# *Beadandó dolgozat*

## *1. Koncepció/elvárás:*

Dolgozatom témájának egy olyan szoftvert fogok bemutatni amely, egy gépi látáshoz kapcsolódó problémát old meg, de mit is takar pontosan a gépi látás, mint informatikai terület?

Lényegében egy általános gyűjtőfogalom eljárásokra és rendszerekre, amelyekkel (mozgó) kép alapú adatgyűjtés és –kiértékelés után vagy annak hatására valamilyen vezérlési, szabályozási vagy gépi értelmezési mechanizmus indul be. Tehát ez egyfajta mesterséges intelligencia, amely például a képfeldolgozás területén alkalmaznak. Az én feladatom is ebben a képfelismerő témakörben van a gépi látáson belül, de emellett számos területen megjelenik még, mint például a mozgókép feldolgozás, ahol ezek az algoritmusok szintén felismerő feladatokat töltenek be.

Vissza kanyarodva a dolgozatom témájához, ez nem lesz más, mint egy pénzérme felismerő alkalmazás. Jelen esetben euro pénzérméket ismer fel és emellett értékel ki. Leegyszerűsítve a program egy képről képes felismerni a pénzérméket és azokat nem csak detektálja, hanem az értéküket is képes megmondani. Tehát input adatként kap egy képet melyen valahány darab pénzérme található, majd kimenetként kapunk egy olyan képet, amelyen megjelölve vannak a címletek az értékükkel együtt.

Ez a következő képpen néz ki:


*1. ábra input-output megjelenés: https://raw.githubusercontent.com/j05t/coin_detector/master/screenshot.png*

Látható, hogy a szoftver több címletű érmét is képes egyszerre felismerni és azokat jól detektálni, még azokon a helyeken is ahol az érmék össze érnek vagy éppen több ugyan olyan helyezkedik el egymás mellett. Ezen felül még képes az 1 centestől egészen a 2 eurós érméig mindent pontosan felismerni, azaz a forgalomban lévő összes eurós érmét, ez a mellékelt képen is jól látszik.

## 2. *Fejlesztői dokumentáció:*

1. ***Követelmények:*** A szoftvernek egy input kép alapján fel kell tudni ismerni a pénzérméket, úgy, hogy több pénzérme is szerepel a képen jelen esetben egy adott ország fizető eszköze, az egységesség érdekében. Illetve az összeérő tehát nem teljesen külön álló objektumok felismerésének a képességével is rendelkeznie kell, végül pedig az output képen a keresett pénz jelölése majd az érétkük feltüntetése.

2. ***Futási környezet:*** asztali vagy hordozható számítógépen biztosított a futása Windows és Linux környezetben, speciális hardver igénye nincsen.

3. ***Használt program nyelv:*** Python 3.7 és Thonny IDE és a következő beépített könyvtárak: math, numpy as np, argparse, glob, cv2, train_test_split, MLPClassifier

4. ***Használt algoritmusok és a program ismertetése:*** A programunk tehát egy pénzérme felismerő és kiértékelő program.

   ***Az 1.-8.-ik sorig*** a szükséges beépített könyvtárak importálása látható.

```
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split

import math
import numpy as np
import argparse
import glob
import cv2
```

*2. ábra 1.-8.-ik sor*

   Ezek jelentése:

   - MLPClassifer: Ez a modell optimalizálja a log-loss funkciót.
   - train_test_split: Split tömbök vagy mátrixok random teszt alcsoportja
   - math: Szabvány matematikai funkciók biztosítása
   - numpy: Tudományos számítástechnikai alapcsomag
   - argparse: Felhasználóbarát parancssori interfészek írását teszi lehetővé.
   - cv2: OpenCV elérését biztosítja

   ***A 9.-14.-ik sorig*** a tervező és építő argumentumok kialakítása, mint a felhasznált kép importálása a fejlesztői környezetbe és annak a kezelése.

```
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--image", required=True, help="path to image")
args = vars(ap.parse_args())

image = cv2.imread(args["image"])
```

*3. ábra 9.-14.-ik sor*

***A 15.-18.-ik sorig*** a szoftver számára adott kép átméretezése történik a képarányok megtartása mellett, mivel így egyszerűbb a munka és így mindig fix méretűre szorítja be a program az éppen kapott képet.

```
d = 1024 / image.shape[1]
dim = (1024, int(image.shape[0] * d))
image = cv2.resize(image, dim, interpolation=cv2.INTER_AREA)
```

*4. ábra 15.-18.-ik sor*

***A 19. és 20. programsorban*** az output kép gyanánt készítünk, egy másolatot majd a képet manipuláljuk egész pontosan a feldolgozását egy szürke árnyalatossá konvertálással kezdjük meg.

```
output = image.copy()
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

*5. ábra 19. és 20. sor*

***A 20.-26.-ig sorig*** több fontosabb mellék manipulációt készítünk el. Először is a fényviszonyoknak megfelelően kontrasztot állítunk be, majd, egy CLAHE objektumot hozunk létre a kontrasztkorlátozó adaptív hisztogramkiegyenlítés alkalmazására és végül egy maszkot is az eredeti képhez a detektáláshoz.

```
output = image.copy()
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)


clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
gray = clahe.apply(gray)
def calcHistogram(img):
    m = np.zeros(img.shape[:2], dtype="uint8")
    (w, h) = (int(img.shape[1] / 2), int(img.shape[0] / 2))
    cv2.circle(m, (w, h), 60, 255, -1)
```

*6. ábra 20.-26. sor*

***A 27.-31.-ig sorig*** első körben a calcHist nevű függvényünknek paraméterként át kell adnunk a képet, a szín csatornát, a maszkot, és magát a tartományt is. Majd vissza adjuk a normalizált hisztogrammot.

```
    return cv2.normalize(h, h).flatten()
def calcHistFromFile(file):
    img = cv2.imread(file)
    return calcHistogram(img)
```

*7. ábra 27.-31. sor*

A következő lényeges dolog a ***33.-37.-ik sorig*** történik, ahol is felsoroljuk azokat az anyagtípusokat, amelyet a programnak ismerni kell és megadjuk, azokat a fájlokat ahol ezek a minta anyagok vannak a szoftver számára.

```
Material = Enum(('Copper', 'Brass', 'Euro1', 'Euro2'))
sample_images_copper = glob.glob("sample_images/copper/*")
sample_images_brass = glob.glob("sample_images/brass/*")
sample_images_euro1 = glob.glob("sample_images/euro1/*")
sample_images_euro2 = glob.glob("sample_images/euro2/*")
```

*8. ábra 33.-37. sor*

Majd a ***40.-51.-ik sorig*** kiszámoljuk és eltároljuk az egyes anyagokhoz tartozó labeleket melyeket a korábban létrehozott x és y tömbben fogunk eltárolni egy for ciklus segítségével.

```
for i in sample_images_copper:
    X.append(calcHistFromFile(i))
    y.append(Material.Copper)
for i in sample_images_brass:
    X.append(calcHistFromFile(i))
    y.append(Material.Brass)
for i in sample_images_euro1:
    X.append(calcHistFromFile(i))
    y.append(Material.Euro1)
for i in sample_images_euro2:
    X.append(calcHistFromFile(i))
    y.append(Material.Euro2)
```

*9. ábra 40.-51. sor*

Majd a képen alkalmazzuk a Gaussian elmosódást ott ahol a kép pixelei közelebb álnak a középponthoz és az első argumentumunk maga a forráskép a második pedig a kernel mérete lesz a harmadik pedig a szigma, ami 0 az OpenCV automatikus érzékeléséhez. Ez a következő képpen néz ki a gyakorlatban.

```
blurred = cv2.GaussianBlur(gray, (7, 7), 0)
```

*10. ábra Gaussian elmosódás*

Ezt a következő sor követi:

```
circles = cv2.HoughCircles(blurred, cv2.HOUGH_GRADIENT, dp=2.2, minDist=100,
                           param1=200, param2=100, minRadius=50, maxRadius=120)
```

*11. ábra felismerő objektum*

Itt a következő dolog történik: a circles gyakorlatilag egy olyan vektor lesz ami x,y és r koordinátákat eltárol minden egyes felismert kör objektumhoz, ami nálunk a pénzérmék, hiszen az érmék is kör alakúak. Továbbá a képünk már szürke árnyalatos, a cv_hough segít abban, hogy meghatározza az észlelési módot, a dp a felbontás inverz arányát adja, meg míg a minDist az észlelt középpontok közti minimális távolság, a param1 és param2 a belső Canny szélérzékelőnek a felső küszöbértéke és a küszöbértéket tartalmazza a középpont érzékeléséhez. Végül pedig a minRadius és maxRadius ami az érmék minimális és maximális sugarának a méretét tartalmazzák.

Az anyag, amit érzékelünk és egész pontosan annak az anyagának úgymond előrejelzésére készítünk egy függvényt, ami a következő képpen néz ki:

```
def predictMaterial(roi):
    hist = calcHistogram(roi)
    s = clf.predict([hist])
    return Material[int(s)]
```

*12. ábra 'anyag' érzékelése*

Majd ***136.-176.-ik sorig*** több dolgot is végre hajtunk:

- hozzá adjuk a sugarakat az ennek a célnak fenntartott lista elemeihez
- koordináták és a sugarak konvertálása egész számokká
- koordinátákat adunk a listához
- a nem szükséges régiókat kivonjuk a képből
- anyagtípus felismerés és annak a listázása
- hozzád az érmékről maszkot a fájlunkhoz ergo készül egy maszk az érmékről
- a detektált érmékhez kontúr vonalat rajzolunk és kezd össze állni a kimeneti képünk.
- végül pedig át adjuk a legnagyobb átmérőt az érmék közül, ami jelen esetben a 2 eurós lesz.

```
diameter = []
materials = []
coordinates = []
count = 0
if circles is not None:

    for (x, y, r) in circles[0, :]:
        diameter.append(r)
    circles = np.round(circles[0, :]).astype("int")
    for (x, y, d) in circles:
        count += 1
        coordinates.append((x, y))
        roi = image[y - d:y + d, x - d:x + d]
        material = predictMaterial(roi)
        materials.append(material)
        if False:
            m = np.zeros(roi.shape[:2], dtype="uint8")
            w = int(roi.shape[1] / 2)
            h = int(roi.shape[0] / 2)
            cv2.circle(m, (w, h), d, (255), -1)
            maskedCoin = cv2.bitwise_and(roi, roi, mask=m)
            cv2.imwrite("extracted/01coin{}.png".format(count), maskedCoin)
        cv2.circle(output, (x, y), d, (0, 255, 0), 2)
        cv2.putText(output, material,
                    (x - 40, y), cv2.FONT_HERSHEY_PLAIN,
                    1.5, (0, 255, 0), thickness=2, lineType=cv2.LINE_AA)
biggest = max(diameter)
i = diameter.index(biggest)
```

*13. ábra 163.-176. sor*

***180.-201.-ik sorig*** méretezünk mindent a legnagyobb átmérőjű érméhez viszonyítva, hogy később az értékük meghatározása könnyebb legyen, ezt persze valamilyen szinten a felhasználó fogja befolyásolni.

```
if materials[i] == "Euro2":
    diameter = [x / biggest * 25.75 for x in diameter]
    scaledTo = "Scaled to 2 Euro"
elif materials[i] == "Brass":
    diameter = [x / biggest * 24.25 for x in diameter]
    scaledTo = "Scaled to 50 Cent"
elif materials[i] == "Euro1":
    diameter = [x / biggest * 23.25 for x in diameter]
    scaledTo = "Scaled to 1 Euro"
elif materials[i] == "Copper":
    diameter = [x / biggest * 21.25 for x in diameter]
    scaledTo = "Scaled to 5 Cent"
else:
    scaledTo = "unable to scale.."

i = 0
total = 0
while i < len(diameter):
    d = diameter[i]
    m = materials[i]
    (x, y) = coordinates[i]
    t = "Unknown"
```

*14. ábra 180.-201. sor*

A következőkben a készítő próbál belőni egy hibahatárt, ami annyit jelent, hogy a rendelkezésre álló átmérő méreteket próbálja beszorítani különböző tartományokba melyek természetes az egyes érmék méretének a tartományát jelenti, és ha az adott átmérőn valamelyik ismert tartományba esik, akkor képes megmondani, hogy ez melyik címletet jelöli meg, különben ismeretlenként kezeli azt az átmérőt. Az erre szolgáló algoritmus:

```
if materials[i] == "Euro2":
    diameter = [x / biggest * 25.75 for x in diameter]
    scaledTo = "Scaled to 2 Euro"
elif materials[i] == "Brass":
    diameter = [x / biggest * 24.25 for x in diameter]
    scaledTo = "Scaled to 50 Cent"
elif materials[i] == "Euro1":
    diameter = [x / biggest * 23.25 for x in diameter]
    scaledTo = "Scaled to 1 Euro"
elif materials[i] == "Copper":
    diameter = [x / biggest * 21.25 for x in diameter]
    scaledTo = "Scaled to 5 Cent"
else:
    scaledTo = "unable to scale.."
```

*15. ábra hibahatár beállítása*

Majd a konkrét bekategorizálás, hogy tulajdonképpen melyik érme mennyit is ér:

```
if math.isclose(d, 25.75, abs_tol=1.25) and m == "Euro2":
    t = "2 Euro"
    total += 200
elif math.isclose(d, 23.25, abs_tol=2.5) and m == "Euro1":
    t = "1 Euro"
    total += 100
elif math.isclose(d, 19.75, abs_tol=1.25) and m == "Brass":
    t = "10 Cent"
    total += 10
elif math.isclose(d, 22.25, abs_tol=1.0) and m == "Brass":
    t = "20 Cent"
    total += 20
elif math.isclose(d, 24.25, abs_tol=2.5) and m == "Brass":
    t = "50 Cent"
    total += 50
elif math.isclose(d, 16.25, abs_tol=1.25) and m == "Copper":
    t = "1 Cent"
    total += 1
elif math.isclose(d, 18.75, abs_tol=1.25) and m == "Copper":
    t = "2 Cent"
    total += 2
elif math.isclose(d, 21.25, abs_tol=2.5) and m == "Copper":
    t = "5 Cent"
    total += 5
```

*16. ábra konkrét bekategorizálás*

Ezt követően alkotunk egy ideiglenes kimeneti képet, melyen még apróbb simításokat végez a készítő:

```
cv2.putText(output, t,
            (x - 40, y + 22), cv2.FONT_HERSHEY_PLAIN,
            1.5, (255, 255, 255), thickness=2, lineType=cv2.LINE_AA)
i += 1
```

*17. ábra ideiglenes kép megalkotása*

Ilyen, például amit a program legelején is véghez vittünk, azaz a kép méretezése természetesen jelen esetben is az arányok megtartása mellett.

```
d = 768 / output.shape[1]
dim = (768, int(output.shape[0] * d))
image = cv2.resize(image, dim, interpolation=cv2.INTER_AREA)
output = cv2.resize(output, dim, interpolation=cv2.INTER_AREA)
```

*18. ábra kép újabb méretezése, de már az output képé*

Végső lépésként a kimeneti képet úgy módosítja, hogy a dokumentáció elején látható érme felismerési egyedi stílusokat ráhelyezi a képre, mint az érmék

kiemelése körvonallal és a rajtuk lévő értékeknek a feltüntetése. Majd végül természetesen egy végső kimeneti képet is adunk, mely a szoftver lefutásának az eredménye lesz.

```
cv2.putText(output, scaledTo,
            (5, output.shape[0] - 40), cv2.FONT_HERSHEY_PLAIN,
            1.0, (0, 0, 255), lineType=cv2.LINE_AA)
cv2.putText(output, "Coins detected: {}, EUR {:2}".format(count, total / 100),
            (5, output.shape[0] - 24), cv2.FONT_HERSHEY_PLAIN,
            1.0, (0, 0, 255), lineType=cv2.LINE_AA)
cv2.putText(output, "Classifier mean accuracy: {}%".format(score),
            (5, output.shape[0] - 8), cv2.FONT_HERSHEY_PLAIN,
            1.0, (0, 0, 255), lineType=cv2.LINE_AA)

cv2.imshow("Output", np.hstack([image, output]))
cv2.waitKey(0)
```

*19. ábra végső kimeneti kép tökéletesítése*

***5. Hatékonyság:*** A program képes úgy is felismerni a pénzérméket, hogy nem a számozott, tehát amelyen a forgalom szerinti értéke van, feltüntetve felé néz, hanem ha az le van fordítva ez a lehetőség az átmérők felismerésének és kezelésnek az előnye, illetve azt is képes kezelni, ha két vagy több érme össze ér.

***6. Fejlesztési lehetőség:*** Véleményem szerint a szoftvert egy olyan fejlesztési lehetőség tudná előre vinni, hogy több ország érme címleteit legyen képes felismerni illetve azt, ha esetleg az érmék fedik egymást, illetve, hogy mobil vagy tablet eszközön is működő képes legyen akár úgy is, hogy Real Time azaz, kamerára tovább fejlesztve. Itt arra gondolok, hogy aplikáció szinten kezelné a kamerát, és ahogy azt valamilyen felületre kihelyezett érmék felé vinnék, jelezni azokat.

## *3. Felhasználó dokumentáció/használati útmutató:*

1. ***Szoftver csomag tartalma:***
   - *gepilatasdokumentacio.pdf dokumentáció*
   - *gepilatasbeadnado.py*
   - *A kód txt fájlként is: gepilatas.txt*
   - *bemenetni input kép image.jpeg*

2. ***Rövid útmutató:***
   - Ajánlott a dokumentáció részletes áttekintése a szoftver megértése és működésének átlátásához. A dokumentáció megnyitásához bármilyen pdf olvasó megfelelő.
   - A program futtatásához szükséges egy python IDE javasolt a Thonny amely innét ingyenesen letölthető: https://thonny.org
   - A Thonny IDE-ba vagy bemásoljuk a txt fájlból a kódot vagy importáljuk a gepilatasbeadando.py fájlt és futtatjuk.

- Fontos, hogy a feldolgozni kívánt kép és a forráskód egy mappában legyen.

3. ***Első lépések:*** A korábbiakban említetteken kívül a zavartalan futáshoz, szükséges néhány ingyenes python könyvtár letöltés melyet a program használ (az utasításokat kérem, kövesse pontosan):

   - ***Windows rendszer esetén:***
     - pip install numpy
     - pip install matplotlib
     - pip install glob
     - pip install opencv-python
     - pip install argparse
     - pip install -U scikit-learn

   - ***UNIX rendszer esetén:***
     - sudo apt-get update
     - sudo apt-get upgrade
     - pip install numpy sklearn scipy matplotlib
     - pip install numpy
     - pip install matplotlib
     - pip install glob
     - pip install opencv-python
     - pip install argparse
     - pip install -U scikit-learn

4. **Eredmény:** Célszerű a kimeneti képet is abba a mappába kimenteni ahol a szoftver többi fájlja is megtalálható, a könnyebb ellenőrzés céljából.

## *4. Források:*

https://github.com/j05t/coin_detector

https://docs.opencv.org/3.3.1/d3/db4/tutorial_py_watershed.html

https://hu.wikipedia.org/wiki/Gépi_látás

http://mialmanach.mit.bme.hu/fogalomtar/gepi_latas

http://progalap.elte.hu/downloads/seged/eTananyag/lecke28_lap1.html

https://raw.githubusercontent.com/j05t/coin_detector/master/screenshot.png

https://github.com/j05t/coin_detector/blob/master/detect_coins.py

## *5. Tartalomjegyzék:*

1. Koncepció/ elvárás

2. Fejlesztői dokumentáció
    1. Követelmény
    2. Futási környezet
    3. Használt program nyelv
    4. Használt algoritmusok és a program ismertetése
    5. Hatékonyság
    6. Fejlesztési lehetőség

3. Felhasználói dokumentáció
    1. Szoftver csomag tartalma
    2. Rövid útmutató
    3. Első lépések
    4. Eredmény

4. Források