Universitatea din București
Facultatea de Matematică și Informatică
Informatică

Bucă Mihnea-Vicențiu
Anul 3
Grupa 352

# Concepte și Aplicații în Vederea Artificială
# Tema 2

**Obiectiv**
Scopul acestei teme este implementarea unui sistem automat de detectare și recunoaștere facială a personajelor din serialul de desene animate *Laboratorul lui Dexter* folosind algoritmi de Vedere Artificială discutați la curs și implementați parțial la laborator.

Pentru realizarea proiectului, am folosit:

- *Python* 3.13.1
- *PyTorch* 2.7.0.dev20250118+cpu
- *OpenCV* 4.10.0
- *Sklearn* 1.5.2
- *Pickle* 4.0
- *NumPy* 2.1.2
- *Matplotlib* 3.9.2

Am folosit Jupiter Notebook pentru a scrie codul și pentru a testa funcționalitatea acestuia. Editorul de text folosit a fost Visual Studio Code.

**Mențiuni**
Soluția lucrează cu alte fișiere prin path-uri relative la folderul în care se află (Folder-ul **/cod**).
Soluția cuprinde o secțiune numită Paths, unde se introduce path-ul către setul de testare, denumit *test_data*, ce cuprinde imaginiile care vor fi supuse evaluarii.

```
save_train_data = 'crops/'
cluster_train_data = 'clusters/'
path_train_data = '../testare/'
test_data = '../evaluare/fake_test/' # path to test data
npy_response = '352_Mihnea-Vicentiu_Buca/'
npy_load_data = 'curr_data/'
```

Parametrii se pot modifica ca atare pentru a funcționa evaluarea pe setul de testare dorit.
Pentru a rula soluția este necesar să se execute toate celulele din Jupiter Notebook.
Soluția nu va mai creea alte modele de antrenare sau de clustering, ci va folosi cele deja existente.
Soluția produce un folder local cu numele **352_Mihnea-Vicentiu_Buca** ce conține fișierele de output necesare evaluarii.

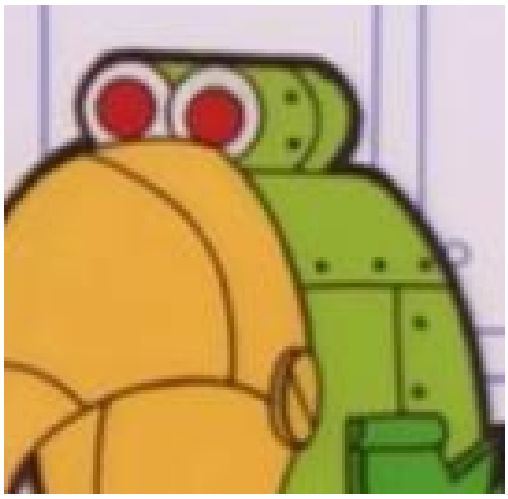
**Structura documentației**
Documentația se împarte în 3 secțiuni:

- **Creearea setului de antrenare** – unde se realizează croparea zonelor de interes din imaginile de antrenare, labeling-ul acestora și clusterizarea lor
- **Antrenarea modelului de evaluare** – unde se realizează antrenarea unui model **CNN** de evaluare folosind setul de date creat anterior pentru a recunoaște personajele din serialul de desene animate
- **Detectarea fețelor** – unde se realizează calculul scorului pentru fiecare imagine și se selectează cele mai potrivite bounding box-uri

## Creearea setului de antrenare

Pentru a crea setul de antrenare vom folosi setul de imagini din folder-ul **antrenare** împreună cu adnotările respective. Astfel vom putea crea positive descriptors pentru modelul de antrenare, dar întâmpinăm o prima problemă anume nu avem negative descriptors. Pentru a rezolva această problemă, vom cropa zone random din imagini, vom vedea dacă acestea se intersectează cu/au o suprapunere cu zonele de interes și vom folosi acestea ca negative descriptors.

(a) Exemplu de positive descriptor



(b) Exemplu de negative descriptor

Descrierea figurilor

Din păcate, această metodă se dovedește a performa slab, deoarece nu avem o diversitate mare de negative descriptors, am vrea niște zone care să se poată intersecta puțin cu zonele de interes, sau în termeni mai precisi, strong negative descriptors.

Pentru a putea obține aceste zone, mai întâi vom clasifica diferitele zone de interese (bounding boxurile) în **(aspect_ratio, height)** pentru fiecare dintre cele 5 categorii bazate pe personaljele din serial și personajele din fundal, apoi vom aplica algoritmul de clustering **KMeans** pentru a obține niște **(apsect_ratio, height)** (în cazul meu 5 pentru fiecare categorie) bazate pe bounding boxul fetelor diferitlor personaje din desen.

Acest lucru ne ajuta mai mult decât o simplă cropare randomizată, rămâne să verificăm dacă aceste zone sunt suficient de bune pentru a fi folosite ca negative descriptors, vom folsi o metrică de **IoU** (Intersection over Union) pentru a vedea cât de bine se potrivesc aceste zone cu zonele de interes. În urma rulării algoritmului obține în jur de 36.000 de exemple negative iar pentru fiecare personaj în jur de 1.000 de exemplare. Constanta aleasă a fost de 0.3 pentru a putea obține un echilibru între zonele care se potrivesc și zonele care

Exemplu de strong negative descriptor

nu se potrivesc. Vom salva descriptorii fiecare intr-un **numpy** array, și vom merge spre umrătorul pas.

```
for (aspect_ratio, height) in random_clusters:
height = int(height)
width = int(aspect_ratio * height)

if width > image.shape[1] or height > image.shape[0]:
    continue

x1 = np.random.randint(0, image.shape[1] - width)
y1 = np.random.randint(0, image.shape[0] - height)
x2 = x1 + width
y2 = y1 + height

if x2 > image.shape[1] or y2 > image.shape[0]:
    continue

rect_are_good = True
for (x1_, y1_, x2_, y2_) in rects:
    if not intersection_over_union_scaled((x1, y1, x2, y2), (x1_, y1_, x2_, y2_), 0.1):
        rect_are_good = False
        break

if rect_are_good:
    crop = cv2.resize(image[y1:y2, x1:x2], (64, 64))
    cv2.imwrite(get_path('negative', curr_run['negative']), crop)
    curr_run['negative'] += 1
```

# Antrenarea modelului de evaluare

Aceasta arhitectură de CNN este un model compact și eficient conceput pentru clasificarea imaginilor binare (pozitive și negative), utilizând **convoluții separabile pe adâncime** pentru a reduce complexitatea modelului fără a sacrifica prea mult performanța, implementarea a fost facută cu *PyTorch*.

# 1. Dataset

**Preprocesare date:**

- Imaginile sunt normalizate la intervalul $[0, 1]$ pentru a accelera convergența în timpul antrenării.
- Dimensiunile imaginilor sunt reordonate la formatul $(N, C, H, W)$, conform așteptărilor PyTorch.

**Labeluri:**

- Exemplele pozitive au eticheta 1, iar cele negative 0.

Această preprocesare pregătește datele pentru a fi compatibile cu convoluțiile bidimensionale și reduce riscul ca valorile ridicate ale pixelilor să ducă la instabilitate numerică.

# 2. Layer-ul de convoluție separabilă (DepthwiseSeparableConv)

**Motivație:** Convoluțiile separabile pe adâncime reduc numărul total de parametri și operațiuni:

- **Convoluție pe adâncime:** Fiecare canal este procesat individual.
- **Convoluție punctuală:** Combină informația între canale.

**Avantaje:**

- Scade consumul de memorie și timpul de calcul.

# 3. Structura CNN

## Blocuri convoluționale

Arhitectura are 3 blocuri convoluționale, fiecare compus din:

1. **Convoluție separabilă pe adâncime:** Crește expresivitatea rețelei, procesând fiecare canal individual și combinând rezultatele între canale.
2. **Funcția de activare ReLU:** Introduce nelinearitatea pentru a permite modelului să învețe relații complexe.
3. **Batch Normalization:** Normalizează valorile activărilor, accelerând convergența și stabilizând antrenarea.
4. **Pooling (MaxPool2d):** Reduce dimensiunea spațială a imaginilor, extrăgând caracteristici semnificative și reducând riscul de overfitting.

## Global Average Pooling (GAP)

**Rol:** Reduce dimensionalitatea spațială la o singură valoare per canal (128), păstrând caracteristicile globale.
**Motivație:** GAP este mai robust la supraînvățare și permite generalizare mai bună comparativ cu straturile complet conectate voluminoase.

### Clasificator Fully-Connected

1. **Linear(128 → 256):** Proiectează caracteristicile GAP într-un spațiu latent mai mare.
2. **ReLU + Dropout:** ReLU introduce nelinearitatea, iar Dropout reduce overfitting-ul.
3. **Linear(256 → 1):** Produce probabilitatea finală de clasificare folosind o funcție de activare Sigmoid.

## Justificarea designului arhitecturii

1. **Convoluții separabile pe adâncime:**
   - Alegerea acestora permite un compromis între eficiență computațională și acuratețe, fiind utilizate cu succes în rețele precum MobileNet.
   - Reduce drastic numărul de parametri comparativ cu convoluțiile standard.
2. **Pooling stratificat:**
   - Pooling-ul reduce dimensiunea intrărilor în pași succesivi, reducând dimensiunea calculului și rezumând caracteristicile esențiale.
3. **Global Average Pooling (GAP):**
   - Permite arhitecturii să fie mai robustă la pozițiile caracteristicilor în imagine.
   - Evită supraînvățarea, ceea ce este crucial pentru dataset-uri mici sau moderate.
4. **Dropout:**
   - Reduce probabilitatea de supraînvățare în straturile complet conectate.
5. **Sigmoid la ieșire:**
   - Adecvată pentru clasificarea binară, returnând o probabilitate între $[0, 1]$.

## Avantaje ale arhitecturii

1. **Eficiență:** Convoluțiile separabile pe adâncime reduc complexitatea calculului, permițând antrenarea rapidă chiar și pe hardware modest.
2. **Flexibilitate:** Arhitectura poate fi extinsă sau ajustată pentru clasificare multi-clasă cu modificări minore.
3. **Robustețe:** GAP și Dropout reduc riscul de overfitting, îmbunătățind generalizarea.

În cadrul acestui proiect, am folosit și antrenat cinci modele diferite pentru a evalua performanța în recunoașterea fețelor. Am ales să folosim două tipuri de funcții de pierdere: BSELoss și MSELoss, pentru a compara eficiența acestora în diferite scenarii.

**Modelel MSELoss:** Am antrenat toate modelepentru fiecare dintre personajele Dexter, DeeDee, Dad, Mom și toate fețele folosind funcția de pierdere MSELoss. Fiecare model a fost antrenat pe un set de date specific persoanei respective, pentru a îmbunătăți acuratețea recunoașterii feței acelei persoane.
Am folosit aceeași structură de rețea neuronală pentru toate modelele, pentru a menține consistența și a putea compara direct performanțele acestora. Alegerea funcțiilor de pierdere diferite a fost motivată de dorința de a explora cum afectează acestea procesul de antrenare și rezultatele finale.
Modelele au fost antrenate pe setul de date creeat la subpunctul anteroir cu datele specifice fiecruia anteroir cu detele selectate specific pentru fiecare personaj.

```python
class DepthwiseSeparableConv(nn.Module):
    def __init__(self, in_channels, out_channels, kernel_size=3, padding=1):
        super(DepthwiseSeparableConv, self).__init__()
        self.depthwise = nn.Conv2d(in_channels, in_channels, kernel_size, padding=padding, groups=in
        self.pointwise = nn.Conv2d(in_channels, out_channels, kernel_size=1)

    def forward(self, x):
        x = self.depthwise(x)
        x = self.pointwise(x)
        return x

class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.model = nn.Sequential(
            DepthwiseSeparableConv(3, 32),
            nn.ReLU(),
            nn.BatchNorm2d(32),
            nn.MaxPool2d(2),

            DepthwiseSeparableConv(32, 64),
            nn.ReLU(),
            nn.BatchNorm2d(64),
            nn.MaxPool2d(2),

            DepthwiseSeparableConv(64, 128),
            nn.ReLU(),
            nn.BatchNorm2d(128),
            nn.MaxPool2d(2),

            nn.AdaptiveAvgPool2d(1),  # GAP
            nn.Flatten(),
            nn.Linear(128, 256),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(256, 1),
            nn.Sigmoid()
        )

    def forward(self, x):
        x = self.model(x)
        return x
```

# Detectarea fețelor

Codul implementează un pipeline pentru detectarea fețelor pe imagini, utilizând *aspect ratios* și înălțimi generate de clustere predefinite. Modelul preia un set de date de testare, aplică detectarea fețelor folosind un model antrenat și filtrează rezultatele finale.

## 1. Funcții auxiliare

### a. `is_completely_overlapped`

- **Scop:** Verifică dacă un dreptunghi (bounding box) este complet suprapus peste altul.
- **Intrări:**
  - `box1`, `box2`: Coordonatele dreptunghiurilor sub formă de $(x_1, y_1, x_2, y_2)$.
- **Ieșire:** Returnează `True` dacă unul dintre dreptunghiuri este complet cuprins în celălalt.

### b. `sliding_window`

- **Scop:** Generează ferestre (*windows*) de dimensiuni fixe peste imagine.
- **Intrări:**
  - `image`: Imaginea pe care se aplică fereastra glisantă.
  - `step_size`: Dimensiunea pasului pentru deplasarea ferestrei.
  - `window_size`: Dimensiunile ferestrei (lățime și înălțime).
- **Ieșire:** `Yield` (generator) al ferestrelor și coordonatelor lor.

### c. `filter_boxes`

- **Scop:** Filtrează dreptunghiurile detectate utilizând un prag pentru IoU și verificarea suprapunerii complete.
- **Intrări:**
  - `best_results`: Lista dreptunghiurilor și scorurilor lor.
  - `iou_threshold`: Prag pentru Intersection over Union (default 0.25).
- **Ieșire:** Lista dreptunghiurilor filtrate și a scorurilor asociate.

## 2. Funcția principală: `detect_faces_best_aspect_ratios`

### Descriere generală

Aceasta funcție utilizează un model antrenat pentru a detecta fețe în imagini, folosind dimensiuni și proporții ale ferestrelor derivate din clustere. Rezultatele sunt filtrate și salvate.

### Etape detaliate

1. **Încarcă clusterele:**
   - Fiecare cluster (ex.: `dad`, `mom`) este încărcat din fișiere salvate (`kmeans.pkl`).
   - Se colectează toate centrele clusterelor pentru a genera combinații de *aspect ratios* și înălțimi.
2. **Selectare dimensiuni ferestre:**
   - Din centrele clusterelor, se selectează un număr limitat (20 sau mai puține) de *aspect ratios* și înălțimi ale ferestrelor.
3. **Fereastră glisantă:**
   - Pentru fiecare imagine și fiecare combinație de *aspect ratio* și înălțime:
     - Se generează ferestre glisante (*sliding window*) pe imagine.
     - Ferestrele sunt redimensionate la $64 \times 64$ pixeli pentru a fi compatibile cu modelul.
4. **Clasificare fețe:**
   - Ferestrele sunt procesate în *batch-uri* prin modelul pre-antrenat.
   - Se păstrează doar ferestrele cu probabilitate peste pragul stabilit (0.90).
5. **Filtrare și NMS:**
   - Se aplică **Non-Maximum Suppression (NMS)** pentru a elimina suprapunerile între ferestre.
   - Se aplică funcția `filter_boxes` pentru a elimina suprapunerile complete.
6. **Salvarea rezultatelor:**
   - Coordonatele ferestrelor detectate, numele fișierelor și scorurile sunt salvate în format `.npy`.

### Parametri

- `test_data`: Calea către imaginile de test.
- `model`: Modelul antrenat pentru clasificare.
- `model_name`: Numele modelului (pentru salvarea rezultatelor).
- `cluster_train_data`: Directorul unde sunt salvate clusterele.
- `step_size`: Dimensiunea pasului pentru *sliding window*.
- `threshold`: Pragul pentru detectarea unei fețe (0.90).

## 3. Componente principale ale pipeline-ului

- **Clusterizarea:** Dimensiunile ferestrelor sunt determinate din clustere (*aspect ratios* și înălțimi).
- **Fereastră glisantă:** Permite procesarea tuturor regiunilor posibile dintr-o imagine.
- **Modelul pre-antrenat:** Clasifică ferestrele ca fiind față sau non-față.
- **NMS și filtrare:** Optimizează rezultatele prin eliminarea redundantă a ferestrelor.

## 4. Avantaje

- **Flexibilitate:** Utilizează *aspect ratios* dinamice și dimensiuni generate din datele de antrenament.
- **Robustețe:** Aplicarea NMS și a filtrării suprapunerilor îmbunătățește calitatea detectărilor.
- **Modularitate:** Codul este structurat, facilitând înlocuirea modelului sau a clusterelor.

## 5. Limitări

- **Eficiență:** *Sliding window* poate fi lent pe imagini mari.
- **Dependință de clustere:** Calitatea detectării depinde de calitatea clusterelor predefinite.
- **Resurse:** Redimensionarea și clasificarea tuturor ferestrelor necesită memorie și timp.

```
def detect_faces_best_aspect_ratios(test_data, model, model_name, cluster_train_data, task="task1",
    face_recognition_results = []

    # Preload all clusters
    clusters = {name: pickle.load(open(cluster_train_data + name + '_kmeans.pkl', 'rb'))
                for name in ['dad', 'mom', 'dexter', 'deedee', 'unknown']}


    all_cluster_centers = []
    for name, kmeans in clusters.items():
        all_cluster_centers.extend(kmeans.cluster_centers_)

    all_cluster_centers = np.vstack(all_cluster_centers)

    min_len = min(20, all_cluster_centers.shape[0])

    detections_for_model = []
    file_names_for_model = []
    scores_for_model = []

    for image_path in os.listdir(test_data):
        image = cv2.imread(test_data + image_path)
        best_results = []

        random_indices = np.random.choice(all_cluster_centers.shape[0], min_len, replace=False)

        aspect_ratios = all_cluster_centers[random_indices, 0]
        heights = all_cluster_centers[random_indices, 1]

        for aspect_ratio, height in zip(aspect_ratios, heights):
            window_height = int(height)
            window_width = int(aspect_ratio * height)
            window_size = (window_width, window_height)

            batch_windows = []
            batch_coords = []

            for x, y, window in sliding_window(image, step_size, window_size):
                if x + window_width > image.shape[1] or y + window_height > image.shape[0]:
```

```
                    continue

                resized_window = cv2.resize(window, (64, 64))
                resized_window = torch.tensor(resized_window, dtype=torch.float32) / 255.0
                resized_window = resized_window.permute(2, 0, 1)
                batch_windows.append(resized_window)
                batch_coords.append((x, y, x + window_width, y + window_height))

            if batch_windows:
                batch_windows = torch.stack(batch_windows)
                with torch.no_grad():
                    probs = model(batch_windows).squeeze().tolist()

                best_results.extend([(batch_coords[i], probs[i]) for i in range(len(batch_windows))

        if len(best_results) > 0:
            # Apply NMS
            bboxes, scores = zip(*best_results)
            bboxes = torch.tensor(bboxes, dtype=torch.float32)
            scores = torch.tensor(scores, dtype=torch.float32)
            indices = nms(bboxes, scores, 0.25)
            best_results = [(bboxes[idx].int().tolist(), scores[idx].item()) for idx in indices]

            # Filter results
            best_results = filter_boxes(best_results, 0.25)


            # Draw results
            for bbox, _ in best_results:
                detections_for_model.append(bbox)
                file_names_for_model.append(image_path)
                scores_for_model.append(_)

                # x1, y1, x2, y2 = bbox
                # cv2.rectangle(image, (x1, y1), (x2, y2), (0, 0, 255), 2)

            # face_recognition_results.append(image)

        print("Processed", image_path)

    np.save(npy_response + f'{task}/detections_{model_name}.npy', detections_for_model)
    np.save(npy_response + f'{task}/file_names_{model_name}.npy', file_names_for_model)
    np.save(npy_response + f'{task}/scores_{model_name}.npy', scores_for_model)
    # return face_recognition_results
```



**Bucă Mihnea-Vicențiu**

**Anul 3 Informatică**

**Grupa 352**