დოქტორანტის ბიბლიოთეკა

მარიამ ჩხაიძე, ოთარ თავდიშვილი,
გიორგი ჩიჩუა, სოფიო ბარნოვი

# ხელოვნური ინტელექტი

(პრაქტიკულის მეთოდური მითითებანი)


„სტუ-ს IT კონსალტინგის სამეცნიერო ცენტრი”

საქართველოს ტექნიკური უნივერსიტეტი

მარიამ ჩხაიძე, ოთარ თავდიშვილი,
გიორგი ჩიჩუა, სოფიო ბარნოვი

სადოქტორო პროგრამა „ინფორმატიკა“

# ხელოვნური ინტელექტი

(პრაქტიკული სამუშაოს მეთოდური მითითებანი)


დამტკიცებულია:

სტუ-ს „IT კონსალტინგის
სამეცნიერო ცენტრის“ სარედაქციო
კოლეგიის მიერ ოქმი N7, 15.10.2020

თბილისი

2020

**უაკ 004.5**

განხილულია ხელოვნური ინტელექტის გამოყენების არეალები და მაგალითები. კერძოდ, ისეთი სფეროები და საკითხები, როგორიცაა ავტონომიური დაგეგმვა, თამაშების მოდელირება, ტრანსპორტის ავტონომიური მართვა, სამედიცინო დიაგნოსტიკა, სამედიცინო გამოსახულებების დამუშავება, საბანკო სექტორი, ეკონომიკური პროცესების პროგნოზირება, რისკების შეფასება, დაზღვევა, განათლების სფერო, ნევროლოგია, ფსიქოლოგია, ლინგვისტიკა, ქცევის მოდელირება, რეესტრი, გამოსახულებების ანალიზი, კრიმინალისტიკა, უსაფრთხოება და ა.შ.

წიგნში შემოთავაზებულია აღნიშნული თეორიული საკითხების შესაბამისი პრაქტიკული ამოცანების გადაწყვეტა დოქტორანტებთან ერთად პრაქტიკულ მეცადინეობებზე.

მეთოდური მითითებები რეკომენდებულია ინფორმატიკის სპეციალობის დოქტორანტებისათვის, ინფორმაციული და კომუნიკაციური ტექნოლოგიების სფეროში (ICT 0613).

**რეცენზენტები:**

პროფ. ა. ცინცაძე (სტუ)
აკად. დოქტორი გ. კივილაძე (ლიბერთი ბანკის წამყვანი პროგრამისტი)





**ISBN 978-9941-8-2866-9**

## სასწავლო კურსის მიზანი

გაუღრმავოს ცოდნა ხელოვნური ინტელექტის ცალკეული დარგების მუშაობის პრინციპების, მეთოდების, თანამედროვე მიდგომებისა და პარადიგმების შესახებ, ისეთი მიმართულებით, როგორიცაა ავტონომიური დაგეგმვა, თამაშების მოდელირება, ტრანსპორტის ავტონომიური მართვა, სამედიცინო დიაგნოსტიკა და ა.შ. ინფორმატიკის სფეროში, სხვადასხვა პრობლემების გადასაჭრელად სტუდენტს შეასწავლოს ხელოვნური ინტელექტის თანამედროვე კონცეფციებისა და ტექნოლოგიების გამოყენების საკითხები.

## საგნის შესწავლის შედეგად მიღებული ცოდნა და შეძენილი უნარები

*განსაზღვრავს* ხელოვნური ინტელექტის ცალკეული დარგების მეთოდებსა და ალგორითმებს, ასევე პროგრამული და ტექნიკური იმპლემენტაციის საკითხებს შესაბამის ბიბლიოთეკების, ფრეიმვორკების და აპლიკაციების გამოყენებით.

*აიდენტიფიცირებს* ხელოვნური ინტელექტის მეთოდების გამოყენების აქტუალურობას და აუცილებლობას თანამედროვე საინფორმაციო სისტემებში ეთიკური საკითხების გათვალისწინებით.

*არჩევს* ხელოვნური ინტელექტის დარგებისთვის ამოცანებს და მისი გადაწყვეტის გზებს შესაბამისი მეთოდების და ალგორითმების გამოყენებით

*აკეთებს* დასაბუთებულ დასკვნებს კონკრეტული ინტელექტუალური სისტემის მდგომარეობის, შესაძლო პრობლემების და მოქმედების შემდგომი ბიჯების დასახვის შესახებ, შესაბამისი ეთიკური ნორმების გათვალისწინებით

*აფორმირებს* ლიტერატურის გამოყენებით მიღებული გამოცდილების საფუძველზე, ხელოვნური ინტელექტუალური სისტემების დაგეგმვას და მუშაობის პროცესში მიმდინარე სხვადასხვა რთული პროცესების ანალიზს

*აწარმოებს* ხელოვნური ინტელექტის სფეროში კვლევის მიღებული შედეგების დასაბუთებულად და გარკვევით წარმოჩენას, საერთაშორისო სამეცნიერო საზოგადოებასთან ამ საკითხებთან დაკავშირებულ პოლემიკაში ჩართვას.

## საათების განაწილება (სტუდენტის დატვირთვა)

კრედიტების რაოდენობა 7. ლექცია -30 სთ., პრაქტიკული (კომპიუტერულ კლასში) – 30 სთ. დამოუკიდებელი მუშაობა 112 სთ.

## პრაქტიკული სამუშაოს თემები:

**1. ინტელექტი – ბუნებრივი და ხელოვნური:** ხელოვნური ინტელექტის კლასიფიკაცია, **ხელოვნური ინტელექტის განვითარების ეტაპები:** არსებული მაგალითების ვიზუალური წარმოდგენა, ხელოვნური ინტელექტის განვითარების ისტორიასთან დაკავშირებული მნიშვნელოვანი მოვლენების და პირების ბიოგრაფიების მოძიება და წარმოდგენა.

**2. ხელოვნური ინტელექტის გამოყენების არეალები და მაგალითები:** ავტონომიური დაგეგმვა, თამაშების მოდელირება, ტრანსპორტის ავტონომიური მართვა, სამედიცინო დიაგნოსტიკა, სამედიცინო გამოსახულებების დამუშავება, საბანკო სექტორი, ეკონომიკური პროცესების პროგნოზირება, რისკების შეფასება, დაზღვევა, განათლების სფერო, ნევროლოგია, ფსიქოლოგია, ლინგვისტიკა, ქცევის მოდელირება, რეესტრი, გამოსახულებების ანალიზი, კრიმინალისტიკა, უსაფრთხოება და ა.შ. - ყოველი დარგისთვის გამოყენების მკაფიო მაგალითების წარმოდგენა.

**3. ხელოვნური ინტელექტის მათემატიკური საფუძვლები:** გრაფები, სიმრავლეები, ალბათობის თეორია - პრაქტიკული სავარჯიშოების ამოხსნა.

**პრობლემათა გადაწყვეტა** - ძიების ალგორითმების ბლოკ-სქემების ჩამოყალიბება.

**4. ინტელექტუალური აგენტები.** აგენტების სტრუქტურა და პროგრამები, მარტივი რეფლექსური აგენტები, რეფლექსური აგენტები მოდელით, აგენტები მიზნობრივი ფუნქციით, სარგებლიანობაზე დაფუძნებული აგენტები, სწავლადი აგენტები, ინტელექტუალური აგენტების რეალურ გარემოში არსებული მაგალითები და მათი ჩამოყალიბება.

**5. მანქანური სწავლება და ღრმა სწავლება.** მანქანური სწავლების განსაზღვრება, სწავლება „მასწავლებლით“ (Supervised learning). სწავლება „მასწავლებლის გარეშე“ (Unsupervised Learning). მანქანური სწავლების მიდგომები. თვითსწავლება, რეგრესია, კლასტერიზაცია. თვითორგანიზებადი სისტემები, მანქანური სწავლების გამოყენების მაგალითები და კონკრეტული ამომცნობი სისტემების ჩვენება, პრაქტიკული ამოცანა მანქანურ სწავლებაში - გეომეტრიული ფიგურების ამოცნობა.

**6. ხელოვნური ნეირონული ქსელები.** ბუნებრივი ნეირონი და მისი სტრუქტურული ელემენტები, ძირითადი ფუნქციები. ნეირონული დოქტრინა და პარადიგმების ცვლა, ხელოვნური ნეირონი, მისი სტრუქტურა და თვისებები, ერთშრიანი და მრავალშრიანი ნეირონული ქსელები, კლასიფიკაცია, პერსეპტრონები. ციფრების ამოცნობის მაგალითი ნეირონული ქსელებით MatLab-ის გამოყენებით.

**7. სახეთა ამოცნობა და ბიომეტრია.** ბიომეტრიული ტექნოლოგიები, ვერიფიკაცია, იდენტიფიკაცია. ბიომეტრიული ამოცნობის მეთოდების კლასიფიკაცია. ბიომეტრიული ნიშნები (თვალი, პაპილარული ნახატი, ხმა, სახე და სხვა). ადამიანის პირისახის გამოსახულების, პაპილარული ნახატის მიღება და დამუშავების მაგალითი OpenCV-ს გამოყენებით.

**8. ცოდნა და ცოდნის წარმოდგენის საშუალებები, ექსპერტული სისტემები მონაცემები და ცოდნა, ცოდნის წარმოდგენის მოდელები.** ცოდნის პრაქტიკულად მიღების მეთოდები, ექსპერტული სისტემების დანიშნულება, სტრუქტურა, კლასიფიკაცია. ექსპერტული სისტემების მაგალითები, მარტივი ექსპერტული სისტემის შემუშავების მაგალითი Clips-ის გამოყენებით.

**9.კომპიუტერული ხედვა.** გამოსახულების ფორმირება და წარმოდგენა, გამოსახულების დისკრეტიზაცია და ქვანტირება, გამოსახულებათა გაუმჯობესება და რესტავრაცია, გამოსახულებათა ანალიზი, გამოსახულების სეგმენტაცია. გამოსახულების ანალიზის და სეგმენტაციის პრაქტიკული მაგალითები Matlab-ის გამოყენებით.

**10. ბუნებრივი ენის დამუშავება.**წესებზე დაფუძნებული და სტატისტიკური NLP. ძირითადი ამოცანები: სინტაქსი, სემანტიკა, დისკურსი, დიალოგები. ქართული ბუნებრივი ენის დამუშავების მაგალითები

**11. თამაშები და მანქანური შემოქმედება.** ხელოვნური ინტელექტის როლი და მონაწილეობის მაგალითები კომპიუტერულ თამაშებში. მომხმარებლის მიერ არამართვადი პერსონაჟები, ქაოსის, ხალხთა მოძრავი მასების, ამინდის, გზის ძიების და გასწრება-დაწევის პროცესების მართვა. პრაქტიკული ამოცანა კონკრეტული თამაშის მაგალითზე .

**12. რობოტები.** ხელოვნური ინტელექტის ადგილი რობოტოტექნიკაში. რობოტების გამოყენება სხვადასხვა სფეროში. - არსებული მაგალითების ჩვენება

**13. ევოლუციური მოდელირება–გენეტიკური ალგორითმები, ჰიბრიდული ინტელექტუალური სისტემები.** გენეტიკური ალგორითმის მედიცინაში გამოყენების მაგალითი,

არსებული ჰიბრიდული ინტელექტუალური სისტემების მიმოხილვა.

**14. ხელოვნური ინტელექტის ბიბლიოთეკები, ფრეიმვორკები და აპლიკაციები**

Tensorflow, DeepLearning Studio, OpenCV, Matlab, Clips,Lips, Panda, DeepLearning Studio, Theano, Keras, Apache Mahout, Apache Spark MLlib, - თითოეულის დათვალიერება, მიმოხილვა და ძირითადი სამუშაო პრინციპების გაცნობა.

**15. ხელოვნური ინტელექტის ეთიკა და ფილოსოფია, ხელოვნური ინტელექტის სამომავლო განვითარება.** ეთიკის საკითხები ხელოვნურ ინტელექტში - ეთიკის პრინციპების ჩამოყალიბება. არსებული საფრთხეების და რისკების მაგალითების მოყვანა.

## პრაქტიკული სამუშაო #5

### ამოცანა 1: გეომეტრიული ფიგურების ამოცნობა გამოსახულებებზე

პრაქტიკული სამუშაო სრულდება დაპროგრამების ენა C++-ზე

პირველ რიგში, არსებული გამოსახულებების ბაზა დავყოთ სასწავლო და საკონტროლო ნაკრებებად. მუშაობას შევუდგეთ სასწავლო ნაკრების გამოყენებით. წავიკითხოთ გრაფიკული ფაილები

პროგრამიდან და არსებული გამოსახულება გადავიყვანოთ ბინარულ ფორმატში პროგრამული კოდის შემდეგი ფრაგმენტის გამოყენებით:

```
#include <iostream>
#include <stdio.h>
#include <string.h>
#include <malloc.h>
unsigned char *read_bmp(char *fname,int* _w, int* _h) {
unsigned char head[54];
FILE *f = fopen(fname,"rb");      // BMP header is 54 bytes
fread(head, 1, 54, f);
int w = head[18] + ( ((int)head[19]) << 8) + ( ((int)head[20]) << 16) + ( ((int)head[21]) << 24);
int h = head[22] + ( ((int)head[23]) << 8) + ( ((int)head[24]) << 16) + ( ((int)head[25]) << 24);
int lineSize = (w / 8 + (w / 8) % 4);
int fileSize = lineSize * h;
unsigned char *img = malloc(w * h), *data = malloc(fileSize);      // skip the header
fseek(f,54,SEEK_SET);      // skip palette - two rgb quads, 8 bytes
fseek(f, 8, SEEK_CUR);      // read data
fread(data,1,fileSize,f);      // decode bits
int i, j, k, rev_j;
for(j = 0, rev_j = h - 1; j < h ; j++, rev_j--) {
   for(i = 0 ; i < w / 8; i++) {
          int fpos = j * lineSize + i, pos = rev_j * w + i * 8;
for(k = 0 ; k < 8 ; k++)
     img[pos + (7 - k)] = (data[fpos] >> k ) & 1;         }     }
free(data);     *_w = w; *_h = h;     return img; }
int main() {     int w, h, i, j;
unsigned char* img = read_bmp("test1.bmp", &w, &h);
for(j = 0 ; j < h ; j++)
 {
     for(i = 0 ; i < w ; i++)
          cout<< img[j * w + i] ? '0' : '1' ;
      cout<<"\n";
 }
 return 0;
}
```

მიღებული გამოსახულებები გამოვიყენოთ ჯერ სწავლებისთვის, ამისთვის ყველა გამოსახულებისათვის (ამ შემთხვევაში უკვე გვაქვს ყოველი გამოსახულების შესაბამისი გეომეტრიული ფიგურის ამსახველი ბინარული მასივები, მათი ზედდებით (სუპერპოზიციით) გამოვთვალოთ მინი პორტრეტი და მაქსი პორტრეტი:

მინი პორტრეტში აისახება მხოლოდ ის მახასიათებელი ნიშანი, რომელიც აქვს ყველა გამოსახულების შესაბამისს რეალიზაციას ერთდროულად. ამ შემთხვევაში ლაპარაკია პიქსელის არსებობაზე გამოსახულებაში. ხოლო მაქსი პორტრეტში აისახება ყოველი პიქსელი, რომელიც შეიძლება ქონდეს შესაბამისად, მინი პორტრეტის მიღების პროგრამული კოდი იქნება:

```
for (k = 0; k < 20; k++) {//გვაქვს 20 რეალიზაცია
  for ( i = 0; i < h; i++ ) {
     for (j = 0; j < w; j++ ) {
        mini[i][j] *= img[i][j];
  }   }
  for ( i = 0; i < h; i++ ) {
     for (j = 0; j < w; j++ ) {
        maxi[i][j] += img[i][j];
  } }  }
```

ამის შემდეგ ვიწყებთ ამოცნობის პროცედურის განხორციელებას. როგორც შესაბამისი ლიტერატურიდან უკვე ვიცით, უცნობი რეალიზაციის ცალსახა ამოცნობა სრულდება მხოლოდ პირველი სიტუაციის შესრულებისას, ანუ მაშინ, როდესაც უცნობი რეალიზაცია მთლიანად თავსდება მაქსი პორტრეტში და მინი პორტრეტი მთლიანად თავსდება უცნობ რეალიზაციაში.

```
for (k = 0; k < 20; k++) {//გვაქვს 20 უცნობი რეალიზაცია
for (p = 0; p < 7; p++) {//ეტალონების რაოდენობა
  for ( i = 0; i < h; i++ ) {
     for (j = 0; j < w; j++ ) {
     if ( img[i][j] && !maxi{i][j] ) && ( !img[i][j] && min[i][j] ) {
          index = p;
           break;
      } } }
} }
cout<<geo_shapes[index]<<endl;  //ამოცნობილი ფიგურის გამოტანა,
```

სადაც geo_shapes[7] = {“samkutxedi”; ”kvadrati”; ”wre”; ”rombi”; ”trapecia”; “wrfe”; “otxkutxedi”};

### ამოცანა 2: ნაბეჭდი ტექსტის ამოცნობა

ამოცანის შესასრულებლად გამოიყენება დაპროგრამების გარემო C++.

ამოცანა შედგება ორი ეტაპისაგან: 1. სწავლება, 2. ამოცნობა.

ამოცნობის პროცესი იყოფა ეტაპებად:

1. სურათის მიღება;
2. სტრიქონების გამოყოფა;
3. სტრიქონში ასოების გამოყოფა;
4. გამოყოფილი ასოების ამოცნობა.

პროგრამას შესასვლელზე მიეწოდება რაიმე ნაბეჭდი ტექსტის ამსახველი სურათი (.png , .gif , .bmp ან .jpg ფაილი);

ამ სურათიდან პირველ ეტაპზე ხდება ცალკეული სიმბოლოების - ასოების გამოყოფა, ანუ Cell ობიექტების დადგენა, რომელთა სტრუქტურაა:

```
class Cell
{
        public int Top;
        public int Bottom;
        public int Left;
        public int Right;
}
```

ამისთვის ჯერ გამოიყოფა სტრიქონები, შესაბამისი სტრუქტურა:

```
class Line
{
        public int Top;
        public int Bottom;
}
```

რისთვისაც ციკლში ვამოწმებთ

```
public static bool scanHorisontal(Bitmap bmp, int y, int x1, int x2)
```

ფუნქციის მნიშვნელობებს. ეს ფუნქცია ამოწმებს ჰორიზონტალურ ხაზში არის თუ არა შავი წერტილი, შემდეგ ხდება Cells ობიექტის გამოთვლა, რომლის ტიპია:

```
public class Cells
    {
        public List<CellRow> cellRows = new List<CellRow>();
    }
```

ხოლო CellRow არის:

```
public class CellRow
    {
        public List<ComplexCell> cells = new List<ComplexCell>();
```

```
    }
```
სადაც ComplexCell არის:
```
public class ComplexCell : Cell
    {   public CharacterBMP CBMP = new CharacterBMP();
        public CharacterVector CV = new CharacterVector();
        public char c = ' ';
    }
```
ამ კლასში CBMP არის ასოს ბიტმაპი და CV არის ასოს შესაბამისი ვექტორი.
ასოებისთვის, პროგრამაში განიხილება 64x64 ზომის სურათები, რომელთა გამოთვლა (სკალირება) ხდება შემდეგი კოდით:
```
Bitmap bmp2 = new Bitmap(64, 64);
                    double Kwid = 64/Width;
                    double Khig = 64/Height;
                    for (int k = 0; k < 64; k++)
                        for (int m = 0; m < 64; m++)
                        {
            Color color = bmp.GetPixel(Left + (int)(k / Kwid), Top + (int)(m / Khig));
                            bmp2.SetPixel(k, m, color);
                        }
```
ვექტორის გამოთვლა ხდება ფუნქციით:
```
public void ComputeVec(CharacterBMP cbmp)
        {
            for (int i = 0; i < 64; i++)
            {
                V[i] = 0;
                for (int j = 0; j < 64; j++)
                {
                    if (TTRClass.isBlack(cbmp.bmp.GetPixel(i, j)))
                        V[i] = V[i] + 1;    }
            }
            for (int i = 0; i < 64; i++)
            {
                V[64 + i] = 0;
                for (int j = 0; j < 64; j++)
                {
                    if (TTRClass.isBlack(cbmp.bmp.GetPixel(j, i)))
                        V[64 + i] = V[64 + i] + 1; }
            }
            V[64 + 64] = cbmp.Height / cbmp.Width;
            V[64 + 64 + 1] = cbmp.Width / cbmp.Height;
            Width = cbmp.Width;
            Height = cbmp.Height;
        }
```
ყველა ასოს გამოთვლის შემდეგ ხდება მათი ამოცნობა. ამისთვის გამოიყენება მახალანობისის მანძილის ფუნქცია Distance:
```
public double Distance(CharacterVector cv1, CharacterVector cv2)
        {double R = 0;
            for (int i = 0; i < 64 + 64 + 2; i++)
            {  R = R + (cv1.V[i] - cv2.V[i]) * (cv1.V[i] - cv2.V[i]);
              }
            R = Math.Sqrt(R);
            return R;
```

}

რომელ სიმბოლოზე - ასოზეც აღწევს ეს ფუნქცია მინიმუმს, ე. ი. ის ასოა ამოცნობის შედეგი.

## პრაქტიკული სამუშაო #6

### ამოცანა 1: ციფრების ამოცნობა კონვოლუციური (ნახვევი) ნეირონული ქსელით MatLab-ის გამოყენებით

ამოცანის შესასრულებლად გამოიყენება პროგრამა Matlab.

შექმენით ღრმა სწავლების ნეირონული ქსელი კლასიფიკაციის ამოცანისთვის. ეს მაგალითი გვიჩვენებს, თუ როგორ უნდა შევქმნათ და მოვამზადოთ მარტივი კონვოლუციური ნეირონული ქსელი ღრმა სწავლების კლასიფიკაციისთვის. კონვოლუციური ნეირონული ქსელები გამოიყენება ღრმა სწავლებისთვის და განსაკუთრებით, გამოსახულების ამოცნობისთვის.

პირველ რიგში ამოცანის გადაწყვეტისას გაითვალისწინეთ შემდეგი თანმიმდევრობები:

1) ჩამოტვირთეთ და შეისწავლეთ გამოსახულების მონაცემები.
2) განსაზღვრეთ ქსელის არქიტექტურა.
3) მიუთითეთ სწავლების (ტრენინგის) ვარიანტები.
4) მოამზადეთ ქსელი.
5) მოახდინეთ პროგნოზირება ახალი მონაცემებისას და გამოთვალეთ კლასიფიკაციის სიზუსტე.

#### 1. ჩატვირთეთ და შეისწავლეთ გამოსახულების მონაცემები

დავიწყოთ მონაცემთა მომზადება. ჩამოტვირთეთ ციფრების ნიმუშების მონაცემები, როგორც გამოსახულებების არსებული მონაცემთა ბაზა 'DigitDataset'. ბრძანება imageDatastore ავტომატურად აღწერს სურათებს ფოლდერის სახელების საფუძველზე და ინახავს მონაცემებს ImageDatastore ობიექტად. სურათის მონაცემთა ბაზა საშუალებას გაძლევთ შეინახოთ დიდი გამოსახულების მონაცემები. განვიხილოთ გამოსახულების კლასიფიკაციის მაგალითი. მონაცემთა ბაზა თუ გვექნება სხვა ტიპის ფაილების, მაგ., ტექსტურის სახით, ასეთ შემთხვევაში საერთო სამუშაო პროცესი დარჩება უცვლელი.

მაგალითად, წარმოვიდგინოთ, რომ გვაქვს გამოსახულებების მონაცემთა ბაზა, რომელსაც გამოვიყენებთ სწავლებისთვის. პირველ რიგში შევქმნით imagedatastore - ტიპს. ასეთი ობიექტის შექმნის შემდეგ, imds შეინახავს ლინკს ჩვენს მონაცემთა ბაზაზე. ეს ობიექტი შეიცავს დამატებით მეთოდებს, რომელსაც ახლა განვიხილავთ. გადავიდეთ MATLAB-ის ბრძანების ფანჯარაში და მივუთითოთ მისამართი ჩვენი გამოსახულებების ბაზის. ჩვენი ბაზა არის ხელნაწერი ციფრების ბაზა 0-დან 9-მდე. კლასიკური მაგალითია, რითიც დაიწყო პრინციპში ღრმა სწავლება და ეს არის პირველი ბაზა, რომელზეც იყო აწყობილი ღრმა სწავლების ალგორითმი. მოვნიშნავთ მისამართს, რომ მივუთითოთ ბაზა

```
path='C:\Program Files\Polyspace\R2019a\toolbox\nnet\nndemos\nndatasets
\DigitDataset'
```

Imds=imageDatastore(path);

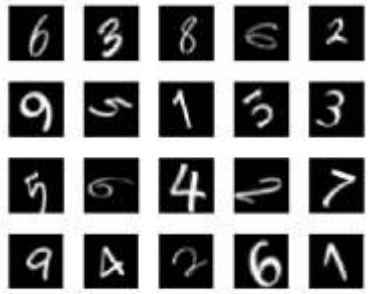
ამ დროს შეცდომა გამოდის, რადგან ის ვერ ხედავს ჩვენს ფოლდერში ვერანაირ სურათს, ის ხედავს მხოლოდ სხვადასხვა ფოლდერებს. ამიტომაც Matlab-ში გამოდის შეცდომა, რის გამოც უნდა მივუთითოთ 'includeSubfolders',

```
imds=imageDatastore(path,'includeSubfolders',true,'LabelSource','foldernames')
```

იმისათვის, რომ დავაყენოთ ნიშნები უნდა დავუმატოთ ჩვენს ფუნქციაში წყვილი თვისება: imds= imageDatastorge(path, 'LabelSource', true, 'folder names', ამის შემდეგ აღარ იქნება ნიშან-თვისებები ცარიელი. ეს იქნება შევსებული ფოლდერების დასახელებებით. ასე, რომ ახლა პრაქტიკულად ყველაფერი მზად გვაქვს, რომ imds ობიექტს გადავცეთ სწავლებისთვის. ეს ობიექტი შეძლებს აიღოს ბაზიდან რამდენიმე გამოსახულება, შექმნას ქსელი, დააყენოს პარამეტრები. მოვახდინოთ რამდენიმე სურათის ჩვენება მონაცემთა ბაზიდან:

```
figure;
perm = randperm(10000,20);
for i = 1:20
    subplot(4,5,i);
```

```
    imshow(imds.Files{perm(i)});
end
```



საერთოდ ძალიან კარგი იქნება, თუ ჩვენ ამ მონაცემებს დავყოფთ ნაწილებად სასწავლო და სატესტო (საკონტროლო) ნაკრებებად. ამის გაკეთება შეიძლება შემდეგნაირად:

Methods (imds) მიმართავს ბრძანება splitEaelLabel (მონაცემთა ბაზის დაყოფა) და ბრძანება CountEaelLabel (ითვლის ფაილთა რაოდენობას).

Imds.countEalabel

```
labelCount = countEachLabel(imds)
labelCount = 10×2 table
```

| N | Label | Count | | N | Label | Count |
|---|---|---|---|---|---|---|
| 1 | 0 | 1000 | | 6 | 5 | 1000 |
| 2 | 1 | 1000 | | 7 | 6 | 1000 |
| 3 | 2 | 1000 | | 8 | 7 | 1000 |
| 4 | 3 | 1000 | | 9 | 8 | 1000 |
| 5 | 4 | 1000 | | 10 | 9 | 1000 |

ჩვენ ვხედავთ ცხრილს. ციფრი 0 - შეიცავს 1000 გამოსახულებას ანუ თითოეული ციფირი შეიცავს 1000 ხელნაწერ გამოსახულებას. თითოეული გამოსახულება არის 28*28*1 პიქსელი.

```
img = readimage(imds,1);
size(img)
ans = 1×2
    28    28
```

**2. შეგვიძლია დავყოთ მონაცემები სასწავლო და სატესტო ნაკრებებად ჩვენი არჩევანით:**

[train, test]=imds.splitEaelLabel(0,75).

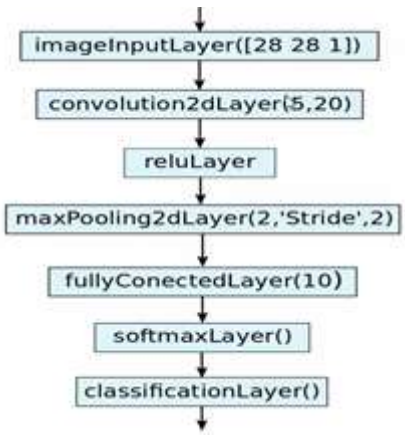
ასევე მას უნდა მივანიჭოთ გამოსასვლელი არგუმენტები. პირველი არგუმენტი ეს არის სწავლება და მცირე ტესტი. გამოსასვლელზე ვხედავთ, რომ მივიღეთ ორი არგუმენტი Train და test. Train შეიცავს 0,75 იგივე 75% სასწავლო რეალიზაციებს და დანარჩენი 35% იქნება ტესტირებისთვის. თუ არ გვინდა, რომ ეს გამოსახულებები მიყვებოდეს თანმიმდევრულად, მაშინ უნდა მივანიჭოთ კიდევ ერთი არგუმენტი `'randomize'`.

```
numTrainFiles = 750;
[trainData,testData]=splitEachLabel(imds, 750, 'randomize')
```

ჩვენი მიზანია შრეების პარამეტრების ოპტიმიზაცია, მაქსიმალურად კარგი შედეგის მისაღებად.

**3. ავაგოთ კონვოლუციური (ნახვევი) ნეირონული ქსელის არქიტექტურა**

```
layers = [
    imageInputLayer([28 28 1])
    convolution2dLayer(5,20)
    reluLayer
    maxPooling2dLayer(2,'Stride',2)
```



```
    fullyConnectedLayer(10)
    softmaxLayer
    classificationLayer];
>> Layer
```

**ImageInputLayer ([28 28 1]**-ში ვუთითებთ სურათის ზომას, რომელიც ამ შემთხვევაში არის [28*28*1]. ეს ციფრები შეესაბამება სიმაღლეს, სიგანეს და არხის ზომას. ციფრული მონაცემები შედგება ნაცრისფერი ჩრდილებისგან, ამიტომ არხის ზომა არის 1. ფერადი სურათისთვის არხის ზომა იქნება 3, რაც RGB მნიშვნელობებს შეესაბამება.

**convolution2dLayer(5,20)**- კონვოლუციურ შრეში პირველი არგუმენტი არის filterSize (ფილტრის ზომა), რომელიც არის ფილტრების სიმაღლე და სიგანე 5*5-ზე, შეგიძლიათ გამოვიყენოთ ასევე სხვადასხვა ზომა ფილტრც. მეორე არგუმენტი არის numFilters (ფილტრების რაოდენობა).

**ReLU Layer** - არის არაწრფივი აქტივაციის ფუნქცია. აქტივაციის ყველაზე გავრცელებული ფუნქცია რექტიფიკაცია წრფივი ერთეული (ReLU), რომელსაც პარამეტრები არ აქვს.

**Max Pooling Layer** - არის ფილტრის ზომა [2*2], რომლის ბიჯიც ('Stride') არის 2.

**Fully Connected Layer (10)** - სრულად ბმული ქსელი. როგორც მისი სახელიდან ჩანს, სრულად დაკავშირებული ფენა არის ფენა, რომელშიც ნეირონები უკავშირდება წინა ფენის ყველა ნეირონს. ეს ფენა აერთიანებს წინა მახასიათებლების მიერ სურათზე ნასწავლ ყველა შედეგს. ბოლო სრულად დაკავშირებული ფენა აერთიანებს მახასიათებლებს სურათების კლასიფიკაციისთვის. ამიტომ, საბოლოოდ სრულად დაკავშირებული შრის OutputSize პარამეტრი უდრის მონაცემების კლასების რაოდენობას. ამ მაგალითში გამომავალი ზომაა 10, რაც შეესაბამება 10 კლასს.

**Softmax Layer** - აქტივაციის ფუნქცია ახდენს სრულად დაკავშირებული ფენის გამოსვლის ნორმალიზებას. ამ შრეს არ აქვს პარამეტრები.

**Classification Layer** - კლასიფიკაციის ფენა საბოლოო ფენა არის კლასიფიკაციის ფენა.

ახლა ჩვენი მიზანია არჩეული სასწავლო ნაკრების სწავლება ქსელისთვის და ამის შემდეგ შევამოწმოთ სატესტო ნაწილი და ვნახოთ თუ როგორ ისწავლა ქსელმა და რა შედეგებს მოგვცემს.

### 4. დასწავლის პარამეტრები.

ქსელის სტრუქტურის განსაზღვრის შემდეგ მივუთითოთ სწავლების პარამეტრები. სწავლებისთვის გამოვიყენოთ გრადიენტული დაშვების ალგორითმი, რომლის საწყისი მნიშვნელობაა 0,01, ხოლო ეპოქების (ბიჯების) რაოდენობაა 4. ეპოქა არის ტრენინგის სრული ციკლი ტრენინგის მთელ მონაცემთა ნაკრებზე.

**პროგრამული კოდი:**

```
%path%
path='C:\Program
Files\Polyspace\R2019a\toolbox\nnet\nndemos\nndatasets\DigitDataset'
imds=imageDatastore(path,'includeSubfolders',true,'LabelSource','foldernames')
[trainData,testData]=splitEachLabel(imds, 750, 'randomize')
%Arrchitecture%
Layer=[...
    imageInputLayer([28 28 1]),...
    convolution2dLayer(5,20),...
    reluLayer,...
    maxPooling2dLayer(2,'stride',2),...
    fullyConnectedLayer(10),...
    softmaxLayer,...
    classificationLayer];
%optimization%
options=trainingOptions('sgdm')
options=trainingOptions('sgdm', 'InitialLearnRate', 0.0005, 'MaxEpochs', 25,
'plots', 'training-progress')
net=trainNetwork(trainData, Layer, options)
t=classify(net,testData)
```

```
sum(t==testData.Labels)/numel(t)
```

**5. შედეგი:**

```
>> convolutionnet
path =  'C:\Program Files\Polyspace\R2019a\toolbox\nnet\nndemos\nndatasets\DigitDataset'
imds =  ImageDatastore with properties:
                    Files: {
                        ' ...\R2019a\toolbox\nnet\nndemos\nndatasets\DigitDataset\0\image10000.png';
                        ' ...\R2019a\toolbox\nnet\nndemos\nndatasets\DigitDataset\0\image9001.png';
                        ' ...\R2019a\toolbox\nnet\nndemos\nndatasets\DigitDataset\0\image9002.png'
                         ... and 9997 more
                        }
                    Labels: [0; 0; 0 ... and 9997 more categorical]
    AlternateFileSystemRoots: {}
                ReadSize: 1
                 ReadFcn: @readDatastoreImage
trainData =
  ImageDatastore with properties:
                    Files: {
                        ' ...\R2019a\toolbox\nnet\nndemos\nndatasets\DigitDataset\0\image10000.png';
                        ' ...\R2019a\toolbox\nnet\nndemos\nndatasets\DigitDataset\0\image9001.png';
                        ' ...\R2019a\toolbox\nnet\nndemos\nndatasets\DigitDataset\0\image9002.png'
                         ... and 7497 more
                        }
                    Labels: [0; 0; 0 ... and 7497 more categorical]
    AlternateFileSystemRoots: {}
                ReadSize: 1
                 ReadFcn: @readDatastoreImage
testData =
  ImageDatastore with properties:
                    Files: {
                        ' ...\R2019a\toolbox\nnet\nndemos\nndatasets\DigitDataset\0\image9006.png';
                        ' ...\R2019a\toolbox\nnet\nndemos\nndatasets\DigitDataset\0\image9010.png';
                        ' ...\R2019a\toolbox\nnet\nndemos\nndatasets\DigitDataset\0\image9019.png'
                         ... and 2497 more
                        }
                    Labels: [0; 0; 0 ... and 2497 more categorical]
    AlternateFileSystemRoots: {}
                ReadSize: 1
                 ReadFcn: @readDatastoreImage
options =
  TrainingOptionsSGDM with properties:
                    Momentum: 0.9000
             InitialLearnRate: 0.0100
    LearnRateScheduleSettings: [1×1 struct]
            L2Regularization: 1.0000e-04
        GradientThresholdMethod: 'l2norm'
            GradientThreshold: Inf
```

```
                MaxEpochs: 30
            MiniBatchSize: 128
                  Verbose: 1
          VerboseFrequency: 50
            ValidationData: []
       ValidationFrequency: 50
        ValidationPatience: Inf
                  Shuffle: 'once'
           CheckpointPath: ''
      ExecutionEnvironment: 'auto'
                WorkerLoad: []
                 OutputFcn: []
                     Plots: 'none'
          SequenceLength: 'longest'
      SequencePaddingValue: 0
      DispatchInBackground: 0
options =
  TrainingOptionsSGDM with properties:

                Momentum: 0.9000
        InitialLearnRate: 5.0000e-04
   LearnRateScheduleSettings: [1×1 struct]
        L2Regularization: 1.0000e-04
    GradientThresholdMethod: 'l2norm'
        GradientThreshold: Inf
                MaxEpochs: 25
            MiniBatchSize: 128
                  Verbose: 1
          VerboseFrequency: 50
            ValidationData: []
       ValidationFrequency: 50
        ValidationPatience: Inf
                  Shuffle: 'once'
           CheckpointPath: ''
      ExecutionEnvironment: 'auto'
                WorkerLoad: []
                 OutputFcn: []
                     Plots: 'training-progress'
          SequenceLength: 'longest'
      SequencePaddingValue: 0
      DispatchInBackground: 0
Training on single CPU.
Initializing input data normalization.
|=========================================================================================|
| Epoch  | Iteration | Time Elapsed  | Mini-batch | Mini-batch  | Base Learning |
|        |           |   (hh:mm:ss)  |  Accuracy  |     Loss    |      Rate     |
```

```
|=========================================================================================|
|           1 |           1 |     00:00:00 |      11.72% |         13.5438 |      0.0005 |
|           1 |          50 |     00:00:06 |      70.31% |          2.0759 |      0.0005 |
|           2 |         100 |     00:00:13 |      79.69% |          1.1465 |      0.0005 |
|           3 |         150 |     00:00:19 |      83.59% |          0.7098 |      0.0005 |
|           4 |         200 |     00:00:25 |      89.84% |          0.5933 |      0.0005 |
|           5 |         250 |     00:00:31 |      92.19% |          0.2863 |      0.0005 |
|           6 |         300 |     00:00:38 |      94.53% |          0.1859 |      0.0005 |
|           7 |         350 |     00:00:45 |      94.53% |          0.2633 |      0.0005 |
|           7 |         400 |     00:00:51 |      96.88% |          0.1282 |      0.0005 |
|           8 |         450 |     00:00:57 |      98.44% |          0.0712 |      0.0005 |
|           9 |         500 |     00:01:04 |     100.00% |          0.0031 |      0.0005 |
|          10 |         550 |     00:01:11 |     100.00% |          0.0020 |      0.0005 |
|          11 |         600 |     00:01:17 |      99.22% |          0.0582 |      0.0005 |
|          12 |         650 |     00:01:23 |      99.22% |          0.0102 |      0.0005 |
|          13 |         700 |     00:01:29 |      98.44% |          0.0381 |      0.0005 |
|          13 |         750 |     00:01:36 |      99.22% |          0.0259 |      0.0005 |
|          14 |         800 |     00:01:43 |     100.00% |          0.0001 |      0.0005 |
|          15 |         850 |     00:01:49 |     100.00% |       8.5763e-05 |      0.0005 |
|          16 |         900 |     00:01:56 |     100.00% |          0.0001 |      0.0005 |
|          17 |         950 |     00:02:06 |     100.00% |          0.0001 |      0.0005 |
|          18 |        1000 |     00:02:18 |     100.00% |       2.2942e-05 |      0.0005 |
|          19 |        1050 |     00:02:29 |     100.00% |       2.6292e-06 |      0.0005 |
|          19 |        1100 |     00:02:39 |     100.00% |       9.6246e-05 |      0.0005 |
|          20 |        1150 |     00:02:48 |     100.00% |       1.3318e-06 |      0.0005 |
|          21 |        1200 |     00:02:58 |     100.00% |       8.8937e-06 |      0.0005 |
|          22 |        1250 |     00:03:10 |     100.00% |       1.9568e-05 |      0.0005 |
|          23 |        1300 |     00:03:24 |     100.00% |       1.8753e-05 |      0.0005 |
|          24 |        1350 |     00:03:35 |     100.00% |       4.6993e-06 |      0.0005 |
|          25 |        1400 |     00:03:46 |     100.00% |       4.3500e-06 |      0.0005 |
|          25 |        1450 |     00:03:57 |     100.00% |       4.9443e-06 |      0.0005 |
|=========================================================================================|

net =
  SeriesNetwork with properties:
    Layers: [7×1 nnet.cnn.layer.Layer]
t =
  2500×1 categorical array
ans =
    0.9852
```


## ამოცანა 2: ნეირონული ქსელის სწავლება ხელნაწერი ციფრების სასწავლო ნაკრებზე - MNIST, Keras ბიბლიოთეკის გამოყენებით

ამოცანის შესასრულებლად გამოიყენება ბიბლიოთეკები: Tensorflow, MNIST, Keras.
Tensorflow ბიბლიოთეკისა და სასწავლო ნაკრების იმპორტირება

```
import tensorflow.compat.v2 as tf
import tensorflow_datasets as tfds
```

Tensorflow 2 ვერსიის გარემოს ჩართვა

```
tf.enable_v2_behavior()
```

ნაბიჯი 1: MNIST სასწავლო ნაკრების ჩატვირთვა
ჩატვირთვა ხდება შემდეგი არგუმენტებით:
* 'mnist': სასწავლო ნაკრების ჩატვირთვა.
* split=['train', 'test']: რეალიზაციების სასწავლო და სატესტო ნაკრებებად დაყოფა
* `shuffle_files`: MNIST სასწავლო ნაკრებ ინახება მხოლოდ ერთ ფაილში, მაგრამ უფრო დიდი ზომის სასწავლო ნაკრებებისთვის რომლებიც რამოდენიმე ფაილში ინახება, საჭიროა მათი არევა ამოცნობის შედეგების გაუმჯობესების მიცნით.
* `as_supervised`: აბრუნებს tuple ცვლადის ტიპს `(img, label)` dict ტიპის მაგივრად `{'image': img, 'label': label}`
* with_info: აბრუნებს tuple ცვლადის ტიპს (tf.data.Dataset, tfds.core.DatasetInfo) დამატებითი ინფორმაციით

```
(ds_train, ds_test), ds_info = tfds.load(
    'mnist',
    split=['train', 'test'],
    shuffle_files=True,
    as_supervised=True,
    with_info=True,
)
```

ამოცნობის არხის აგება
* `ds.map`: Tensorflow-ს მონაცემთა ბაზა ახდენს გამოსახულების მოწოდებას tf.uint8 ფორმატში, როდესაც ნეირონული ქსელის მოდელი ელოდება tf.float32 ფორმატს, ამიტომ ხდება გამოსახულებების ნორმალიზება

* `ds.cache` რადგანაც სასწავლო ნაკრები ეტევა ოპერატიულ მეხსიერებაში, მოხდეს რეალიზაციების ქეშირება მათი რიგითობის არევამდე ოპტიმიზაციის მიზნით.<br/>
__შენიშვნა:__ ქეშირების შემდეგ უნდა მოხდეს გამოსახულებათა შემთხვევითი გარდაქმნები.
* `ds.shuffle`: სასწავლო ნაკრების ელემენტების რიგითობის არევა<br/>
* `ds.batch`: სასწავლო ნაკრების არევის შემდეგ მოხდეს მისი დაჯგუფება, თითოეული ეპოქისთვის უნიკალური ჯგუფის მიწოდების მიზნით.
* `ds.prefetch`: სასწავლო ნაკრების ელემენტების წინასწარ წაკითხვა მყარი დისკიდან, რაოდენობა განისაზღვრება ავტომატურად.

```
def normalize_img(image, label):
  """Normalizes images: `uint8` -> `float32`."""
  return tf.cast(image, tf.float32) / 255., label
ds_train = ds_train.map(
    normalize_img, num_parallel_calls=tf.data.experimental.AUTOTUNE)
ds_train = ds_train.cache()
ds_train = ds_train.shuffle(ds_info.splits['train'].num_examples)
ds_train = ds_train.batch(128)
ds_train = ds_train.prefetch(tf.data.experimental.AUTOTUNE)
```

შეფასების არხის აგება
ტესტირების არხი არის სწავლების არხის მსგავსი, მცირედი განსხვავებებით:
 * არ ხდება `ds.shuffle()`-ის გამოძახება.
 * ქეშირება ხდება დაჯგუფების შემდეგ (რადგანაც ჯგუფები შეიძლება იყოს ერთიდაიგივე, ეპოქებს შორის)

```
ds_test = ds_test.map(
    normalize_img, num_parallel_calls=tf.data.experimental.AUTOTUNE)
ds_test = ds_test.batch(128)
ds_test = ds_test.cache()
ds_test = ds_test.prefetch(tf.data.experimental.AUTOTUNE)
```

ბიჯი 2: სასწავლო მოდელის შექმნა და სწავლება.

შემავალი არხის მიერთება, Keras ბიბლიოთეკასთან.
მოდელის აგება
ნეიროქსელის მოდელი წარმოადგენს სტანდარტულ პირდაპირი გავრცელების ნეირონულ ქსელს.
შემავალ ვექტორს გადაეცემა გამოსახულების მატრიცა განზომილებით 28x28 პიქსელი ხოლო გამომავალი ვექტორი შედგება 10 ელემენტისგან და შეესაბამება ამოსაცნობი რიცხვების რაოდენობას 1-დან 10-ამდე.
ნეიროქსელის მოდელი შედგება შემდეგი ფუნქციებისგან:
* tf.keras.layers.Flatten: ახდენს შემავალი ორგანზომილებიანი მატრიცის 28x28, ერთგანზომილებიან ვექტორზე დაყვანას.
* tf.keras.layers.Dense: ფარული შრის ჩამატება ნეირონების რაოდენობით 128 და "relu" აქტივაციის ფუნქციის მინიჭება.
* tf.keras.layers.Dense: გამომავალი შრის აგება ელემენტების რაოდენბით 10 და "softmax" აქტივაციის ფუნქციის მინიჭება.

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28, 1)),
  tf.keras.layers.Dense(128,activation='relu'),
```

```
  tf.keras.layers.Dense(10, activation='softmax')
])
```

- მოდელის კომპილაცია

```
model.compile(
  loss='sparse_categorical_crossentropy',
  optimizer=tf.keras.optimizers.Adam(0.001),
  metrics=['accuracy'], )
```

- მოდელის სწავლება

```
model.fit(
  ds_train,
  epochs=6,
  validation_data=ds_test,
)
```

## ამოცანა 3: ობიექტის ამოცნობა RetinaNet ნეირონული ქსელის გამოყენებით

ამოცანა შესრულებულია დაპროგრამების ენა Python-ზე, Keras ბიბლიოთეკის გამოყენებით.

მოცემული მოდელის დანიშნულება არის ობიექტების ლოკალიზება გამოსახულებაში და ამავე დროს მათი კლასიფიცირება სხვადასხვა კატეგორიებად.

ობიექტის ამოსაცნობად განკუთვნილი მოდელები შეიძლება დაიყოს ერთსაფეხურიან და ორსაფეხურიან მოდელებად. ორსაფეხურიანი მოდელი არის უფრო ზუსტი, თუმცა ეს მიიღწევა ნაკლები სისწრაფის სანაცვლოდ. მოცემულ მაგალითში ჩვენ გამოვიყენებთ RetinaNet ერთსაფეხურიან ამომცნობ მოდელს, რომელიც ზუსტია და ეშვება სწრაფად.

RetinaNet-ი იყენებს პირამიდულ ქსელს, რათა ეფექტურად ამოიცნოს ობიექტები, გამოსახულების სხვადასხვა მასშტაბის შემთხვევაში და შეიცავს ახალ ფოკალური დანაკარგის ფუნქციას, რათა შეამსუბუქოს უკანა და წინა პლანებს შორის არსებული კლასობრივი დისბალანსი.

**წყაროები:**

- RetinaNet Paper
- Feature Pyramid Network Paper
-
- `import os`
- `import re`
- `import zipfile`
-
- `import numpy as np`
- `import tensorflow as tf`
- `from tensorflow import keras`
-
- `import matplotlib.pyplot as plt`
- `import tensorflow_datasets as tfds`

COCO2017 სასწავლო ნაკრების ჩამოტვირთვა

COCO2017-ის სრული ნაკრებით სწავლება მოითხოვს საკმაოდ დიდ დროს რომელიც შედგება დაახლოებით 118000 გამოსახულებისგან, აქედან გამომდინარე სწავლებისთვის ჩვენ გამოვიყენებთ 500 გამოსახულებისგან შემდგარ მცირე ზომის ქვესიმრავლეს.

```
url = "https://github.com/srihari-humbarwadi/datasets/releases/download/v0.1.0/data.zip"
```

```
filename = os.path.join(os.getcwd(), "data.zip")
keras.utils.get_file(filename, url)
with zipfile.ZipFile("data.zip", "r") as z_fp:
    z_fp.extractall("./")
```

– **დამხმარე ფუნქციების აღწერა**

შემომსაზღვრელი ჩარჩოები შეიძლება იყოს წარმოდგენილი მრავალი სახით, ყველაზე გავრცელებული ფორმატებია:

- კუთხეების კოორდინატების შენახვა [xmin, ymin, xmax, ymax]
- ჩარჩოს ცენტრის კოორდინატებისა და მისი ზომების შენახვა [x, y, width, height]

რადგანაც ჩვენ გვესაჭიროება ორივე ფორმატი, ამიტომ შევქმნით ფუნქციას, რომელიც მოახდენს ამ ორ ფორმატს შორის გარდაქმნას.

```
def swap_xy(boxes):
  """ანაცვლებს ჩარჩოების x და y კოორდინატების რიგითობას
  არგუმენტები:
    ჩარჩოები: ტენზორი ზომებით `(num_boxes, 4)` - წარმოადგენს შემომსაზღვრელ ჩარჩოებს.
  აბრუნებს:  წანაცვლებულ ჩარჩოებს იგივე ზომებით

  """
    return tf.stack([boxes[:, 1], boxes[:, 0], boxes[:, 3], boxes[:, 2]],
                    axis=-1)

def convert_to_xywh(boxes):
    """ცვლის ჩარჩოს ფორმატს - ცენტრი, სიგანე და სიმაღლე

    არგუმენტები:
      ჩარჩოები: ტენზორი რანგით 2 ან უფრო მაღალი და ზომებით
      `(..., num_boxes, 4)`. წარმოადგენს შემომსაზღვრელ ჩარჩოს,
      სადაც თითოეულ ჩარჩოს აქვს ფორმატი `[xmin, ymin, xmax, ymax]`.

    აბრუნებს:
      გარდაქმნილ ჩარჩოებს იგივე ზომებით რაც არგუმენტ boxes გააჩნია.
    """
    return tf.concat(
      [(boxes[...,:2]+boxes[...,2:]) / 2.0,boxes[...,2:]-boxes[..., :2]],
        axis=-1,
    )

def convert_to_corners(boxes):
    """გარდაქმნის ჩარჩოს ფორმატს კუთხის კოორდინატებზე

    არგუმენტები:
      ჩარჩოები: ტენზორი რანგით 2 ან უფრო მაღალი, ზომებით `(..., num_boxes,
4)`
        წარმოადგენს შემომსაზღვრელ ჩარჩოებს სადაც თითოეულ ჩარჩოს აქვს
ფორმატი
        `[x, y, width, height]`.

    აბრუნებს:
```

```
        გარდაქმნილ ჩარჩოებს იგივე ზომებით რაც არგუმენტ boxes გააჩნია.
    """
 return tf.concat(
   [boxes[...,:2]-boxes[...,2:] / 2.0,boxes[...,:2]+boxes[...,2:] / 2.0],
        axis=-1,
    )
```

თანაკვეთის გაერთიანებასთან ფარდობის დაწყვილებულად გამოთვლა

როგორც მოგვიანებით ვნახავთ ამ მაგალითში, ანკერულ ჩარჩოებს მივაკუთვნებთ ეტალონურ ჩარჩოებს მათი გადაფარვის ხარისხის მიხედვით. ამისათვის საჭირო იქნება გამოითვალოს ყველა ანკერულ და ეტალონურ ჩარჩოს წყვილს შორის თანაკვეთის ფარდობა მათ გაერთიანებასთან.

```
def compute_iou(boxes1, boxes2):
    """ითვლის მოცემული ჩარჩოების ნაკრებების თანაკვეთის ფარდობას მათ
გაერთიანებასთან დაწყვილებულად.

  არგუმენტი:
      boxes1: ტენზორი ზომებით `(N, 4)` შემომსაზღვრელი ჩარჩოებია,
              სადაც თითოეულ ჩარჩოს აქვს შემდეგი ფორმატი -
                  `[x, y, width, height]`.
      boxes2: ტენზორი ზომებით `(M, 4)` შემომსაზღვრელი ჩარჩოებია,
              სადაც, თითოეულ ჩარჩოს აქვს შემდეგი ფორმატი -
                  `[x, y, width, height]`.
    აბრუნებს:
      დაწყვილებული ჩარჩოების თანაკვეთის მათივე გაერთიანებასთან ფარდობით
მიღებულ მატრიცას, ზომებით `(N, M)`, სადაც i-ურ სტრიქონსა და j-ურ სვეტში
      არსებული ელემენტის მნიშვნელობა boxes1 და boxes2 სიმრავლეებიდან i-ური
და j-ური ჩარჩოების თანაკვეთის ფარდობაა მათივე გაერთიანებასთან.
    """
    boxes1_corners = convert_to_corners(boxes1)
    boxes2_corners = convert_to_corners(boxes2)
    lu = tf.maximum(boxes1_corners[:, None, :2], boxes2_corners[:, :2])
    rd = tf.minimum(boxes1_corners[:, None, 2:], boxes2_corners[:, 2:])
    intersection = tf.maximum(0.0, rd - lu)
    intersection_area = intersection[:, :, 0] * intersection[:, :, 1]
    boxes1_area = boxes1[:, 2] * boxes1[:, 3]
    boxes2_area = boxes2[:, 2] * boxes2[:, 3]
    union_area = tf.maximum(
        boxes1_area[:, None] + boxes2_area - intersection_area, 1e-8
    )
    return tf.clip_by_value(intersection_area / union_area, 0.0, 1.0)

def visualize_detections(
 image, boxes, classes, scores, figsize=(7,7), linewidth=1,color=[0,0,1]
):
    """ამოცნობის ვიზუალიზაცია"""
    image = np.array(image, dtype=np.uint8)
    plt.figure(figsize=figsize)
    plt.axis("off")
    plt.imshow(image)
    ax = plt.gca()
    for box, _cls, score in zip(boxes, classes, scores):
```

```
            text = "{}: {:.2f}".format(_cls, score)
            x1, y1, x2, y2 = box
            w, h = x2 - x1, y2 - y1
            patch = plt.Rectangle(
             [x1, y1], w, h, fill=False, edgecolor=color, linewidth=linewidth
            )
            ax.add_patch(patch)
            ax.text(
                x1,
                y1,
                text,
                bbox={"facecolor": color, "alpha": 0.4},
                clip_box=ax.clipbox,
                clip_on=True,
            )
        plt.show()
        return ax
```

**ანკერების გენერატორის იმპლემენტაცია**

ანკერულ ჩარჩოს აქვს ფიქსირებული ზომები და გამოიყენება ამოსაცნობი ობიექტის შემომსაზღვრელი ჩარჩოს ადგილმდებარეობის პროგნოზირებისათვის. ის ამას ახდენს ობიექტის ცენტრსა და ანკერულ ჩარჩოს ცენტრს შორის არსებული დაშორების რეგრესირებით, შემდეგ ის იყენებს ანკერული ჩარჩოს სიგანეს და სიმაღლეს რათა მოახდინოს ამოსაცნობი ობიექტის ფარდობითი მასშტაბი. RetinaNet-ის შემთხვევაში ნიშანთა რუკის თითოეულ ლოკაციას გააჩნია ცხრა ანკერული ჩარჩო. (სამი სხვადასხვა მასშტაბითა და სამი თანაფარდობით)

```
class AnchorBox:
    """აგენერირებს ანკერულ ჩარჩოებს.

    მოცემული კლასი ანხორციელებს ოპერაციებს ანკერული ჩარჩოების
დასაგენერირებლად
    ნიშანთა რუკებისათვის, ბიჯებით `[8, 16, 32, 64, 128]`. სადაც თითოეული
ანკერული
    ჩარჩოს აქვს შემდეგი ფორმატი `[x, y, width, height]`.

    ცვლადები:
      aspect_ratios: float ტიპის რიცხვების ჩამონათვალი რაც წარმოადგენს
      ანკერული ჩარჩოების გვერდების თანაფარდობას ნიშანთა რუკის თითოეულ
ლოკაციაზე.
      scales: float ტიპის რიცხვების ჩამონათვალი რაც წარმოადგენს ანკერული
ჩარჩოების მასშტაბს ნიშანთა რუკის თითოეულ ლოკაციაზე.
      num_anchors: ანკერული ჩარჩოების რაოდენობაა ნიშანთა რუკის თითოეულ
ლოკაციაზე.
      areas: float ტიპის რიცხვების ჩამონათვალი, რაც წარმოადგენს ანკერული
ჩარჩოების ფართობებს თითოეული ნიშანთა რუკისათვის, ნიშანთა პირამიდაში.
      strides: float ტიპის რიცხვების ჩამონათვალი რომელიც წარმოადგენს ბიჯებს
თითოეული ნიშანთა რუკისათვის, ნიშანთა პირამიდაში.
    """

    def __init__(self):
        self.aspect_ratios = [0.5, 1.0, 2.0]
```

```
        self.scales = [2 ** x for x in [0, 1 / 3, 2 / 3]]

        self._num_anchors = len(self.aspect_ratios) * len(self.scales)
        self._strides = [2 ** i for i in range(3, 8)]
        self._areas = [x ** 2 for x in [32.0, 64.0, 128.0, 256.0, 512.0]]
        self._anchor_dims = self._compute_dims()

    def _compute_dims(self):
        """ითვლის ანკერული ჩარჩოს ზომებს, ყველა თანაფარდობისა და
მასშტაბისათვის,
        ნიშანთა პირამიდის ყველა დონეზე.
        """
        anchor_dims_all = []
        for area in self._areas:
            anchor_dims = []
            for ratio in self.aspect_ratios:
              anchor_height = tf.math.sqrt(area / ratio)
              anchor_width = area / anchor_height
              dims = tf.reshape(
               tf.stack([anchor_width, anchor_height], axis=-1), [1, 1, 2]
                )
                for scale in self.scales:
                    anchor_dims.append(scale * dims)
            anchor_dims_all.append(tf.stack(anchor_dims, axis=-2))
        return anchor_dims_all

    def _get_anchors(self, feature_height, feature_width, level):
        """აგენერირებს ანკერულ ჩარჩოს, მოცემული ნიშანთა რუკის ზომისა და
დონისათვის

        არგუმენტები:
 feature_height: მთელი რიცხვი, რომელიც წარმოადგენს ნიშანთა რუკის სიმაღლეს.
 feature_width: მთელი რიცხვი, რომელიც წარმოადგენს ნიშანთა რუკის სიგანეს.
     level: მთელი რიცხვი, რომელიც ნიშანთა რუკის დონეა ნიშანთა პირამიდაში.

    აბრუნებს:
       ანკერულ ჩარჩოებს შემდეგი ზომებით
          `(feature_height * feature_width * num_anchors, 4)`
        """
        rx = tf.range(feature_width, dtype=tf.float32) + 0.5
        ry = tf.range(feature_height, dtype=tf.float32) + 0.5
        centers = tf.stack(tf.meshgrid(rx, ry), axis=-1) *
                     self._strides[level - 3]
        centers = tf.expand_dims(centers, axis=-2)
        centers = tf.tile(centers, [1, 1, self._num_anchors, 1])
        dims = tf.tile(
       self._anchor_dims[level - 3], [feature_height, feature_width, 1, 1]
        )
        anchors = tf.concat([centers, dims], axis=-1)
        return tf.reshape(
          anchors, [feature_height * feature_width * self._num_anchors, 4]
        )
```

```
    def get_anchors(self, image_height, image_width):
        """აგენერირებს ანკერულ ჩარჩოებს ნიშანთა პირამიდის ყველა ნიშანთა
რუკისათვის.

        არგუმენტები:
          image_height: შემავალი გამოსახულების სიმაღლე.
          image_width: შემავალი გამოსახულების სიგანე.

        აბრუნებს:
        ერთ ტენზორად დაჯგუფებულ ანკერულ ჩარჩოებს ყველა ნიშანთა
რუკისათვის,
        ზომებით -  `(total_anchors, 4)`

        """
        anchors = [
            self._get_anchors(
                tf.math.ceil(image_height / 2 ** i),
                tf.math.ceil(image_width / 2 ** i),
                i,
            )
            for i in range(3, 8)
        ]
        return tf.concat(anchors, axis=0)
```

– მონაცემთა პრეპარირება

გამოსახულებების პრეპარირებაა შედგება ორი ბიჯისგან:

- გამოსახულების მასშტაბირება: გამოსახულებების მასშტაბირება ხდება ისე, რომ უმცირესი გვერდი შედგება 800 პიქსელისგან. მასშტაბირების შემდეგ თუ უდიდესი გვერდი აჭარბებს 1333 პიქსელს, მისთვის ხდება მაქსიმალური ზღურბლის დაწესება 1333 პიქსელის სახით;
- ცვლილებების ზედდება: შემთხვევითი მასშტაბის რხევები და ჰორიზონტალური ბრუნები არის ერთადერთი ცვლილებები რომელთა გამოყენებაც ხდება გამოსახულებებზე.

გამოსახულებებთან ერთად, ხდება შემომსაზღვრელი ჩარჩოების მასშტაბირება და მობრუნება, საჭიროების მიხედვით.

```
def random_flip_horizontal(image, boxes):
    """ აბრუნებს გამოსახულებას ჰორიზონტალურად 50%-იანი ალბათური შანსით.
    არგუმენტები:
      image: სამგანზომილებიანი ტენზორი ზომებით `(height, width, channels)`
წარმოადგენს გამოსახულებას.
      boxes: ტენზორი ზომებით `(num_boxes, 4)` წარმოადგენს შემომსაზღვრელ
ჩარჩოებს,რომელთაც აქვთ ნორმალიზებული კოორდინატები.

    აბრუნებს:
      შემთხვევითად შემობრუნებულ გამოსახულებებსა და ჩარჩოებს
    """
    if tf.random.uniform(()) > 0.5:
      image = tf.image.flip_left_right(image)
      boxes = tf.stack(
      [1 - boxes[:, 2], boxes[:, 1], 1 - boxes[:, 0], boxes[:, 3]], axis=-1
```

```
        )
    return image, boxes

def resize_and_pad_image(
    image, min_side=800.0, max_side=1333.0, jitter=[640, 1024], stride=128.0
):
    """ახდენს გამოსახულების მასშტაბირებასა და მის შევსებას, გვერდების
თანაფარდობის
      დაცვით

    1. ახდენს გამოსახულების მასშტაბირებას ისე, რომ მისი უმცირესი გვერდი
       `min_side`-ის მნიშვნელობის ტოლია.
    2. თუ უდიდესი გვერდი `max_side`-ის მნიშვნელობის ტოლია, მოხდეს
       გამოსახულების მასშტაბირება რათა უდიდესი გვერდი გაუტოლდეს
`max_side`-ს.
    3. შეივსოს გამოსახულება 0-ებით მარჯვენა და ქვედა მხარეს რათა გაყოფადი
გახდეს
    `stride`-ზე

    არგუმენტები:
    image:     სამგანზომილებიანი ტენზორი ზომებით `(height, width, channels)`
               წარმოადგენს გამოსახულებას.
    min_side: გამოსახულების უმცირესი გვერდი დაიყვანება მოცემული ცვლადის
               მნიშვნელობაზე თუ `jitter`-ის მნიშვნელობა არის None.
   max_side: თუ მასშტაბირების შემდეგ გამოსახულების უდიდესი გვერდი აჭარბებს
               მოცემული ცვლადის მნიშვნელობას, მასშტაბირება ხდება ისე, რომ
               უდიდესი გვერდი გაუტოლდეს მოცემული ცვლადის მნიშვნელობას.
  jitter:   float ტიპის რიცხვების ჩამონათვალი რომელიც შეიცავს მინიმალურ და
               მაქსიმალურ ზომებს მასშტაბირებისა და რხევებისათვის.
      stride:   უმცირესი ნიშანთა რუკის ბიჯი ნიშანთა პირამიდაში.
               გამოითვლება შემდეგნაირად `image_size / feature_map_size`.

    აბრუნებს:
      image: მასშტაბირებულ და შევსებულ გამოსახულებას.
      image_shape: გამოსახულების ფორმა შევსებამდე.
      ratio: გამოსახულების მასშტაბირების კოეფიციენტი.
    """
    image_shape = tf.cast(tf.shape(image)[:2], dtype=tf.float32)
    if jitter is not None:
        min_side = tf.random.uniform((), jitter[0], jitter[1], dtype=tf.float32)
    ratio = min_side / tf.reduce_min(image_shape)
    if ratio * tf.reduce_max(image_shape) > max_side:
        ratio = max_side / tf.reduce_max(image_shape)
    image_shape = ratio * image_shape
    image = tf.image.resize(image, tf.cast(image_shape, dtype=tf.int32))
    padded_image_shape = tf.cast(
        tf.math.ceil(image_shape / stride) * stride, dtype=tf.int32
    )
    image = tf.image.pad_to_bounding_box(
```

```
        image, 0, 0, padded_image_shape[0], padded_image_shape[1]
    )
    return image, image_shape, ratio

def preprocess_data(sample):
    """იყენებს პრეპარირების ბიჯს ერთ ნიმუშზე

    არგუმენტები:
      sample: dict ტიპის ცვლადი რომელიც წარმოადგენს ერთ სასაწავლო ნიმუშს.

    აბრუნებს:
      image:     მასშტაბირებული და შევსებული გამოსახულება შემთხვევითი
                 ჰორიზონტალური ბრუნით.
      bbox:      შემომსაზღვრელი ჩარჩოები ზომებით `(num_objects, 4)`,
            სადაც თითოეულ ჩარჩოს აქვს ფორმატი `[x, y, width, height]`.
    class_id: ტენზორი, რომელიც ობიექტების კლასის საიდენტიფიკაციო
                 ნომერია, ზომებით `(num_objects,)`.
    """
    image = sample["image"]
    bbox = swap_xy(sample["objects"]["bbox"])
    class_id = tf.cast(sample["objects"]["label"], dtype=tf.int32)

    image, bbox = random_flip_horizontal(image, bbox)
    image, image_shape, _ = resize_and_pad_image(image)

    bbox = tf.stack(
        [
            bbox[:, 0] * image_shape[1],
            bbox[:, 1] * image_shape[0],
            bbox[:, 2] * image_shape[1],
            bbox[:, 3] * image_shape[0],
        ],
        axis=-1,
    )
    bbox = convert_to_xywh(bbox)
    return image, bbox, class_id
```

➢ ჭდეების კოდირება

ჭდეები რომლებიც შედგება შემომსაზღვრელი ჩარჩოებისა და კლასის საიდენტიფიკაციო ნომრებისგან, საჭიროა მოხდეს მათი მიზნის ვექტორებად გარდაქმნა სწავლების პროცესში. ეს გარდაქმნა შედგება შემდეგი ბიჯებისგან:

- ანკერული ჩარჩოების გენერირება მოცემული გამოსახულების ზომებისათვის.
- ეტალონური ჩარჩოების ანკერული ჩარჩოებისათვის მინიჭებაა.
- ის ანკერული ჩარჩოები რომლებიც არ არიან მინიჭებული არცერთი ობიექტისათვის, იქნებიან მინიჭებული უკანა პლანის კლასისათვის ან მოხდება მათი იგნორირება, თანაკვეთის გაერთიანებასთან ფარდობის შედეგის მიხედვით.
- გენერირდება კლასიფიკაციისა და რეგრესიისათვის საჭირო მიზნის ვექტორები ანკერული ჩარჩოების გამოყენებით.
- `class LabelEncoder:`

```
    """გარდაქმნის ჭდეებს მიზნის ვექტორებად სწავლებისათვის.

    მოცემულ კლასს გააჩნია ფუნქციები მიზნის ვექტორების დასაგენერირებლად ნიმუშების
    ჯგუფისათვის, რომელიც შედგება შემავალი გამოსახულებებისგან, ობიექტების
    შემომსაზღვრელი ჩარჩოებისგან და მათი საიდენტიფიკაციო ნომრებისგან.

    ცვლადები:
      anchor_box: ანკერული ჩარჩოების გენერატორი შემომსაზღვრელი ჩარჩოების კოდირებისათვის.
      box_variance: მასშტაბირების კოეფიციენტები რომლებიც გამოიყენება
      შემომსაზღვრელი ჩარჩოების სამიზნე გამოსახულებების მასშტაბირებისათვის.
    """

    def __init__(self):
        self._anchor_box = AnchorBox()
        self._box_variance = tf.convert_to_tensor(
            [0.1, 0.1, 0.2, 0.2], dtype=tf.float32
        )

  def _match_anchor_boxes(
    self, anchor_boxes, gt_boxes, match_iou=0.5, ignore_iou=0.4
    ):
        """ანკერულ ჩარჩოებს უსადაგებს ეტალონურ ჩარჩოებს, თანაკვეთის
        გაერთიანებასთან ფარდობის მიხედვით.

  1. ითვლის თანაკვეთის გაერთიანებასთან ფარდობას დაწყვილებულად
    M `anchor_boxes`-ისა და  N `gt_boxes`-ისათვის რათა მივიღოთ `(M, N)`
          ზომების მატრიცა.

  2. თითოეულ ეტალონურ ჩარჩოს სტრიქონში მაქსიმალური თანაკვეთის
      გაერთიანებასთან ფარდობით, მიესადაგება ანკერული ჩარჩო რომლის
      თანაკვეთის გაერთიანება ფარდობასთან არის `match_iou`-ის
      მნიშვნელობაზე მეტი.
  3. თუ სტრიქონში არსებული მაქსიმალური თანაკვეთის გაერთიანება არის
          ნაკლები ვიდრე `ignore_iou`-ს მნიშვნელობა, ანკერული ჩარჩო
          მიენიჭება უკანა პლანის კლასს.
  4. დარჩენილი ანკერული ჩარჩოები სწავლების დროს დაიგნორდება
          რომელთაც არ გააჩნია არანაირი კლასი.

  არგუმენტები:
    anchor_boxes: float ტიპის ტენზორი ზომებით `(total_anchors, 4)`
      რომელიც არის ყველა ანკერული ჩარჩო, მოცემული შემავალი
      გამოსახულების ზომისათვის, სადაც თითოეულ ანკერულ ჩარჩოს აქვს
      `[x, y, width, height]` ფორმატი.
    gt_boxes: float ტიპის ტენზორი ზომებით `(num_objects, 4)`, რომელიც
    არის ეტალონური ჩარჩოები, სადაც თითოეულ ჩარჩოს აქვს
    `[x, y, width, height]` ფორმატი.
```

```
    match_iou: float ტიპის ცვლადი რომელიც წარმოადგენს თანაკვეთის
       გაერთიანებასთან ფარდობის მინიმალურ ზღვარს. იგი განსაზღვრავს
           შესაძლებელია თუ არა, რომ ეტალონური ჩარჩო მიენიჭოს ანკერულ
           ჩარჩოს.
    ignore_iou: float ტიპის ცვლადი, რომელიც თანაკვეთის გაერთიანებასთან
      ფარდობის ზღვარია, რომლის საშუალებითაც ხდება ანკერული ჩარჩოს
       უკანა პლანის კლასისათვის მინიჭება.

          აბრუნებს:
     matched_gt_idx: ამოცნობილი ობიექტის ინდექსს.
     positive_mask:  ანკერული ჩარჩოების ნიღაბი, რომელიც განსაზღვრავს თუ
                  რომელი ჩარჩოსათვის მოხდა ეტალონური ჩარჩოს მინიჭება.
    ignore_mask: ანკერული ჩარჩოების ნიღაბი, განსაზღვრავს იმ ჩარჩოებს,
          რომლებიც უნდა იქნან გამოტოვებული სწავლების დროს.
          """
          iou_matrix = compute_iou(anchor_boxes, gt_boxes)
          max_iou = tf.reduce_max(iou_matrix, axis=1)
          matched_gt_idx = tf.argmax(iou_matrix, axis=1)
          positive_mask = tf.greater_equal(max_iou, match_iou)
          negative_mask = tf.less(max_iou, ignore_iou)
    ignore_mask = tf.logical_not(tf.logical_or(positive_mask, negative_mask))
          return (
               matched_gt_idx,
               tf.cast(positive_mask, dtype=tf.float32),
               tf.cast(ignore_mask, dtype=tf.float32),
          )

    def _compute_box_target(self, anchor_boxes, matched_gt_boxes):
    """გარდაქმნის ეტალონურ ჩარჩოებს მიზნის ვექტორებად,სწავლებისთვის"""
       box_target = tf.concat(
        [
       (matched_gt_boxes[:, :2] - anchor_boxes[:, :2]) / anchor_boxes[:, 2:],
            tf.math.log(matched_gt_boxes[:, 2:] / anchor_boxes[:, 2:]),
          ],
        axis=-1,
     )
       box_target = box_target / self._box_variance
     return box_target

     def _encode_sample(self, image_shape, gt_boxes, cls_ids):
          """ქმნის ჩარჩოსა და მიზნის ვექტორს ერთი ნიმუშისათვის"""
     anchor_boxes = self._anchor_box.get_anchors(image_shape[1],
        image_shape[2])
          cls_ids = tf.cast(cls_ids, dtype=tf.float32)
          matched_gt_idx, positive_mask, ignore_mask =
             self._match_anchor_boxes(
               anchor_boxes, gt_boxes
          )
      matched_gt_boxes = tf.gather(gt_boxes, matched_gt_idx)
      box_target = self._compute_box_target(anchor_boxes, matched_gt_boxes)
```

```
        matched_gt_cls_ids = tf.gather(cls_ids, matched_gt_idx)
        cls_target = tf.where(
            tf.not_equal(positive_mask, 1.0), -1.0, matched_gt_cls_ids
            )
        cls_target = tf.where(tf.equal(ignore_mask, 1.0), -2.0, cls_target)
            cls_target = tf.expand_dims(cls_target, axis=-1)
            label = tf.concat([box_target, cls_target], axis=-1)
            return label

        def encode_batch(self, batch_images, gt_boxes, cls_ids):
            """ქმნის ჩარჩოსა და მიზნის ვექტორებს ჯგუფისათვის"""
            images_shape = tf.shape(batch_images)
            batch_size = images_shape[0]

       labels = tf.TensorArray(dtype=tf.float32, size=batch_size,
          dynamic_size=True)
           for i in range(batch_size):
              label = self._encode_sample(images_shape, gt_boxes[i],
                  cls_ids[i])
              labels = labels.write(i, label)
            batch_images =
     tf.keras.applications.resnet.preprocess_input(batch_images)
        return batch_images, labels.stack()
```

### ResNet50-ის მაგისტრალის აგება

RetinaNet იყენებს ResNet-ზე დაფუძნებულ მაგისტრალს, რომლის გამოყენების დროსაც ხდება ნიშანთა პირამიდის ქსელის აგება. ამ ამოცანაში ჩვენ ვიყენებთ ResNet50-ს მაგისტრალად და შედეგად დაბრუნებულ ნიშანთა რუკებს ბიჯებით 8, 16, და 32.

```
def get_backbone():
 """აგებს ResNet50-ს წინასწარ ნასწავლი imagenet-ის წონითი კოეფიციენტებით"""
    backbone = keras.applications.ResNet50(
        include_top=False, input_shape=[None, None, 3]
    )
    c3_output, c4_output, c5_output = [
        backbone.get_layer(layer_name).output
 for layer_name in ["conv3_block4_out", "conv4_block6_out", "conv5_block3_out"]
   ]
   return keras.Model(
     inputs=[backbone.inputs], outputs=[c3_output, c4_output, c5_output]
  )
```

### ნიშანთა პირამიდული ქსელის აგება, Keras-ის შრის სახით

```
class FeaturePyramid(keras.layers.Layer):
   """აგებს ნიშანთა პირამიდას, ნიშანთა რუკის გამოყენებით.

   ცვლადები:
      num_classes: კლასების რაოდენობა სასწავლო ნაკრებში.
```

```
    backbone:    მაგისტრალი საიდანაც ხდება ნიშანთა პირამიდის აგება
                 ამჟამად აქვს მხოლოდ ResNet50-ის მხარდაჭერა.
  """

    def __init__(self, backbone=None, **kwargs):
        super(FeaturePyramid, self).__init__(name="FeaturePyramid", **kwargs)
        self.backbone = backbone if backbone else get_backbone()
        self.conv_c3_1x1 = keras.layers.Conv2D(256, 1, 1, "same")
        self.conv_c4_1x1 = keras.layers.Conv2D(256, 1, 1, "same")
        self.conv_c5_1x1 = keras.layers.Conv2D(256, 1, 1, "same")
        self.conv_c3_3x3 = keras.layers.Conv2D(256, 3, 1, "same")
        self.conv_c4_3x3 = keras.layers.Conv2D(256, 3, 1, "same")
        self.conv_c5_3x3 = keras.layers.Conv2D(256, 3, 1, "same")
        self.conv_c6_3x3 = keras.layers.Conv2D(256, 3, 2, "same")
        self.conv_c7_3x3 = keras.layers.Conv2D(256, 3, 2, "same")
        self.upsample_2x = keras.layers.UpSampling2D(2)

    def call(self, images, training=False):
        c3_output, c4_output, c5_output = self.backbone(images,
training=training)
        p3_output = self.conv_c3_1x1(c3_output)
        p4_output = self.conv_c4_1x1(c4_output)
        p5_output = self.conv_c5_1x1(c5_output)
        p4_output = p4_output + self.upsample_2x(p5_output)
        p3_output = p3_output + self.upsample_2x(p4_output)
        p3_output = self.conv_c3_3x3(p3_output)
        p4_output = self.conv_c4_3x3(p4_output)
        p5_output = self.conv_c5_3x3(p5_output)
        p6_output = self.conv_c6_3x3(c5_output)
        p7_output = self.conv_c7_3x3(tf.nn.relu(p6_output))
        return p3_output, p4_output, p5_output, p6_output, p7_output
```

კლასიფიკაციისა და ჩარჩოს რეგრესიის თავების აგება.

RetinaNet-ს გააჩნია რამდენიმე თავი, შემომსაზღვრელი ჩარჩოს რეგრესიისა და ობიექტის კლასის ალბათობების პროგნოზირებისათვის. ხდება ამ თავების გაზიარება ნიშანთა პირამიდის ყველა ნიშანთა რუკას შორის.

```
def build_head(output_filters, bias_init):
    """აგებს კლასის/ჩარჩოს პროგნოზირების თავს

    არგუმენტები:
      output_filters: ნახვევი ფილტრების რაოდენობა ბოლო შრეში.
      bias_init: წანაცვლების კოეფიციენტი ბოლო ნახვევი შრისათვის.

    აბრუნებს: keras-ის თანმიმდევრული მოდელი, რომელიც კლასიფიკაცია ან
              ჩარჩოს რეგრესიის თავია `output_filters`-ისდა მიხედვით.
    """
    head = keras.Sequential([keras.Input(shape=[None, None, 256])])
    kernel_init = tf.initializers.RandomNormal(0.0, 0.01)
    for _ in range(4):
      head.add(
```

```
            keras.layers.Conv2D(256, 3, padding="same",
                    kernel_initializer=kernel_init)
            )
            head.add(keras.layers.ReLU())
        head.add(
            keras.layers.Conv2D(
                output_filters,
                3,
                1,
                padding="same",
                kernel_initializer=kernel_init,
                bias_initializer=bias_init,
            )
        )
        return head
```

RetinaNet-ის აგება ქვეკლასებად დაყოფილი მოდელის საშუალებით.

```
class RetinaNet(keras.Model):
    """Keras-ის ქვეკლასებად დაყოფილი მოდელი, რომელიც იყენებს RetinaNet
    არქიტექტურას.

    ცვლადები:
      num_classes: კლასების რაოდენობა სასწავლო ნაკრებში.
      backbone: მაგისტრალი საიდანაც ხდება ნიშანთა პირამიდის აგება
        ამჟამად აქვს მხოლოდ ResNet50-ის მხარდაჭერა.
    """

    def __init__(self, num_classes, backbone=None, **kwargs):
        super(RetinaNet, self).__init__(name="RetinaNet", **kwargs)
        self.fpn = FeaturePyramid(backbone)
        self.num_classes = num_classes

        prior_probability = tf.constant_initializer(-np.log((1 - 0.01) / 0.01))
        self.cls_head = build_head(9 * num_classes, prior_probability)
        self.box_head = build_head(9 * 4, "zeros")

    def call(self, image, training=False):
        features = self.fpn(image, training=training)
        N = tf.shape(image)[0]
        cls_outputs = []
        box_outputs = []
        for feature in features:
            box_outputs.append(tf.reshape(self.box_head(feature), [N, -1, 4]))
            cls_outputs.append(
                tf.reshape(self.cls_head(feature), [N, -1, self.num_classes])
            )
        cls_outputs = tf.concat(cls_outputs, axis=1)
        box_outputs = tf.concat(box_outputs, axis=1)
        return tf.concat([box_outputs, cls_outputs], axis=-1)
```

ინდივიდუალური შრის აგება, პროგნოზების დეკოდირებისათვის

```
class DecodePredictions(tf.keras.layers.Layer):
    """ Keras-ის შრე რომელიც ახდენს RetinaNet-ის მოდელის პროგნოზების
                დეკოდირებას.

  ცვლადები:
    num_classes: კლასების რაოდენობა სასწავლო ნაკრებში
    confidence_threshold: კლასის მინიმალური ალბათობა რომლის ქვემოთაც ხდება
            ჩამოჭრა ამოცნობის დროს.

    nms_iou_threshold: თანაკვეთის გაერთიანებასთან ფარდობის ზღურბლი NMS
                            ოპერაციისათვის
    max_detections_per_class: ამოცნობათა მაქსიმალური რაოდენობა თითო კლასს
            შორის, რომელთა შენარჩუნებაც მოხდება.
    max_detections: ამოცნობათა მაქსიმალური რაოდენობა ყველა კლასს შორის,
        რომელთა შენარჩუნებაც მოხდება.

    box_variance: მასშტაბირების კოეფიციენტი, რომელიც გამოიყენება
          შემომსაზღვრელი ჩარჩოს პროგნოზირებისათვის
    """

    def __init__(
        self,
        num_classes=80,
        confidence_threshold=0.05,
        nms_iou_threshold=0.5,
        max_detections_per_class=100,
        max_detections=100,
        box_variance=[0.1, 0.1, 0.2, 0.2],
        **kwargs
    ):
        super(DecodePredictions, self).__init__(**kwargs)
        self.num_classes = num_classes
        self.confidence_threshold = confidence_threshold
        self.nms_iou_threshold = nms_iou_threshold
        self.max_detections_per_class = max_detections_per_class
        self.max_detections = max_detections

        self._anchor_box = AnchorBox()
        self._box_variance = tf.convert_to_tensor(
            [0.1, 0.1, 0.2, 0.2], dtype=tf.float32
        )

    def _decode_box_predictions(self, anchor_boxes, box_predictions):
        boxes = box_predictions * self._box_variance
        boxes = tf.concat(
          [
             boxes[:, :, :2] * anchor_boxes[:, :, 2:] + anchor_boxes[:, :, :2],
```

```
                tf.math.exp(boxes[:, :, 2:]) * anchor_boxes[:, :, 2:],
            ],
            axis=-1,
        )
        boxes_transformed = convert_to_corners(boxes)
        return boxes_transformed

    def call(self, images, predictions):
        image_shape = tf.cast(tf.shape(images), dtype=tf.float32)
        anchor_boxes = self._anchor_box.get_anchors(image_shape[1],
image_shape[2])
        box_predictions = predictions[:, :, :4]
        cls_predictions = tf.nn.sigmoid(predictions[:, :, 4:])
    boxes = self._decode_box_predictions(anchor_boxes[None, ...],
             box_predictions)

     return tf.image.combined_non_max_suppression(
            tf.expand_dims(boxes, axis=2),
            cls_predictions,
            self.max_detections_per_class,
            self.max_detections,
            self.nms_iou_threshold,
            self.confidence_threshold,
            clip_boxes=False,
        )
```

გლუვი L1 დანაკარგის ფუნქციისა და ფოკალური დანაკარგის ფუნქციების იმპლემენტაცია Keras-ის ინდივიდუალური ფუნქციების სახით

```
class RetinaNetBoxLoss(tf.losses.Loss):
    """გლუვი L1 დანაკარგის ფუნქციის იმპლემენტაცია"""

    def __init__(self, delta):
        super(RetinaNetBoxLoss, self).__init__(
            reduction="none", name="RetinaNetBoxLoss"
        )
        self._delta = delta

    def call(self, y_true, y_pred):
        difference = y_true - y_pred
        absolute_difference = tf.abs(difference)
        squared_difference = difference ** 2
        loss = tf.where(
            tf.less(absolute_difference, self._delta),
            0.5 * squared_difference,
            absolute_difference - 0.5,
        )
        return tf.reduce_sum(loss, axis=-1)
```

```
class RetinaNetClassificationLoss(tf.losses.Loss):
    """ფოკალური დანაკარგის ფუნქციის იმპლემენტაცია"""

    def __init__(self, alpha, gamma):
        super(RetinaNetClassificationLoss, self).__init__(
            reduction="none", name="RetinaNetClassificationLoss"
        )
        self._alpha = alpha
        self._gamma = gamma
    def call(self, y_true, y_pred):
        cross_entropy = tf.nn.sigmoid_cross_entropy_with_logits(
            labels=y_true, logits=y_pred
        )
      probs = tf.nn.sigmoid(y_pred)
      alpha = tf.where(tf.equal(y_true, 1.0), self._alpha, (1.0 - self._alpha))
      pt = tf.where(tf.equal(y_true, 1.0), probs, 1 - probs)
      loss = alpha * tf.pow(1.0 - pt, self._gamma) * cross_entropy
      return tf.reduce_sum(loss, axis=-1)

class RetinaNetLoss(tf.losses.Loss):
    """შემფუთველი კლასი, ორი დანაკარგის ფუნქციის გასაერთიანებლად"""

    def __init__(self, num_classes=80, alpha=0.25, gamma=2.0, delta=1.0):
        super(RetinaNetLoss, self).__init__(reduction="auto",
name="RetinaNetLoss")
        self._clf_loss = RetinaNetClassificationLoss(alpha, gamma)
        self._box_loss = RetinaNetBoxLoss(delta)
        self._num_classes = num_classes

    def call(self, y_true, y_pred):
        y_pred = tf.cast(y_pred, dtype=tf.float32)
        box_labels = y_true[:, :, :4]
        box_predictions = y_pred[:, :, :4]
        cls_labels = tf.one_hot(
            tf.cast(y_true[:, :, 4], dtype=tf.int32),
            depth=self._num_classes,
            dtype=tf.float32,
        )
     cls_predictions = y_pred[:, :, 4:]
     positive_mask = tf.cast(tf.greater(y_true[:, :, 4], -1.0),
          dtype=tf.float32)
      ignore_mask = tf.cast(tf.equal(y_true[:, :, 4], -2.0), dtype=tf.float32)
      clf_loss = self._clf_loss(cls_labels, cls_predictions)
      box_loss = self._box_loss(box_labels, box_predictions)
      clf_loss = tf.where(tf.equal(ignore_mask, 1.0), 0.0, clf_loss)
      box_loss = tf.where(tf.equal(positive_mask, 1.0), box_loss, 0.0)
      normalizer = tf.reduce_sum(positive_mask, axis=-1)
      clf_loss = tf.math.divide_no_nan(tf.reduce_sum(clf_loss, axis=-1),
            normalizer)
```

```
        box_loss = tf.math.divide_no_nan(tf.reduce_sum(box_loss, axis=-1),
              normalizer)
        loss = clf_loss + box_loss
        return loss
```

### სასწავლო პარამეტრების მინიჭება

```
model_dir = "retinanet/"
label_encoder = LabelEncoder()

num_classes = 80
batch_size = 2

learning_rates = [2.5e-06, 0.000625, 0.00125, 0.0025, 0.00025, 2.5e-05]
learning_rate_boundaries = [125, 250, 500, 240000, 360000]
learning_rate_fn = tf.optimizers.schedules.PiecewiseConstantDecay(
    boundaries=learning_rate_boundaries, values=learning_rates
)
```

### მოდელის ინიციალიზაცია და კომპილაცია

```
resnet50_backbone = get_backbone()
loss_fn = RetinaNetLoss(num_classes)
model = RetinaNet(num_classes, resnet50_backbone)

optimizer = tf.optimizers.SGD(learning_rate=learning_rate_fn, momentum=0.9)
model.compile(loss=loss_fn, optimizer=optimizer)
```

### უკუგამოძახების ფუნქციების აღწერა

```
callbacks_list = [
    tf.keras.callbacks.ModelCheckpoint(
        filepath=os.path.join(model_dir, "weights" + "_epoch_{epoch}"),
        monitor="loss",
        save_best_only=False,
        save_weights_only=True,
        verbose=1,
    )
]
```

COCO2017 სასწავლო ნაკრების ჩატვირთვა TensorFlow Datasets-ის გამოყენებით

```
# სრული სასწავლო ნაკრების ჩასატვირთად - `data_dir=None`

(train_dataset, val_dataset), dataset_info = tfds.load(
    "coco/2017", split=["train", "validation"], with_info=True, data_dir="data"
)
```

### tf.data კონვეიერის გამართვა

რათა დავრწმუნდეთ, რომ მოდელს ეფექტურად მიეწოდება მონაცემები, ჩვენ გამოვიყენებთ tf.data პროგრამულ ინტერფეისს შემავალი კონვეიერის ასაგებად. შემავალი კონვეიერი შედგება შემდეგი ძირითადი ბიჯებისგან.

- გამოვიყენოთ პრეპარირების ფუნქცია ნიმუშებზე.

- შევქმნათ ფიქსირებული სიდიდის ჯგუფები. რადგანაც ჯგუფში არსებულ გამოსახულებებს შეიძლება ჰქონდეთ სხვადასხვა ზომები და სხვადასხვა რაოდენობის ობიექტები, ჩვენ გამოვიყენებთ padded_batch-ს მართკუთხა ჩარჩოების შესაქმნელად და მათი საჭიროებისამებრ შესავსებად.
- შევქმნათ მიზნის ვექტორები ჯგუფში არსებული თითოეული ნიმუშისათვის LabelEncoder-ის გამოყენებით.

```
autotune = tf.data.experimental.AUTOTUNE
train_dataset = train_dataset.map(preprocess_data, num_parallel_calls=autotune)
train_dataset = train_dataset.shuffle(8 * batch_size)
train_dataset = train_dataset.padded_batch(
    batch_size=batch_size, padding_values=(0.0, 1e-8, -1), drop_remainder=True
)
train_dataset = train_dataset.map(
    label_encoder.encode_batch, num_parallel_calls=autotune
)
train_dataset = train_dataset.apply(tf.data.experimental.ignore_errors())
train_dataset = train_dataset.prefetch(autotune)

val_dataset = val_dataset.map(preprocess_data, num_parallel_calls=autotune)
val_dataset = val_dataset.padded_batch(
    batch_size=1, padding_values=(0.0, 1e-8, -1), drop_remainder=True
)
val_dataset = val_dataset.map(label_encoder.encode_batch, num_parallel_calls=autotune)
val_dataset = val_dataset.apply(tf.data.experimental.ignore_errors())
val_dataset = val_dataset.prefetch(autotune)
```

მოდელის სწავლება

```
# სრულ სასწავლო ნაკრებზე სწავლების დროს, მოახდინეთ მომდევნო სტრიქონების დეკომენტირება.
# train_steps_per_epoch = dataset_info.splits["train"].num_examples // batch_size
# val_steps_per_epoch = \
#   dataset_info.splits["validation"].num_examples // batch_size

# train_steps = 4 * 100000
# epochs = train_steps // train_steps_per_epoch

epochs = 1

# გაშვება 100 სასწავლო და 50 სატესტო ბიჯზე,
# წაშალეთ `.take` სრულ სასწავლო ნაკრებზე სწავლების დროს.

model.fit(
    train_dataset.take(100),
    validation_data=val_dataset.take(50),
```

```
    epochs=epochs,
    callbacks=callbacks_list,
    verbose=1,
)
```

წონითი კოეფიციენტების ჩატვირთვა

```
# შეცვალეთ ქვემოთ მოცემული ცვლადი `model_dir`-ზე როდესაც არ იყენებთ
# გადმოწერილ წონით კოეფიციენტებს.
weights_dir = "data"

latest_checkpoint = tf.train.latest_checkpoint(weights_dir)
model.load_weights(latest_checkpoint)
```

გამომავალი მოდელის აგება

```
image = tf.keras.Input(shape=[None, None, 3], name="image")

predictions = model(image, training=False)
detections = DecodePredictions(confidence_threshold=0.5)(image, predictions)
inference_model = tf.keras.Model(inputs=image, outputs=detections)
```

ამოცნობის ვიზუალიზაცია

```
def prepare_image(image):
    image, _, ratio = resize_and_pad_image(image, jitter=None)
    image = tf.keras.applications.resnet.preprocess_input(image)
    return tf.expand_dims(image, axis=0), ratio


val_dataset = tfds.load("coco/2017", split="validation", data_dir="data")
int2str = dataset_info.features["objects"]["label"].int2str

for sample in val_dataset.take(2):
    image = tf.cast(sample["image"], dtype=tf.float32)
    input_image, ratio = prepare_image(image)
    detections = inference_model.predict(input_image)
    num_detections = detections.valid_detections[0]
    class_names = [
        int2str(int(x)) for x in detections.nmsed_classes[0][:num_detections]
    ]
    visualize_detections(
        image,
        detections.nmsed_boxes[0][:num_detections] / ratio,
        class_names,
        detections.nmsed_scores[0][:num_detections],
    )
```

## პრაქტიკული სამუშაო #7

### ამოცანა : ადამიანის პირისახის დეტექცია და ამოცნობა რეალურ დროში

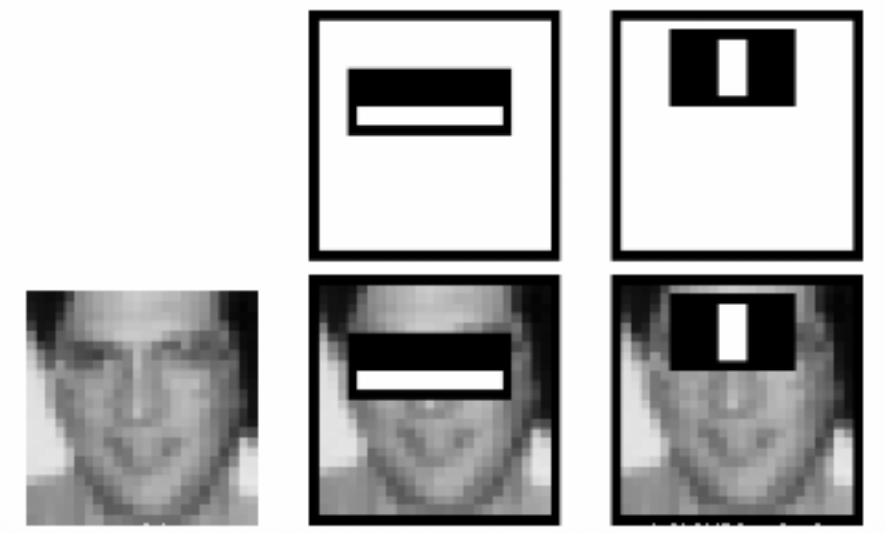
სამუშაო შესრულებულია C# პროგრამირების ენაზე, კოდში გამოყენებულია და ძირითადად ეყრდნობა ემგუ სვ-ის (Emgu cv), რომელიც არის “OpenCV” გამოსახულების დამუშავების ბიბლიოთეკის

გადამყვანი .Net პლათფორმის ენებზე. OpenCV მაღალი წარმადობის გამოსახულების დამუშავების ბიბლიოთეკა ინტელისგან.

ობიექტის დასანახად ვიყენებთ „Haar Feature-based Cascade Classifiers“ რომელიც იქნა შემოთავაზებული Paul Viola and Michael Jones -ის მიერ მათ ნაშრომში "Rapid Object Detection using a Boosted Cascade of Simple Features" 2001 წელს. ესა არის მანქნის სწავლებაზე დაფუძნებული ხერხი, სადაც კასკადური ფუნქცია არის ნასწავლი ბევრი დადებითი და უარყოფითი სურათებით. შემდეგ იგი გამოიყენება რათა აღმოვაჩინოთ უკვე ნასწავლი ობიექტი სხვა სურათებშიც.

ამ მეთოდის მუშაობის პრინციპია შეადაროს kernel-ის მიერ შექმნილი სხვადასხვა მახასიათებლები თეთრ პიქსელთა სიკაშკაშის საერთო მნიშვნელის და შავ პიქსელთა სიკაშკაშის საერთო მნიშვნელის სხვაობა, უკვე ნასწავლ ჩვენებებს.

სურათის დამუშავების პროცესში, იგი იყოფა 24x24 სიდიდის სურათებად და მათზე ხდება სათითაოდ ყველა features-ის შედარება, რომელსაც სწავლების პროცესსში ქონდა ყველაზე მაღალი აღმოჩენის წარმატობა.

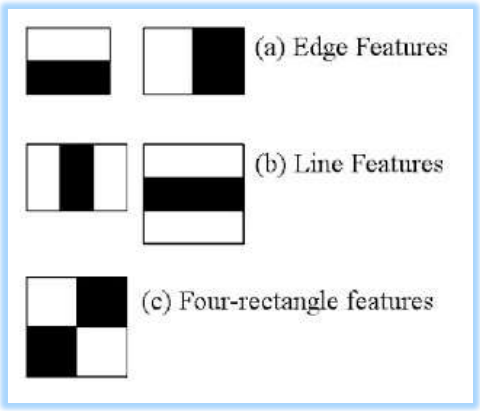


OpenCV თან მოყვება უკვე ნასწავლი ადამიანის სახეების ამოცნობის Haar Cascade Classifiers, რომელიც მოიცავს 6000 წარმატებულ features, რადგან ასეთი რაოდენობა ზედმეტად დიდი დროს წაიღებდა თითოეული 24x24 სურათის მონაკვეთის შესამოწმებლად, ავტორმა დაყო ეს features-ები 38 სთეიჯად, 1, 10, 25, 25 და 50features პირველი ხუთ სთეიჯში,

პირველი გაშვებისას, თუ არ იქნა მოძებნილი მანამდე ნასწავლი მასალა, გაგვიხსნის პირველ რიგში გაფრთხილების ფანჯარას, რომელიც შეგვატყობინებს, რომ სასწავლო მასალის ბიბლიოთეკა ცარიელია;

მას შემდეგ, რაც დავაკლიკავთ „OK” ღილაკზე, გაიხსნება ძირითადი სამუშაო ფანჯარა, სადაც შეგვიძლია ნახოთ:

1. ძირითადი გამოსახულების იმიჯბოქსი, რომელში ხდება კამერის საშუალებით მიღებული გამოსახულების გამოტანა, ასევე ამოცნობილი თუ ამოუცნელი სახეების ვიზუალურ ჩვენება.



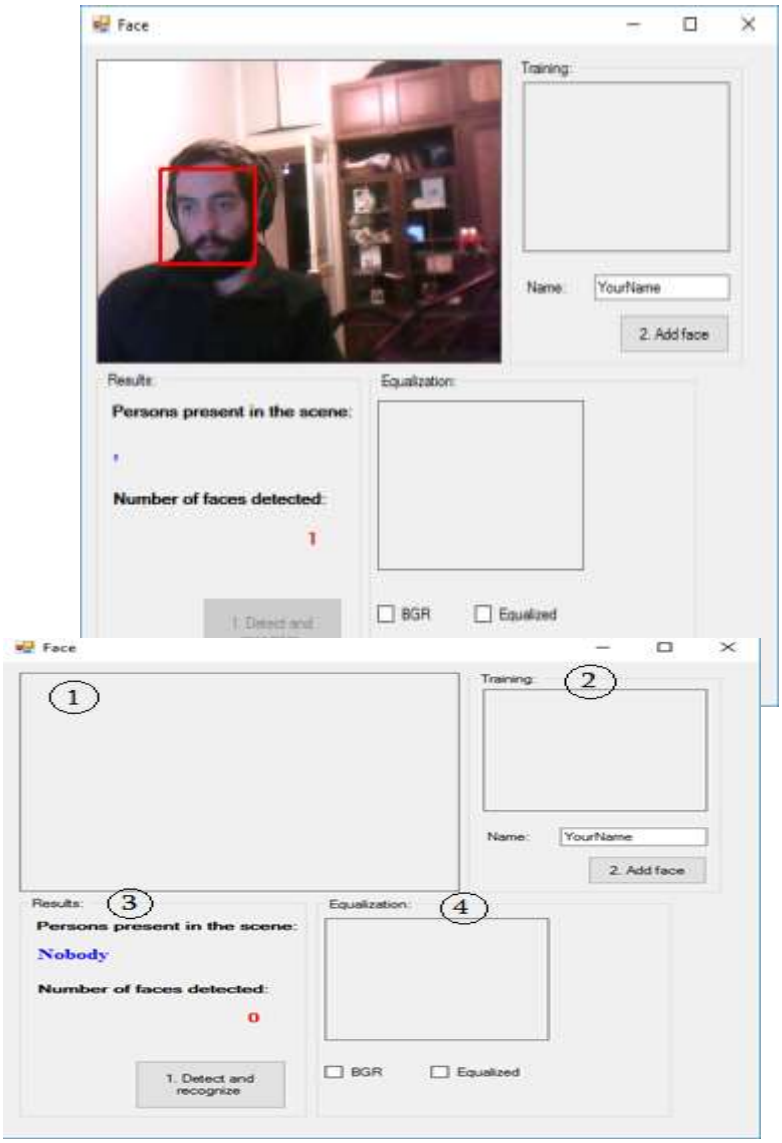
2. ტრეინიგ სექტორში, არის ნასწავლი სახის ვიზუალური საჩვენებელი იმიჯბოქსი და სახელის მისანიჭებელი ჩასაწერი ადგილი, და ღილაკი ახმოჩენილი სახის დასამახსოვრებლად.


3. რეზულტატების განყოფილება რომელშიც, გვიჩვენებს რამდენი პერსონა იკნა აღმოჩენილი და მათ სახელებს, ასევე მთავარი გამშვები ღილაკიც აქ არის მოთავსებული.

4. აქ შეგვიძლია სხვადასხვა ვიზუალური გამოსახულებით ვნახოთ სწავლებისთვის მომზადებული ფოტოსურათი.

ძირითადი ფანჯრის გახსნის შემდეგ, იმისთვის, რომ პროგრამა დაიწყოს მუშაობა და მიიღოს გამოსახულება კამერიდან უნდა დავაჭიროთ ღილაკს „1. Detect and Recognize", რის

შემდეგაც გააქტიურებს „იდელ ჰნდლერს" ფუნქცია FrameGrabber -ისთვის და გააიუქმებს ხელმეორე დაჭერისგან საკუთარ თავს.

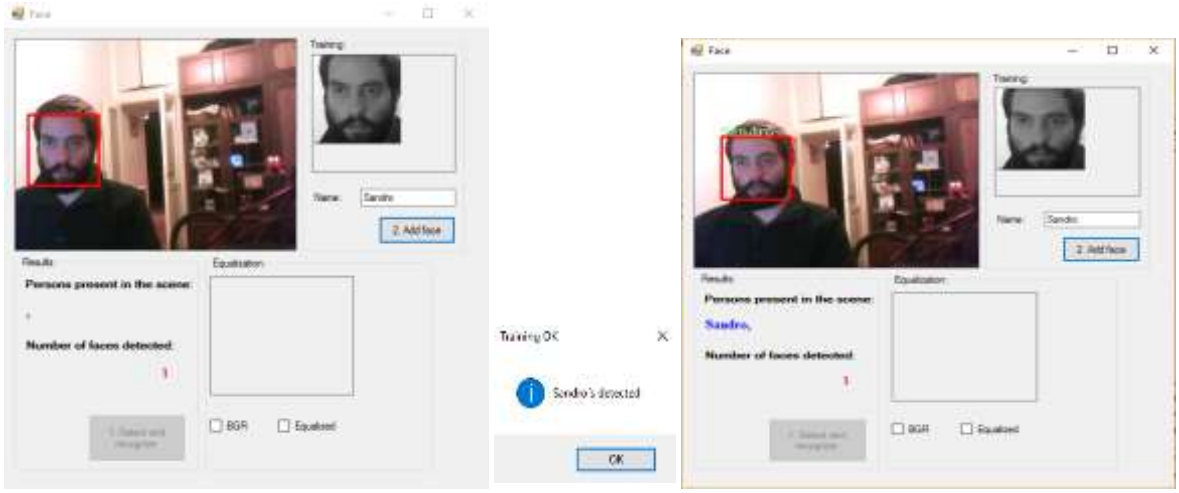



აპილიკაცია ყოველ ჯერზე, როდესაც თავისუფალია, იღებს ახალ კადრს, დაამუშვებს მას, haarcascade-ის მეშვეობით ცდილობს დაინახოს კონკრეტულ კადრში არის თუ არა ადამიანის პირდაპირი

სახე. თუ შეძლო დანახვა, იღებს კადრის იმ მონაკვეთს, სადაც სახე არის და გადაყავს 100 პიქსელი 100 პიქსელის ფორმატში რასაც გადასცემს EigenObjectRecognizer-ს თუ გვაქვს ერთი ფოტო შენახული ამოსაცნობი პირის;

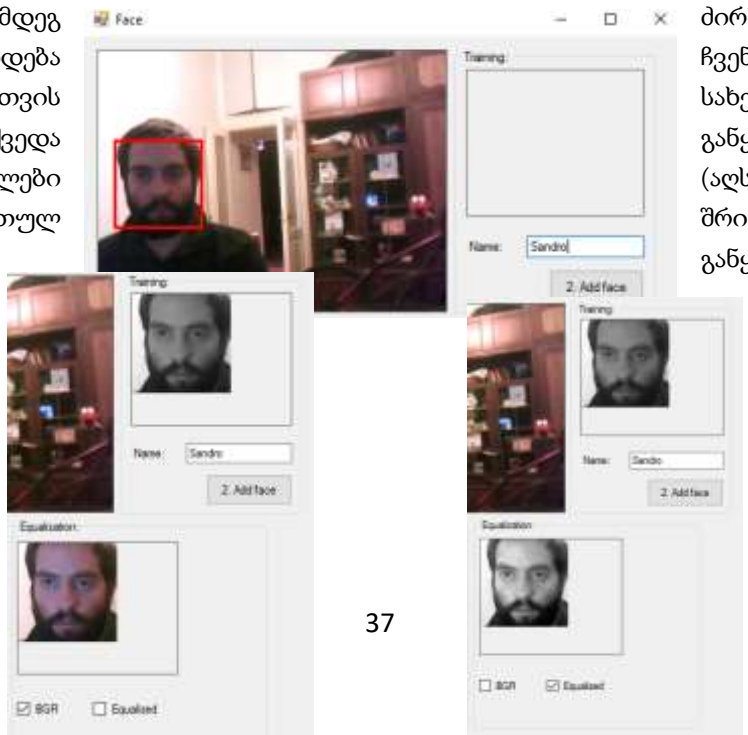
სწავლებისას Name: გრაფაში უნდა ჩავწეროთ ძირითად კამერაზე დანახული პირის სახელი და ამის შემდეგ დავაკლიკოთ “2. Add face” ღილაკს,

გაგვიხსნის ინფორმატორულ ფანჯარას და შეგვატყობინებს, რომ ჩვენს მიერ ჩატარებული სწავლება წარმატებით შესრულდა, ძირითად ფანჯარაში სწავლების გრაფაში გამოჩნდება ფოტო სურათი პირის რომელიც შეინახება ბაზაში. სწორად ამოცნობის დამაკმაყოფილებელი პროცენტის მისაღებად საჭიროა საშუალოდ 10-ჯერ მოხდეს ერთ ადამიანზე სწავლების ჩატარება, ასევე მიეცეს რამოდენიმე




მაგალითი რომლის ამოცნობაც არაა საჭირო.

ამის შემდეგ ფანჯარაში მოხდება ამოცნობილი სახისთვის კადრზე ასევე ქვედა იწერება სახელები კადრზე ქართულ ხერხდება და ქვედა




შესაძლებელია)

ძირითად გამოსახულების ჩვენის სახის ამოცნობა და სახელის დაწერა როგორც განყოფილებაში სადაც (აღსანიშნავია რომ შრიფტის ამოტანა ვერ განყოფილებაში

სურვილისამებრ, შეგვიძლია ექვილაიზე განყოფილებაში ვნახოთ ბოლო ნასწავლი სახის ბგრ და ექვილაიზდ გამოსახულებები, მომავალში ვგეგმავთ, რომ გამოსახულების სწავლება და ამოცნობა ხდებოდეს ექულიზედ გამოსახულებიდან თუ სწორად ამოცნობის ხარისხი იქნება უფრო მაღალი.

**შესაბამისი პროგრამული კოდი:**

```
using System;
usingSystem.Collections.Generic;
usingSystem.Drawing;
usingSystem.Windows.Forms;
using Emgu.CV;
usingEmgu.CV.Structure;
usingEmgu.CV.CvEnum;
using System.IO;
usingSystem.Diagnostics;
namespaceMultiFaceRec
{
public partial class FrmPrincipal : Form {
//Declararation of all variables, vectors and haarcascades
Capture grabber;
Image<Bgr, Byte>Cframe;
Image<Bgr, Byte>resultRGB, TrainedBgr  =null;

Image<Gray, byte>TestGray, result, TrainedFace = null;
Image<Gray, byte> gray = null;
HaarCascade face;
//HaarCascade eye;
MCvFont font = newMCvFont(FONT.CV_FONT_HERSHEY_TRIPLEX, 0.5d, 0.5d);
List<string> labels = newList<string>();
List<string>NamePersons = newList<string>();
List<Image<Gray, byte>>trainingImages = newList<Image<Gray, byte>>();
intContTrain, NumLabels, t;
string name, names = null;
publicFrmPrincipal()
   {
InitializeComponent();
//Load haarcascades for face detection
face = newHaarCascade("haarcascade_frontalface_default.xml");
//eye = new HaarCascade("haarcascade_eye.xml");
try
            {
//Load of previustrainned faces and labels for each image
stringLabelsinfo = File.ReadAllText(Application.StartupPath +
"/TrainedFaces/TrainedLabels.txt");
string[] Labels = Labelsinfo.Split('%');
NumLabels = Convert.ToInt16(Labels[0]);
ContTrain = NumLabels;
stringLoadFaces;
```

```
for (inttf = 1; tf< NumLabels+1; tf++)
                {
LoadFaces = "face" + tf + ".bmp";
trainingImages.Add(newImage<Gray, byte>(Application.StartupPath + "/TrainedFaces/"
+ LoadFaces));
labels.Add(Labels[tf]);
                }
            }
catch(Exception e){
//MessageBox.Show(e.ToString());
MessageBox.Show("Library is empty.", "No facees in Lib", MessageBoxButtons.OK,
MessageBoxIcon.Exclamation);
Directory.CreateDirectory(Application.StartupPath + "/TrainedFaces");
File.CreateText(Application.StartupPath + "/TrainedFaces/TrainedLabels.txt");
            }
    }
private void button1_Click(object sender, EventArgs e)
    {
//Initialize the capture device
grabber = newCapture();
grabber.QueryFrame();
//Initialize the FrameGraber event
Application.Idle += newEventHandler(FrameGrabber);
            button1.Enabled = false;    }
private void button2_Click(object sender, System.EventArgs e)
    {
DetectFace();
    }
public void DetectFace()
{ try {
//Trained face counter
ContTrain = ContTrain + 1;
//Get a gray frame from capture device
gray = grabber.QueryGrayFrame().Resize(320, 240,
Emgu.CV.CvEnum.INTER.CV_INTER_CUBIC);
//Face Detector
MCvAvgComp[][] facesDetected = gray.DetectHaarCascade(
face, 1.2,10,
Emgu.CV.CvEnum.HAAR_DETECTION_TYPE.DO_CANNY_PRUNING,
newSize(20, 20));
//Action for each element detected
foreach (MCvAvgComp f infacesDetected[0])
                {
TrainedFace = Cframe.Copy(f.rect).Convert<Gray, byte>();
TrainedBgr = Cframe.Copy(f.rect);
break;
                }
//resize face detected image for force to compare the same size with the
//test image with cubic interpolation type method
TrainedFace = result.Resize(100, 100, Emgu.CV.CvEnum.INTER.CV_INTER_CUBIC);
trainingImages.Add(TrainedFace);
labels.Add(textBox1.Text);
//Show face added in gray scale
imageBox1.Image = TrainedFace;
// resize Bgr face Detected Image
TrainedBgr = resultRGB.Resize(100, 100, Emgu.CV.CvEnum.INTER.CV_INTER_CUBIC);
//Write the number of triained faces in a file text for further load
File.WriteAllText(Application.StartupPath + "/TrainedFaces/TrainedLabels.txt",
trainingImages.ToArray().Length.ToString() + "%");
```

```
//Write the labels of triained faces in a file text for further load
for (inti = 1; i<trainingImages.ToArray().Length + 1; i++)
                    {
trainingImages.ToArray()[i - 1].Save(Application.StartupPath + "/TrainedFaces/face"
+ i + ".bmp");
File.AppendAllText(Application.StartupPath + "/TrainedFaces/TrainedLabels.txt",
labels.ToArray()[i - 1] + "%");
                    }
MessageBox.Show(textBox1.Text + "´s detected", "Training OK", MessageBoxButtons.OK,
MessageBoxIcon.Information);
                }
catch(Exception ex )
                {
MessageBox.Show("Face detection exception: " + ex, "Training Fail",
MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
                }}
voidFrameGrabber(object sender, EventArgs e) {
label3.Text = "0";
//label4.Text = "";
NamePersons.Add("");
//Get the current frame form capture device
Cframe = grabber.QueryFrame().Resize(320, 240,
Emgu.CV.CvEnum.INTER.CV_INTER_CUBIC);
//Convert it to Grayscale
gray = Cframe.Convert<Gray, Byte>();
//Face Detector
MCvAvgComp[][] facesDetected = gray.DetectHaarCascade(
face, 1.2,10,
Emgu.CV.CvEnum.HAAR_DETECTION_TYPE.DO_CANNY_PRUNING,
newSize(20, 20));
//Action for each element detected
foreach (MCvAvgComp f infacesDetected[0])
   {
      t = t + 1;
result = Cframe.Copy(f.rect).Convert<Gray, byte>().Resize(100, 100,
Emgu.CV.CvEnum.INTER.CV_INTER_CUBIC);
resultRGB = Cframe.Copy(f.rect).Resize(100, 100,
Emgu.CV.CvEnum.INTER.CV_INTER_CUBIC);
//draw the face detected in the 0th (gray) channel with blue color
Cframe.Draw(f.rect, newBgr(Color.Red), 2);
if (trainingImages.ToArray().Length != 0) {
//TermCriteria for face recognition with numbers of trained images like
maxIteration
MCvTermCriteriatermCrit = newMCvTermCriteria(ContTrain, 0.001);
//Eigen face recognizer
EigenObjectRecognizer recognizer = newEigenObjectRecognizer(
trainingImages.ToArray(),
labels.ToArray(),3000,reftermCrit);
name = recognizer.Recognize(result);
//Draw the label for each face detected and recognized
Cframe.Draw(name, ref font, newPoint(f.rect.X - 2, f.rect.Y - 2),
newBgr(Color.LightGreen));
                        }
NamePersons[t-1] = name;
NamePersons.Add("");
//Set the number of faces detected on the scene
              label3.Text = facesDetected[0].Length.ToString();
       }
         t = 0;
//Names concatenation of persons recognized
```

```
for (intnnn = 0; nnn<facesDetected[0].Length; nnn++)
          {
names = names + NamePersons[nnn] + ", ";
           }
//Show the faces procesed and recognized
imageBoxFrameGrabber.Image = Cframe;
           label4.Text = names;
names = "";
// Equalization image processing Testing
//Gray Checked
if (CHBGray.Checked&& !CHBEqualized.Checked)
           {
imageBoxGray.Image = TrainedBgr;
           }
//Equalized Checked
if (CHBEqualized.Checked&& !CHBGray.Checked  )
           {
   TestGray = TrainedFace;
   TestGray._EqualizeHist();
   imageBoxGray.Image = TestGray;
           }
//Clear the list(vector) of names
NamePersons.Clear(); }
private void CHBGray_CheckedChanged(object sender, EventArgs e)
    { if (CHBGray.Checked)
       { CHBEqualized.Checked = false;   }
    }
private void CHBEqualized_CheckedChanged(object sender, EventArgs e)
    {  if (CHBEqualized.Checked)
      {  CHBGray.Checked = false;   }
    }
  }
}
```

## პრაქტიკული სამუშაო #9

### ამოცანა: ციფრული გამოსახულების ფილტრაცია

სამუშაო სრულდება პროგრამა Matlab-ის გამოყენებით.

გამოსახულების ფილტრაცია (5*5) განზომილების გამასაშუალებელი ფილტრის საშუალებით - averaging filter

I=imread('coins.png');

H=ones(5,5)/25

I2=imfilter(I,h);

Imshow(I), title('Original Image');

Figure, imshow (I2), title ('filtered Image').

Analyzing and Enhacing Images

Removing Noise from Image - ახდენს გამოსახულებიდან რამდენიმე ტიპის ხმაურის მოშორებას, გამოსახულების გაწმენდას ხმაურისგან.

I=imnoise(I, type), სადაც type არის სტრინგი კონკრეტული ხმაურის მითითებით:

'gaussian' - გაუსის თეთრი ხმაური მუდმივი მათ. მოლ-ით და დისპერსიით.

'salt & peper'

J=imnoise (I, type, parameters) - გამოსახულებას ადებს ხმაურს და ასევე საზღვრავს დამატებით პარამეტრს, მაგალითად, სიკაშკაშის დიაპაზონს 0-დან 1-მდე.

J=imnoise (I, 'salt & peper', d) – „მარილი და წიწაკა" ტიპის ხმაურის დადება I გამოსახულებაზე, სადაც d ხმაურის სიმკვრივეა. საზოგადოდ, d=0.05, 0.02 და სხვა. განვიხილოთ ხმაურის სხვადასხვა სიმკვრივეები.

**განვიხილოთ მაგალითი:**

```
I=imread('eight.tif ');
J=imnoise (I, 'salt & pepper', 0.02)
Figure, imshow (I)
Figure, imshow (J)
```

ხშირად ხმაურის მოსაცილებლად იყენებენ მედიანურ ფილტრს, რომელიც გამასაშუალებელი ფილტრის მსგავსია. მედიანური ფილტრი წარმოადგენს არაწრფივ ოპერატორს. ამისთვის გამოიყენება ბრძანებები

B=medfilt2 (A, [m n]) – A მატრიცაზე ორგანზომილებიანი მედიანური ფილტრის გამოყენება.

B=medfilt2 (A) – (3*3) განზომილების მედიანური ფილტრის გამოყენება A მატრიცაზე.

B=medfilt2 (A,'indexed', …) - მედიანური ფილტრის გამოყენება ინდექსირებულ A გამოსახულებაზე, სადაც ... ადგილას იწერება 0 ან 1, იმის მიხედვით, თუ როგორია A მატრიცის კლასი - Unit8 თუ double.

იმ შემთხვევაში, თუ მედიანის მნიშვნელობა წილადია, მაშინ მას ჩამოაცილებენ წილად ნაწილს. თუ ის შეიცავს ყველა ნულზე გადასვლის წერტილებს.

**მაგალითი:**

„მარილი და წიწაკა“ ტიპის ხმაურისგან გამოსახულების გაწმენდა

```
I=imread('eight.tif ');
J=imnoise (I, 'salt & pepper', 0.02);
K=medfilt2(J)
imshow (J), Figure, imshow (K)
```

გამასაშუალოებელი და მედიანური ფილტრების შედარების მაგალითი:

ხმაურის ტიპი: 'salt & pepper', გამოიყენება 2-განზომილებიანი (3*3) ფილტრი

```
I=imread('eight.tif ');
imshow (I)
```

„მარილი და წიწაკა“ ტიპის ხმაურის დადება გამოსახულებაზე

```
J=imnoise (I, 'salt & pepper', 0.02);
Figure, imshow (J)
```

გამასაშუალოებელი ფილტრის გამოყენება გამოსახულების გასასუფთავებლად ხმაურისაგან

```
K=filter2(fspecial ('average', 3), J)/255;
Figure, imshow (K)
```

მედიანური ფილტრის გამოყენება გამოსახულების ხმაურისგან გასაწმენდად

```
L=medfilter2(J, [3 3]);
Figure, imshow (L)
```

H=fspecial(type) - ქმნის წინასწარ გასაზღვრულ სპეციფიურ 2D ფილტრს h, რომლის გამოსასვლელი გამოიყენება imfilter ფუნქციის რეალიზაციისთვის

H=fspecial(type, parameters)

გამოსახულებაზე ხმაურის მოშორება ადაპტური ფილტრის გამოყენებით - Removing noise by adaptive filtering. ამისათვის გამოიყენება ვინერის ფილტრი, რომელიც წარმოადგენს დაბალი სიხშირის ფილტრს, რომელიც ემყარება მოცემული პიქსელის მიდამოში (ფანჯარაში) არსებული სტატისტიკების შეფასებას - მოცემული პიქსელის მიდამოში ლოკალური საშუალოს და დისპერსიის შეფასებას.

**მაგალითი:**

ვთქვათ, გვაქვს რაიმე გამოსახულება

```
I=imread('coins.png');
```

Figure, imshow (I)
დავადოთ მას გაუსის ხმაური შესაბამისი პარამეტრით (ან სხვა ხმაური)
J=imnoise (I, ‘gaussian’, 0, 0.025);
Figure, imshow(J);
მოვაშოროთ ეს ხმაური ვინერის ფილტრის გამოყენებით
K=wiener2(J, [5 5]);
Figure, imshow(K);

## პრაქტიკული სამუშაო #10

### ამოცანა 1: ადამიანის სქესის ამოცნობა სახელის მიხედვით

საიტი შედგება ორი გვერდისაგან: main.html და process.html
main.html ში შეგვყავს სახელი და საბმიტ ღილაკზე დაწოლისას ეს სახელი გადაეცემა process.html გვერდს. რომელიც localStorage ში ნახულობს, თუ ნაცნობი სახელია, გამოაქვს სქესი, წინააღმდეგ შემთხვევაში ხდება სქესის მითითება მომხმარებლის მიერ და localStorage ში შენახვა.
main.html

```
<html>
<body>
<form action="process.html" method="get">
        <h1 align="center">შემოიტანეთ სახელი</h1>
        <p align="center"><input type="text" name="firstname"></p>
        <p align="center"><input type="submit" value="Ok"></p>
</form>
</body>
</html>
```

process.html

```
<html>
<head>
<script>
var objNames = [];
var fn = "";
function indexOf(name)
{
        n=-1;
        for (i = 0; i < objNames.length; i++)
        {
                if (objNames[i].firstname == name)
                {
                        n = i;
                        break;
                }
        }
        return n;
}
function myStart(){
        var gender = "უცნობია";
        var urlParams = new URLSearchParams(window.location.search);
        fn = urlParams.get("firstname");
        document.getElementById("firstname").innerHTML = fn;
        var json = localStorage.getItem("names");
        if (json == null || json == "null")
```

```
        {
                objNames = [{firstname:fn, gender:"უცნობია"}];
        }
        else
        {
                objNames = JSON.parse(json);
                index = indexOf(fn);
                if (index == -1)
                {
                        objNames.push({firstname:fn, gender:"უცნობია"});
                }
                else
                {
                        gender = objNames[index].gender;
                }
        }
        localStorage.setItem("names", JSON.stringify(objNames));
        document.getElementById("outputGender").innerHTML = gender;
}
function select2()
{
        var  gender = "უცნობია";
        index = indexOf(fn);
        if(document.all.gender[0].checked){
                gender = "მამრობითი";
        }
        if(document.all.gender[1].checked){
                gender = "მდედრობითი";
        }
        objNames[index].gender = gender;
        localStorage.setItem("names", JSON.stringify(objNames));
        document.getElementById("outputGender").innerHTML = gender;
}
function clear2()
{
        localStorage.setItem("names", null);
}
</script>
</head>
<body onload="myStart()">
<p id="firstname"></p>
<input type="radio" name="gender">მამრობითი<br/>
<input type="radio" name="gender">მდედრობითი<br/>
<input type="button" value="აირჩიე" onclick="select2()"/>
<input type="button" value="გაასუფთავე" onclick="clear2()"/>
<p id="outputGender"></p>
<a href="main.html">უკან</a>
</body>
</html>
```

### ამოცანა 2: ტექსტის თემატიკის დადგენა გულუბრყვილო ბაიესის ალგორითმის გამოყენებით

ამოცანა სრულდება Python დაპროგრამების ენაზე.

სიტყვების გამოყოფა:

```
1. def extract_tweet_words(tweet_words):
2.       words = []
3.       alpha_lower = string.ascii_lowercase
4.       alpha_upper = string.ascii_uppercase
5.       numbers = [str(n) for n in range(10)]
6.       for word in tweet_words:
7.       cur_word = ''
8.       for c in word:
9.       if (c not in alpha_lower) and (c not in alpha_upper) and (c not in numbers):

10.                 if len(cur_word) >= 2:
11.               words.append(cur_word.lower())
12.            cur_word = ''
13.             continue
14.             cur_word += c
15.       if len(cur_word) >= 2:
16.       words.append(cur_word.lower())
17. return words
```

მივიღოთ ტრენინგის მონაცემები tweet– იდან.

```
1. def get_tweet_training_data():
2.          f = open('training.txt', 'r')
3.          training_data = []
4.          for l in f.readlines():
5.              l = l.strip()
6.              tweet_details = l.split()
7.              tweet_id = tweet_details[0]
8.              tweet_label = tweet_details[1]
9.              tweet_words = extract_words(tweet_details[2:])
10.             training_data.append([tweet_id, tweet_label, tweet_words])
11.         f.close()
12. return training_data
```

მივიღოთ სატესტო მონაცემები, მათი კლასიფიცირების მიზნით

```
1. def get_tweet_test_data():
2.      f = open('test.txt', 'r')
3.      validation_data = []
4.      for l in f.readlines():
5.          l = l.strip()
6.          tweet_details = l.split(' ')
7.          tweet_id = tweet_details[0]
8.          tweet_words = extract_words(tweet_details[1:])
9.          validation_data.append([tweet_id, '', tweet_words])
10.
11.     f.close()
12.
```

```
13. return validation_data
```

მივიღოთ სიტყვების სია ტრენინგის მონაცემებში.

```
1.  def get_words(training_data):
2.      words = []
3.      for data in training_data:
4.          words.extend(data[2])
5.      return list(set(words))
```

მივიღოთ თითოეული სიტყვის ალბათობა tweet– ის ტრენინგის მონაცემებში.

```
1.  def get_tweet_word_prob(training_data, label = None):
2.      words = get_words(training_data)
3.      freq = {}
4.
5.      for word in words:
6.          freq[word] = 1
7.
8.      total_count = 0
9.      for data in training_data:
10.         if data[1] == label or label == None:
11.             total_count += len(data[2])
12.             for word in data[2]:
13.                 freq[word] += 1
14.
15.     prob = {}
16.     for word in freq.keys():
17.         prob[word] = freq[word]*1.0/total_count
18.
19.     return prob
```

მივიღოთ მოცემული იარლიყის ალბათობა.

```
1.  def get_tweet_label_count(training_data, label):
2.      count = 0
3.      total_count = 0
4.      for data in training_data:
5.          total_count += 1
6.          if data[1] == label:
7.              count += 1
8.      return count*1.0/total_count
```

გამოვიყენოთ Naive Bayes მოდელი, როგორც აღწერილია ქვემოთ.

```
1.  def label_tweet_data(test_data, sports_word_prob, politics_word_prob, sports_prob
    , politics_prob):
2.      labels = []
3.      for data in test_data:
4.          data_prob_sports = sports_prob
5.          data_prob_politics = politics_prob
6.
```

```
        for word in data[2]:
            if word in sports_word_prob:
                data_prob_sports *= sports_word_prob[word]
                data_prob_politics *= politics_word_prob[word]
            else:
                continue

        if data_prob_sports >= data_prob_politics:
            labels.append([data[0], 'Sports', data_prob_sports, data_prob_politic
s])
        else:
            labels.append([data[0], 'Politics', data_prob_sports, data_prob_polit
ics])

    return labels
```

დვაბეჭდოთ ეტიკეტიანი ან კატეგორიზირებული ტესტის მონაცემები

```
def print_labelled_data(labels):
    f_out = open('test_labelled_output.txt', 'w')
    for [tweet_id, label, prob_sports, prob_politics] in labels:
        f_out.write('%s %s\n' % (tweet_id, label))

    f_out.close()

```

წავიკითხოთ ტრენინგისა და ტესტის მონაცემები

```
training_data = get_tweet_training_data()
test_data = get__tweet_test_data()
```

მივიღოთ ყოველი სიტყვის ალბათობა

```
word_prob = get_tweet_word_prob(training_data)
sports_word_prob = get_tweet_word_prob(training_data, 'Sports')
politics_word_prob = get_tweet_word_prob(training_data, 'Politics')
```

მივიღოთ თითოეული ეტიკეტის ალბათობა.

```
sports_prob = get_tweet_label_count(training_data, 'Sports')
politics_prob = get_tweet_label_count(training_data, 'Politics')
```

ნორმალიზება სიტყვებისთვის.

```
for (word, prob) in word_prob.items():
    sports_word_prob[word] /= prob
    politics_word_prob[word] /= prob
```

ტესტის მონაცემების ეტიკეტირება და დაბეჭდვა.

```
test_labels = label_tweet_data(test_data, sports_word_prob, politics_word_prob, s
ports_prob, politics_prob)
```

```
2. print_labelled_data(test_labels)
```

## რეკომენდებული ლიტერატურა:


1. ხელოვნური ინტელექტის დეპარტამენტი, ხელოვნური ინტელექტი. 2018; CD-4208;
2. Jussi Tohka. Introduction to Pattern Recognition. 2013. (CD - 2891)

Graham Kendall, Artificial Neural Network (manual) - 2016; CD - 4208

3. Stuart Russel, Peter Norwig, Artificial Intelligence: A Modern Approach (3th Edition), Pearson Education Limited, 2014.
4. Bernard Marr, “Artificial Intelligence in Practice”, Wiley, 2019



გადაეცა წარმოებას 12..11.2020. ხელმოწერილია დასაბეჭდად 20.11.2020.
ოფსეტური ქაღალდის ზომა 60X84 1/16. პირობითი ნაბეჭდი თაბახი 3. ტირაჟი 50 ეგზ.

სტუ-ს „IT კონსალტინგის სამეცნიერო ცენტრი“,
თბილისი, მ.კოსტავას 77

ISBN 978-9941-8-2866-9