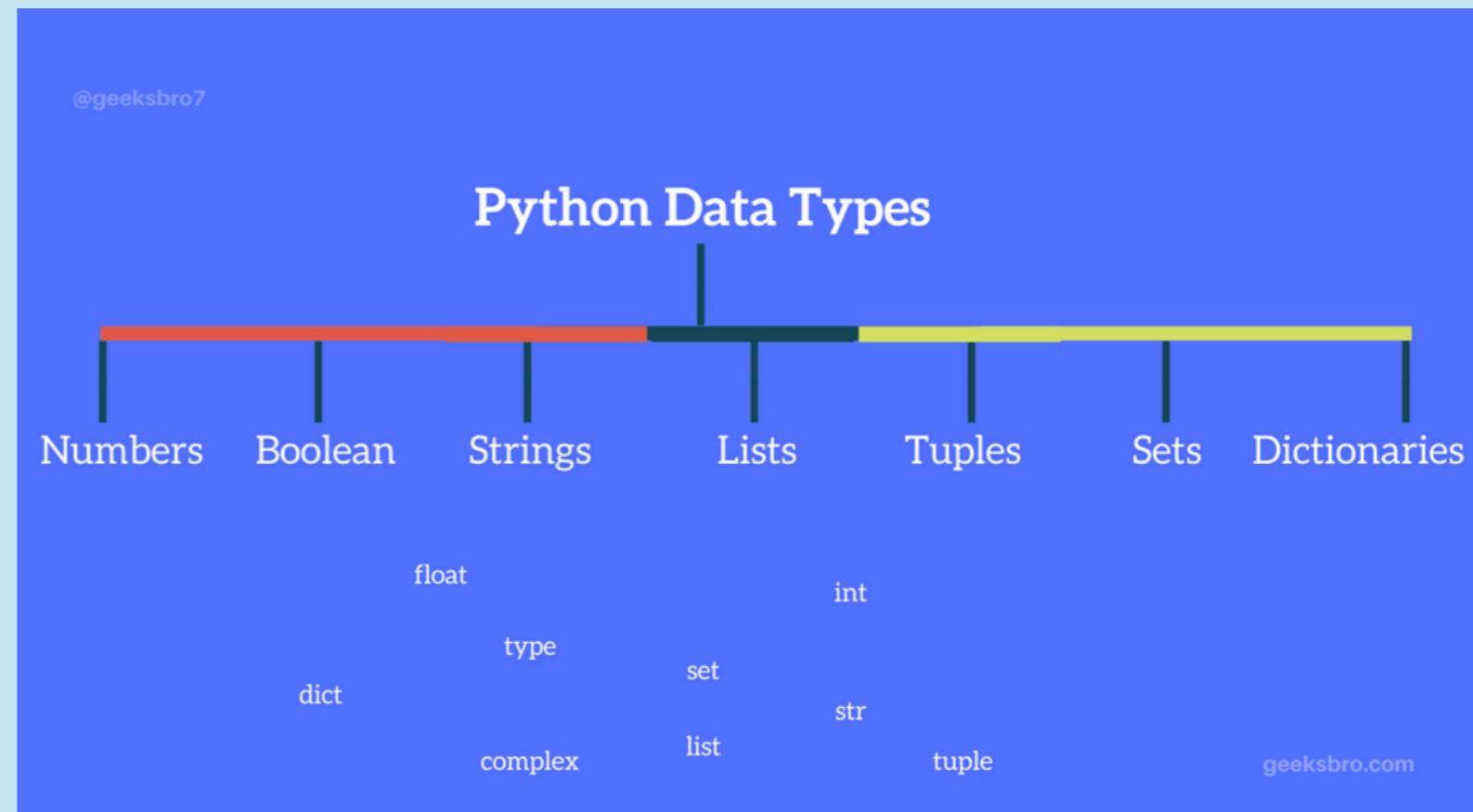
TIPURI DE DATE

Un lucru fundamental al unui limbaj de programare o reprezintă tipurile de date cu care acesta știe să lucreze. Python oferă o gamă destul de redusă de tipuri de date comparativ cu alte limbaje de programare.

Tipul de date "Numbers": - se referă la toate datele de tip numeric (numere naturale, întregi, rationale și reale). Spre deosebire de limbajele clasice de programare (C, C++, Java) toate tipurile de numere sunt incorporate într-un singur tip de date.

Tipul de date "Boolean": un tip de date ce reprezintă un bit (0 sau 1)/Adevărat/Fals. Este folosit pentru implementarea elementelor logice (Condiții).

Tipul de Date Strings: - este folosit pentru a stoca siruri de caractere.



TIPURI DE DATE

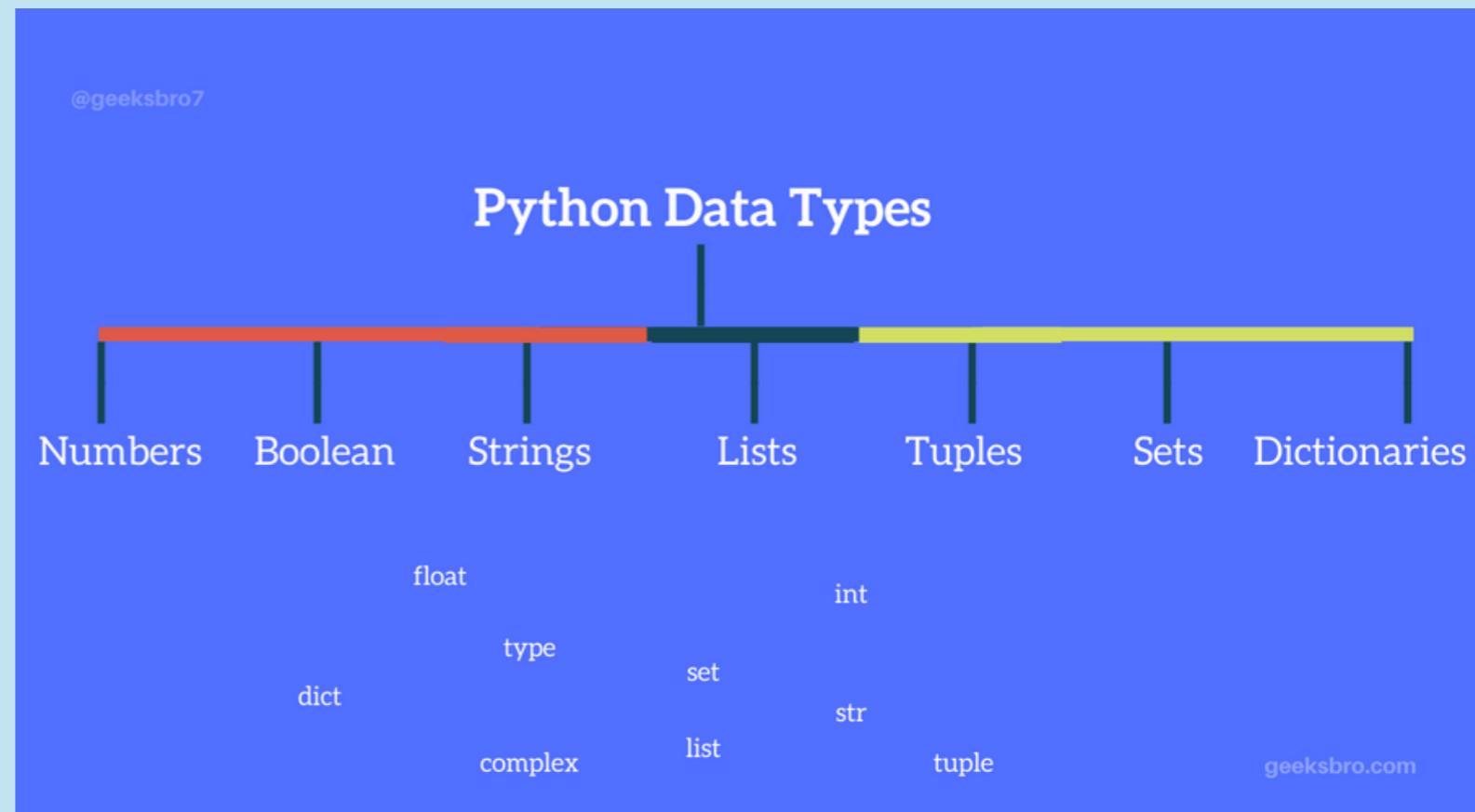
Structuri de date:

1) Liste: se folosesc pentru a stoca mai multe date intr-o singură variabilă(Zilele săptămânii,un meniu, o listă de utilizatori). Spre deosebire de vectorii din C, Listele nu au restricție la tipul de date pe care le pot conține, o listă poate fi formată din date de tipuri diferite.

2) Tuples(Pereche): conține o pereche de date, este folosit în metodele care returnează mai mult de o variabilă.

3) Sets : Este echivalent unei mulțimi în matematică

4) Dicționare: Sunt folosite pentru a stoca date indexate de o cheie(Pereche keye:data), echivalent cu un HashTable din C.



BIBLIOTECI

Avantajul cel mai mare al limbajului Python este implicarea comunității de programatori care îl folosesc. Modularitatea este unul din avantajele sale, pentru asta Python se bucură de o gamă mare de biblioteci. Vom prezenta câteva biblioteci esențiale în domeniul inteligenței artificiale.

1) NumPy - poate cea mai importantă bibliotecă folosită în acest domeniu, această bibliotecă permite efectuarea calculelor de algebra liniară (operații cu matrici) cât și modificarea matricilor.

```
In [2]: import numpy as np  
  
matriceNula = np.zeros((3,3))  
  
In [3]: matriceUnitate = np.ones((3,3))  
  
In [4]: suma = matriceNula + matriceUnitate  
  
In [6]: print (suma)  
[[1. 1. 1.]  
 [1. 1. 1.]  
 [1. 1. 1.]]  
  
In [8]: produs = matriceNula * matriceUnitate  
print(produs)  
[[0. 0. 0.]  
 [0. 0. 0.]  
 [0. 0. 0.]]  
  
In [10]: altaMatrice = np.array([[1,2,3],[5,4,6],[7,8,9]])  
  
In [11]: print(altaMatrice*matriceUnitate)  
[[ 2.  3.  4.]  
 [ 6.  5.  7.]  
 [ 8.  9. 10.]]  
  
In [12]: print(altaMatrice * altaMatrice)  
[[ 1  4  9]  
 [25 16 36]  
 [49 64 81]]
```



2) Matplotlib - bazată pe biblioteca de grafice din Matlab, această bibliotecă este capabilă de generare a diferitelor grafice, foarte importante pentru a vedea evoluția datelor și pentru a optimiza programul.

```
In [13]: import matplotlib.pyplot as plt

In [20]: x = range(0,8)
np.array(x)

Out[20]: array([0, 1, 2, 3, 4, 5, 6, 7])

In [15]: y = np.sin(x)

In [16]: plt.plot(x,y)

Out[16]: [
```

3) cv2 (OpenCV) - bibliotecă folosită pentru procesarea de imagini, foarte importantă pentru recunoașterea obiectelor(Strada,trotuar,obstacole).

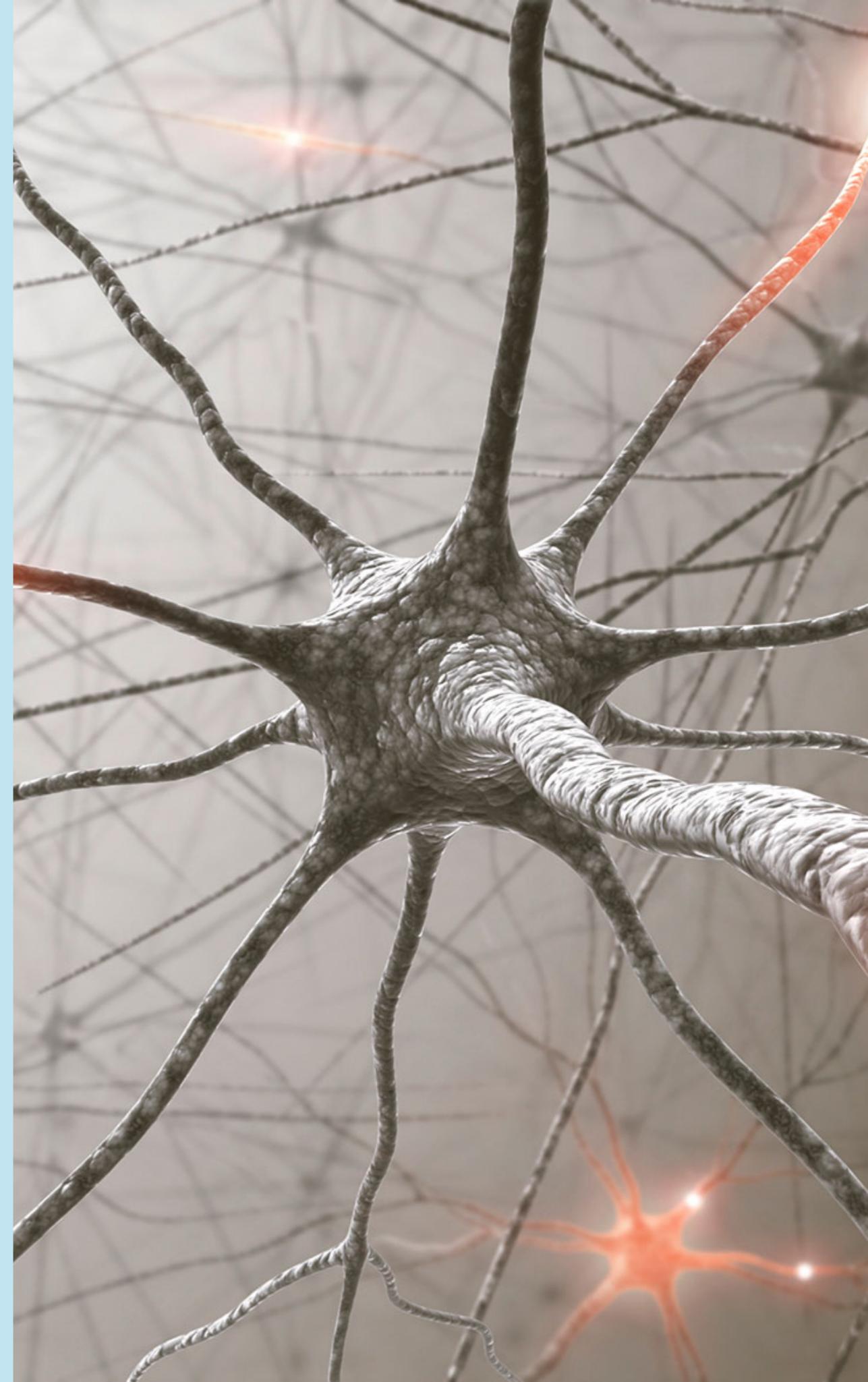
```
def canny(image):
    gray=cv2.cvtColor(image,cv2.COLOR_RGB2GRAY)
    blur=cv2.GaussianBlur(gray,(5,5),0)
    canny=cv2.Canny(blur,50,150)
    return canny
```

-funcție folosită pentru a transforma o imagine coloră în alb negru-

II. REȚEUA ARTIFICIALĂ DE NEURONI

Rețelele neurale(RN, în engleză: **ANN** de la *artificial neural network*) sunt o ramură din știința inteligenței artificiale, și constituie totodată, principal, un obiect de cercetare și pentru neuroinformatică. Rețelele neurale artificiale caracterizează ansambluri de elemente de procesare simple, puternic interconectate și operând în paralel, care urmăresc să interacționeze cu mediul înconjurător într-un mod asemănător creierelor biologice și care prezintă capacitatea de a învăța. Ele sunt compuse din neuroni artificiali, sunt parte a inteligenței artificiale și își au, conceptual, originea ca și neuroni artificiali, în biologie. Nu există pentru RNA o definiție general acceptată a acestor tipuri de sisteme, dar majoritatea cercetătorilor sunt de acord cu definirea rețelelor neurale artificiale ca rețele de elemente simple puternic interconectate prin intermediul unor legături numite interconexiuni prin care se propagă informație numerică.

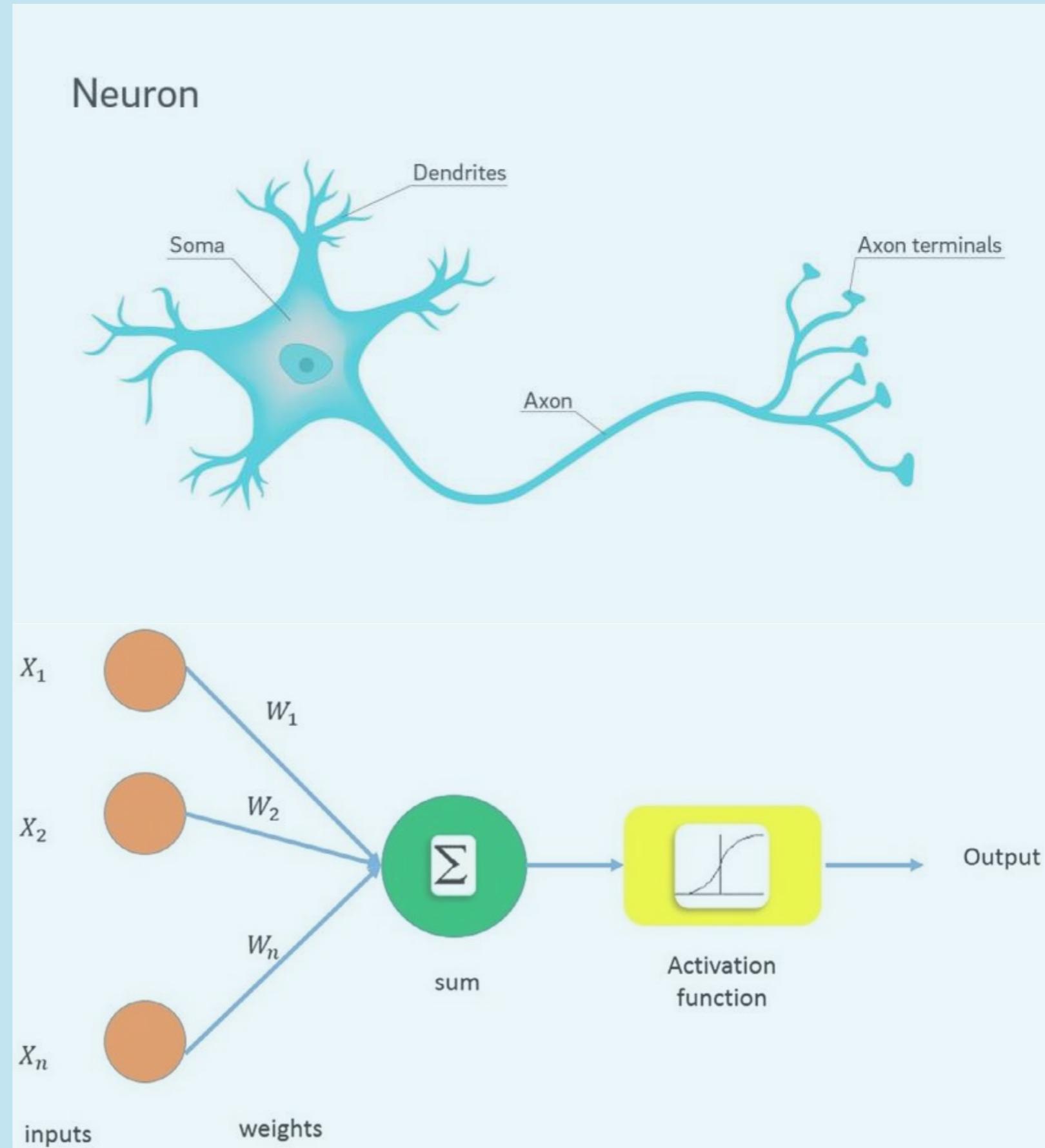
Originea acestor rețele trebuie căutată în studierea rețelelor bioelectrice din creier formate de neuroni și sinapsele acestora. Principala trăsătură a acestor rețele este capacitatea de a *învăța pe bază de exemple*, folosindu-se de experiența anterioară pentru a-și îmbunătăți performanțele.



NEURONUL

Neuronul este elementul fundamental al unei rețele neuronale biologice. În componența sa intră axonii (locul pe unde intră impulsul electric) și dendritele (locul prin care este transim impulsul electric). După modelul biologic al unui neuron a fost realizat și elementul fundamental al unei Rețele Neurale Artificiale -> Perceptronul.

În următorul capitol vom explica funcționarea acestuia și impactul pe care îl are într-o rețea neurală.



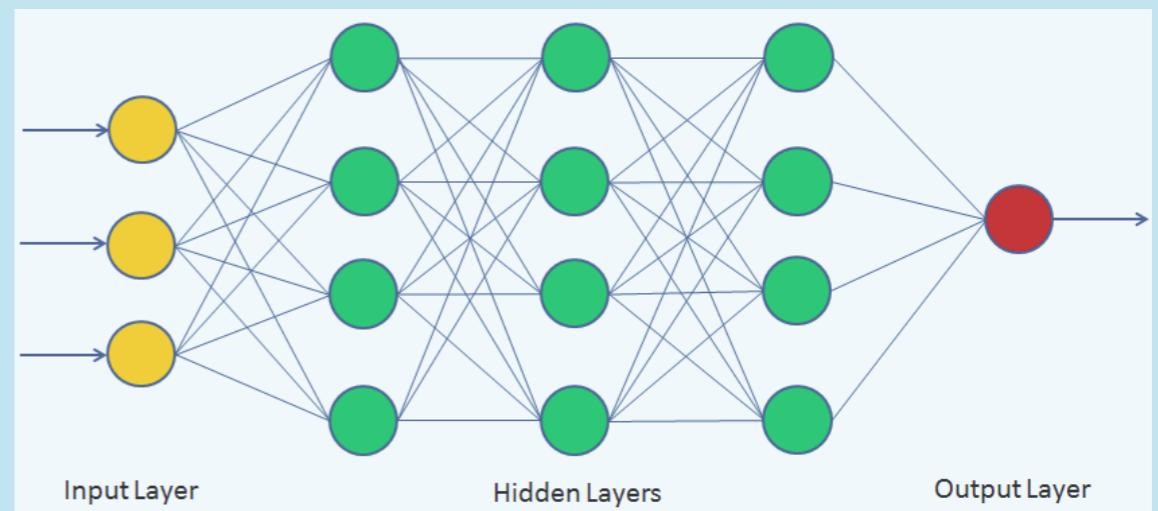
II.2 MODALITĂȚI DE A REPREZENTA O REȚEA DE NEURONI

O rețea de neuroni este un sistem matematic complex, capabil să proceseze un număr foarte mare de date și să își ajusteze singur parametrii (auto-optimizare). O rețea de neuroni poate fi reprezentată printr-un graf orientat, aceasta fiind și structura de date aleasă de cei de la Google în realizarea bibliotecii TensorFlow.

În funcție de starea în care se află rețeaua de neuroni, acesteia îi poate fi atribuit un tip de graf orientat. Când se află în procesul de procesarea a datelor direcția este dată de la nodurile de intrare la cele de ieșire, iar în timpul optimizării direcția este inversată, de la nodurile de ieșire la cele de intrare. Drumul dintre 2 noduri (W_i) este un drum cu cost, valoarea acestuia fiind parametrul ce se optimizează pentru a crește eficiența rețelei.

Pentru a putea fi reprezentă pe un Computer rețeaua este modelată folosind matrici.

O rețea de neuroni este alcătuită din 2 sau mai mulți Perceptroni. Aceștia sunt impărtiți în 3 straturi.

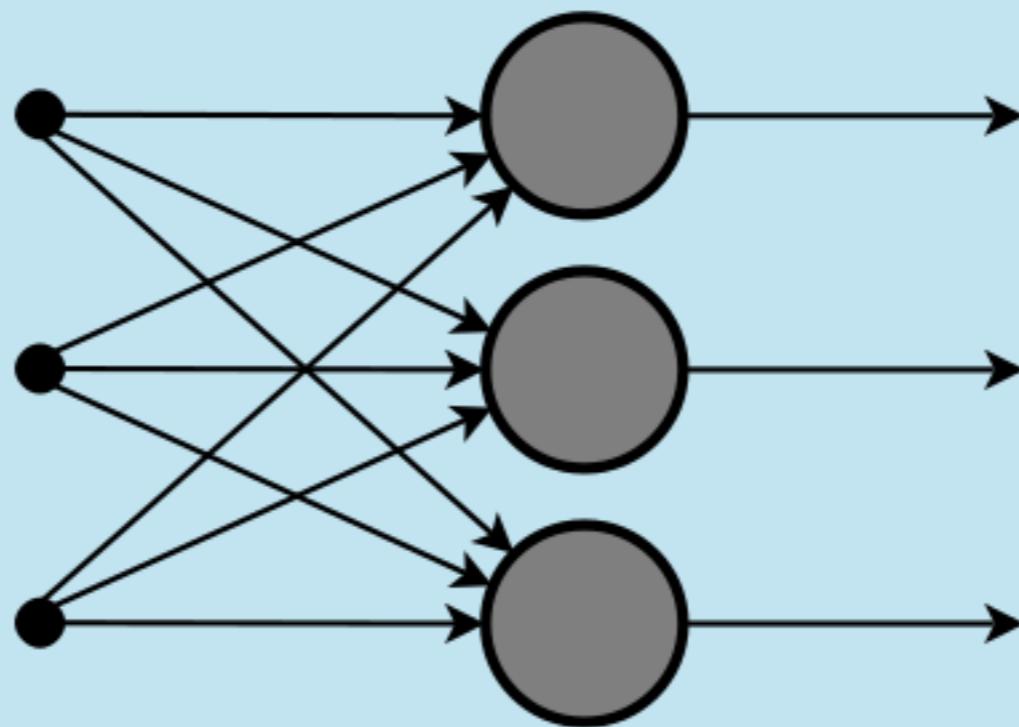


- 1) Stratul de intrare, conține un rând de n Perceptroni cu rol în procesarea datelor de intrare, acestia nu trebuie să fie echivalenți cu numărul datelor de intrare.
- 2) Stratul ascuns, acesta este cel mai complex strat al unei rețele și cel mai dens, este un strat optional și este folosit doar pentru a optimiza o rețea de neuroni.
- 3) Stratul de ieșire, acesta are rol de a procesa datele înainte de a ieși din rețea, acest strat are în componență sa o funcție de activare, de exemplu funcția Sigmoid (folosită în rețelele cu scop de generare a unei probabilități).

II.3 TIPURI DE REȚELE DE NEURONI

1) FeedForward Neural Network - rețeaua simplă de neuroni

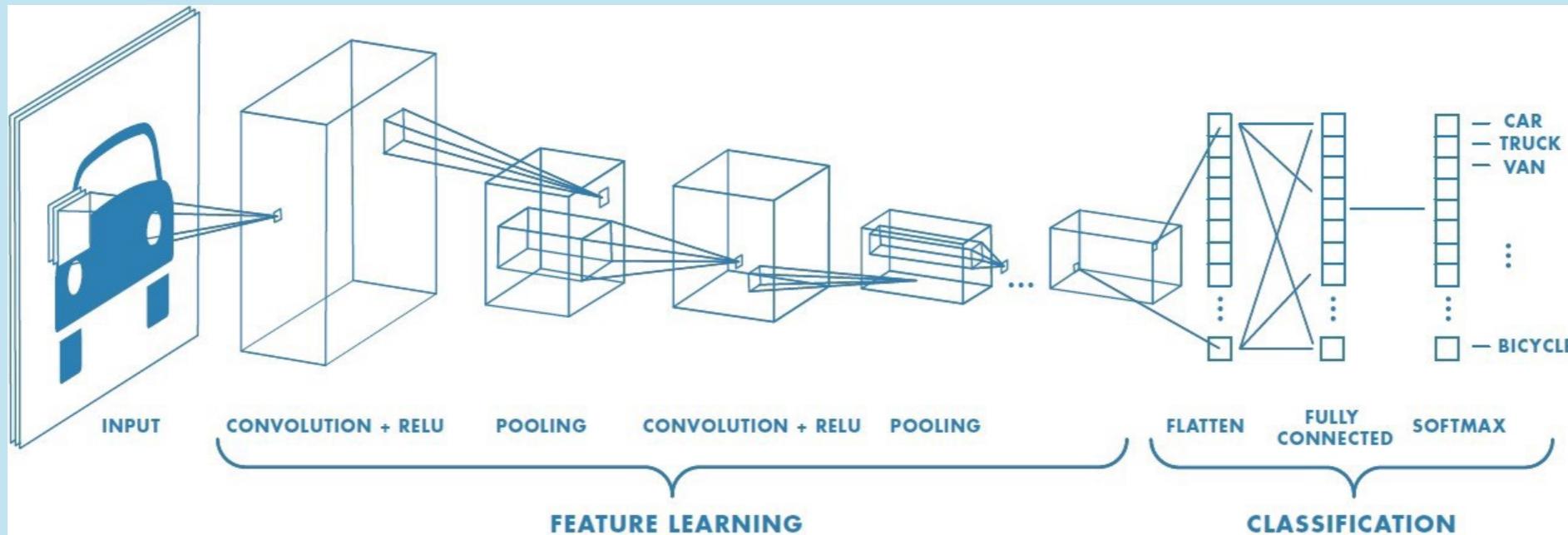
Aceasta este cea mai simplă rețea neurală de implementat, este alcătuită din 3 straturi (Intrare-Ascuns-leșire). Această rețea are aplicații în probleme de statistică, precum evaluarea prețurilor, generarea ratelor de natalitate dintr-un recensământ sau generarea probabilității de a face atac de cord în funcție de un număr mare de factori.



2) Rețea Convoluțională de neuroni

Rețeaua simplă de neuroni e foarte bună pe procesarea datelor din statistici, este eficientă din punct de vedere a timpului și al spațiului de stocare. Problema acestei rețele de neuroni apare când e nevoie să fie procesate și alte tipuri de date precum cele grafice, în acest caz fiind foarte ineficientă și inprecisă. Această problemă apare deoarece rețeaua simplă de neuroni efectuează operații pe vectori, iar datele grafice sunt sub formă de matrice.

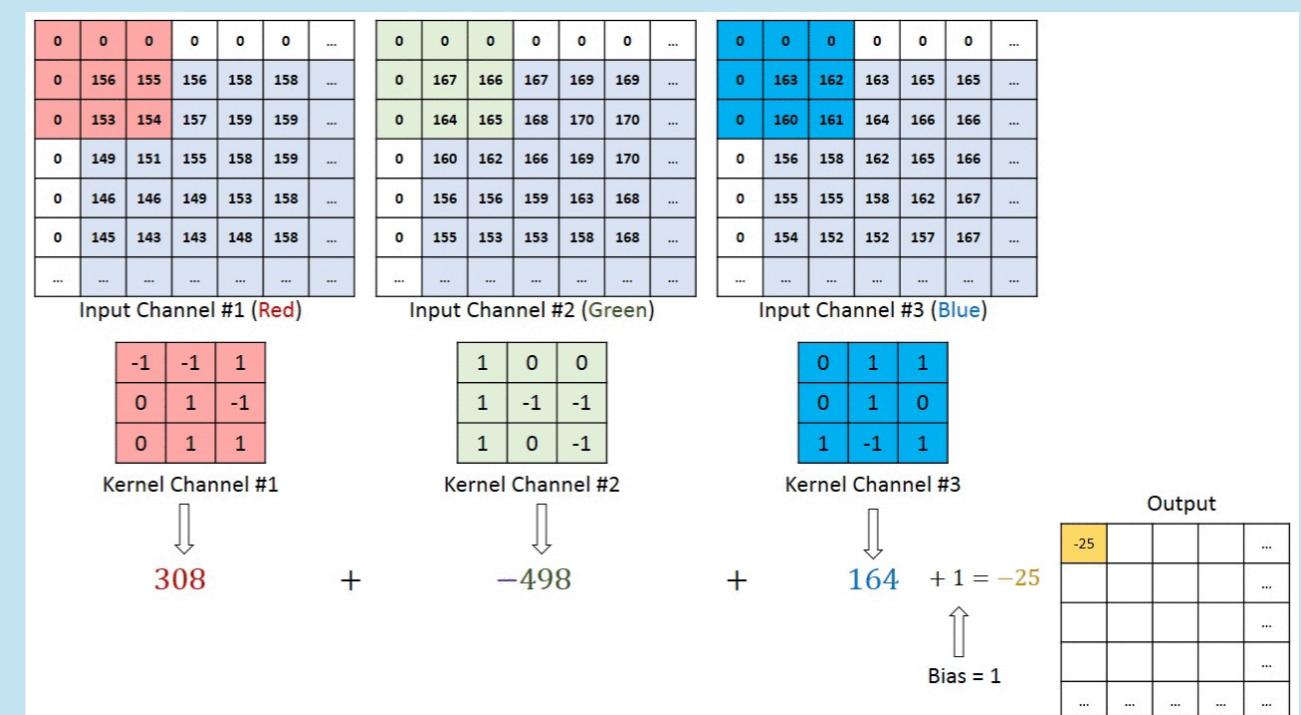
Pentru a rezolva această problemă au fost introduse 3 tipuri noi de straturi:Stratul Convoluțional,Stratul Pooling și stratul de Aplatizare.



1)Stratul convoluțional - Acest strat are rol de a procesa datele unei imagini, spre deosebire de un strat clasic de neuroni acesta efectuează operații pe matrici.Scopul său este de a prelua detaliile importante alei unei imagini, pentru asta este suprapusă o Matrice Nucleu peste imagini, această are dimensiunii mai mici decât imaginea,și este pătratică,pentru a extrage informații aceasta are un filtru(parametrii matricei)

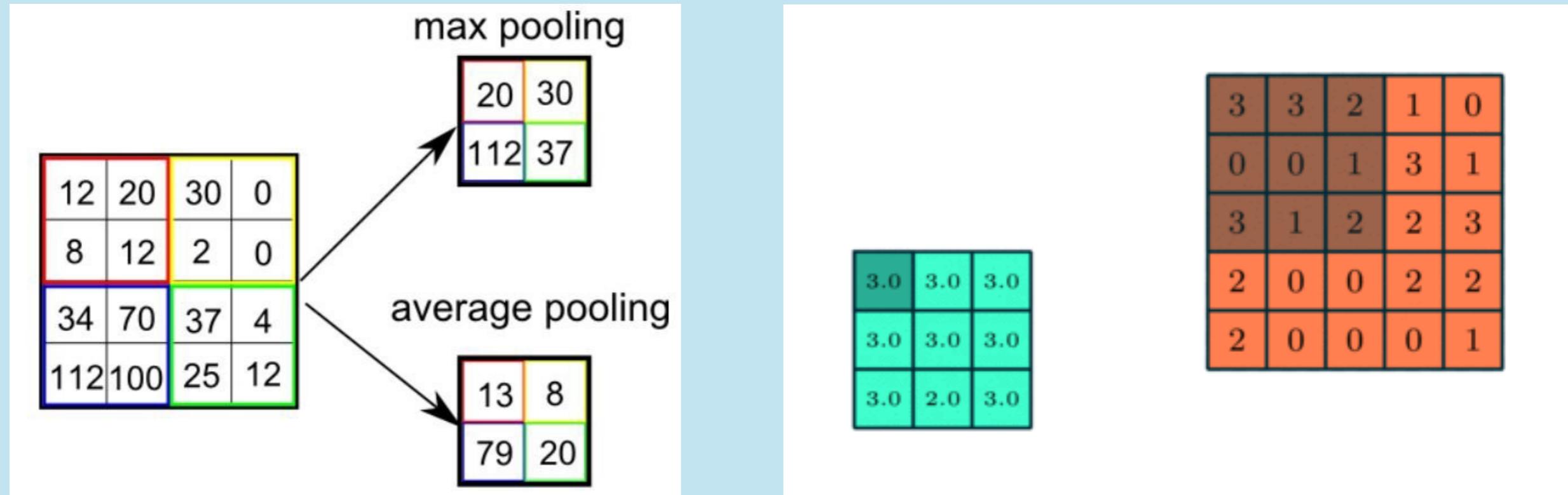
$$\sum_n a_{i,j} * W_{i,j}$$

a->valorile pixelilor matricei imagine
b->valorile filtrului de pe matricea nucleu



2)Stratul pentru “Pooling” -Acest strat are rol în a reduce mărimea matricei conveționale precedente, păstrând doar caracteristicile importante, pentru a putea crește performanța rețelei.Pentru generarea matricei reduse se folosesc funcții matematice de statistică precum funcția $\max(x)$ sau $\text{avg}(x)$.

$$\max(b_{i,j}) \downarrow$$



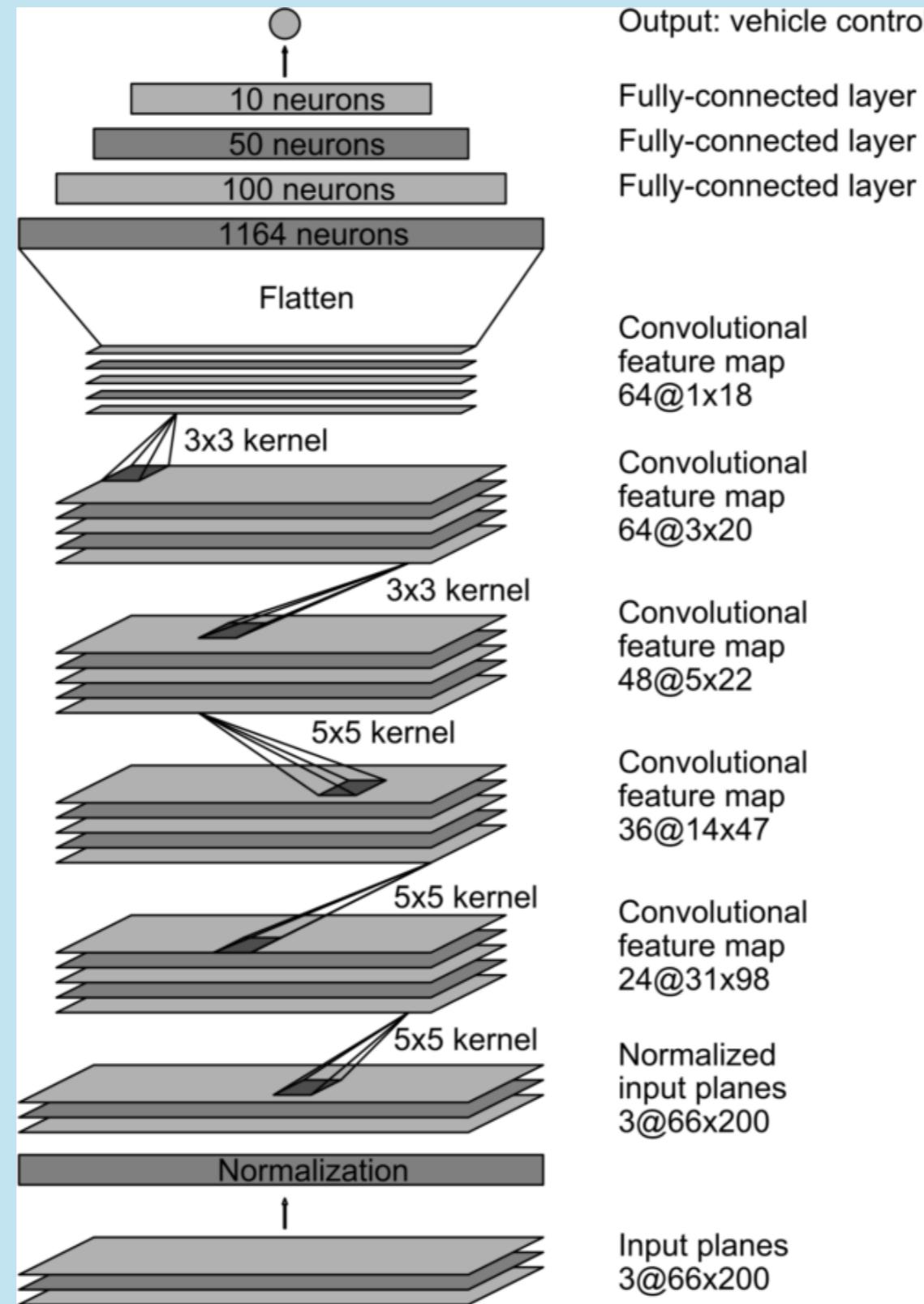
3)Stratul de Aplatizare după ce imaginile au fost procesate în straturile conveționale datele trebuie trimise unei rețele simple de neuroni, cum aceasta nu poate procesa matrici, stratul de aplatizare are rolul de a transforma o matrice $M(n \times n)$ într-un vector de lungime n^2

Architectura unei rețele conveționale

În funcție de problema pe care trebuie să o rezolve, o rețea convețională poate fi modelată după un anumit model.

1)Modelul LeNet este cel mai simplu model de rețea convețională și poate fi folosit în aplicații simple precum identificarea semnelor de circulație.Acest model este prezentat în prima imagine cu rețeaua convețională.

2) Modelul NvidiaNet acesta este un model mai avansat de rețea convoluțională. A fost dezvoltat de către Nvidia și este folosit în principal pentru a implementa sisteme de pilot automat autovehiculelor. Din acest motiv am ales modelul lor arhitectural pentru a implementa rețeau neurală.



II.4 REALIZAREA CALCULELOR ÎNTR-O REȚEA DE NEURONI

În domeniul Inteligenței Artificiale rețelele de neuroni sunt o aplicație a invățării supervizate, adică au nevoie să de un evaluator, pentru a-și putea crește eficiența, cel mai simplu acest lucru se face prin folosirea unui set de date pentru antrenare, deja clasificat care îi este dat rețelei după care rezultatul produs de rețea este comparat cu rezultatul actual. În acest capitol vom discuta care este drumul datelor de când intră în rețea și până ies din aceasta. Vom înțelege cum se realizează calculele într-o rețea și cum o unitate simplă ca un Perceptron, într-un număr mare poate realiza chestii complexe.

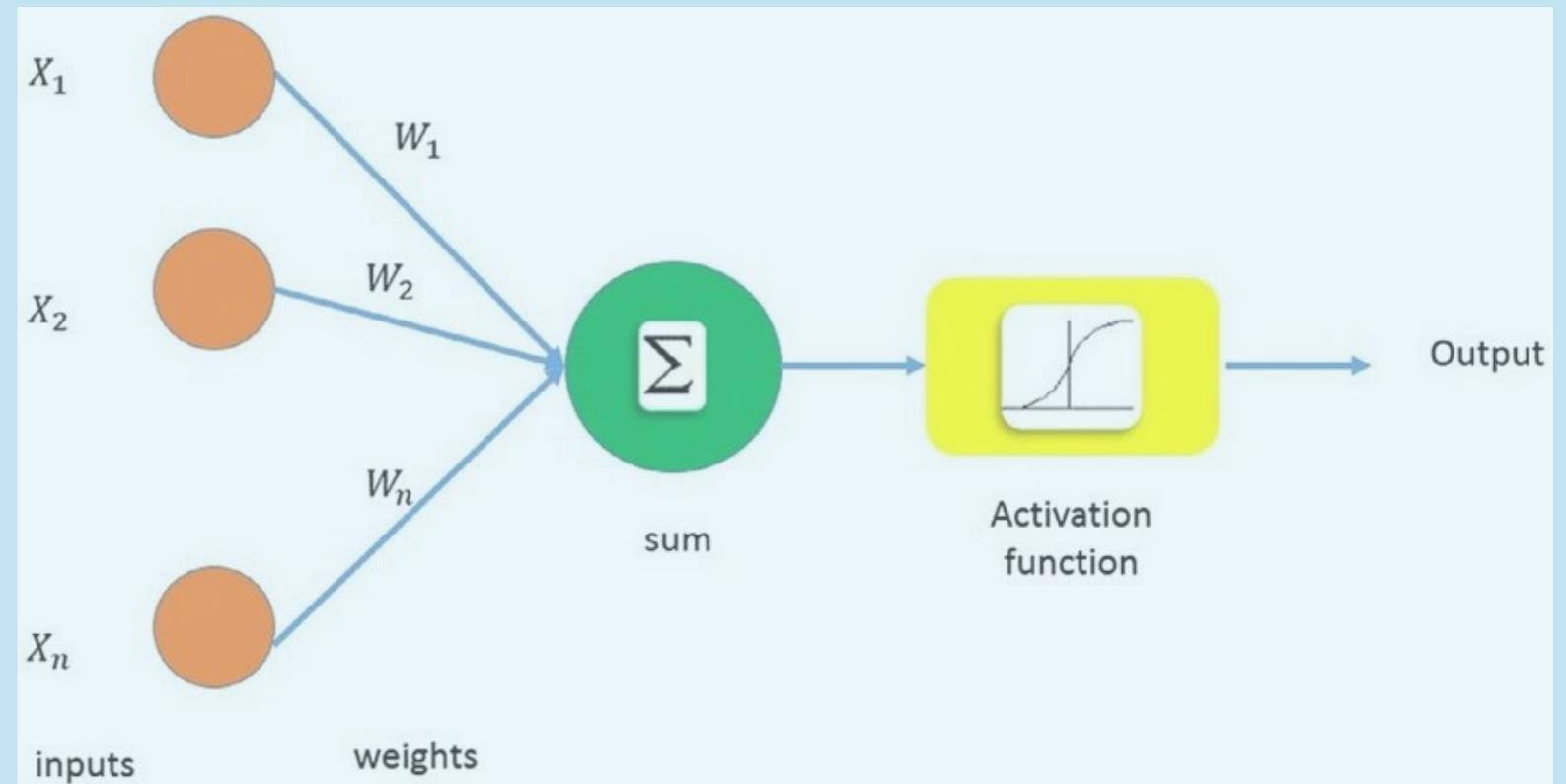
Algoritmul Feed-Foward stă la baza acestui proces.

- 1) Datele sunt preluate din nodurile de intrare
- 2) Se realizează calcule pe fiecare nod de pe substratul curent.
- 3) Datele trec printr-o funcție de activare
- 4) Nodurile curente devin noduri de intrare și se avansează pe substratul următor
- 5) Dacă nu ne aflăm pe ultimul substrat se reia pasul 1.
- 6) Datele sunt scoase din rețea

Calcul valorilor intr-un Perceptron(pasul 2)

Pentru a calcula valoarea unui nod(Perceptron) se realizează suma produsului dintre valoarea nodurilor de intrare vecine nodului curent și costul drumului.Acest rezultat după devine parametru al unei funcții de activare, rezultatul acestei funcții este valoarea nodului.

$$\sum_i^n X_i * W_i$$

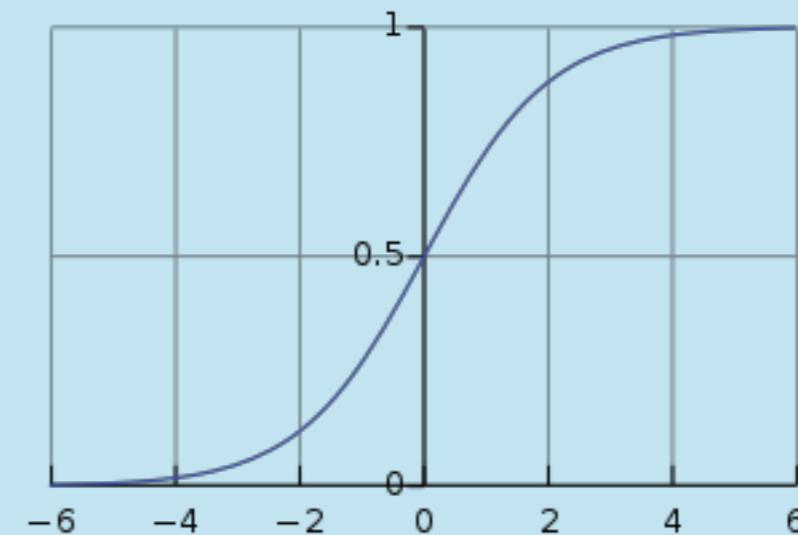


Funcții de activare:

1) Funcția sigmoid

$$f(x) = \frac{1}{1 + e^{-x}}$$

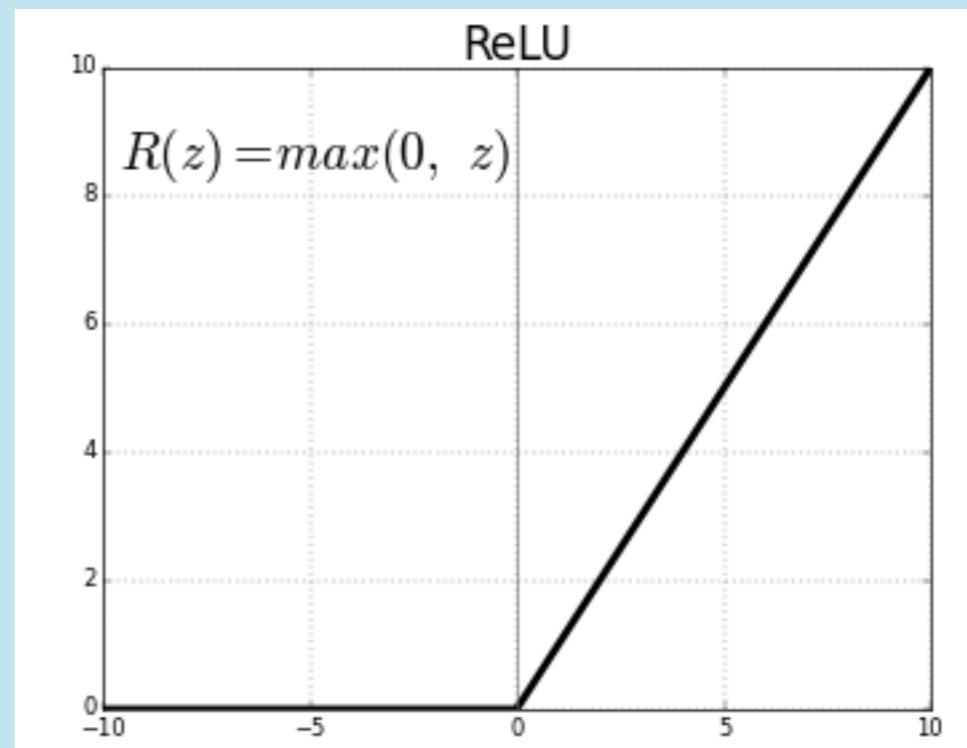
Această funcție este folosită când vrem să obținem probabilități.



2) Funcția ReLU

$$R(x) = \max(0, x)$$

Această funcție se folosește când dorim să nu avem rezultate negative, spre exemplu la evaluarea costului unui produs, nu putem avea prețuri negative.



3) Funcția SoftMax

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^k e^{z_j}}$$

Această funcție este folosită în stratul de ieșire pentru a produce rezultate bazate pe procent, această funcție se aplică pe tot vectorul(stratul final) pentru ca rezultatele să fie legate între ele(Nu putem spune că un obiect este 70% minge și 100% cub).

II.5 REDUCEREA ERORILOR

O rețea de neuroni neantrenată va genera erori. Rețeaua trebuie să fie capabilă să se optimizeze, pentru acest lucru este nevoie de a aplica noțiuni de analiză matematică (derivata funcției) pentru a optimiza rețeaua.

Generarea unei erori. Pentru calcularea unei erori se folosește funcția "Cross-Entropy", o funcție cu rol în statistică care poate fi folosită pentru a genera o eroare. Aceasta se bazează pe rezultatul rețelei de neuroni și rezultatul corect.

$$H_{y'}(y) := - \sum_i (y'_i \log(y_i) + (1 - y'_i) \log(1 - y_i))$$

Pentru a reduce erorile trebuie să fie optimizate costurile drumurilor dintre noduri, coborârea în gradient este un algoritm ce dorește să minimizeze eroarea unei rețele apelând la derivata de ordin I a funcției ce generează eroarea.

- 1) Se derivează funcția
- 2) Se dă o valoare aleatorie
- 3) Dacă rezultatul este negativ, variabila trebuie crescută
- 4) Dacă rezultatul e pozitiv, variabila trebuie scăzută
- 5) Dacă rezultatul e 0 s-a găsit un minim local.

Se repetă de mai multe ori algoritmul până a găsi minimum global.

```
def calculate_error(line_parameters, points, y):  
    m = points.shape[0]  
    p = sigmoid(points*line_parameters)  
    cross_entropy= -(1/m)*(np.log(p).T * y + np.log(1-p).T * (1-y))  
    return cross_entropy
```

-calculul erorii pe un perceptron-

```
def gradient_descent(line_parameters, points, y, alpha):  
    for i in range(2000):  
        m = points.shape[0]  
        p = sigmoid(points*line_parameters)  
        gradient = points.T * (p - y)*(alpha/m)  
        line_parameters = line_parameters - gradient  
        w1 = line_parameters.item(0)  
        w2 = line_parameters.item(1)  
        b = line_parameters.item(2)  
        x1 = np.array([points[:,0].min(),points[:, 0].max() ] )  
        x2 = -b / w2 +x1 * (-w1/w2)  
        draw(x1,x2)
```

-cobișirea în gradient pe un perceptron-

II.6 OPTIMIZAREA REȚELEI

Pentru a optimiza o rețea de neuroni nu este suficient să calculăm eroarea pe un singur nod. Scopul algoritmului Back-Propagation este de a optimiza toate nodurile de la final spre început. Acest algoritm poate fi considerat inversului algoritmului Foward-Propagation folosit pentru a calcula un rezultat pe rețea.

Implementarea algoritmului:

- 1) Se începe de la capătul rețelei
- 2) Se calculează eroarea
- 3) Se aplică coborârea în gradient
- 4) Se calculează noua valoare pe acel nod
- 5) Se merge pe nodul precedent(înapoi) și se folosește valoarea calculată precedent pentru a genera un rezultat
- 6) Dacă nu suntem la nodul de intrare se repetă pasul 2

Folosind acest algoritm se optimizează drumurile dintre nodurile rețelei. Acest algoritm se aplică de mai multe ori până se atinge rezultatul dorit.

Optimizarea în timp real al unui perceptron.

Un perceptron descrie funcția liniară(de grad I) și se poate folosi pentru a rezolva probleme de regresie liniară(Căutarea unei funcții care știe să clasifice datele în 2 categorii).

```
import numpy as np
import matplotlib.pyplot as plt

def draw(x1, x2):
    ln = plt.plot(x1, x2)
    plt.pause(0.0001)
    ln[0].remove()

def sigmoid(score):
    return 1/(1+ np.exp(-score))

def calculate_error(line_parameters, points, y):
    m = points.shape[0]
    p = sigmoid(points*line_parameters)
    cross_entropy= -(1/m)*(np.log(p).T * y + np.log(1-p).T * (1-y))
    return cross_entropy

def gradient_descent(line_parameters, points, y, alpha):
    for i in range(2000):
        m = points.shape[0]
        p = sigmoid(points*line_parameters)
        gradient = points.T * (p - y)*(alpha/m)
        line_parameters = line_parameters - gradient
        w1 = line_parameters.item(0)
        w2 = line_parameters.item(1)
        b = line_parameters.item(2)
        x1 = np.array([points[:,0].min(),points[:, 0].max()])
        x2 = -b / w2 +x1 * (-w1/w2)
        draw(x1,x2)

n_pts = 100
np.random.seed(0)
b = np.ones(n_pts)
top_region = np.array([np.random.normal(10, 2, n_pts), np.random.normal(12, 2, n_pts), b]).transpose()
bottom_region = np.array([np.random.normal(5, 2, n_pts), np.random.normal(6, 2, n_pts), b]).transpose()
all_points = np.vstack((top_region, bottom_region))
line_parameters = np.matrix([np.zeros(3)]).T
# x1 = np.array([bottom_region[:,0].min(),top_region[:, 0].max()])
# x2 = -b / w2 +x1 * (-w1/w2)

y = np.array([np.zeros(n_pts), np.ones(n_pts)]).reshape(n_pts*2, 1)

_, ax = plt.subplots(figsize=(4, 4))
ax.scatter(top_region[:, 0], top_region[:, 1], color='r')
ax.scatter(bottom_region[:, 0],bottom_region[:,1], color='b')
gradient_descent(line_parameters, all_points,y, 0.06)
plt.show()

calculate_error(line_parameters, all_points, y)
```



-Evoluția unui perceptron reprezentat de graficul funcției liniare pentru a rezolva - problema de clasificare.

CAPITOLUL III: REALIZAREA ȘI IMPLEMENTAREA REȚELEI

În capitolul anterior am discutat despre ce este o rețea de neuroni și cum funcționează una, pentru acest proiect voi folosi o rețea convoluțională de neuroni după arhitectură NvidiaNet. Acest model de rețea este unul care necesită un calculator foarte puternic pentru a putea fi antrenată într-un mod eficient, de aceea rețea nu este realizată local ci pe un server oferit de Google prin intermediul GoogleColab.

Google Colab este o platformă bazată pe mediul de dezvoltare Jupyter Notebook. Jupyter Notebook este un mediu de dezvoltare, server-based ce folosește interpretatorul Python, acesta este făcut pentru dezvoltarea software în domeniul științific și al inteligenței artificiale. Spre deosebire de un IDE obișnuit, Google Colab permite executarea treptată a liniilor de cod, vizualizarea în timp real a datelor și o depanare intuitivă.

Serverul Google Colab oferă următoarele specificații tehnice:

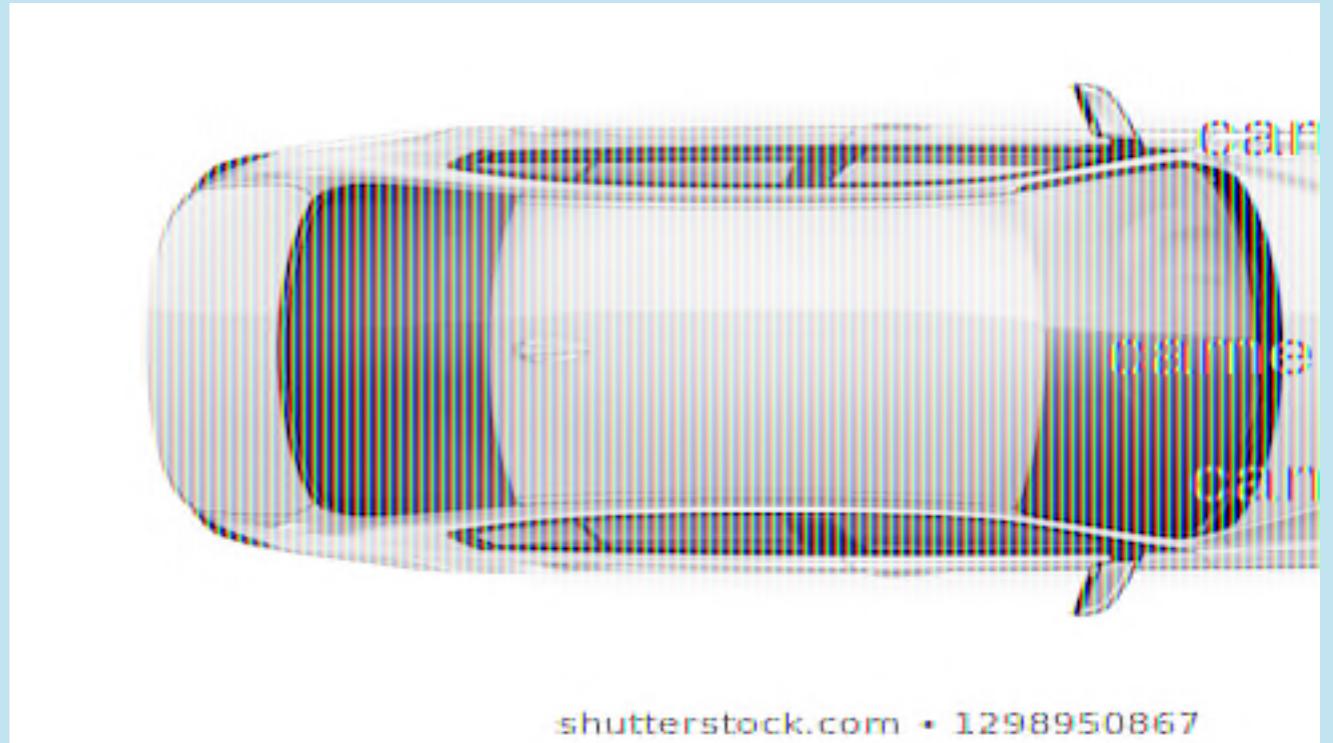
- Placă Video Tesla k80, 12 GB GDDR5 VRAM
- Procesor hyper-threaded Xeon 2.3 GHz, 45MB Cache
- 13 GB RAM
- 33 GB memorie SSD



III.2 PRELUAREA DATELOR

Datele sunt preluate de pe autovehicul folosind un sistem de 3 camere montate și orientate ca în imaginea alăturată. Un computer va citi de asemenea datele legate de accelerație, frânare și unghiul de rotație. Toate aceste date numerice sunt stocate într-un fișier CSV, iar imaginile într-un folder separat identificate prin formatul {locația-camerei}-{data-la-care-a-fost-făcută-captura}.

În cazul simulatorului acesta salvează deja datele în formatul cerut. Pentru a prelua datele se folosește modul de antrenament(Training Mode) și se conduce autovehiculul pe prima pistă. Rețeaua de neuroni folosește conceptul de "Clonare a comportamentului" pentru a controla autovehicului, în funcție de cum conduce șoferul autovehicului se va adapta și va ajunge să imite stilul de condus al șoferului. Acest tip de învățare este preluat din domeniul Psihologiei și este modul prin care copii învață să facă chestii esențiale.



shutterstock.com • 1298950867



```
[1]: datadir = "atestat"
columns = ['center','left','right','steering','throttle','reverse','speed']
data = pd.read_csv(os.path.join(datadir, 'driving_log.csv'), names = columns)
pd.set_option('display.max_colwidth',-1)
data.head()
```

	center	left	right
0	/Users/andreimoiceanu/Desktop/Data2/IMG/center_2019_01_24_19_42_33_419.jpg	/Users/andreimoiceanu/Desktop/Data2/IMG/left_2019_01_24_19_42_33_419.jpg	/Users/andreimoiceanu/Desktop/Data2/IMG/right_2019_01_24_19_42_33_419.jpg
1	/Users/andreimoiceanu/Desktop/Data2/IMG/center_2019_01_24_19_42_33_526.jpg	/Users/andreimoiceanu/Desktop/Data2/IMG/left_2019_01_24_19_42_33_526.jpg	/Users/andreimoiceanu/Desktop/Data2/IMG/right_2019_01_24_19_42_33_526.jpg
2	/Users/andreimoiceanu/Desktop/Data2/IMG/center_2019_01_24_19_42_33_631.jpg	/Users/andreimoiceanu/Desktop/Data2/IMG/left_2019_01_24_19_42_33_631.jpg	/Users/andreimoiceanu/Desktop/Data2/IMG/right_2019_01_24_19_42_33_631.jpg
3	/Users/andreimoiceanu/Desktop/Data2/IMG/center_2019_01_24_19_42_33_735.jpg	/Users/andreimoiceanu/Desktop/Data2/IMG/left_2019_01_24_19_42_33_735.jpg	/Users/andreimoiceanu/Desktop/Data2/IMG/right_2019_01_24_19_42_33_735.jpg
4	/Users/andreimoiceanu/Desktop/Data2/IMG/center_2019_01_24_19_42_33_839.jpg	/Users/andreimoiceanu/Desktop/Data2/IMG/left_2019_01_24_19_42_33_839.jpg	/Users/andreimoiceanu/Desktop/Data2/IMG/right_2019_01_24_19_42_33_839.jpg

```
[2]: def path_leaf(path):
    head, tail = ntpath.split(path)
    return tail
data['center'] = data['center'].apply(path_leaf)
data['left'] = data['left'].apply(path_leaf)
data['right'] = data['right'].apply(path_leaf)
data.head()
```

	center	left	right	steering	throttle	reverse	speed
0	center_2019_01_24_19_42_33_419.jpg	left_2019_01_24_19_42_33_419.jpg	right_2019_01_24_19_42_33_419.jpg	0.0	0.0	0.0	0.000055
1	center_2019_01_24_19_42_33_526.jpg	left_2019_01_24_19_42_33_526.jpg	right_2019_01_24_19_42_33_526.jpg	0.0	0.0	0.0	0.000078
2	center_2019_01_24_19_42_33_631.jpg	left_2019_01_24_19_42_33_631.jpg	right_2019_01_24_19_42_33_631.jpg	0.0	0.0	0.0	0.000059
3	center_2019_01_24_19_42_33_735.jpg	left_2019_01_24_19_42_33_735.jpg	right_2019_01_24_19_42_33_735.jpg	0.0	0.0	0.0	0.000096
4	center_2019_01_24_19_42_33_839.jpg	left_2019_01_24_19_42_33_839.jpg	right_2019_01_24_19_42_33_839.jpg	0.0	0.0	0.0	0.000038

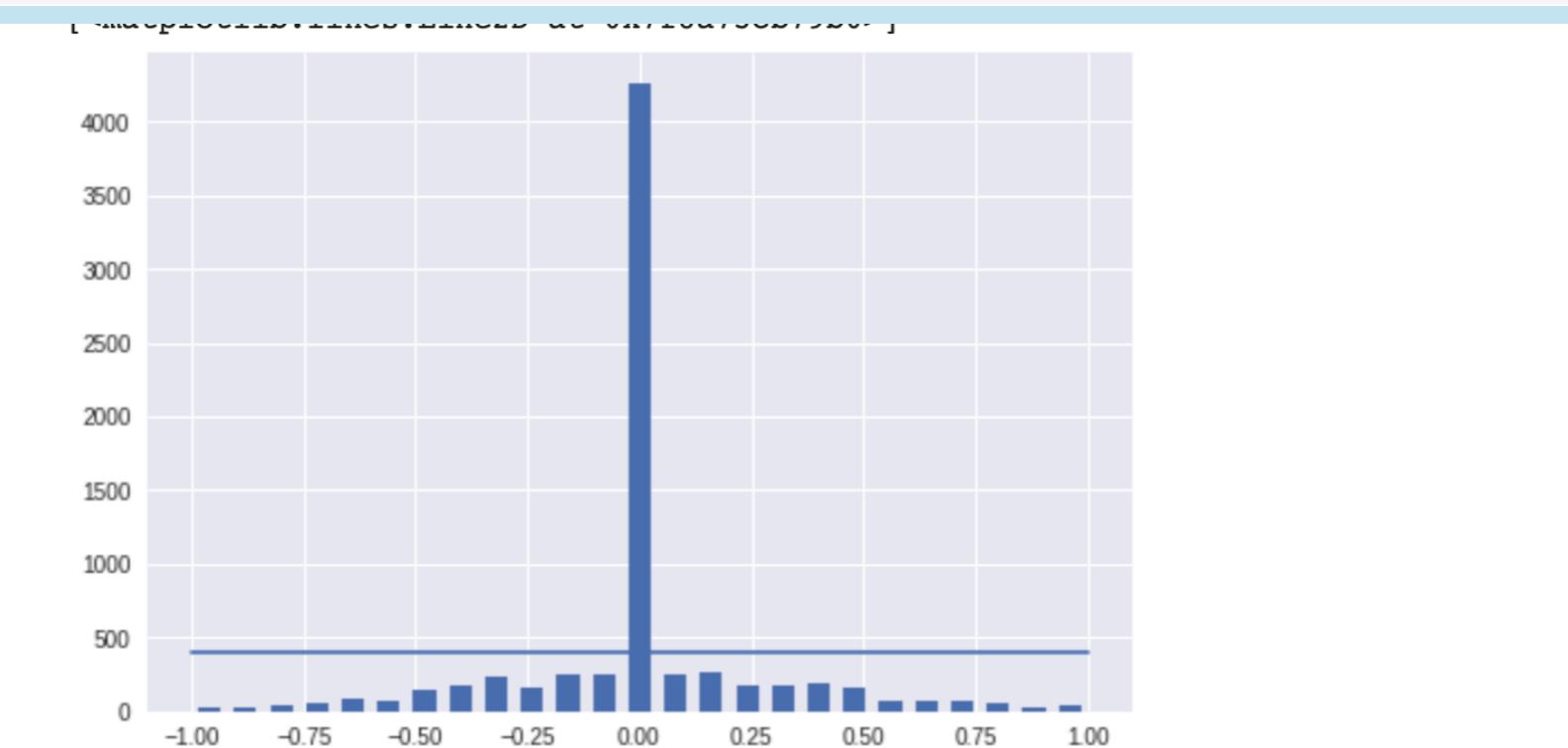
-codul pentru preluare a datelor-

În prima parte datele sunt luate din locația de unde au fost salvate, după care pentru a putea fi procesate mai ușor se elimină locația din numele fișierelor cu funcția **path_leaf(path)**.

III.3 PRELUCRAREA DATELOR

Prelucrarea datelor reprezintă cea mai amplă parte din realizarea unui model de inteligență artificială, datele trebuie modelate în aşa fel încât să poată fi procesate de rețeaua neuronală. Cum nu lucrăm cu sisteme ideale datele preluate din simulator pot aduce o optimizare slabă din cauza modului în care sunt ele prezentate. O problemă majoră este că în timpul antrenamentului în cea mai mare parte a timpului vehiculul mergea în față, prin urmare datele legate de mersul în linie dreaptă predomină setul de date și reprezintă o ponderă prea mare pentru a putea antrena corect o rețea de neuroni. Acest lucru se observă prin vizualizarea lor sub forma unei histogramme.

```
[ ] num_bins = 25
samples_per_bin = 400
hist, bins = np.histogram(data['steering'], num_bins)
center = (bins[:-1]+ bins[1:]) * 0.5
print(bins)
plt.bar(center, hist, width = 0.05)
plt.plot((np.min(data['steering']), np.max(data['steering'])),(samples_per_bin, samples_per_bin) )
```



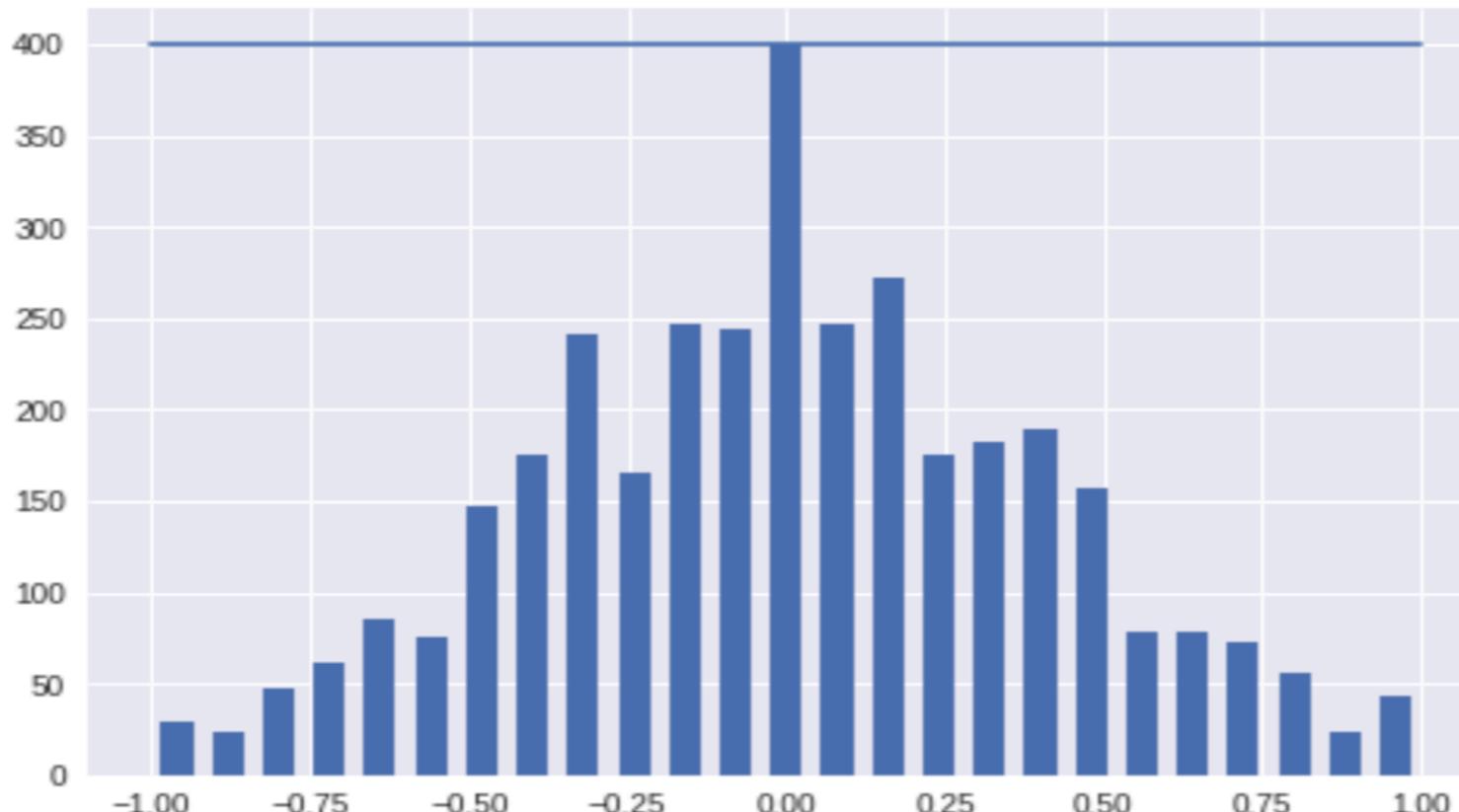
Observăm că prin eliminarea datelor ce depășesc 400 de înregistrări putem obține un set de date mult mai bun.

```
[ ] print('total data:', len(data))
remove_list = []
for j in range(num_bins):
    list_ = []
    for i in range(len(data['steering'])):
        if data['steering'][i] >= bins[j] and data['steering'][i] <= bins[j+1]:
            list_.append(i)
    list_ = shuffle(list_)
    list_ = list_[samples_per_bin:]
    remove_list.extend(list_)

print('removed:', len(remove_list))
data.drop(data.index[remove_list], inplace = True)
print('remaining:', len(data))

hist, _ = np.histogram(data['steering'], (num_bins))
plt.bar(center, hist, width = 0.05)
plt.plot((np.min(data['steering'])), np.max(data['steering'])),(samples_per_bin, samples_per_bin))
```

```
↳ total data: 7375
      removed: 3857
      remaining: 3518
      [<matplotlib.lines.Line2D at 0x7f8a73287358>]
```



Următorul pas după preluarea datelor numerice este de a corela imaginile cu interpretările numerice și de a genera un nou tabel, acest lucru se poate realiza ușor deoarece imaginile sunt legate de datele numerice prin indexul lor.

```
print(data.iloc[1])
def load_img_steering(datadir, df):
    image_path = []
    steering = []
    for i in range(len(data)):
        indexed_data = data.iloc[i]
        center, left, right = indexed_data[0], indexed_data[1], indexed_data[2]
        image_path.append(os.path.join(datadir, center.strip()))
        steering.append(float(indexed_data[3]))
        # left image append
        image_path.append(os.path.join(datadir, left.strip()))
        steering.append(float(indexed_data[3])+0.15)
        # right image append
        image_path.append(os.path.join(datadir, right.strip()))
        steering.append(float(indexed_data[3])-0.15)
    image_paths = np.asarray(image_path)
    steerings = np.asarray(steering)
    return image_paths, steerings

image_paths, steerings = load_img_steering(datadir + '/IMG', data)
```

```
center      center_2019_01_24_19_42_37_871.jpg
left        left_2019_01_24_19_42_37_871.jpg
right       right_2019_01_24_19_42_37_871.jpg
steering    -0.287784
throttle   0.580591
reverse     0
speed       18.9447
Name: 43, dtype: object
```

Pentru a antrena o rețea de neuroni datele trebuie împărțite în două seturi.

1)Un set de antrenament - de care rețeaua se poate folosi pentru a înțelege datele

2)Un set de test - prin care este evaluată performanța rețelei

Folosind blocul de cod de mai jos a fost realizat acest lucru, de asemenea datele au mai fost vizualizate încă odată pentru a fi sigur că sunt bune.

```
[ ] X_train, X_valid, y_train, y_valid = train_test_split(image_paths, steerings, test_size = 0.2, random_state = 6)
    print('Training samples: {} \n Valid samples: {}'.format(len(X_train), len(X_valid)))
```

↳ Training samples: 8443

Valid samples: 2111

```
[ ] fig, axes = plt.subplots(1, 2, figsize=(12,4))
    axes[0].hist(y_train, bins=num_bins, width=0.05,color='blue')
    axes[0].set_title('Training set')
    axes[1].hist(y_valid, bins=num_bins, width=0.05,color='red')
    axes[1].set_title('Validation set')
```

↳ Text(0.5, 1.0, 'Validation set')



Proving mai bine setul de imagini se poate observa o nouă problemă, numărul lor este prea mic.O soluție ar fi generarea de noi imagini dar asta ar însemna și generarea de noi date numerice.O soluție optimă este de a folosi librăria OpenCV pentru a realiza imagini aleatori bazate pe imaginile oferite.

Acest lucru se realizează prin 4 procese:

1)Mărirea sau micșorarea imaginii

2)Schimbarea luminozității

3)Rotatire

4)Translație

Acstea procedee sunt utilizate într-un mod aleator prin funcția **random_augment()**,pentru a nu pierde datele numerice imaginilor noi le este atribuit și valoarea unghiului de rotație atribuit imaginii originale.

Pentru a putea fi folosite de rețea neurală acestea imagini trebuie formatare după modul pe care cei de la Nvidia îl recomandă.Se observă de asemenea ca anumite detalii ale imaginii sunt irelevante pentru autovehicul aşa că ele vor fi scoase.

```
[ ] def zoom(image):
    zoom = iaa.Affine(scale=(1, 1.3))
    image = zoom.augment_image(image)
    return image

[ ] def pan(image):
    pan = iaa.Affine(translate_percent = {
        "x": (-0.1, 0.1),
        "y": (-0.1, 0.1)
    })
    image = pan.augment_image(image)
    return image

[ ] def img_random_brightness(image):
    brightness = iaa.Multiply((0.2, 1.2))
    image = brightness.augment_image(image)
    return image

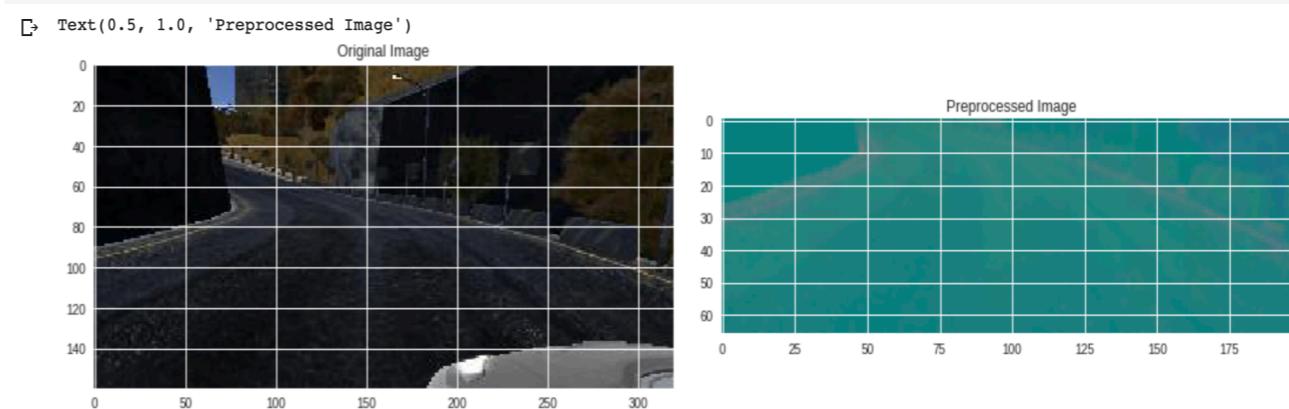
[ ] def img_random_flip(image, steering_angle):
    image = cv2.flip(image, 1)
    steering_angle = -steering_angle
    return image, steering_angle

[ ] def random_augment(image, steering_angle):
    image = mpimg.imread(image)
    if np.random.rand() < 0.5:
        image = pan(image)
    if np.random.rand() < 0.5:
        image = zoom(image)
    if np.random.rand() < 0.5:
        image = img_random_brightness(image)
    if np.random.rand() < 0.5:
        image, steering_angle = img_random_flip(image, steering_angle)
    return image, steering_angle
```

```
[ ] def img_preprocess(img):
    img = img[60:135, ::]
    img = cv2.cvtColor(img, cv2.COLOR_RGB2YUV)
    img = cv2.GaussianBlur(img, (3,3), 0)
    img = cv2.resize(img, (200,66))
    img = img/255
    return img

[ ] image = image_paths[100]
original_image = mpimg.imread(image)
preprocessed_image = img_preprocess(mpimg.imread(image))

fig, axes = plt.subplots(1,2, figsize = (15, 10))
fig.tight_layout()
axes[0].imshow(original_image)
axes[0].set_title("Original Image")
axes[1].imshow(preprocessed_image)
axes[1].set_title("Preprocessed Image")
```



O ultimă ajustare se face asupra setului de date înainte de a fi introduse în rețea de neuroni. Pentru a reduce timpul de antrenament și pentru a crește performanța rețelei setul de date este separat în subseturi de câte 100 de bucăți.

```
[ ] def batch_generator(image_paths, steering_ang, batch_size, isTraining):
    while True:
        batch_img = []
        batch_steering = []

        for i in range(batch_size):
            random_index = random.randint(0, len(image_paths) - 1)

            if isTraining:
                im, steering = random_augment(image_paths[random_index], steering_ang[random_index])
            else:
                im = mpimg.imread(image_paths[random_index])
                steering = steering_ang[random_index]

            im = img_preprocess(im)
            batch_img.append(im)
            batch_steering.append(steering)
    yield (np.asarray(batch_img), np.asarray(batch_steering))
```

III.4 MODELAREA ANALIZAREA ȘI OPTIMIZAREA REȚELEI DE NEURONI

Pentru a modela o rețea de neuroni am folosit librăria Keras, bazată pe Framework-ul Google Tensorflow, această librărie permite modelarea eficientă a unei rețele de neuroni și elimină bătaia de cap prin re-implementarea diferitelor funcții matematice. Pe lângă funcțiile prezentate la capitolul II biblioteca Keras oferă și alte funcții matematice .

```
[ ] def nvidia_model():
    model = Sequential()
    model.add(Conv2D(24, (5,5), strides = (2,2), input_shape = (66,200,3), activation='elu'))
    model.add(Conv2D(36, (5, 5), strides = (2,2), activation = 'elu'))
    model.add(Conv2D(48, (5, 5), strides = (2,2), activation = 'elu'))
    model.add(Conv2D(64, (3, 3), activation = 'elu'))
    model.add(Conv2D(64, (3, 3), activation = 'elu'))
    #   model.add(Dropout(0.5))

    model.add(Flatten())
    model.add(Dense(100, activation = 'elu'))
    # model.add(Dropout(0.5))

    model.add(Dense(50 , activation = 'elu'))
    #   model.add(Dropout(0.5))
    model.add(Dense(10, activation = 'elu'))
    model.add(Dense(8, activation = 'elu'))
    #   model.add(Dropout(0.5))
    model.add(Dense(1))
    adam = Adam(lr = 1e-4)
    model.compile(loss = 'mse', optimizer = adam)
    return model
```

După cum se poate observa, în realizarea rețelei de neuroni se păstrează arhitectura NvidiaNet.

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 31, 98, 24)	1824
conv2d_2 (Conv2D)	(None, 14, 47, 36)	21636
conv2d_3 (Conv2D)	(None, 5, 22, 48)	43248
conv2d_4 (Conv2D)	(None, 3, 20, 64)	27712
conv2d_5 (Conv2D)	(None, 1, 18, 64)	36928
flatten_1 (Flatten)	(None, 1152)	0
dense_1 (Dense)	(None, 100)	115300
dense_2 (Dense)	(None, 50)	5050
dense_3 (Dense)	(None, 10)	510
dense_4 (Dense)	(None, 8)	88
dense_5 (Dense)	(None, 1)	9

Total params: 252,305
Trainable params: 252,305
Non-trainable params: 0

None

```
[ ] l.fit_generator(batch_generator(X_train, y_train, 100,1), steps_per_epoch = 300, epochs = 10, validation_data = batch_generator(X_valid,y_valid,100,0), validation_steps = 200, verbose = 1, shuf
```

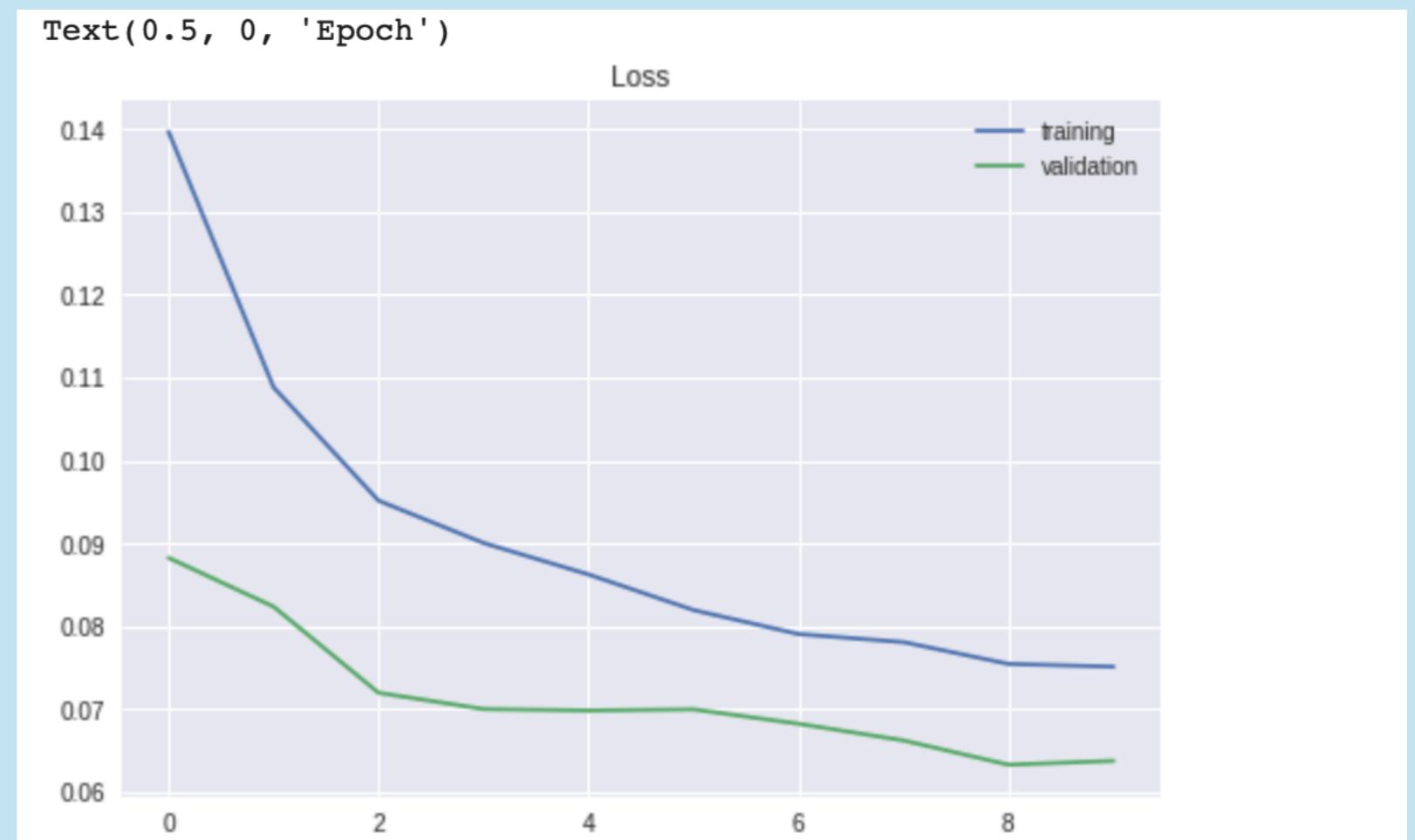
-sumarul rețelei generate-

Odată generată începe procesul de antrenament, care durează aproximativ 30 de minute.

```
Epoch 1/10
300/300 [=====] - 153s 510ms/step - loss: 0.1397 - val_loss: 0.0883
Epoch 2/10
300/300 [=====] - 146s 488ms/step - loss: 0.1088 - val_loss: 0.0824
Epoch 3/10
300/300 [=====] - 145s 483ms/step - loss: 0.0952 - val_loss: 0.0720
Epoch 4/10
300/300 [=====] - 146s 485ms/step - loss: 0.0901 - val_loss: 0.0700
Epoch 5/10
300/300 [=====] - 149s 498ms/step - loss: 0.0863 - val_loss: 0.0698
Epoch 6/10
300/300 [=====] - 150s 501ms/step - loss: 0.0820 - val_loss: 0.0700
Epoch 7/10
300/300 [=====] - 152s 508ms/step - loss: 0.0791 - val_loss: 0.0683
Epoch 8/10
300/300 [=====] - 151s 504ms/step - loss: 0.0781 - val_loss: 0.0662
Epoch 9/10
300/300 [=====] - 154s 514ms/step - loss: 0.0755 - val_loss: 0.0633
Epoch 10/10
300/300 [=====] - 154s 513ms/step - loss: 0.0751 - val_loss: 0.0638
```

O problemă mare a unei rețele de neuroni este că aceasta poate să memoreze datele, și să dea răspunsuri în funcție de ce a memorat în loc să gândească singură, acest lucru poate face rețeaua incapabilă să se adapteze la probleme noi, spre exemplu schimbarea pistei în cazul acestui proiect, acest lucru se numește Over Fitting. Pe de altă parte se poate întâmpla și fenomenul opus, în care rețeaua nu este capabilă să ofere soluții corecte, fiind prost antrenată acest fenomen se numește Under Fitting. Aceste probleme se rezolvă prin modificarea parametrilor din algoritmul generator de rețele(modificarea numărului de noduri, de

straturi sau a funcției de activare). Pentru a putea analiza performanța unei rețele se va realiza un grafic al erorii bazat pe datele oferite de setul de antrenament și cel de test. Dacă graficul setului de validare se află deasupra celui de test apare problema de Over Fitting, dacă se află sub și distanța dintre ele e prea mare și în continuă creștere apare problema de Under Fitting. După încercări repetitive modelul prezentat în acest capitol este cel final. Un grafic bun poate fi observat în graficul alăturat. Odată ce s-a ajuns la forma finală rețeaua este salvată și descărcată de pe server pentru a putea fi implementată.



IMPLEMENTAREA REȚELEI INTR-UN SISTEM SERVER-CLIENT

Odată finalizată rețeaua de neuroni aceasta trebuie doar implementată în simulator. Pentru simulator, rețeaua este văzută pur și simplu o funcție matematică, pentru un set de date de intrare este produs un rezultat. În acest caz pentru imaginile de pe camere rețeaua întoarce niște comenzi pe care le oferă autovehicului. Comunicarea dintre Simulator și rețea are loc printr-un sistem **Server-Client**, unde **Serverul** este **Calculatorul** și **Clientul** este **Simulatorul**. Serverul este scris în python și folosește biblioteca flask (în principal folosită pentru servere web). Odată deschis serverul așteaptă o cerere din partea clientului care se realizează prin intrarea în modul de autonomie, după care are loc o comunicare continuă între cele 2 sisteme până când simulatorul este oprit. Funcția **connect** așteaptă conectarea clientului (intrarea în modul autonom) iar funcția **telemetry** se ocupă de procesarea în timp real a informațiilor după care un răspuns este trimis înapoi prin funcția **send_control**. Funcția **img_preprocess** are același rol ca în rețeaua de neuroni.

```
import socketio
import eventlet
import numpy as np
from flask import Flask
from keras.models import load_model
import base64
from io import BytesIO
from PIL import Image
import cv2
sio = socketio.Server()

app = Flask(__name__) #__main__
speed_limit = 20

def img_preprocess(img):
    img = img[60:135, :, :]
    img = cv2.cvtColor(img, cv2.COLOR_RGB2YUV)
    img = cv2.GaussianBlur(img, (3,3), 0)
    img = cv2.resize(img, (200,66))
    img = img/255
    return img

@sio.on('telemetry')
def telemetry(sid, data):
    speed = float(data['speed'])
    image = Image.open(BytesIO(base64.b64decode(data['image'])))
    image = np.asarray(image)
    image = img_preprocess(image)
    image = np.array([image])
    steering_angle = float(model.predict(image))
    throttle = 1.0 - speed/speed_limit
    print("{} {} {}".format(steering_angle, throttle, speed))
    send_control(steering_angle, throttle)

@sio.on('connect')
def connect(sid, environ):
    print('Connected')
    send_control(0, 0)

def send_control(steering_angle, throttle):
    sio.emit('steer', data = {
        'steering_angle': steering_angle.__str__(),
        'throttle': throttle.__str__()
    })

if __name__ == '__main__':
    model = load_model('model.h5')
    app = socketio.Middleware(sio, app)
    eventlet.wsgi.server(eventlet.listen('0.0.0.0', 4567)),app)
```

CAPITOLUL IV: DESCRIEREA APLICAȚIEI

Obiectul principal al acestui proiect îl constituie rețeaua de neuroni. Într-o implementare reală este necesar un set de 3 camere care să fie montat pe mașină și un computer de tip RaspberryPI care să aibă control asupra mașinii. Până a fi testat pe un autovehicul real rețeaua trebuie perfectionată pe diverse simulatoare. Am ales simulatorul oferit de Udacity, o platformă de învățare online, aceștia oferă cursuri de licență pentru cei dorinci să aprofundeze în domeniul autovehiculelor autonome (contra cost) și un simulator gratuit pentru cei care doresc să își înceapă proiectele în acest domeniu.

IV.2 CERINȚE DE SISTEM

Aplicație fizică:

- 3 camere de rezoluție UHD
- Computer RaspberryPI

Aplicație virtuală (pentru rularea simulatorului):

- Computer ce rulează Windows 10/OS X/Linux
- 4 GB RAM
- Intel i3
- Nvidia GTX 750 +

IV.3 INSTAL AREA APLICAȚIEI

Pași de urmat pentru instalarea aplicației:

1)Instalați Python 2 sau 3

2)Execuți install.bash

IV.4 UTILIZAREA APLICAȚIEI

Pentru a porni aplicația execuți run.bash

Pentru modul de antrenament -> alegeți **Training Mode**,pentru a porni înregistrarea apăsați record.Conduceți autovehiculul din săgeți sau tastele **WASD**.

Pentru modul de autonomie -> alegeți **Autonomous Mode**

Aplicația dispune de 2 trasee, unul muntos și unul cu deșert.Rețeaua a fost antrenată în deșert și este capabilă să se conducă atât pe acel traseu cât și pe cel muntos pe care nu l-a văzut niciodată.

IV.5 FOLOSIREA REȚELEI NEURONALE ÎN PROIECTE PE VIITOR

Deși acest atestat este gata și este capabil de lucruri complexe, proiectul este departe de a fi terminat.

Lucruri ce doresc să le implementez pe viitor:

- I. Recunoașterea indicatoarelor și marcajelor rutiere
- II. Recunoașterea și evitarea obstacolelor
- III. Măsuri de siguranță
- IV. Implementarea într-un mediu mai complex traficul din GTA V
- V. Implementarea pe un autovehicul fizic

V. BIBLIOGRAFIE

Despre Python:

- Wikipedia
- Numpy Cheat Sheet
- MatPlot Lib Cheat Sheet

Rețele Neuronale:

- Tipuri de rețele neuronale
- Rețele Convolutionale de Neuroni
- Wikipedia - rețele de neuroni
- Wikipedia - Funcția Sigmoidă
- Wikipedia - Gradient Descent

Surse Teoretice:

- Cursuri Data-Camp
- Curs Udemy

Simulatorul Udacity

Alte proiecte personale:

- Aplicație Web cu recunoaștere facială
- Aplicație Web pentru gestionarea unui buget
- Aplicație de android Curiozități despre România