

Ministerul Educației, Culturii și Cercetării al Republicii Moldova

Universitatea Tehnică a Moldovei

Facultatea Calculatoare, Informatică și Microelectronică

Departamentul Ingineria Software și Automatică

Proiect de curs

Disciplina: Analiza și Proiectarea Algoritmilor

Tema: Elaborarea unei aplicații software destinată rezolvării unei probleme
utilizând diverși algoritmi (**Rețelele Neronale de Convoluție**)

Developing a software application to solve a problem using various algorithms
(**Convolutional neural networks**)

A efectuat:

st. gr. TI-173 Heghea Nicolae

Au verificat:

lector univ. Catruc Mariana

Andrievschi-Bagrin Veronica

Cuprins

1. Introducere.....	3
2. Formularea problemei prin abstractizarea unei situații	5
3. Definirea datelor problemei și stabilirea relației între date și soluția problemei	6
4. Descrierea algoritmilor aleși	7
4.1 Convoluția (Convolution).....	8
4.2 Normalizarea (Activation ReLu)	10
4.3 Unificarea (Pooling)	11
4.4 Straturile ascunse (Fully connected layers)	12
4.5 Instruirea (Training).....	13
4.6 Clasificarea	14
5. Implementarea algoritmilor într-un limbaj de programare structurat	16
6. Dezvoltarea unei interfețe grafice pentru aplicația software	20
7. Analiza rezultatelor obținute	21
Concluzii.....	22
Bibliografie.....	23
Anexe.....	24

1. Introducere

A venit timpul pentru inteligența artificială (AI). AI are acum puteri atât de multe în aplicații din lumea reală, de la recunoașterea facială la traducători de limbi și asistenți precum *Siri* și *Alexa*. Împreună cu aceste aplicații pentru consumatori, companiile din diferite sectoare folosesc din ce în ce mai mult puterea AI în operațiunile lor. Îmbunătățirea AI promite beneficii considerabile pentru întreprinderi și economii, prin contribuția sa la creșterea productivității și inovației. În același timp, impactul AI asupra muncii este probabil să fie profund. Unele ocupații, precum și cererea de anumite abilități vor scădea, în timp ce altele vor crește și multe se vor schimba, pe măsură ce oamenii vor lucra alături de mașini, mereu în evoluție și din ce în ce mai capabile.

Definiția cea mai acceptată a inteligenței artificiale a fost dată de *John McCarthy* în 1955: “o mașină care se comportă într-un mod care ar putea fi considerat inteligent, dacă ar fi vorba de un om”. Mai amplu, inteligența artificială vizează studiul și designul agenților inteligenți, sisteme care percep mediul înconjurător și maximizează șansele propriului succes prin comportament.

O trăsătură des întâlnită a inteligenței artificiale este că sistemul respectiv este capabil să învețe, cu sau chiar fără ajutoare externe, cu scopul de a se îmbunătăți permanent.

Problemele centrale ale cercetării în AI sunt rațiunea, cunoașterea, planificarea, procesul de învățare, prelucrarea limbajului natural (comunicare), percepția, precum și manipularea obiectelor fizice.

Rețelele neuronale artificiale (ANN) sunt sisteme de calcul inspirate de rețelele neuronale biologice din care se constituie creierul animalelor. Astfel de sisteme învață (îmbunătățesc treptat capacitatea lor) să facă sarcini luând în considerare exemplele, în general fără programare specifică sarcinilor. De exemplu, în recunoașterea imaginii, acestea

ar putea învăța să identifice imagini care conțin pisici analizând exemple de imagini care au fost marcate manual ca "pisică" sau "fără pisică" și utilizând rezultatele analitice pentru a identifica pisicile din alte imagini. Ele au găsit cele mai multe utilizări în aplicații dificil de exprimat cu un algoritm de calcul tradițional folosind programarea bazată pe reguli.

O rețea neuronală profundă (DNN) este o rețea neurală artificială (ANN) cu mai multe straturi între straturile de intrare și ieșire. DNN găsește manipularea matematică corectă pentru a transforma intrarea în ieșire, fie că este vorba de o relație liniară sau de o relație neliniară. Rețeaua trece prin straturile care calculează probabilitatea fiecărei ieșiri. De exemplu, un DNN care este instruit să recunoască rasele de câini va trece peste imaginea dată și va calcula probabilitatea ca câinele din imagine să fie de o anumită rasă.

Rețelele neuronale convoluționale (ConvNet sau CNN) reprezintă o categorie de rețele neuronale care s-au dovedit a fi foarte eficiente în domenii precum recunoașterea și clasificarea imaginilor.

2. Formularea problemei prin abstractizarea unei situații

Vizionând filme din categoria SCI-FI, bazate pe teme de AI, mi-a fost interesant cum se petrece toată această magie la nivel de cod, funcții matematice, etc.

ConvNet-urile au descoperiri majore în domeniul învățării profunde și se descurcă foarte bine pentru recunoașterea imaginilor, putem de asemenea să folosim CNN-urile pentru prelucrarea limbajului natural și analiza vorbirii. În această poveste, mă concentrez asupra viziunii pe computer (recunoașterea imaginilor).

Aplicația care am făcut-o ne va spune dacă în imagine este un câine sau o pisică.

Da, foarte simplu, dar acesta este primul pas în studierea rețelele neuronale de convoluție.

3. Definirea datelor problemei și stabilirea relației între date și soluția problemei

Când un computer vede o imagine, va vedea o serie de valori ale pixelilor. În funcție de rezoluția și mărimea imaginii, va fi afișată o serie de numere de ex. $32 \times 32 \times 3$ (3 se referă la valorile RGB). Fiecare dintre aceste numere este dat o valoare de la 0 la 255, care descrie intensitatea pixelilor la acel moment. Aceste numere, deși nu ne sunt semnificative când realizăm clasificarea imaginilor, sunt singurele intrări disponibile pentru calculator. Ideea este că dai computerului această serie de numere și va scoate numere care descriu probabilitatea ca imaginea să fie o anumită clasă (.80 pentru pisică, .20 pentru câine).

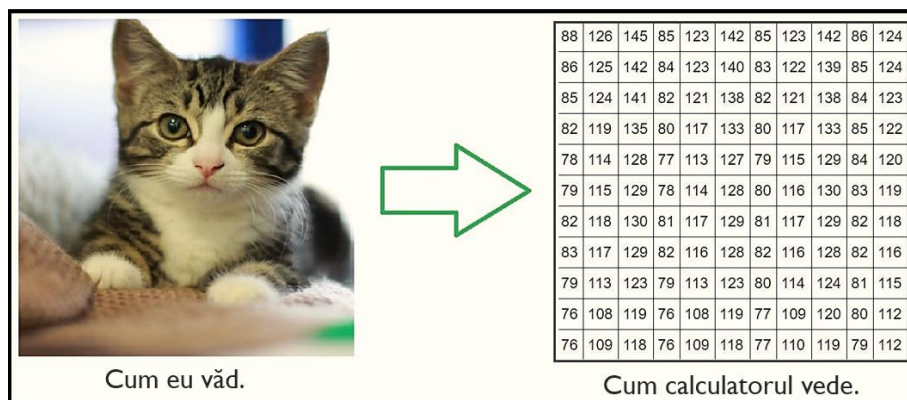


Figura 3.1 Interpretarea imaginii vizuale

Acum, că cunoaștem problema, precum și intrările și ieșirile, să ne gândim cum să abordăm acest lucru. Ceea ce vrem, e ca computerul să fie capabil să facă diferența între toate imaginile pe care le ofer și să-ți deie seama de caracteristicile unice care fac un câine un câine sau care fac o pisică o pisică. Când ne uităm la o imagine a unui câine, îl putem clasifica ca atare dacă imaginea are caracteristici identificabile, cum ar fi labele sau 4 picioare. Într-un mod similar, computerul poate efectua o clasificare a imaginilor prin căutarea unor caracteristici de nivel inferior, cum ar fi marginile și curbele, și apoi construirea unor concepte mai abstracte printr-o serie de straturi convoluționale.

4. Descrierea algoritmilor aleși

Cel mai bine de a descrie algoritmii este de a răspunde la întrebările magice *De ce ?*
Cum? Ce?

De ce CNN ?

1. Nu avem nevoie de a oferi caracteristici, CNN înțelege caracteristicile corecte de la sine. Mult mai bine cu cât merge mai profund.
2. Este ușor de studiat.
3. Cum CNN-urile înțeleg datele. (ex. Imaginile)

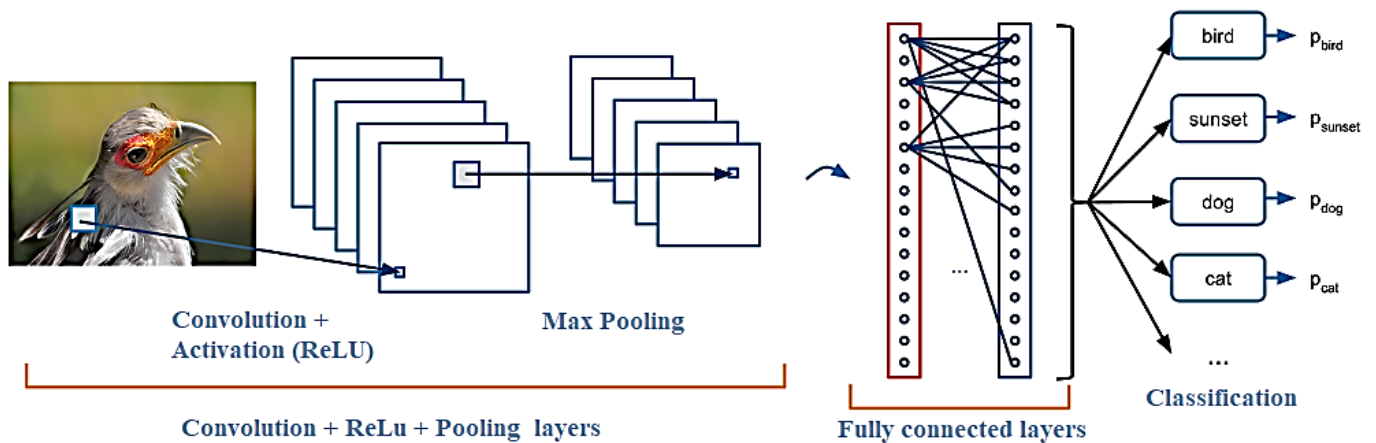


Figura 4.1 Model de arhitectură CNN

Pas cu pas, cum funcționează ConvNet?

1. Convoluția (Convolution)
2. Normalizarea (Activation ReLu)
3. Unificare (Pooling)
4. Straturile ascunse (Fully Connected)
5. Instruirea (Training)

4.1 Convoluția (Convolution)

Primul strat dintr-un CNN este întotdeauna un strat Convoluțional. După cum am menționat anterior, intrarea este o matrice de $32 \times 32 \times 3$ de valori ale pixelilor. Acum, cel mai bun mod de a explica un strat convoluțional este să vă imaginați o lanternă care luminează în partea din stânga sus a imaginii. Să spunem că lumina care luminează, acoperă o zonă de 5×5 *pixeli*. Și acum, să ne imaginăm că această lanternă alunecă în toate zonele imaginii de intrare. În termeni de învățare mecanică, această lanternă este numită filtru (sau uneori denumit neuron sau kernel), iar regiunea pe care luminează, se numește câmp receptiv. Acum, acest filtru este, de asemenea, o serie de numere (numerele se numesc potențialuri sau parametri). O notă foarte importantă este că adâncimea acestui filtru trebuie să fie la fel ca și adâncimea intrării (acest lucru face ca matematica să funcționeze), deci dimensiunile acestui filtru sunt de $5 \times 5 \times 3$. Acum, să luăm prima poziție este filtrul, de exemplu. Ar fi colțul din stânga sus. Pe măsură ce filtrul se alunecă sau se învâрте, în jurul imaginii de intrare, se înmulțește valorile din filtru cu valorile pixelilor originale ale imaginii (numite înmulțire inteligentă a elementului de calcul). Aceste multiplicări sunt toate rezumate (din punct de vedere matematic, aceasta ar fi 75 de multiplicări în total). Deci, acum aveți un singur număr. Rețineți că acest număr este doar reprezentativ când filtrul este în partea stângă sus a imaginii. Acum, repetăm acest proces pentru fiecare locație din volumul de intrare. (Pasul următor ar fi mutarea filtrului la dreapta cu 1 unitate, apoi din nou cu 1, și așa mai departe). Fiecare locație unică pe volumul de intrare produce un număr. După ce ați glisat filtrul peste toate locațiile, veți afla că ceea ce ați obținut este o serie de numere de $28 \times 28 \times 1$, pe care o numim o hartă de activare sau o hartă a funcțiilor. Motivul pentru care obțineți o matrice de 28×28 este că există 784 de locații diferite pe care un filtru de 5×5 se poate potrivi pe o imagine 32×32 de intrare.

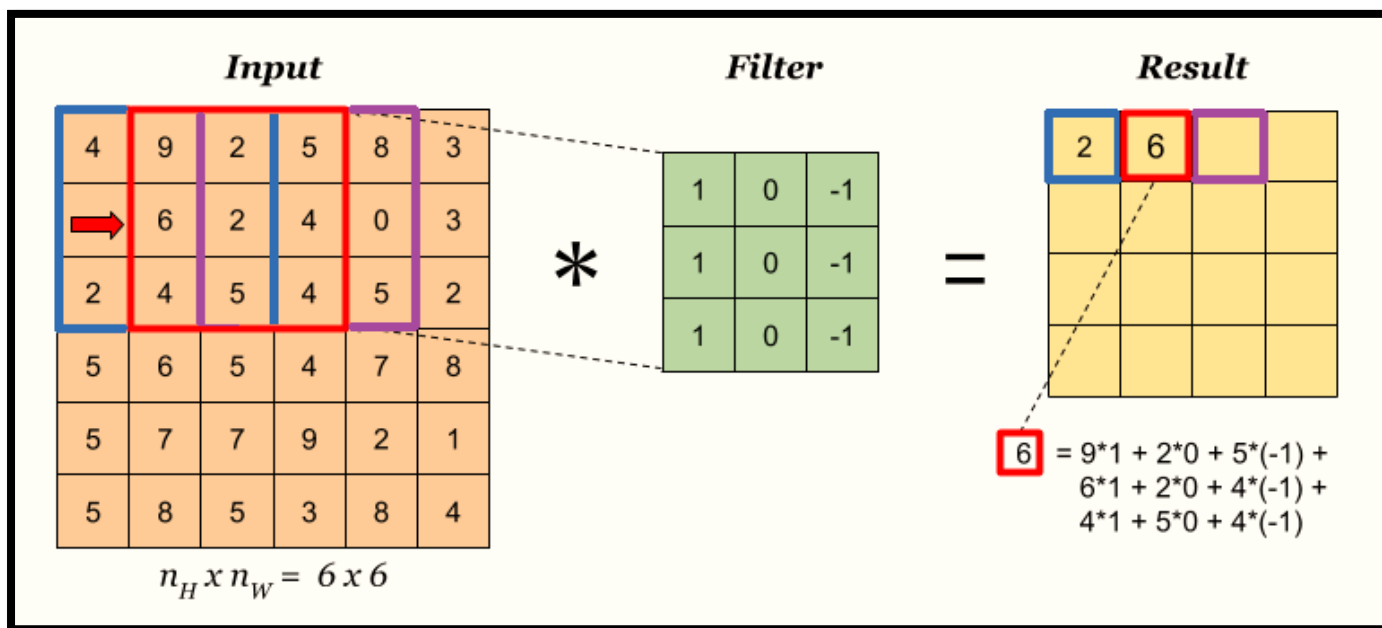


Figura 4.1.1 Convoluția

- Input – imaginile de intrare
- Filter – mai sunt numite și (*weights, kernels sau features*)
- Matricea rezultat: “Convolved Feature” sau “Feature Map”

Convoluția reprezentată matematic: $Y = W * X + b$

unde:

Y – matrice de caracteristică

W – firiltrul

X – datele de intrare

b – bias

4.2 Normalizarea (Activation ReLu)

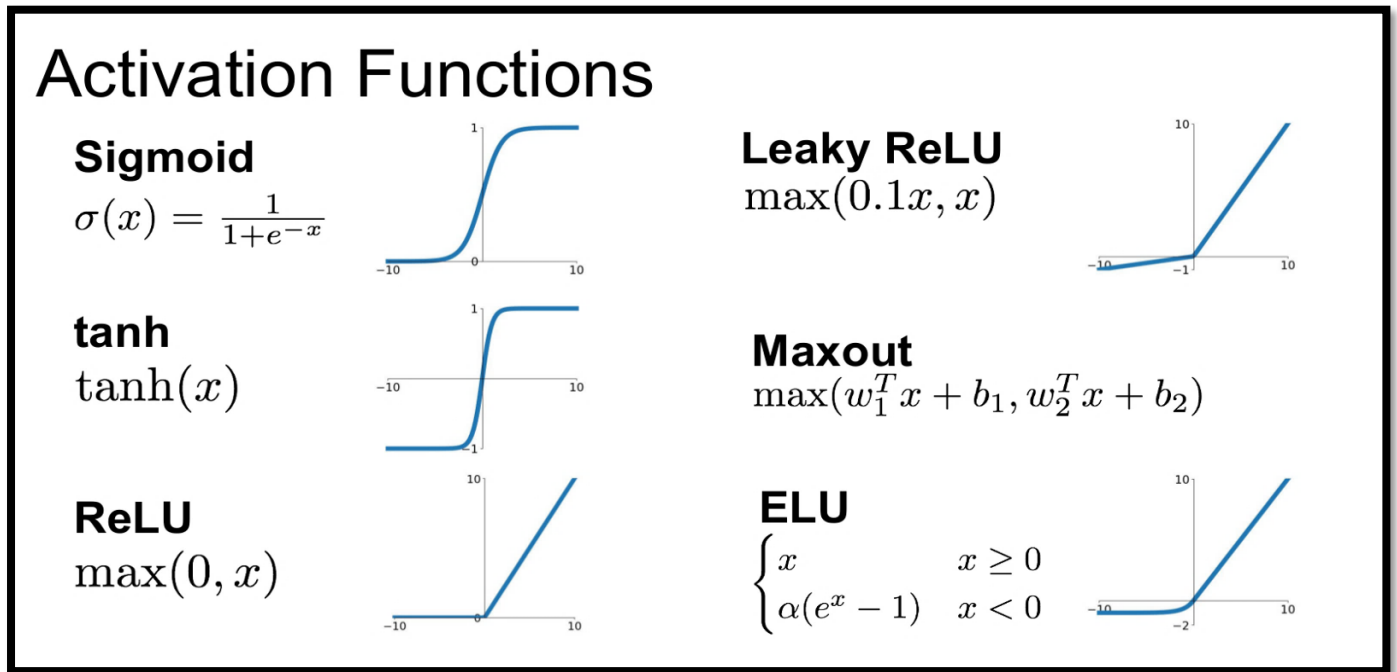


Figura 4.2.1 Diferite tipuri de funcții de activare

Funcția pe care eu am folosit-o, este ReLu.

O unitate liniară rectificată are ieșirea 0 (zero) dacă intrarea este negativă și aceeași valoare dacă este pozitivă.

Procesul de normalizare poate fi înțeles din figurile 4.2.2 , 4.2.3

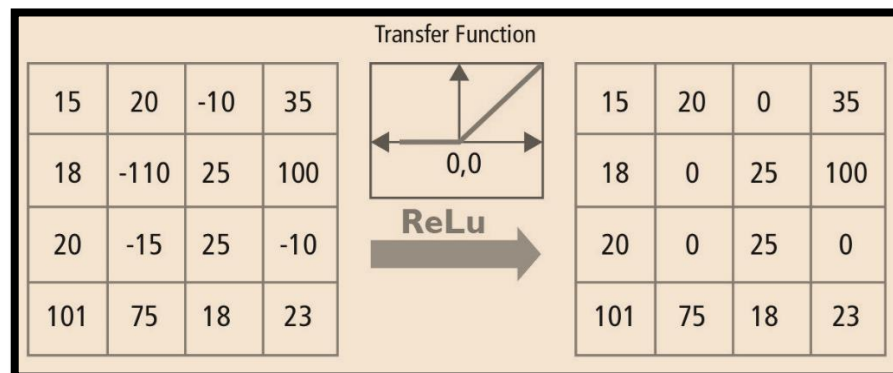


Figura 4.2.2 Transferul funcției

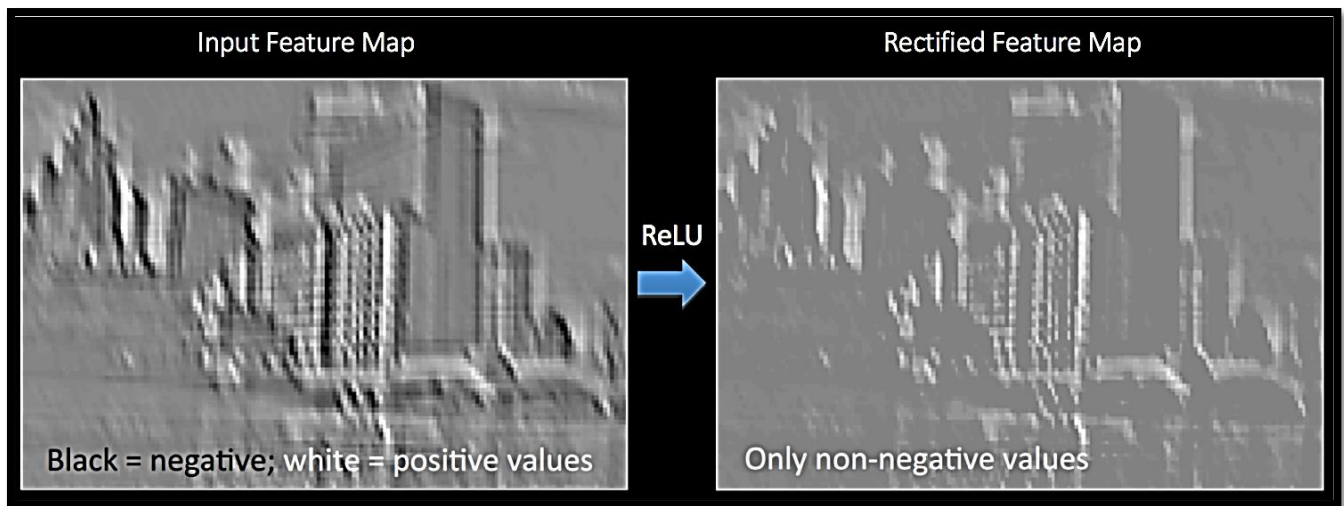


Figura 4.2.3 ReLu reprezentat ca imagine

4.3 Unificarea (Pooling)

Scopul principal al punerii în comun este reducerea mărimii unei imagini prin luarea valorilor max din fereastră.

Procesul de unificare poate fi înțeles din figurile 4.3.1 , 4.3.2

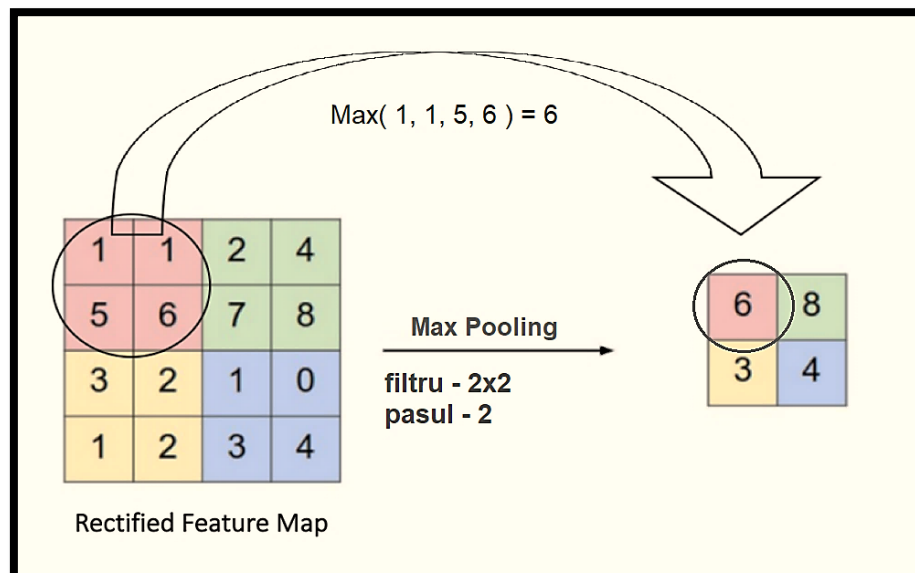


Figura 4.3.1 Unificarea

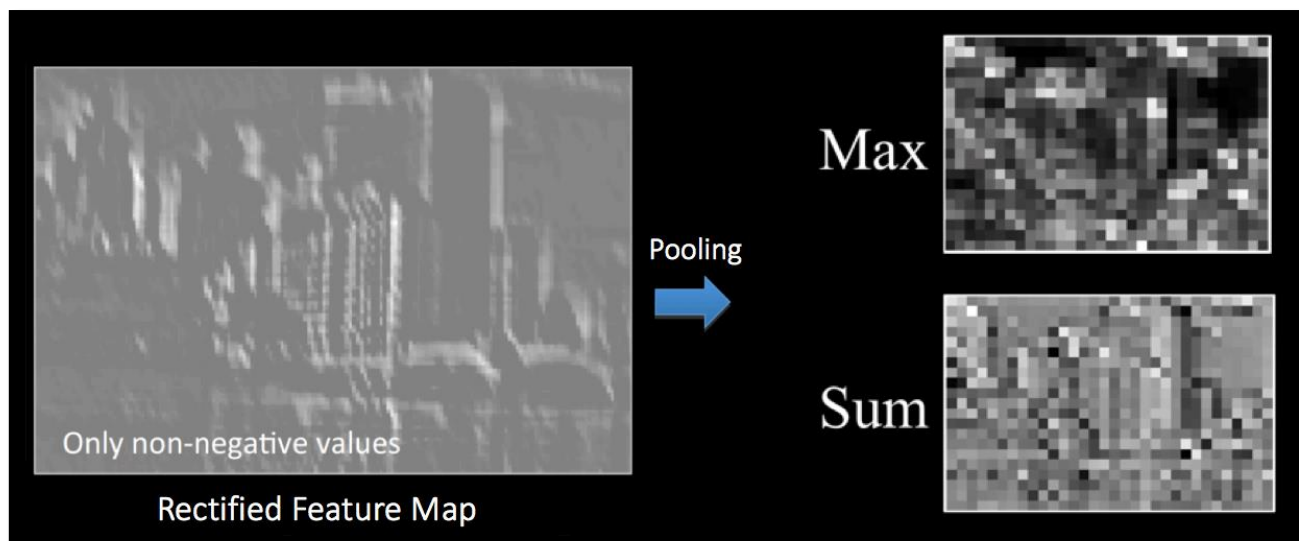


Figura 4.3.2 Uniicarea

4.4 Straturile ascunse (Fully connected layers)

Acum, că putem detecta aceste caracteristici de nivel înalt, înghețul de pe tort este atașat un strat complet conectat la sfârșitul rețelei. Acest strat are, în esență, un volum de și trimite un vector N dimensional unde N este numărul de clase din care programul trebuie să aleagă. De exemplu, dacă doriți un program de clasificare numerică, N ar fi 10 deoarece există 10 cifre. Fiecare număr din acest N vector dimensional reprezintă probabilitatea unei anumite clase. Există și alte modalități prin care puteți reprezenta rezultatul, dar vă arăt doar abordarea softmax).

Modul în care funcționează acest strat conectat complet este faptul că analizează rezultatul stratului anterior (care, după cum ne amintim, ar trebui să reprezinte hărțile de activare ale funcțiilor de nivel înalt) și determină ce caracteristici se corelează cel mai mult cu o anumită clasă.

În esență, un strat de FC privește ce caracteristici de nivel înalt se corelează cel mai mult cu o anumită clasă și are potențialuri deosebite, calculează produsele între potențialuri și stratul anterior, și obține probabilitățile corecte pentru diferitele clase.

În figura 4.4.1 , 4.4.2 puteți vedea un exemplu de FC.

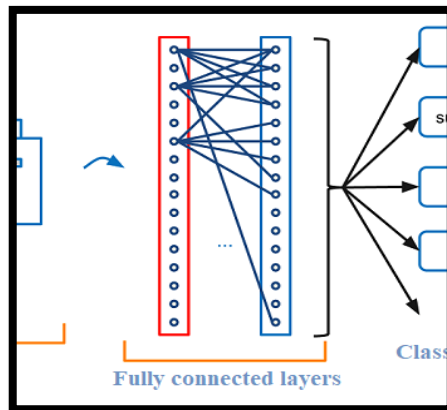


Figura 4.4.1 Stratul FC

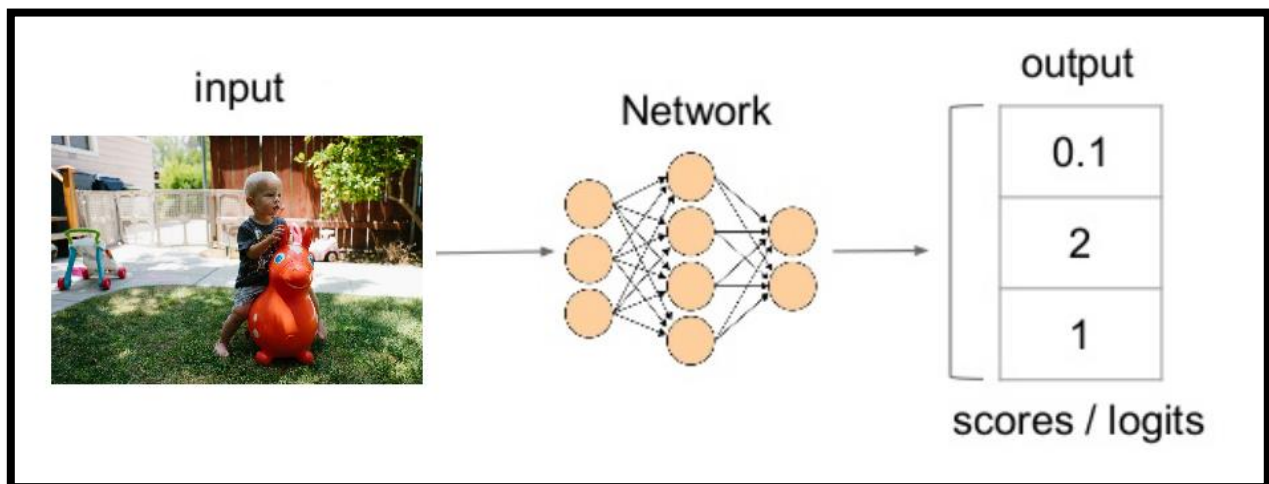


Figura 4.4.2 Rezultatul stratului FC

4.5 Instruirea (Training)

Este probabil cea mai importantă parte din CNN. S-ar putea să existe o mulțime de întrebări pe care le-ați avut în timpul lecturii. Cum știu filtrele din primul strat conv de a căuta marginile și curbele? Cum știe stratul complet conectat ce hărți de activare trebuie să se uite? Cum știu filtrele din fiecare strat ce valori trebuie să aibă?

Modul în care computerul își poate ajusta valorile filtrului este prin intermediul unui proces de antrenament denumit *backpropagation*.

Funcția de pierdere:

$$E_{total} = \sum \frac{1}{2} (actual - prevăzut)^2, (E - eroarea)$$

Actualizarea filtrelor

$$w = w_i - \eta \frac{dL}{dW}, \quad w - weights (filtru)$$

$w_i - filtrul\ inițial$

$\eta - rata\ de\ instruire$

4.6 Clasificarea

Clasificarea poate fi înțeleasă astfel, având un vector de scoruri spre clasele, v-a da o probabilitate corespunzătoare scorului. Probabilitatea are valorile între 0.0 și 1.0. Suma tuturor probabilităților este 1.0.

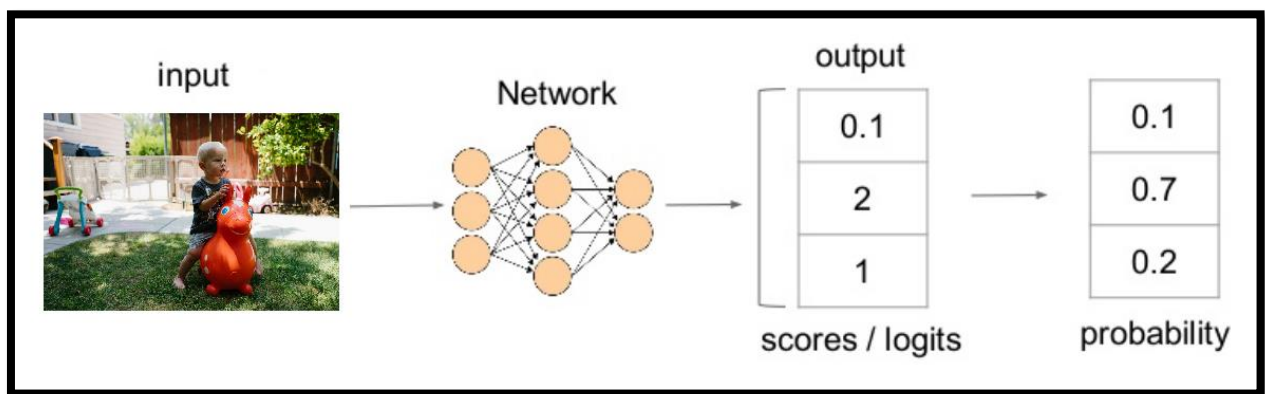


Figura 4.6.1 Clasificarea (SoftMax)

Ce este un CNN ?

O rețea neuronală de convoluție este o rețea de diferite tipuri de straturi conectate secvențial împreună.

Deci algoritmul CNN are următoarele etape

- Convoluția (Convolution)
- Normalizarea (Activation ReLu)
- Unificarea (Pooling)
- Straturile ascunse (Fully connected layers)
- Instruirea (Training)
- Clasificarea

Exemple de algoritmi CNN sunt prezentate în Figura 4.2.

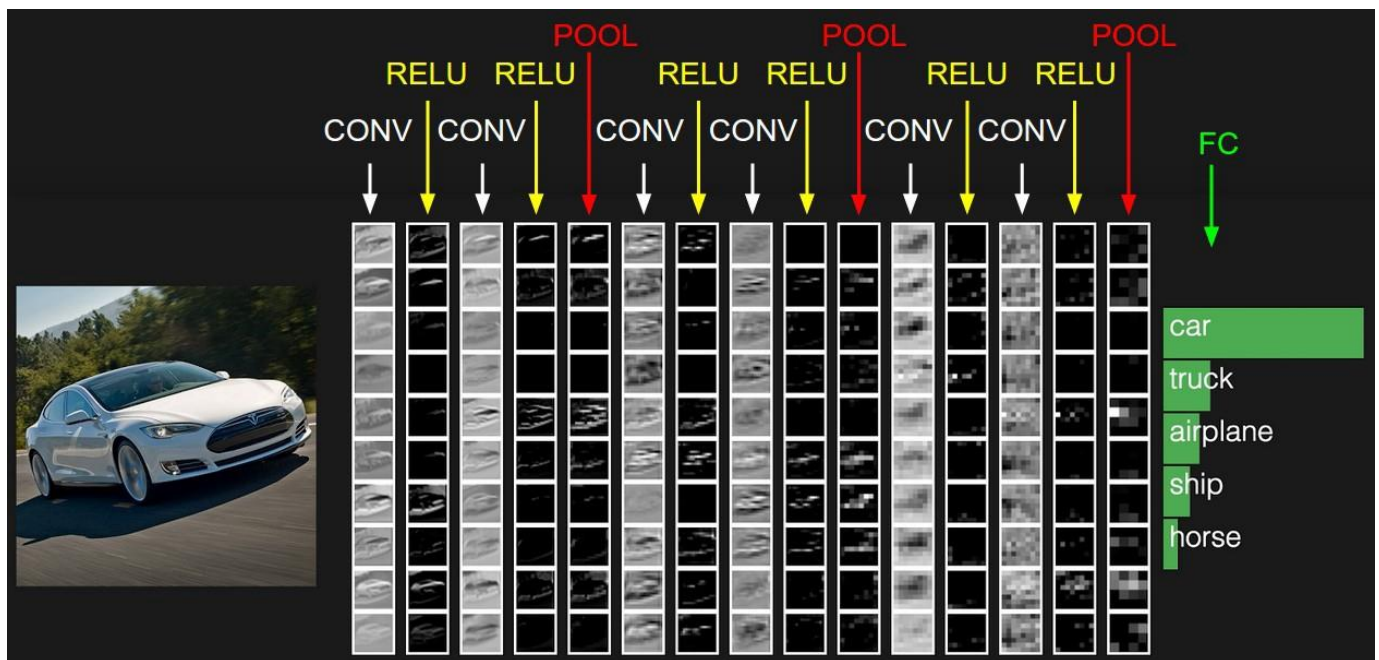


Figura 4.2 Algoritm CNN

5. Implementarea algoritmilor într-un limbaj de programare structurat

Limbajul care am ales este Python. Pentru că cu acest limbaj este ușor de început de a studia Deep Learning.

Implementarea algoritmilor întodeauna începe cu organizarea datelor. Date în programul meu sunt imagini care sunt salvate în mape, exemplu figura 5.1. Sunt aproximativ 25.000 mii de imagini.

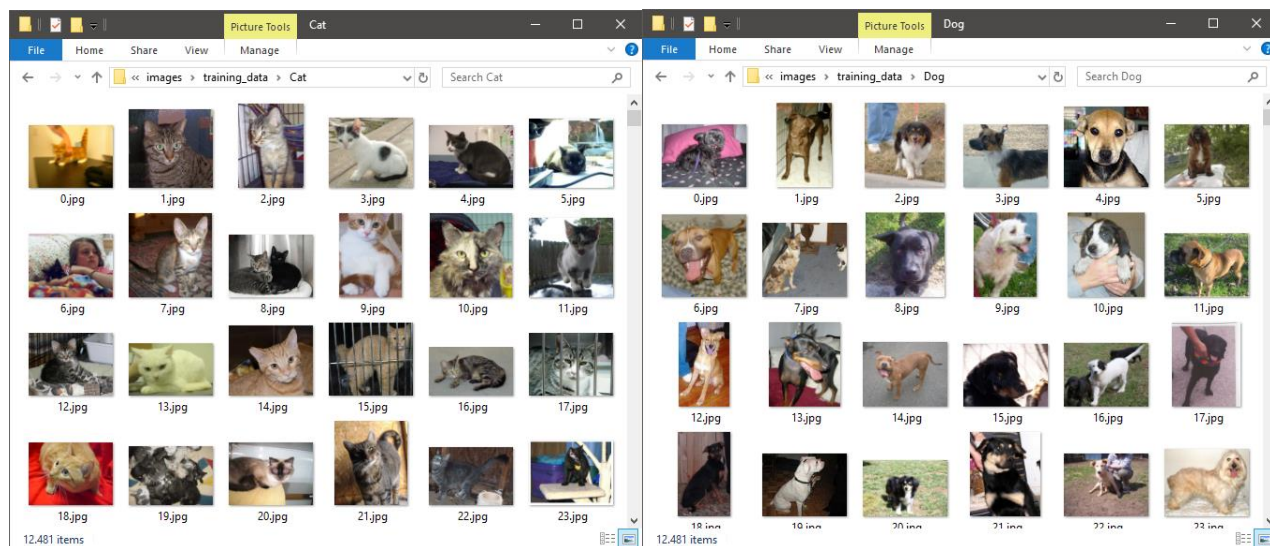


Figura 5.1 Imagini

Deci aceste date trebuie de încărcat într-o listă. Funcția care încarcă aceste imagini este `create_training_data()`,

```
def create_training_data():
    for category in CATEGORIES:
        path = os.path.join(DATADIR, category)
        class_num = CATEGORIES.index(category)
        for img in tqdm(os.listdir(path)):
            try:
                img_array = cv2.imread(os.path.join(path, img), cv2.IMREAD_GRAYSCALE)
                new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
                training_data.append([new_array, class_num])
            except Exception as e:
                pass
```


apoi toate imaginile trebuie redimensionate ca să fie de aceeași marime (eu am setat 50x50, ca să nu fie prea mari. calculator slab). Apoi salvate în fișiere aparte, caracteristicile și etichetele. Încărcarea poate fi văzută în figura 5.2, iar salvarea în *x.pickle* și *y.pickle* în figura 5.3.

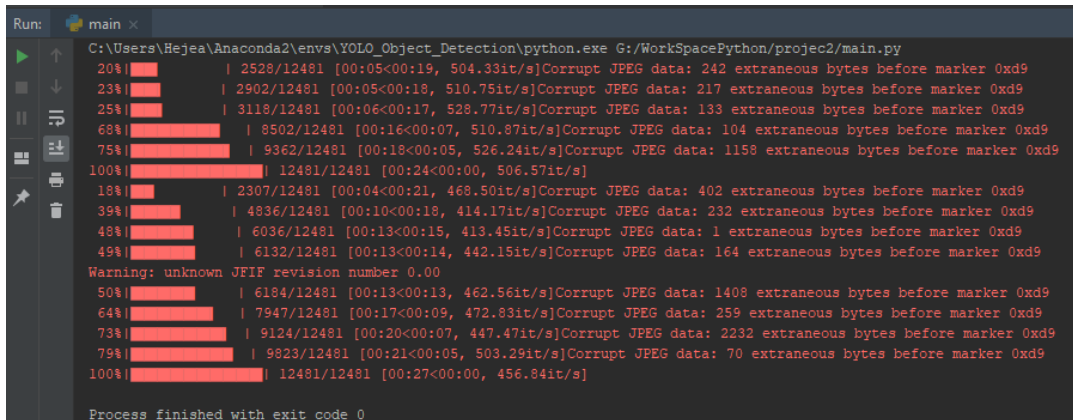


Figura 5.2 Încărcarea imaginilor

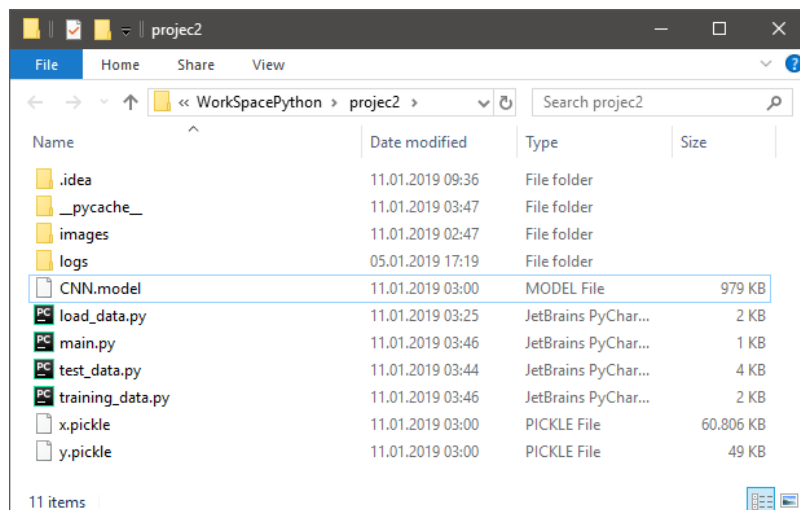


Figura 5.3 fișierele salvate

funcția de încărcare

```
def load():
    import pickle
    create_training_data()
    x = []
    y = []
```

```

for features, label in training_data:
    x.append(features)
    y.append(label)
x = np.array(x).reshape(-1, IMG_SIZE, IMG_SIZE, 1)
pickle_out = open("x.pickle", "wb")
pickle.dump(x, pickle_out)
pickle_out.close()
pickle_out = open("y.pickle", "wb")
pickle.dump(y, pickle_out)
pickle_out.close()

```

Următorul pas cu ajutorul bibliotecilor propuse de TensorFlow și Keras am creat modelu de rețea, și lam pus să se antreneze. Apoi îl voi salva întrun fișier *CNN.model*.

```

def training ():
    pickle_in = open("x.pickle", "rb")
    X = pickle.load(pickle_in)
    pickle_in = open("y.pickle", "rb")
    y = pickle.load(pickle_in)

    X = tf.keras.utils.normalize(X, axis=1)
    y = np.asarray(y)

    model = Sequential()
    model.add(ks.layers.Flatten(input_shape=X.shape[1:]))
    model.add(ks.layers.Dense(32, activation=tf.nn.relu))
    model.add(ks.layers.Dense(32, activation=tf.nn.relu))
    model.add(ks.layers.Dense(2, activation=tf.nn.softmax))

    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy']) # what to track

    model.fit(X, y, epochs=3, validation_split=0.0)
    model.save('CNN.model')

```

Proceul de antrenare poate fi văzut în figura 5.4

```
Run: main x
C:\Users\Hejea\Anaconda2\envs\YOLO_Object_Detection\python.exe G:/WorkspacePython/projec2/main.py
Epoch 1/3
2019-01-11 09:43:10.813809: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions
2019-01-11 09:43:10.818497: I tensorflow/core/common_runtime/process_util.cc:69] Creating new thread pool with d

32/24906 [.....] - ETA: 7:29 - loss: 0.6976 - acc: 0.4062
448/24906 [.....] - ETA: 34s - loss: 0.7028 - acc: 0.4933
960/24906 [>.....] - ETA: 17s - loss: 0.6979 - acc: 0.4896
1504/24906 [>.....] - ETA: 11s - loss: 0.6984 - acc: 0.4794
2016/24906 [=>.....] - ETA: 8s - loss: 0.6970 - acc: 0.4896
2496/24906 [==>.....] - ETA: 7s - loss: 0.6962 - acc: 0.4948
3008/24906 [==>.....] - ETA: 6s - loss: 0.6957 - acc: 0.4907
3520/24906 [===>.....] - ETA: 5s - loss: 0.6950 - acc: 0.4955
4032/24906 [===>.....] - ETA: 5s - loss: 0.6947 - acc: 0.4973
4544/24906 [====>.....] - ETA: 4s - loss: 0.6935 - acc: 0.5044
5056/24906 [====>.....] - ETA: 4s - loss: 0.6939 - acc: 0.5045
5568/24906 [====>.....] - ETA: 4s - loss: 0.6931 - acc: 0.5077
6080/24906 [=====>.....] - ETA: 3s - loss: 0.6933 - acc: 0.5097
6592/24906 [=====>.....] - ETA: 3s - loss: 0.6930 - acc: 0.5112
7040/24906 [=====>.....] - ETA: 3s - loss: 0.6930 - acc: 0.5105
7552/24906 [=====>.....] - ETA: 3s - loss: 0.6927 - acc: 0.5122
8064/24906 [=====>.....] - ETA: 2s - loss: 0.6930 - acc: 0.5107
8544/24906 [=====>.....] - ETA: 2s - loss: 0.6929 - acc: 0.5119
9056/24906 [=====>.....] - ETA: 2s - loss: 0.6921 - acc: 0.5153
9536/24906 [=====>.....] - ETA: 2s - loss: 0.6913 - acc: 0.5193
```

Figura 5.5 Procesul de antrenare

Și în final trebuie să testăm acest model. Testarea are loc prin încărcarea unei imagini și returnarea categoriei din care face parte ea, unutilizând modelul antrenat mai sus.

```
def nameCategory(self, img_path):
    img1 = load_img(img_path)
    img2 = prepare2gray(img1)

    prediction = self.model.predict([img2])

    category = CATEGORIES[int(prediction[0][0])]

    return category
```

6. Dezvoltarea unei interfețe grafice pentru aplicația software

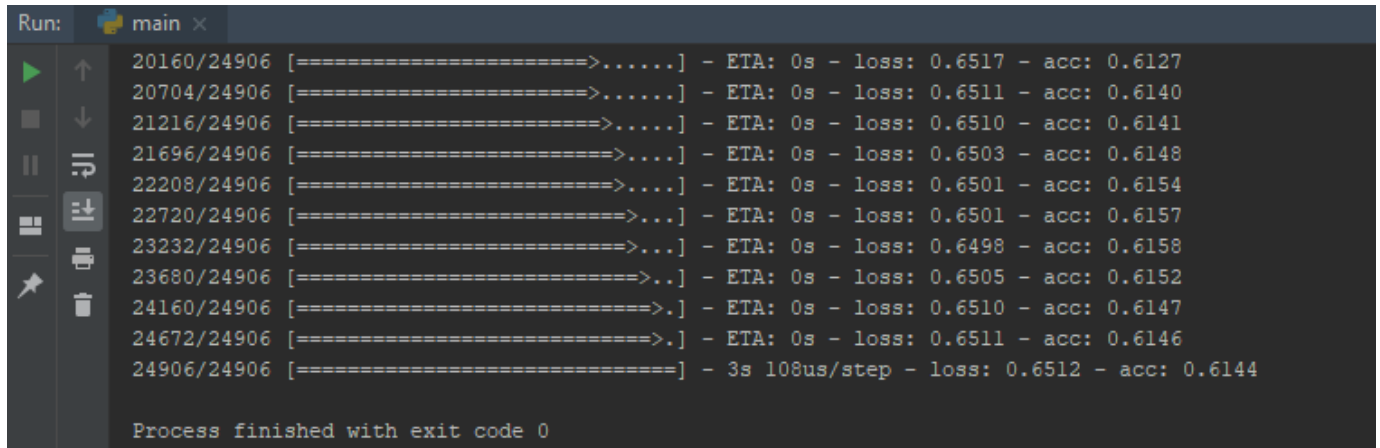
Interfața aplicației este simplă. Are un meniu din care putem selecta tipul imaginii de test (câini sau pisici). Și două butoane de traversare. Interfața aplicației este reprezentată în figura 6.1.



Figura 5.1 Interfața aplicației

7. Analiza rezultatelor obținute

Modelul nu are o exactitate înaltă din câteva motive. Prea puține date. Și caracteristicile calculatorul meu sunt prea slabe de a executa multe iterații. Dar după doar 5 iterații ne arată o acuratețe de 60%, figura 7.1.



```
Run: main x
20160/24906 [=====>.....] - ETA: 0s - loss: 0.6517 - acc: 0.6127
20704/24906 [=====>.....] - ETA: 0s - loss: 0.6511 - acc: 0.6140
21216/24906 [=====>.....] - ETA: 0s - loss: 0.6510 - acc: 0.6141
21696/24906 [=====>....] - ETA: 0s - loss: 0.6503 - acc: 0.6148
22208/24906 [=====>....] - ETA: 0s - loss: 0.6501 - acc: 0.6154
22720/24906 [=====>...] - ETA: 0s - loss: 0.6501 - acc: 0.6157
23232/24906 [=====>...] - ETA: 0s - loss: 0.6498 - acc: 0.6158
23680/24906 [=====>..] - ETA: 0s - loss: 0.6505 - acc: 0.6152
24160/24906 [=====>.] - ETA: 0s - loss: 0.6510 - acc: 0.6147
24672/24906 [=====>.] - ETA: 0s - loss: 0.6511 - acc: 0.6146
24906/24906 [=====] - 3s 108us/step - loss: 0.6512 - acc: 0.6144

Process finished with exit code 0
```

Figura 7.1

Concluzii

Proiectul de an la disciplina Analiza și Proiectarea Algoritmilor, este unul din cele mai voluminoase proiecte care le-am avut în decursul studiilor.

În acest proiect, am înteles termenii principali din spatele rețelelor neuronale convoluționale. Există multe detalii pe care le-am simplificat / s-au sarit, dar pe viitor o să oferă mult mai mult timp pentru a studia mai profund modul în care funcționează rețelele neuronale convoluționale.

Bibliografie

Siraj Raval

<https://www.youtube.com/channel/UCWN3xxRkmTPmbKwht9FuE5A>

Adit Deshpande

<https://medium.com/deep-math-machine-learning-ai/chapter-8-0-convolutional-neural-networks-for-deep-learning-364971e34ab2>

Madhu Sanjeevi (Mady)

<https://medium.com/deep-math-machine-learning-ai/chapter-8-0-convolutional-neural-networks-for-deep-learning-364971e34ab2>

Madhu Sanjeevi (Mady)

<https://medium.com/deep-math-machine-learning-ai/chapter-8-0-convolutional-neural-networks-for-deep-learning-364971e34ab2>

Google / Google Translate

<https://translate.google.com>

<https://www.google.md>

Anexe

main.py

```
import load_data
import training_data
import test_data

load_data.load()

training_data.training()

test_data.test()
```

load_data.py

```
import numpy as np
import os
import cv2
from tqdm import tqdm

DATADIR = "G:/WorkSpacePython/projec2/images/training_data/"
CATEGORIES = ["Cat", "Dog"]

IMG_SIZE = 50
training_data = []

def create_training_data():
    for category in CATEGORIES: # do dogs and cats
        path = os.path.join(DATADIR, category) # create path to dogs and cats
        class_num = CATEGORIES.index(category) # get the classification (0 or a 1).
        0=dog 1=cat

        for img in tqdm(os.listdir(path)): # iterate over each image per dogs and
cats
            try:
                img_array = cv2.imread(os.path.join(path, img), cv2.IMREAD_GRAYSCALE)
# convert to array
                new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE)) # resize to
normalize data size
                training_data.append([new_array, class_num]) # add this to our
training_data
            except Exception as e: # in the interest in keeping the output clean...
                pass

def load():
    import pickle

    create_training_data()

    x = []
    y = []
    for features, label in training_data:
```



```

        x.append(features)
        y.append(label)

x = np.array(x).reshape(-1, IMG_SIZE, IMG_SIZE, 1)

pickle_out = open("x.pickle", "wb")
pickle.dump(x, pickle_out)
pickle_out.close()

pickle_out = open("y.pickle", "wb")
pickle.dump(y, pickle_out)
pickle_out.close()

```

training_data.py

```

import pickle
import tensorflow as tf
from tensorflow.python import keras as ks
from tensorflow.python.keras.layers import Conv2D, MaxPooling2D
from tensorflow.python.keras.layers import Dense, Activation, Flatten
from tensorflow.python.keras.models import Sequential
import numpy as np

def training():
    pickle_in = open("x.pickle", "rb")
    X = pickle.load(pickle_in)

    pickle_in = open("y.pickle", "rb")
    y = pickle.load(pickle_in)

    X = tf.keras.utils.normalize(X, axis=1)
    y = np.asarray(y)

    model = Sequential()
    model.add(ks.layers.Flatten(input_shape=X.shape[1:]))
    model.add(ks.layers.Dense(32, activation=tf.nn.relu))
    model.add(ks.layers.Dense(32, activation=tf.nn.relu))
    model.add(ks.layers.Dense(2, activation=tf.nn.softmax))

    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy']) # what to track

    model.fit(X, y, epochs=5, validation_split=0.0)

    model.save('CNN.model')

```

test_data.py

```

import tensorflow as tf
import cv2
from tkinter import *
from PIL import Image, ImageTk, ImageDraw, ImageFont

CATEGORIES = ["Cat" , "Dog"]
IMG_SIZE = 50

```

```

def load_img(filepath):
    img_array = cv2.imread(filepath, cv2.IMREAD_UNCHANGED)
    return img_array

def prepare2gray(img):
    img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    img = cv2.resize(img, (IMG_SIZE, IMG_SIZE))
    return img.reshape(-1, IMG_SIZE, IMG_SIZE, 1)

def client_exit():
    exit()

class Window(Frame):
    def __init__(self, master=None):
        Frame.__init__(self, master)
        self.master = master
        self.count = 0
        self.categ = "Cat"
        self.imgPath = "images/test_data/{}/{}.jpg"
        self.model = tf.keras.models.load_model("CNN.model")
        self.init_window()
        self.showImg(0)

# Creation of init_window
def init_window(self):
    self.master.title("GUI")
    self.pack(fill=BOTH, expand=1)

    menu = Menu(self.master)
    self.master.config(menu=menu)

    file = Menu(menu)
    file.add_command(label="Exit", command=client_exit)
    menu.add_cascade(label="File", menu=file)

    chooseCateg = Menu(menu)
    chooseCateg.add_command(label="Cat", command=self.chooseCat)
    chooseCateg.add_command(label="Dog", command=self.chooseDog)
    menu.add_cascade(label="Categorie", menu=chooseCateg)

    menu.add_command(label=" < ", command=self.prevImg)
    menu.add_command(label=" > ", command=self.nextImg)

def showImg(self, c):
    path = self.imgPath.format(self.categ, c + 12480)
    load = Image.open(path)

    iw = float(load.size[0])
    ih = float(load.size[1])
    w = 500
    h = int((w * ih) / iw)
    load = load.resize((w, h), Image.ANTIALIAS)

    self.master.geometry("{}x{}".format(w + 4, h + 4))

    categ = self.nameCategory(path)

    render = ImageTk.PhotoImage(load)

```

```

        render.paste(load)

        # labels can be text or images
        img = Label(self, image=render)
        img.image = render
        self.showText(categ)
        img.place(x=0, y=0)

    def showText(self, text):
        text = Label(self, text=text)
        text.pack()
        text.place(x=0, y=0)

    def nextImg(self):
        if self.count > 18:
            self.count = 0
        else:
            self.count += 1
        self.showImg(self.count)

    def prevImg(self):
        if self.count < 1:
            self.count = 19
        else:
            self.count -= 1
        self.showImg(self.count)

    def chooseCat(self):
        self.categ = "Cat"
        self.showImg(self.count)

    def chooseDog(self):
        self.categ = "Dog"
        self.showImg(self.count)

    def nameCategory(self, img_path):
        img1 = load_img(img_path)
        img2 = prepare2gray(img1)

        prediction = self.model.predict([img2])

        category = CATEGORIES[int(prediction[0][0])]

        return category

def test():
    root = Tk()
    root.geometry("500x500")

    # creation of an instance
    app = Window(root)

    # mainloop
    root.mainloop()

```