Tema 2 - CAVA

Detectarea si recunoasterea faciala a personajelor din serialul de desene animate Familia Simpson

1. Introducere

Scopul acestei teme este implementarea unui sistem automat de detectare si recunoastere faciala a personajelor din serialul "The Simpsons" folosind algoritmi de Vedere Artificiala.

2. Task 1- Detectarea faciala

2.1. Generarea de exemple pozitive și negative

Pentru a putea antrena modelul SVC pe descriptorii de date, am pregătit datele pentru antrenare astfel: pentru datele pozitive, încărcate imaginile si salvez fetele; pentru a genera date negative, am generat patch-uri random pentru care m-am asigurat ca nu are un scor de intersection over union prea mare (am generat cate 2 patch-uri pe imagine cu iou mai mic decat 0.05, 0.1 si 0.3). De asemenea, toate fetele si patch-urile salvate au avut dimensiunea de (224, 224) cu 3 canale.

In fisierul run_task1.py voi genera exemplele pozitive si negative doar daca nu exista.

2.2. Calcularea descriptorilor pozitivi si negativi

In fisierul facialdetector.py, inspirata de laboratorul 11, am folosit functiile pentru calcularea descriptorilor pozitivi si negativi, pe care ii salveaza pentru a putea fi folositi pentru o rulare ulterioara a programului.

```
# ? RESNET POSITIVE DESCRIPTORS
def get_positive_descriptors_resnet(self):
  images_path = os.path.join(POSITIVE, '*.jpg')
  files = glob.glob(images_path)
  num_images = len(files)
  positive_descriptors = []
  model = models.resnet18(pretrained=True)
  layer = model._modules.get('avgpool')
  model.eval()
  scaler = transforms.Resize((IMG_SIZE, IMG_SIZE))
```

```
  normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
  std=[0.229, 0.224, 0.225])
  to_tensor = transforms.ToTensor()
  for i in range(num_images):
    img = cv.imread(files[i])
    img = Image.fromarray(img)
    t_img = Variable(normalize(to_tensor(scaler(img))).unsqueeze(0))
    my_embedding = torch.zeros(512)
    def copy_data(m, i, o):
      my_embedding.copy_(o.data.reshape(o.data.size(1)))
    h = layer.register_forward_hook(copy_data)
    model(t_img)
    h.remove()
    my_embedding = my_embedding.numpy()
    positive_descriptors.append(my_embedding)
  positive_descriptors = np.array(positive_descriptors)
  return positive_descriptors
```

Codul este similar pentru descriptorii negativi.

2.3. Clasificator Liniar pentru calcularea descriptorilor

Tot inspirat din laborator, am folosit functia care antreneaza un model liniar SVC pe descriptorii pozitivi si negativi.

```
def train_classifier(self, training_examples, train_labels,
ignore_restore=True):
  svm_file_name = os.path.join(SAVED, 'best_model')
  if os.path.exists(svm_file_name) and ignore_restore:
    self.best_model = pickle.load(open(svm_file_name, 'rb'))
    return

  best_accuracy , best_c, best_model  = 0, 0, None
  Cs = [10 ** -5, 10 ** -4,  10 ** -3,  10 ** -2]
  for c in Cs:
    model = LinearSVC(C=c)
    model.fit(training_examples, train_labels)
    acc = model.score(training_examples, train_labels)
    if acc > best_accuracy:
      best_accuracy = acc
      best_c = c
      best_model = deepcopy(model)

  scores = best_model.decision_function(training_examples)
```

```
    self.best_model = best_model
    positive_scores = scores[train_labels > 0]
    negative_scores = scores[train_labels <= 0]
```

2.4. Sliding window pentru detectarea faciala

Functia new_run() citeste cate o imagine pe rand, dupa care face detectare faciala pe patch-uri din imagine. Am ales ca dimensiunile patch-urilor sa fie procente din latimea imaginii, (10%, 20%, 30% etc) urmand sa fie redimensionate la (224, 224), pentru care se calculează descriptorii si clasificatorul. De asemenea, pentru optimizare din punct de vedere al timpului, "trimis" la evaluare doar patch-urile care au un procent de cel putin 50% de galben (deoarece toate fetele personajelor au o nuanta de galben).

De asemenea, am aplicat functia de 'non maximal suppresion' care trimite ca rezultat final versiunea cea mai buna a aceleiasi fete gasite.

```
for patch_size in patch_sizes:
  for y in range(0, num_rows - patch_size, 10):
    for x in range(0, num_cols - patch_size, 10):
      mask_patch = mask[y : y + patch_size, x : x + patch_size]
      no_zero = cv.countNonZero(mask_patch)

      if no_zero > (patch_size ** 2) / 2:
        bbox_curent = [x, y, x + patch_size, y + patch_size]
        xmin, ymin, xmax, ymax = bbox_curent[0], bbox_curent[1],
bbox_curent[2], bbox_curent[3]
        img_patch = img[ymin:ymax, xmin:xmax]

        img_patch = Image.fromarray(img_patch)
        t_img =
Variable(normalize(to_tensor(scaler(img_patch))).unsqueeze(0))
        my_embedding = torch.zeros(512)

        h = layer.register_forward_hook(copy_data)
        model(t_img)
        h.remove()
        descr = my_embedding.numpy()
        score = np.dot(descr, w)[0] + bias

        if score > threshold:
          image_detections.append(bbox_curent)
          image_scores.append(score)
```

2.5. Rularea programului principal

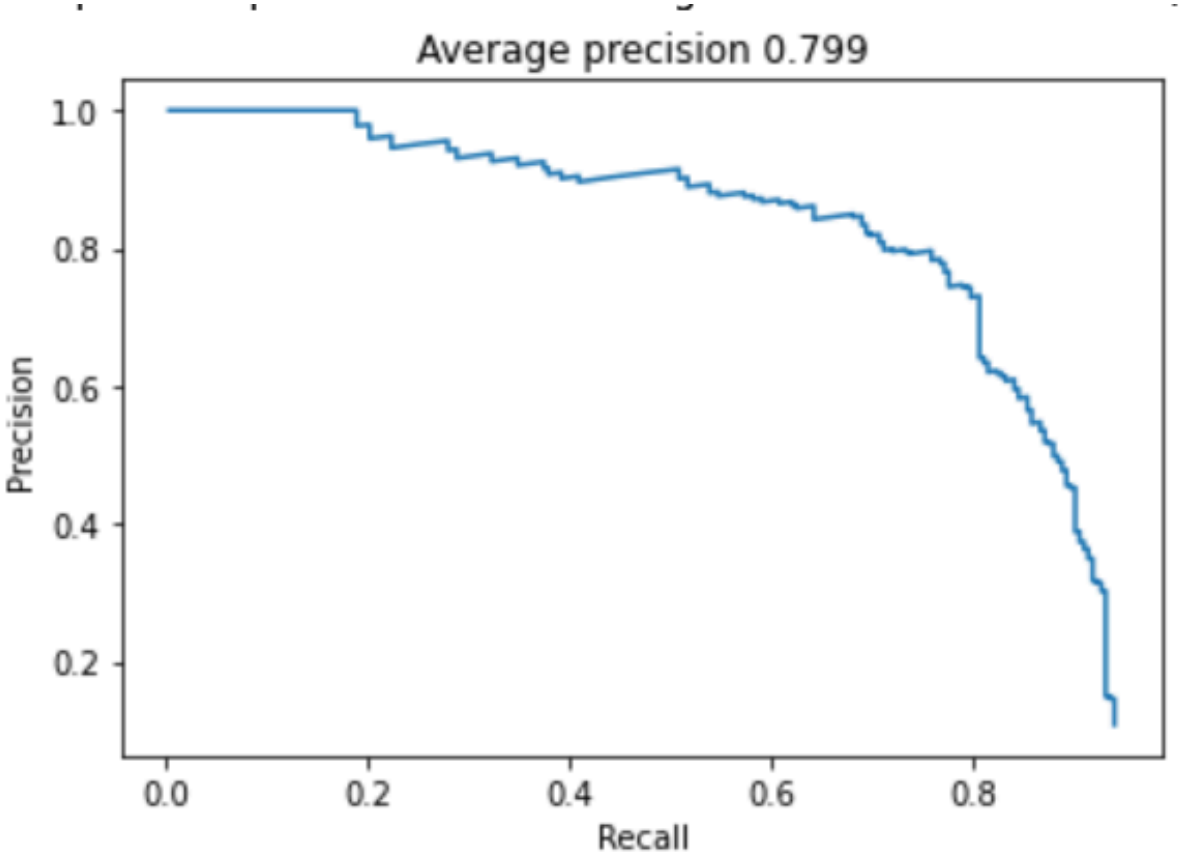
In fisierul run_task1.py am utilizat doar cele mai relevante functii pentru a face codul cat mai lizibil, unde am si salvat submisiile.

```
def main():
  # * GENEREZ EXEMPLE POZITIVE SI NEGATIVE
  generate_if_necesarry(IMG_SIZE)
  # * PARAMETERS
  if not os.path.exists(SAVED):
    os.mkdir(SAVED)
  # * FACIAL DETECTOR
  fd : FacialDetector = FacialDetector()
  # * positive features
  positive_features_path = os.path.join(SAVED,
'descriptori_exemple_pozitive.npy')
  # * analyze features only if necessary
  if os.path.exists(positive_features_path):
    positive_features = np.load(positive_features_path)
    print('Incarcat descriptori pozitive')
  else:
    print('Construiesc descriptori pozitive - Resnet')
    positive_features = fd.get_positive_descriptors_resnet()
    np.save(positive_features_path, positive_features)

  # * negative features
  negative_features_path = os.path.join(SAVED,
'descriptori_exemple_negative.npy')
  if os.path.exists(negative_features_path):
    negative_features = np.load(negative_features_path)
    print('Incarcat descriptori negative')
  else:
    print('Construiesc descriptori negative - Resnet')
    negative_features = fd.get_negative_descriptors_resnet()
    np.save(negative_features_path, negative_features)

  # * clasificator
  training_examples = np.concatenate((np.squeeze(positive_features),
np.squeeze(negative_features)), axis=0)
  train_labels = np.concatenate((np.ones(positive_features.shape[0]),
np.zeros(negative_features.shape[0])))
  fd.train_classifier(training_examples, train_labels)
```

```
    detections, scores, file_names = fd.new_run()

    if not os.path.exists(MY_DIR):
        os.mkdir(MY_DIR)
    if not os.path.exists(SAVE_SOLUTION_TASK1):
        os.mkdir(SAVE_SOLUTION_TASK1)

    np.save(SAVE_SOLUTION_TASK1 + 'detections_all_faces.npy', detections)
    np.save(SAVE_SOLUTION_TASK1 + 'scores_all_faces.npy', scores)
    np.save(SAVE_SOLUTION_TASK1 + 'file_names_all_faces.npy', file_names)
```

2.6. Rezultate

Dupa fine-tuning, rezultatul cel mai bun obținut a fost următorul:



3. Task 2 - Recunoastere faciala

Pentru recunoasterea faciala am încercat o abordare diferita fata de task 1, pentru care am reusit o performanta mai buna. Am antrenat o rețea neuronala convolutionala (la care am adaugat weights din preantrenare) pe care am antrenat-o ulterior pe fetele personajelor.

3.1. Generarea dataseturilor de antrenare si validare

Structura datasetului pe care il creeaza programul arata in felul următor:

```
.
├── train
│   ├── bart
│   ├── homer
│   ├── lisa
│   ├── marge
│   ├── unknown
│   └── negative
└── val
    ├── bart
    ├── homer
    ├── lisa
    ├── marge
    ├── unknown
    └── negative
```

Acesta este similar cu generarea pozitivelor si negativelor de la task 1, doar ca difera modul in care au fost asezate in foldere.

3.2. Retele Neuronale Convolutionale

Pentru realizarea recunoasterii faciale am folosit metoda de transfer learning folosind pytorch pe care am antrenat un model pe 6 clase. Am folosit dataloaders pentru a încarca datele. O functie foarte importanta este cea de train, care spre deosebire de Keras/Tensorflow, trebuie implementata manual.

```
def train_network(self, epochs = 20):
  for epoch in range(1, epochs + 1):
    train_acc, train_loss = self.compute_epoch(self.train_load,
training=True)
    val_acc, val_loss = self.compute_epoch(self.val_load,
training=False)


def compute_epoch(self, dataload, training = False):
  if training:
    self.model.train()
  else:
    self.model.eval()
```

```
  total_loss = 0.0
  total_correct = 0
  examples = 0

  for x, y in tqdm(dataload):
    if training:
      self.optimizer.zero_grad()

    x = x.to(self.device)
    y = y.to(self.device)

    pred = self.model.forward(x)
    loss = self.loss_function(pred, y)

    if training:
      loss.backward()
      self.optimizer.step()

    total_loss += loss.data.item() * x.size(0)
    total_correct  += (torch.max(pred, 1)[1] == y).sum().item()
    examples += x.shape[0]

  accuracy = total_correct / examples
  calc_loss = total_loss / len(dataload.dataset)

  return accuracy, calc_loss
```

3.3. Recunoastere faciala

Folosind un sliding window asemanator cu cel de la task-ul 1, am folosit modelul antrenat pentru a detecta o clasa corespunzătoare. De asemenea, pentru mai multe detectari corecte pentru aceeasi fata, voi pastra doar cea mai buna, folosind supresia maximala.

```
# ? COMPUTE NON MAXIMAL SUPRESSION
def non_maximal_suppression2(image_detections, image_scores,
image_labels):
    to_return = []
    iou_threshold = 0.3
    labels_set = nub(image_labels)
    for label in labels_set:
        best_bboxes = []
        zipall = zip(image_detections, image_scores, image_labels)
        # * filtrez dupa label
```

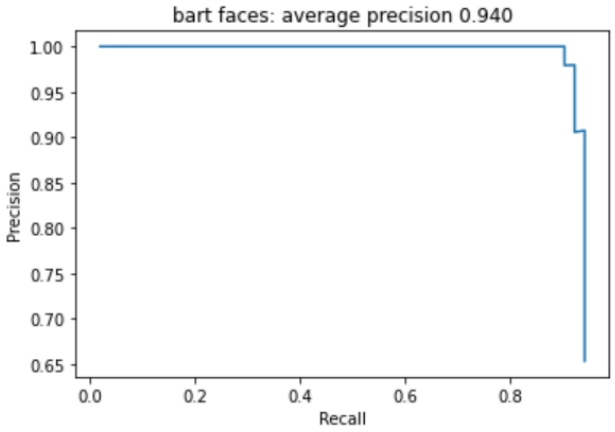
```
        filtered = filter(lambda x : x[2] == label , zipall)
        # * sortez dupa score
        sorted_data = sorted(filtered, key=lambda x: x[1], reverse=True)
        # * pastrez cele mai bune bounding boxes
        best_bboxes.append(sorted_data[0])
        sorted_data.pop()
        for bbox in sorted_data:
            should_remove = False
            for best_box in best_bboxes:
                if intersection_over_union(bbox[0], best_box[0]) >
iou_threshold or same_center(bbox[0], best_box[0]):
                    should_remove = True
            if should_remove == False:
                best_bboxes.append(bbox)
        to_return += best_bboxes
    unzipped = list(zip(*to_return))
    return unzipped[0], unzipped[1], unzipped[2]
```

In final, incarc predictiile in fisierele corespunzătoare.

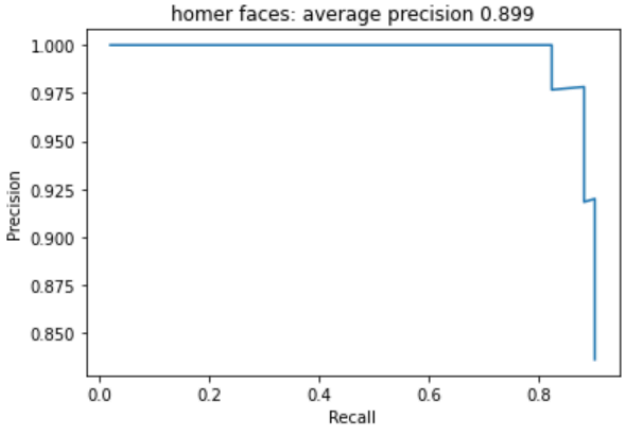
```
with torch.no_grad():
  for imgs, xmin, ymin, xmax, ymax in test_loader:
    imgs = imgs.cuda()
    predictions = model.forward(imgs)
    pred_size = len(predictions)

    for i in range(pred_size):
      coords = (int(xmin[i]), int(ymin[i]), int(xmax[i]), int(ymax[i]))
      label = np.argmax(predictions[i].cpu())
      if label != 5:
        detections.append(coords)
        scores.append(float(predictions[i][label]))
        labels.append(label)
```

3.4. Rezultate

```
(75, 4)
(75,)
(75, 4)
```

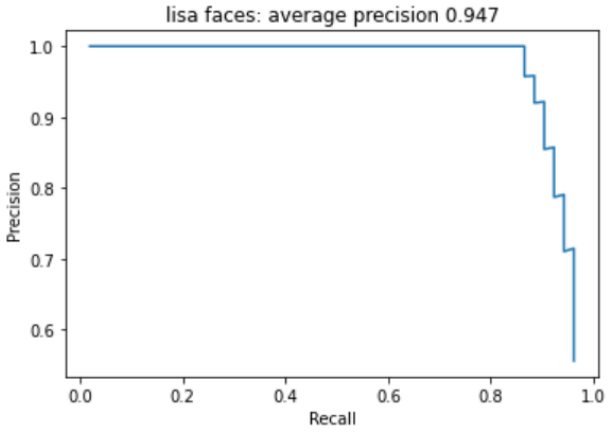



```
(55, 4)
(55,)
(55, 4)
```




```
(90, 4)
(90,)
(90, 4)
```

```
(90,)
(90, 4)
```




```
(83, 4)
(83,)
(83, 4)
```