Universitatea din Bucuresti
Facultatea de Matematica si Informatica

Documentatie Tema 2 CAVA

Capatina Razvan Nicolae, grupa 352

Alegerea Hiperparametrilor pentru Fereastra Glisanta

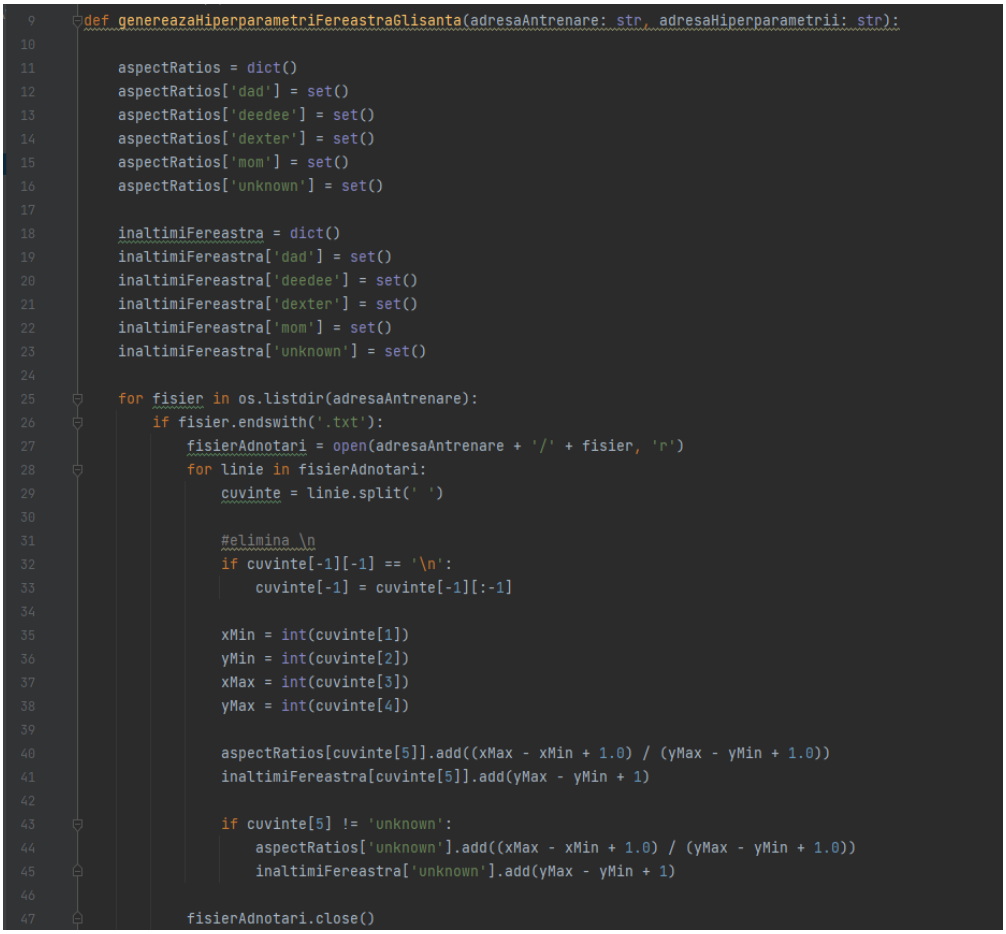
O prima problema ce trebuie rezolvata este gasirea unei metode de a alege ce dimensiuni si aspect ratio-uri vor avea ferestrele glisante folosite ulterior in detectarea faciala.

O fereastra glisanta poate fi descrisa doar printr-o singura dimensiune (latime sau inaltime) si aspect ratio-ul sau, cealalalta dimensiune fiind calculabila.

In implementarea proiectului am folosit urmatoarele idei:

- Pentru fiecare personaj in parte (Dexter, Dee Dee, Mom, Dad) trec prin toate imaginile care contin aparitii ale acestuia si salvez toate inaltimile si aspect ratio-urile distincte gasite.
- Decid cate inaltimi diferite si cate aspect ratio-uri diferite doresc sa am pentru acest personaj si aplic K Means Clustering pe valorile gasite, unde K este numarul ales.
- Obtin astfel hiperparametrii necesari ferestrei glisante.
- Metoda este aplicata pentru fiecare personaj in parte, obtinand astfel hiperparametrii personalizati pentru fiecare.
- In cazul detectiei de la Task-ul 1 toate valorile gasite pentru toate personajele in parte pot fi reunite inainte de aplicarea clusterizarii, astfel incat ferestrele glisante sa permita o gama mai variata de fete posibile.
- Pentru detectia pe o imagine ferestrele glisante vor fi de forma $(a, b) \in A \times B$, unde A este multimea inaltimilor clusterizate, iar B este multimea aspect ratio-urilor clusterizate.
- In implementare am folosit in timpul clusterizarii valoarea $K = 4$ in cazul aspect ratio-urilor si $K = 6$ in cazul inaltimilor, avand astfel in total 24 de ferestre glisante diferite.

Fragmente de cod:



```python
def genereazaHiperparametriFereastraGlisanta(adresaAntrenare: str, adresaHiperparametrii: str):

    aspectRatios = dict()
    aspectRatios['dad'] = set()
    aspectRatios['deedee'] = set()
    aspectRatios['dexter'] = set()
    aspectRatios['mom'] = set()
    aspectRatios['unknown'] = set()

    inaltimiFereastra = dict()
    inaltimiFereastra['dad'] = set()
    inaltimiFereastra['deedee'] = set()
    inaltimiFereastra['dexter'] = set()
    inaltimiFereastra['mom'] = set()
    inaltimiFereastra['unknown'] = set()

    for fisier in os.listdir(adresaAntrenare):
        if fisier.endswith('.txt'):
            fisierAdnotari = open(adresaAntrenare + '/' + fisier, 'r')
            for linie in fisierAdnotari:
                cuvinte = linie.split(' ')

                #elimina \n
                if cuvinte[-1][-1] == '\n':
                    cuvinte[-1] = cuvinte[-1][:-1]

                xMin = int(cuvinte[1])
                yMin = int(cuvinte[2])
                xMax = int(cuvinte[3])
                yMax = int(cuvinte[4])

                aspectRatios[cuvinte[5]].add((xMax - xMin + 1.0) / (yMax - yMin + 1.0))
                inaltimiFereastra[cuvinte[5]].add(yMax - yMin + 1)

                if cuvinte[5] != 'unknown':
                    aspectRatios['unknown'].add((xMax - xMin + 1.0) / (yMax - yMin + 1.0))
                    inaltimiFereastra['unknown'].add(yMax - yMin + 1)

            fisierAdnotari.close()
```

Figure 1: Cod Incarcare Hiperparametrii

```
os.makedirs(adresaHiperparametrii, exist_ok=True)


NUM_CLUSTER_ASPECT_RATIO = 4
NUM_CLUSTER_ASPECT_RATIO_UNKNOWN = 1 * NUM_CLUSTER_ASPECT_RATIO
for numePersonaj in aspectRatios:
    fisier = open(adresaHiperparametrii + '/' + numePersonaj + '_aspectRatios.txt', 'w')
    for aspectRatio in aspectRatios[numePersonaj]:
        fisier.write(str(aspectRatio) + '\n')
    fisier.close()

    fisier = open(adresaHiperparametrii + '/' + numePersonaj + '_aspectRatiosClustered.txt', 'w')
    numClusterAspectRatio = NUM_CLUSTER_ASPECT_RATIO
    if numePersonaj == 'unknown':
        numClusterAspectRatio = NUM_CLUSTER_ASPECT_RATIO_UNKNOWN
    aspectRatiosClustered = cluster.KMeans(n_clusters=numClusterAspectRatio, random_state=7)
    aspectRatiosClustered.fit(np.array(list(aspectRatios[numePersonaj])).reshape(-1, 1))
    for clusterCenter in aspectRatiosClustered.cluster_centers_.flatten():
        fisier.write(str(clusterCenter) + '\n')
    fisier.close()


NUM_CLUSTER_INALTIME_FEREASTRA = 6
NUM_CLUSTER_INALTIME_FEREASTRA_UNKNOWN = 1 * NUM_CLUSTER_INALTIME_FEREASTRA
for numePersonaj in inaltimiFereastra:
    fisier = open(adresaHiperparametrii + '/' + numePersonaj + '_inaltimiFereastra.txt', 'w')
    for inaltimeFereastra in inaltimiFereastra[numePersonaj]:
        fisier.write(str(inaltimeFereastra) + '\n')
    fisier.close()

    fisier = open(adresaHiperparametrii + '/' + numePersonaj + '_inaltimiFereastraClustered.txt', 'w')
    numClusterInaltimeFereastra = NUM_CLUSTER_INALTIME_FEREASTRA
    if numePersonaj == 'unknown':
        numClusterInaltimeFereastra = NUM_CLUSTER_INALTIME_FEREASTRA_UNKNOWN
    inaltimiFereastraClustered = cluster.KMeans(n_clusters=numClusterInaltimeFereastra, random_state=7)
    inaltimiFereastraClustered.fit(np.array(list(inaltimiFereastra[numePersonaj])).reshape(-1, 1))
    for clusterCenter in inaltimiFereastraClustered.cluster_centers_.flatten():
        fisier.write(str(clusterCenter) + '\n')
    fisier.close()
```

Figure 2: Cod Clusterizare Hiperparametrii

Generarea Exemplelor Negative

- Exemplele negative au fost generate decupand zone din imaginile de antrenare primite.
- Trebuie evitate zonele ce se suprapun cu bounding box-urile detectiilor faciale.
- In aceasta situatie poate fi folosit Intersection over Union pentru a controla suprapunerea dintre decupajele extrase si bounding box-urile fetelor.
  - Pentru valori mici ale IoU avem garantia ca decuparile nu se suprapun cu etichetele pozitive, dar scad sansele sa obtinem exemple puternic negative, ceea ce ar fi utile in antrenarea modelelor.
  - Pentru valori mari ale IoU putem incurca procesul de antrenare, daca decuparile aleatoare ajung sa incadreze perfect fete.
  - Limita superioara de 0.3 (aceeasi cu cea de la suprimarea non-maximelor) pentru generarea de exemple negative a dat rezultate bune.
- Pasii realizati:
  - Se alege in mod aleatoriu o imagine din cele de antrenare.
  - Se alege aleatoriu un aspect ratio si o inaltime din multimea valorilor clusterizate de la detectia hiperparametrilor.
  - Se alege in mod aleatoriu pozitia de start a decuparii (coltul stanga-sus).
  - Exista situatii in care dupa alegerea coltului nu putem face o decupare valida (IoU intre decupare si detectiile faciale este mai mare decat pragul ales). In acest caz incercam din nou, realegand alt aspect ratio, alta inaltime si alt colt stanga-sus.

  - Daca nu reusim dupa un numar stabilit de incercari sa obtinem o decupare inseamna ca ori imaginea este incarcata cu detectii faciale, ori am ales un aspect ratio / o inaltime / un punct de start ce ne ingreuneaza gasirea unei decupari valide. In acest caz revenim la pasul unde alegem aleator o imagine si repetam tot procedeul. Avem astfel o siguranta mai mare ca algoritmul nu se va bloca.

- Un numar de exemple negative de 20.000, respectiv 30.000 au dat rezultate bune in practica.

Fragmente de cod:

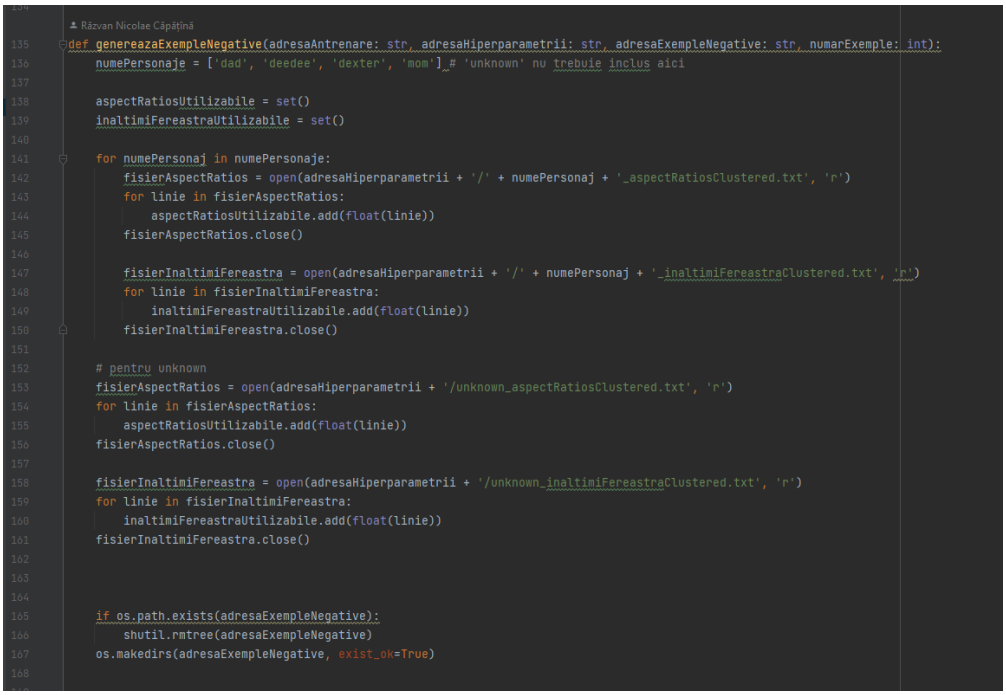
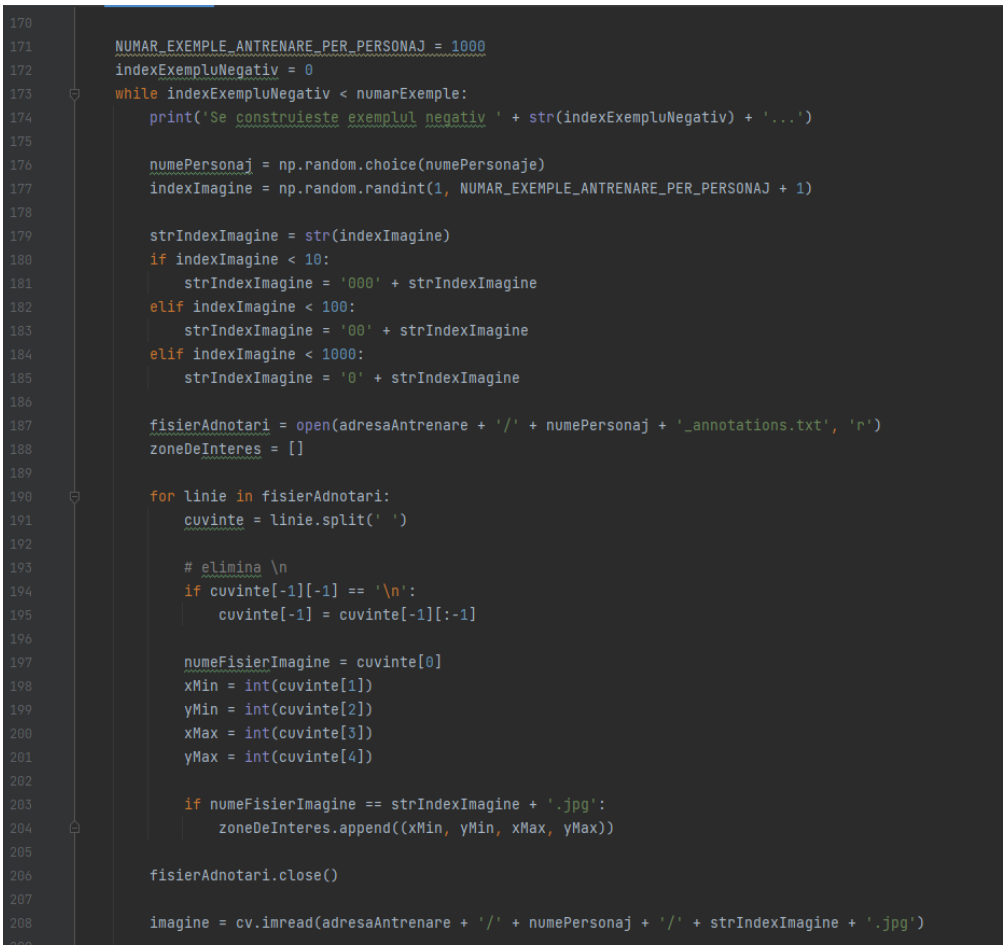
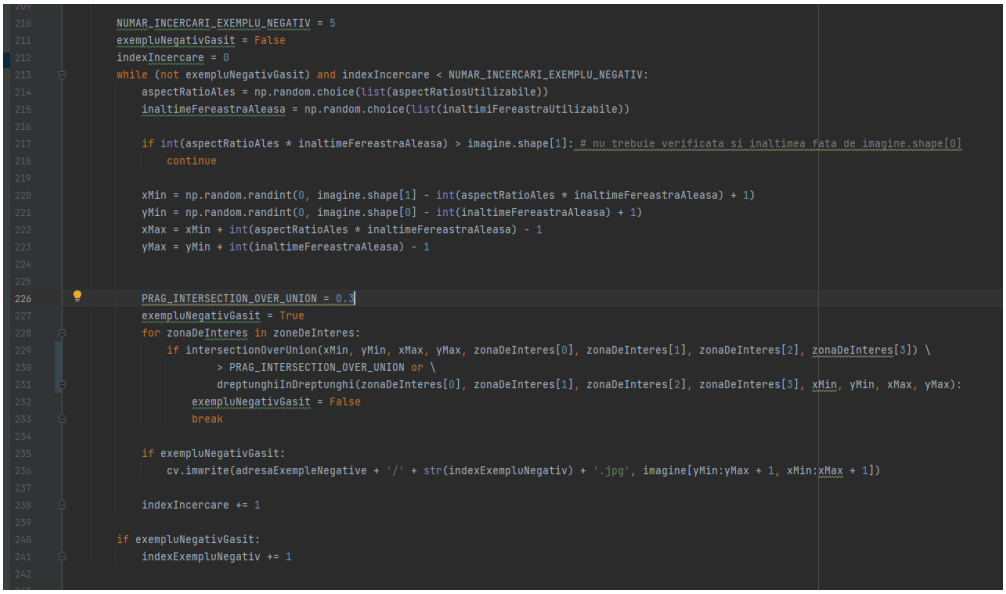


Figure 3: Cod Incarcare Hiperparametrii Exemple Negative



```
NUMAR_EXEMPLE_ANTRENARE_PER_PERSONAJ = 1000
indexExempluNegativ = 0
while indexExempluNegativ < numarExemple:
    print('Se construieste exemplul negativ ' + str(indexExempluNegativ) + '...')

    numePersonaj = np.random.choice(numePersonaje)
    indexImagine = np.random.randint(1, NUMAR_EXEMPLE_ANTRENARE_PER_PERSONAJ + 1)

    strIndexImagine = str(indexImagine)
    if indexImagine < 10:
        strIndexImagine = '000' + strIndexImagine
    elif indexImagine < 100:
        strIndexImagine = '00' + strIndexImagine
    elif indexImagine < 1000:
        strIndexImagine = '0' + strIndexImagine

    fisierAdnotari = open(adresaAntrenare + '/' + numePersonaj + '_annotations.txt', 'r')
    zoneDeInteres = []

    for linie in fisierAdnotari:
        cuvinte = linie.split(' ')

        # elimina \n
        if cuvinte[-1][-1] == '\n':
            cuvinte[-1] = cuvinte[-1][:-1]

        numeFisierImagine = cuvinte[0]
        xMin = int(cuvinte[1])
        yMin = int(cuvinte[2])
        xMax = int(cuvinte[3])
        yMax = int(cuvinte[4])

        if numeFisierImagine == strIndexImagine + '.jpg':
            zoneDeInteres.append((xMin, yMin, xMax, yMax))

    fisierAdnotari.close()

    imagine = cv.imread(adresaAntrenare + '/' + numePersonaj + '/' + strIndexImagine + '.jpg')
```

Figure 4: Cod Incarcare a Bounding Box-urilor Faciale din Imaginea Aleasa



```
NUMAR_INCERCARI_EXEMPLU_NEGATIV = 5
exempluNegativGasit = False
indexIncercare = 0
while (not exempluNegativGasit) and indexIncercare < NUMAR_INCERCARI_EXEMPLU_NEGATIV:
    aspectRatioAles = np.random.choice(list(aspectRatiosUtilizabile))
    inaltimeFereastraAleasa = np.random.choice(list(inaltimiFereastraUtilizabile))

    if int(aspectRatioAles * inaltimeFereastraAleasa) > imagine.shape[1]: # nu trebuie verificata si inaltimea fata de imagine.shape[0]
        continue

    xMin = np.random.randint(0, imagine.shape[1] - int(aspectRatioAles * inaltimeFereastraAleasa) + 1)
    yMin = np.random.randint(0, imagine.shape[0] - int(inaltimeFereastraAleasa) + 1)
    xMax = xMin + int(aspectRatioAles * inaltimeFereastraAleasa) - 1
    yMax = yMin + int(inaltimeFereastraAleasa) - 1


    PRAG_INTERSECTION_OVER_UNION = 0.3
    exempluNegativGasit = True
    for zonaDeInteres in zoneDeInteres:
        if intersectionOverUnion(xMin, yMin, xMax, yMax, zonaDeInteres[0], zonaDeInteres[1], zonaDeInteres[2], zonaDeInteres[3]) \
                > PRAG_INTERSECTION_OVER_UNION or \
                dreptunghiInDreptunghi(zonaDeInteres[0], zonaDeInteres[1], zonaDeInteres[2], zonaDeInteres[3], xMin, yMin, xMax, yMax):
            exempluNegativGasit = False
            break

    if exempluNegativGasit:
        cv.imwrite(adresaExempleNegative + '/' + str(indexExempluNegativ) + '.jpg', imagine[yMin:yMax + 1, xMin:xMax + 1])

    indexIncercare += 1

if exempluNegativGasit:
    indexExempluNegativ += 1
```

Figure 5: Cod Verificare IoU intre Exemplul Negativ si Bounding Box-uri

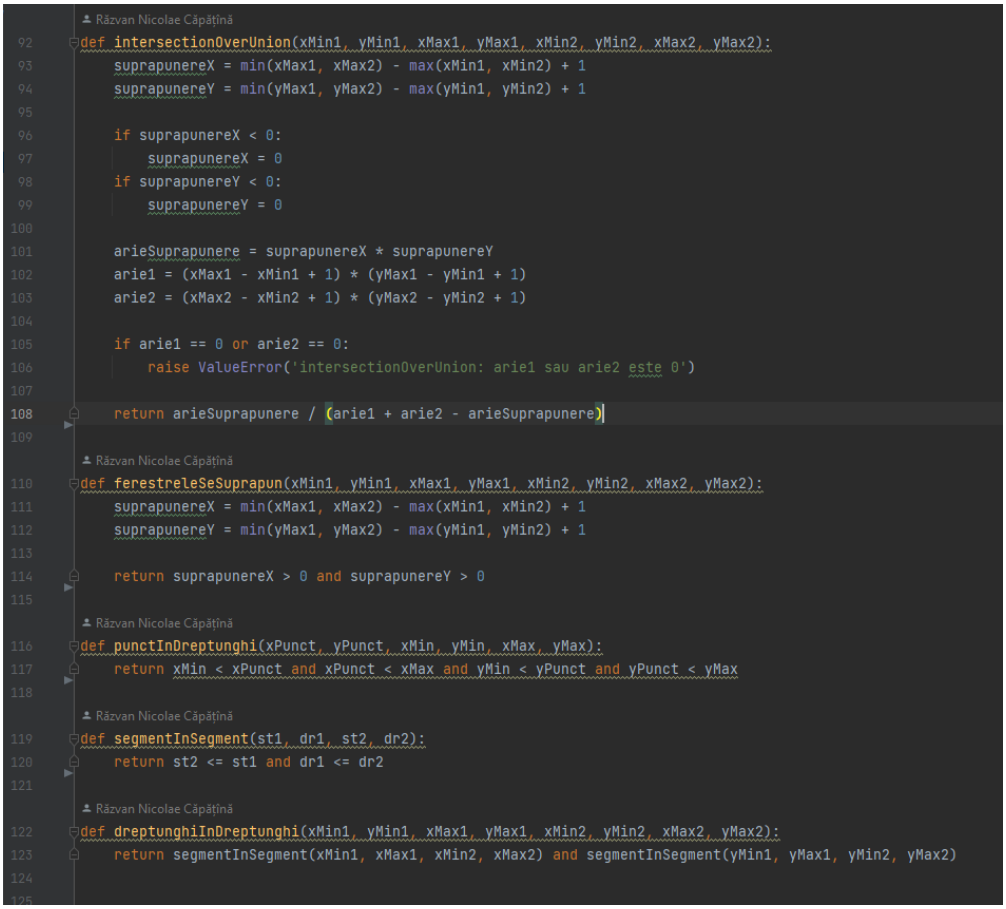


Figure 6: Cod IoU si alte Metode

Modele de Antrenare

Retea Neuronala Convolutionala

Implementarea finala foloseste o retea neuronala convolutionala, avand urmatoarea structura:

Figure 7: Structura Retea Neuronala Convolutionala

Aceasta structura poate categoriza binar, deoarece ultimul strat contine doar un nod si foloseste functia de activare sigmoida. Aceeasi retea a fost folosita si pentru Task-ul 1, unde face diferenta intre fata si non-fata, dar si pentru Task-ul 2, unde face diferenta intre un personaj anume si orice altceva.

Diferenta intre task-uri a fost doar intre ce etichetari au avut datele primite la etapa de antrenare. Pentru Task-ul 1, exemplele pozitive erau toate din fisierul de antrenare (incluzand si flip-uri pe orizontala), iar exemplele negative erau cele generate automat (incluzand si flip-uri pe orizontala/verticala/orizontala si verticala a acestora). Pentru Task-ul 2, exemplele pozitive erau cele din fisierul de antrenare unde eticheta era cea a personajului respectiv (imagine originala si flip pe orizontala). Exemplele negative erau reprezentate de restul imaginilor din fisierul de antrenare (incluzand si flip-uri pe orizontala), alaturi de toate imaginile generate automat (incluzand si flip-uri orizontala/verticala/orizontala si verticala a acestora).

Dimensiunea datelor de intrare este (dimensiuneImagine[0], dimensiuneImagine[1], 3), luand in calcul si culorile existente in imagine. In implementare, dimensiuneImagine este un tuplu cu valorile (64, 64). Astfel, dupa ce fereastra glisanta s-a pozitionat pe o anumita zona din imagine aceasta este decupata si redimensionata la (64, 64, 3), urmand sa fie data retelei convolutionale.

Pentru etapa de antrenare s-a folosit urmatoarea implementare:

```
302
303         # Antrenare Model
304
305         toateImaginile = np.concatenate((self.imaginiPozitive, self.imaginiNegative), axis=0)
306         toateEtichetele = np.concatenate((np.ones(self.imaginiPozitive.shape[0]), np.zeros(self.imaginiNegative.shape[0])))
307         if self.numePersonaj == 'deedee':
308             self.modelInvatare.fit(toateImaginile, toateEtichetele, epochs=3, batch_size=64)
309         else:
310             self.modelInvatare.fit(toateImaginile, toateEtichetele, epochs=2, batch_size=64)
311
312         print('Acuratete Model: ', self.modelInvatare.evaluate(toateImaginile, toateEtichetele)[1]) # 0 = loss, 1 = accuracy
313
```

Figure 8: Antrenare Retea Neuronala Convolutionala

In general, s-au folosit 2 epoci de antrenare si un batch size de 64. Doar in cazul personajului DeeDee s-a observat ca ar da rezultate mai bune daca au loc 3 epoci de antrenare in loc de 2.

Legat de numarul de exemple negative date retelei, in cazul Task-ului 1 s-au folosit 30.000 de exemple negative generate folosind modalitatea descrisa mai sus, iar in cazul Task-ului 2 numarul de exemple negative per personaj a variat. Avem astfel cate 20.000 pentru Dad, DeeDee, respectiv Dexter, in timp ce pentru Mom s-au folosit 30.000 de exemple negative.

Functia sigmoid ia valori reale in intervalul [0, 1], trebuie astfel setat un prag care separa afirmativ de negativ in clasificarea binara a modelului.

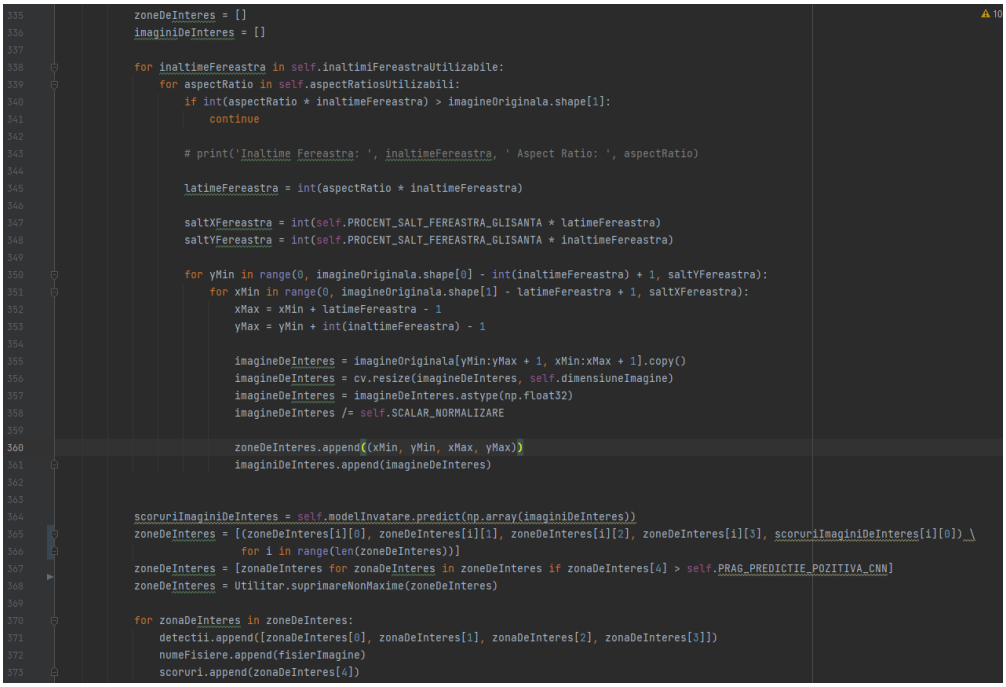
Pentru fiecare personaj in parte s-au folosit urmatoarele praguri diferite:

```
30          if self.numePersonaj == 'unknown':
31              self.PRAG_PREDICTIE_POZITIVA_CNN = 0.94
32          elif self.numePersonaj == 'dad':
33              self.PRAG_PREDICTIE_POZITIVA_CNN = 0.9561
34          elif self.numePersonaj == 'deedee':
35              self.PRAG_PREDICTIE_POZITIVA_CNN = 0.986
36          elif self.numePersonaj == 'dexter':
37              self.PRAG_PREDICTIE_POZITIVA_CNN = 0.952
38          else:# mom
39              self.PRAG_PREDICTIE_POZITIVA_CNN = 0.87
40
```

Figure 9: Praguri Sigmoida Retea Neuronala Convolutionala

Pragul 0.94 corespondent etichetei "unknown" este pentru Task-ul 1.

Urmatoarea implementare a fost folosita pentru a identifica bounding box-urile fetelor intr-o imagine:



```
        zoneDeInteres = []
        imaginiDeInteres = []

        for inaltimeFereastra in self.inaltimiFereastraUtilizabile:
            for aspectRatio in self.aspectRatiosUtilizabili:
                if int(aspectRatio * inaltimeFereastra) > imagineOriginala.shape[1]:
                    continue

                # print('Inaltime Fereastra: ', inaltimeFereastra, ' Aspect Ratio: ', aspectRatio)

                latimeFereastra = int(aspectRatio * inaltimeFereastra)

                saltXFereastra = int(self.PROCENT_SALT_FEREASTRA_GLISANTA * latimeFereastra)
                saltYFereastra = int(self.PROCENT_SALT_FEREASTRA_GLISANTA * inaltimeFereastra)

                for yMin in range(0, imagineOriginala.shape[0] - int(inaltimeFereastra) + 1, saltYFereastra):
                    for xMin in range(0, imagineOriginala.shape[1] - latimeFereastra + 1, saltXFereastra):
                        xMax = xMin + latimeFereastra - 1
                        yMax = yMin + int(inaltimeFereastra) - 1

                        imagineDeInteres = imagineOriginala[yMin:yMax + 1, xMin:xMax + 1].copy()
                        imagineDeInteres = cv.resize(imagineDeInteres, self.dimensiuneImagine)
                        imagineDeInteres = imagineDeInteres.astype(np.float32)
                        imagineDeInteres /= self.SCALAR_NORMALIZARE

                        zoneDeInteres.append((xMin, yMin, xMax, yMax))
                        imaginiDeInteres.append(imagineDeInteres)


        scoruriImaginiDeInteres = self.modelInvatare.predict(np.array(imaginiDeInteres))
        zoneDeInteres = [(zoneDeInteres[i][0], zoneDeInteres[i][1], zoneDeInteres[i][2], zoneDeInteres[i][3], scoruriImaginiDeInteres[i][0]) \
                         for i in range(len(zoneDeInteres))]
        zoneDeInteres = [zonaDeInteres for zonaDeInteres in zoneDeInteres if zonaDeInteres[4] > self.PRAG_PREDICTIE_POZITIVA_CNN]
        zoneDeInteres = Utilitar.suprimareNonMaxime(zoneDeInteres)

        for zonaDeInteres in zoneDeInteres:
            detectii.append([zonaDeInteres[0], zonaDeInteres[1], zonaDeInteres[2], zonaDeInteres[3]])
            numeFisiere.append(fisierImagine)
            scoruri.append(zonaDeInteres[4])
```

Figure 10: Detectare Bounding Box Retea Neuronala Convolutionala

Se observa faptul ca are loc un produs cartezian intre inaltimile valabile si aspect ratio-urile valabile pentru a determina o ferestra glisanta, care apoi este folosita pentru a gasi fetele personajelor.

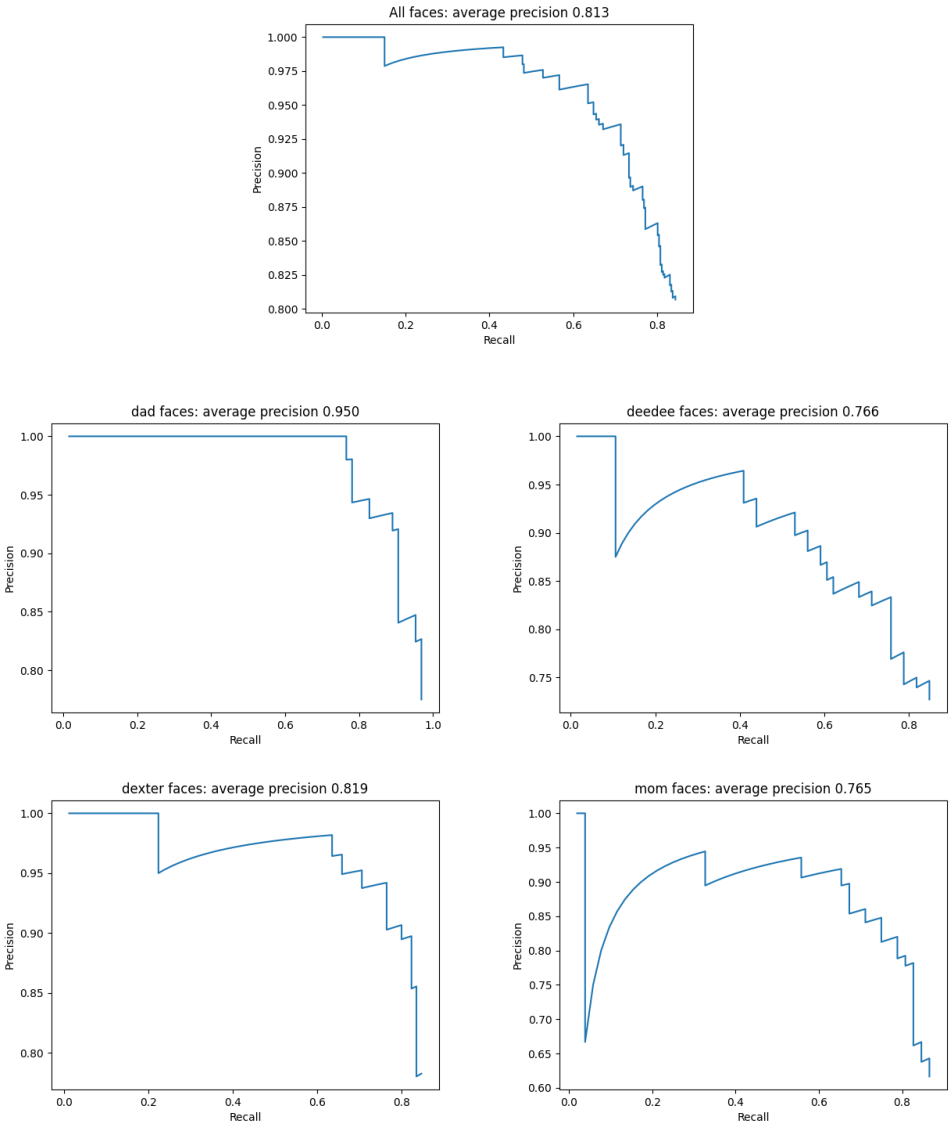
Atunci cand ferestra glisanta traverseaza imaginea se observa utilizarea unui step pe axa OX si pe axa OY. Acest step are formula: dimensiunea ferestrei pe acea axa inmultita cu un procent de salt (in implementare este 0.2). Acest lucru aduce o optimizare ferestrei glisante, deoarece nu va mai traversa pixel cu pixel imaginea, ci mai multi pixeli de o data. De asemenea, acest salt este direct proportional cu fereastra, avand loc deplasari mai mari in cazul ferestrelor mai mari.

Toate decuparile sunt redimensionate la dimensiunea de input a retelei convolutionale si in final sunt pastrate doar cele care depasesc pragul setat pentru functia sigmoida.

Are loc si o suprimare de non-maxime, pragul de IoU folosit fiind 0.3.

Performanta

Pe cele 200 de imagini din datele de validare s-au obtinut urmatoarele grafice Precision-Recall:



Support Vector Classifier si Histograme de Gradienti Orientati

Alt model incercat, dar care nu a oferit rezultate la fel de bune, a fost cel al clasificatorului vector suport liniar, impreuna cu histogramele de gradienti orientati.

Pentru histogramele de gradienti orientati s-au folosit urmatoarele valori:

```
self.pixeliPerCelula = (12, 12)
self.celulePerBloc = (2, 2)
self.celulePerImagine = (8, 8)
self.numOrientari = 9

self.dimensiuneImagine = (self.pixeliPerCelula[0] * self.celulePerImagine[0], self.pixeliPerCelula[1] * self.celulePerImagine[1])

self.parametriiRegularizare = [(10 ** x) for x in range(-6, 6)]
self.NUMAR_ITERATII_ANTRENARE = 100
```

De asemenea, din imagine se observa ca se realizeaza maxim 100 de iteratii in timpul etapei de antrenare a clasificatorului liniar, iar in privinta factorului de regularizare s-au incercat mai multe valori, pornind de la 10^{-6}, 10^{-5}, ..., pana la 10^{5}, urmand sa se aleaga modelul cu cea mai buna acuratete, implementarea acestei etape fiind urmatoarea:

```
159
160         acurateteMaxima = -1.0
161
162         for parametruRegularizare in self.parametriiRegularizare:
163
164             modelInvatare = svm.LinearSVC(C=parametruRegularizare, max_iter=self.NUMAR_ITERATII_ANTRENARE)
165             totiDescriptorii = np.concatenate((self.descriptoriPozitivi, self.descriptoriNegativi), axis=0)
166             toateEtichetele = np.concatenate((np.ones(self.descriptoriPozitivi.shape[0]), np.zeros(self.descriptoriNegativi.shape[0])))
167             modelInvatare.fit(totiDescriptorii, toateEtichetele)
168
169             acurateteCurenta = modelInvatare.score(totiDescriptorii, toateEtichetele)
170             print('Acuratete Model: ', acurateteCurenta)
171             if acurateteCurenta > acurateteMaxima:
172                 acurateteMaxima = acurateteCurenta
173                 self.modelInvatare = copy.deepcopy(modelInvatare)
174
175         print('Acuratete Maxima Model: ', acurateteMaxima)
176
```