

20 Aprilie 2022

STUDENTI

Facultatea de Matematică și Informatică,  
Universitatea din București

Laura-Maria Tender, grupa 334  
Nicolae Ducal, grupa 334

---

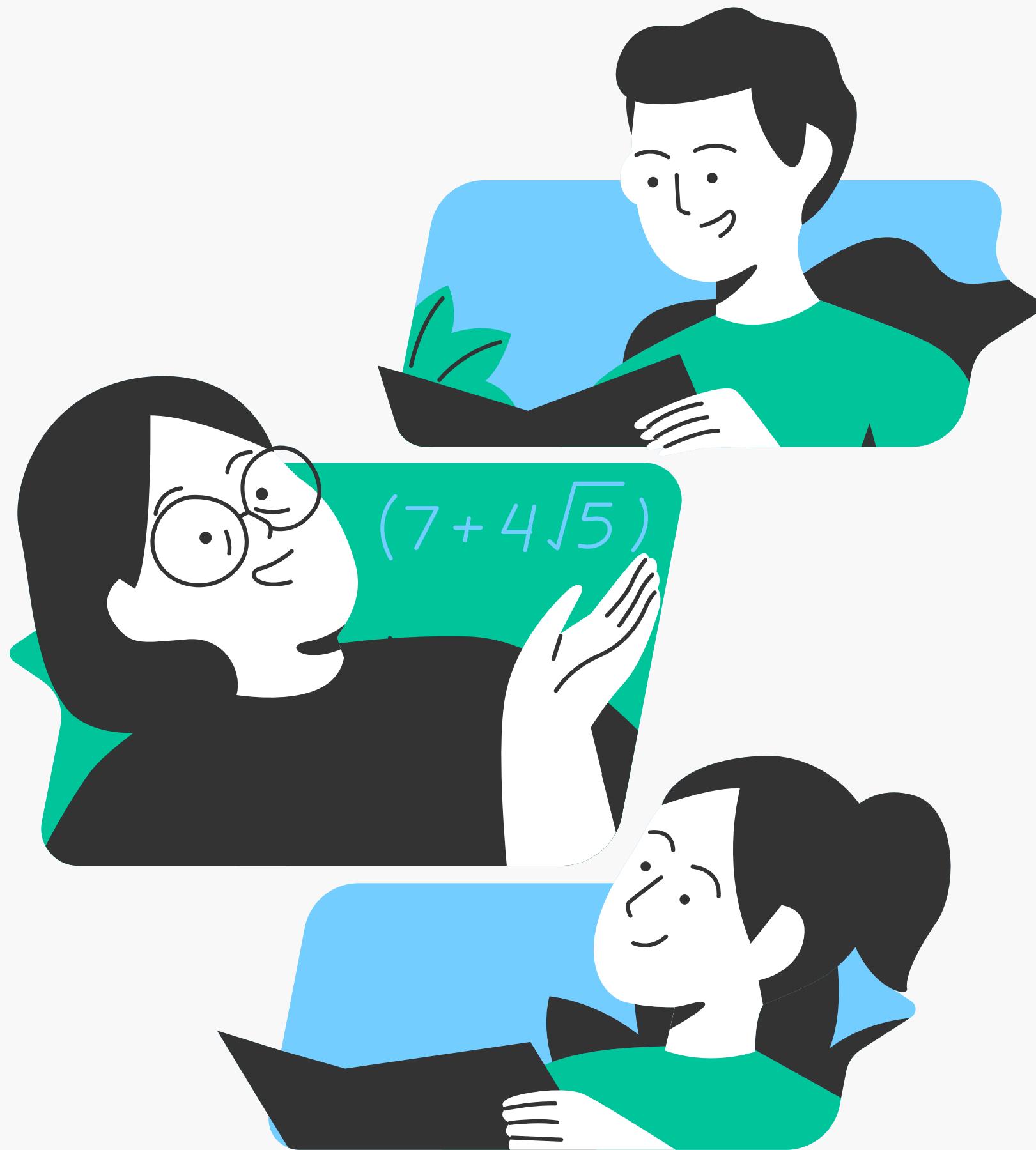
# Aplicații practice ale arhitecturii StargAN

Învățare Automată în Vedere Artificială, 2022

## Cuprins

	Page
I      Introducere	3
II     Ce este StarGAN?	4
III    Modele de StarGAN	6
IV    Face Detection	8
V    Îmbunătățirea bibliotecii	8
VI   Rezultate	10
VII Referințe	12

## I Introducere



În introducere vom prezenta ce își propune proiectul nostru. Codul nostru poate:

- să identifice fețele din imagini și video
- să schimbe atrbute ale acestor fețe folosind StarGAN

În continuare vom detalia abordările încercate și rezultatele obținute.

## II Ce este StarGAN?

### Problema de image-to-image translation

Problema de image-to-image translation constă în modificarea aspectului unei imagini cu alte particularități (de ex. persoană veselă în persoană tristă). Majoritatea soluțiilor ce se ocupă de acest task sunt bazate pe GAN-uri, iar până în anul 2018, modelele existente la moment puteau rezolva problema de image-to-image translation doar pe un domeniu. Pentru a soluționa această problemă pentru mai multe domenii (de ex. schimbarea expresiei faciale pentru 3 sau mai multe expresii) erau necesare antrenarea a  $k(k-1)$  generatori, unde  $k$  este numărul de domenii ale problemei. Această problemă este cunoscută ca multi-domain image-to-image translation.



## II Ce este StarGAN?

### StarGAN

Arhitectura StarGAN a fost introdusă în anul 2018 ca o alternativă pentru soluțiile existente la momentul de față (IcGAN, CycleGAN etc.) care nu puteau efectua image-to-image translation pe mai mult de două domenii. Arhitectura presupune utilizarea unui singur generator care poate învăța translațiile dintre toate domeniile, fără să avem nevoie de antrenare a unui generator diferit pentru fiecare două domenii.

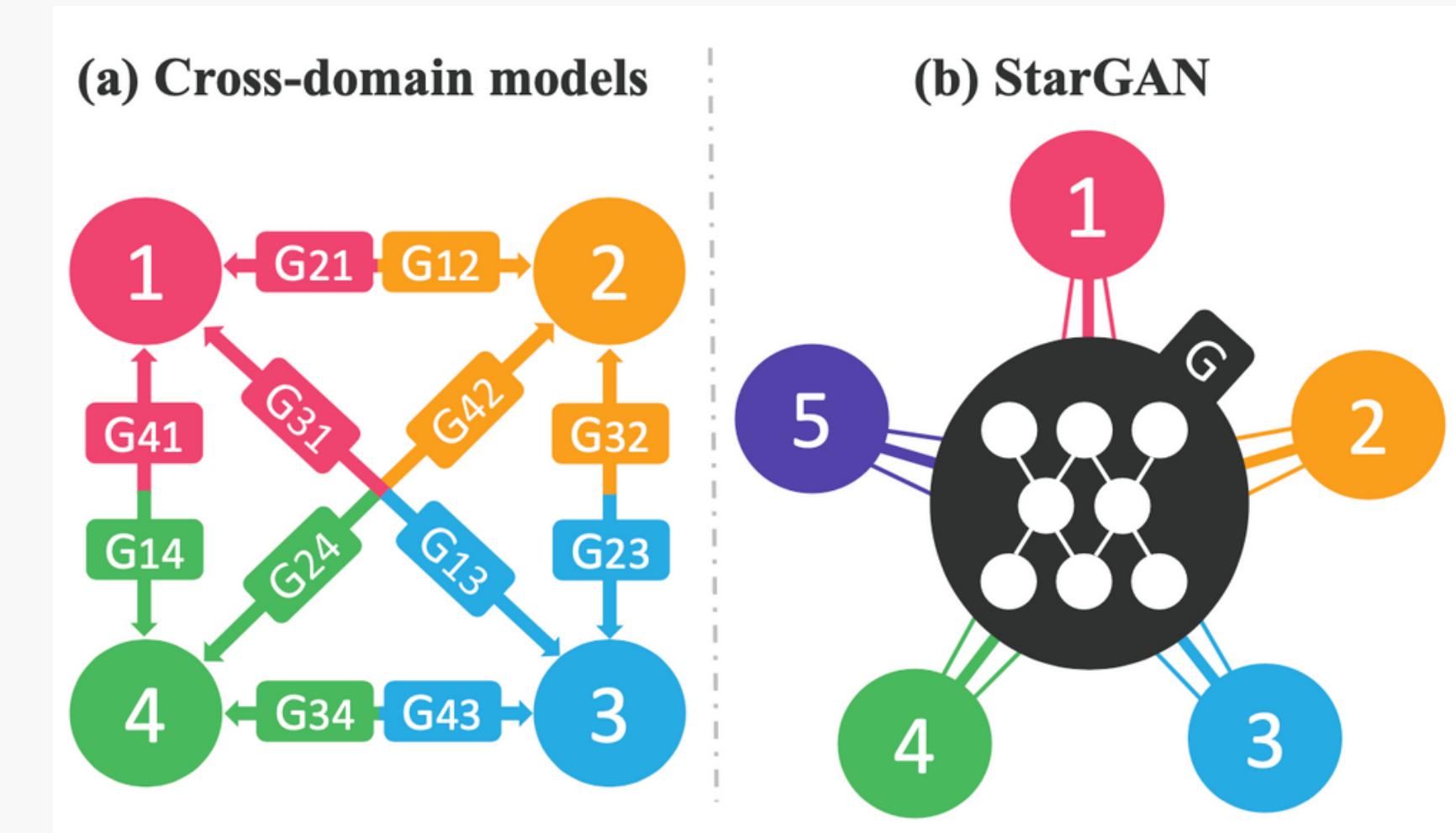
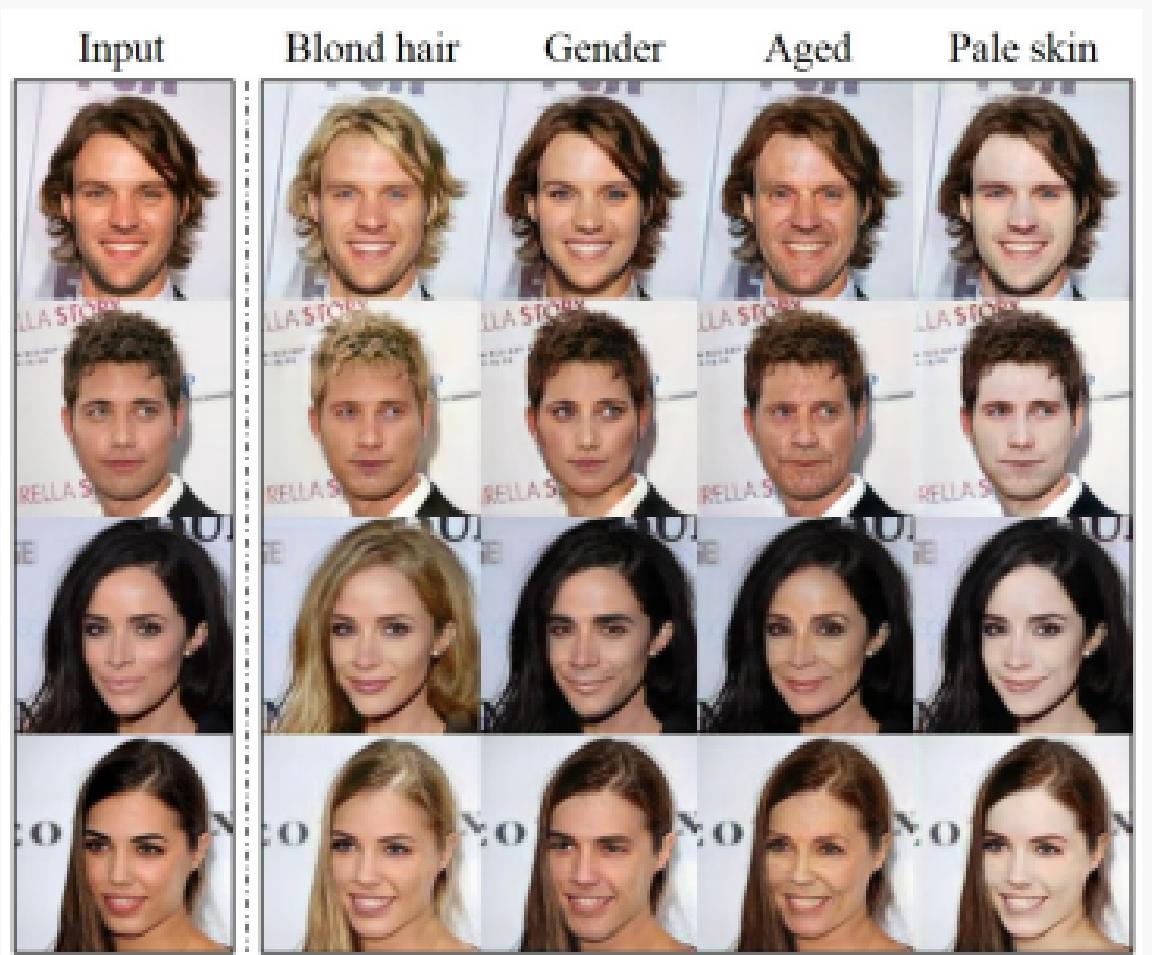


Figura 1. Comparația arhitecturii StarGAN cu cross-domain models.  
(a) Cross-domain models antrenează câte un generator diferit pentru a manipula cu translațiile dintre două domenii; (b) StarGAN antrenează un singur generator pentru același task

### III Modele de StarGAN

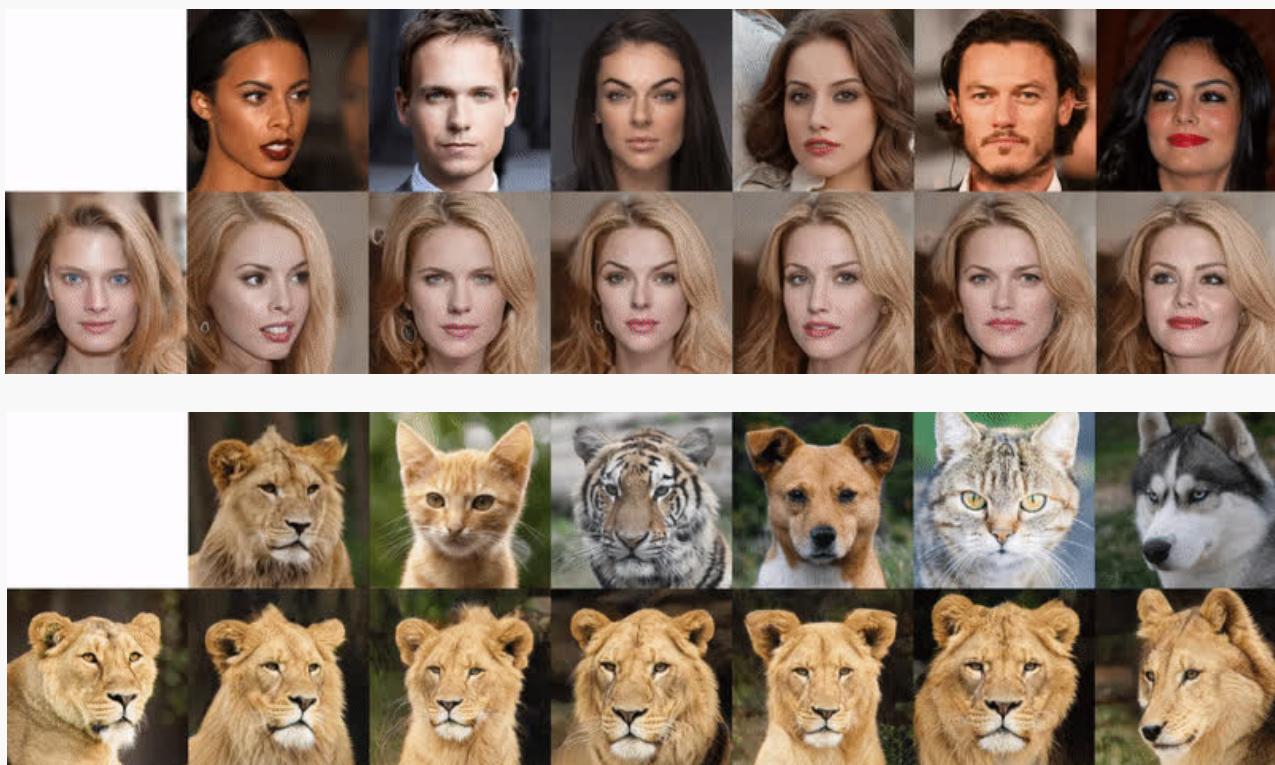
#### StarGAN

Prima versiune de StarGAN prezintă o soluție pentru problema multi-domain image-to-image translation. Practic, modelul primește ca input o imagine cu față unei persoane și modifică caracteristicile feței persoanei (de ex. vârstă, culoarea părului, sexul).

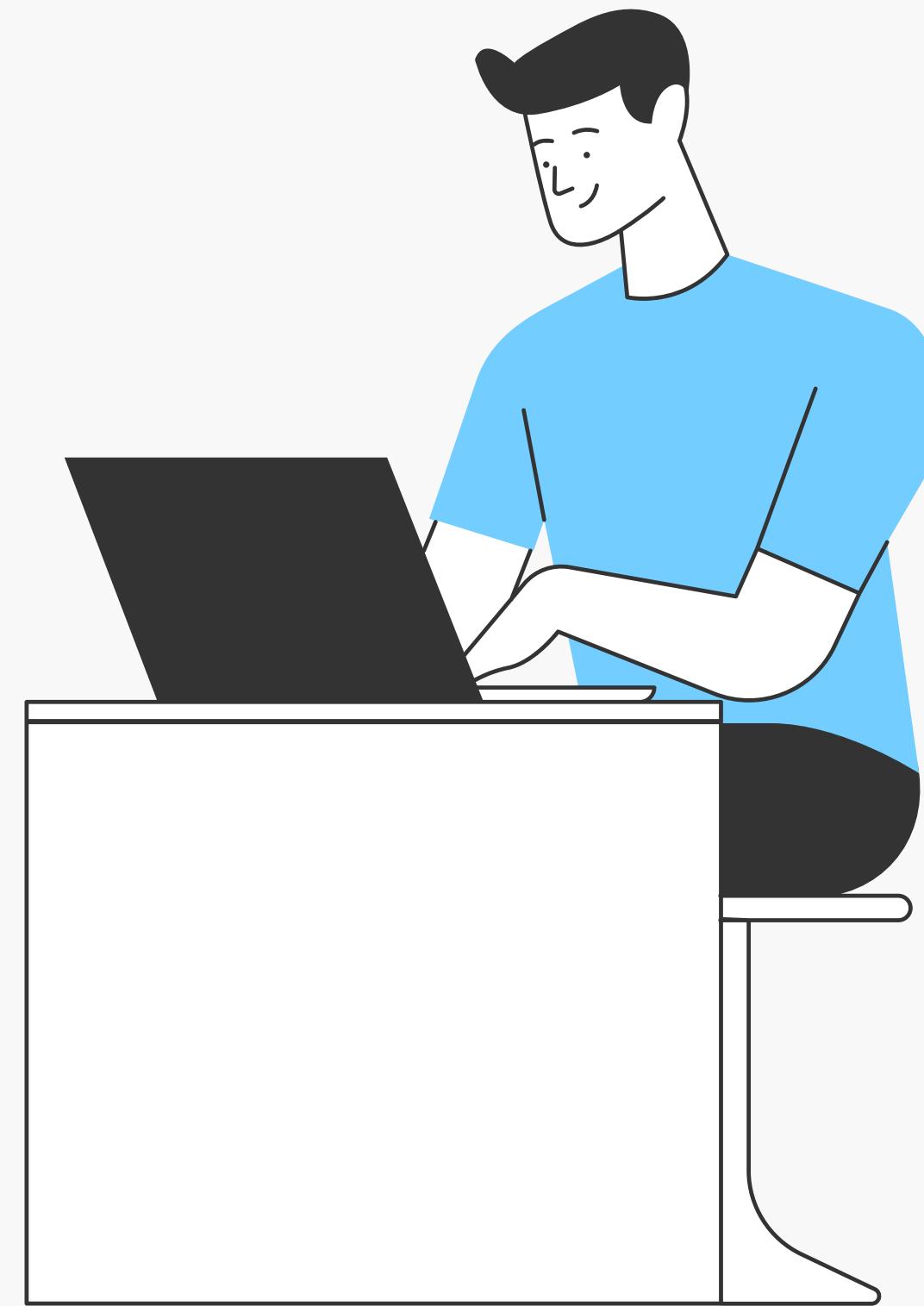


#### StarGAN-v2

StarGAN v2 este o arhitectură care propune o rezolvare pentru sinteza imaginilor, păstrând paradaigma multi-domain image-to-image transaltion. Modelul primește ca input două imagini și le combină, folosind una din imagini ca referință (din care vor fi extrase caracteristici ale feței) și cealaltă ca sursă (care va păstra forma și expresia feței),



### III Modele de StarGAN

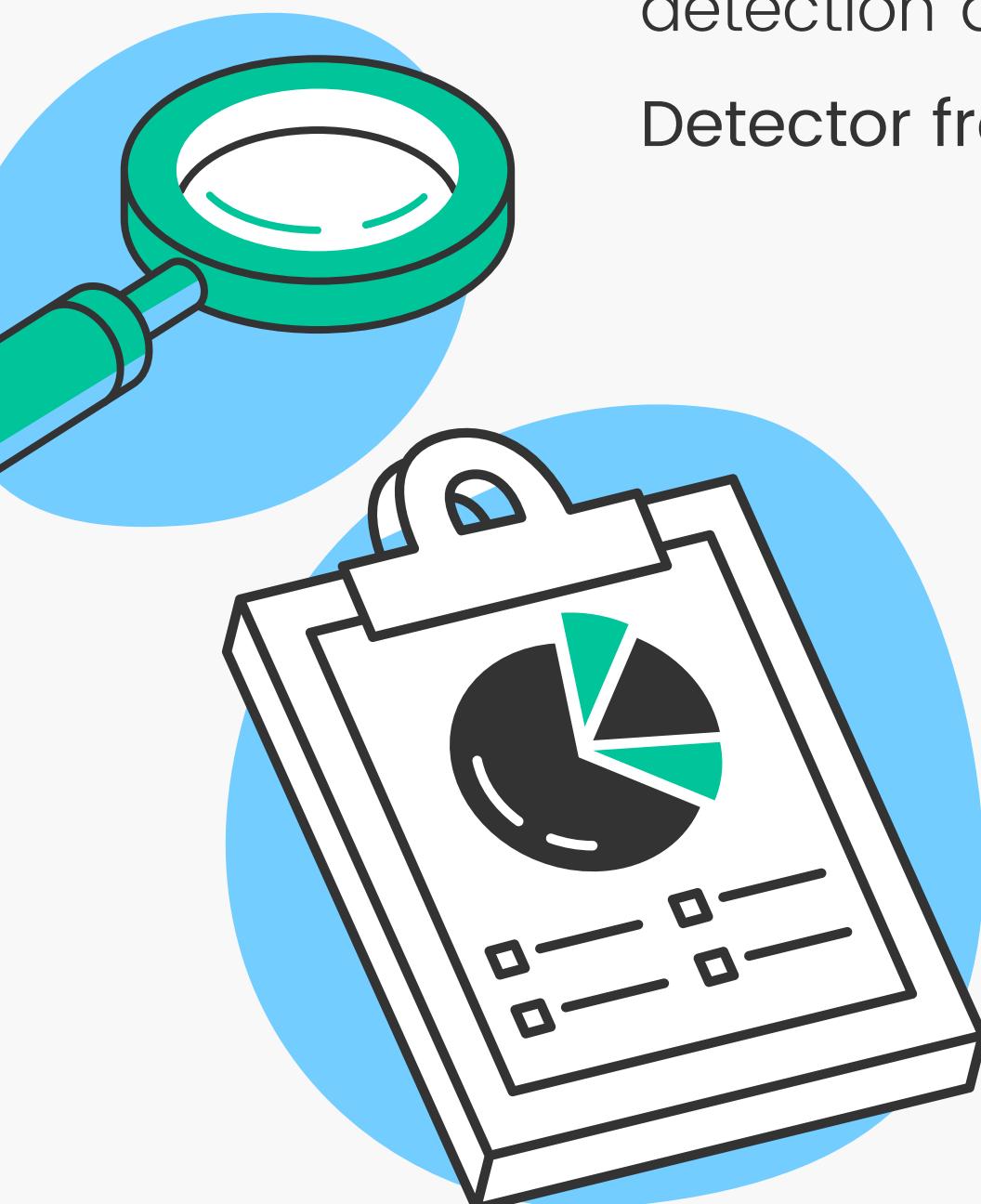


#### Modelul de StarGAN utilizat în cadrul proiectului

Dintre cele două versiuni de StarGAN, am decis să o folosim pe cea inițială, pentru a schimba particularitățile fețelor fără a utiliza imagini de referință.

De asemenea, există mai multe implementări ale arhitecturii de StarGAN cu ajutorul mai multor framework-uri specializează în crearea rețelelor neuronale, cum ar fi [PyTorch](#), [Tensorflow](#) și [nnabla](#). După utilizarea a tuturor trei variante, cea mai user-friendly și comodă pentru utilizare s-a dovedit a fi implementarea din framework-ul nnabla.

## IV Face Detection



Pentru început, proiectul nostru detectează fețele dintr-o imagine. Pentru face detection am încercat cele două abordări enumarate mai jos: MTCNN și CNN Face Detector from dlib.

### Metoda 1

MTCNN from facenet\_pytorch

### Multi-Task Cascaded Convolutional

A fost publicat în 2016 de Zhang et al. E unul dintre cele mai populare și precise tool-uri de face detection. Consta în 3 rețele neuronale în cascadă.

### Metoda 2

CNN Face Detection Model v1 from dlib

### MMOD CNN

MMOD CNN sau Max-Margin Object Detection CNN este un face detector precis și foarte robust, capabil să detecteze fețe din diferite unghiuri sau luminozități.

## IV Implementare (Metoda I)

Crearea detectorului

```
mtcnn = MTCNN(select_largest=False,  
               image_size=128, margin=40, min_face_size=20,  
               factor=0.7, thresholds=[0.8, 0.85, 0.85],  
               post_process = True,  
               keep_all = True,  
               device=device  
)
```

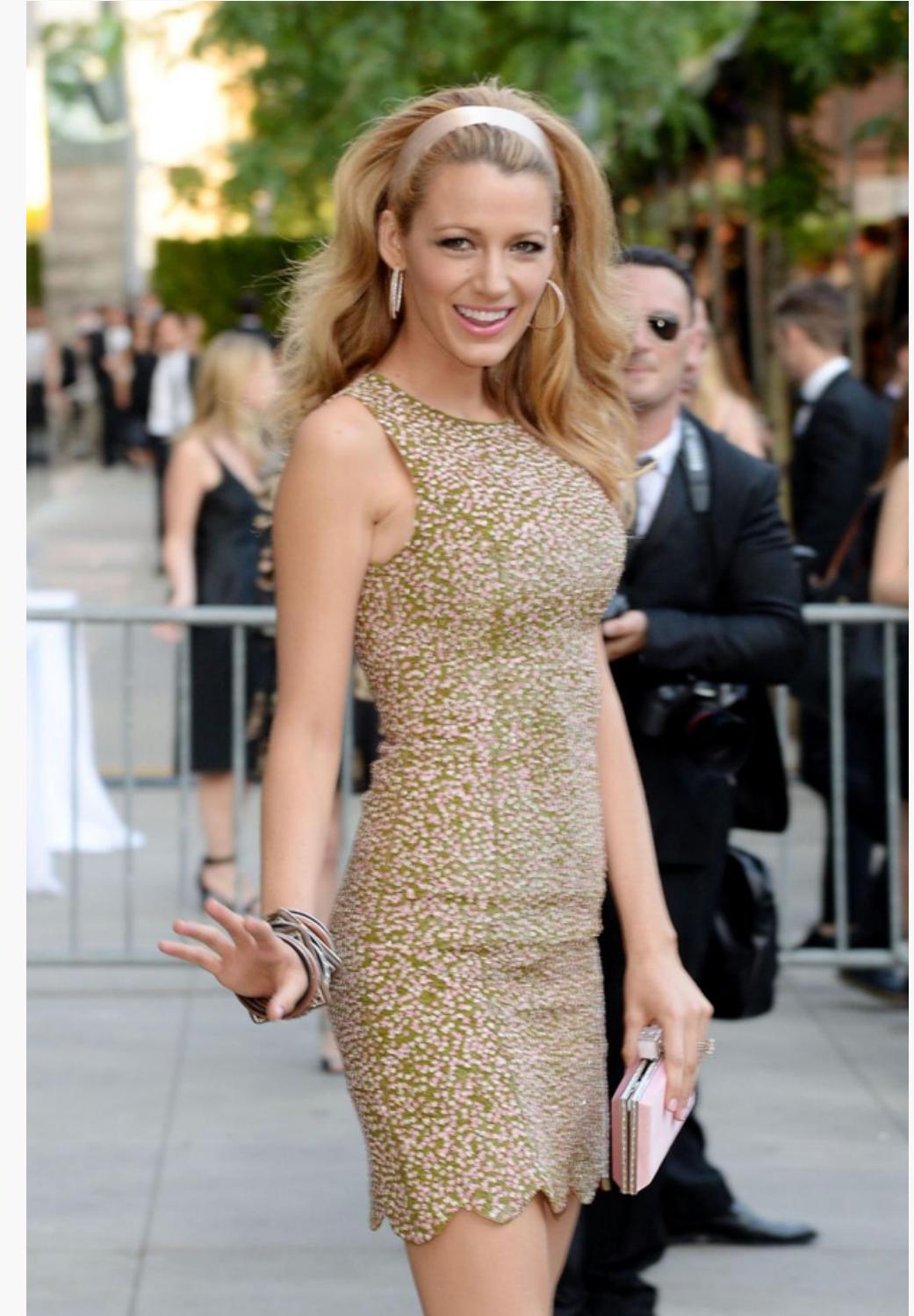
Detectia fețelor

```
aligned = []  
for x, y in loader:  
    x_aligned, prob = mtcnn(x, return_prob=True)  
    for i in range(len(x_aligned)):  
        print(f'picture no {i+1}')  
        if x_aligned[i] is not None:  
            print('Face detected with probability: {:.8f}'.format(prob[i]))  
            aligned.append(x_aligned[i])
```

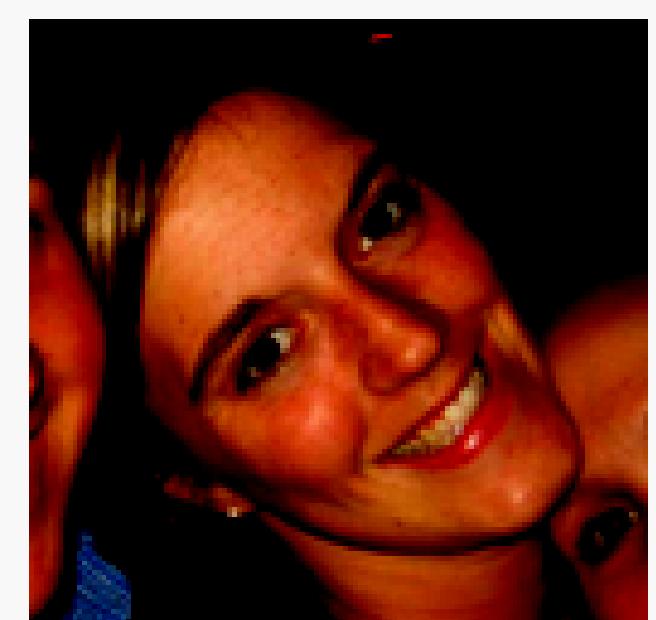
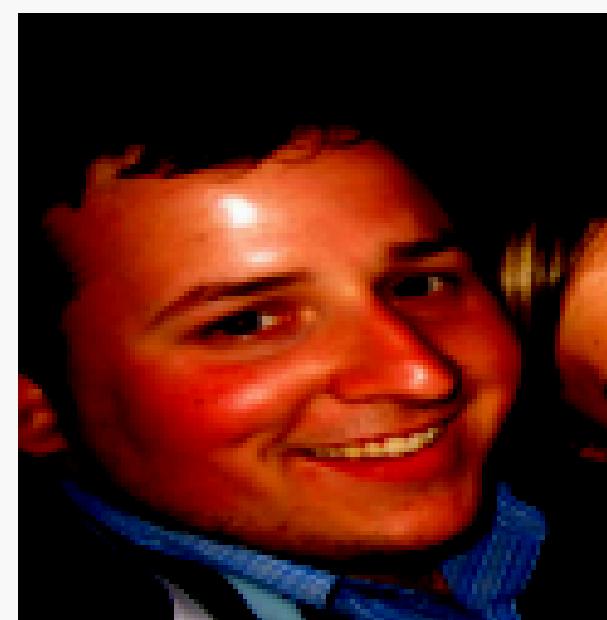
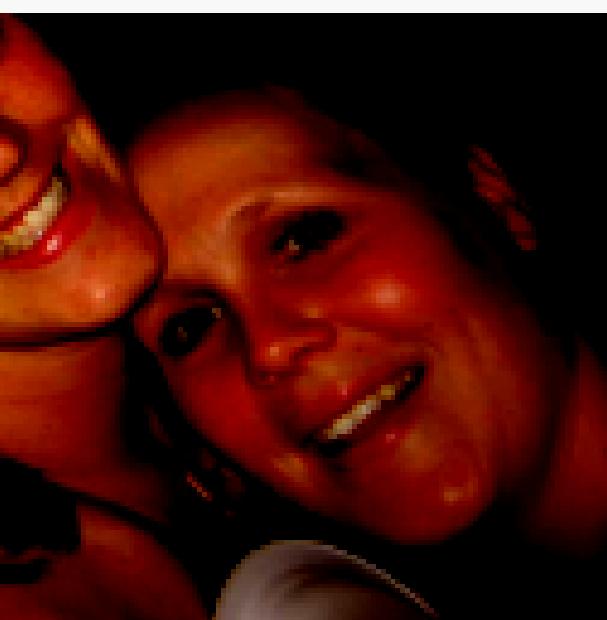
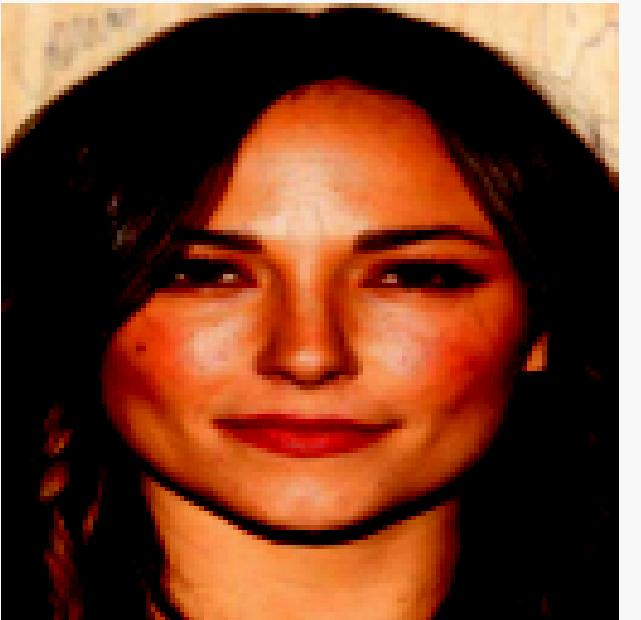
Afișarea rezultatelor

```
plt.figure(figsize=(20, 10))  
columns = 5  
for i, image in enumerate(aligned):  
    plt.subplot(len(aligned) / columns + 1, columns, i + 1)  
    plt.imshow(np.transpose(image.cpu().numpy()))
```

## Câteva poze din dataset-ul inițial



## Rezultate



## IV Implementare (Metoda II)

```
faces = {}
face_detector = dlib.cnn_face_detection_model_v1("mmod_human_face_detector.dat")
nr_faces = 0

for image_name in image_names:
    image = io.imread(folder_name + image_name)

    detected_faces = face_detector(cv2.cvtColor(image[..., ::-1].copy(), cv2.COLOR_BGR2GRAY))
    detected_faces = [[d.rect.left(), d.rect.top(), d.rect.right(), d.rect.bottom()] for d in detected_faces]
    nr_faces += len(detected_faces)
    faces[image_name] = (image, detected_faces)
```

## IV Implementare (Metoda II)

```
def add_margin(img, corners, ratio = 0.4):
    w, H, _ = img.shape
    x1, y1, x2, y2 = corners

    w = x2 - x1
    h = y2 - y1

    delta_x = int(ratio*w)
    delta_y = int(ratio*h)

    new_x1 = max(x1 - delta_x, 0)
    new_x2 = min(x2 + delta_x, H)
    new_y1 = max(y1 - delta_y, 0)
    new_y2 = min(y2 + delta_y, w)

    return [new_x1, new_y1, new_x2, new_y2]

def crop(img, corners):
    x1, y1, x2, y2 = corners
    new_img = img[y1:y2, x1:x2]
    return new_img

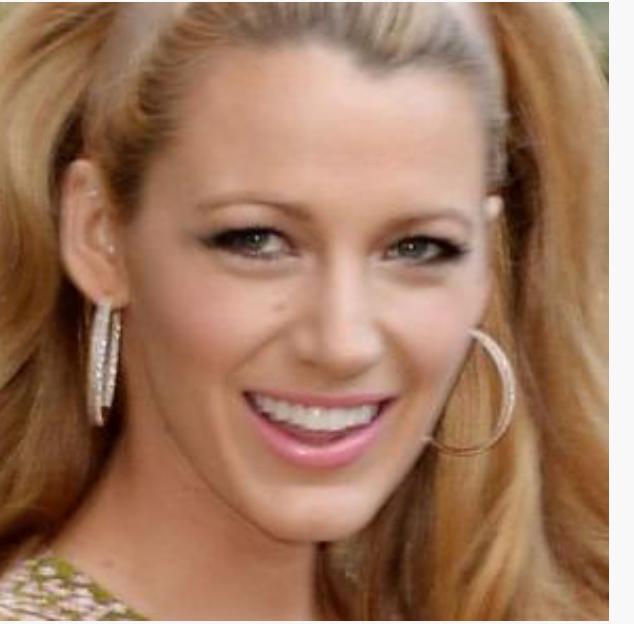
plt.figure(figsize=(20, 10))
columns = 5
i = 0

for image, face_corners in faces.values():
    for corners in face_corners:
        new_corners = add_margin(image, corners)

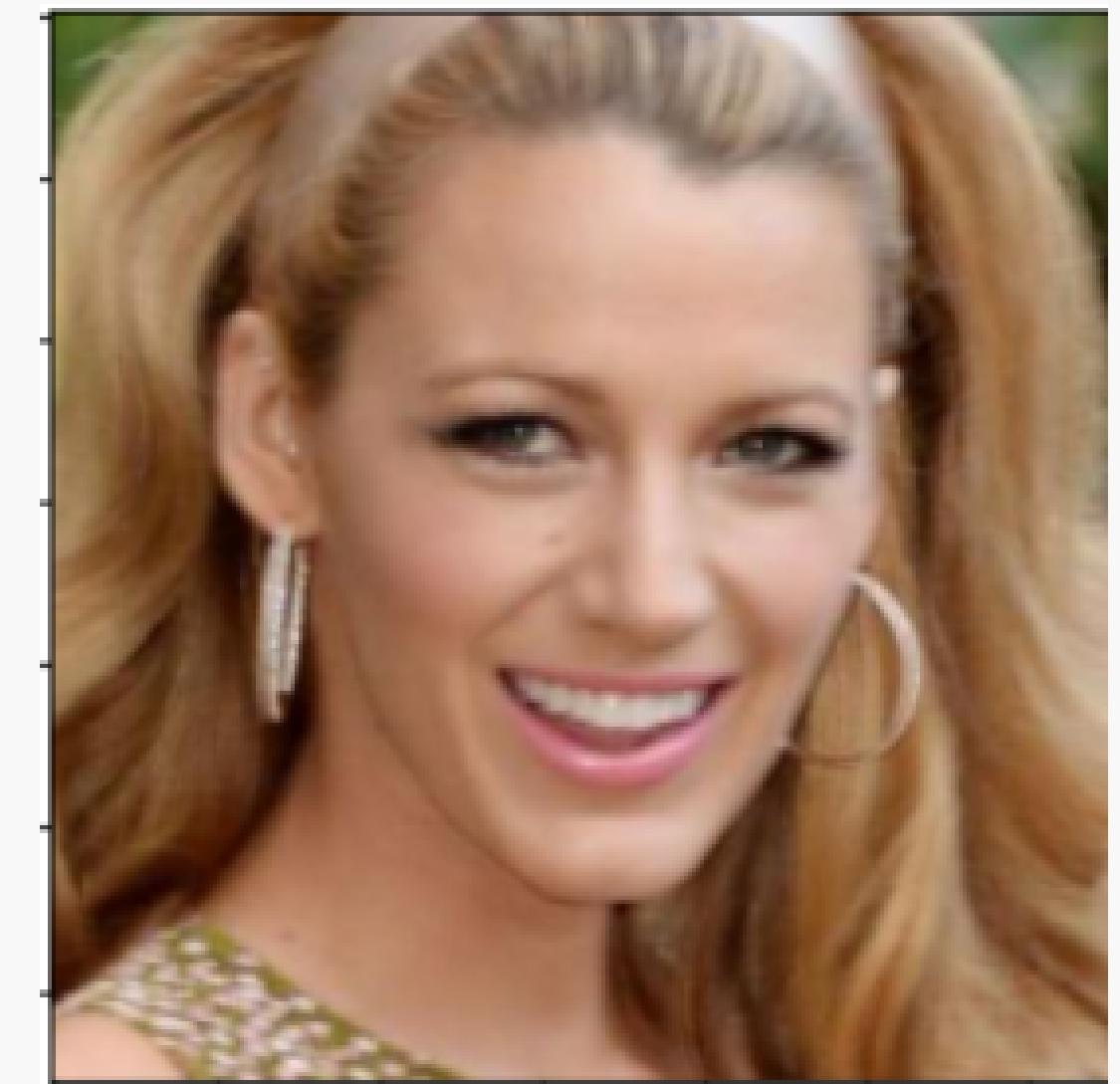
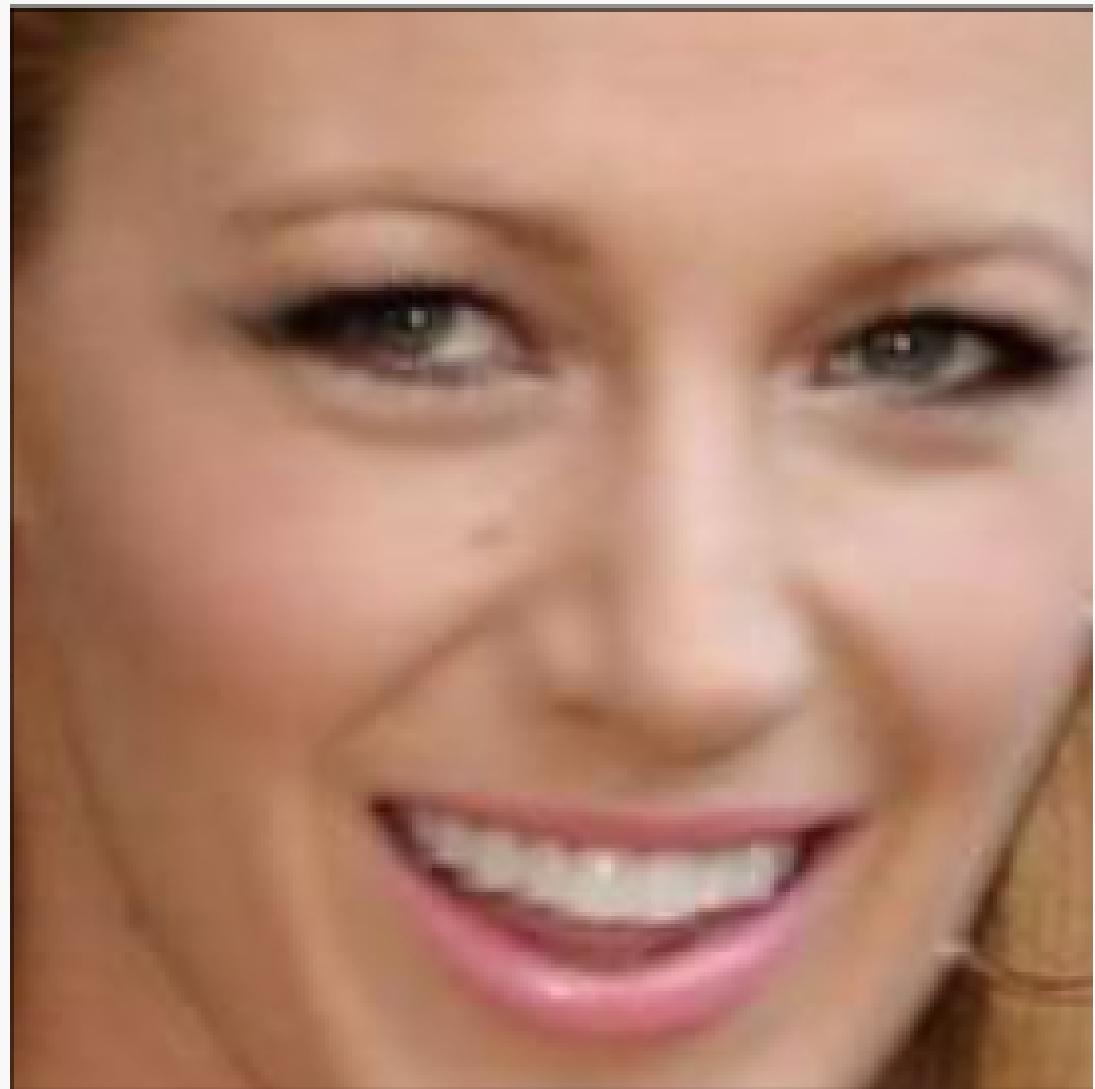
        new_img = crop(image, new_corners)

        plt.subplot(nr_faces / columns + 1, columns, i + 1)
        plt.imshow(new_img)
```

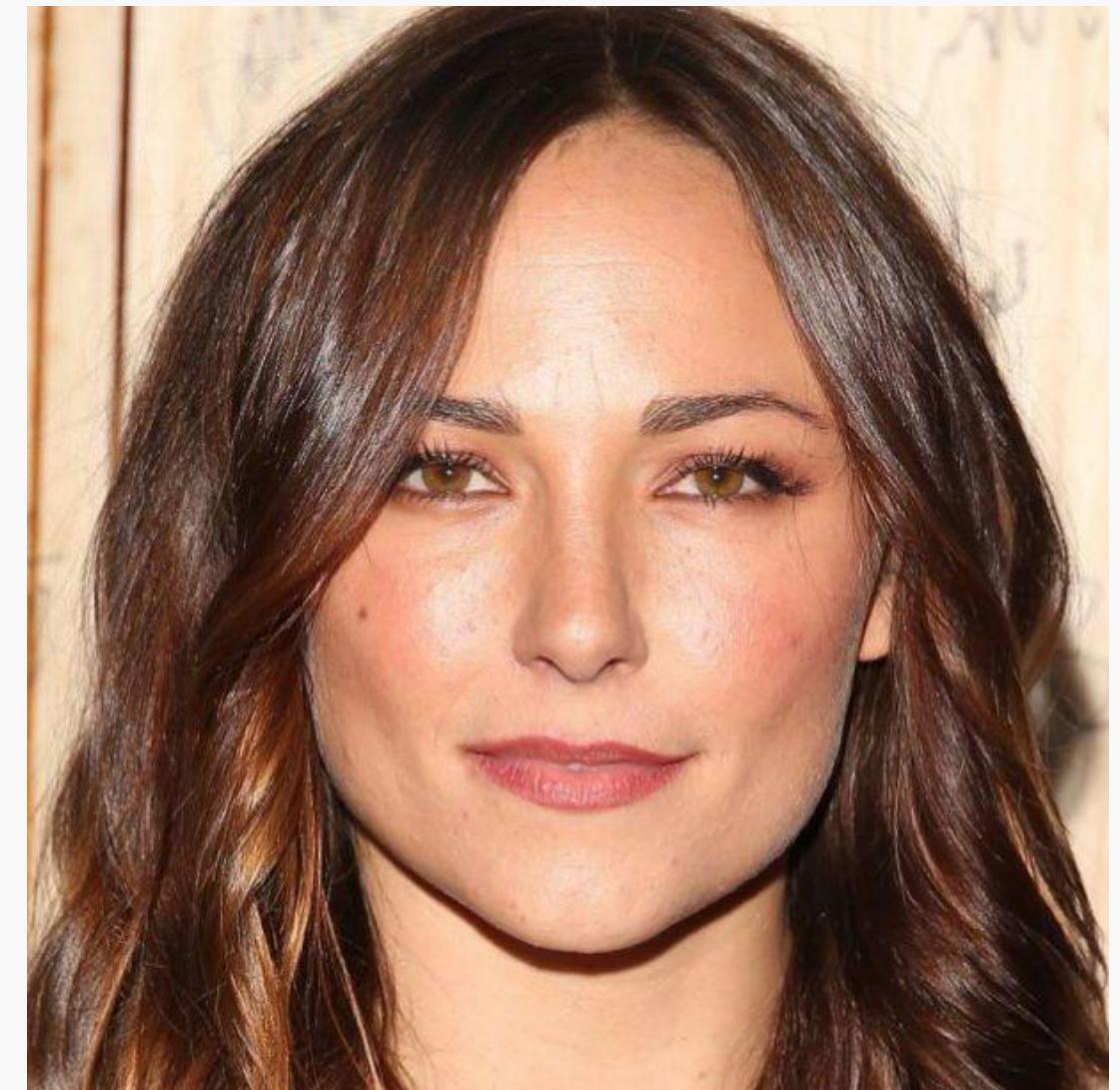
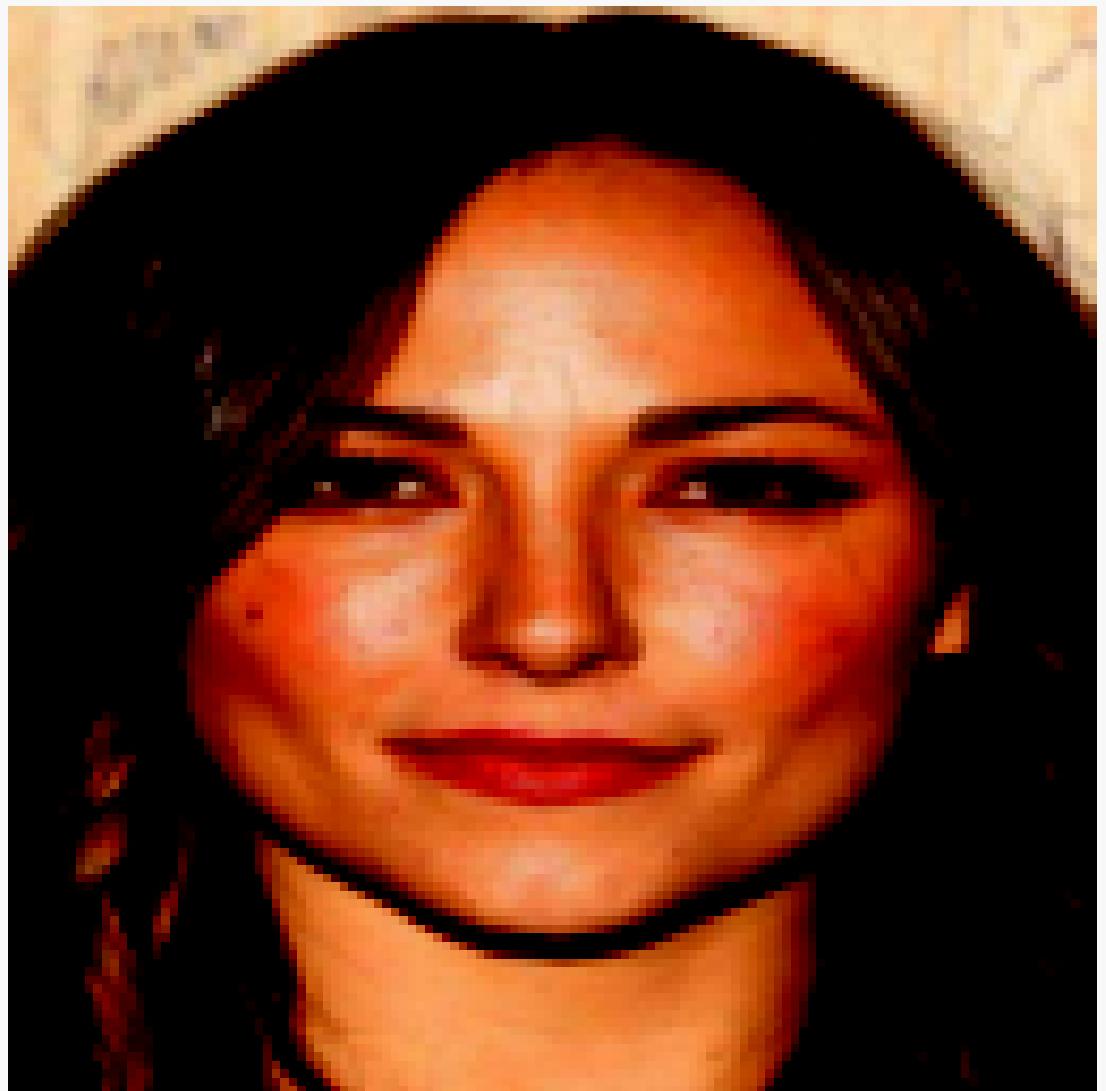
## Rezultate



#### IV Rezultate obținute prin add\_margin



#### IV Comparație între cele două metode

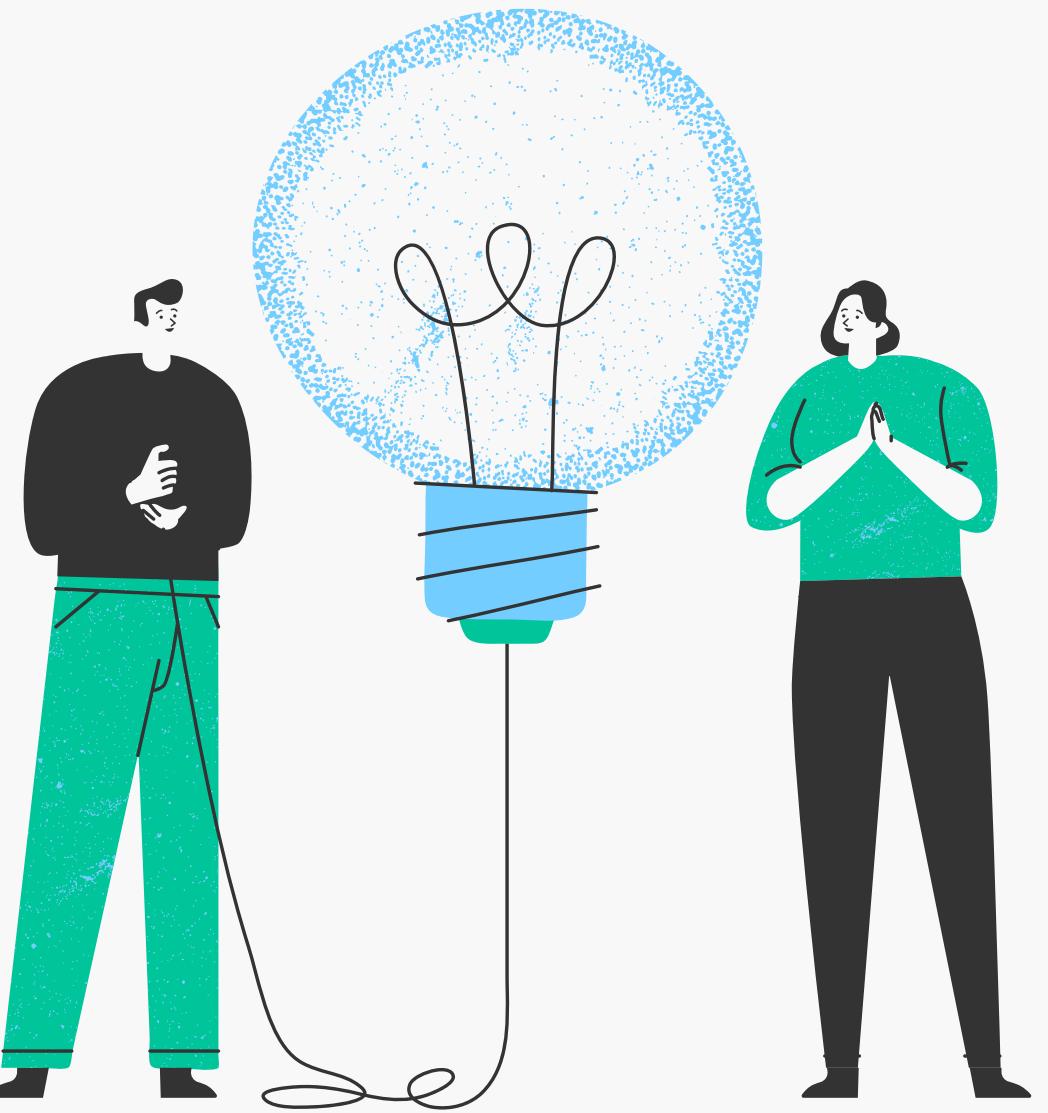


## V Îmbunătățirea bibliotecii

### Îmbunătățirea bibliotecii nnabla/stargan

Una din dificultățile pe care le-am întâlnit în cadrul proiectului a fost lipsa unui mod ușor de utilizare a modelului de StarGAN în cod. Am testat pentru acest proiect modelele preantrenate implementate în PyTorch și nnabla. Ambele implementări necesită rularea modelului din linia de comandă și stocarea pozelor în fișiere. Acest lucru este destul de incomod la partea de procesare a pozelor real-time sau în cod.

Pentru a eficientiza acest proces, am decis să adăugăm funcții adiționale în bibliotecile existente, pentru a putea efectua procesarea imaginilor direct din cod. Îmbunătățirea a fost efectuată pe ambele implementări, PyTorch și nnabla, cea din urmă fiind mai ușor de utilizat și mai intuitivă.



## V Implementarea pentru rularea din cod a StarGAN-ulu

```
def run_stargan_on_images(image_list, used_config, paramfile, attributes=[1, 0, 0, 1, 1]):  
    """  
    Run StarGAN on a list of images and return a list of generated modified images  
    """  
  
    assert os.path.isfile(paramfile) and  
        paramfile.split("/")[-1] == used_config["pretrained_params"],  
        "Corresponding parameter file not found."  
  
    print("Learned attributes choice: {}".format(  
        used_config["selected_attrs"]))  
  
    # Prepare Generator and Discriminator based on user config.  
    generator = functools.partial(  
        model.generator,  
        conv_dim=used_config["g_conv_dim"],  
        c_dim=used_config["c_dim"],  
        repeat_num=used_config["g_repeat_num"])  
  
    x_real = nn.Variable(  
        [1, 3, used_config["image_size"], used_config["image_size"]])  
    label_trg = nn.Variable([1, used_config["c_dim"], 1, 1])  
    with nn.parameter_scope("gen"):  
        x_fake = generator(x_real, label_trg)  
    x_fake.persistent = True  
  
    nn.load_parameters(paramfile) # load learned parameters.
```

```
    # Get fake images attributes  
    assert len(attributes) == used_config['c_dim'],  
        f'Attributes list should contain {used_config["c_dim"]} elements'  
    attributes = np.array(attributes)  
  
    result = []  
    for img in image_list:  
        result.append(generate_from_image(img, attributes,  
                                         x_real, x_fake, label_trg, used_config))  
  
    return np.asarray(result)  
  
def run_stargan_on_image(img, used_config, paramfile, attributes=[1, 0, 0, 1, 1]):  
    """  
    Run StarGAN on a single image and return a single modified generated image  
    """  
  
    return run_stargan_on_images([img], used_config, paramfile, attributes)[0]
```

## V Implementarea pentru rularea din cod a StarGAN-ulu

```
def generate_from_image(image, attributes, x_real, x_fake, label_trg, used_config):
    """
    Generates a fake image using StarGAN
    """

    # Perform image transformation
    new_img = transform_img(image, used_config)

    x_real.d = new_img

    # Generate target domain based on user input.
    label_trg.d = np.reshape(attributes, label_trg.shape)

    # Execute image translation.
    x_fake.forward(clear_no_need_grad=True)

    fake_img = x_fake.d[0]
    fake_img = (fake_img * 0.5) + 0.5
    fake_img = fake_img.transpose((1, 2, 0))

    return fake_img
```

```
def transform_img(img, used_config):
    image_size = used_config["image_size"]

    # Move from channels last to channels first so the model can handle it
    image = img.copy().transpose((2, 0, 1))

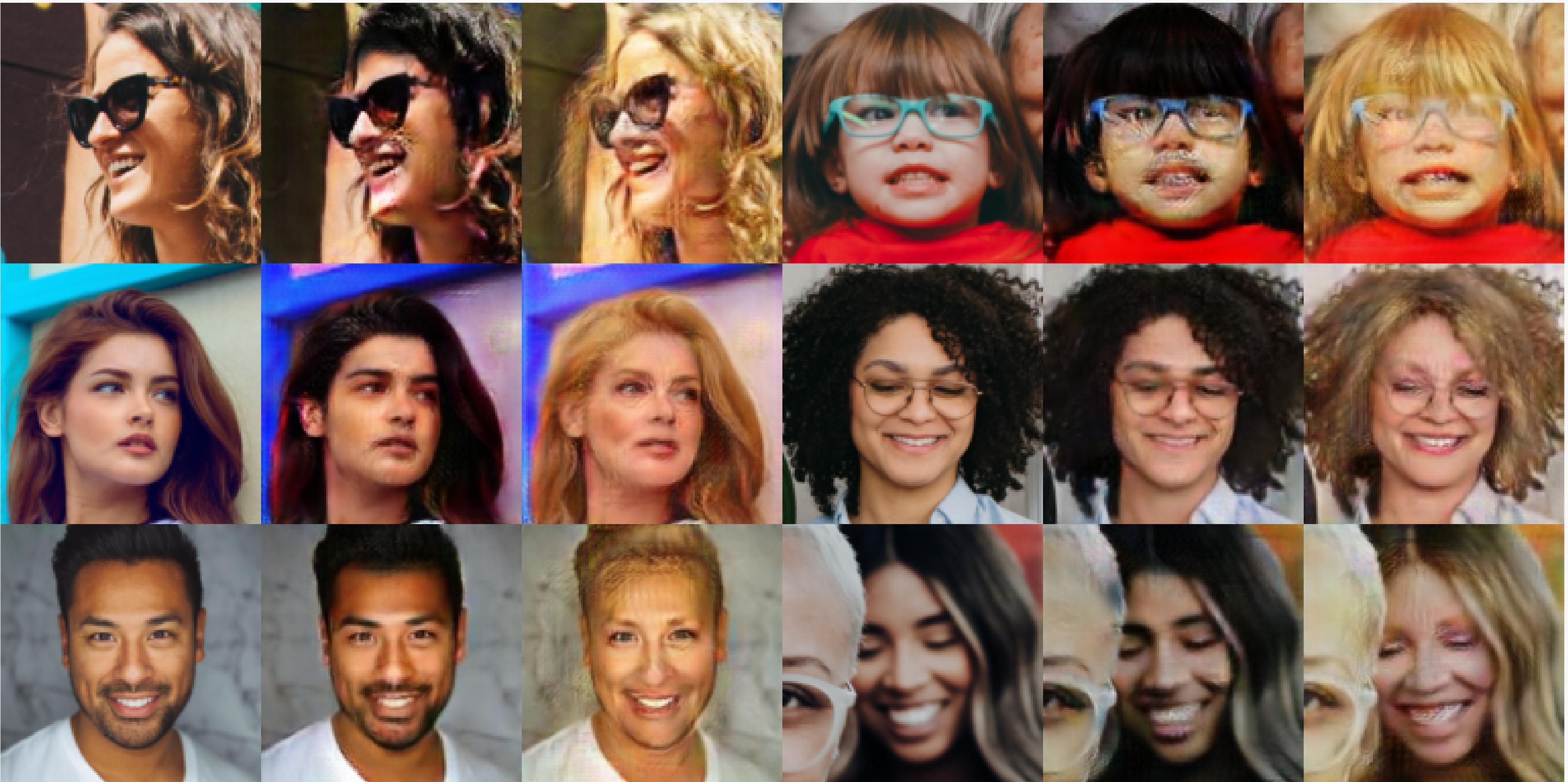
    if image.dtype == np.uint8:
        # Clip image's value from [0, 255] -> [0.0, 1.0]
        image = image / 255.0
        image = (image - 0.5) / 0.5 # Normalize
        image = imresize(image, (image_size, image_size),
                         interpolate='bilinear', channel_first=True)

    return image
```

## VI Rezultate obținute



## VI Rezultate obținute



20 Aprilie 2022

STUDENȚI

Facultatea de Matematică și Informatică,  
Unibuc

Laura-Maria Tender, grupa 334  
Nicolae Ducal, grupa 334

---

**Mulțumim pentru atenție!**

Învățare Automată în Vedere Artificială, 2022

## Referințe

---

1. StarGAN: Unified Generative Adversarial Networks for Multi-Domain Image-to-Image Translation (<https://arxiv.org/abs/1711.09020>)
2. StarGAN v2: Diverse Image Synthesis for Multiple Domains (<https://arxiv.org/abs/1912.01865>)
3. <https://github.com/sony/nnabla-examples/tree/master/image-translation/stargan>
4. <https://colab.research.google.com/github/sony/nnabla-examples/blob/master/interactive-demos/stargan.ipynb#scrollTo=ca5MWxo2u7Z0>
5. <https://github.com/timesler/facenet-pytorch>
6. <https://www.kaggle.com/code/timesler/guide-to-mtcnn-in-facenet-pytorch/notebook>
7. <https://github.com/timesler/facenet-pytorch/blob/master/examples/infer.ipynb>

Codul nostru poate fi găsit aici: <https://github.com/DreamUnibucTeam/Practical-Applications-of-StarGAN>