Assignment B.5 - Modern face recognition with deep learning

Have you noticed that Facebook has developed an uncanny ability to recognize your friends in your photographs? In the old days, Facebook used to make you to tag your friends in photos by clicking on them and typing in their name. Now as soon as you upload a photo, Facebook tags everyone for you like magic. This technology is called face recognition. Facebook's algorithms are able to recognize your friends' faces after they have been tagged only a few times. It's pretty amazing: these algorithms can recognize faces with 98% accuracy, which is pretty much as good as humans can do!

As a human, your brain is wired to recognize faces automatically and instantly. Computers are not capable of doing this, so you have to teach them how to tackle each step in this process. Specifically, a face recognition system goes through four steps: find faces in the image, analyze their facial features, compare against known faces, and make a prediction of the corresponding persons. Here's described the full pipeline.

Table of contents

In this assignment, you will tackle several problems related to face recognition:

1. **Face detection**. Look at a picture and find all the faces in it. -5
- **Pose estimation**. Understand where the face is turned and correct its pose. 5
- **Face encoding**. Pick up unique features from a face that can be used to distinguish it from others. 5
- **Face recognition**. Compare the unique features of a face to those of all the people in a database. 5
- **Personal dataset**. Build a custom face recognition dataset. 2

By the end of this notebook, you will have your own face recognition system.

Required packages

Here are the packages you will need during the assignment.

- Numpy
- Keras
- OpenCV
- Dlib.

Note: In Anaconda Navigator, the package `dlib` can be installed from the **conda-forge** channel. In Google Colab, everything is readily available

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

```
pip install --upgrade tensorflow
```

```
Requirement already satisfied: tensorflow in
/usr/local/lib/python3.10/dist-packages (2.18.0)
Requirement already satisfied: absl-py>=1.0.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (24.3.25)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1
in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.6.0)
Requirement already satisfied: google-pasta>=0.1.1 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: libclang>=13.0.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (18.1.1)
Requirement already satisfied: opt-einsum>=2.3.2 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (3.4.0)
Requirement already satisfied: packaging in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (24.2)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!
=4.21.3,!=4.21.4,!=4.21.5,<6.0.0dev,>=3.20.3 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (4.25.5)
Requirement already satisfied: requests<3,>=2.21.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (2.32.3)
Requirement already satisfied: setuptools in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (75.1.0)
Requirement already satisfied: six>=1.12.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (2.5.0)
Requirement already satisfied: typing-extensions>=3.6.6 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (4.12.2)
Requirement already satisfied: wrapt>=1.11.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.17.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.68.1)
Requirement already satisfied: tensorboard<2.19,>=2.18 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (2.18.0)
Requirement already satisfied: keras>=3.5.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (3.5.0)
Requirement already satisfied: numpy<2.1.0,>=1.26.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.26.4)
Requirement already satisfied: h5py>=3.11.0 in
```

```
/usr/local/lib/python3.10/dist-packages (from tensorflow) (3.12.1)
Requirement already satisfied: ml-dtypes<0.5.0,>=0.4.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (0.4.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (0.37.1)
Requirement already satisfied: wheel<1.0,>=0.23.0 in
/usr/local/lib/python3.10/dist-packages (from astunparse>=1.6.0-
>tensorflow) (0.45.1)
Requirement already satisfied: rich in /usr/local/lib/python3.10/dist-
packages (from keras>=3.5.0->tensorflow) (13.9.4)
Requirement already satisfied: namex in
/usr/local/lib/python3.10/dist-packages (from keras>=3.5.0-
>tensorflow) (0.0.8)
Requirement already satisfied: optree in
/usr/local/lib/python3.10/dist-packages (from keras>=3.5.0-
>tensorflow) (0.13.1)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0-
>tensorflow) (3.4.0)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0-
>tensorflow) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0-
>tensorflow) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0-
>tensorflow) (2024.8.30)
Requirement already satisfied: markdown>=2.6.8 in
/usr/local/lib/python3.10/dist-packages (from tensorboard<2.19,>=2.18-
>tensorflow) (3.7)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0
in /usr/local/lib/python3.10/dist-packages (from
tensorboard<2.19,>=2.18->tensorflow) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from tensorboard<2.19,>=2.18-
>tensorflow) (3.1.3)
Requirement already satisfied: MarkupSafe>=2.1.1 in
/usr/local/lib/python3.10/dist-packages (from werkzeug>=1.0.1-
>tensorboard<2.19,>=2.18->tensorflow) (3.0.2)
Requirement already satisfied: markdown-it-py>=2.2.0 in
/usr/local/lib/python3.10/dist-packages (from rich->keras>=3.5.0-
>tensorflow) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in
/usr/local/lib/python3.10/dist-packages (from rich->keras>=3.5.0-
>tensorflow) (2.18.0)
Requirement already satisfied: mdurl~=0.1 in
/usr/local/lib/python3.10/dist-packages (from markdown-it-py>=2.2.0-
>rich->keras>=3.5.0->tensorflow) (0.1.2)
```

```
import numpy as np
import matplotlib.pyplot as plt
from IPython import display
import cv2
import dlib
import os
import keras
import sklearn
from tensorflow.keras import utils, layers, models, optimizers
from tensorflow.keras.preprocessing import image
from sklearn.preprocessing import LabelBinarizer, MultiLabelBinarizer
from keras.utils import load_img, img_to_array, array_to_img
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

1. Face detection

The first step in your pipeline is face detection. Obviously you need to locate the faces in a photograph before you can try to tell them apart. If you've used any camera in the last 10 years, you've probably seen face detection in action. Face detection is a great feature for cameras. When the camera can automatically pick out faces, it can make sure that all the faces are in focus before it takes the picture. But you'll use it for a different purpose: finding the areas of the image you want to pass on to the next step in your pipeline.

Assignment

Here's what you are required to do for this part of the assignment.

- You are provided with a small dataset of pictures, where each picture contains exactly one face. Extract the faces and their labels (i.e., the person's names). Store them to a new file with the function `dump` in the package `pickle`.
- Normalize the cropped faces (i.e., divide the pixel values by 255), and split them in train set (70%) and test set (30%) with the function `train_test_split` in the package `sklearn`.
- Train a small convnet and check its performance on the test set. Remember: don't use the test images for training.
- Try to improve the performance of the baseline convnet by using all the tricks you have learned in the course.

Provided functions

Here you will find some useful functions to complete the assignment.

```
!wget https://perso.esiee.fr/~najmanl/FaceRecognition/models.zip
!unzip models.zip
```

```
--2024-12-12 16:19:37--
https://perso.esiee.fr/~najmanl/FaceRecognition/models.zip
Resolving perso.esiee.fr (perso.esiee.fr)... 147.215.150.8
Connecting to perso.esiee.fr (perso.esiee.fr)|147.215.150.8|:443...
connected.
HTTP request sent, awaiting response... 200 OK
Length: 100563300 (96M) [application/zip]
Saving to: ‘models.zip’

models.zip          100%[===================>]  95.90M  23.7MB/s    in
5.1s

2024-12-12 16:19:43 (18.8 MB/s) - ‘models.zip’ saved
[100563300/100563300]

Archive:  models.zip
   creating: models/
  inflating: models/shape_predictor_68_face_landmarks.dat
   creating: __MACOSX/
   creating: __MACOSX/models/
  inflating: __MACOSX/models/._shape_predictor_68_face_landmarks.dat
  inflating: models/mmod_human_face_detector.dat
  inflating: __MACOSX/models/._mmod_human_face_detector.dat
  inflating: models/dlib_face_recognition_resnet_model_v1.dat
  inflating:
__MACOSX/models/._dlib_face_recognition_resnet_model_v1.dat
  inflating: models/shape_predictor_5_face_landmarks.dat
  inflating: __MACOSX/models/._shape_predictor_5_face_landmarks.dat
  inflating: __MACOSX/._models
```

```python
hog_detector = dlib.get_frontal_face_detector()
cnn_detector =
dlib.cnn_face_detection_model_v1('models/mmod_human_face_detector.dat'
)

def face_locations(image, model="hog"):

    if model == "hog":
        detector = hog_detector
        cst = 0
    elif model == "cnn":
        detector = cnn_detector
        cst = 10

    matches = detector(image,1)
    rects   = []

    for r in matches:
        if model == "cnn":
            r = r.rect
```

```
        x = max(r.left(), 0)
        y = max(r.top(), 0)
        w = min(r.right(), image.shape[1]) - x + cst
        h = min(r.bottom(), image.shape[0]) - y + cst
        rects.append((x,y,w,h))

    return rects

def extract_faces(image, model="hog"):

    gray  = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
    rects = face_locations(gray, model)
    faces = []

    for (x,y,w,h) in rects:
        cropped = image[y:y+h, x:x+w, :]
        cropped = cv2.resize(cropped, (128,128))
        faces.append(cropped)

    return faces

 def show_grid(faces, figsize=(12,3)):

    n = len(faces)
    cols = 7
    rows = int(np.ceil(n/cols))

    fig, ax = plt.subplots(rows,cols, figsize=figsize)

    for r in range(rows):
        for c in range(cols):
            i = r*cols + c
            if i == n:
                 break
            ax[r,c].imshow(faces[i])
            ax[r,c].axis('off')
            #ax[r,c].set_title('size: ' + str(faces[i].shape[:2]))

def list_images(basePath, validExts=(".jpg", ".jpeg", ".png", ".bmp",
".tif", ".tiff"), contains=None):

    imagePaths = []

    # loop over the directory structure
    for (rootDir, dirNames, filenames) in os.walk(basePath):
        # loop over the filenames in the current directory
        for filename in filenames:
            # if the contains string is not none and the filename does
not contain
            # the supplied string, then ignore the file
            if contains is not None and filename.find(contains) == -1:
```

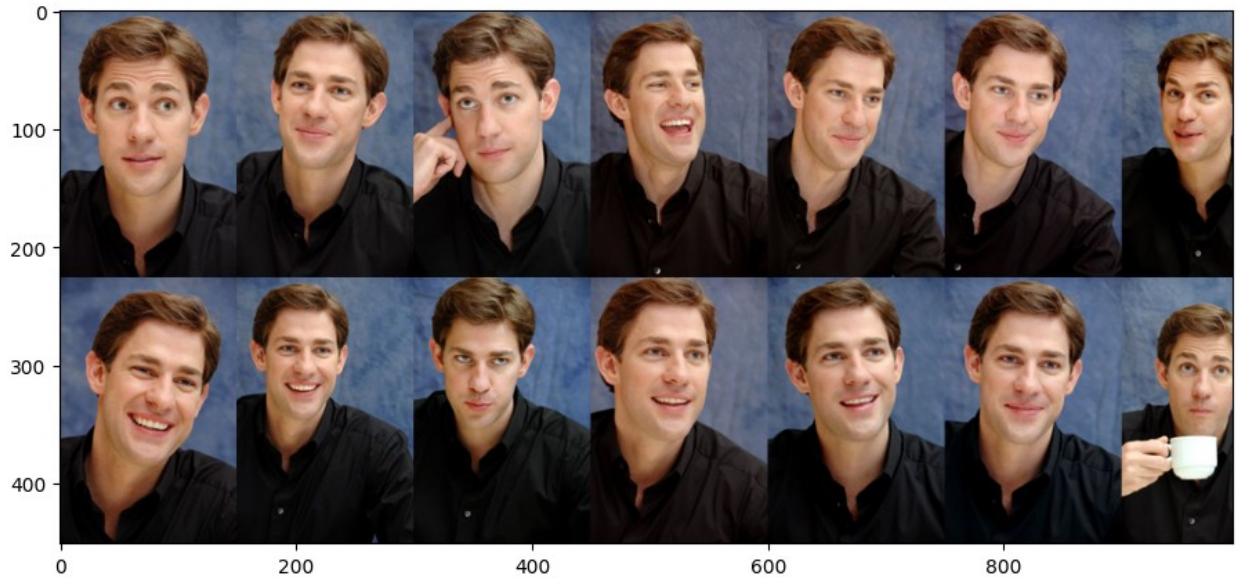
```
                continue

            # determine the file extension of the current file
            ext = filename[filename.rfind("."):].lower()

            # check to see if the file is an image and should be
processed
            if ext.endswith(validExts):
                # construct the path to the image and yield it
                imagePath = os.path.join(rootDir, filename).replace("
", "\\ ")
                imagePaths.append(imagePath)

    return imagePaths
```

Hints

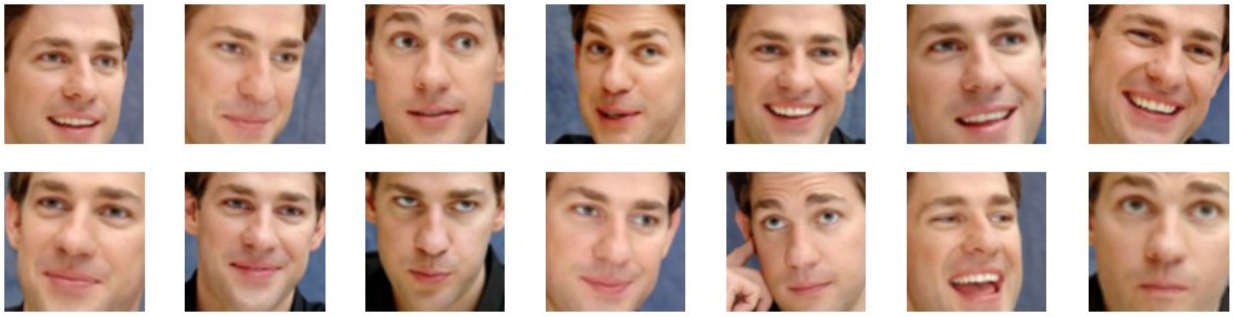
The provided function `extract_faces()` applies face detection to a single input image, and returns a list of 128x128 blocks containing the detected faces.

```
#!wget https://perso.esiee.fr/~najmanl/FaceRecognition/figures.zip
#!unzip figures.zip

image = cv2.imread("figures/faces.png")
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

plt.figure(figsize=(15,5))
plt.imshow(image)
```

```
<matplotlib.image.AxesImage at 0x7f83f3105f60>
```



```
faces = extract_faces(image, "hog")  # Replace 'cnn' with 'hog' for
faster but less accurate results

show_grid(faces)
```



Moreover, the function `list_images()` locates all the jpeg/png/tiff files in a given folder (including its subfolders).

```
#!wget https://perso.esiee.fr/~najmanl/FaceRecognition/data.zip
#n!unzip data.zip

--2024-12-12 16:34:26--
https://perso.esiee.fr/~najmanl/FaceRecognition/data.zip
Resolving perso.esiee.fr (perso.esiee.fr)... 147.215.150.8
Connecting to perso.esiee.fr (perso.esiee.fr)|147.215.150.8|:443...
connected.
HTTP request sent, awaiting response... 200 OK
Length: 91829333 (88M) [application/zip]
Saving to: ‘data.zip.1’

data.zip.1          100%[===================>]  87.58M  24.6MB/s    in
4.6s

2024-12-12 16:34:32 (18.9 MB/s) - ‘data.zip.1’ saved
[91829333/91829333]

Archive:  data.zip
replace data/owen_grady/00000010.jpg? [y]es, [n]o, [A]ll, [N]one,
[r]ename: no
replace __MACOSX/data/owen_grady/._00000010.jpg? [y]es, [n]o, [A]ll,
[N]one, [r]ename: n
replace data/owen_grady/00000004.jpg? [y]es, [n]o, [A]ll, [N]one,
[r]ename: n
replace __MACOSX/data/owen_grady/._00000004.jpg? [y]es, [n]o, [A]ll,
[N]one, [r]ename: n
replace data/owen_grady/00000005.jpg? [y]es, [n]o, [A]ll, [N]one,
[r]ename: n
replace __MACOSX/data/owen_grady/._00000005.jpg? [y]es, [n]o, [A]ll,
[N]one, [r]ename: n
```

```
replace data/owen_grady/00000007.jpg? [y]es, [n]o, [A]ll, [N]one,
[r]ename: n
replace __MACOSX/data/owen_grady/._00000007.jpg? [y]es, [n]o, [A]ll,
[N]one, [r]ename: n
replace data/owen_grady/00000013.jpg? [y]es, [n]o, [A]ll, [N]one,
[r]ename: n
replace __MACOSX/data/owen_grady/._00000013.jpg? [y]es, [n]o, [A]ll,
[N]one, [r]ename: n
replace data/owen_grady/00000012.jpg? [y]es, [n]o, [A]ll, [N]one,
[r]ename: none
replace __MACOSX/data/owen_grady/._00000012.jpg? [y]es, [n]o, [A]ll,
[N]one, [r]ename: n
replace data/owen_grady/00000006.jpg? [y]es, [n]o, [A]ll, [N]one,
[r]ename: n
replace __MACOSX/data/owen_grady/._00000006.jpg? [y]es, [n]o, [A]ll,
[N]one, [r]ename: n
replace data/owen_grady/00000002.jpg? [y]es, [n]o, [A]ll, [N]one,
[r]ename: n
replace __MACOSX/data/owen_grady/._00000002.jpg? [y]es, [n]o, [A]ll,
[N]one, [r]ename:
```

```
imagePaths = list_images("data")
imagePaths
```

```
['data/owen_grady/00000047.jpg',
 'data/owen_grady/00000027.jpeg',
 'data/owen_grady/00000013.jpg',
 'data/owen_grady/00000068.jpg',
 'data/owen_grady/00000015.jpg',
 'data/owen_grady/00000000.jpg',
 'data/owen_grady/00000024.jpg',
 'data/owen_grady/00000016.jpg',
 'data/owen_grady/00000017.jpg',
 'data/owen_grady/00000003.jpg',
 'data/owen_grady/00000029.jpg',
 'data/owen_grady/00000007.jpg',
 'data/owen_grady/00000021.jpg',
 'data/owen_grady/00000083.jpg',
 'data/owen_grady/00000012.jpg',
 'data/owen_grady/00000019.jpg',
 'data/owen_grady/00000001.jpg',
 'data/owen_grady/00000080.jpg',
 'data/owen_grady/00000022.jpg',
 'data/owen_grady/00000070.jpg',
 'data/owen_grady/00000004.jpg',
 'data/owen_grady/00000026.jpg',
 'data/owen_grady/00000006.jpg',
 'data/owen_grady/00000035.jpg',
 'data/owen_grady/00000020.jpg',
 'data/owen_grady/00000030.jpg',
```

```
 'data/owen_grady/00000041.jpg',
 'data/owen_grady/00000010.jpg',
 'data/owen_grady/00000018.jpg',
 'data/owen_grady/00000034.jpg',
 'data/owen_grady/00000005.jpg',
 'data/owen_grady/00000059.jpg',
 'data/owen_grady/00000014.jpg',
 'data/owen_grady/00000002.jpg',
 'data/owen_grady/00000086.jpg',
 'data/john_hammond/00000072.jpg',
 'data/john_hammond/00000013.jpg',
 'data/john_hammond/00000009.jpg',
 'data/john_hammond/00000065.jpg',
 'data/john_hammond/00000038.jpg',
 'data/john_hammond/00000085.jpg',
 'data/john_hammond/00000068.jpg',
 'data/john_hammond/00000037.png',
 'data/john_hammond/00000015.jpg',
 'data/john_hammond/00000073.jpg',
 'data/john_hammond/00000000.jpg',
 'data/john_hammond/00000016.jpg',
 'data/john_hammond/00000003.jpg',
 'data/john_hammond/00000029.jpg',
 'data/john_hammond/00000007.jpg',
 'data/john_hammond/00000021.jpg',
 'data/john_hammond/00000012.jpg',
 'data/john_hammond/00000036.jpg',
 'data/john_hammond/00000001.jpg',
 'data/john_hammond/00000075.png',
 'data/john_hammond/00000022.jpg',
 'data/john_hammond/00000005.png',
 'data/john_hammond/00000004.jpg',
 'data/john_hammond/00000042.jpg',
 'data/john_hammond/00000006.jpg',
 'data/john_hammond/00000032.jpg',
 'data/john_hammond/00000066.jpg',
 'data/john_hammond/00000088.jpg',
 'data/john_hammond/00000008.jpg',
 'data/john_hammond/00000089.jpg',
 'data/john_hammond/00000010.jpg',
 'data/john_hammond/00000090.jpg',
 'data/john_hammond/00000014.jpg',
 'data/john_hammond/00000031.png',
 'data/john_hammond/00000025.jpg',
 'data/john_hammond/00000002.jpg',
 'data/alan_grant/00000063.jpg',
 'data/alan_grant/00000009.png',
 'data/alan_grant/00000023.jpg',
 'data/alan_grant/00000000.jpg',
```

```
 'data/alan_grant/00000024.jpg',
 'data/alan_grant/00000017.jpg',
 'data/alan_grant/00000021.jpg',
 'data/alan_grant/00000028.png',
 'data/alan_grant/00000004.png',
 'data/alan_grant/00000022.jpg',
 'data/alan_grant/00000041.JPG',
 'data/alan_grant/00000006.jpg',
 'data/alan_grant/00000082.jpg',
 'data/alan_grant/00000089.jpeg',
 'data/alan_grant/00000065.png',
 'data/alan_grant/00000027.png',
 'data/alan_grant/00000046.jpg',
 'data/alan_grant/00000031.jpg',
 'data/alan_grant/00000005.jpg',
 'data/alan_grant/00000025.png',
 'data/alan_grant/00000002.jpg',
 'data/alan_grant/00000015.JPG',
 'data/ellie_sattler/00000009.jpg',
 'data/ellie_sattler/00000065.jpg',
 'data/ellie_sattler/00000038.jpg',
 'data/ellie_sattler/00000087.jpg',
 'data/ellie_sattler/00000015.jpg',
 'data/ellie_sattler/00000000.jpg',
 'data/ellie_sattler/00000016.jpg',
 'data/ellie_sattler/00000017.jpg',
 'data/ellie_sattler/00000003.jpeg',
 'data/ellie_sattler/00000007.jpg',
 'data/ellie_sattler/00000083.jpg',
 'data/ellie_sattler/00000012.jpg',
 'data/ellie_sattler/00000001.jpg',
 'data/ellie_sattler/00000022.jpg',
 'data/ellie_sattler/00000054.jpg',
 'data/ellie_sattler/00000070.jpg',
 'data/ellie_sattler/00000064.jpg',
 'data/ellie_sattler/00000004.jpg',
 'data/ellie_sattler/00000042.jpg',
 'data/ellie_sattler/00000020.jpg',
 'data/ellie_sattler/00000032.jpg',
 'data/ellie_sattler/00000030.jpg',
 'data/ellie_sattler/00000008.jpg',
 'data/ellie_sattler/00000037.jpg',
 'data/ellie_sattler/00000028.jpg',
 'data/ellie_sattler/00000069.jpg',
 'data/ellie_sattler/00000005.jpg',
 'data/ellie_sattler/00000059.jpg',
 'data/ellie_sattler/00000014.jpg',
 'data/ellie_sattler/00000025.jpg',
 'data/ellie_sattler/00000002.jpg',
```

```
 'data/claire_dearing/00000054.png',
 'data/claire_dearing/00000063.jpg',
 'data/claire_dearing/00000020.png',
 'data/claire_dearing/00000062.jpg',
 'data/claire_dearing/00000009.jpg',
 'data/claire_dearing/00000033.png',
 'data/claire_dearing/00000015.png',
 'data/claire_dearing/00000068.jpg',
 'data/claire_dearing/00000023.jpg',
 'data/claire_dearing/00000037.png',
 'data/claire_dearing/00000073.jpg',
 'data/claire_dearing/00000024.jpg',
 'data/claire_dearing/00000055.png',
 'data/claire_dearing/00000011.jpg',
 'data/claire_dearing/00000071.jpg',
 'data/claire_dearing/00000003.jpg',
 'data/claire_dearing/00000078.jpg',
 'data/claire_dearing/00000029.jpg',
 'data/claire_dearing/00000028.png',
 'data/claire_dearing/00000083.jpg',
 'data/claire_dearing/00000012.jpg',
 'data/claire_dearing/00000036.jpg',
 'data/claire_dearing/00000000.png',
 'data/claire_dearing/00000072.png',
 'data/claire_dearing/00000004.png',
 'data/claire_dearing/00000001.jpg',
 'data/claire_dearing/00000008.png',
 'data/claire_dearing/00000052.jpg',
 'data/claire_dearing/00000075.png',
 'data/claire_dearing/00000022.jpg',
 'data/claire_dearing/00000043.png',
 'data/claire_dearing/00000005.png',
 'data/claire_dearing/00000016.png',
 'data/claire_dearing/00000026.jpg',
 'data/claire_dearing/00000006.jpg',
 'data/claire_dearing/00000081.png',
 'data/claire_dearing/00000035.jpg',
 'data/claire_dearing/00000038.png',
 'data/claire_dearing/00000074.jpg',
 'data/claire_dearing/00000060.png',
 'data/claire_dearing/00000019.png',
 'data/claire_dearing/00000010.jpg',
 'data/claire_dearing/00000079.jpg',
 'data/claire_dearing/00000031.jpg',
 'data/claire_dearing/00000018.jpg',
 'data/claire_dearing/00000034.jpg',
 'data/claire_dearing/00000076.jpg',
 'data/claire_dearing/00000021.png',
 'data/claire_dearing/00000041.png',
```

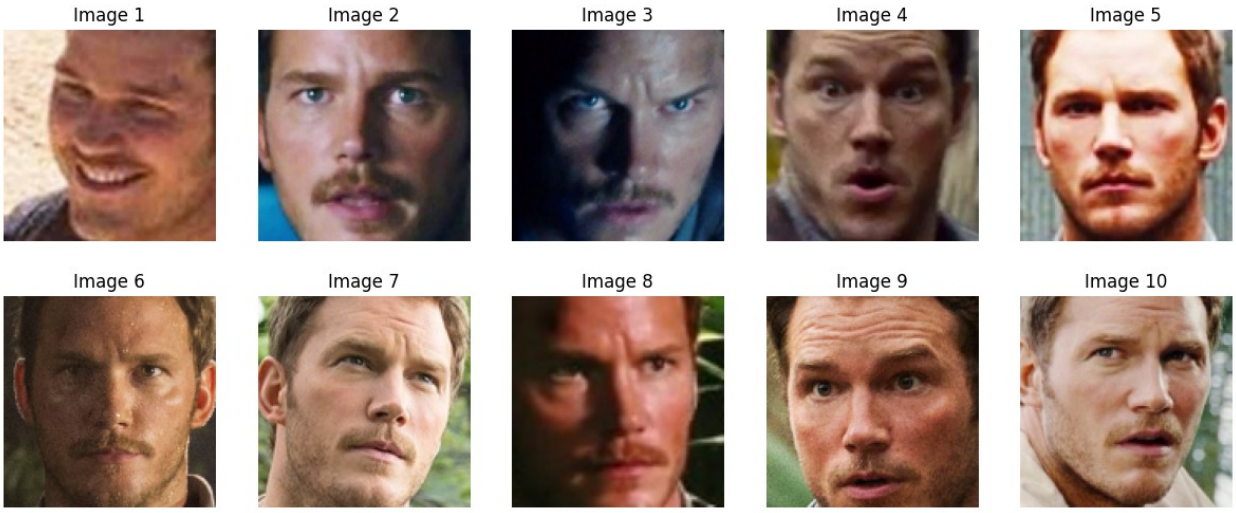
```
 'data/claire_dearing/00000014.jpg',
 'data/claire_dearing/00000025.jpg',
 'data/claire_dearing/00000017.png',
 'data/claire_dearing/00000002.jpg',
 'data/ian_malcolm/00000072.jpg',
 'data/ian_malcolm/00000062.jpg',
 'data/ian_malcolm/00000013.jpg',
 'data/ian_malcolm/00000009.jpg',
 'data/ian_malcolm/00000023.jpg',
 'data/ian_malcolm/00000015.jpg',
 'data/ian_malcolm/00000000.jpg',
 'data/ian_malcolm/00000024.jpg',
 'data/ian_malcolm/00000027.jpg',
 'data/ian_malcolm/00000016.jpg',
 'data/ian_malcolm/00000011.jpg',
 'data/ian_malcolm/00000003.jpg',
 'data/ian_malcolm/00000040.jpg',
 'data/ian_malcolm/00000007.jpg',
 'data/ian_malcolm/00000021.jpg',
 'data/ian_malcolm/00000083.jpg',
 'data/ian_malcolm/00000036.jpg',
 'data/ian_malcolm/00000001.jpg',
 'data/ian_malcolm/00000080.jpg',
 'data/ian_malcolm/00000022.jpg',
 'data/ian_malcolm/00000050.jpg',
 'data/ian_malcolm/00000069.png',
 'data/ian_malcolm/00000004.jpg',
 'data/ian_malcolm/00000051.jpg',
 'data/ian_malcolm/00000035.jpg',
 'data/ian_malcolm/00000020.jpg',
 'data/ian_malcolm/00000074.jpg',
 'data/ian_malcolm/00000082.jpg',
 'data/ian_malcolm/00000092.jpg',
 'data/ian_malcolm/00000030.jpg',
 'data/ian_malcolm/00000088.png',
 'data/ian_malcolm/00000008.jpg',
 'data/ian_malcolm/00000019.png',
 'data/ian_malcolm/00000010.jpg',
 'data/ian_malcolm/00000031.jpg',
 'data/ian_malcolm/00000018.jpg',
 'data/ian_malcolm/00000005.jpg',
 'data/ian_malcolm/00000014.jpg',
 'data/ian_malcolm/00000048.jpg',
 'data/ian_malcolm/00000012.png',
 'data/ian_malcolm/00000086.jpg']

list_names = []

for imagePath in imagePaths :
    separate = imagePath.split('/')
```

```
    name_images = separate[1]
    list_names.append(name_images)

print(list_names)
print(len(list_names))
```

```
['owen_grady', 'owen_grady', 'owen_grady', 'owen_grady', 'owen_grady',
'owen_grady', 'owen_grady', 'owen_grady', 'owen_grady', 'owen_grady',
'owen_grady', 'owen_grady', 'owen_grady', 'owen_grady', 'owen_grady',
'owen_grady', 'owen_grady', 'owen_grady', 'owen_grady', 'owen_grady',
'owen_grady', 'owen_grady', 'owen_grady', 'owen_grady', 'owen_grady',
'owen_grady', 'owen_grady', 'owen_grady', 'owen_grady', 'owen_grady',
'owen_grady', 'owen_grady', 'owen_grady', 'owen_grady', 'owen_grady',
'john_hammond', 'john_hammond', 'john_hammond', 'john_hammond',
'john_hammond', 'john_hammond', 'john_hammond', 'john_hammond',
'john_hammond', 'john_hammond', 'john_hammond', 'john_hammond',
'john_hammond', 'john_hammond', 'john_hammond', 'john_hammond',
'john_hammond', 'john_hammond', 'john_hammond', 'john_hammond',
'john_hammond', 'john_hammond', 'john_hammond', 'john_hammond',
'john_hammond', 'john_hammond', 'john_hammond', 'john_hammond',
'john_hammond', 'john_hammond', 'john_hammond', 'john_hammond',
'alan_grant', 'alan_grant', 'alan_grant', 'alan_grant', 'alan_grant',
'alan_grant', 'alan_grant', 'alan_grant', 'alan_grant', 'alan_grant',
'alan_grant', 'alan_grant', 'alan_grant', 'alan_grant', 'alan_grant',
'alan_grant', 'alan_grant', 'alan_grant', 'alan_grant', 'alan_grant',
'alan_grant', 'alan_grant', 'ellie_sattler', 'ellie_sattler',
'ellie_sattler', 'ellie_sattler', 'ellie_sattler', 'ellie_sattler',
'ellie_sattler', 'ellie_sattler', 'ellie_sattler', 'ellie_sattler',
'ellie_sattler', 'ellie_sattler', 'ellie_sattler', 'ellie_sattler',
'ellie_sattler', 'ellie_sattler', 'ellie_sattler', 'ellie_sattler',
'ellie_sattler', 'ellie_sattler', 'ellie_sattler', 'ellie_sattler',
'ellie_sattler', 'ellie_sattler', 'ellie_sattler', 'ellie_sattler',
'ellie_sattler', 'claire_dearing', 'claire_dearing', 'claire_dearing',
'claire_dearing', 'claire_dearing', 'claire_dearing',
'claire_dearing', 'claire_dearing', 'claire_dearing',
'claire_dearing', 'claire_dearing', 'claire_dearing',
'claire_dearing', 'claire_dearing', 'claire_dearing',
'claire_dearing', 'claire_dearing', 'claire_dearing',
'claire_dearing', 'claire_dearing', 'claire_dearing',
'claire_dearing', 'claire_dearing', 'claire_dearing',
'claire_dearing', 'claire_dearing', 'claire_dearing',
'claire_dearing', 'claire_dearing', 'claire_dearing',
'claire_dearing', 'claire_dearing', 'claire_dearing',
'claire_dearing', 'claire_dearing', 'claire_dearing',
'claire_dearing', 'claire_dearing', 'claire_dearing',
'claire_dearing', 'claire_dearing', 'claire_dearing',
'claire_dearing', 'claire_dearing', 'claire_dearing',
'claire_dearing', 'claire_dearing', 'claire_dearing',
```

```
'claire_dearing', 'claire_dearing', 'claire_dearing',
'claire_dearing', 'claire_dearing', 'ian_malcolm', 'ian_malcolm',
'ian_malcolm', 'ian_malcolm', 'ian_malcolm', 'ian_malcolm',
'ian_malcolm', 'ian_malcolm', 'ian_malcolm', 'ian_malcolm',
'ian_malcolm', 'ian_malcolm', 'ian_malcolm', 'ian_malcolm',
'ian_malcolm', 'ian_malcolm', 'ian_malcolm', 'ian_malcolm',
'ian_malcolm', 'ian_malcolm', 'ian_malcolm', 'ian_malcolm',
'ian_malcolm', 'ian_malcolm', 'ian_malcolm', 'ian_malcolm',
'ian_malcolm', 'ian_malcolm', 'ian_malcolm', 'ian_malcolm',
'ian_malcolm', 'ian_malcolm', 'ian_malcolm', 'ian_malcolm',
'ian_malcolm', 'ian_malcolm', 'ian_malcolm', 'ian_malcolm',
'ian_malcolm', 'ian_malcolm', 'ian_malcolm']
218

# Initialisation des listes pour stocker les visages extraits et les
noms des images
cropped_image = []
list_labels = []

# Parcours des chemins d'accès des images
for i in imagePaths:
    # Lecture de l'image à l'aide de OpenCV
    image = cv2.imread(i)

    # Conversion de l'espace de couleur de BGR à RGB
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    # Appel de la fonction pour extraire les visages de l'image
    image = extract_faces(image, "cnn")

    # Vérification si la fonction extract_faces() a renvoyé au moins
un visage
    if image:
        # Ajout du premier visage extrait à la liste cropped_image
        cropped_image.append(image[0])

        # Extraction du nom de l'image à partir du chemin d'accès
        filepath = i
        name = filepath.split('/')[1]

        # Ajout du nom de l'image à la liste list_labels
        list_labels.append(name)

fig, axes = plt.subplots(2, 5, figsize=(15, 6))
for i in range(5):  # Afficher les 5 premières images
    axes[0, i].imshow(cropped_image[i])
    axes[0, i].set_title("Image " + str(i+1))
    axes[0, i].axis('off')

for i in range(5, 10):  # Afficher les 5 images suivantes
```

```
    axes[1, i-5].imshow(cropped_image[i])
    axes[1, i-5].set_title("Image " + str(i+1))
    axes[1, i-5].axis('off')

plt.show()
```



- Normalize the cropped faces (i.e., divide the pixel values by 255), and split them in train set (70%) and test set (30%) with the function train_test_split in the package sklearn.

```
# Initialisation d'une liste pour stocker les images normalisées
normalized_images = []

# Parcours des images extraites
for image in cropped_image:

    # Normalisation de l'image en divisant chaque pixel par 255.0
    normalized_image = image / 255.0

    # Ajout de l'image normalisée à la liste normalized_images
    normalized_images.append(normalized_image)

from sklearn.model_selection import train_test_split
import pickle

# Séparation des données en ensembles d'entraînement et de test
X_train, X_validation, Y_train, Y_validation =
train_test_split(normalized_images, list_labels, test_size=0.3,
random_state=42)


# Sauvegarde de l'ensemble d'entraînement dans un fichier pickle
'train_data.pkl'
with open('train_data.pkl', 'wb') as f:
    # Utilisation de pickle.dump() pour sauvegarder les données dans
```

```
le fichier
    # Les données sont sauvegardées sous forme de tuple (X_train,
Y_train)
    pickle.dump((X_train, Y_train), f)

# Sauvegarde de l'ensemble de test dans un fichier pickle
'validation_data.pkl'
with open('validation_data.pkl', 'wb') as f:
    # Utilisation de pickle.dump() pour sauvegarder les données dans
le fichier
    # Les données sont sauvegardées sous forme de tuple (X_test,
Y_test)
    pickle.dump((X_validation, Y_validation), f)

from sklearn.preprocessing import LabelBinarizer  # Importation de la
classe pour l'encodage des étiquettes

# Création d'une instance de LabelBinarizer pour l'encodage
encoder = LabelBinarizer()

# Encodage des étiquettes d'entraînement en format binaire
# La méthode fit_transform ajuste le transformateur sur Y_train et
encode simultanément les étiquettes
Y_train_encoded = encoder.fit_transform(Y_train)

# Encodage des étiquettes de test en format binaire
# La méthode transform utilise les informations d'encodage apprises
sur Y_train pour encoder Y_test
Y_validation_encoded = encoder.transform(Y_validation)

print(len(normalized_images))
print(len(X_train))
print(len(X_validation))
```

```
208
145
63
```

Train a small convnet and check its performance on the test set. Remember: don't use the test images for training.

```
datagen = ImageDataGenerator(
      rotation_range=40,
      width_shift_range=0.2,
      height_shift_range=0.2,
      shear_range=0.2,
      zoom_range=0.2,
      horizontal_flip=True,
      fill_mode='nearest')
```

```
train_generator = datagen.flow(np.array(X_train), Y_train_encoded,
batch_size=32)

# Define the network
model = models.Sequential()

model.add(layers.Conv2D(32, (3, 3), activation='relu',
input_shape=(128, 128, 3)))
model.add(layers.MaxPooling2D(2,2))
model.add(layers.Conv2D(64, (3,3), activation = 'relu'))
model.add(layers.MaxPooling2D(2,2))
model.add(layers.Conv2D(128, (3,3), activation = 'relu'))
model.add(layers.MaxPooling2D(2,2))
model.add(layers.Conv2D(128, (3,3), activation = 'relu'))
#model.add(layers.Dropout(0.5))
model.add(layers.Flatten())
model.add(layers.Dense(512, activation = 'relu'))
model.add(layers.Dense(6, activation = 'softmax'))
model.summary()

model.compile(loss='categorical_crossentropy',
optimizer=optimizers.Adam(learning_rate=1e-4), metrics=['acc'])

# Train the network. Hint: use .fit(...)
history = model.fit(train_generator, epochs=50, validation_data =
(np.array(X_validation), Y_validation_encoded))

# Évaluation du modèle sur les données de validation et obtention de
la perte (loss) et de la précision (accuracy)
test_loss, test_acc = model.evaluate(np.array(X_validation),
Y_validation_encoded)

print('Validation accuracy: {:2.2f}%'.format(test_acc*100))
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/
convolutional/base_conv.py:107: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)
```

```
Model: "sequential"
```

```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━
┓
┃ Layer (type)                         ┃ Output Shape                ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━
┩
│ conv2d (Conv2D)                      │ (None, 126, 126, 32)        │
```

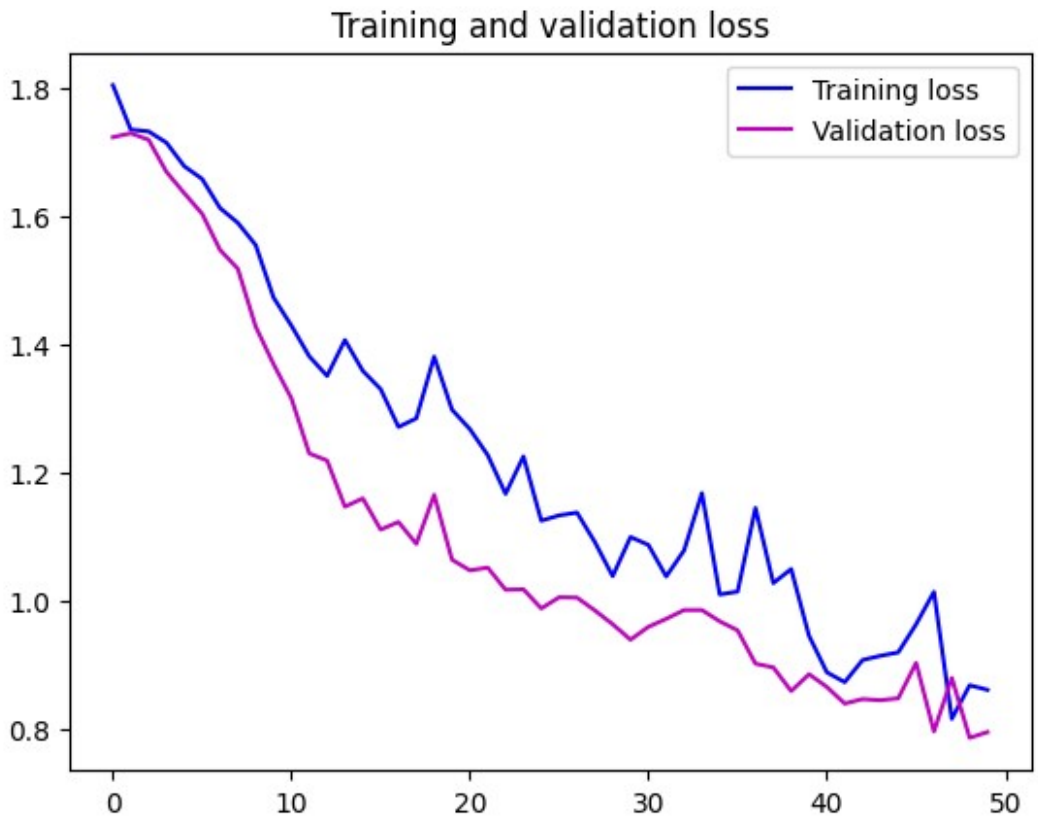
```
896 |
┣━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┻━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┻
━━━━━━━━━━━━━━━━━━━┫
┃ max_pooling2d (MaxPooling2D)          ┃ (None, 63, 63, 32)          ┃
0 ┃
┣━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╋━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╋
━━━━━━━━━━━━━━━━━━━┫
┃ conv2d_1 (Conv2D)                     ┃ (None, 61, 61, 64)          ┃
18,496 ┃
┣━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╋━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╋
━━━━━━━━━━━━━━━━━━━┫
┃ max_pooling2d_1 (MaxPooling2D)        ┃ (None, 30, 30, 64)          ┃
0 ┃
┣━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╋━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╋
━━━━━━━━━━━━━━━━━━━┫
┃ conv2d_2 (Conv2D)                     ┃ (None, 28, 28, 128)         ┃
73,856 ┃
┣━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╋━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╋
━━━━━━━━━━━━━━━━━━━┫
┃ max_pooling2d_2 (MaxPooling2D)        ┃ (None, 14, 14, 128)         ┃
0 ┃
┣━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╋━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╋
━━━━━━━━━━━━━━━━━━━┫
┃ conv2d_3 (Conv2D)                     ┃ (None, 12, 12, 128)         ┃
147,584 ┃
┣━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╋━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╋
━━━━━━━━━━━━━━━━━━━┫
┃ flatten (Flatten)                     ┃ (None, 18432)               ┃
0 ┃
┣━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╋━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╋
━━━━━━━━━━━━━━━━━━━┫
┃ dense (Dense)                         ┃ (None, 512)                 ┃
9,437,696 ┃
┣━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╋━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╋
━━━━━━━━━━━━━━━━━━━┫
┃ dense_1 (Dense)                       ┃ (None, 6)                   ┃
3,078 ┃
┗━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┻━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┻
━━━━━━━━━━━━━━━━━━━┛

 Total params: 9,681,606 (36.93 MB)

 Trainable params: 9,681,606 (36.93 MB)

 Non-trainable params: 0 (0.00 B)

Epoch 1/50

/usr/local/lib/python3.10/dist-packages/keras/src/trainers/
data_adapters/py_dataset_adapter.py:122: UserWarning: Your `PyDataset`
```

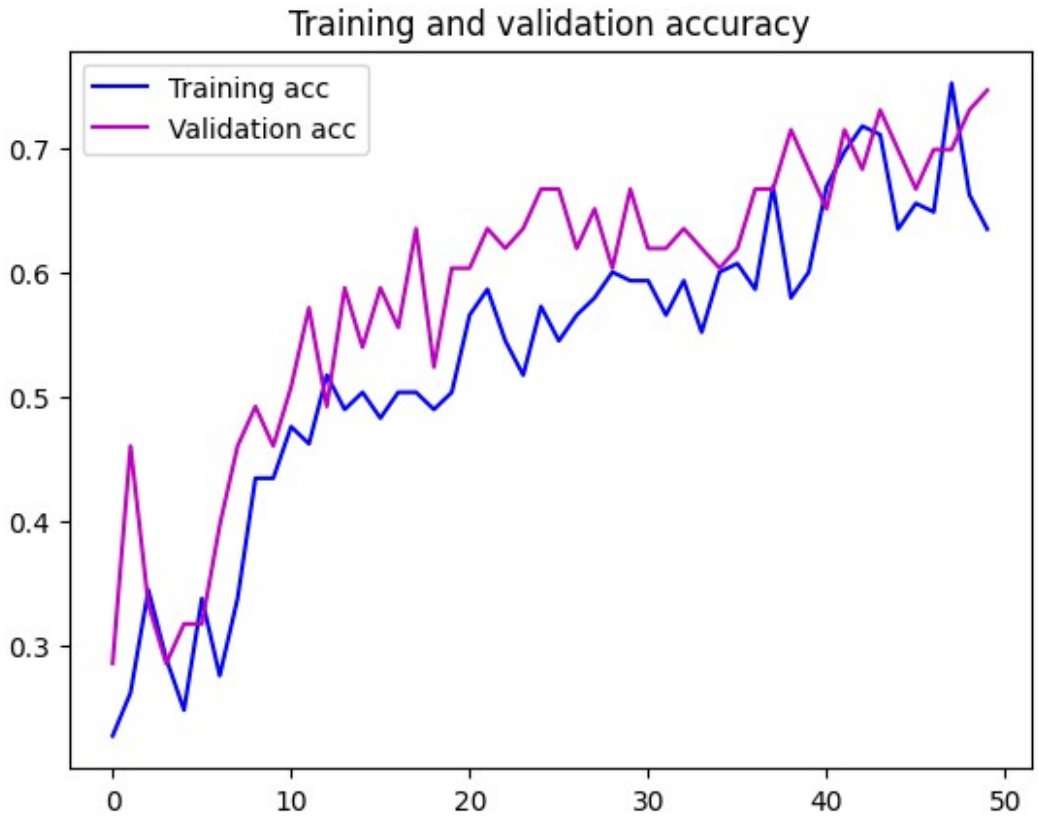
```
class should call `super().__init__(**kwargs)` in its constructor.
`**kwargs` can include `workers`, `use_multiprocessing`,
`max_queue_size`. Do not pass these arguments to `fit()`, as they will
be ignored.
  self._warn_if_super_not_called()

5/5 ━━━━━━━━━━━━━━━━━━━━ 10s 1s/step - acc: 0.2509 - loss: 1.7689 -
val_acc: 0.2857 - val_loss: 1.7237
Epoch 2/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 43ms/step - acc: 0.2807 - loss: 1.7191 -
val_acc: 0.4603 - val_loss: 1.7299
Epoch 3/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - acc: 0.3420 - loss: 1.7434 -
val_acc: 0.3333 - val_loss: 1.7198
Epoch 4/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - acc: 0.3146 - loss: 1.7135 -
val_acc: 0.2857 - val_loss: 1.6696
Epoch 5/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - acc: 0.2278 - loss: 1.6901 -
val_acc: 0.3175 - val_loss: 1.6363
Epoch 6/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - acc: 0.3009 - loss: 1.6645 -
val_acc: 0.3175 - val_loss: 1.6041
Epoch 7/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - acc: 0.2881 - loss: 1.6118 -
val_acc: 0.3968 - val_loss: 1.5479
Epoch 8/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - acc: 0.3624 - loss: 1.5612 -
val_acc: 0.4603 - val_loss: 1.5182
Epoch 9/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - acc: 0.4263 - loss: 1.5765 -
val_acc: 0.4921 - val_loss: 1.4279
Epoch 10/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 37ms/step - acc: 0.3921 - loss: 1.5144 -
val_acc: 0.4603 - val_loss: 1.3693
Epoch 11/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - acc: 0.4542 - loss: 1.4381 -
val_acc: 0.5079 - val_loss: 1.3160
Epoch 12/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - acc: 0.4428 - loss: 1.4165 -
val_acc: 0.5714 - val_loss: 1.2299
Epoch 13/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - acc: 0.5276 - loss: 1.3281 -
val_acc: 0.4921 - val_loss: 1.2188
Epoch 14/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - acc: 0.5323 - loss: 1.3430 -
val_acc: 0.5873 - val_loss: 1.1468
Epoch 15/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - acc: 0.5429 - loss: 1.3113 -
val_acc: 0.5397 - val_loss: 1.1596
```

```
Epoch 16/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - acc: 0.5357 - loss: 1.2558 -
val_acc: 0.5873 - val_loss: 1.1110
Epoch 17/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - acc: 0.5523 - loss: 1.2214 -
val_acc: 0.5556 - val_loss: 1.1228
Epoch 18/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - acc: 0.5422 - loss: 1.2367 -
val_acc: 0.6349 - val_loss: 1.0886
Epoch 19/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - acc: 0.4432 - loss: 1.4382 -
val_acc: 0.5238 - val_loss: 1.1652
Epoch 20/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - acc: 0.4987 - loss: 1.3038 -
val_acc: 0.6032 - val_loss: 1.0639
Epoch 21/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - acc: 0.5807 - loss: 1.2667 -
val_acc: 0.6032 - val_loss: 1.0472
Epoch 22/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 45ms/step - acc: 0.5934 - loss: 1.2391 -
val_acc: 0.6349 - val_loss: 1.0516
Epoch 23/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 47ms/step - acc: 0.5844 - loss: 1.1140 -
val_acc: 0.6190 - val_loss: 1.0169
Epoch 24/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - acc: 0.5497 - loss: 1.1896 -
val_acc: 0.6349 - val_loss: 1.0176
Epoch 25/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - acc: 0.5701 - loss: 1.1369 -
val_acc: 0.6667 - val_loss: 0.9878
Epoch 26/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - acc: 0.5310 - loss: 1.1766 -
val_acc: 0.6667 - val_loss: 1.0052
Epoch 27/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - acc: 0.5644 - loss: 1.1797 -
val_acc: 0.6190 - val_loss: 1.0047
Epoch 28/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - acc: 0.5781 - loss: 1.0777 -
val_acc: 0.6508 - val_loss: 0.9847
Epoch 29/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - acc: 0.6288 - loss: 0.9532 -
val_acc: 0.6032 - val_loss: 0.9632
Epoch 30/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - acc: 0.5697 - loss: 1.1022 -
val_acc: 0.6667 - val_loss: 0.9390
Epoch 31/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - acc: 0.6105 - loss: 1.0183 -
val_acc: 0.6190 - val_loss: 0.9591
Epoch 32/50
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - acc: 0.5861 - loss: 1.0426 -
val_acc: 0.6190 - val_loss: 0.9715
Epoch 33/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - acc: 0.5694 - loss: 1.1034 -
val_acc: 0.6349 - val_loss: 0.9853
Epoch 34/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - acc: 0.5808 - loss: 1.1198 -
val_acc: 0.6190 - val_loss: 0.9853
Epoch 35/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 46ms/step - acc: 0.6093 - loss: 0.9919 -
val_acc: 0.6032 - val_loss: 0.9676
Epoch 36/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 34ms/step - acc: 0.6020 - loss: 1.0465 -
val_acc: 0.6190 - val_loss: 0.9536
Epoch 37/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - acc: 0.5739 - loss: 1.1443 -
val_acc: 0.6667 - val_loss: 0.9015
Epoch 38/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - acc: 0.6917 - loss: 0.9853 -
val_acc: 0.6667 - val_loss: 0.8958
Epoch 39/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - acc: 0.6043 - loss: 0.9773 -
val_acc: 0.7143 - val_loss: 0.8587
Epoch 40/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - acc: 0.6062 - loss: 0.9833 -
val_acc: 0.6825 - val_loss: 0.8851
Epoch 41/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - acc: 0.6839 - loss: 0.8516 -
val_acc: 0.6508 - val_loss: 0.8652
Epoch 42/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - acc: 0.7103 - loss: 0.8787 -
val_acc: 0.7143 - val_loss: 0.8392
Epoch 43/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - acc: 0.7143 - loss: 0.8791 -
val_acc: 0.6825 - val_loss: 0.8462
Epoch 44/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - acc: 0.7193 - loss: 0.9020 -
val_acc: 0.7302 - val_loss: 0.8444
Epoch 45/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - acc: 0.6482 - loss: 0.9051 -
val_acc: 0.6984 - val_loss: 0.8475
Epoch 46/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - acc: 0.6915 - loss: 0.9313 -
val_acc: 0.6667 - val_loss: 0.9028
Epoch 47/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - acc: 0.6488 - loss: 1.0124 -
val_acc: 0.6984 - val_loss: 0.7957
Epoch 48/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 36ms/step - acc: 0.7576 - loss: 0.7872 -
```

```
val_acc: 0.6984 - val_loss: 0.8793
Epoch 49/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - acc: 0.6619 - loss: 0.8703 -
val_acc: 0.7302 - val_loss: 0.7857
Epoch 50/50
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 49ms/step - acc: 0.6124 - loss: 0.8965 -
val_acc: 0.7460 - val_loss: 0.7947
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - acc: 0.7578 - loss: 0.8065
Validation accuracy: 74.60%

# Get the training info
loss     = history.history['loss']
val_loss = history.history['val_loss']
acc      = history.history['acc']
val_acc  = history.history['val_acc']

# Visualize the history plots
plt.figure()
plt.plot(loss, 'b', label='Training loss')
plt.plot(val_loss, 'm', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
plt.figure()
plt.plot(acc, 'b', label='Training acc')
plt.plot(val_acc, 'm', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.show()
```


Training and validation loss
Training loss
Validation loss
1.8
1.6
1.4
1.2
1.0
0.8
0
10
20
30
40
50


Training and validation accuracy
Training acc
Validation acc
0.7
0.6
0.5
0.4
0.3
0
10
20
30
40
50

Try to improve the performance of the baseline convnet by using all the tricks you have learned in the course.

2. Pose estimation

You have isolated the faces in our image. But now you have to deal with the problem that faces turned different directions look totally different to a computer. To account for this, you will try to warp each picture so that the eyes and lips are always in the same place in the image. More concretely, you are going to use an algorithm called face landmark estimation. The basic idea is to locate 68 specific points (called landmarks) that exist on every face: the top of the chin, the outside edge of each eye, the inner edge of each eyebrow, etc. Then, you'll simply rotate, scale and shear the image so that the eyes and mouth are centered as best as possible. This will make face recognition more accurate.

Assignment

Here's what you are required to do for this part of the assignment.

- Further preprocess the face pictures by correcting their pose. You should now have a dataset of cropped, aligned, and normalized faces.
- Re-train your convnets on the modified dataset.
- Evaluate the performance on the test set, and compare it to the scores obtained with your previously trained convnets.

Provided functions

Here you will find some useful functions to complete the assignment.

```
pose68 =
dlib.shape_predictor('models/shape_predictor_68_face_landmarks.dat')
pose05 =
dlib.shape_predictor('models/shape_predictor_5_face_landmarks.dat')

def face_landmarks(face, model="large"):

    if model == "large":
        predictor = pose68
    elif model == "small":
        predictor = pose05

    if not isinstance(face, list):
        rect = dlib.rectangle(0,0,face.shape[1],face.shape[0])
        return predictor(face, rect)
    else:
        rect = dlib.rectangle(0,0,face[0].shape[1],face[0].shape[0])
        return [predictor(f,rect) for f in face]
```

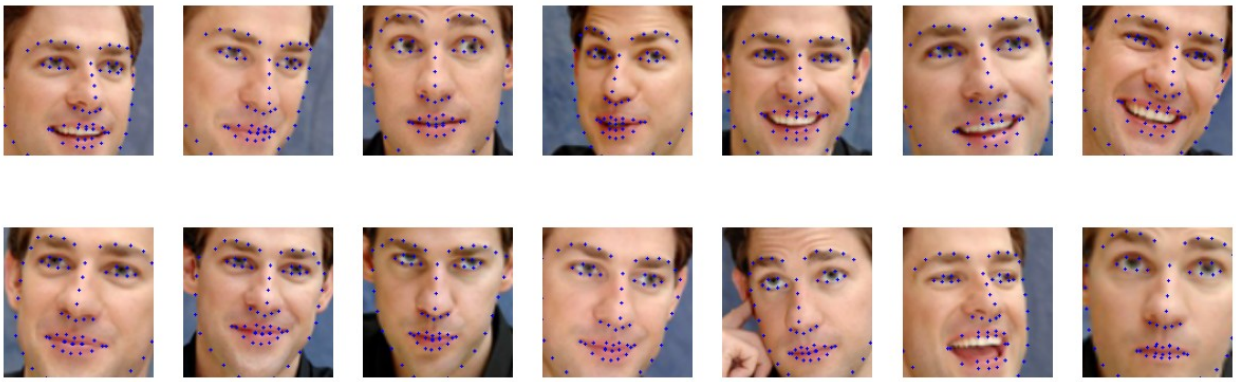
```
def shape_to_coords(shape):
    return np.float32([[p.x, p.y] for p in shape.parts()])

TEMPLATE = np.float32([
    (0.0792396913815, 0.339223741112), (0.0829219487236,
0.456955367943),
    (0.0967927109165, 0.575648016728), (0.122141515615,
0.691921601066),
    (0.168687863544, 0.800341263616), (0.239789390707,
0.895732504778),
    (0.325662452515, 0.977068762493), (0.422318282013, 1.04329000149),
    (0.531777802068, 1.06080371126), (0.641296298053, 1.03981924107),
    (0.738105872266, 0.972268833998), (0.824444363295,
0.889624082279),
    (0.894792677532, 0.792494155836), (0.939395486253,
0.681546643421),
    (0.96111933829, 0.562238253072), (0.970579841181, 0.441758925744),
    (0.971193274221, 0.322118743967), (0.163846223133,
0.249151738053),
    (0.21780354657, 0.204255863861), (0.291299351124, 0.192367318323),
    (0.367460241458, 0.203582210627), (0.4392945113, 0.233135599851),
    (0.586445962425, 0.228141644834), (0.660152671635,
0.195923841854),
    (0.737466449096, 0.182360984545), (0.813236546239,
0.192828009114),
    (0.8707571886, 0.235293377042), (0.51534533827, 0.31863546193),
    (0.516221448289, 0.396200446263), (0.517118861835,
0.473797687758),
    (0.51816430343, 0.553157797772), (0.433701156035, 0.604054457668),
    (0.475501237769, 0.62076344024), (0.520712933176, 0.634268222208),
    (0.565874114041, 0.618796581487), (0.607054002672, 0.60157671656),
    (0.252418718401, 0.331052263829), (0.298663015648,
0.302646354002),
    (0.355749724218, 0.303020650651), (0.403718978315, 0.33867711083),
    (0.352507175597, 0.349987615384), (0.296791759886,
0.350478978225),
    (0.631326076346, 0.334136672344), (0.679073381078, 0.29645404267),
    (0.73597236153, 0.294721285802), (0.782865376271, 0.321305281656),
    (0.740312274764, 0.341849376713), (0.68499850091, 0.343734332172),
    (0.353167761422, 0.746189164237), (0.414587777921,
0.719053835073),
    (0.477677654595, 0.706835892494), (0.522732900812,
0.717092275768),
    (0.569832064287, 0.705414478982), (0.635195811927, 0.71565572516),
    (0.69951672331, 0.739419187253), (0.639447159575, 0.805236879972),
    (0.576410514055, 0.835436670169), (0.525398405766,
0.841706377792),
    (0.47641545769, 0.837505914975), (0.41379548902, 0.810045601727),
    (0.380084785646, 0.749979603086), (0.477955996282, 0.74513234612),
    (0.523389793327, 0.748924302636), (0.571057789237, 0.74332894691),
```

```
    (0.672409137852, 0.744177032192), (0.572539621444,
0.776609286626),
    (0.5240106503, 0.783370783245), (0.477561227414, 0.778476346951)])

TPL_MIN, TPL_MAX = np.min(TEMPLATE, axis=0), np.max(TEMPLATE, axis=0)
MINMAX_TEMPLATE = (TEMPLATE - TPL_MIN) / (TPL_MAX - TPL_MIN)

INNER_EYES_AND_BOTTOM_LIP = np.array([39, 42, 57])
OUTER_EYES_AND_NOSE = np.array([36, 45, 33])

def align_faces(images, landmarks, idx=INNER_EYES_AND_BOTTOM_LIP):
    faces = []
    for (img, marks) in zip(images, landmarks):
        imgDim = img.shape[0]
        coords = shape_to_coords(marks)
        H = cv2.getAffineTransform(coords[idx], imgDim *
MINMAX_TEMPLATE[idx])
        warped = cv2.warpAffine(img, H, (imgDim, imgDim))
        faces.append(warped)
    return faces
```

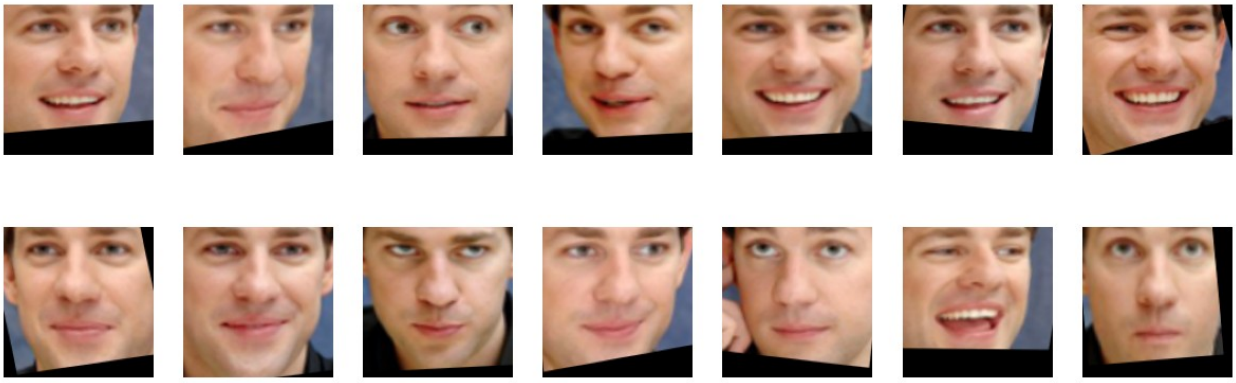
Hints

The provided function `face_landmarks()` computes the landmarks for a list of cropped faces.

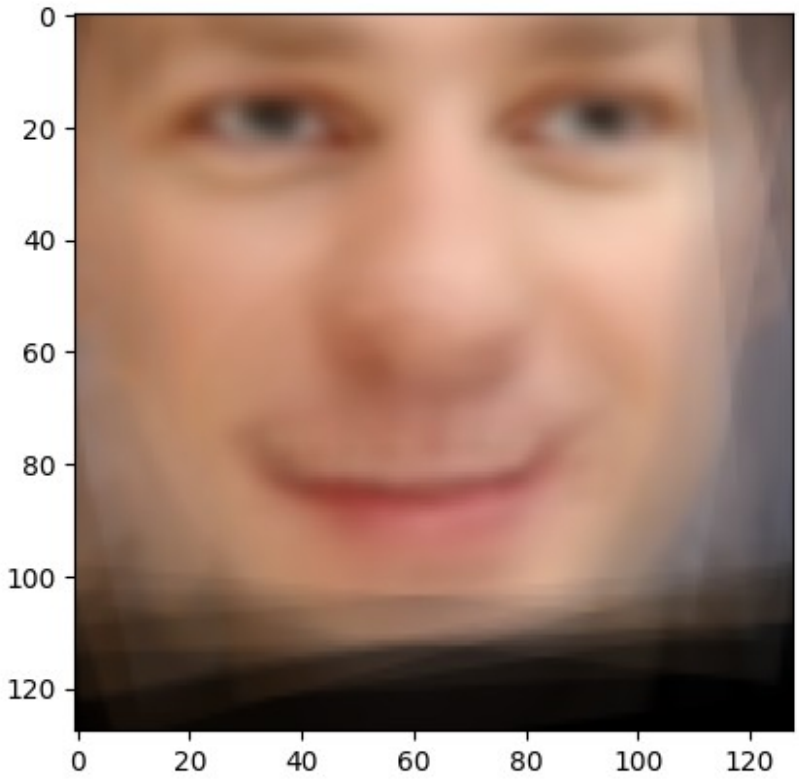
```
landmarks = face_landmarks(faces)

new_faces = []
for (face,shape) in zip(faces, landmarks):
    canvas = face.copy()
    coords = shape_to_coords(shape)
    for p in coords:
        cv2.circle(canvas, (int(p[0]),int(p[1])), 1, (0, 0, 255), -1)
    new_faces.append(canvas)

show_grid(new_faces, figsize=(15,5))
```



```
aligned = align_faces(faces, landmarks)

show_grid(aligned, figsize=(15,5))
```

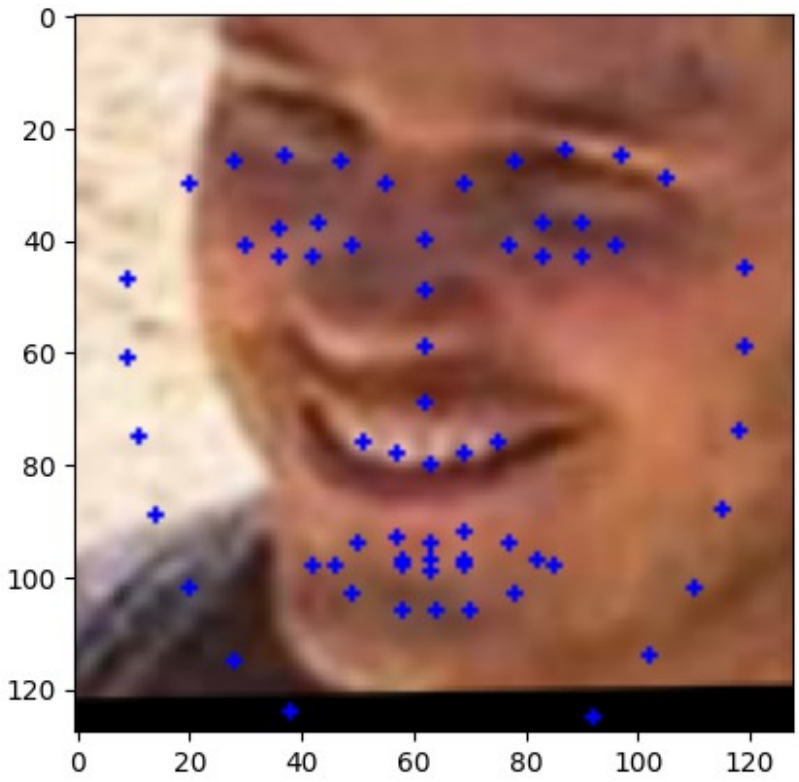


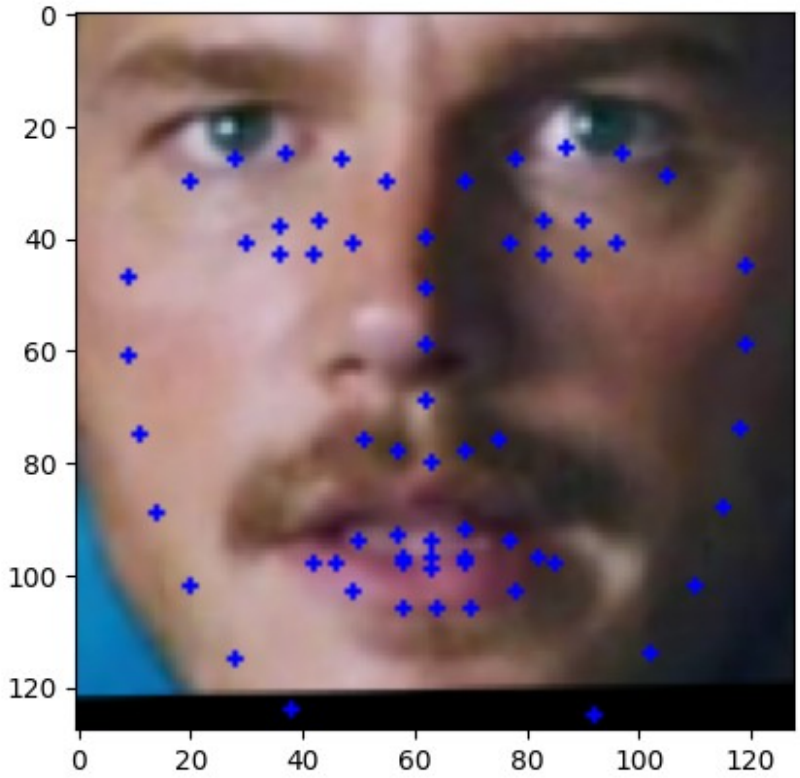
```
plt.imshow( np.stack(aligned,
axis=3).astype(np.float32).mean(axis=3)/255 )

<matplotlib.image.AxesImage at 0x7f83ac2c0bb0>
```

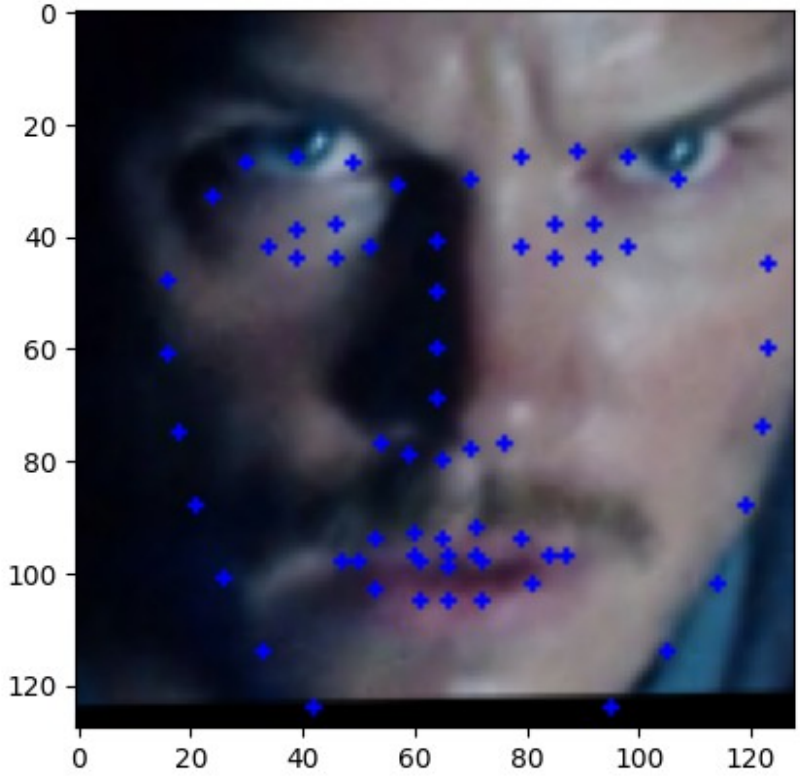


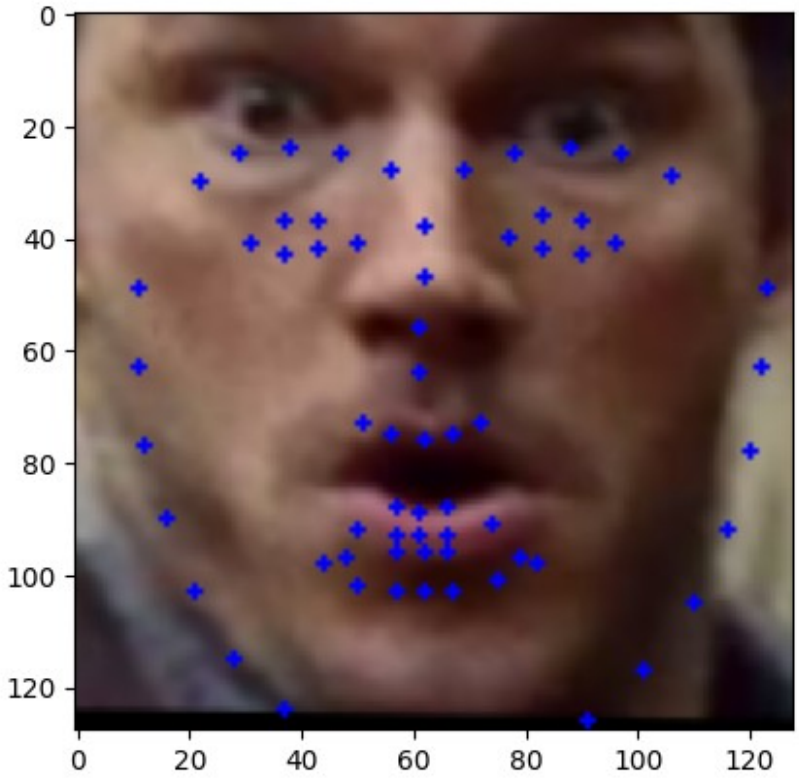

```
gray_images = [cv2.cvtColor(image, cv2.COLOR_RGB2GRAY) for image in
cropped_image]
gray_images = [cv2.convertScaleAbs(image, alpha=(255.0)) for image in
```

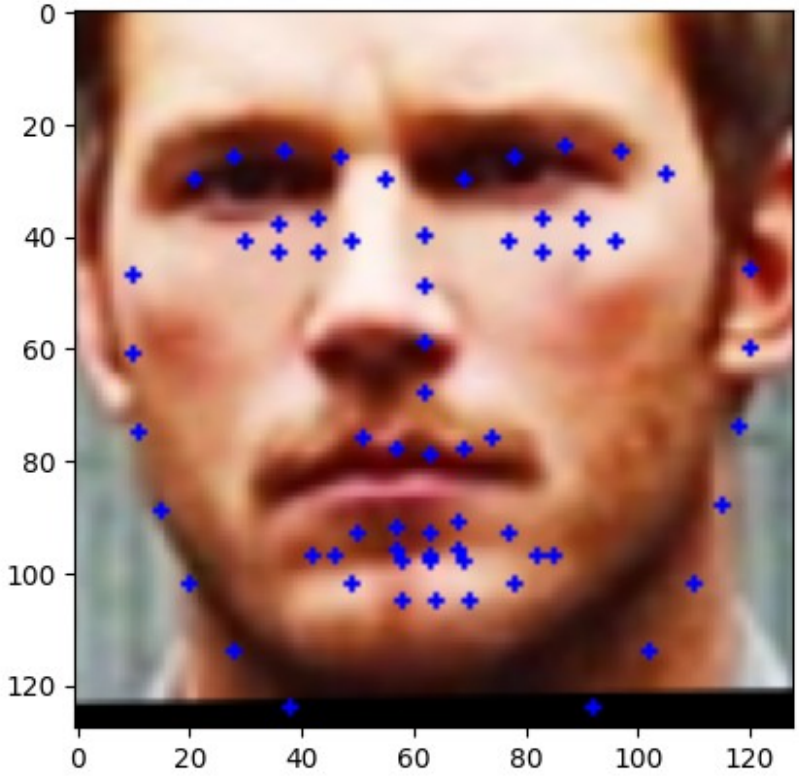
```
cropped_image]

# Détection des landmarks sur les images en niveaux de gris
landmarks_cropped_images = face_landmarks(gray_images)
# Alignement des visages
aligned_faces = align_faces(cropped_image, landmarks_cropped_images)
#landmarks_cropped_images = [face_landmarks(image) for image in
cropped_image]
# Aligner le visage
#aligned_face = align_faces(cropped_image, landmarks_cropped_images)

for image, landmarks in zip(aligned_faces, landmarks_cropped_images):
    canvas = image.copy()
    coords = shape_to_coords(landmarks)
    for p in coords:
        cv2.circle(canvas, (int(p[0]), int(p[1])), 1, (0, 0, 255), -1)
    # Afficher ou sauvegarder chaque image modifiée
    plt.imshow(canvas)
    plt.show()
```

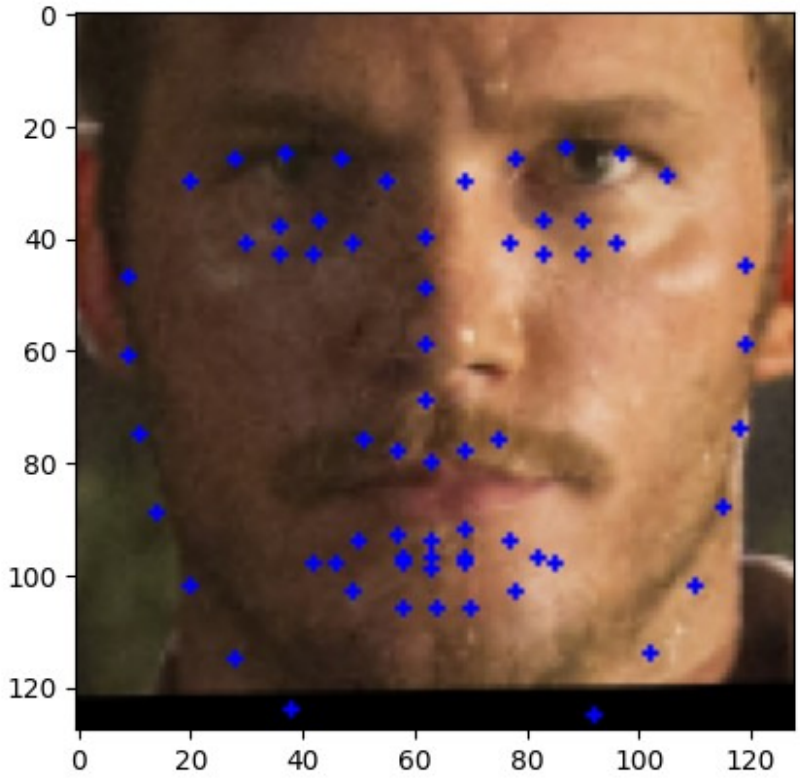


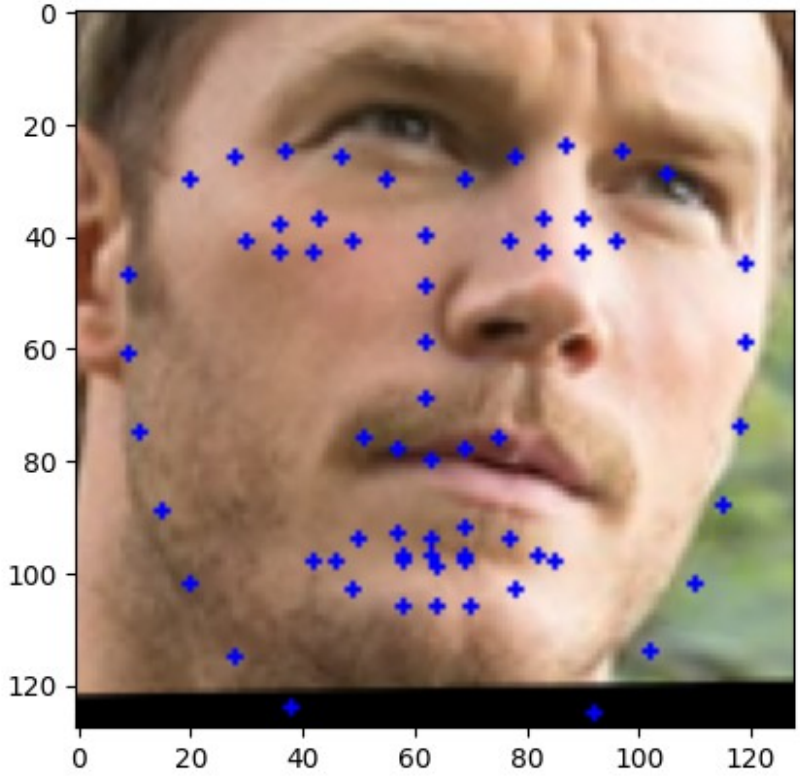

0
20
40
60
80
100
120
0
20
40
60
80
100
120

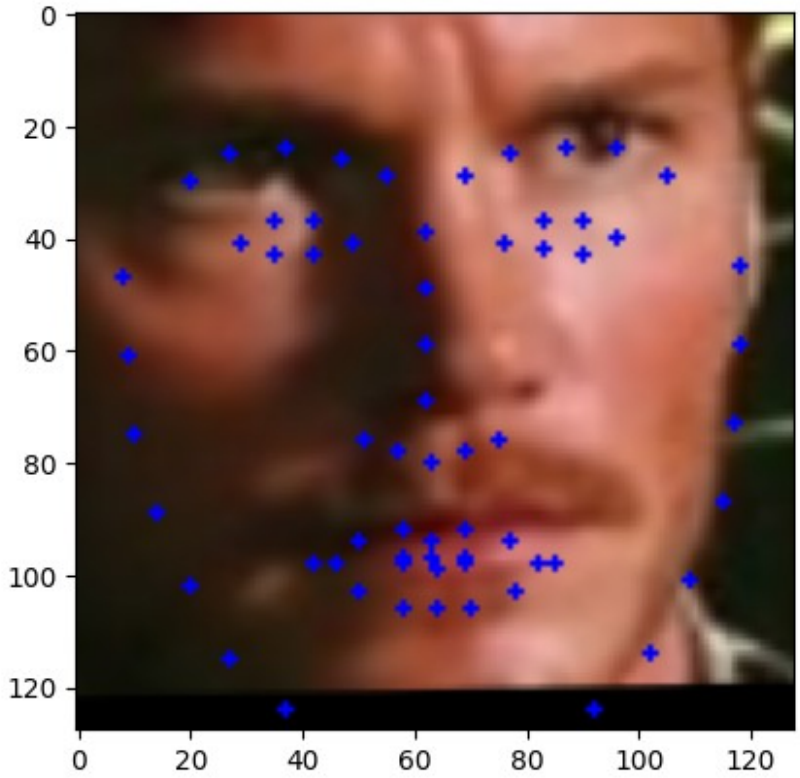

0
20
40
60
80
100
120
0
20
40
60
80
100
120

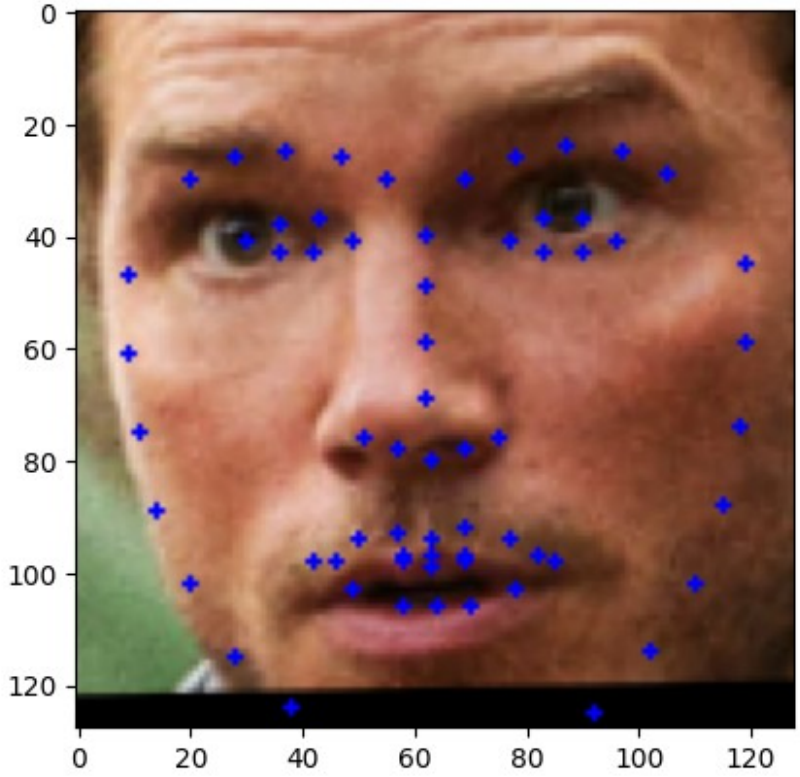

0
20
40
60
80
100
120
0
20
40
60
80
100
120

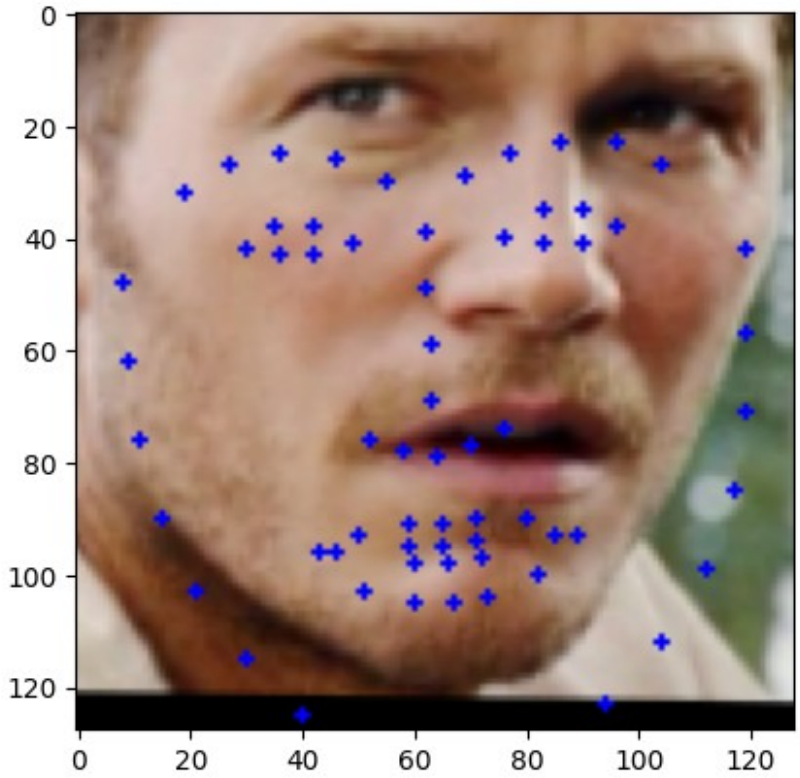

0
20
40
60
80
100
120
0
20
40
60
80
100
120

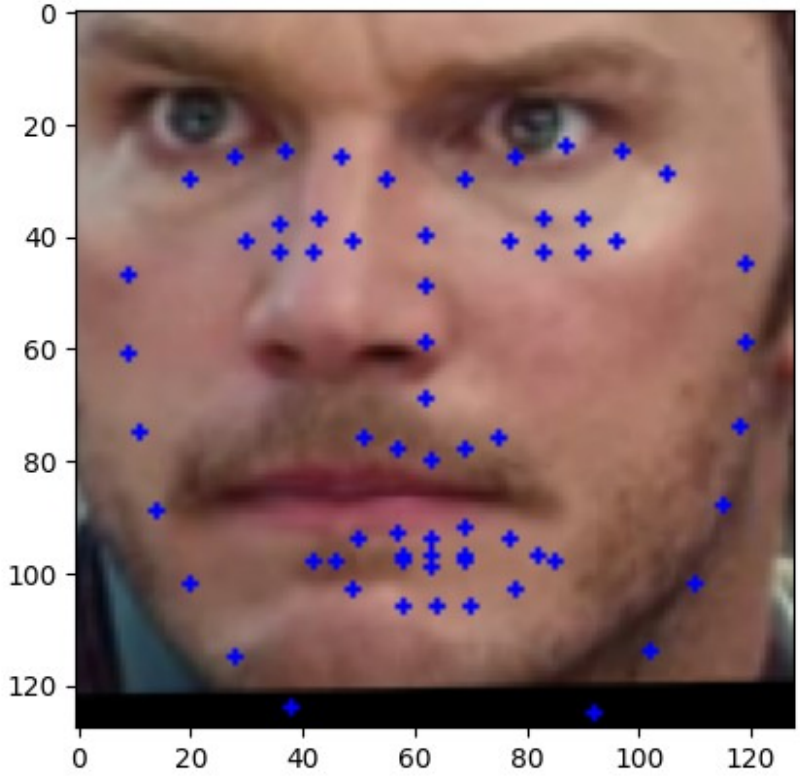

0
20
40
60
80
100
120
0
20
40
60
80
100
120

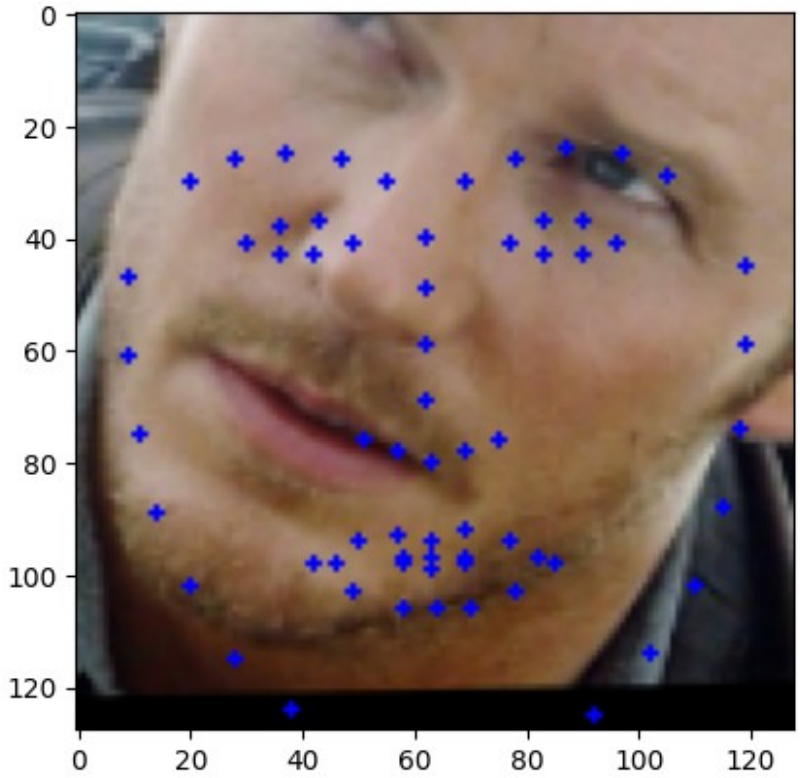

0
20
40
60
80
100
120
0
20
40
60
80
100
120

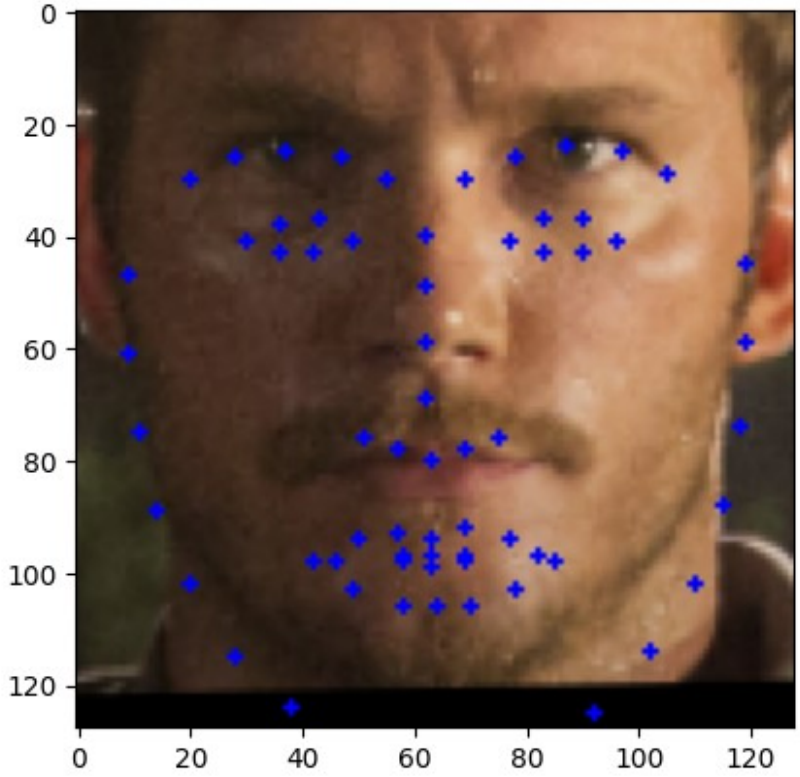

0
20
40
60
80
100
120
0
20
40
60
80
100
120

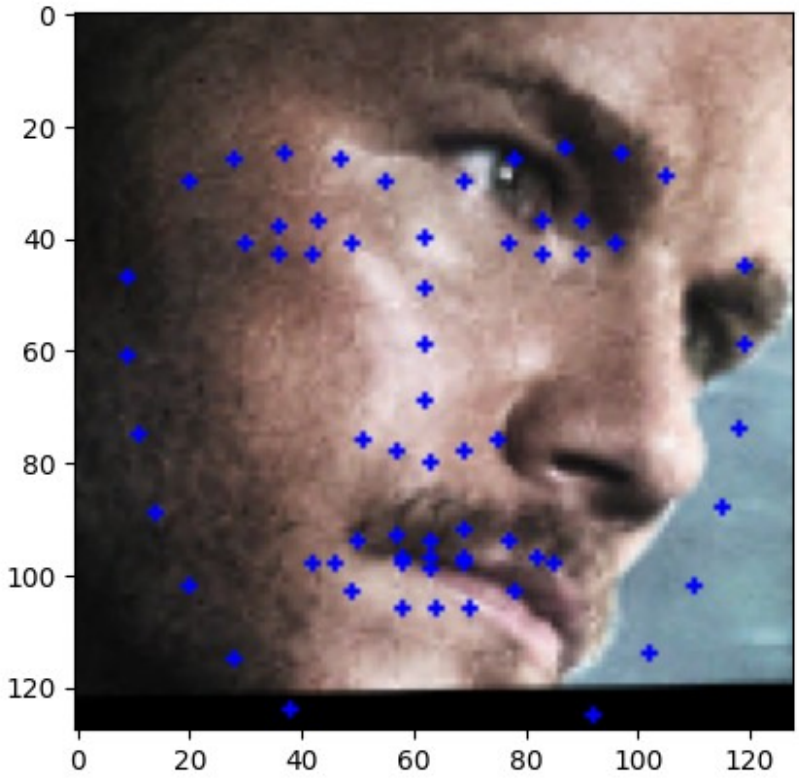

0
20
40
60
80
100
120
0
20
40
60
80
100
120

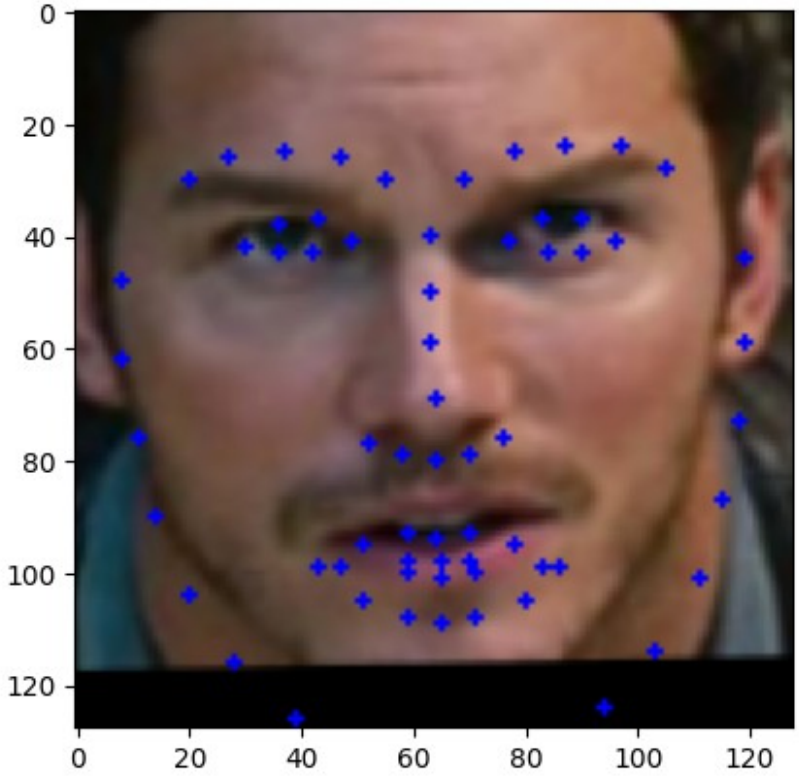

0
20
40
60
80
100
120
0
20
40
60
80
100
120

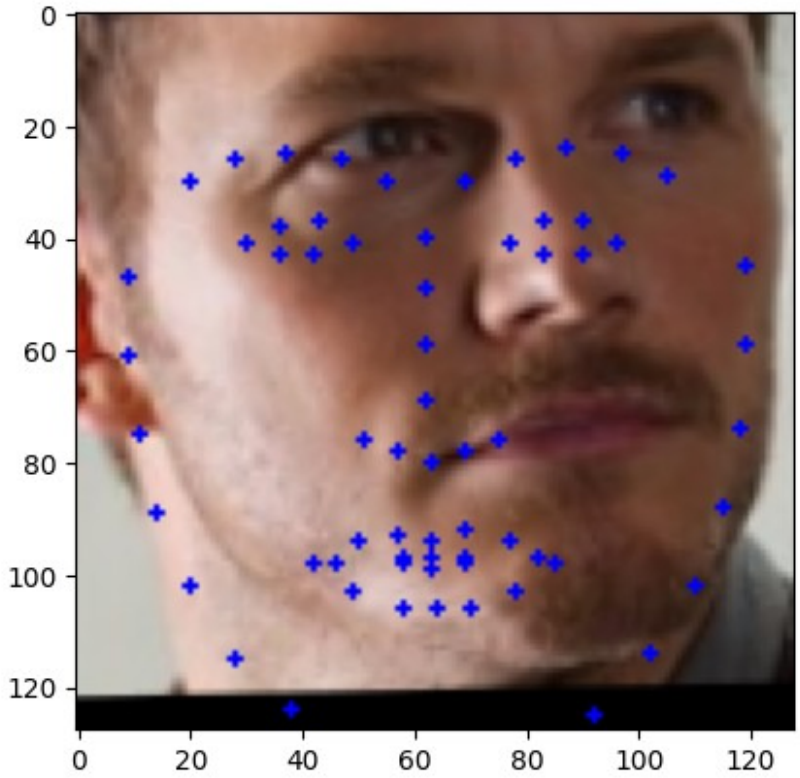

0
20
40
60
80
100
120
0
20
40
60
80
100
120

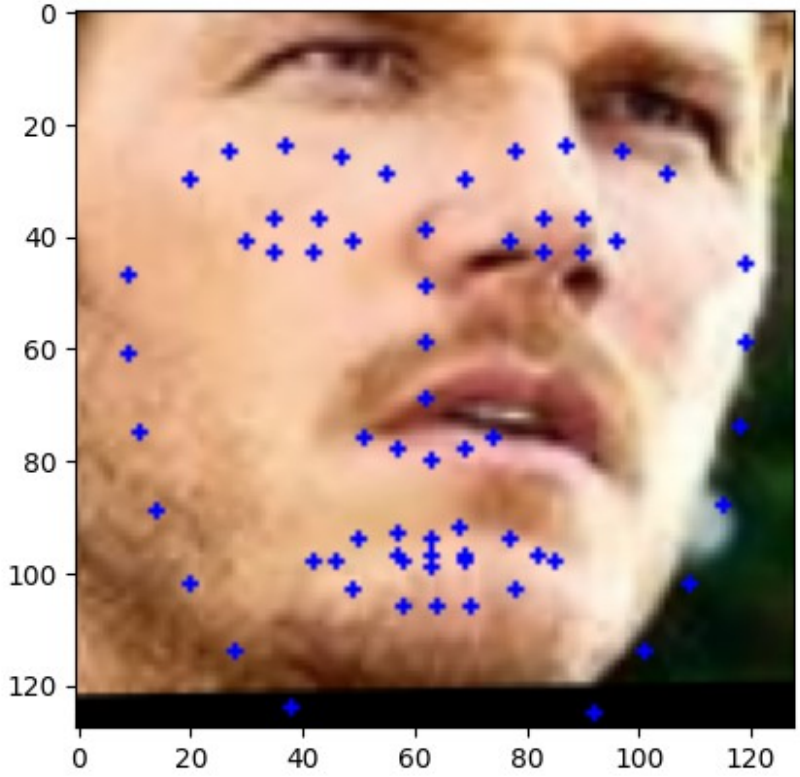

0
20
40
60
80
100
120
0
20
40
60
80
100
120

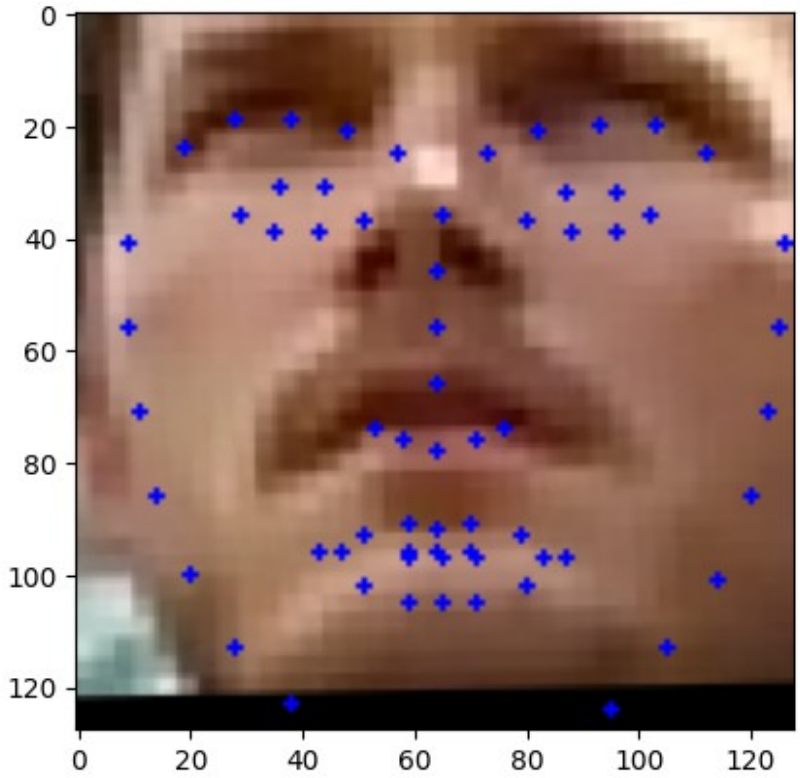

0
20
40
60
80
100
120
0
20
40
60
80
100
120

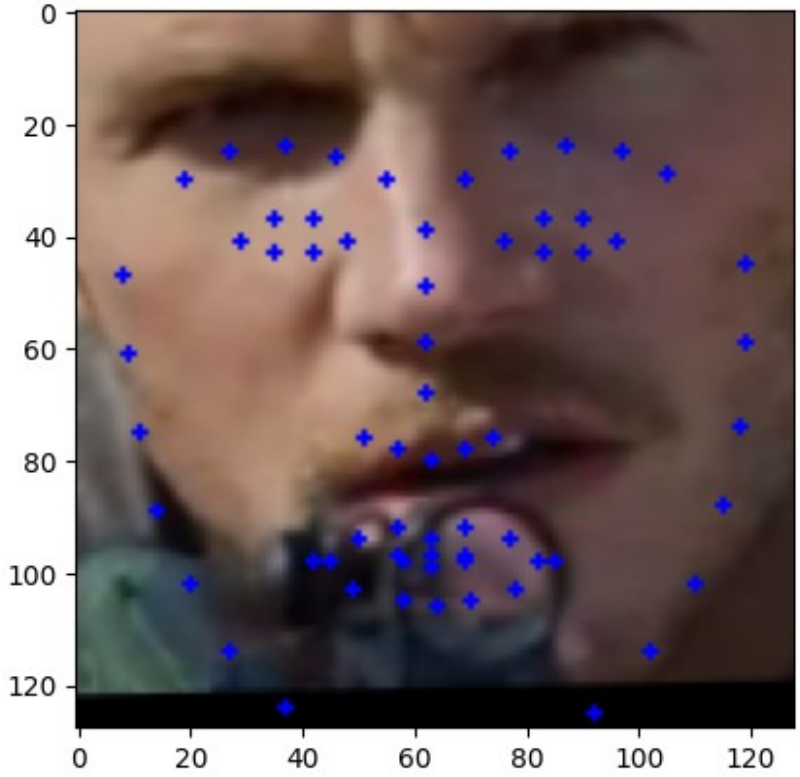

0
20
40
60
80
100
120
0
20
40
60
80
100
120

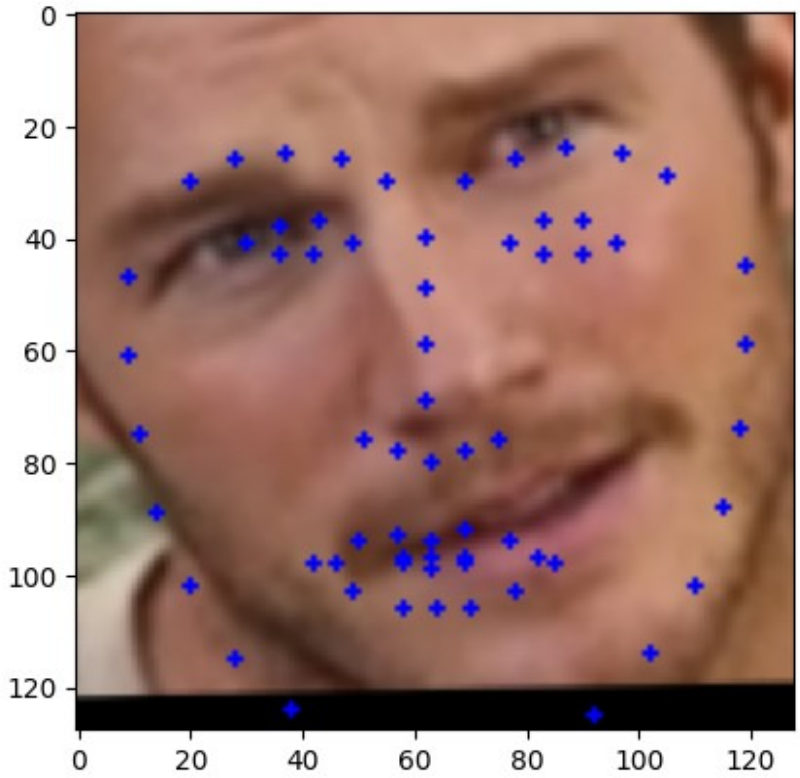

0
20
40
60
80
100
120
0
20
40
60
80
100
120

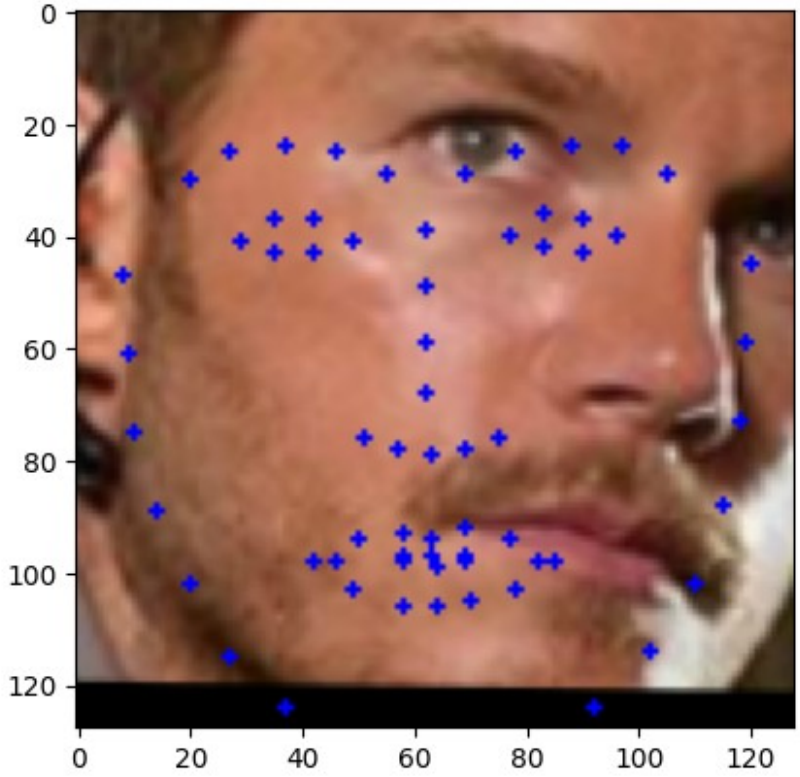

0
20
40
60
80
100
120
0
20
40
60
80
100
120

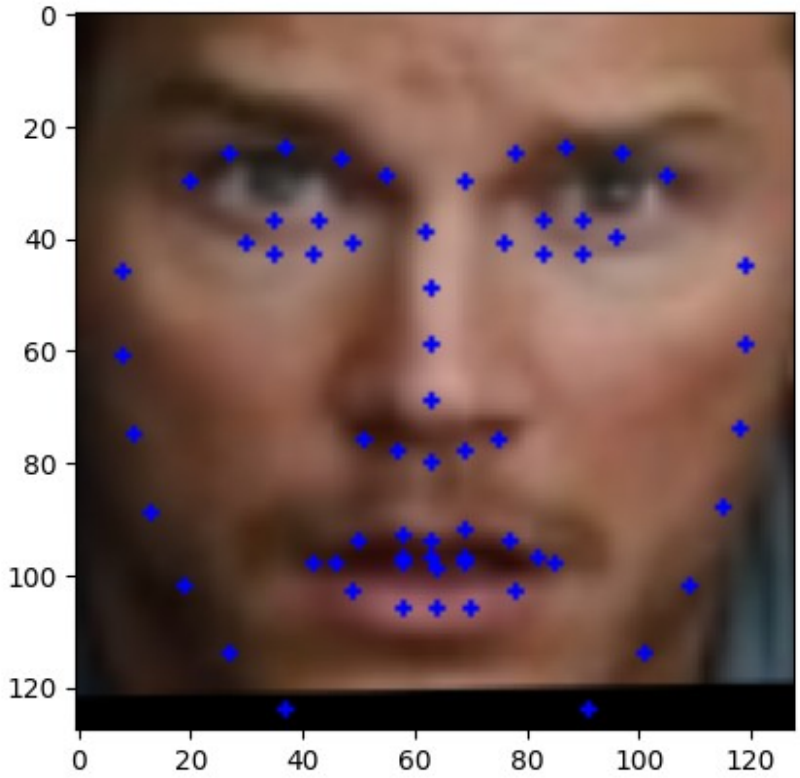

0
20
40
60
80
100
120
0
20
40
60
80
100
120

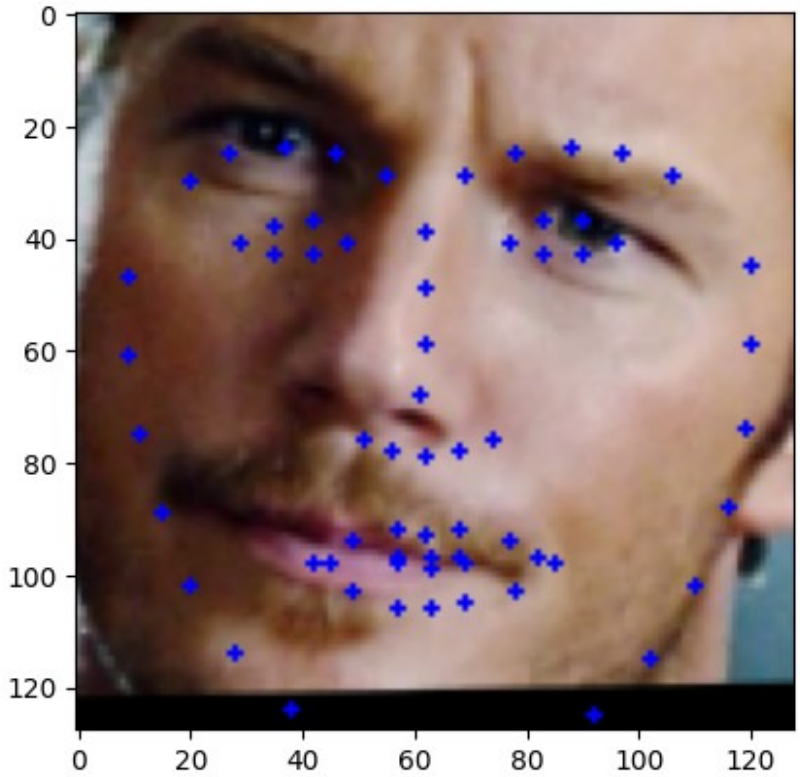

0
20
40
60
80
100
120
0
20
40
60
80
100
120

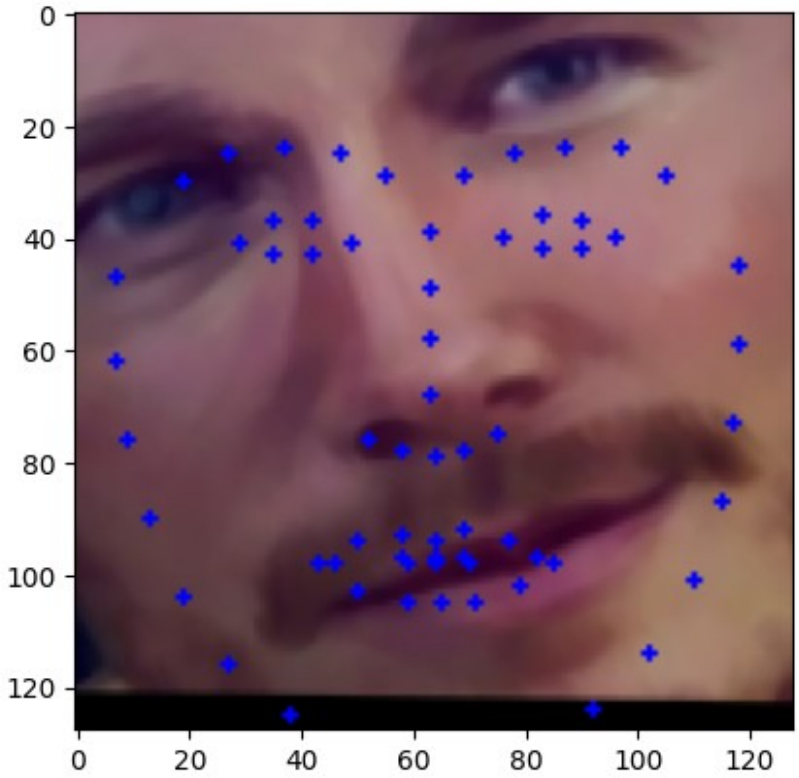

0
20
40
60
80
100
120
0
20
40
60
80
100
120

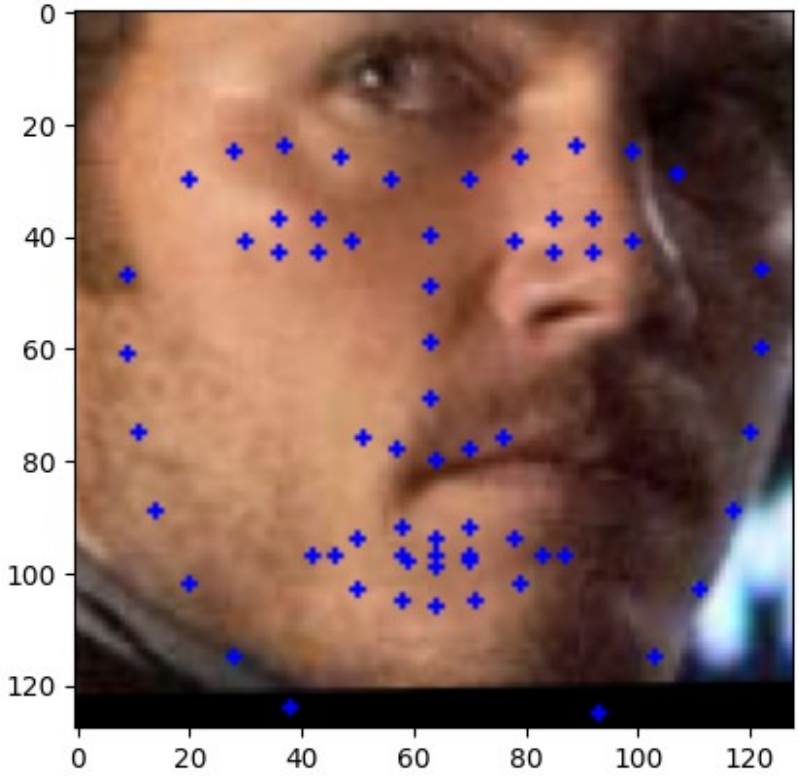

0
20
40
60
80
100
120
0
20
40
60
80
100
120

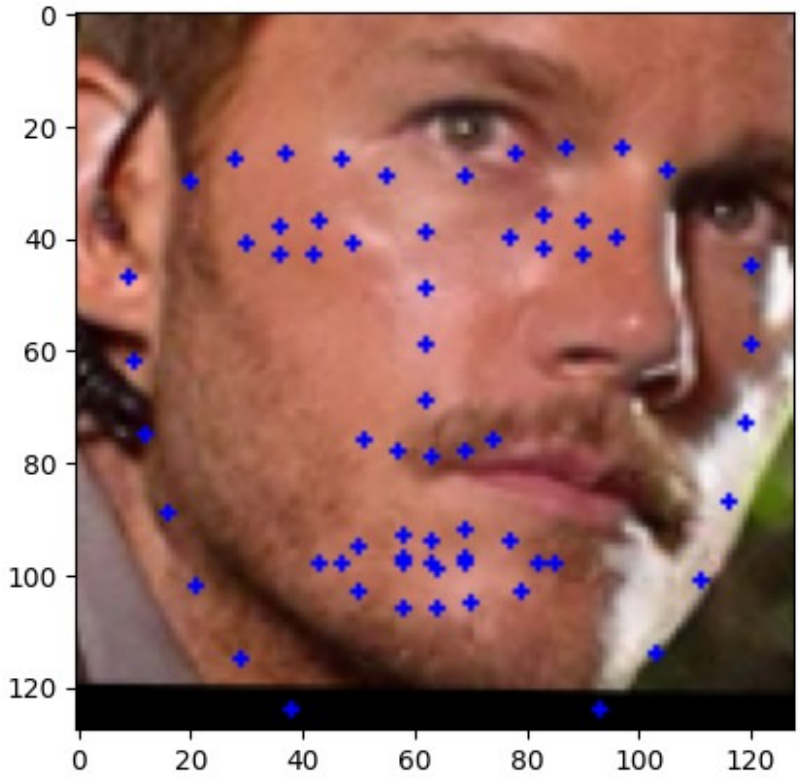

0
20
40
60
80
100
120
0
20
40
60
80
100
120

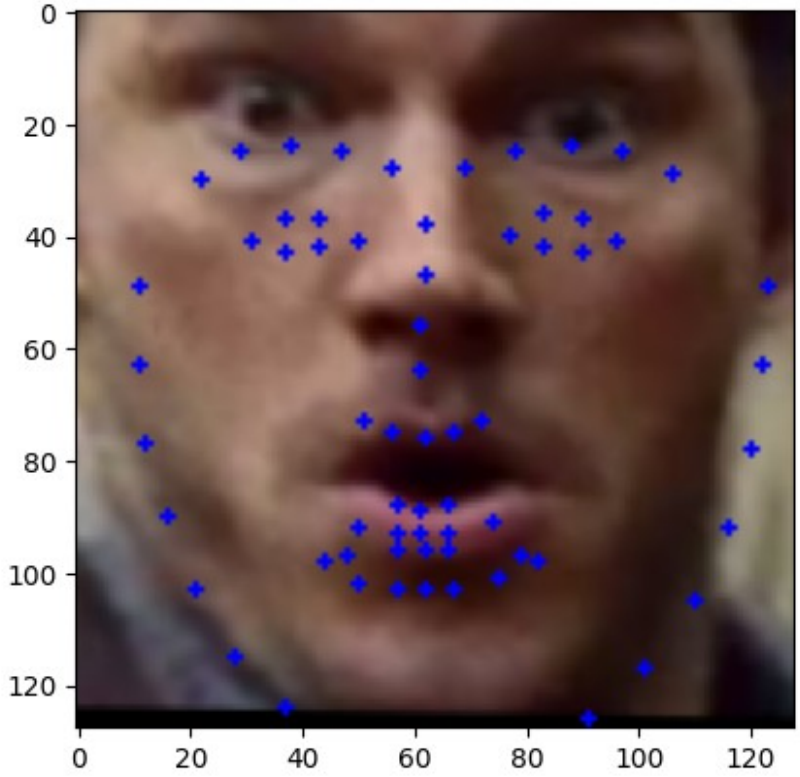

0
20
40
60
80
100
120
0
20
40
60
80
100
120

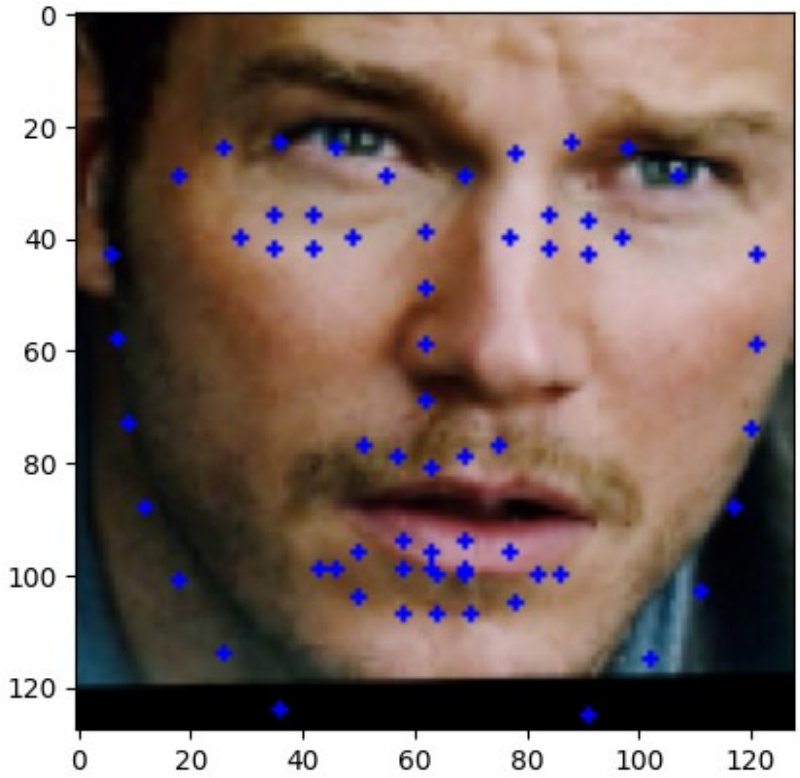

0
20
40
60
80
100
120
0
20
40
60
80
100
120

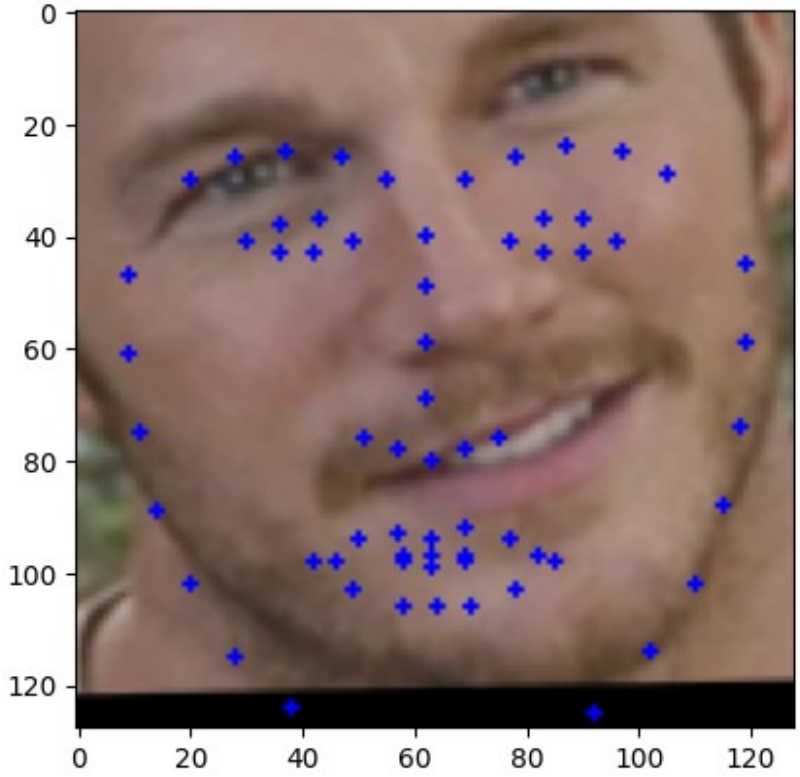

0
20
40
60
80
100
120
0
20
40
60
80
100
120

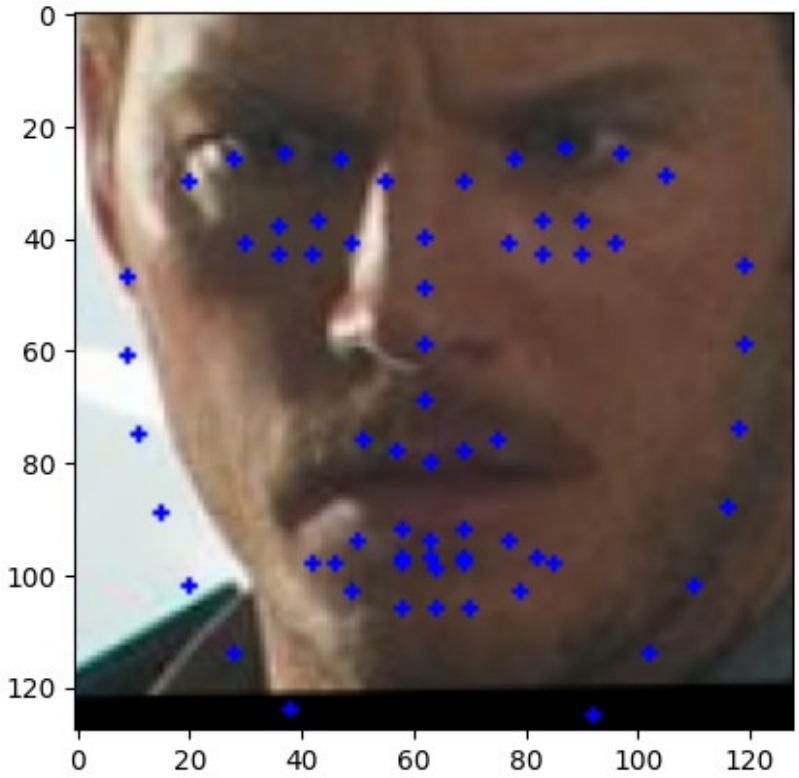

0
20
40
60
80
100
120
0
20
40
60
80
100
120

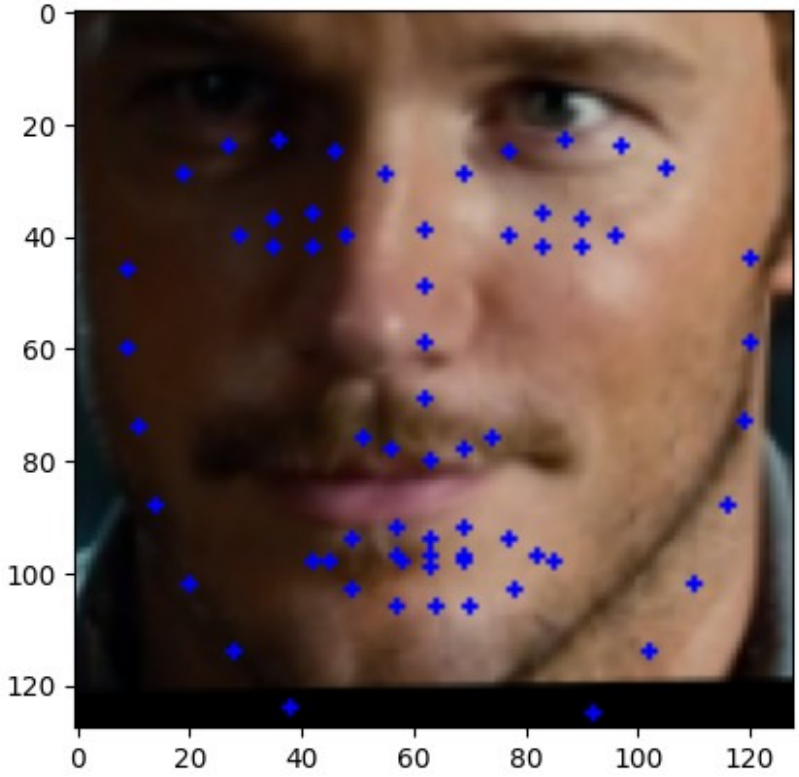

0
20
40
60
80
100
120
0
20
40
60
80
100
120

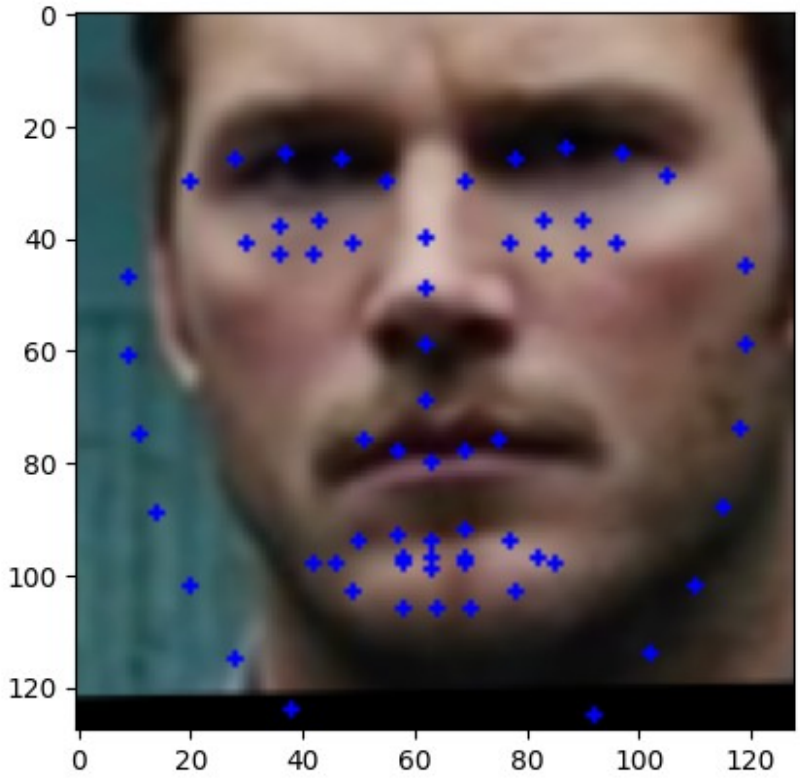

0
20
40
60
80
100
120
0
20
40
60
80
100
120

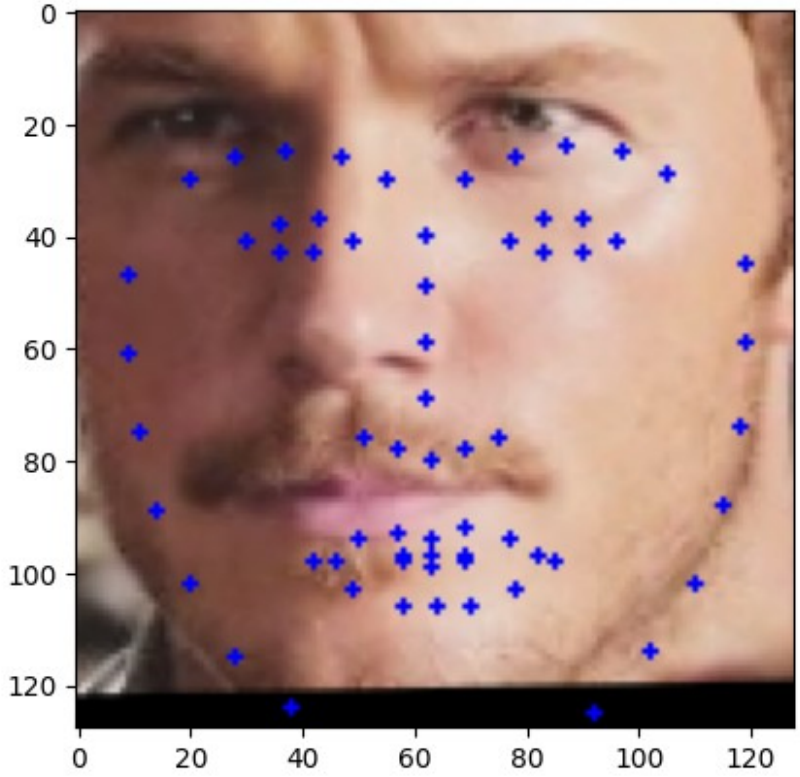

0
20
40
60
80
100
120
0
20
40
60
80
100
120

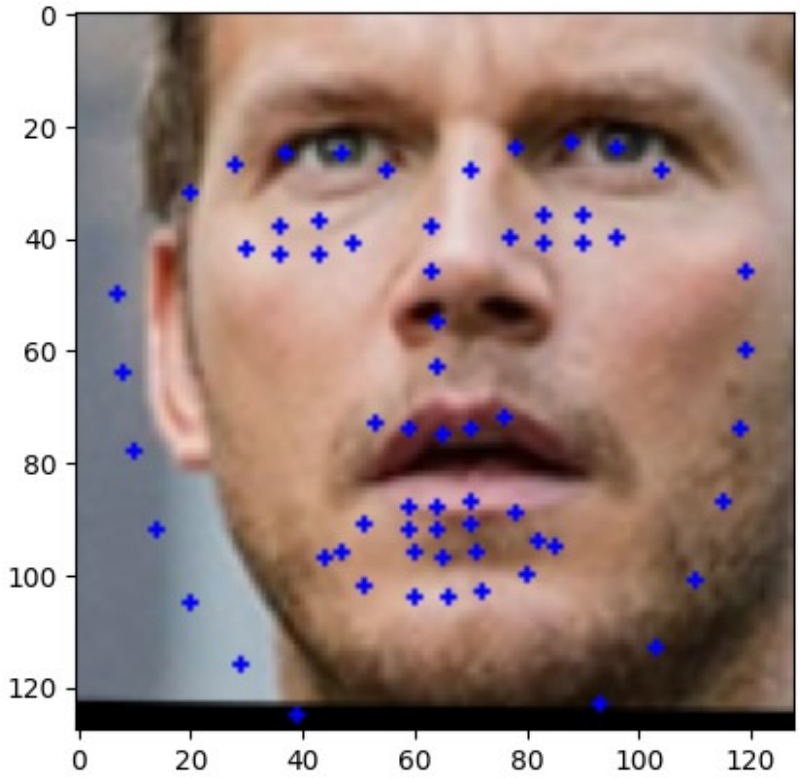

0
20
40
60
80
100
120
0
20
40
60
80
100
120

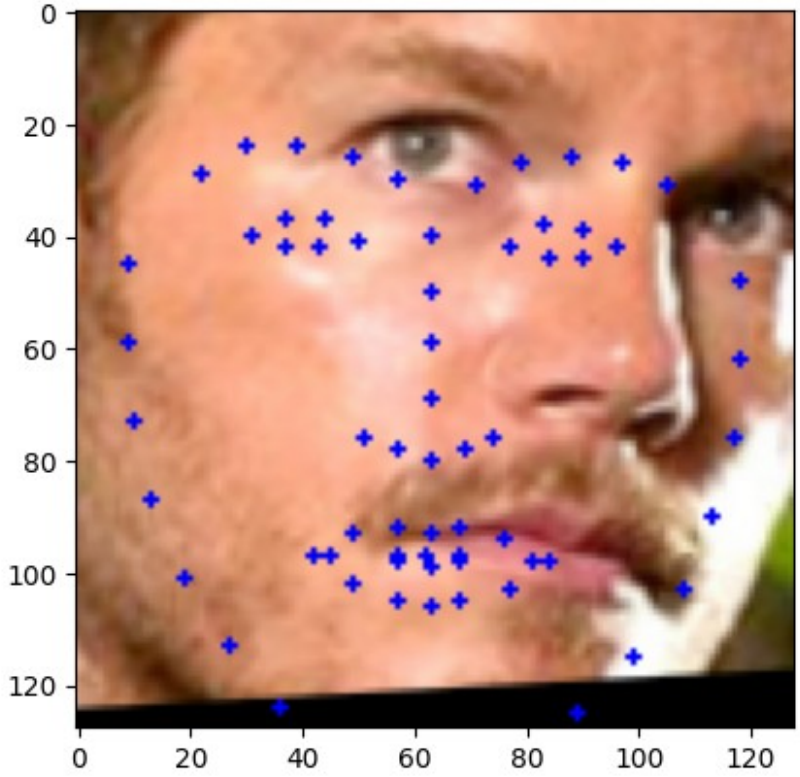

0
20
40
60
80
100
120
0
20
40
60
80
100
120

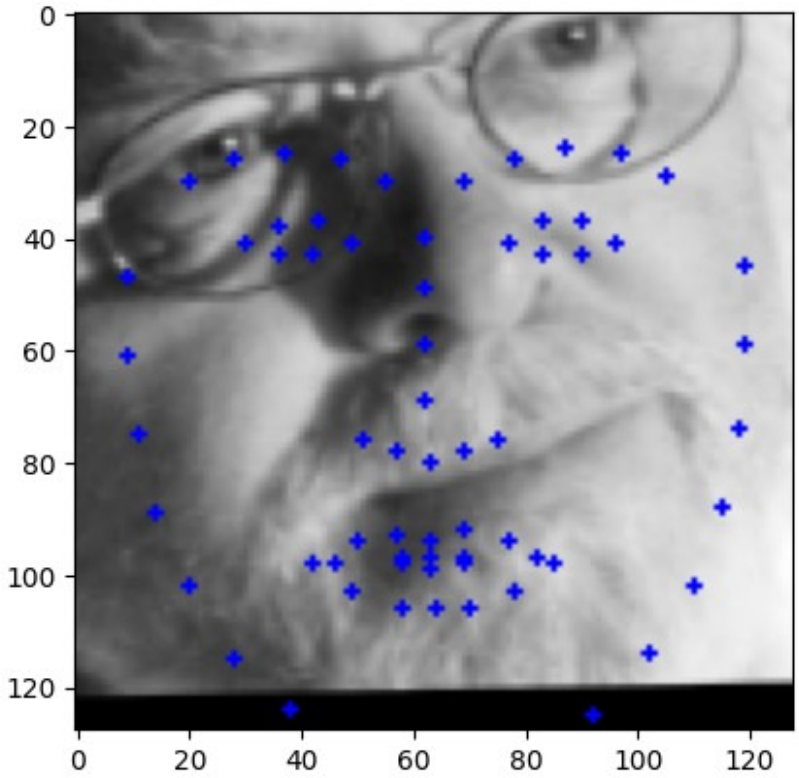

0
20
40
60
80
100
120
0
20
40
60
80
100
120

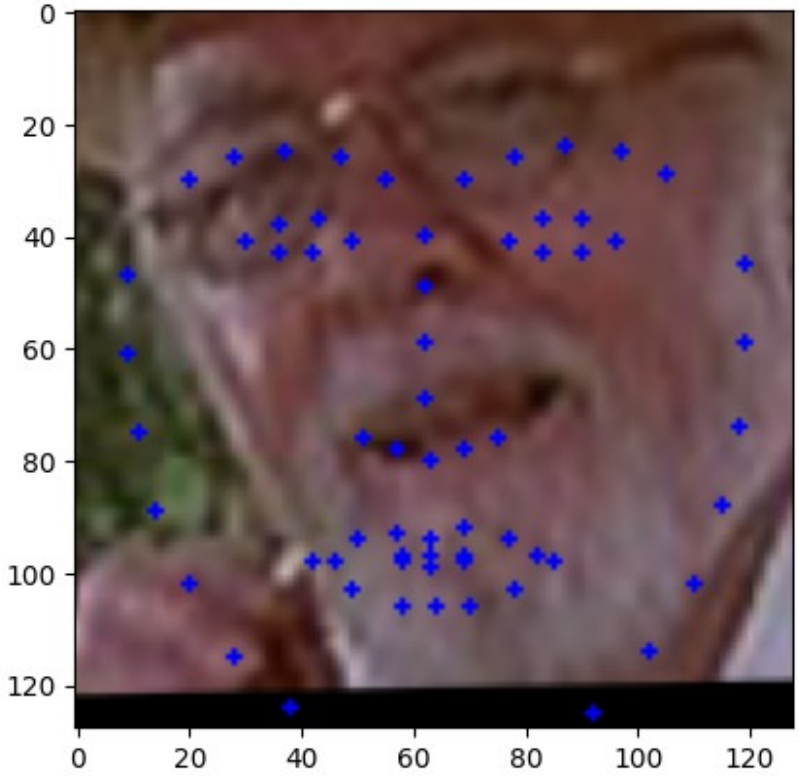

0
20
40
60
80
100
120
0
20
40
60
80
100
120

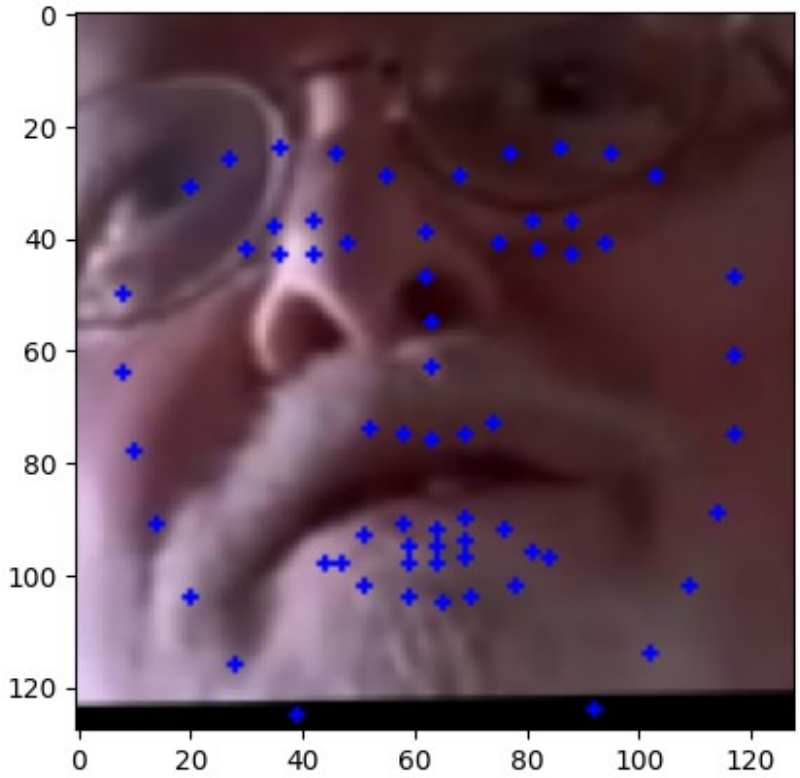

0
20
40
60
80
100
120
0
20
40
60
80
100
120

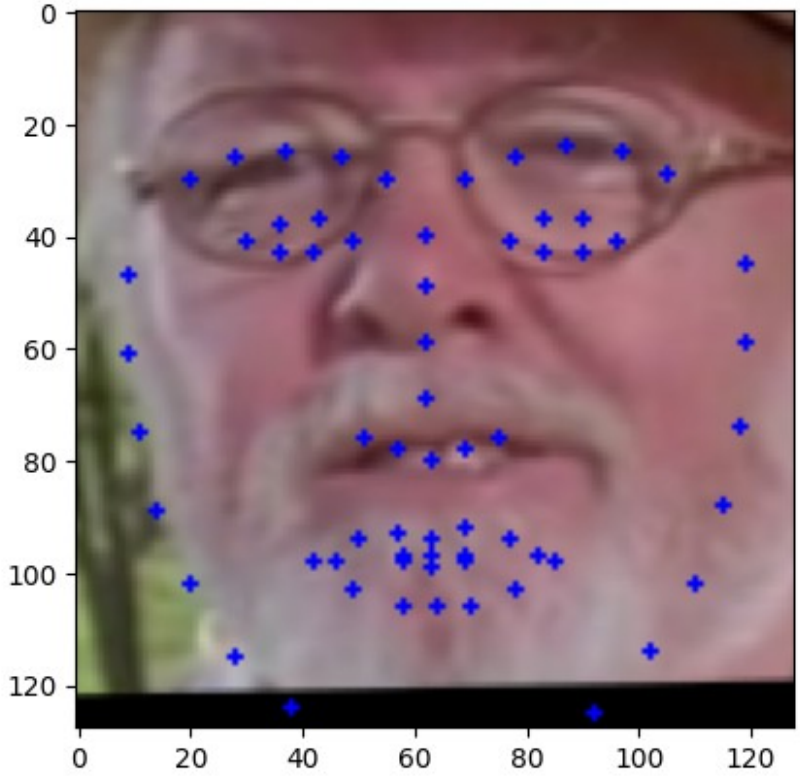

0
20
40
60
80
100
120
0
20
40
60
80
100
120

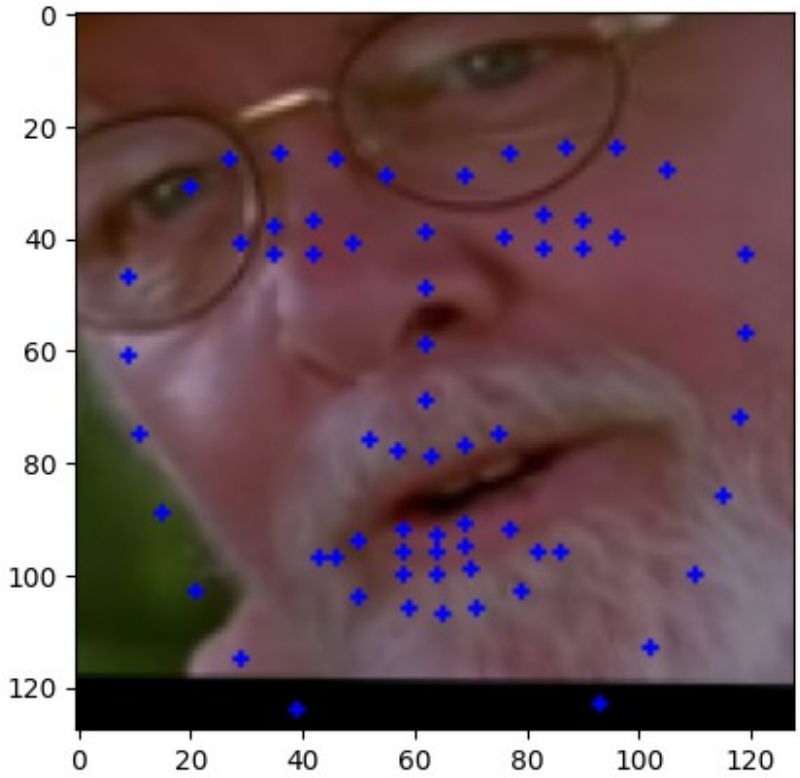

0
20
40
60
80
100
120
0
20
40
60
80
100
120

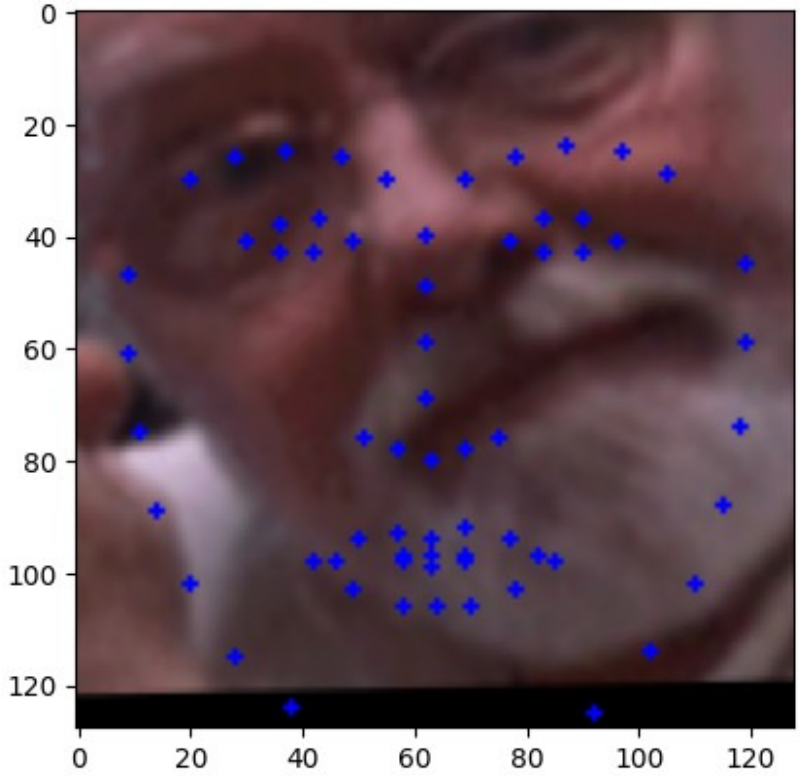

0
20
40
60
80
100
120
0
20
40
60
80
100
120

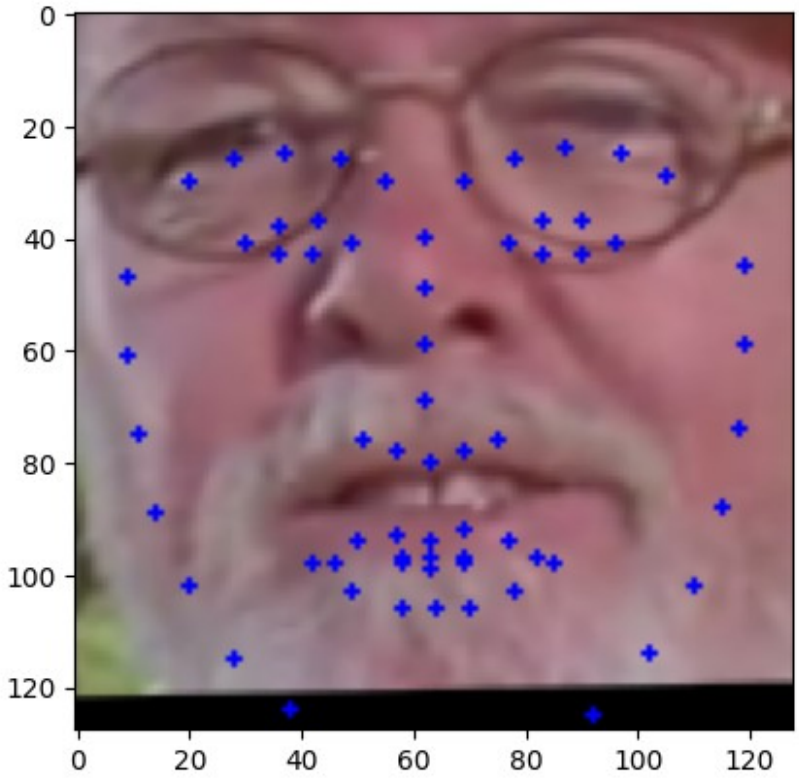

0
20
40
60
80
100
120
0
20
40
60
80
100
120

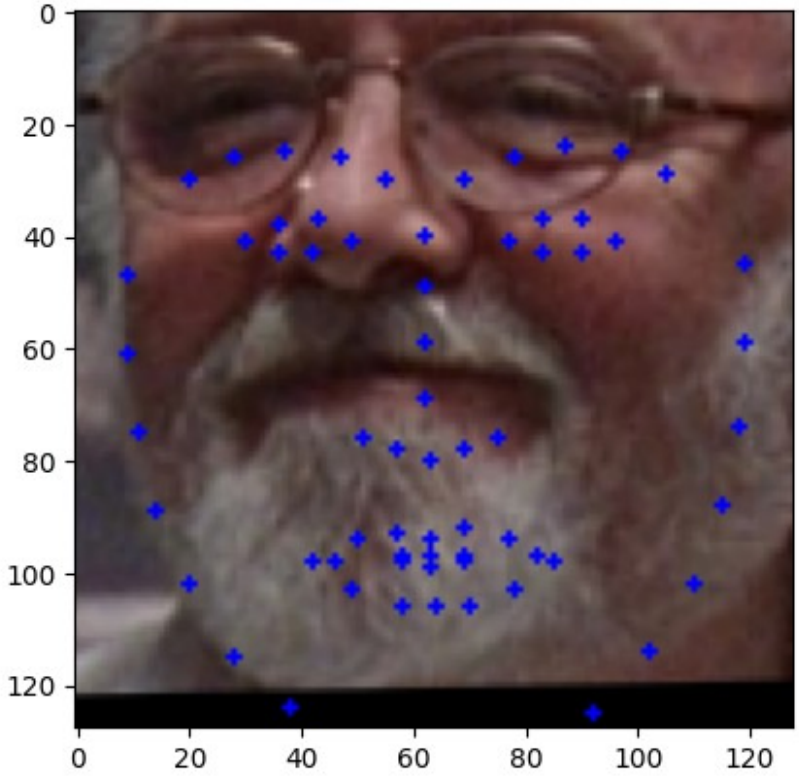

0
20
40
60
80
100
120
0
20
40
60
80
100
120

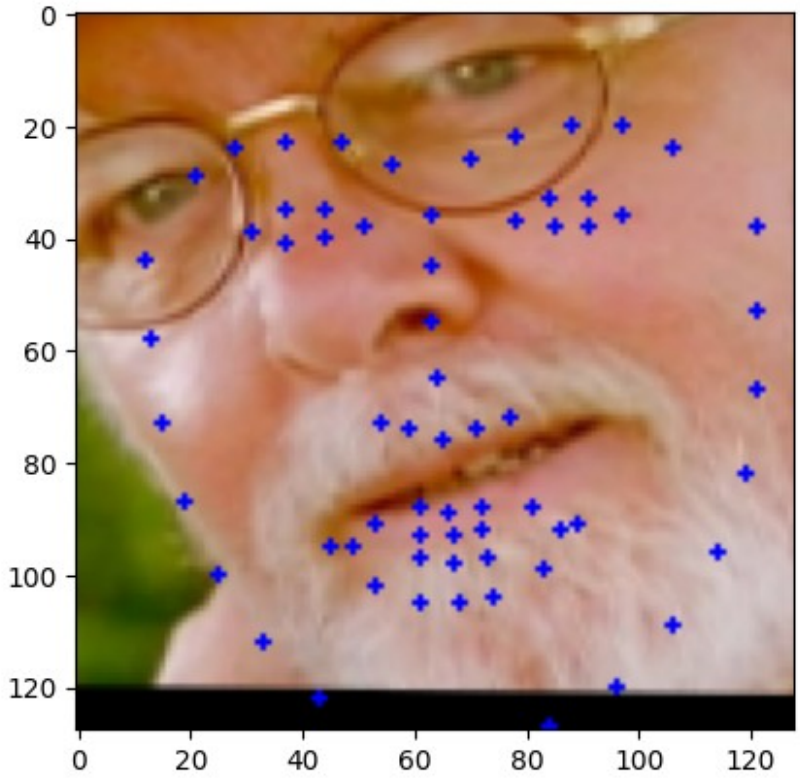

0
20
40
60
80
100
120
0
20
40
60
80
100
120

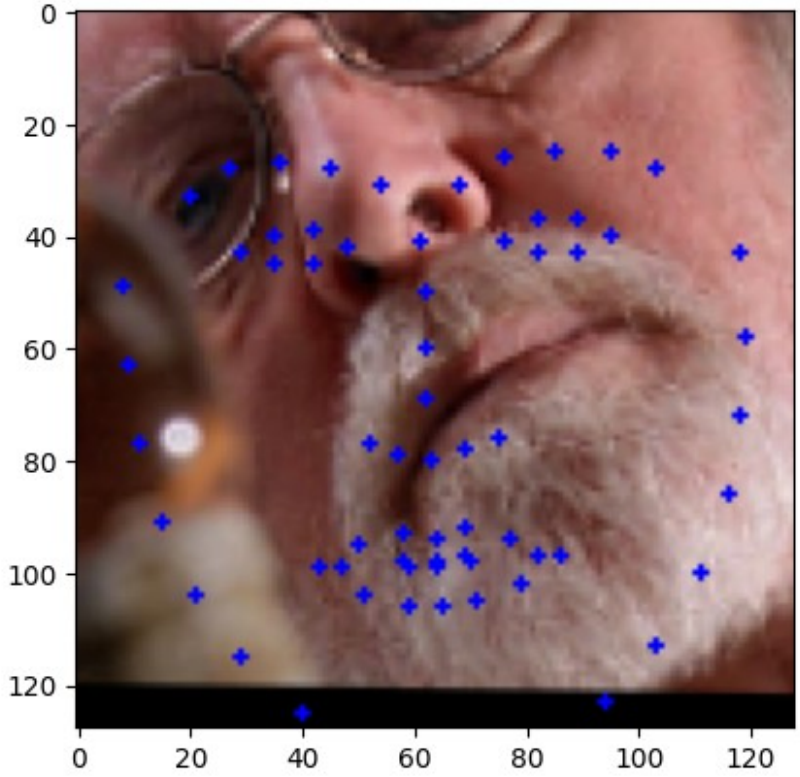

0
20
40
60
80
100
120
0
20
40
60
80
100
120

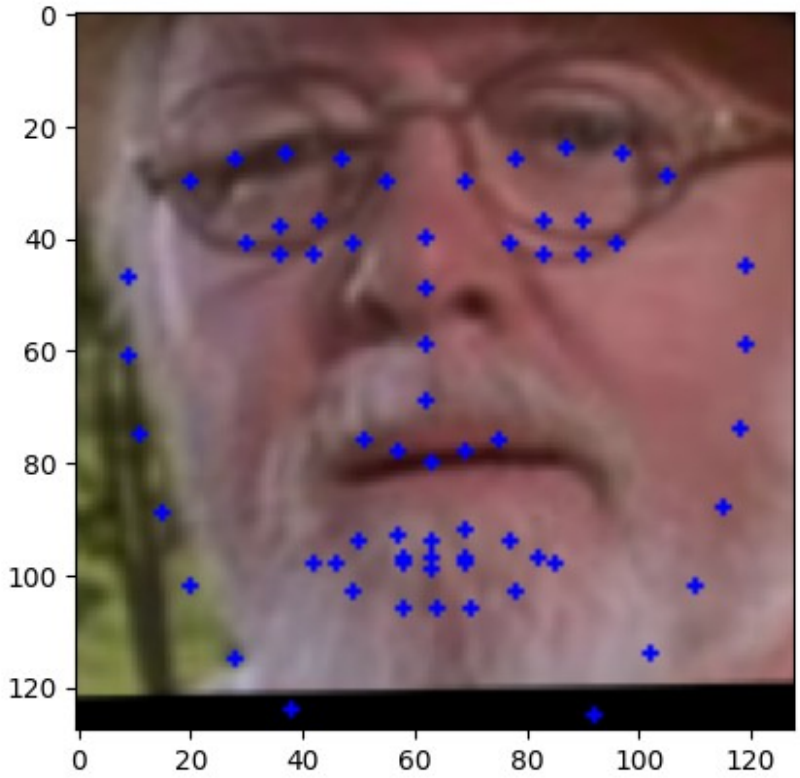

0
20
40
60
80
100
120
0
20
40
60
80
100
120

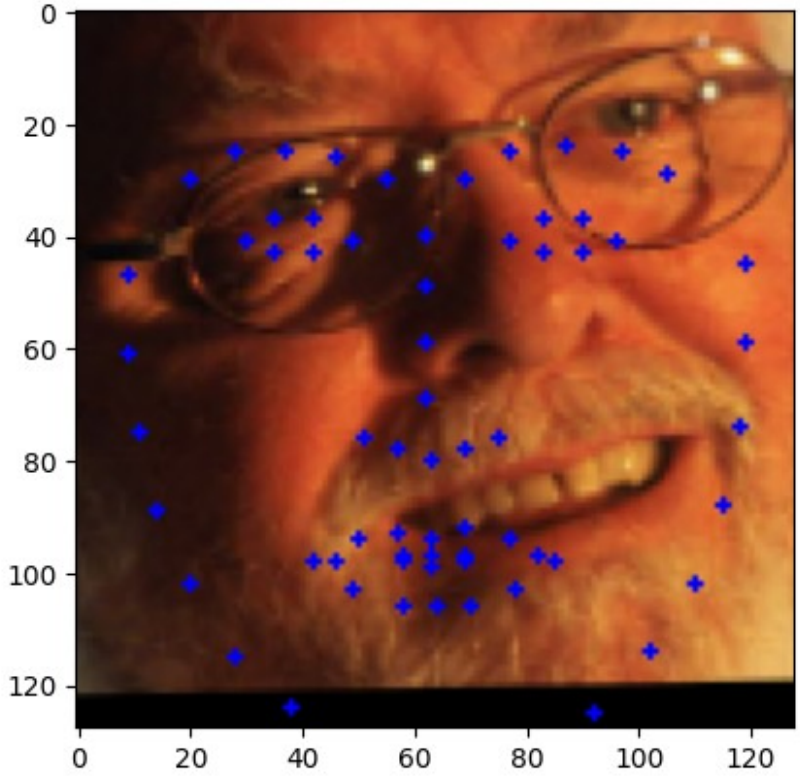

0
20
40
60
80
100
120
0
20
40
60
80
100
120

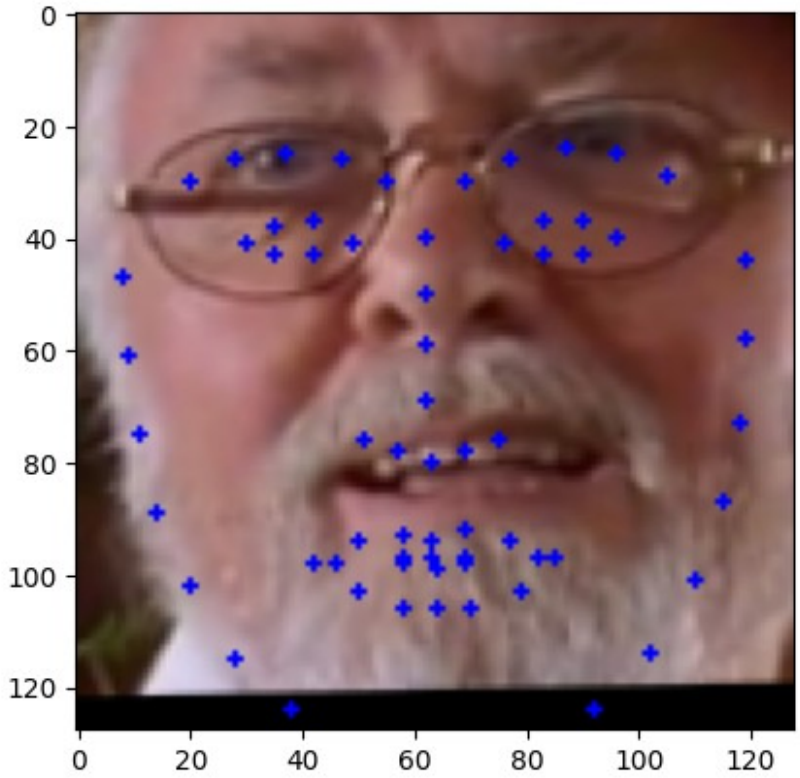

0
20
40
60
80
100
120
0
20
40
60
80
100
120

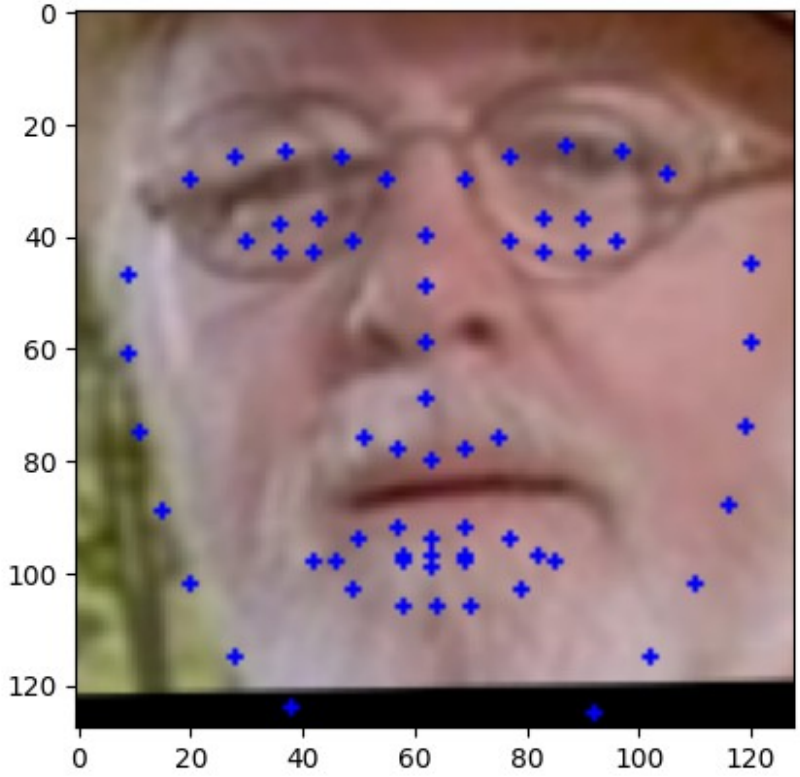

0
20
40
60
80
100
120
0
20
40
60
80
100
120

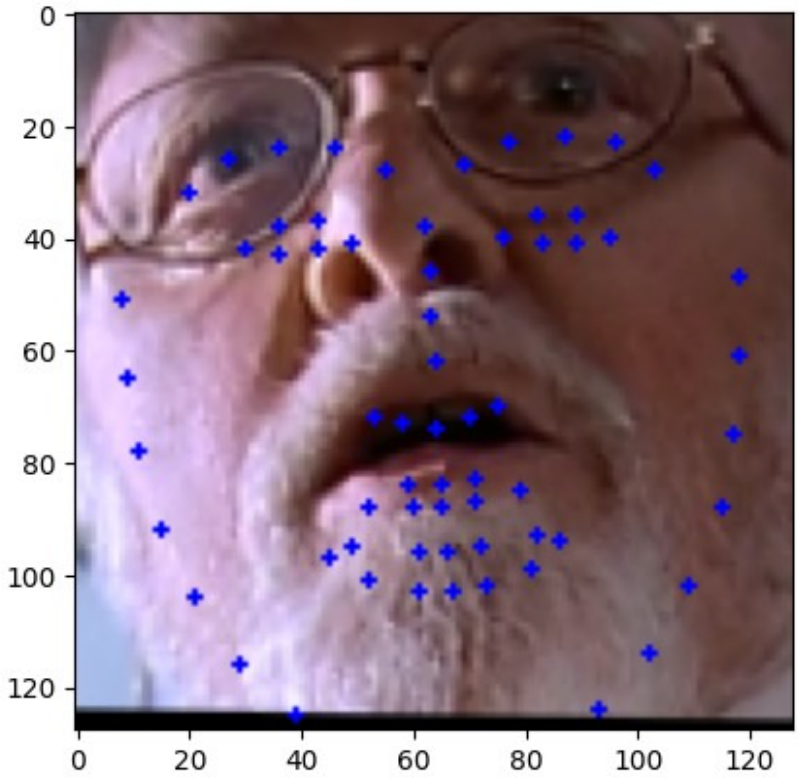

0
20
40
60
80
100
120
0
20
40
60
80
100
120

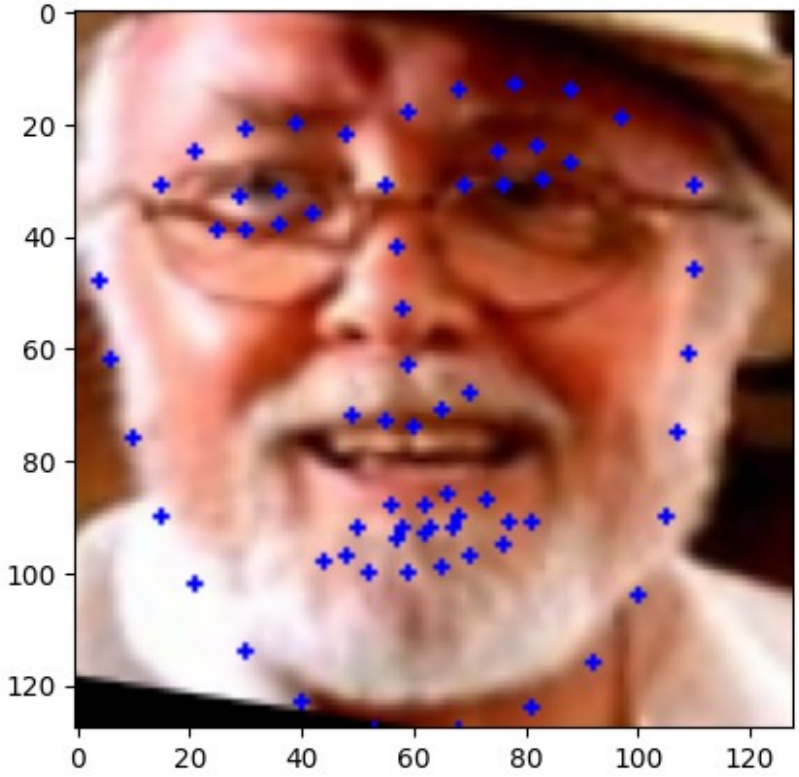

0
20
40
60
80
100
120
0
20
40
60
80
100
120

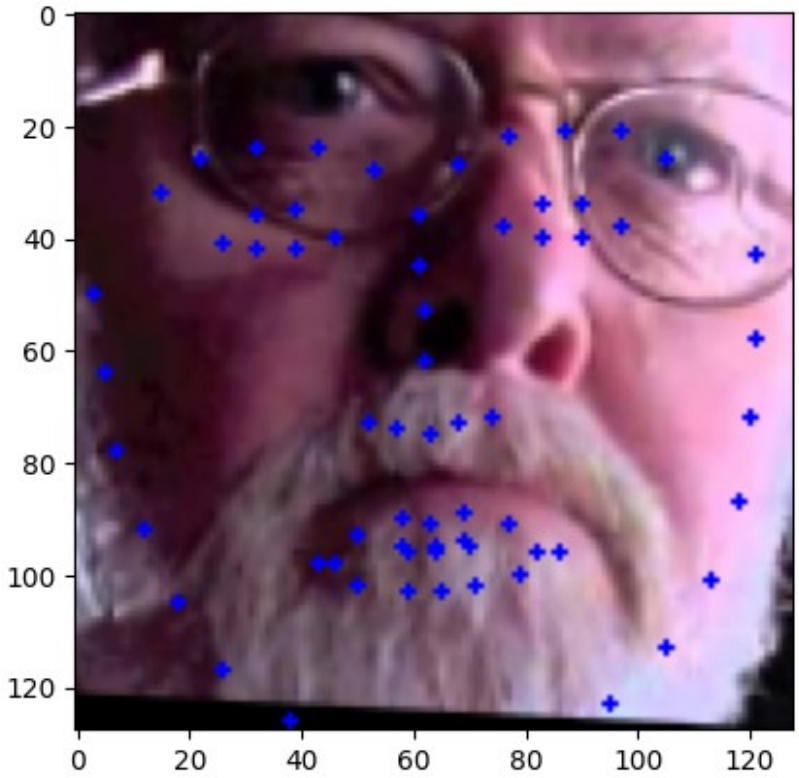

0
20
40
60
80
100
120
0
20
40
60
80
100
120

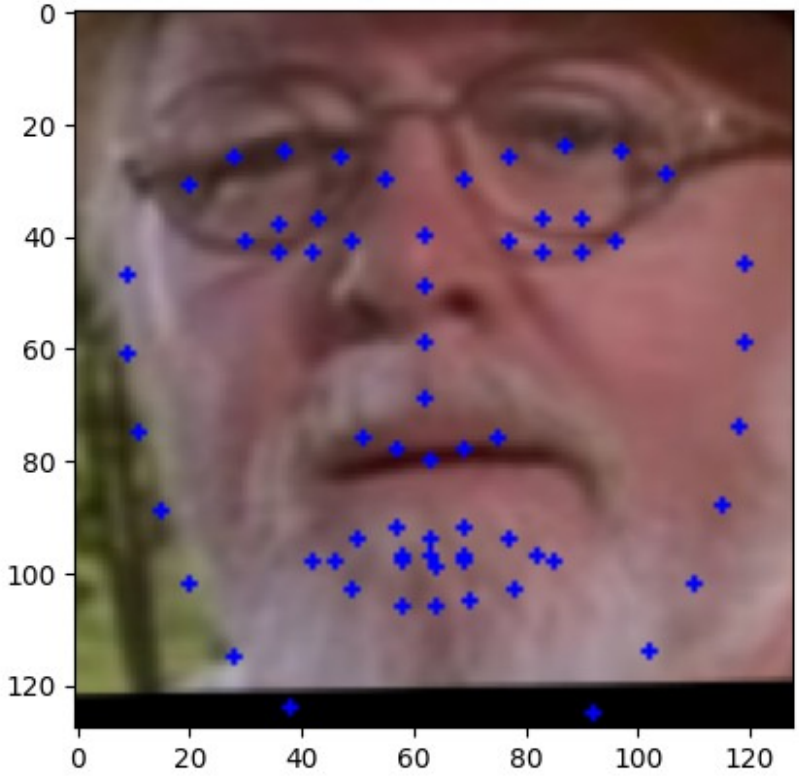

0
20
40
60
80
100
120
0
20
40
60
80
100
120

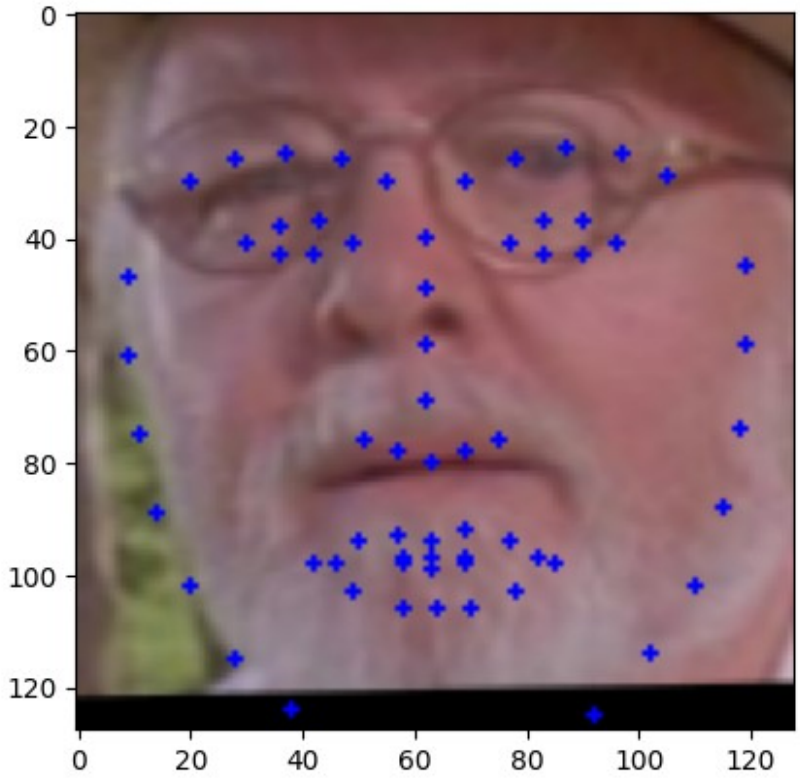

0
20
40
60
80
100
120
0
20
40
60
80
100
120

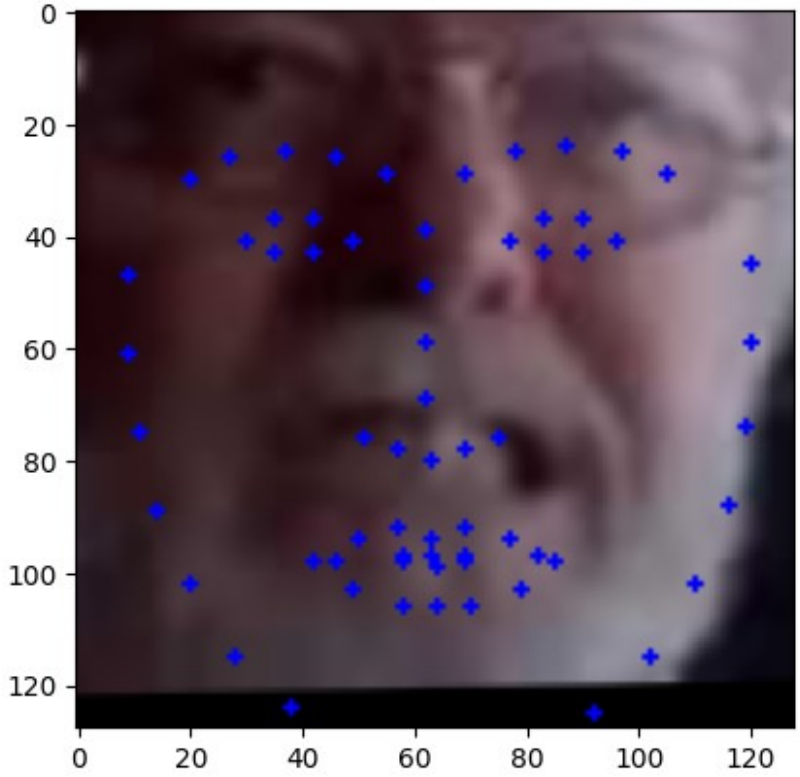

0
20
40
60
80
100
120
0
20
40
60
80
100
120

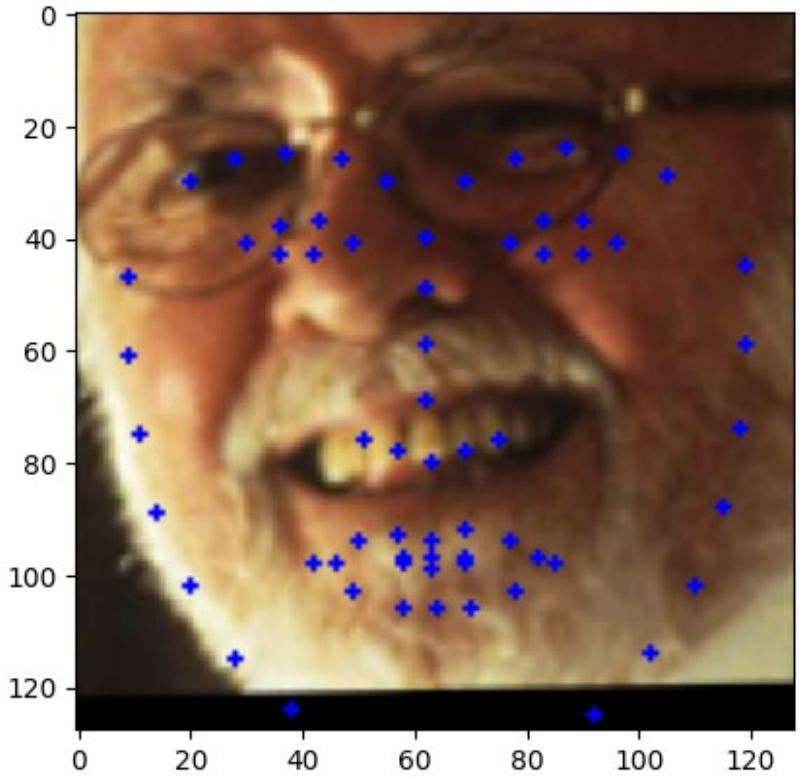

0
20
40
60
80
100
120
0
20
40
60
80
100
120

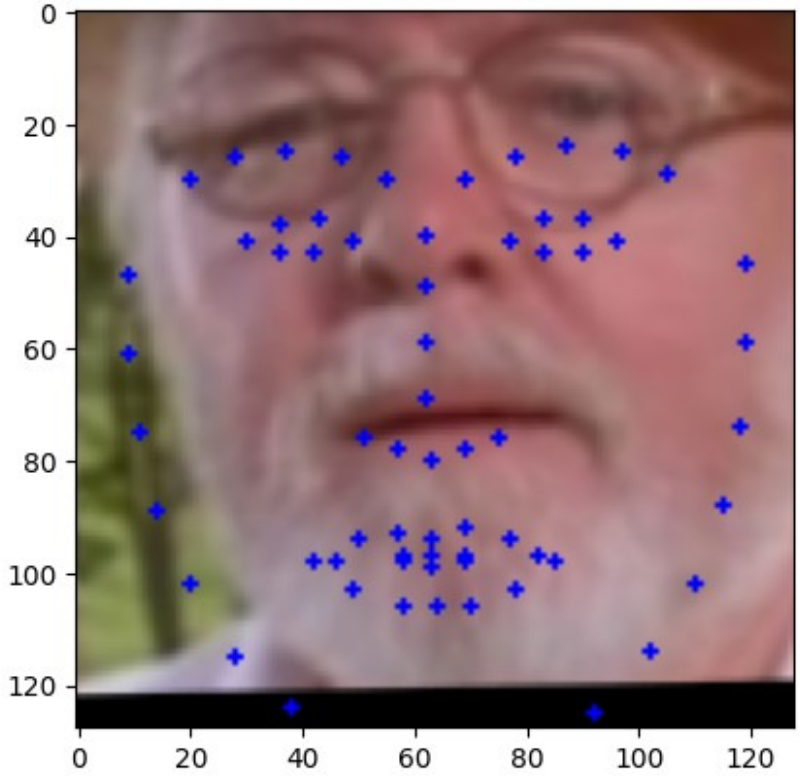

0
20
40
60
80
100
120
0
20
40
60
80
100
120

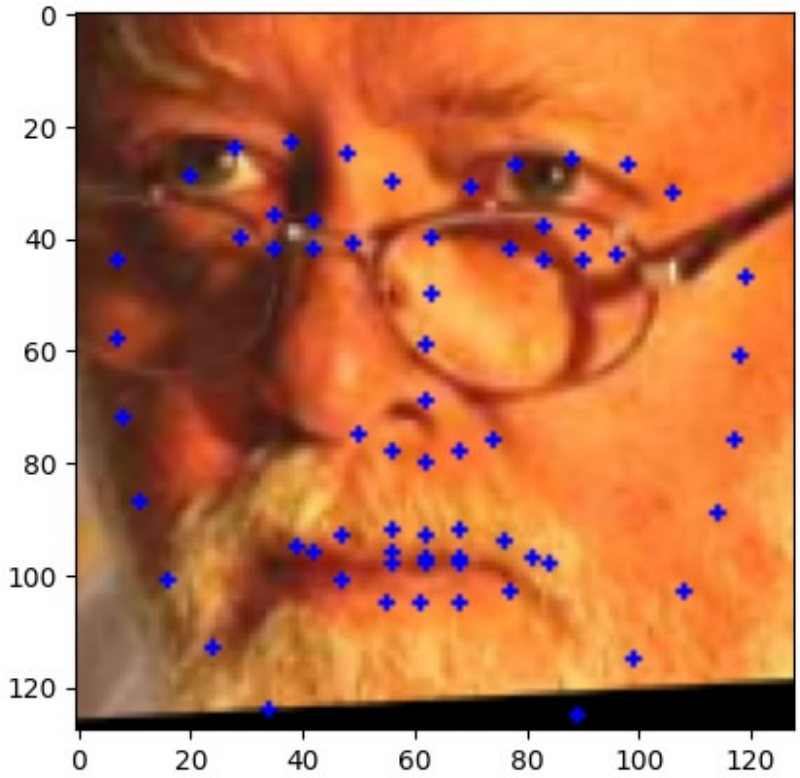

0
20
40
60
80
100
120
0
20
40
60
80
100
120

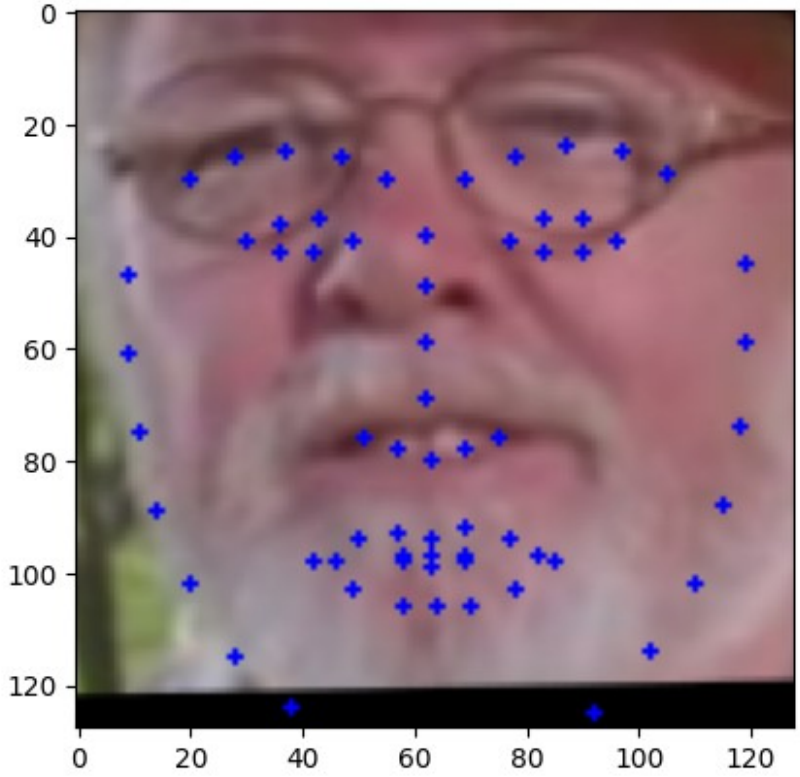

0
20
40
60
80
100
120
0
20
40
60
80
100
120

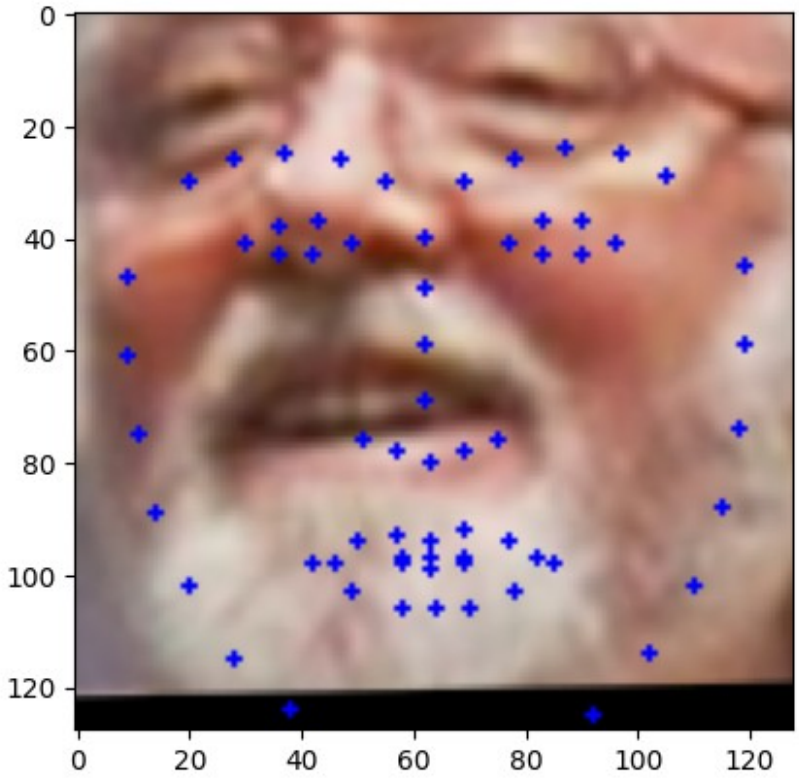

0
20
40
60
80
100
120
0
20
40
60
80
100
120

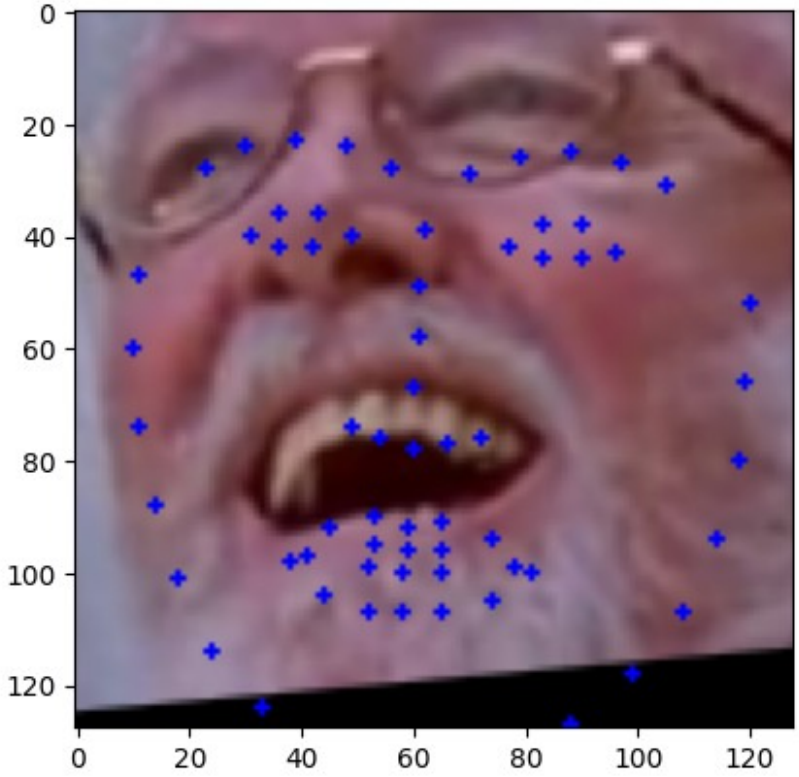

0
20
40
60
80
100
120
0
20
40
60
80
100
120

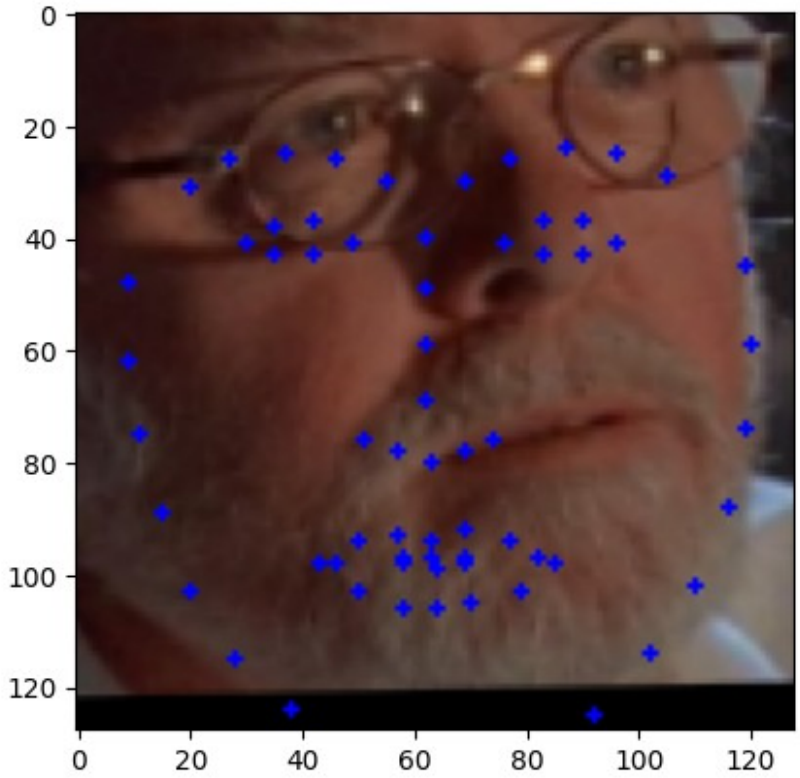

0
20
40
60
80
100
120
0
20
40
60
80
100
120

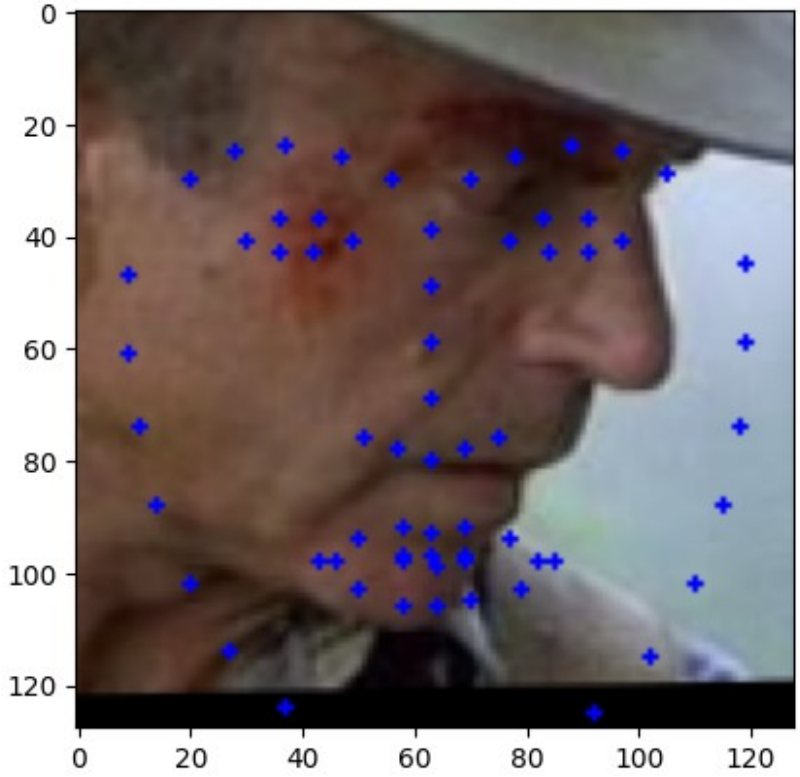

0
20
40
60
80
100
120
0
20
40
60
80
100
120

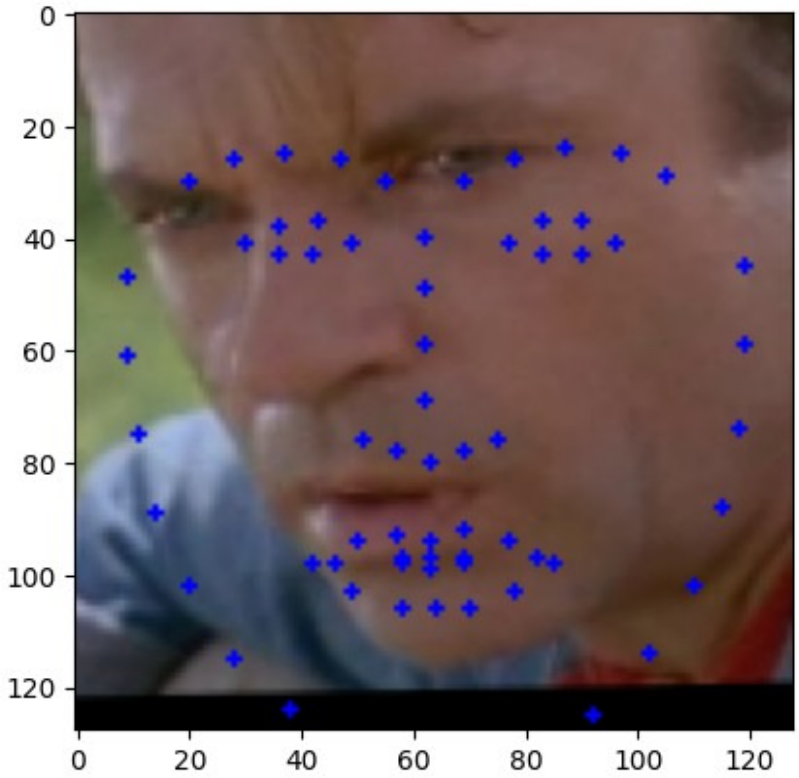

0
20
40
60
80
100
120
0
20
40
60
80
100
120

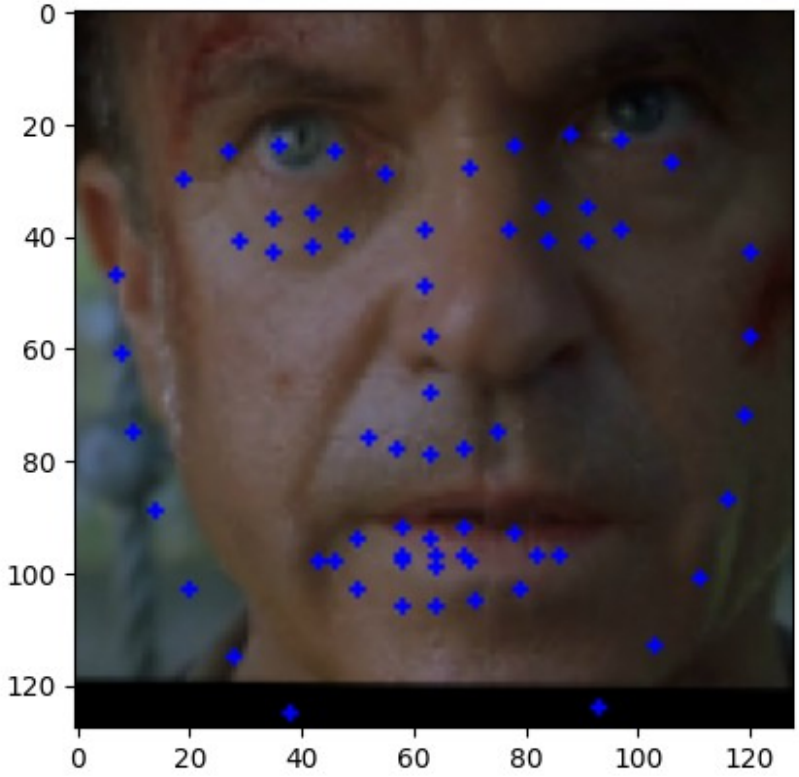

0
20
40
60
80
100
120
0
20
40
60
80
100
120

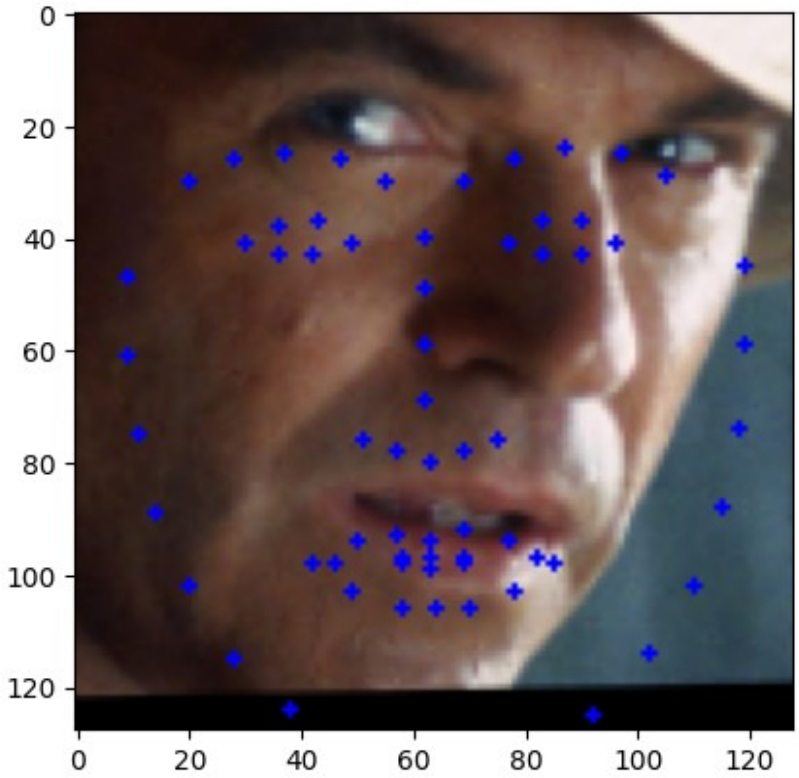

0
20
40
60
80
100
120
0
20
40
60
80
100
120

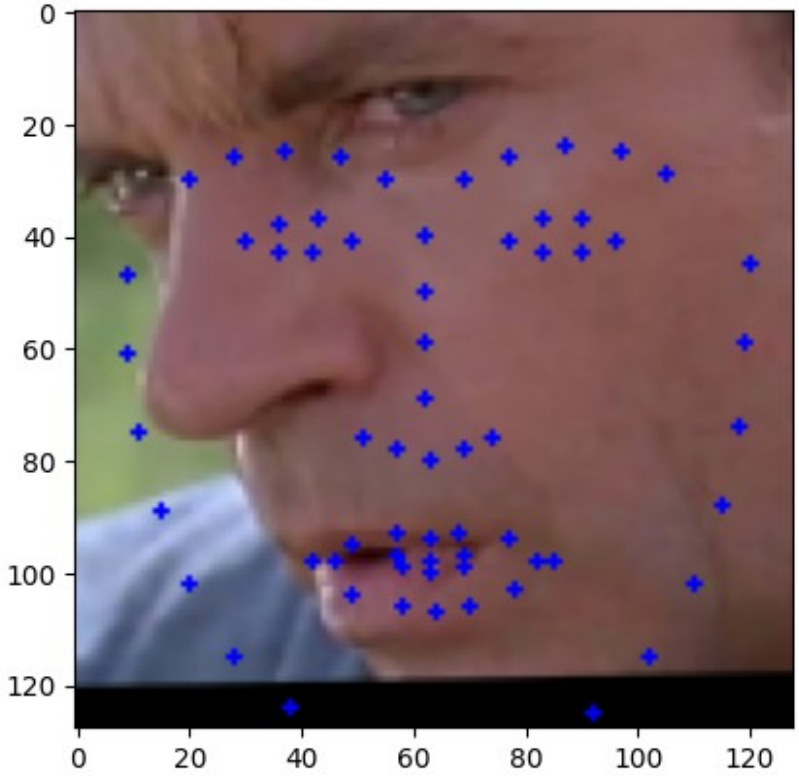

0
20
40
60
80
100
120
0
20
40
60
80
100
120

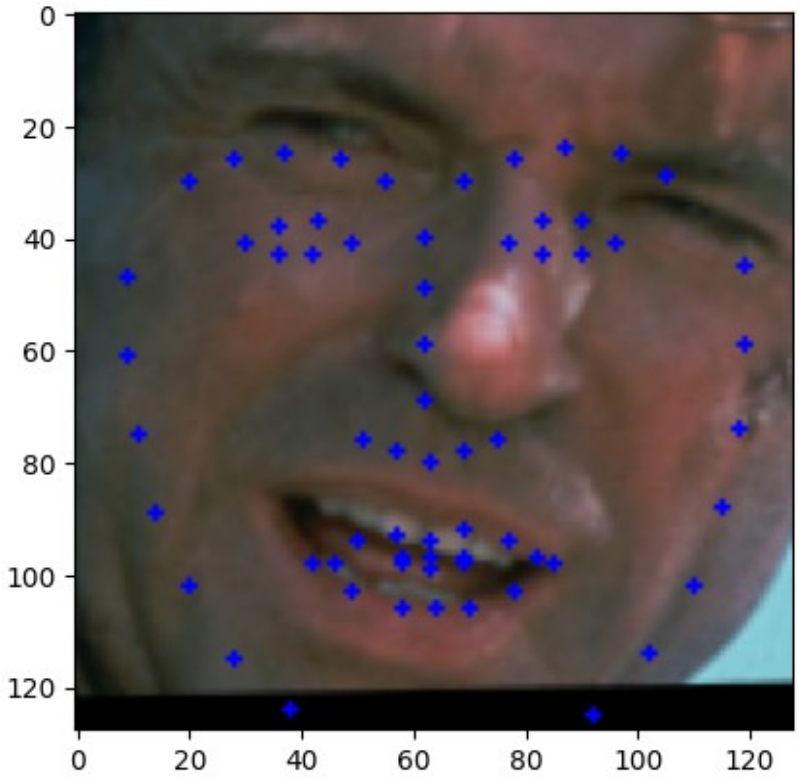

0
20
40
60
80
100
120
0
20
40
60
80
100
120

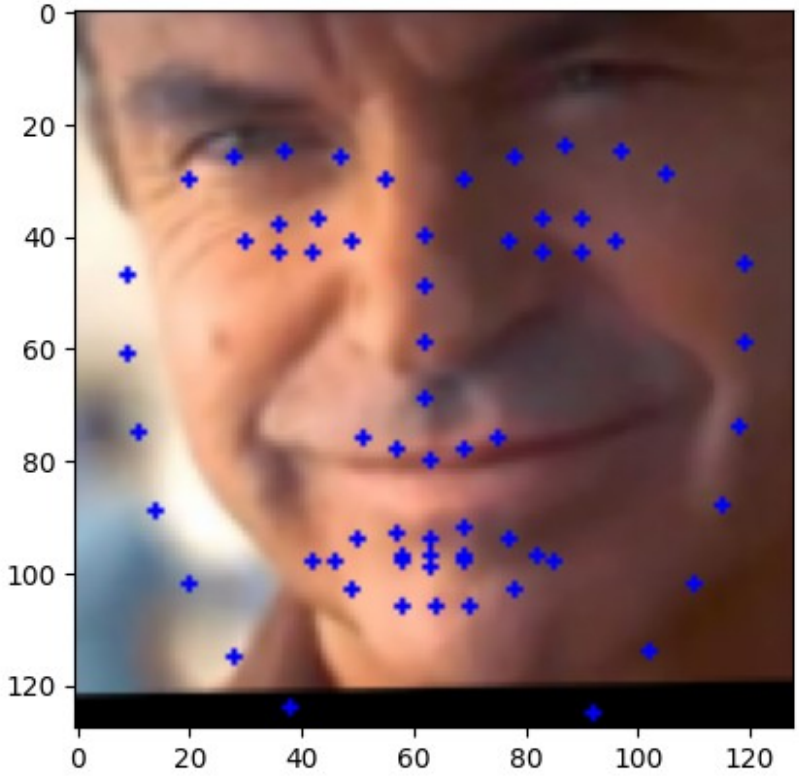

0
20
40
60
80
100
120
0
20
40
60
80
100
120

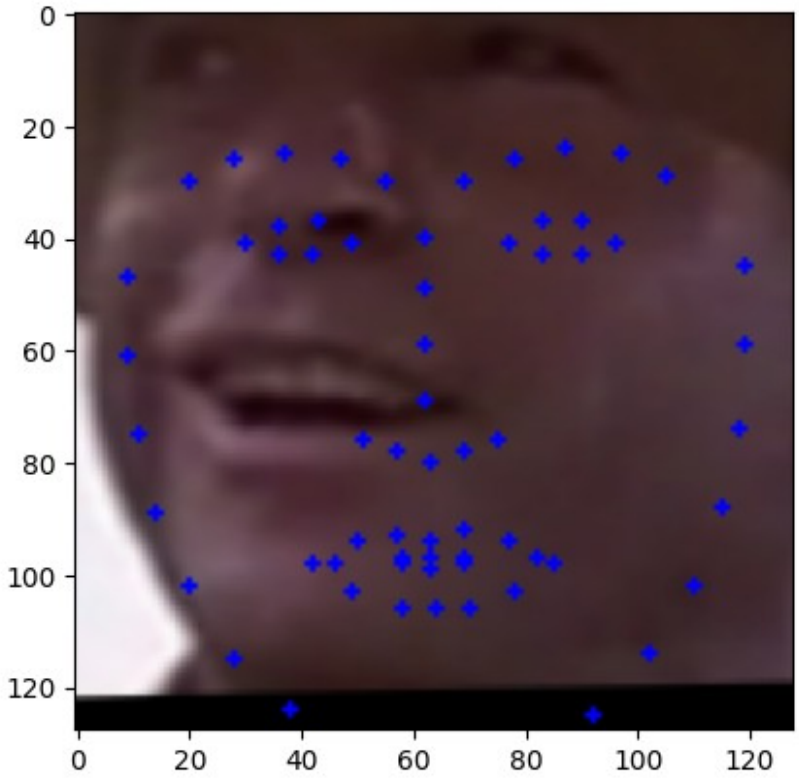

0
20
40
60
80
100
120
0
20
40
60
80
100
120

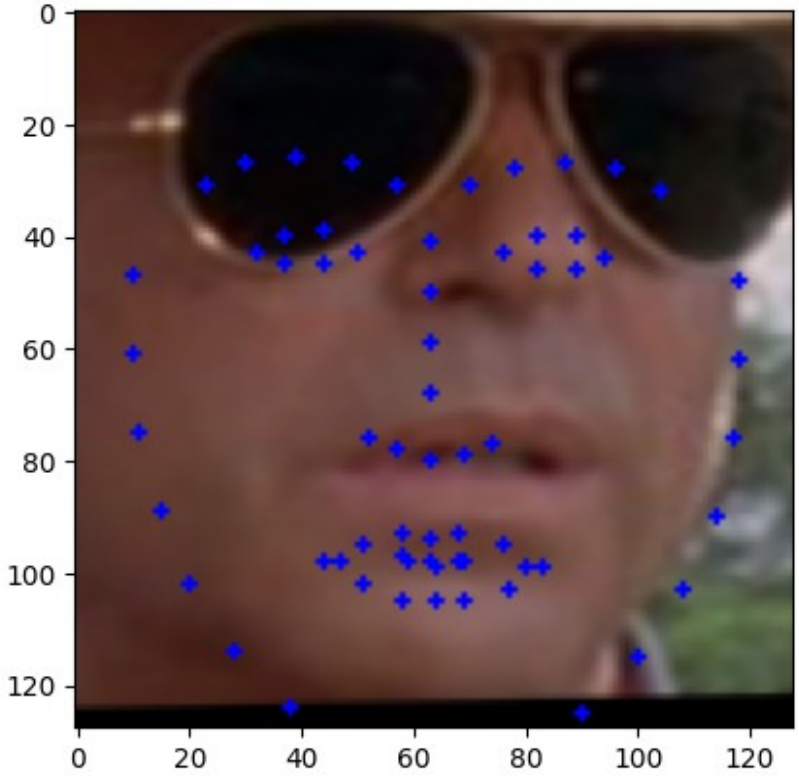

0
20
40
60
80
100
120
0
20
40
60
80
100
120

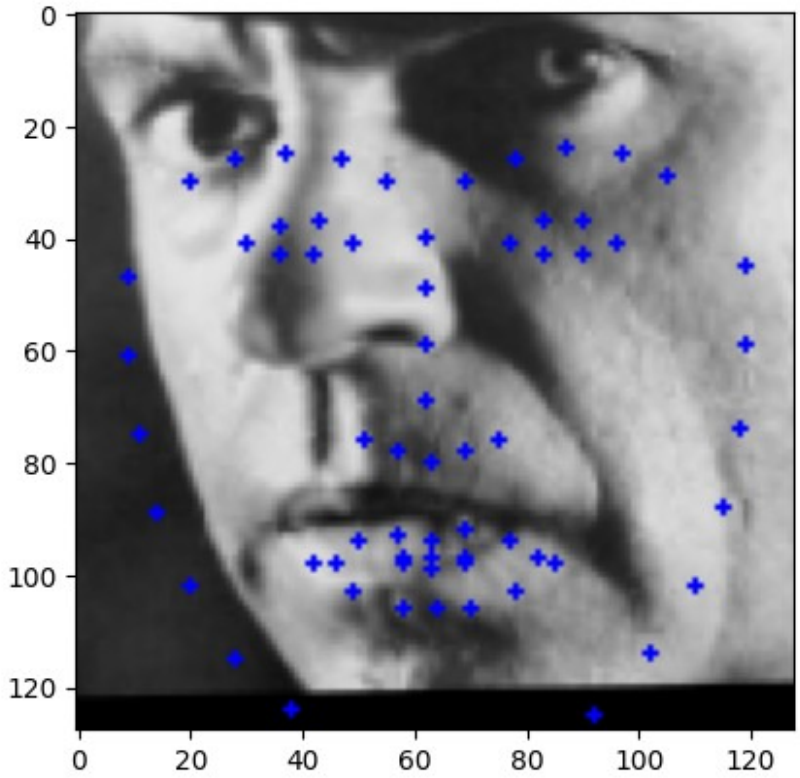

0
20
40
60
80
100
120
0
20
40
60
80
100
120

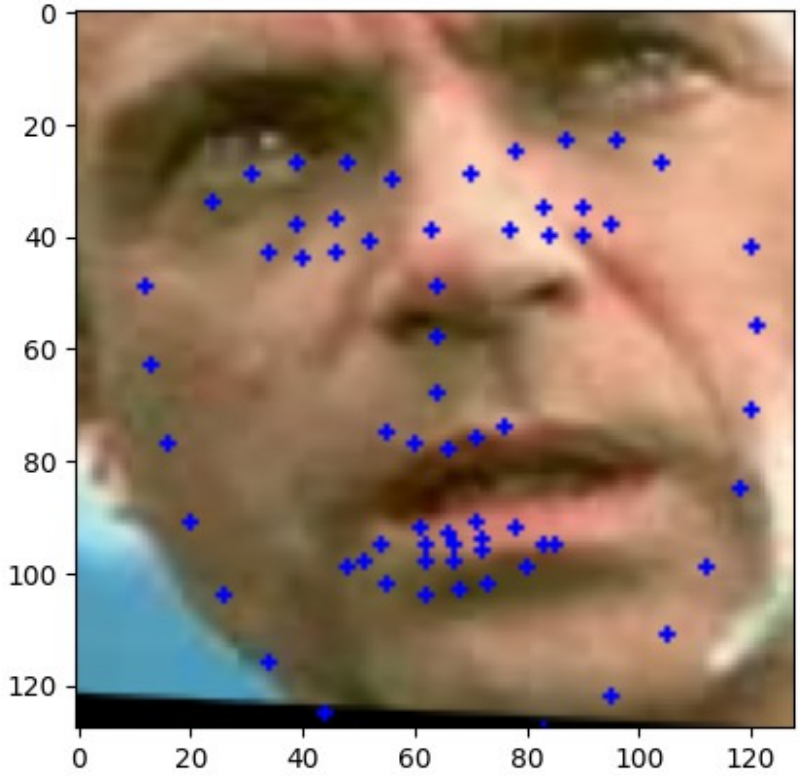

0
20
40
60
80
100
120
0
20
40
60
80
100
120

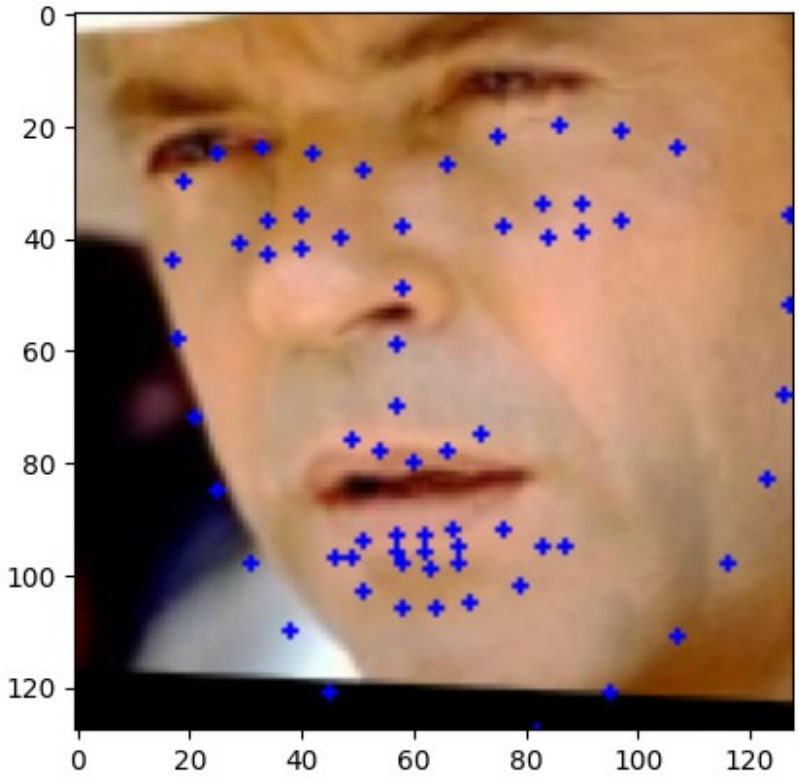

0
20
40
60
80
100
120
0
20
40
60
80
100
120

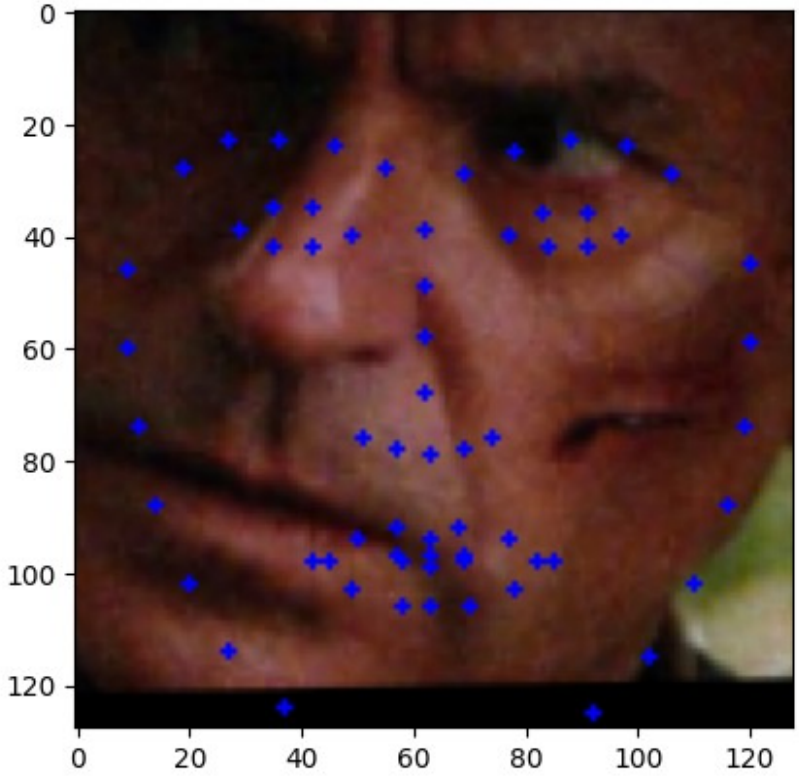

0
20
40
60
80
100
120
0
20
40
60
80
100
120

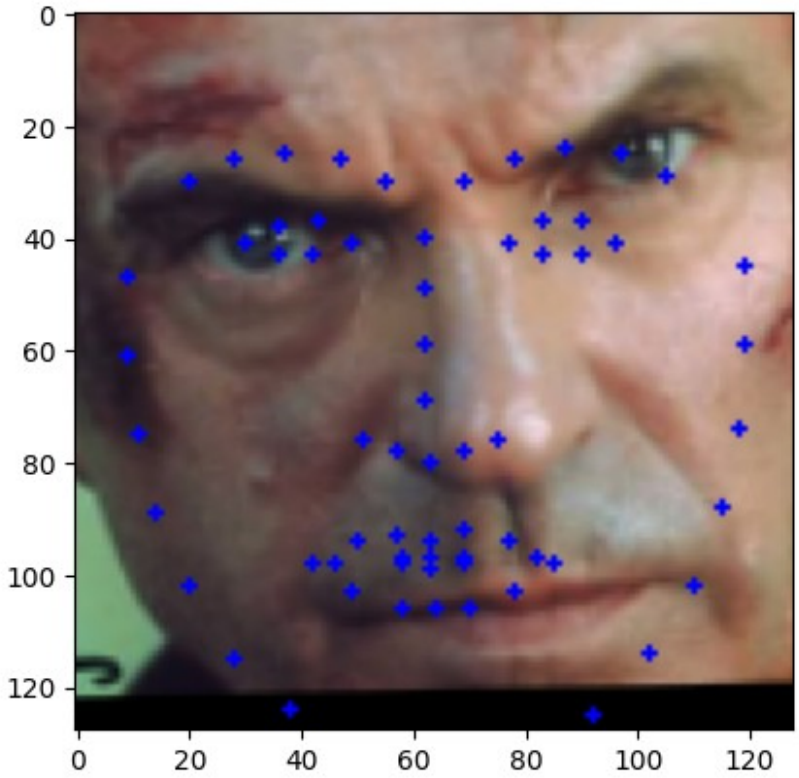

0
20
40
60
80
100
120
0
20
40
60
80
100
120

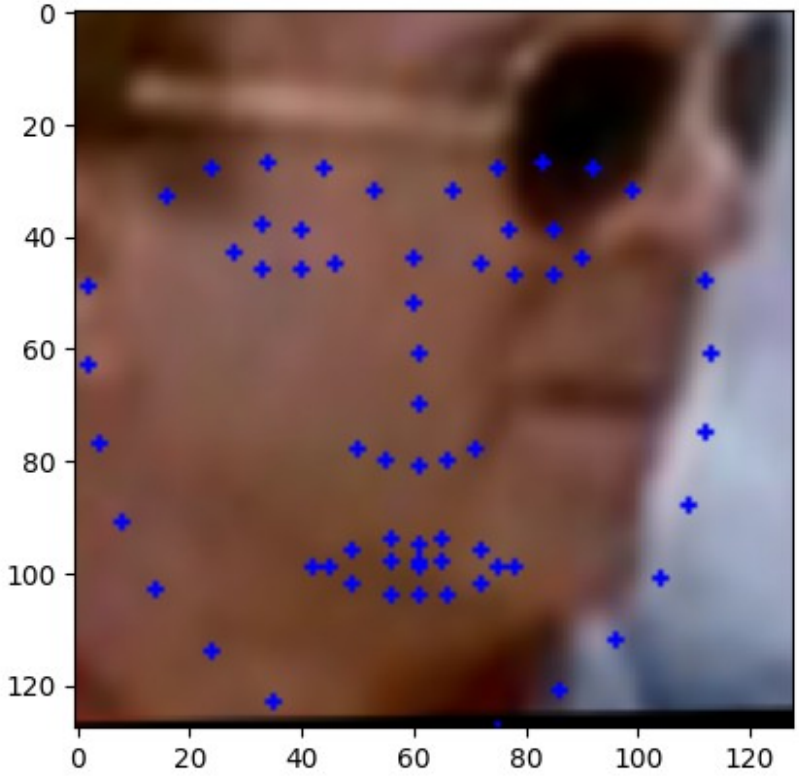

0
20
40
60
80
100
120
0
20
40
60
80
100
120

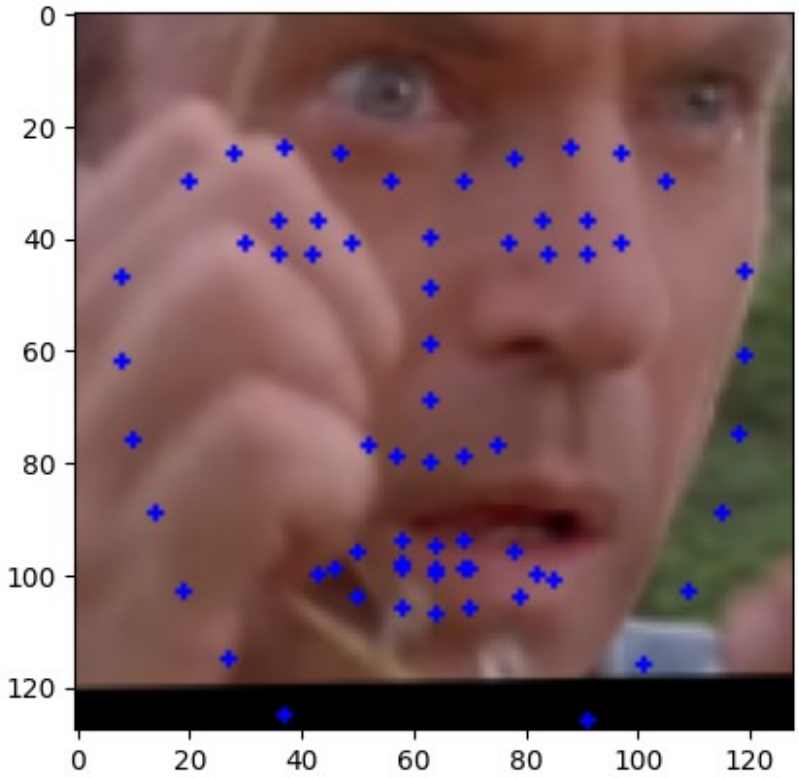

0
20
40
60
80
100
120
0
20
40
60
80
100
120

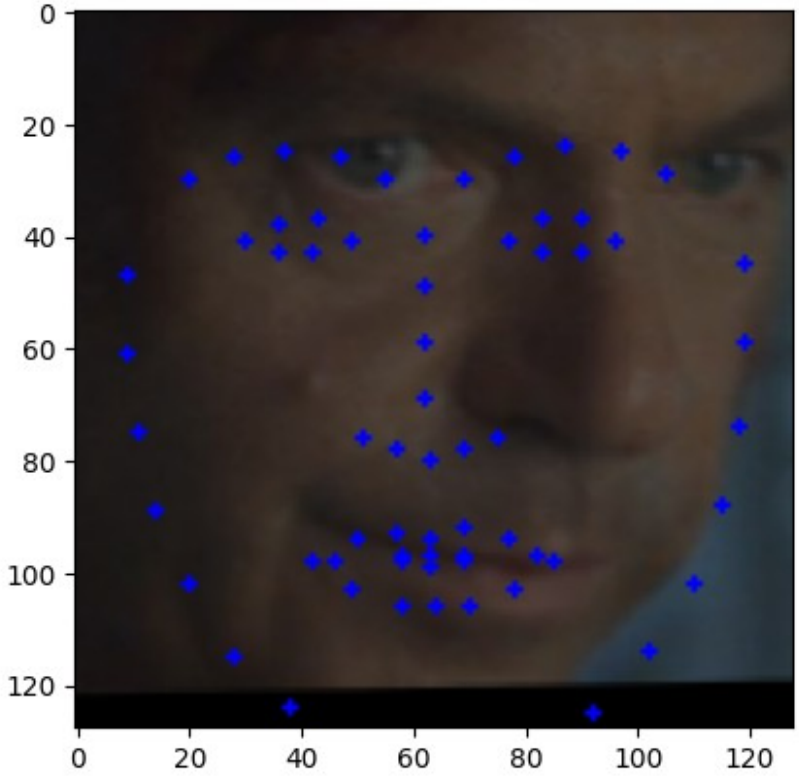

0
20
40
60
80
100
120
0
20
40
60
80
100
120

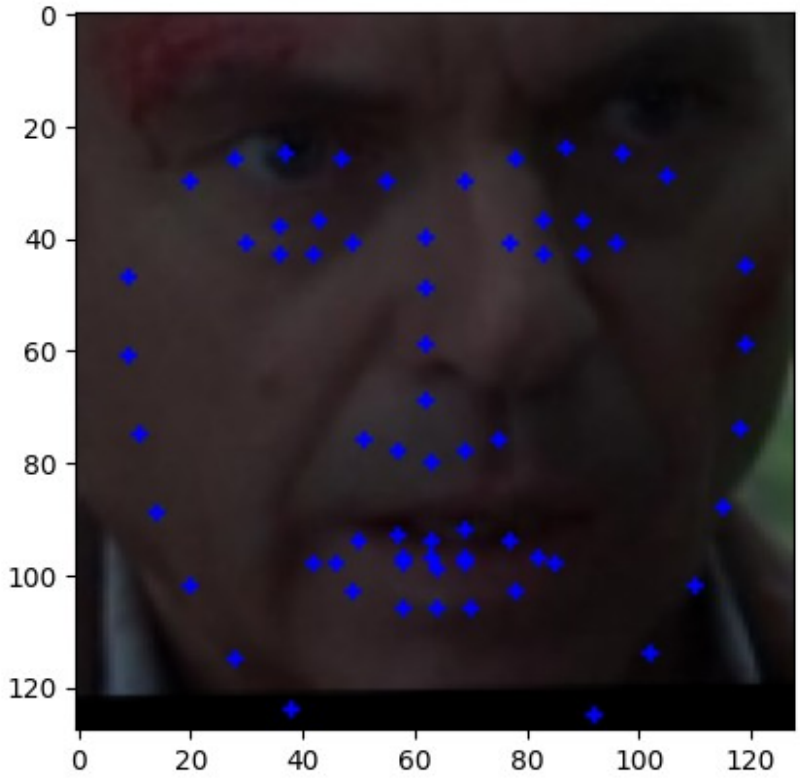

0
20
40
60
80
100
120
0
20
40
60
80
100
120

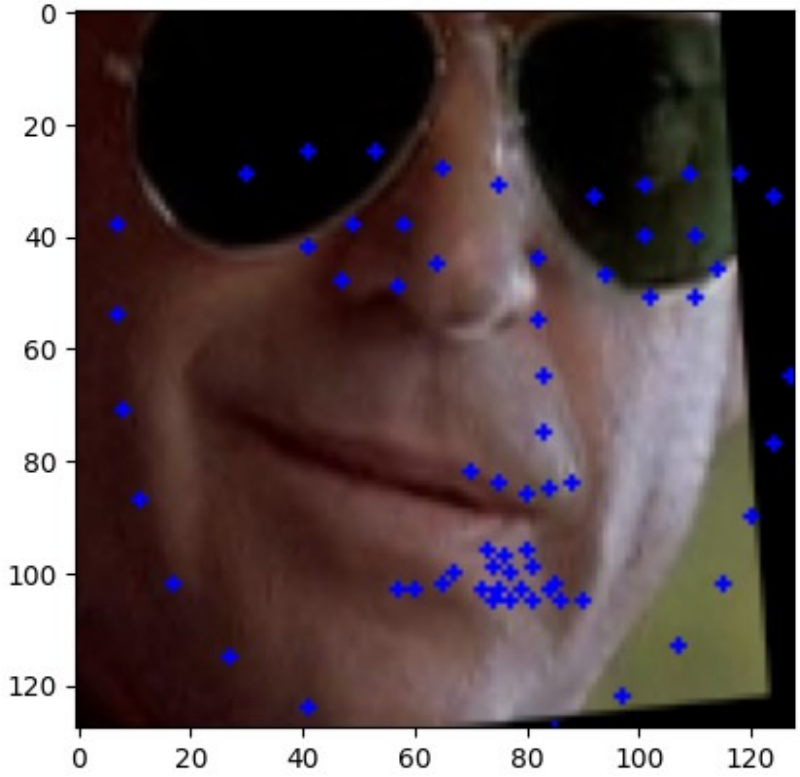

0
20
40
60
80
100
120
0
20
40
60
80
100
120

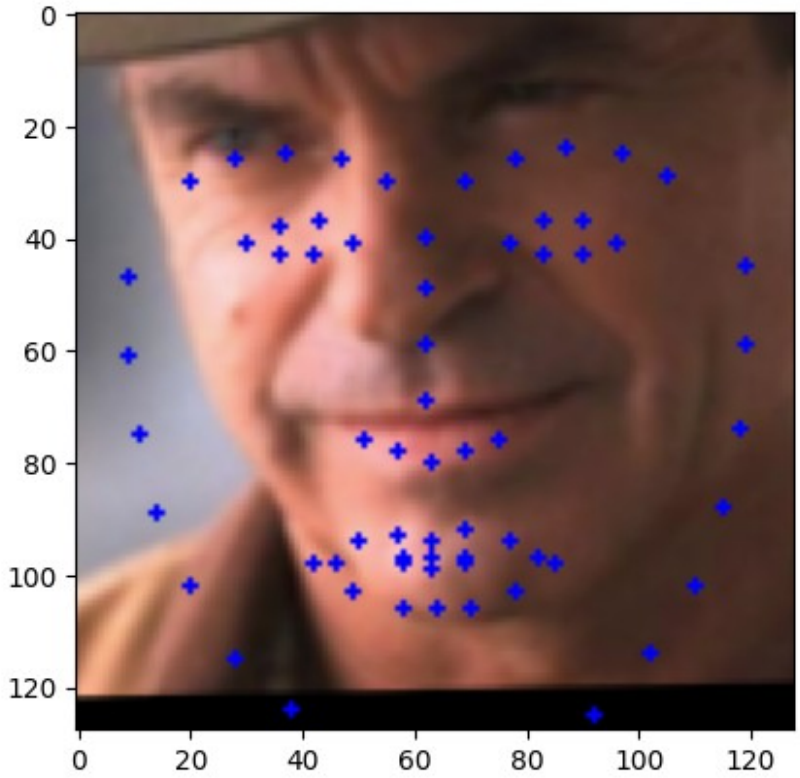

0
20
40
60
80
100
120
0
20
40
60
80
100
120

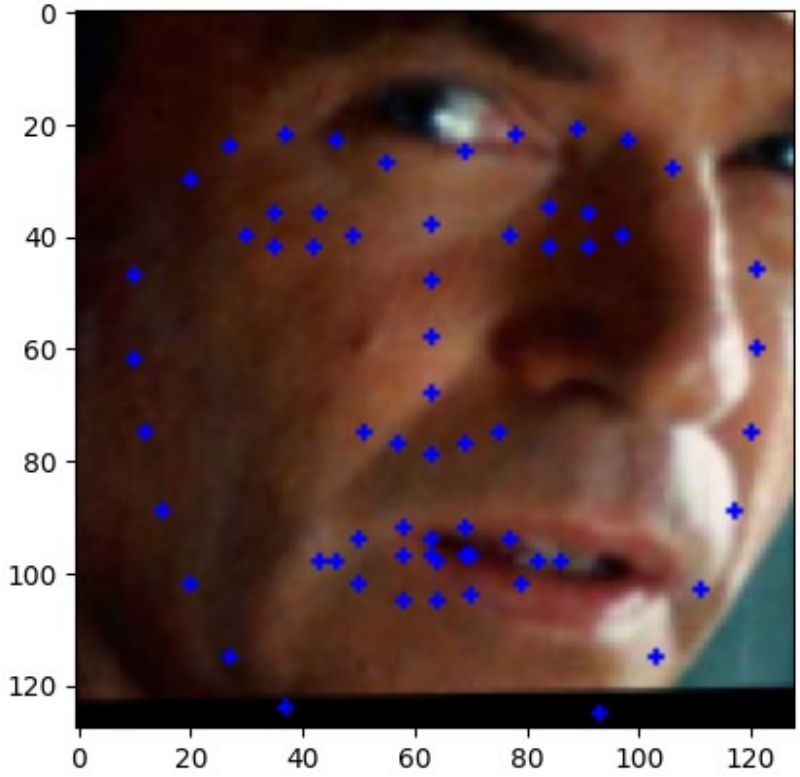

0
20
40
60
80
100
120
0
20
40
60
80
100
120

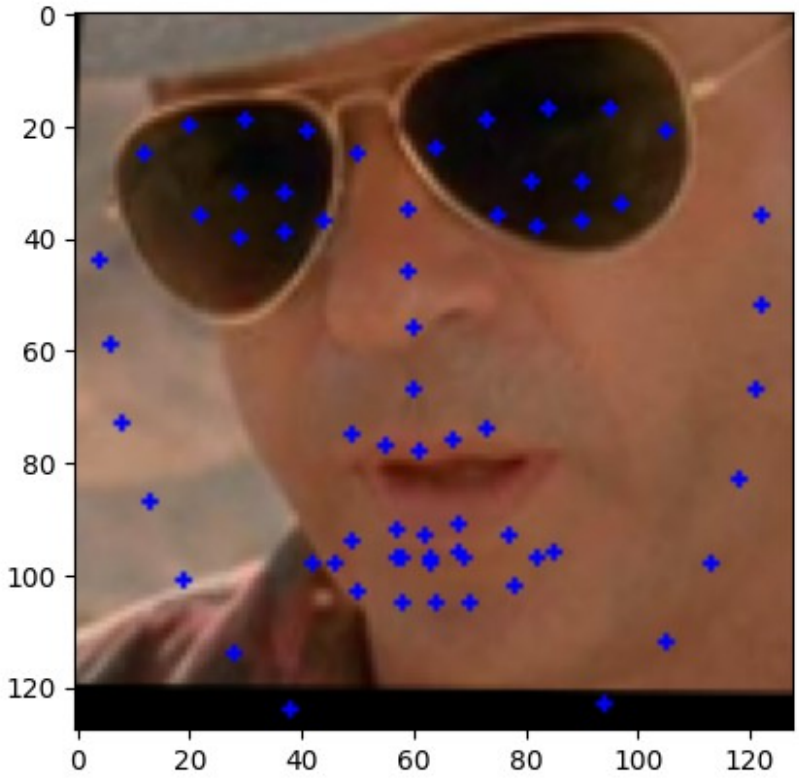

0
20
40
60
80
100
120
0
20
40
60
80
100
120

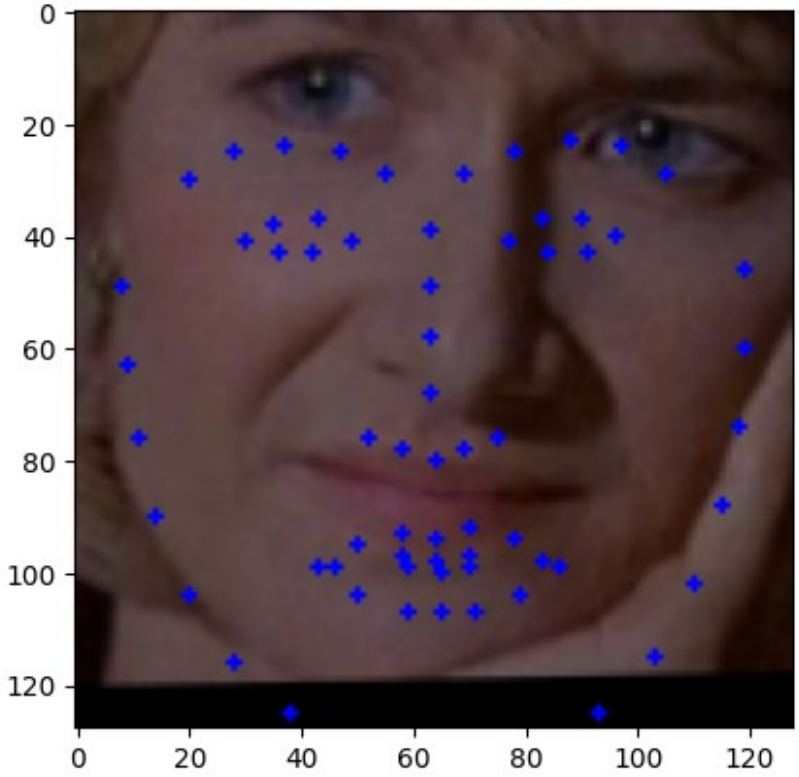

0
20
40
60
80
100
120
0
20
40
60
80
100
120

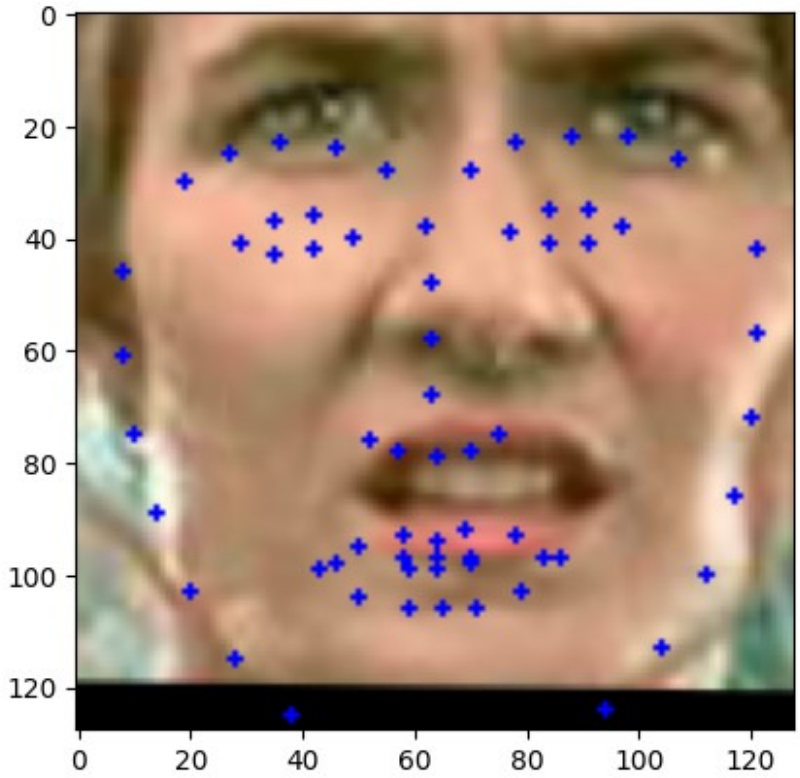

0
20
40
60
80
100
120
0
20
40
60
80
100
120

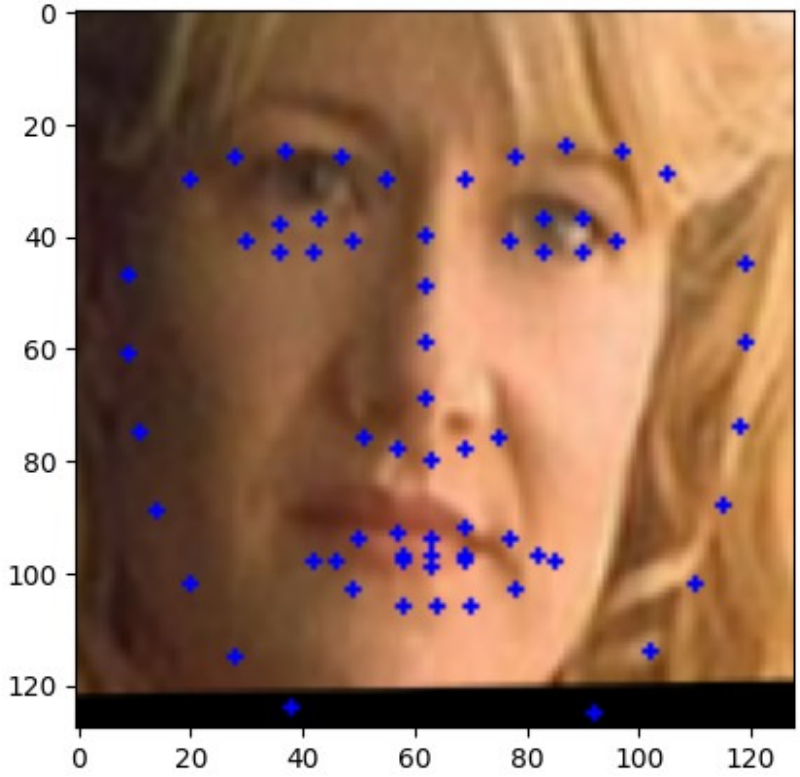

0
20
40
60
80
100
120
0
20
40
60
80
100
120

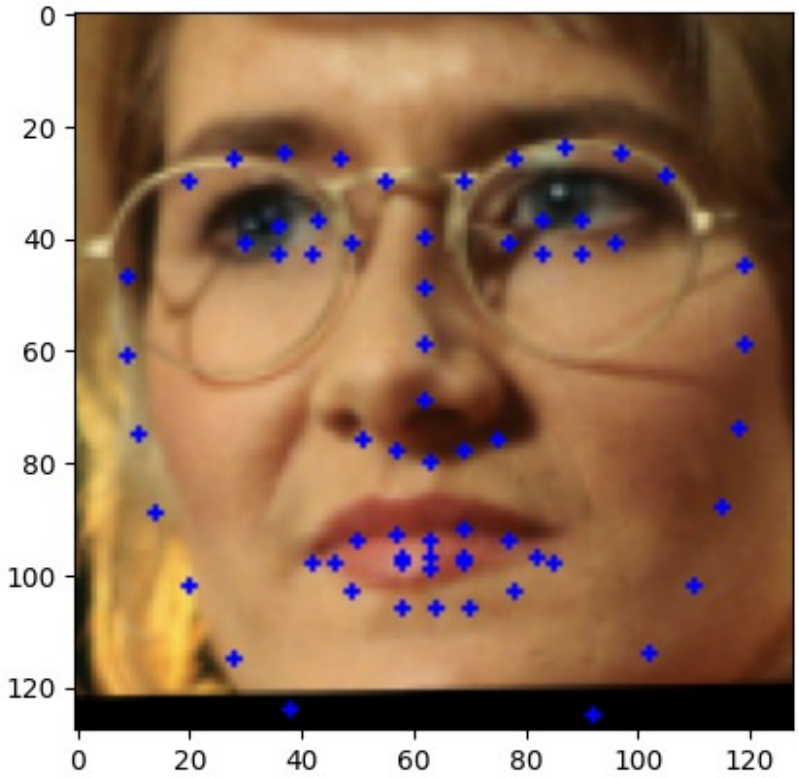

0
20
40
60
80
100
120
0
20
40
60
80
100
120

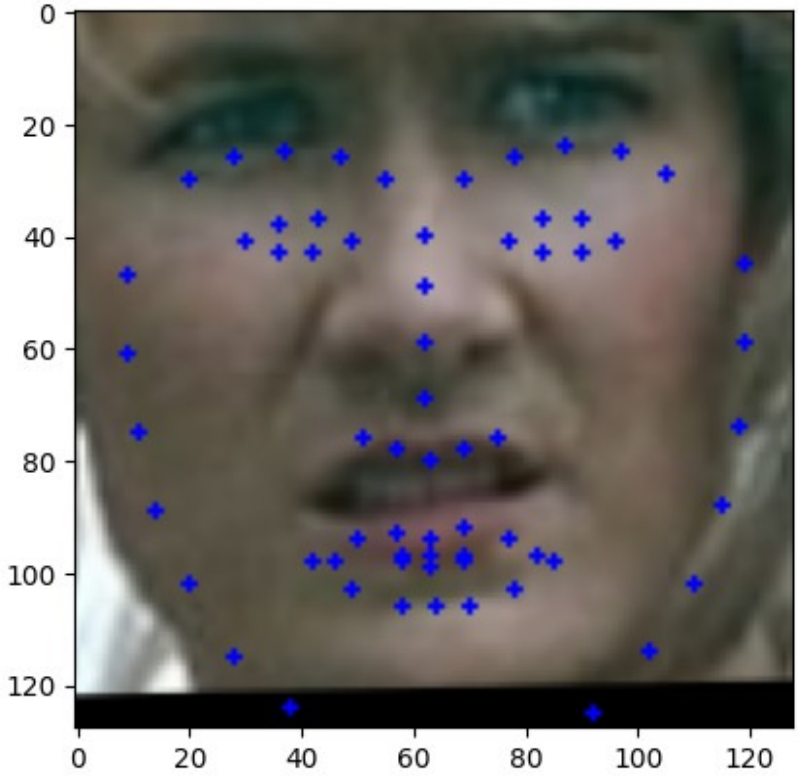

0
20
40
60
80
100
120
0
20
40
60
80
100
120

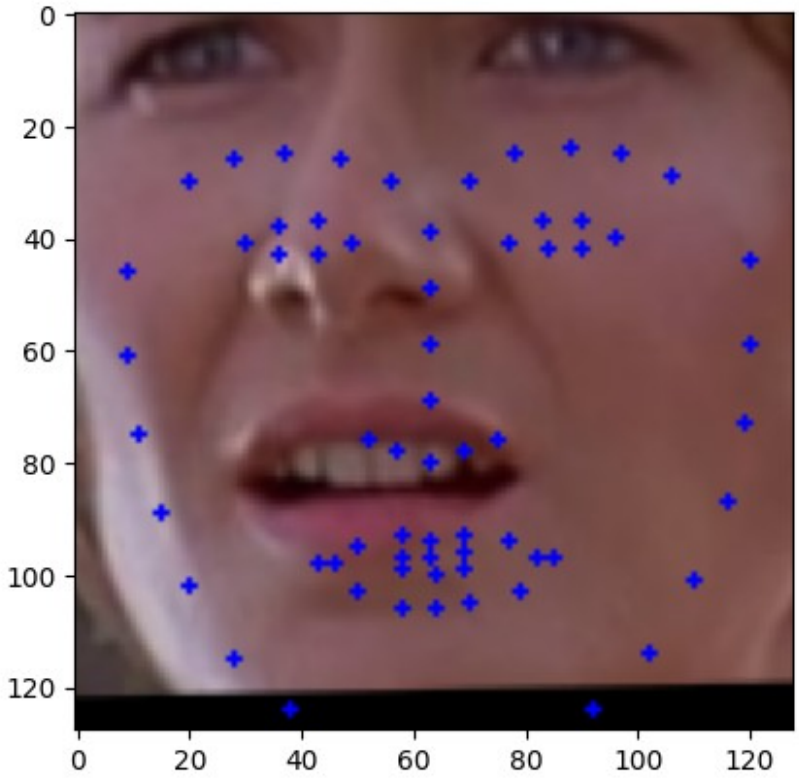

0
20
40
60
80
100
120
0
20
40
60
80
100
120

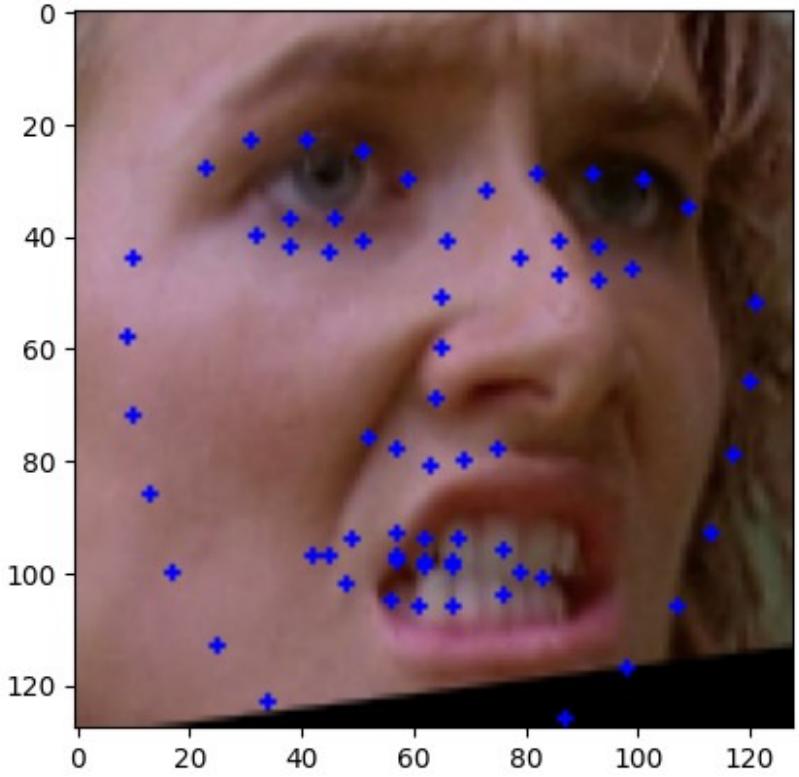

0
20
40
60
80
100
120
0
20
40
60
80
100
120

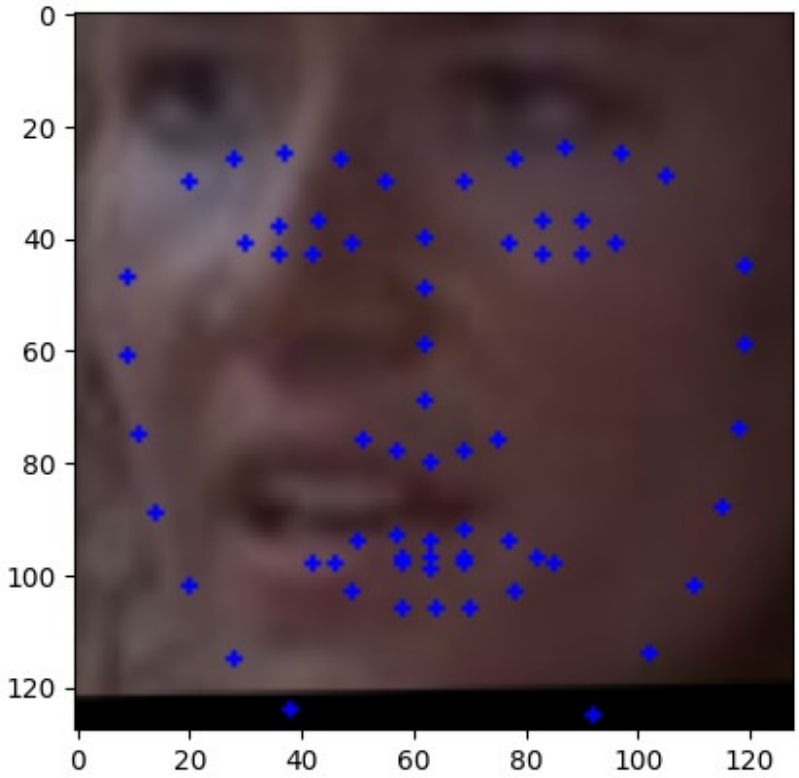

0
20
40
60
80
100
120
0
20
40
60
80
100
120

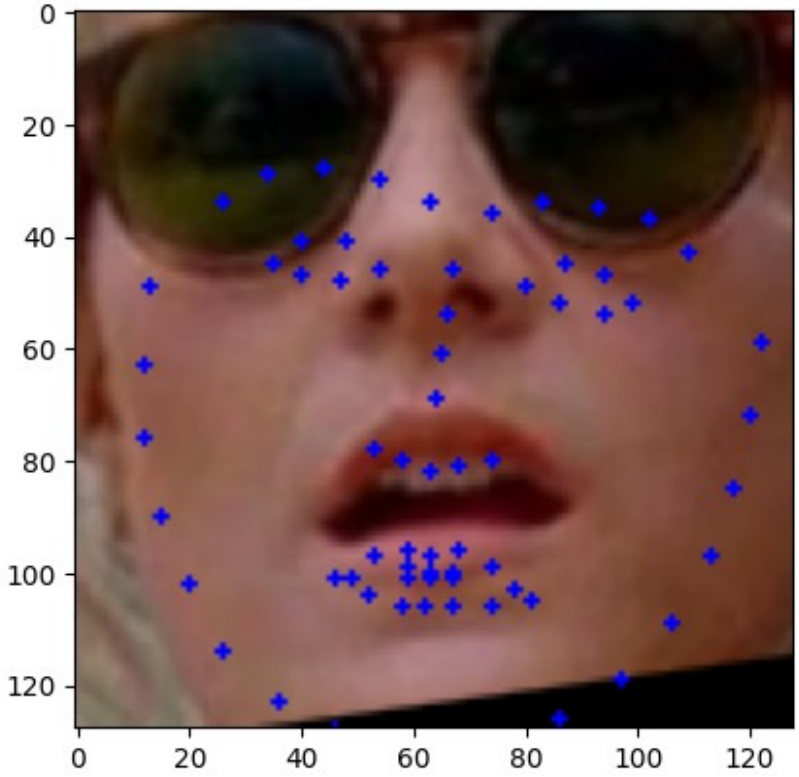

0
20
40
60
80
100
120
0
20
40
60
80
100
120

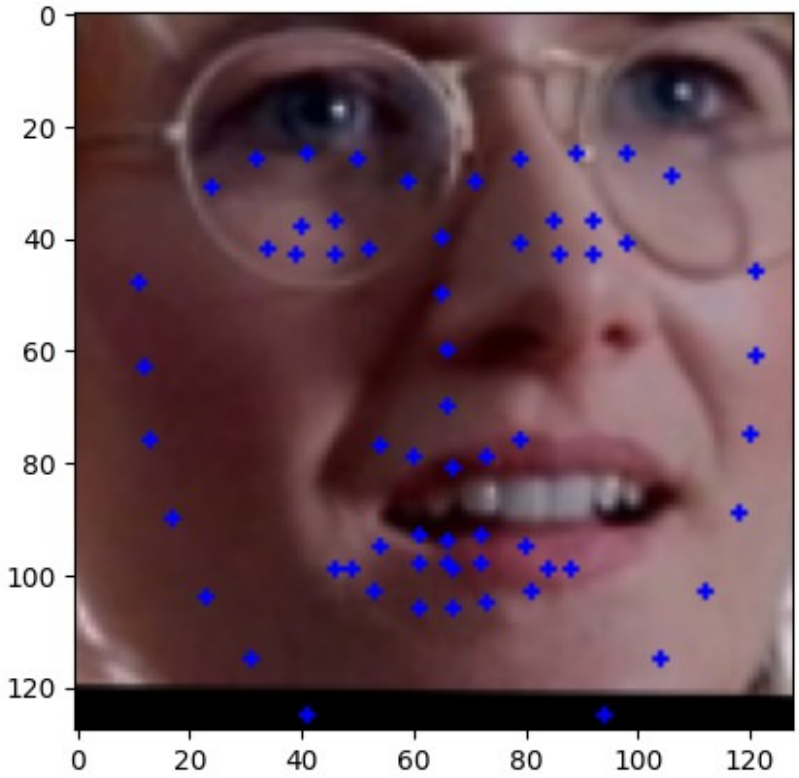

0
20
40
60
80
100
120
0
20
40
60
80
100
120

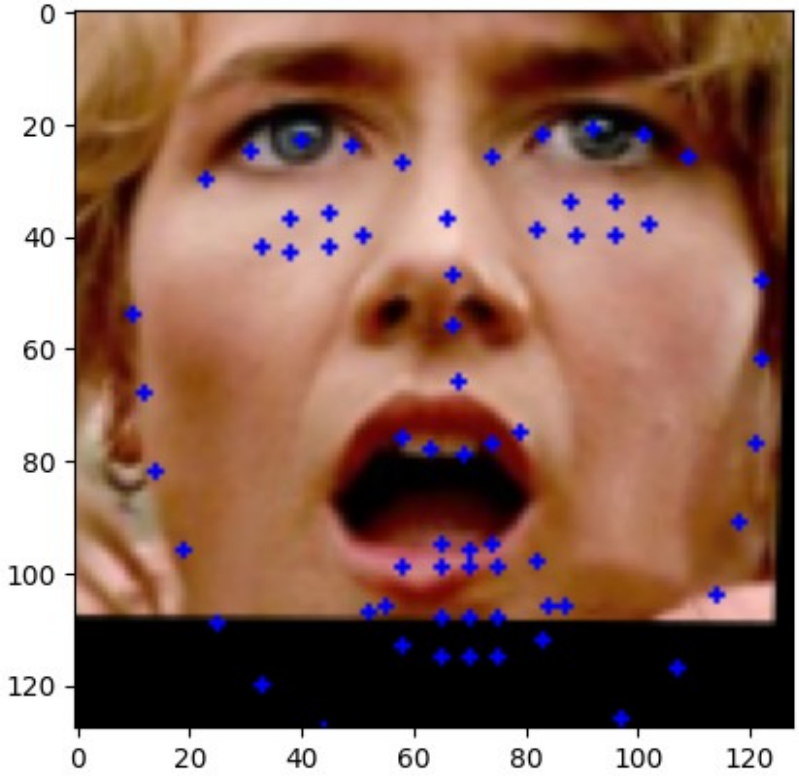

0
20
40
60
80
100
120
0
20
40
60
80
100
120

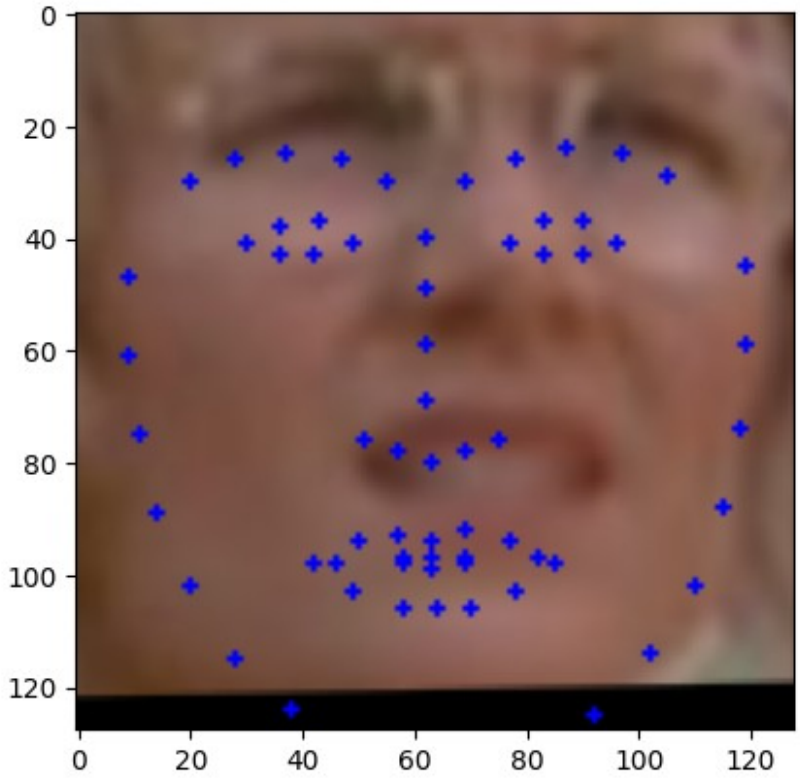

0
20
40
60
80
100
120
0
20
40
60
80
100
120

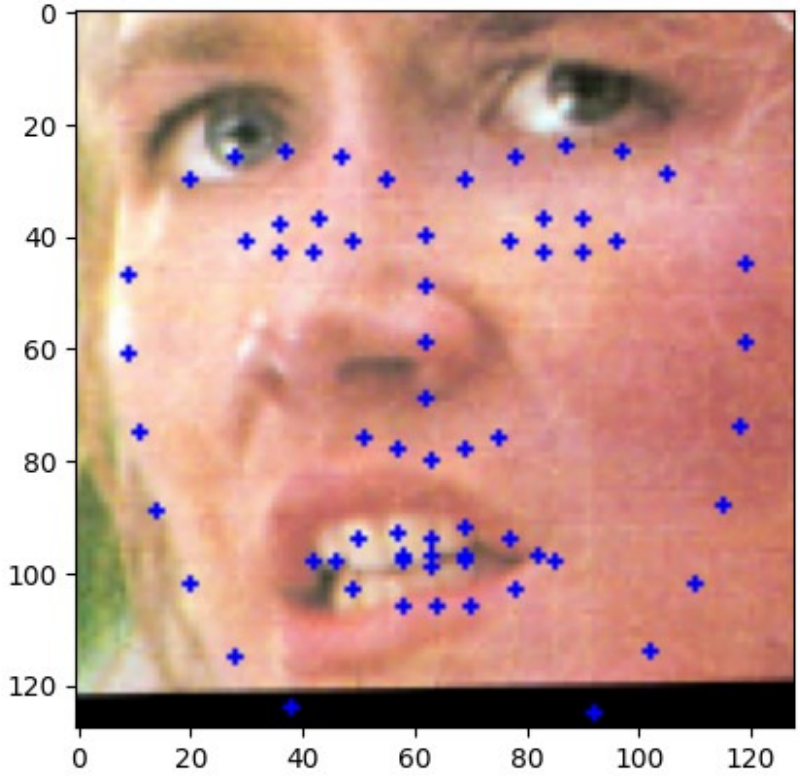

0
20
40
60
80
100
120
0
20
40
60
80
100
120

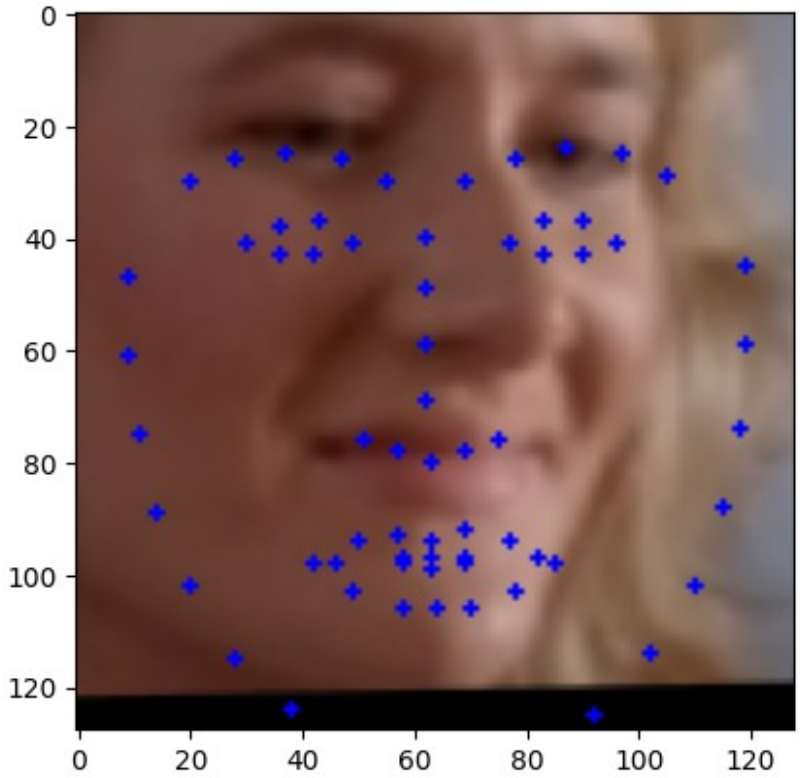

0
20
40
60
80
100
120
0
20
40
60
80
100
120

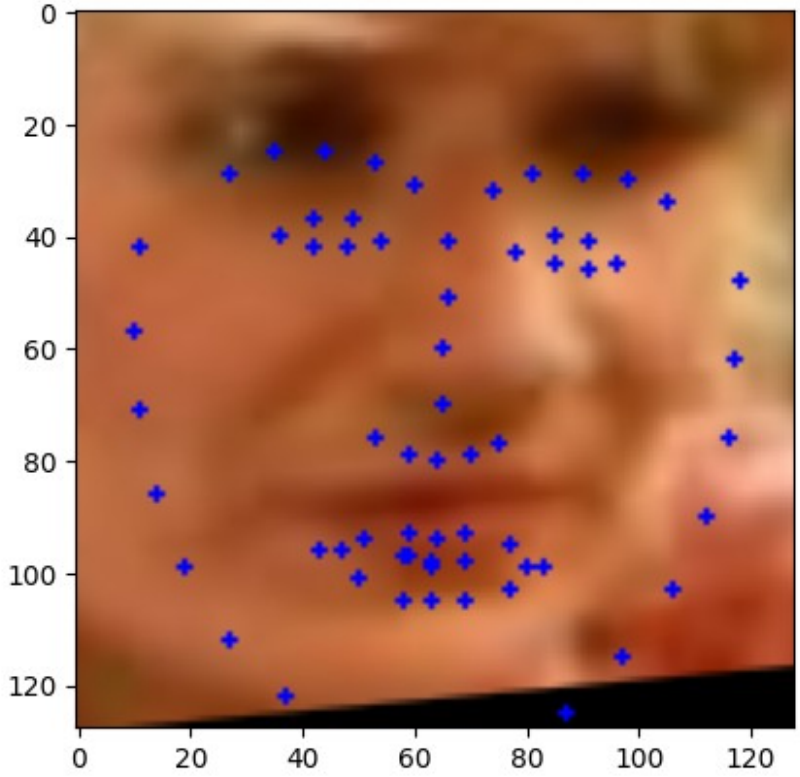

0
20
40
60
80
100
120
0
20
40
60
80
100
120

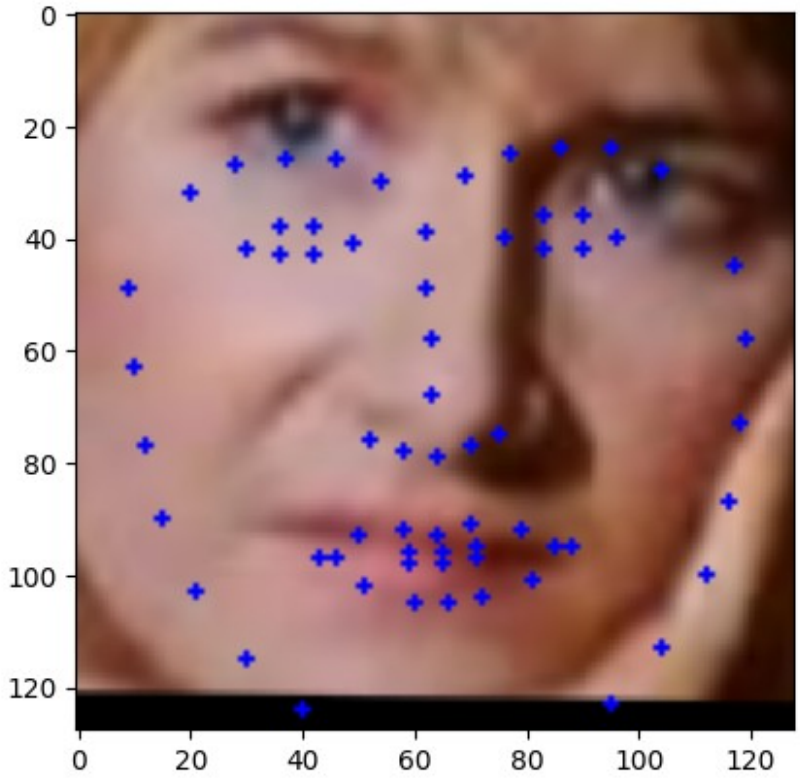

0
20
40
60
80
100
120
0
20
40
60
80
100
120

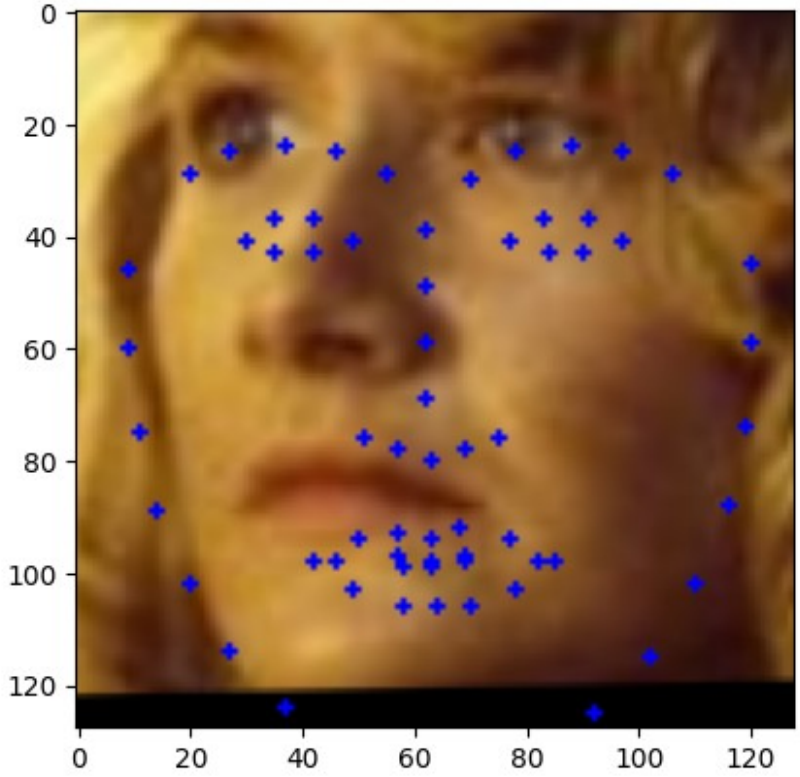

0
20
40
60
80
100
120
0
20
40
60
80
100
120

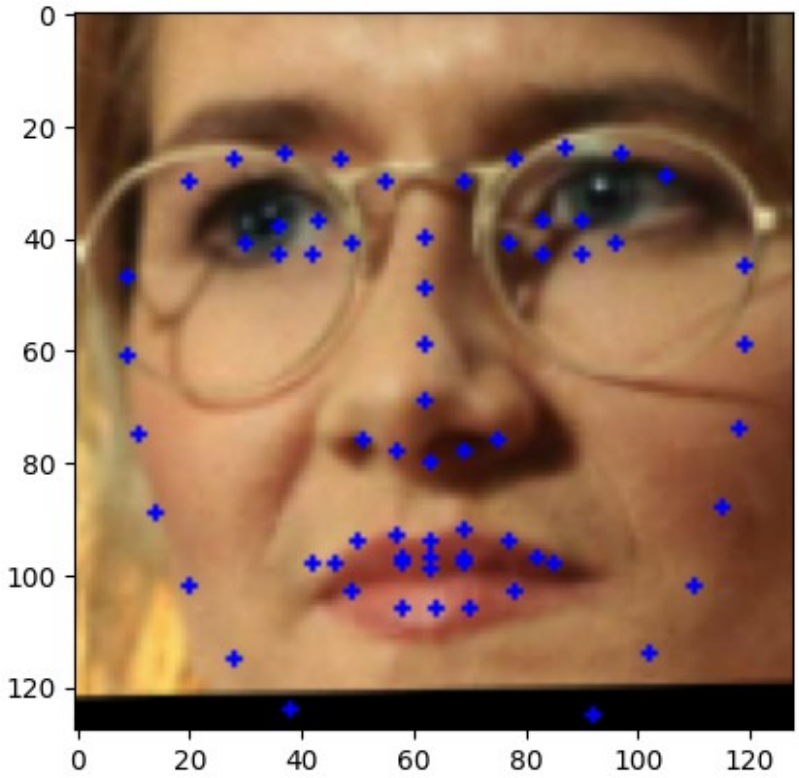

0
20
40
60
80
100
120
0
20
40
60
80
100
120

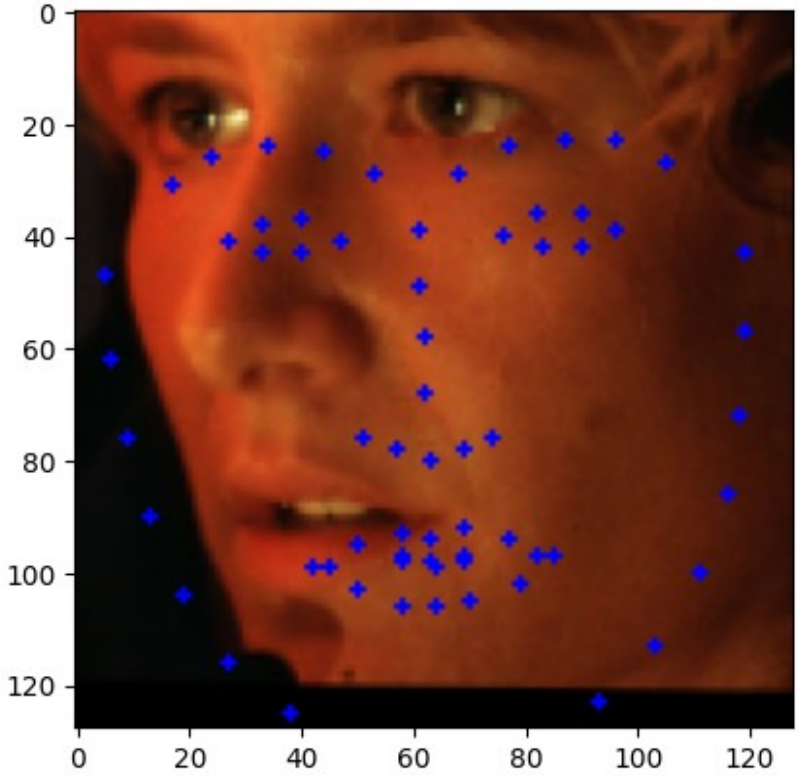

0
20
40
60
80
100
120
0
20
40
60
80
100
120

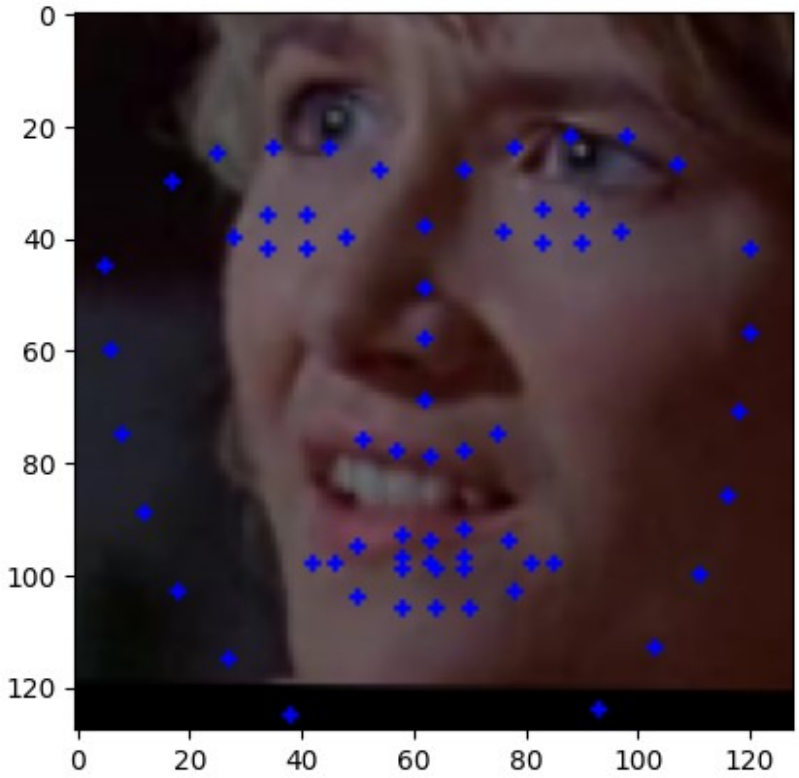

0
20
40
60
80
100
120
0
20
40
60
80
100
120

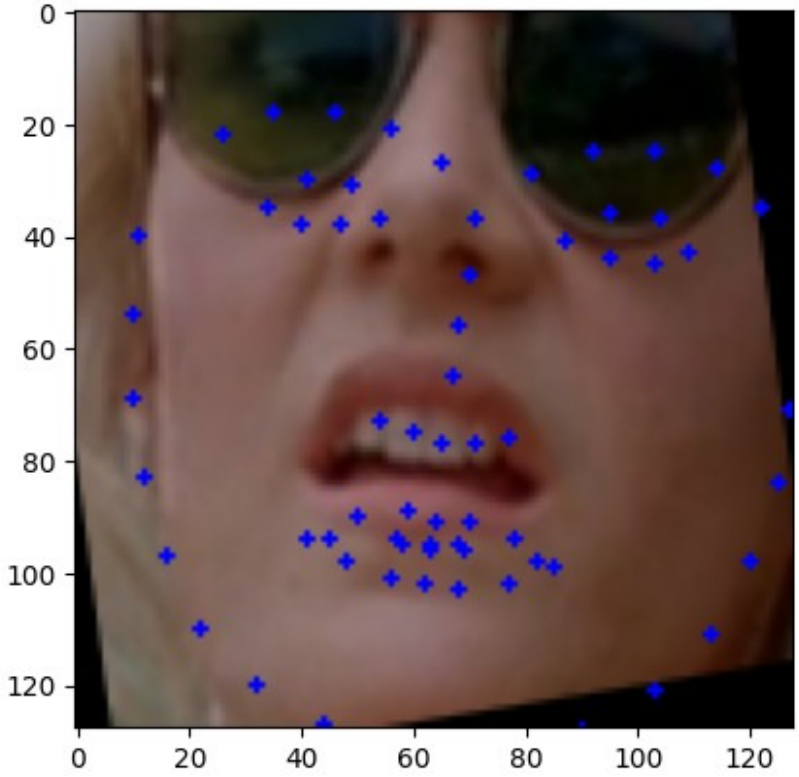

0
20
40
60
80
100
120
0
20
40
60
80
100
120

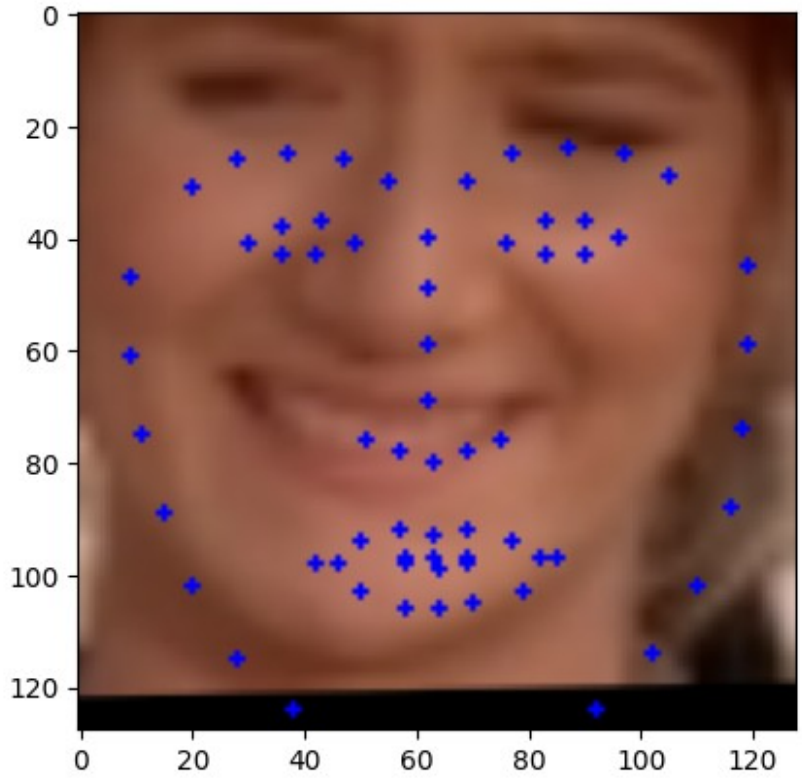

0
20
40
60
80
100
120
0
20
40
60
80
100
120

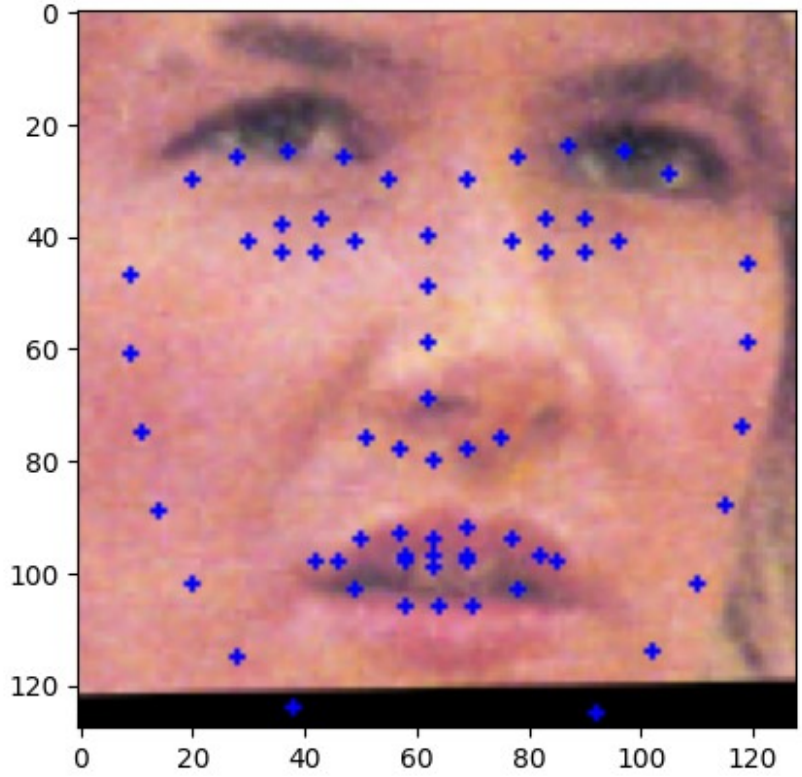

0
20
40
60
80
100
120
0
20
40
60
80
100
120

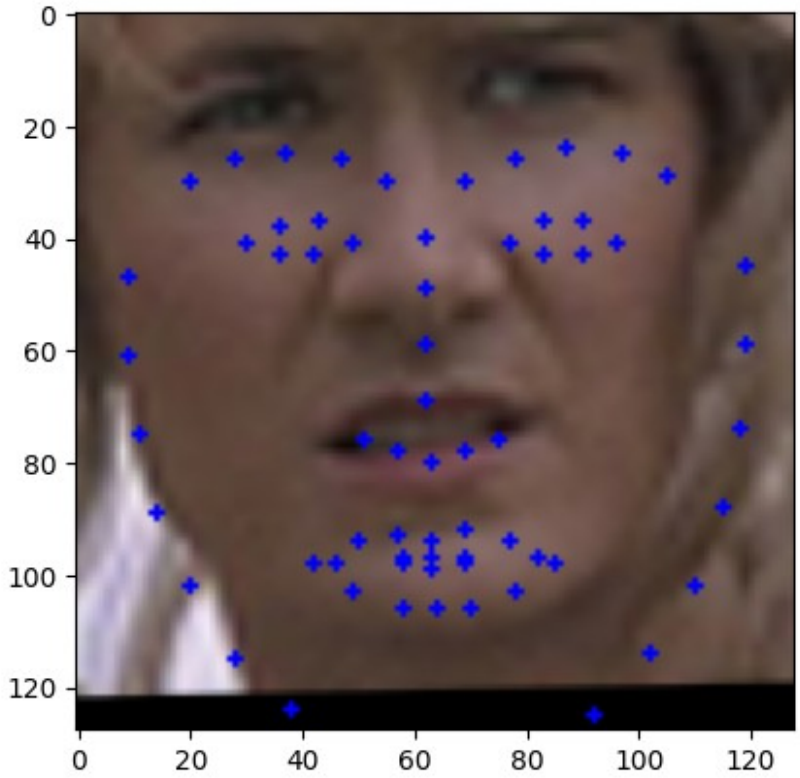

0
20
40
60
80
100
120
0
20
40
60
80
100
120

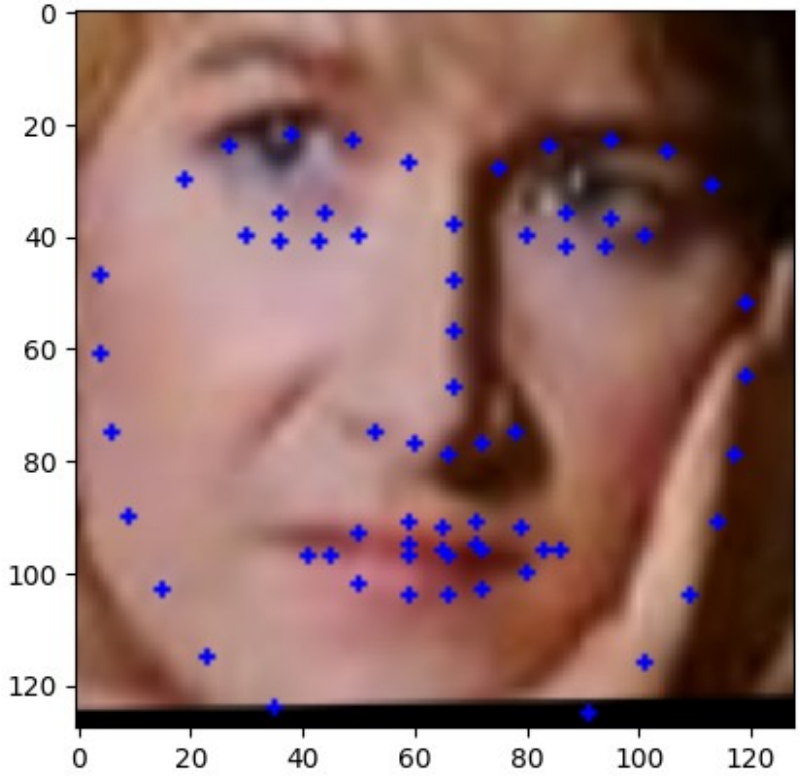

0
20
40
60
80
100
120
0
20
40
60
80
100
120

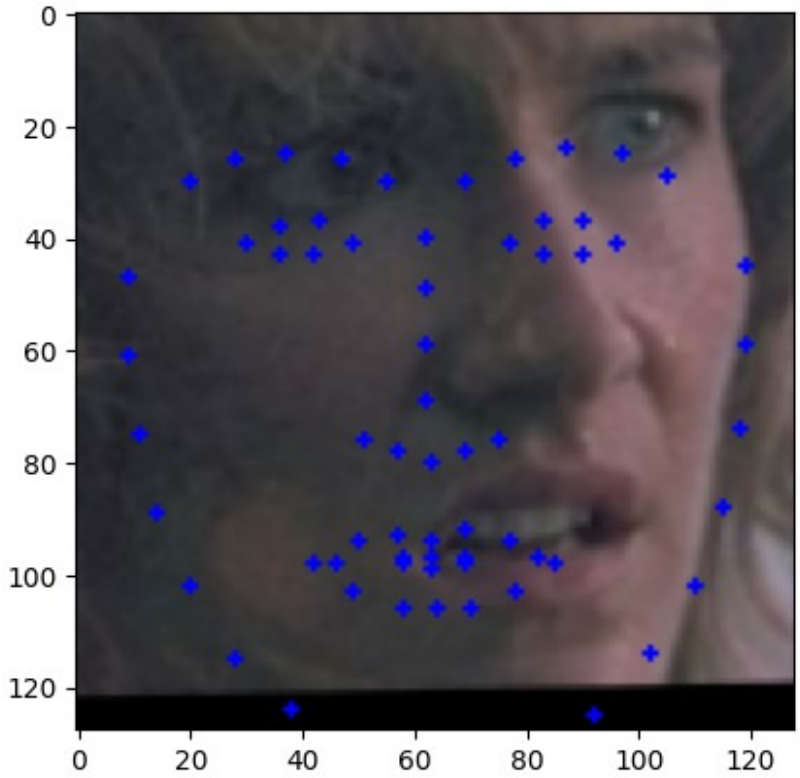

0
20
40
60
80
100
120
0
20
40
60
80
100
120

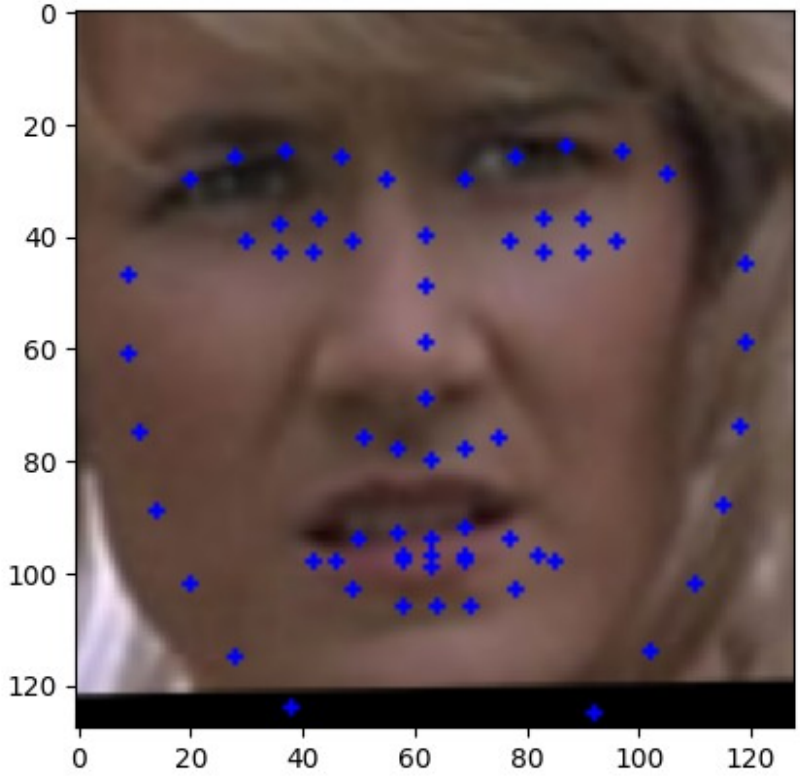

0
20
40
60
80
100
120
0
20
40
60
80
100
120

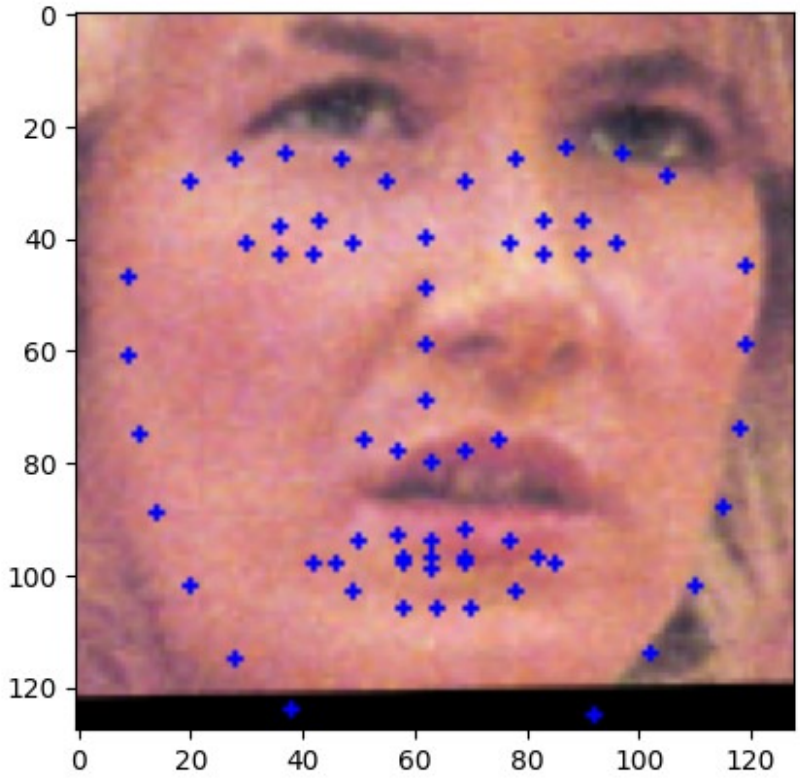

0
20
40
60
80
100
120
0
20
40
60
80
100
120

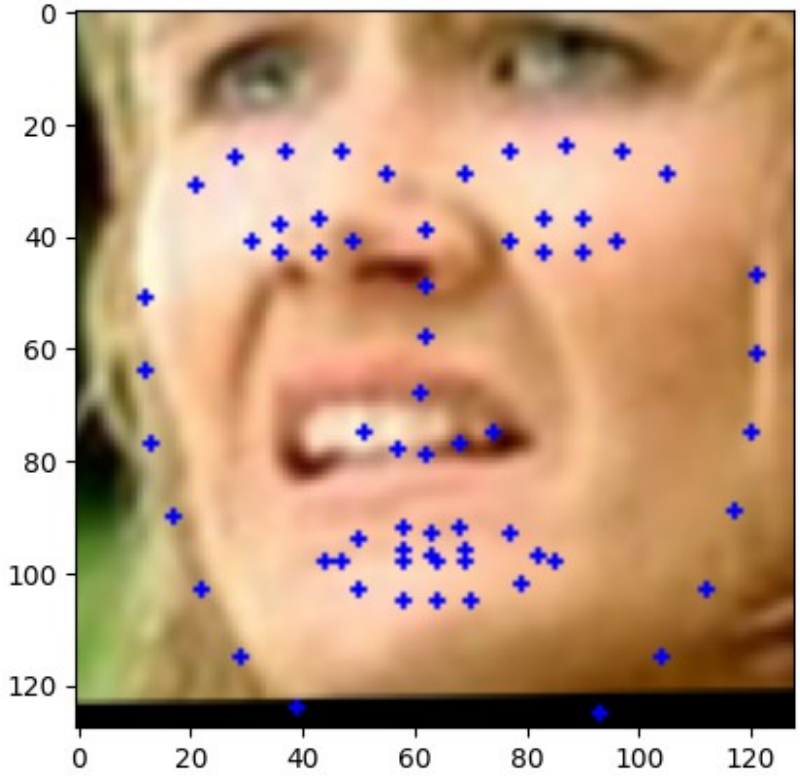

0
20
40
60
80
100
120
0
20
40
60
80
100
120

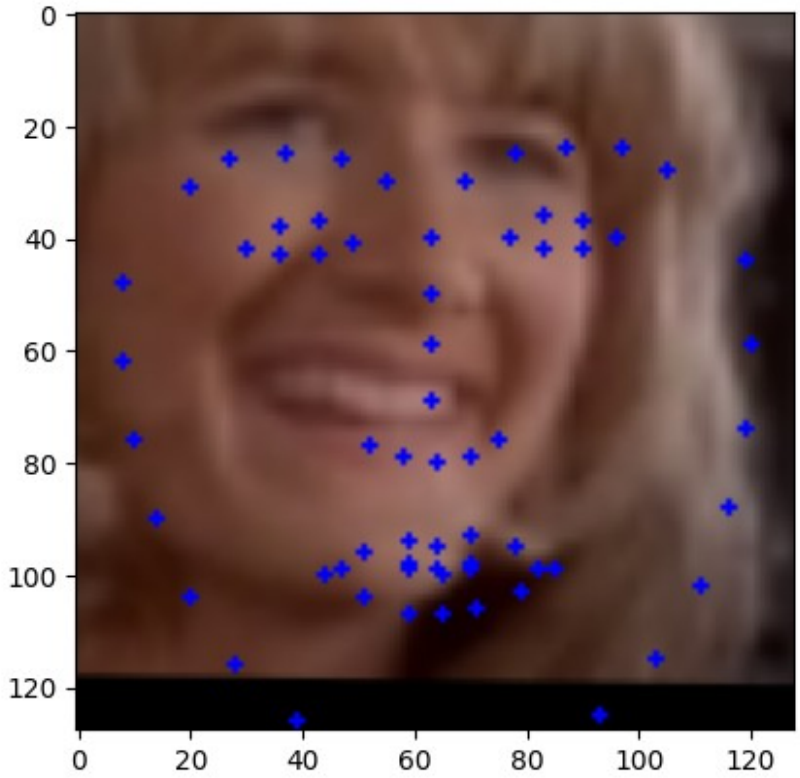

0
20
40
60
80
100
120
0
20
40
60
80
100
120

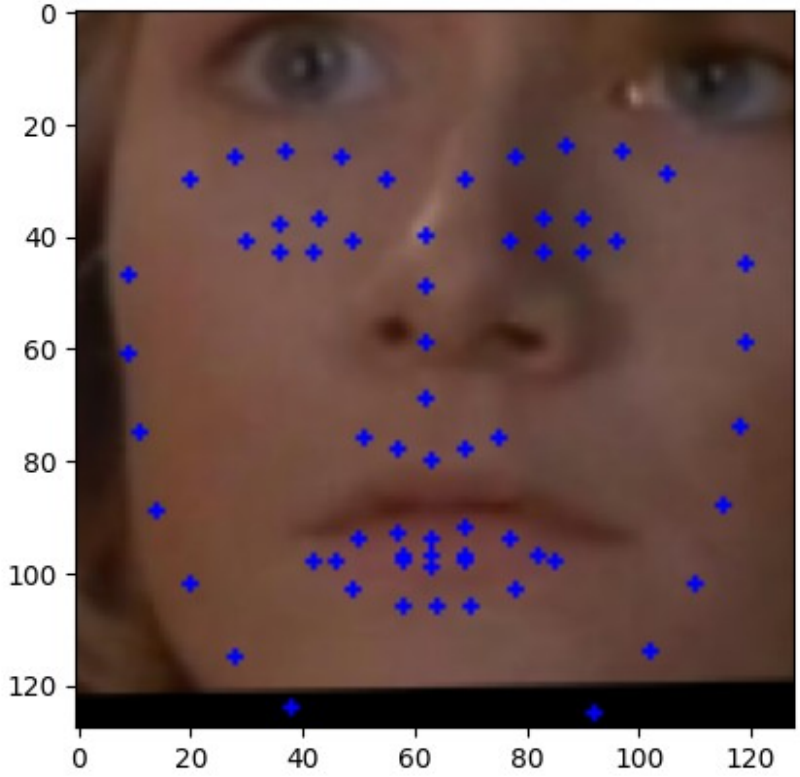

0
20
40
60
80
100
120
0
20
40
60
80
100
120

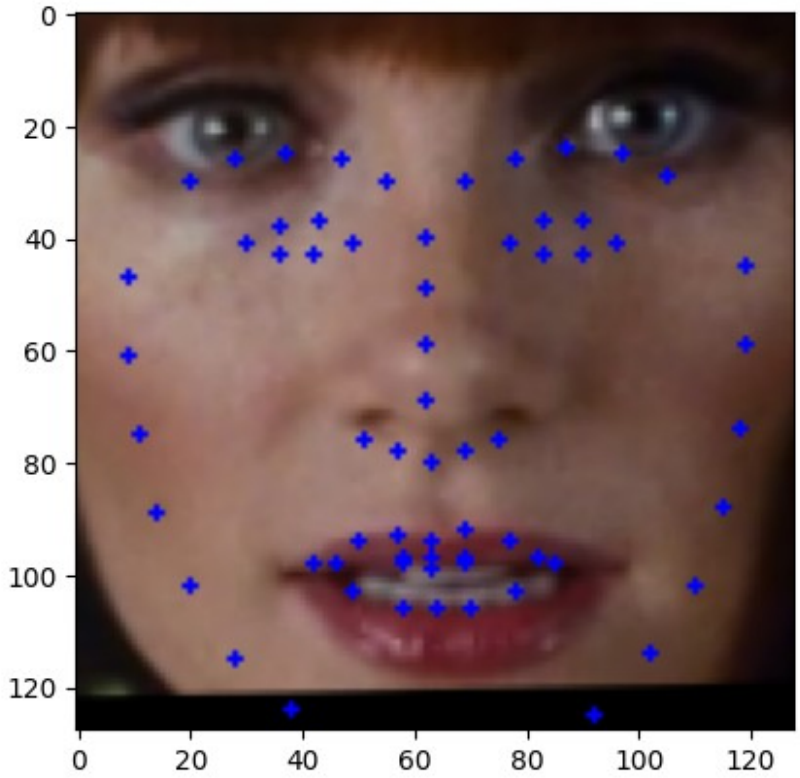

0
20
40
60
80
100
120
0
20
40
60
80
100
120

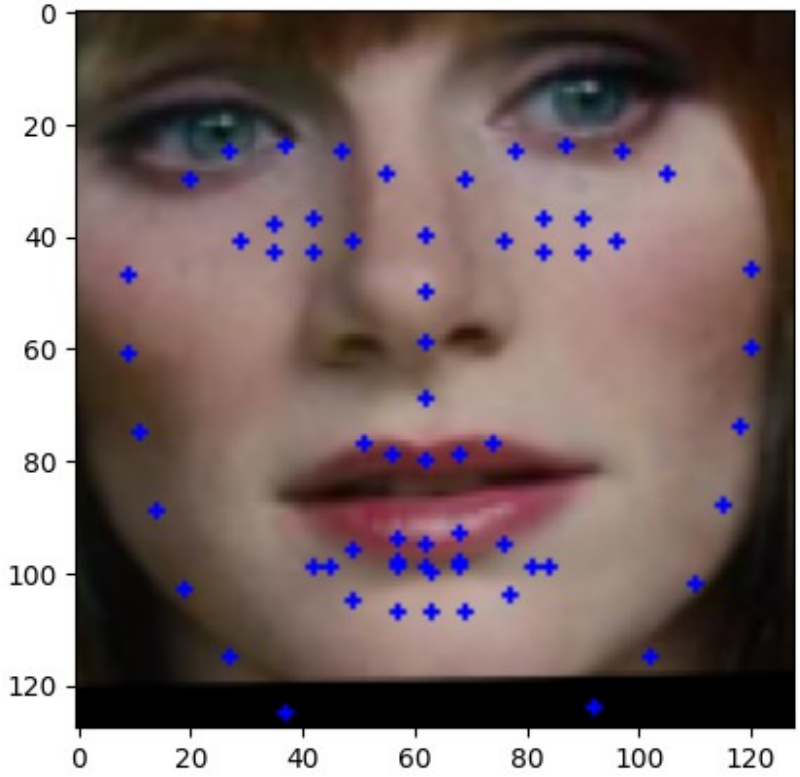

0
20
40
60
80
100
120
0
20
40
60
80
100
120

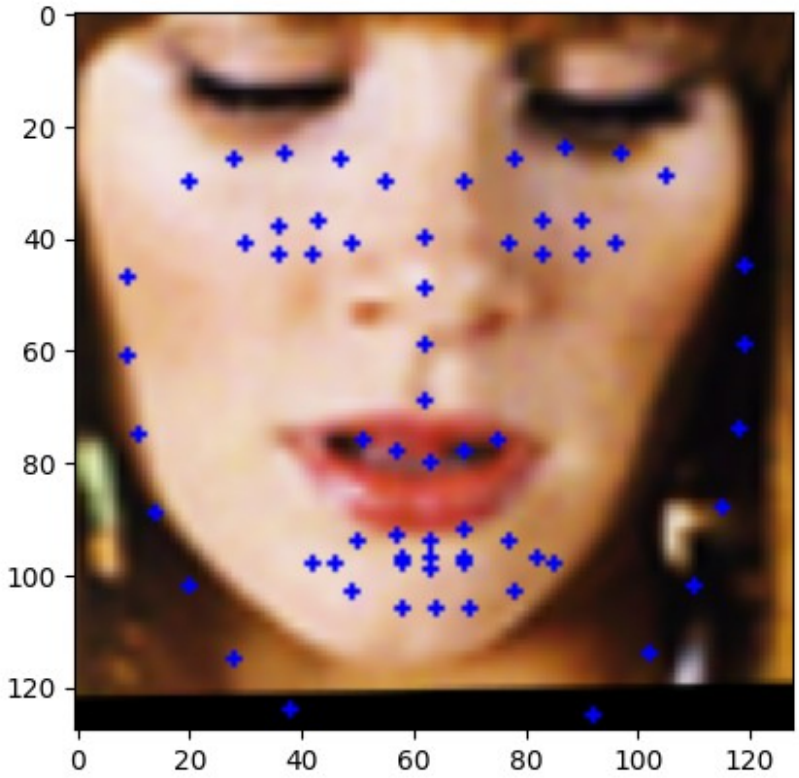

0
20
40
60
80
100
120
0
20
40
60
80
100
120

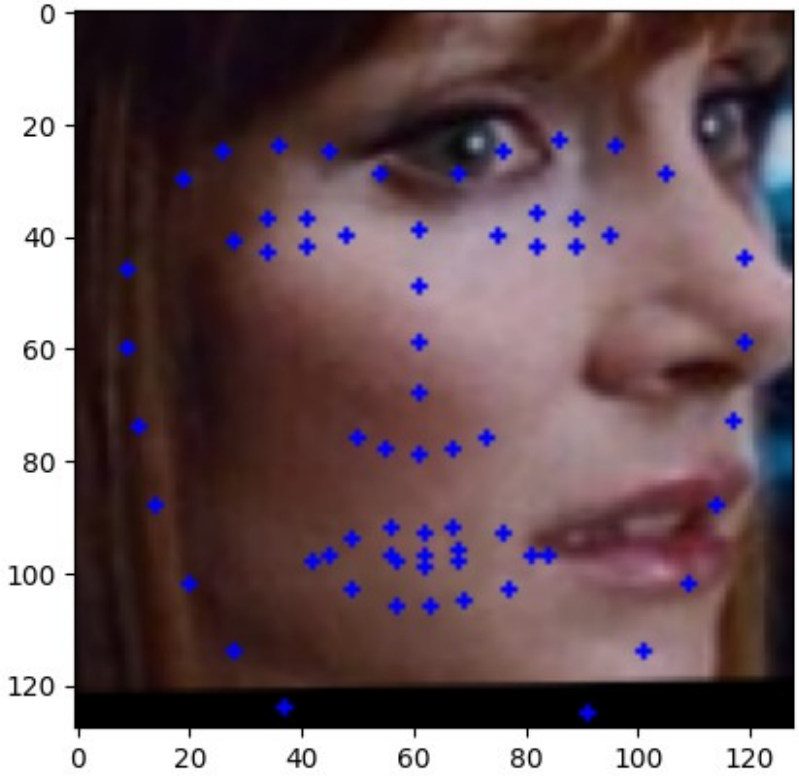

0
20
40
60
80
100
120
0
20
40
60
80
100
120

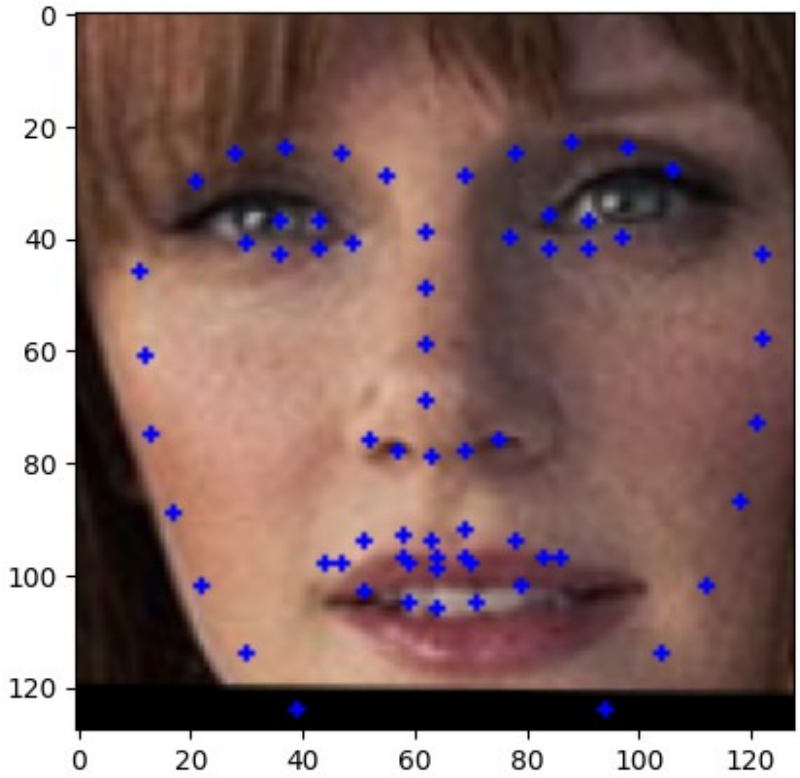

0
20
40
60
80
100
120
0
20
40
60
80
100
120

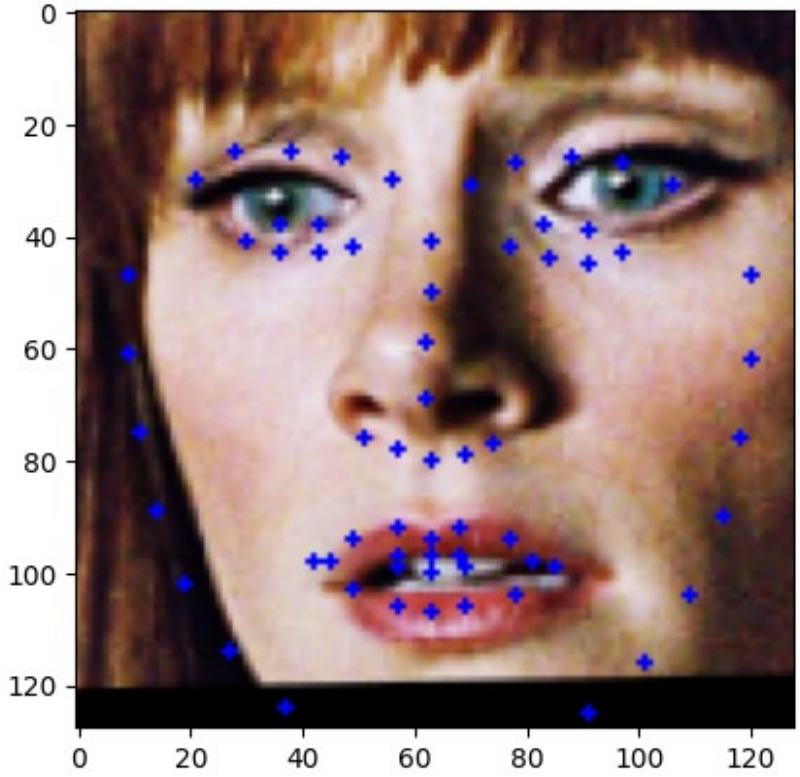

0
20
40
60
80
100
120
0
20
40
60
80
100
120

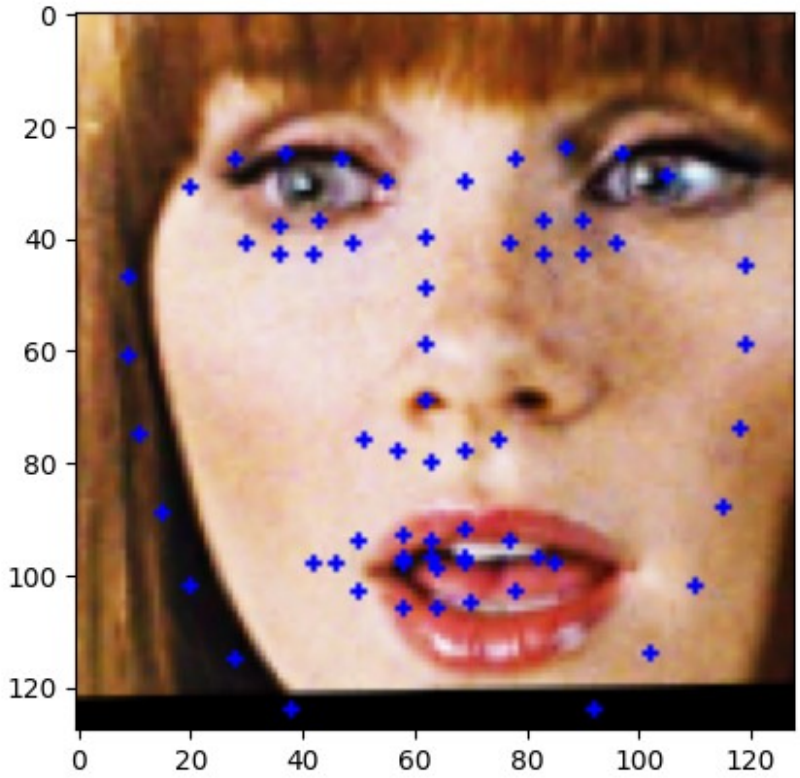

0
20
40
60
80
100
120
0
20
40
60
80
100
120

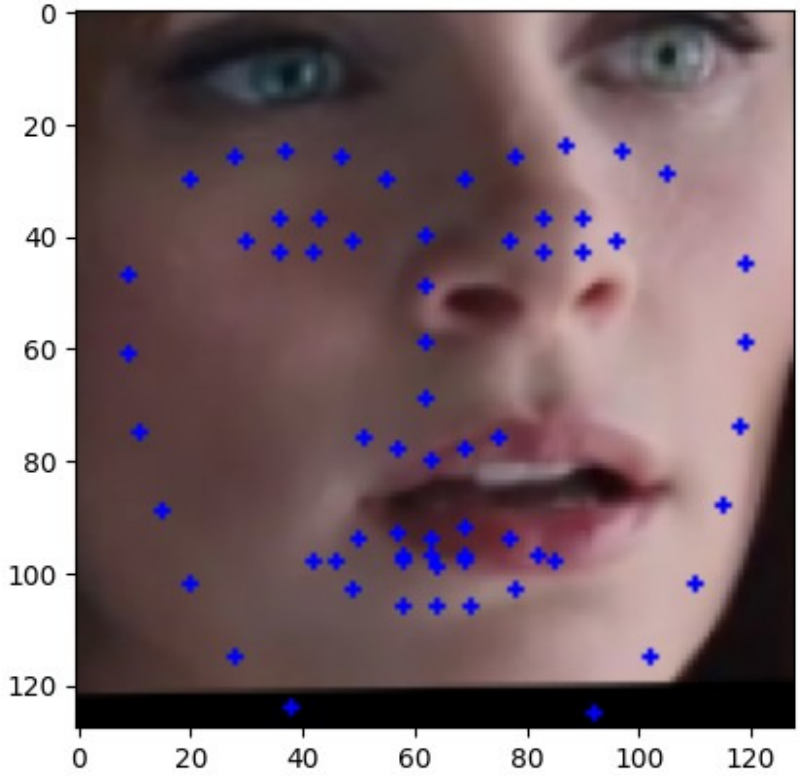

0
20
40
60
80
100
120
0
20
40
60
80
100
120

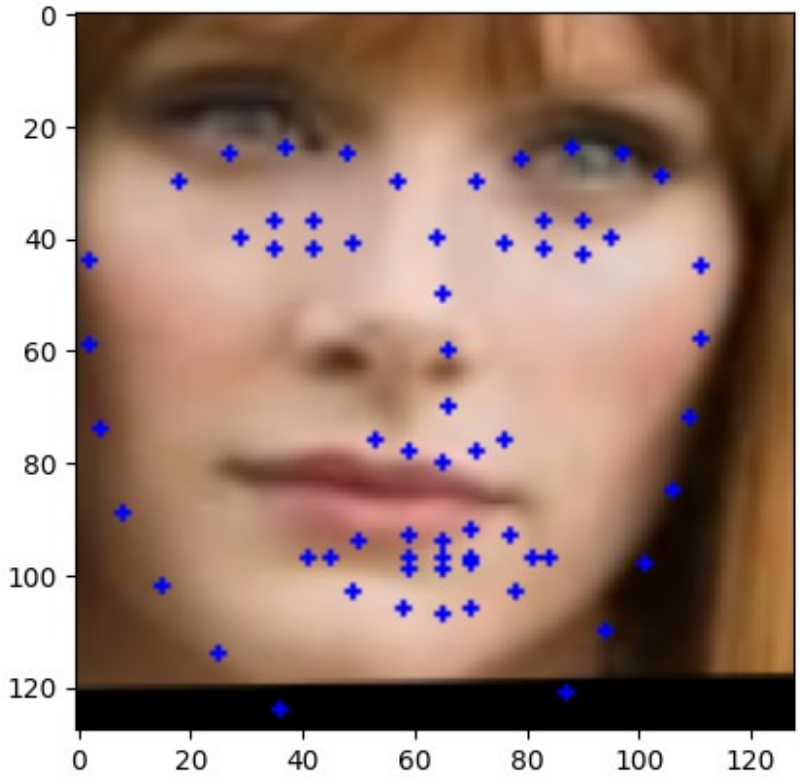

0
20
40
60
80
100
120
0
20
40
60
80
100
120

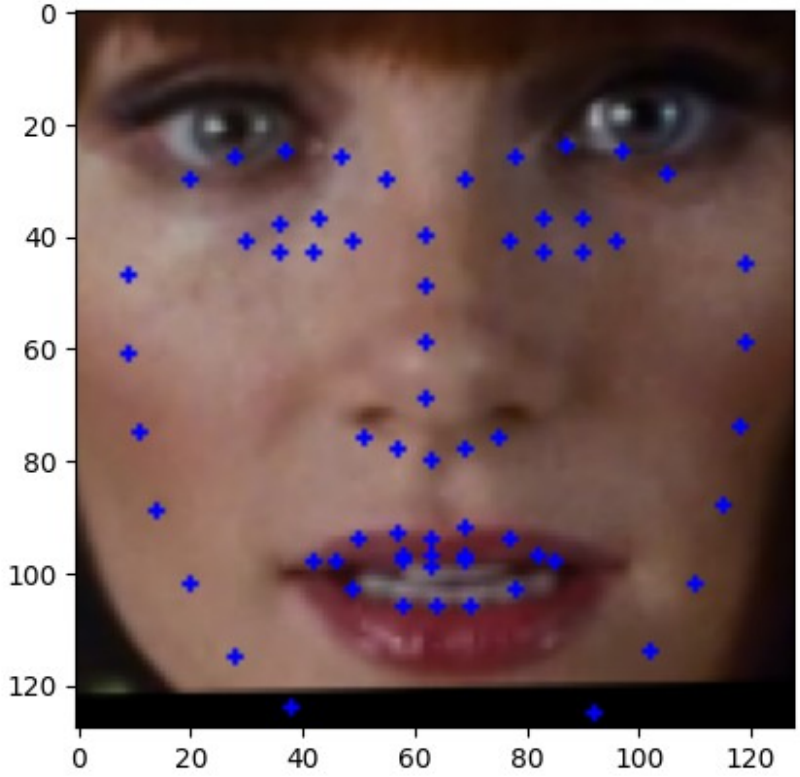

0
20
40
60
80
100
120
0
20
40
60
80
100
120

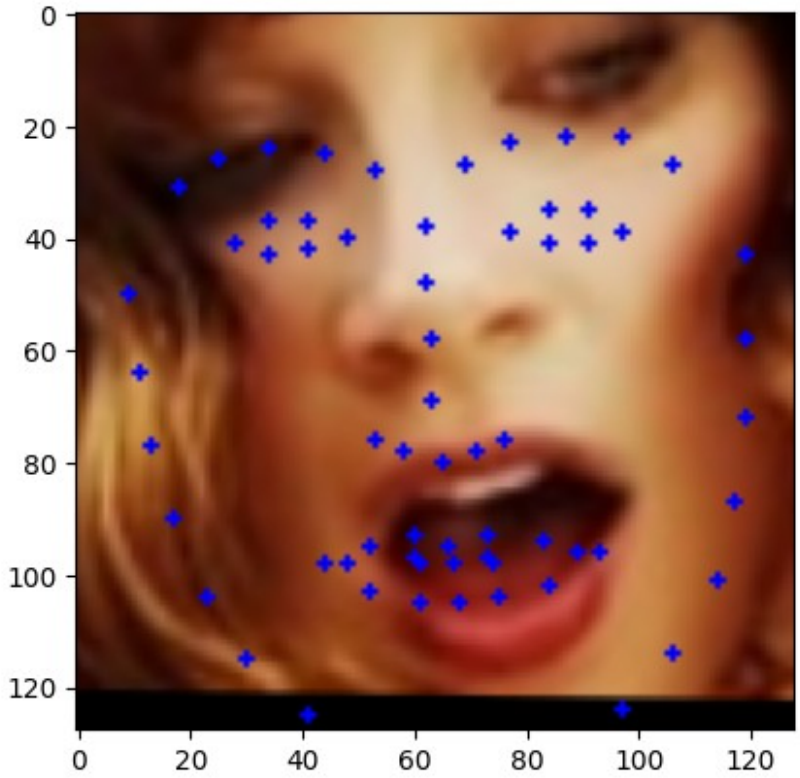

0
20
40
60
80
100
120
0
20
40
60
80
100
120

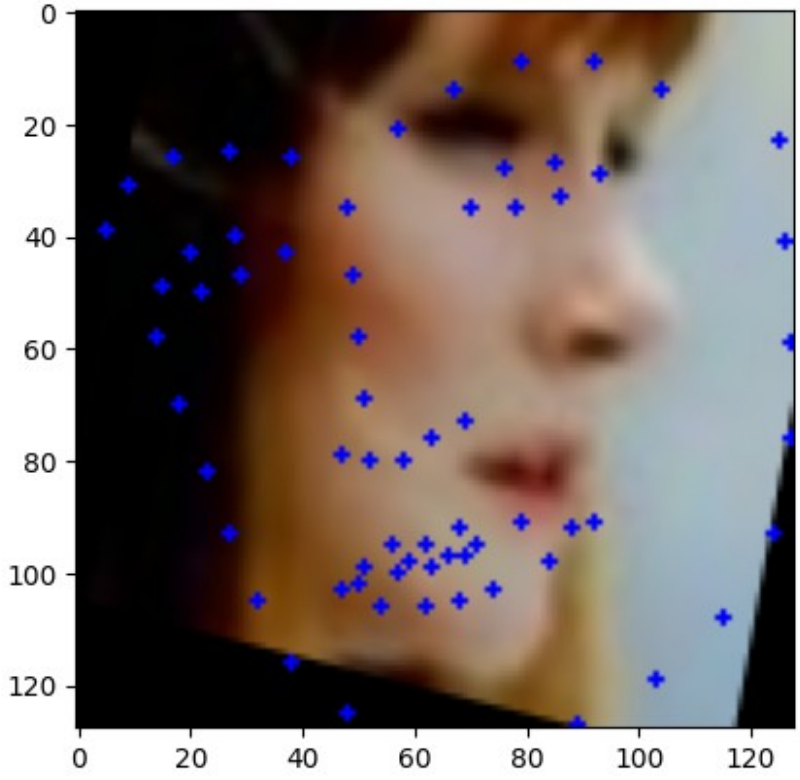

0
20
40
60
80
100
120
0
20
40
60
80
100
120

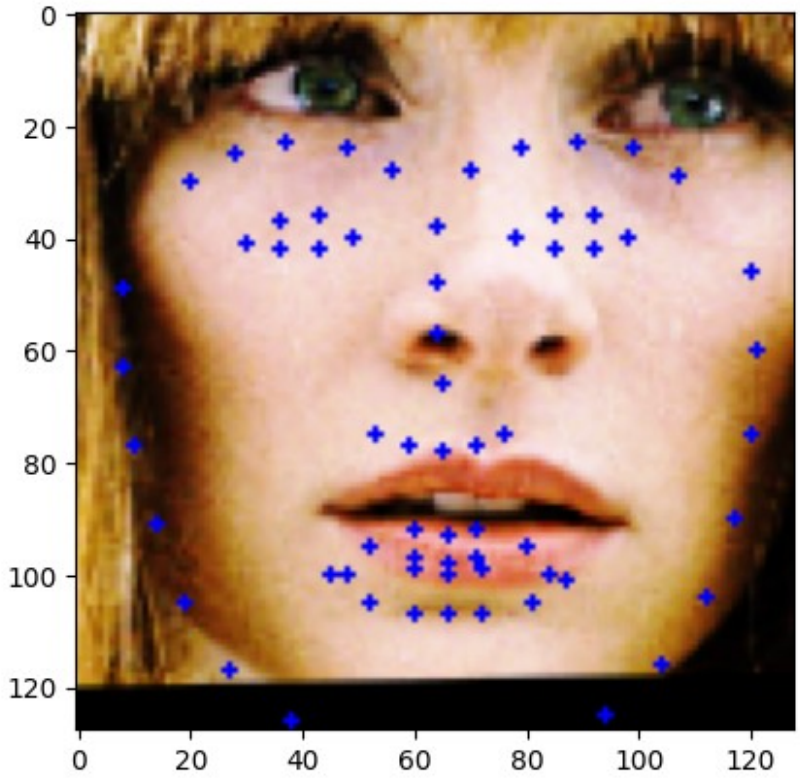

0
20
40
60
80
100
120
0
20
40
60
80
100
120

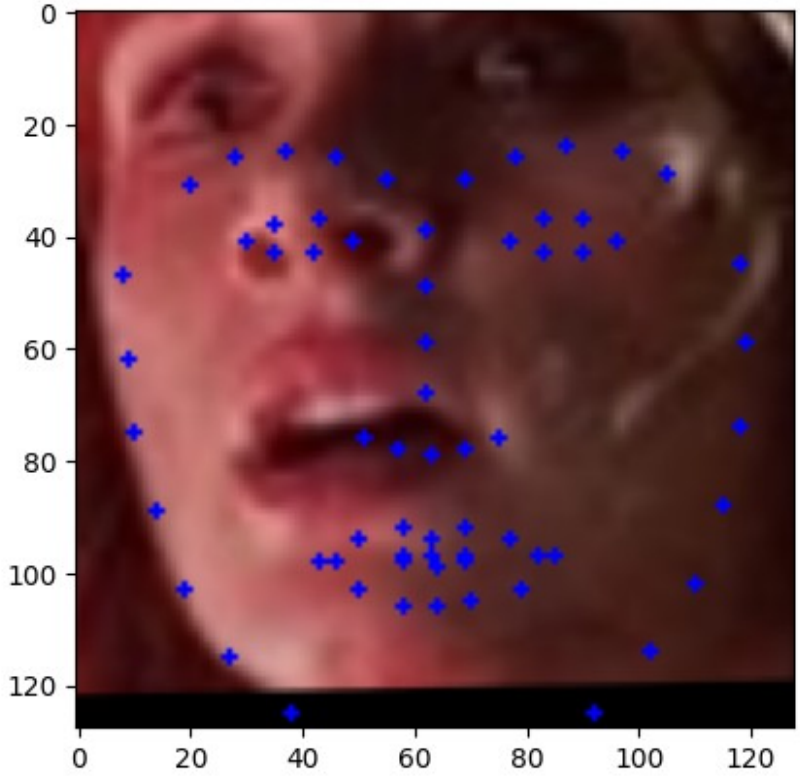

0
20
40
60
80
100
120
0
20
40
60
80
100
120

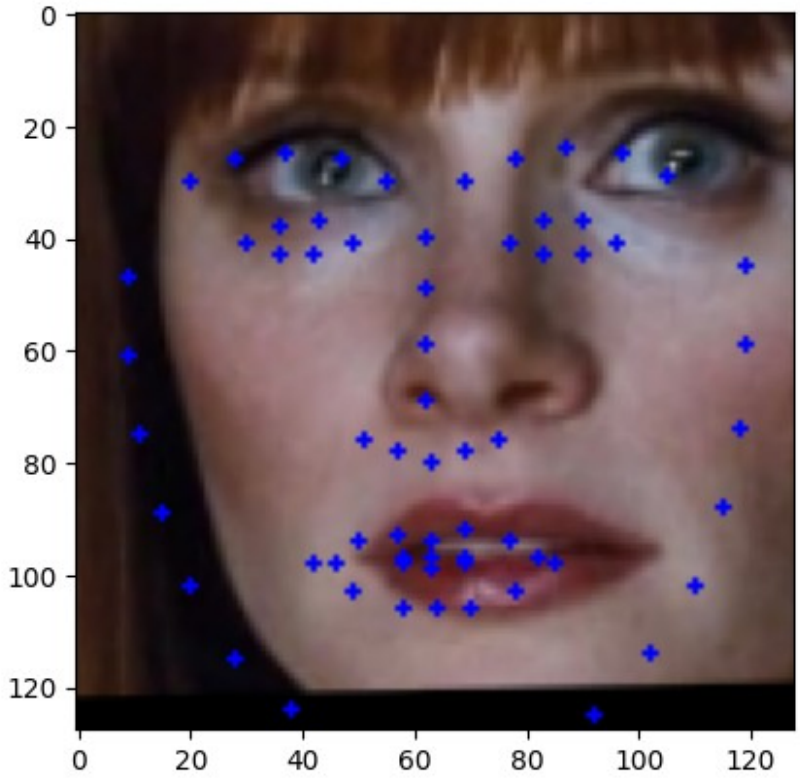

0
20
40
60
80
100
120
0
20
40
60
80
100
120

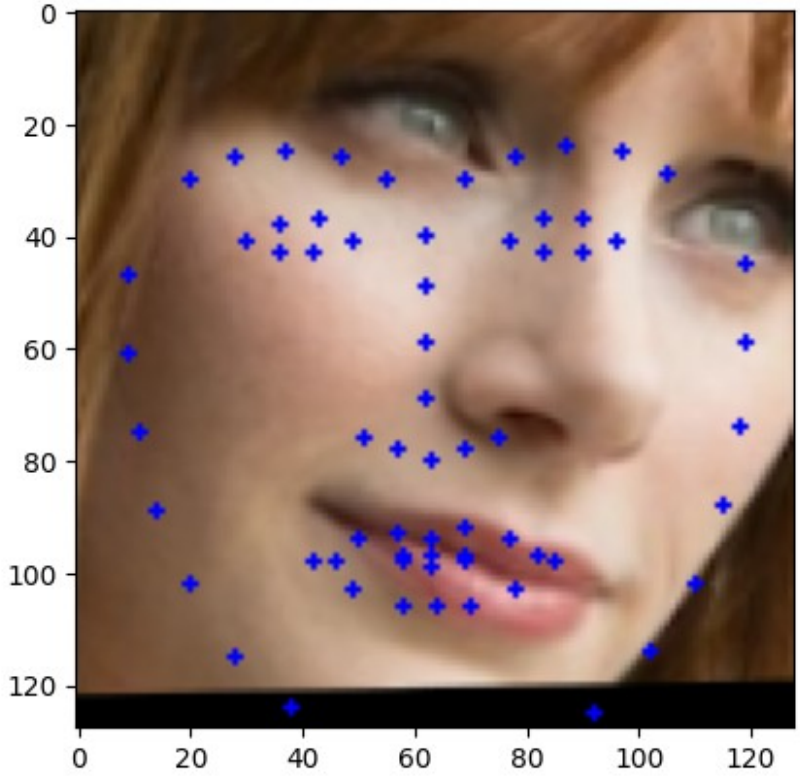

0
20
40
60
80
100
120
0
20
40
60
80
100
120

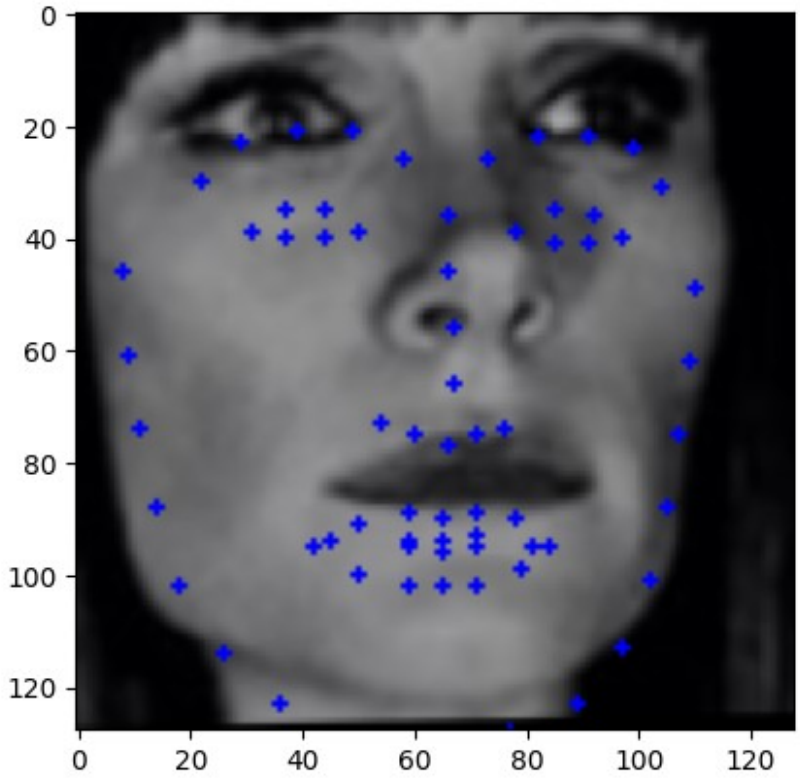

0
20
40
60
80
100
120
0
20
40
60
80
100
120

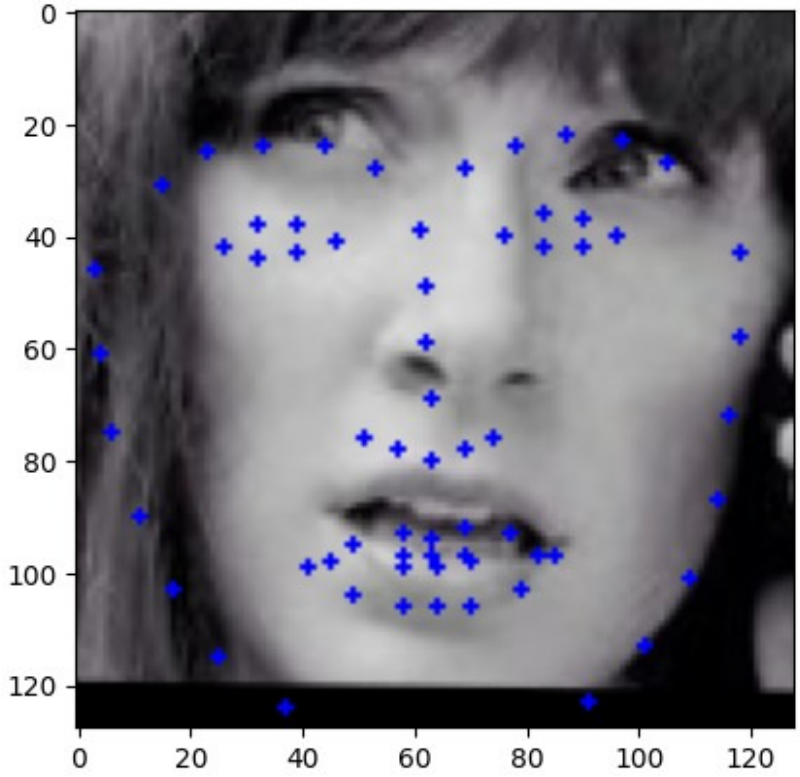

0
20
40
60
80
100
120
0
20
40
60
80
100
120

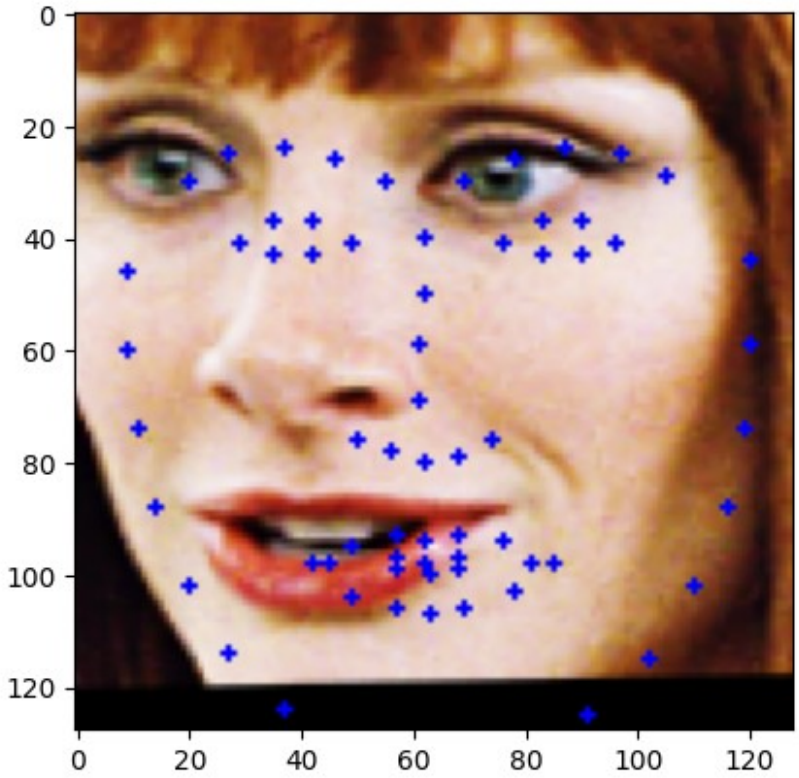

0
20
40
60
80
100
120
0
20
40
60
80
100
120

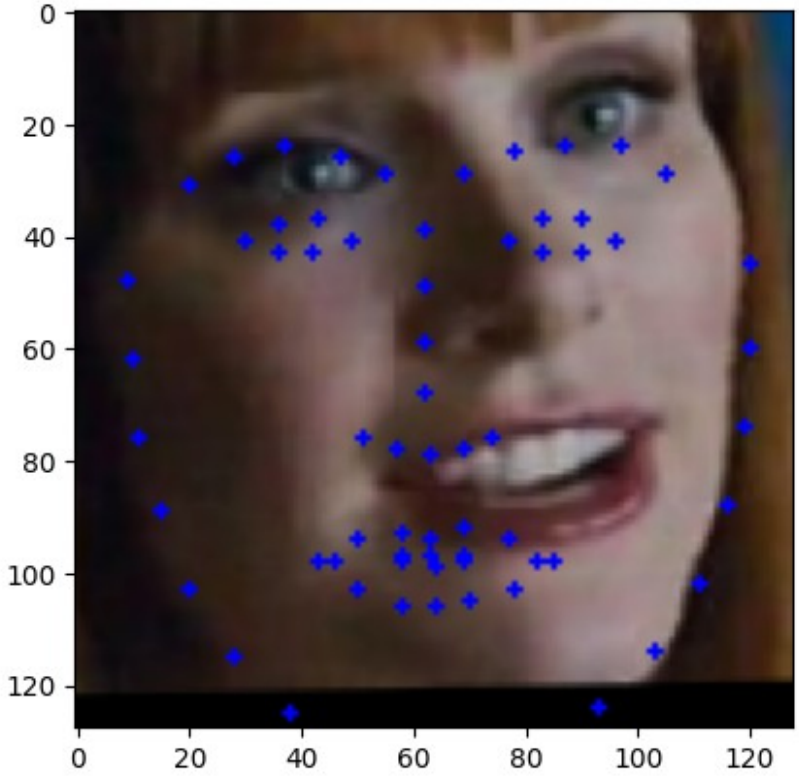

0
20
40
60
80
100
120
0
20
40
60
80
100
120

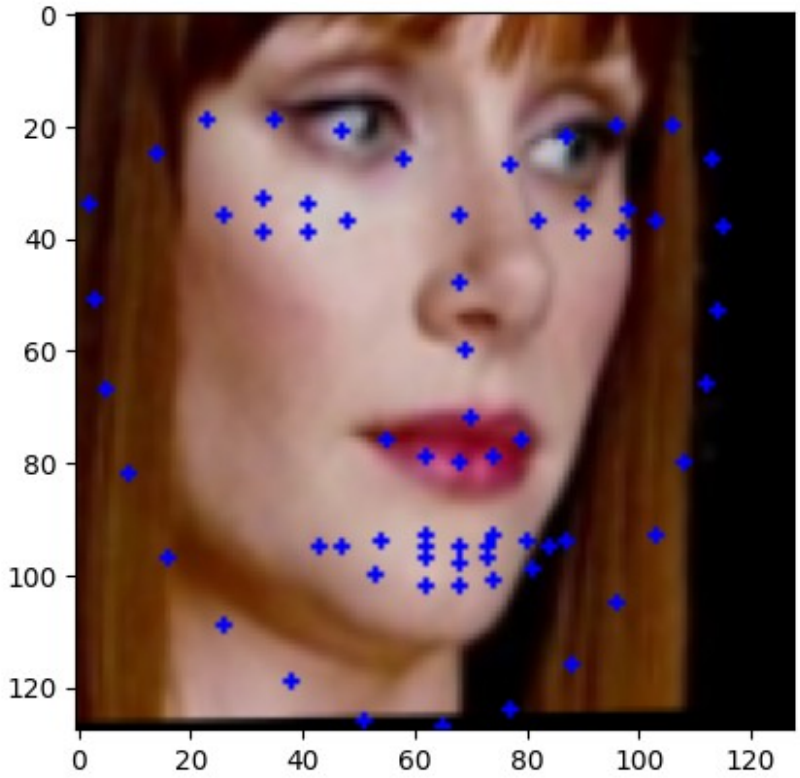

0
20
40
60
80
100
120
0
20
40
60
80
100
120

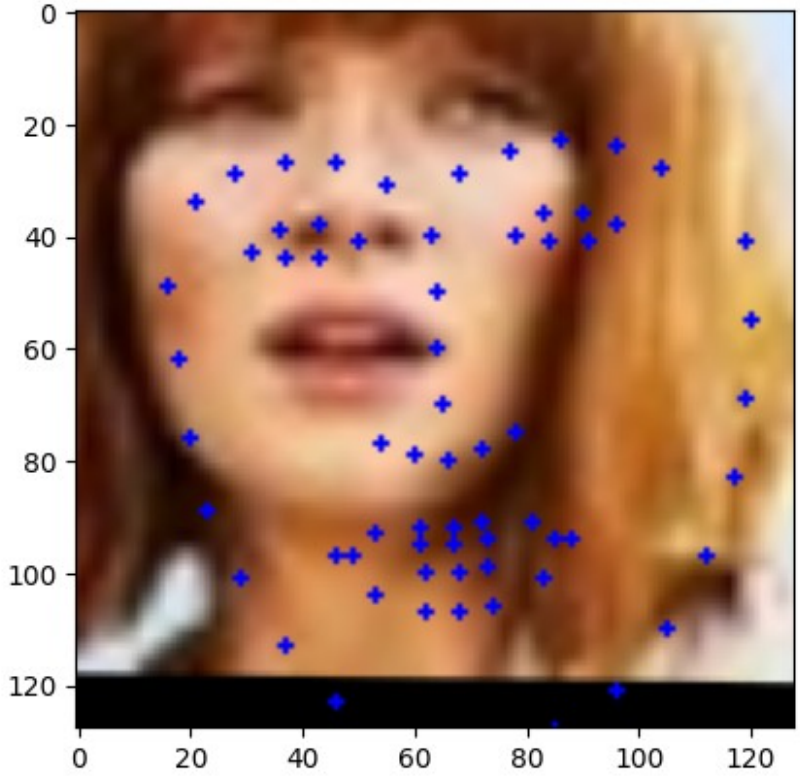

0
20
40
60
80
100
120
0
20
40
60
80
100
120

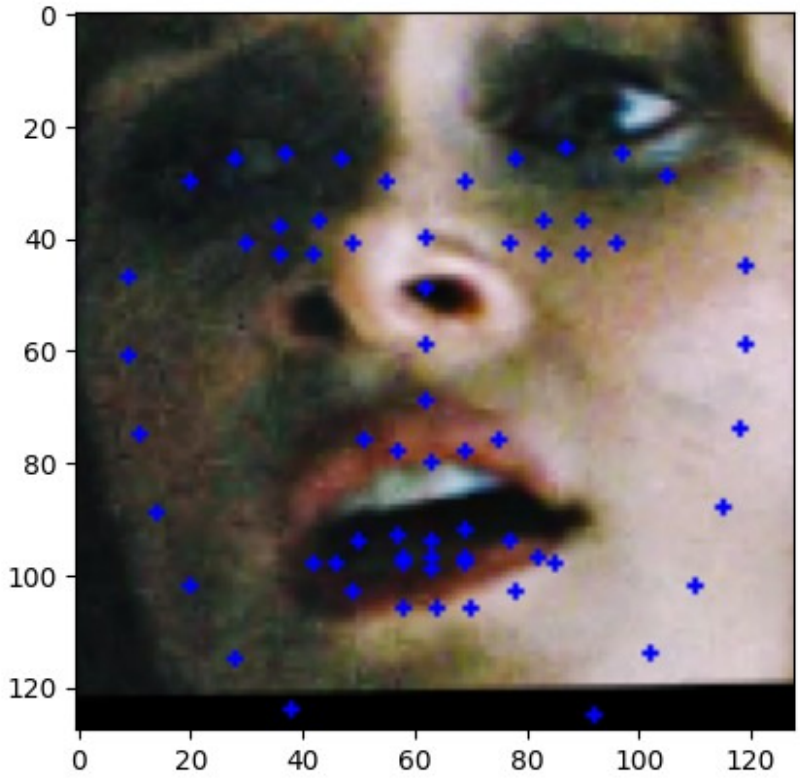

0
20
40
60
80
100
120
0
20
40
60
80
100
120

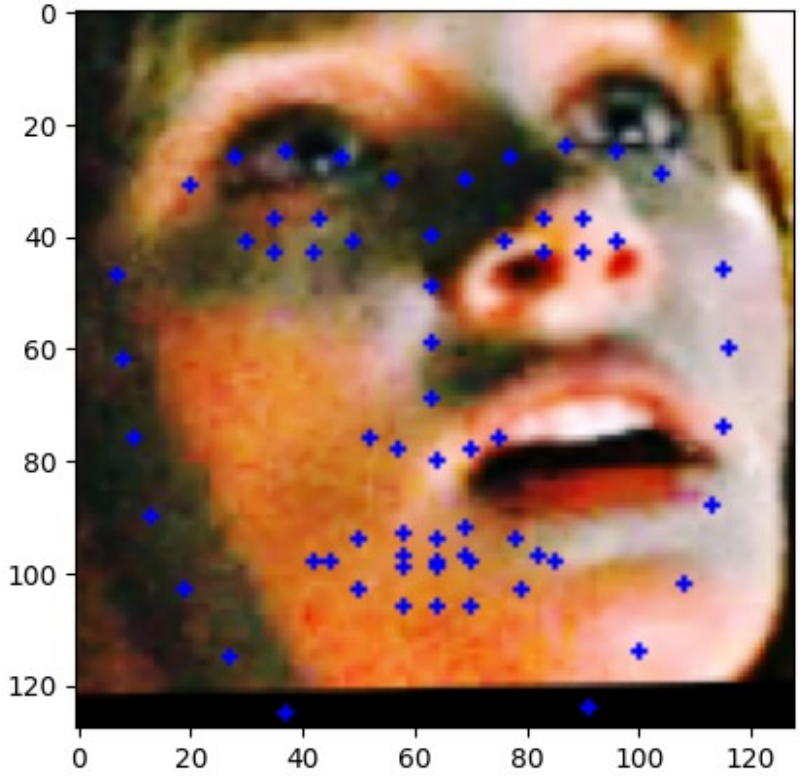

0
20
40
60
80
100
120
0
20
40
60
80
100
120

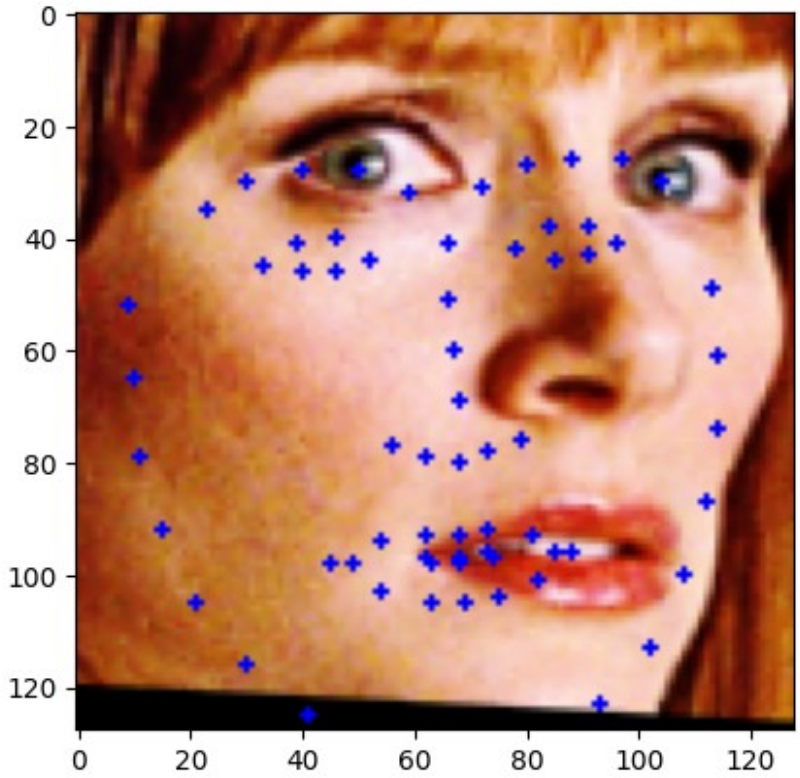

0
20
40
60
80
100
120
0
20
40
60
80
100
120

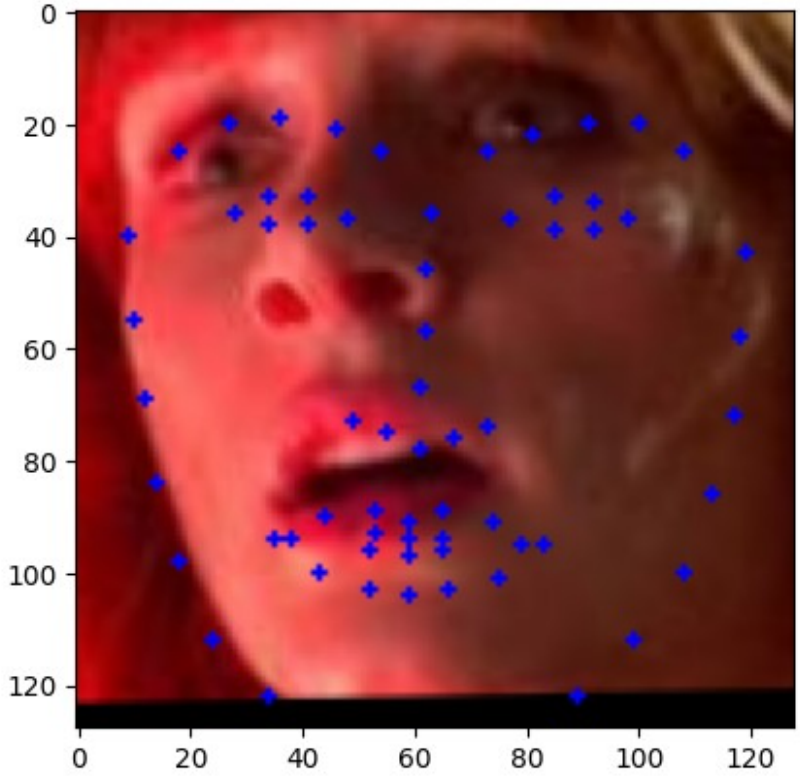

0
20
40
60
80
100
120
0
20
40
60
80
100
120

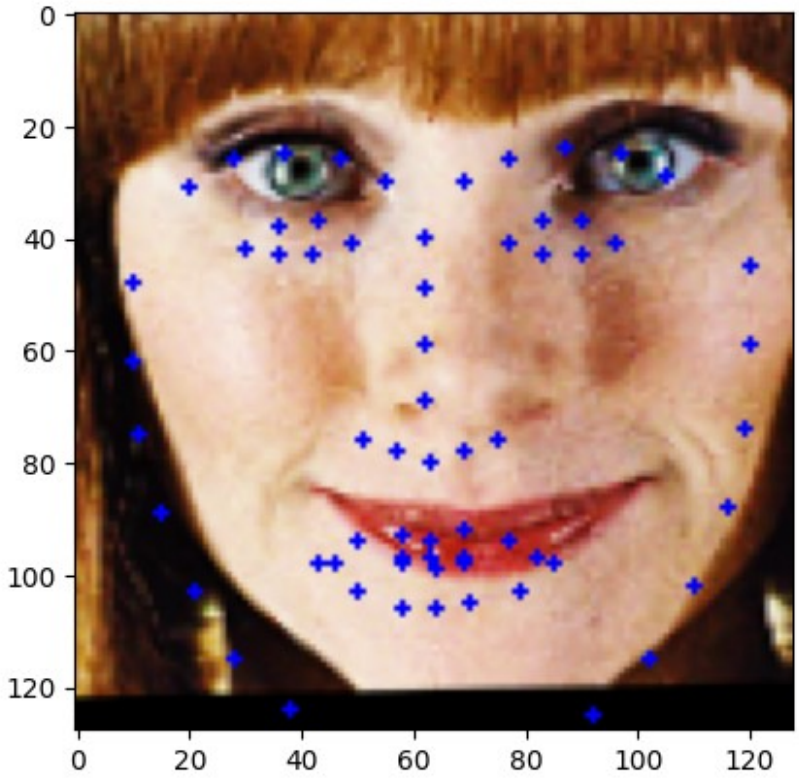

0
20
40
60
80
100
120
0
20
40
60
80
100
120

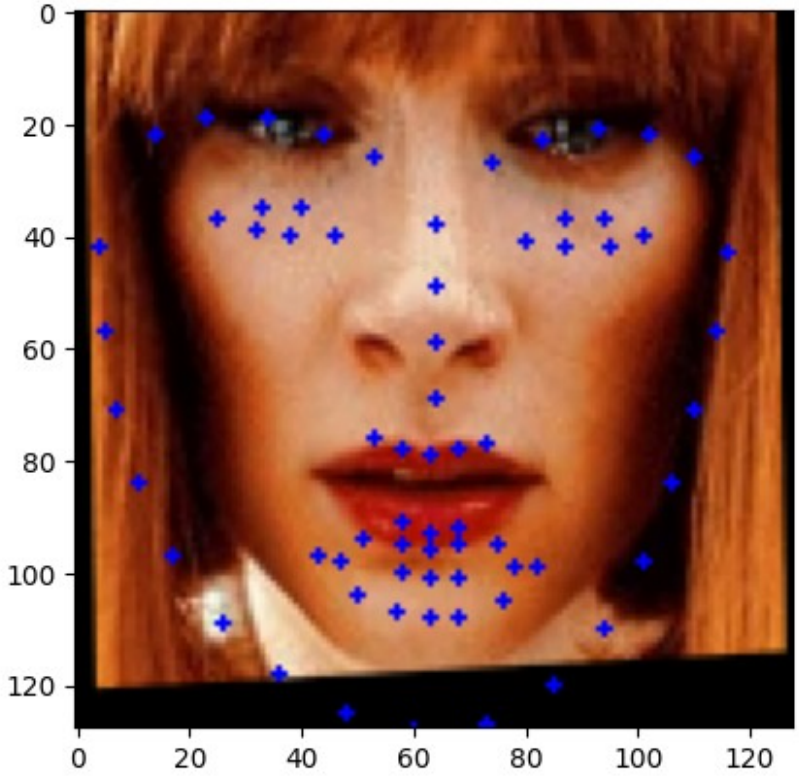

0
20
40
60
80
100
120
0
20
40
60
80
100
120

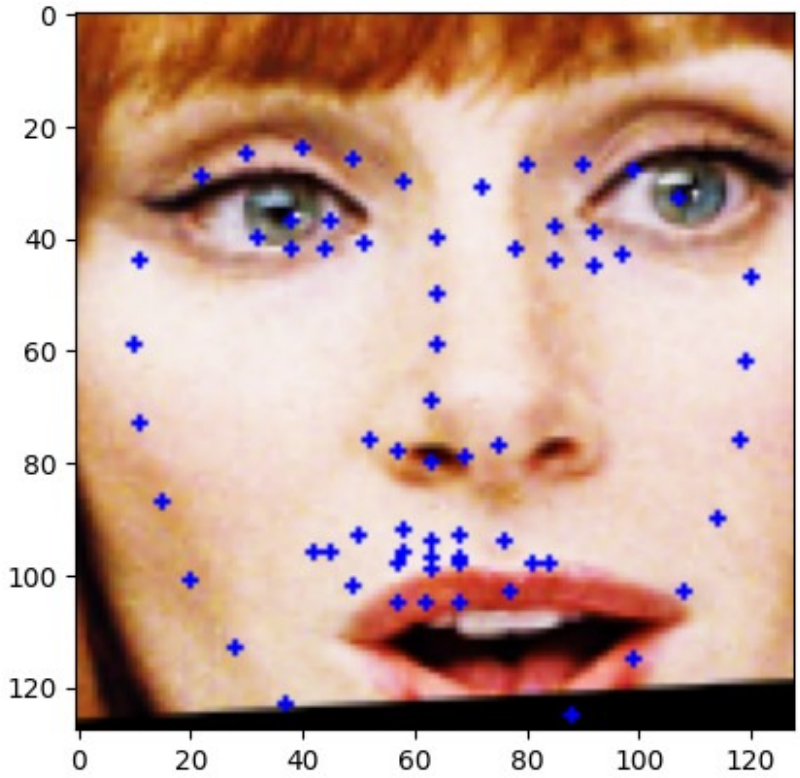

0
20
40
60
80
100
120
0
20
40
60
80
100
120

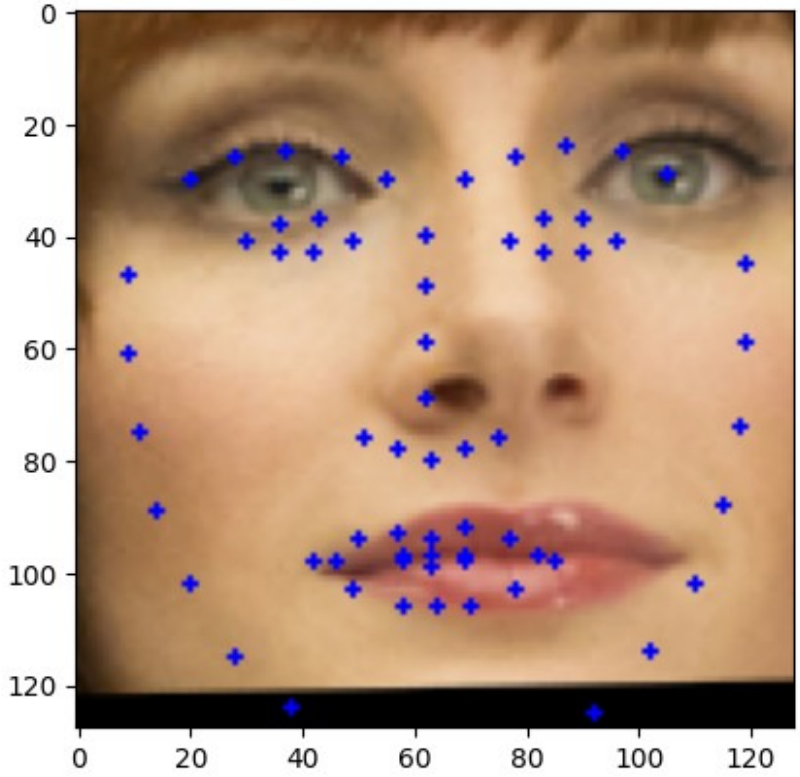

0
20
40
60
80
100
120
0
20
40
60
80
100
120

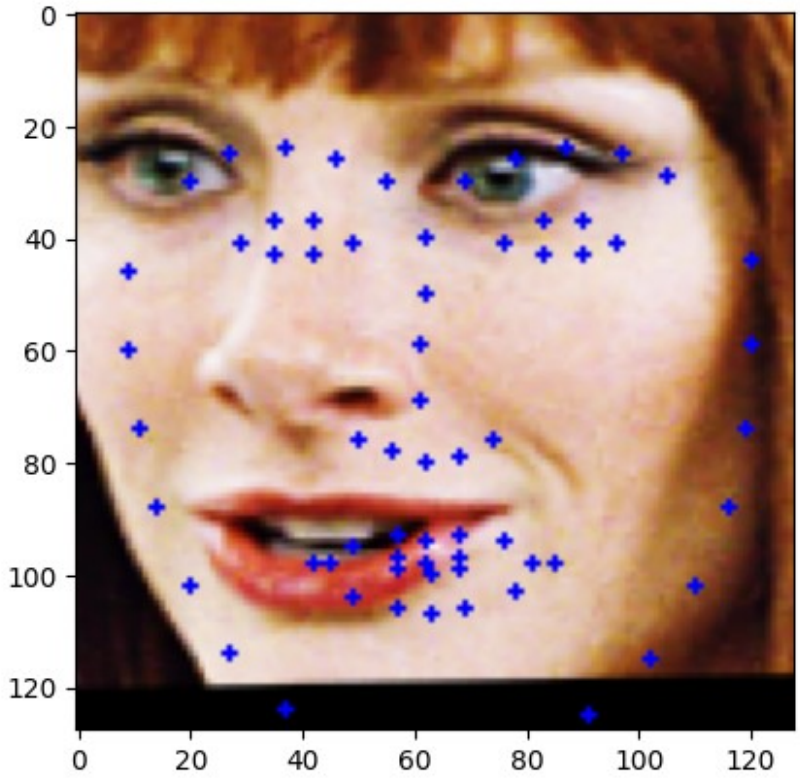

0
20
40
60
80
100
120
0
20
40
60
80
100
120

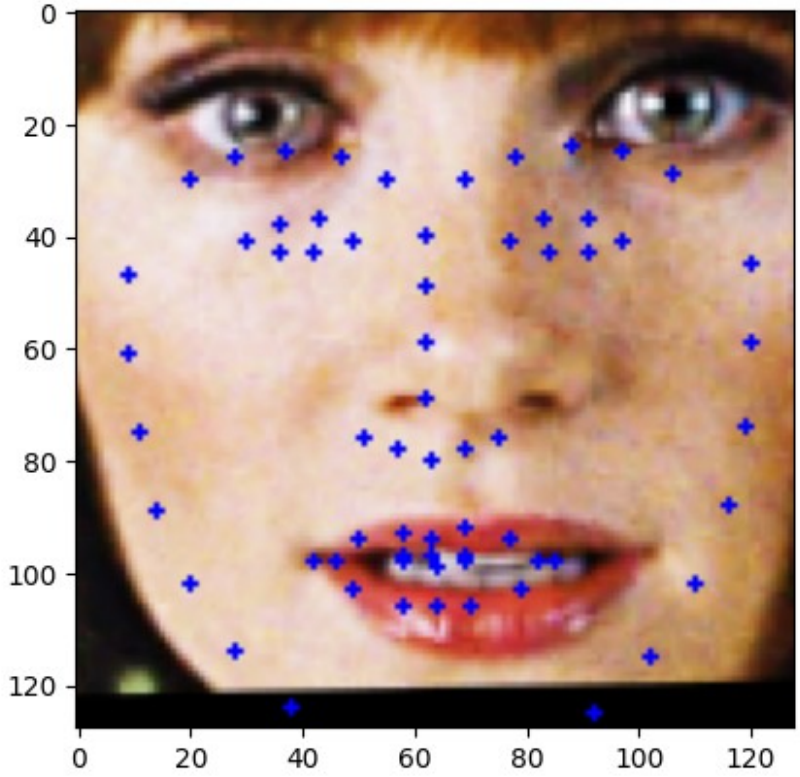

0
20
40
60
80
100
120
0
20
40
60
80
100
120

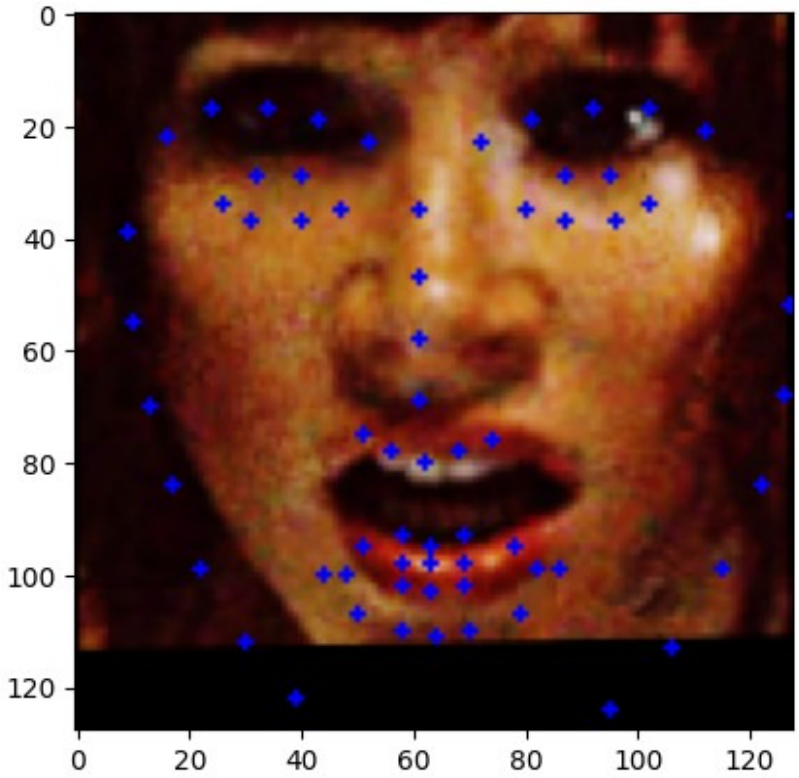

0
20
40
60
80
100
120
0
20
40
60
80
100
120

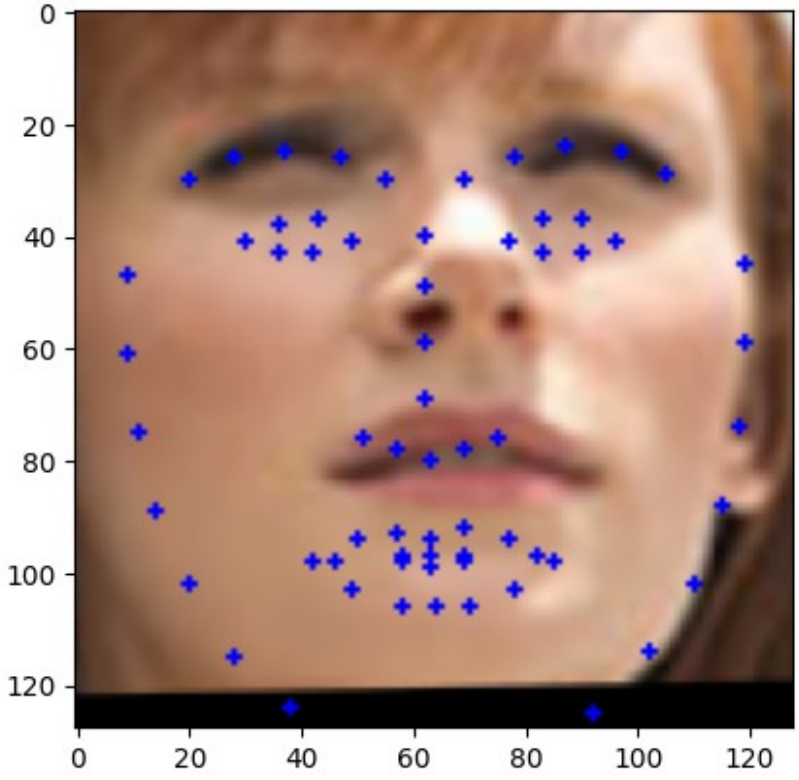

0
20
40
60
80
100
120
0
20
40
60
80
100
120

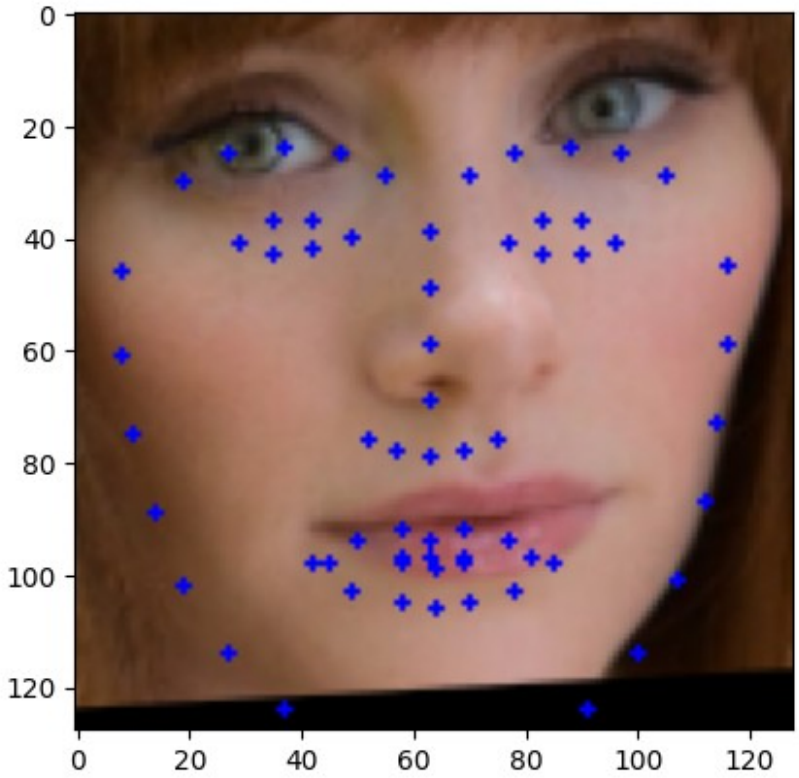

0
20
40
60
80
100
120
0
20
40
60
80
100
120

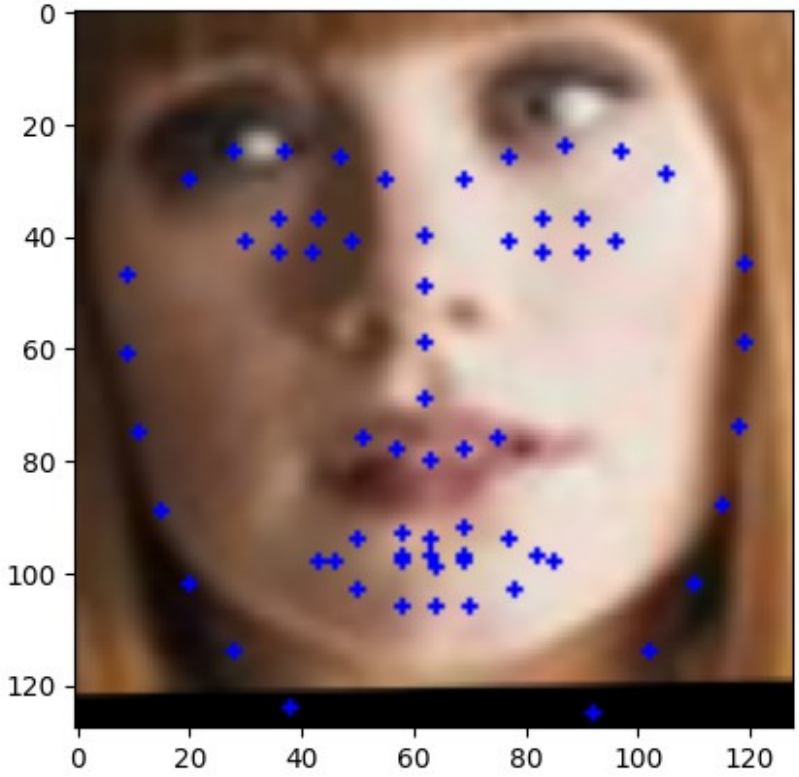

0
20
40
60
80
100
120
0
20
40
60
80
100
120

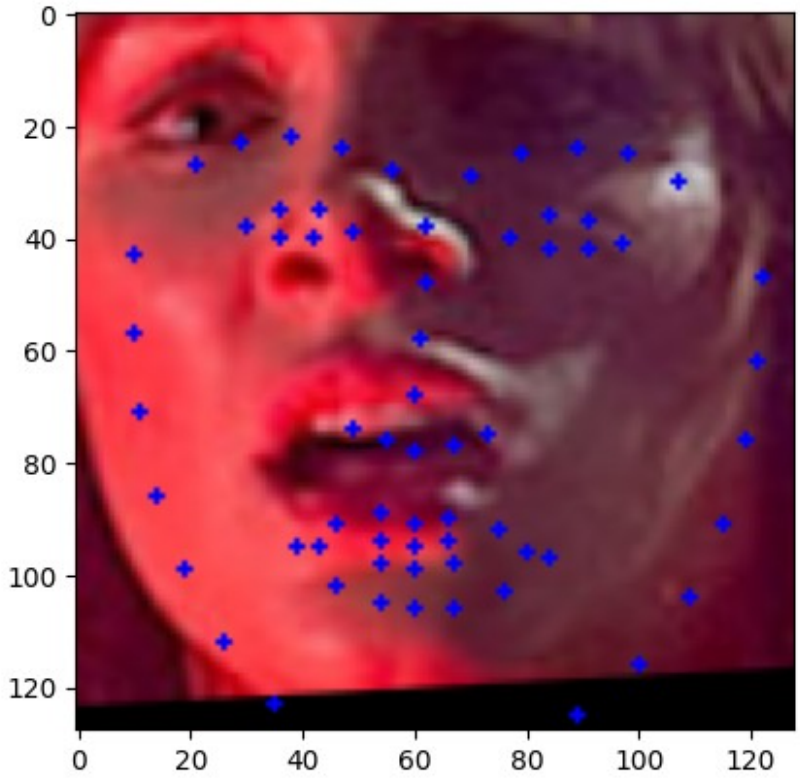

0
20
40
60
80
100
120
0
20
40
60
80
100
120

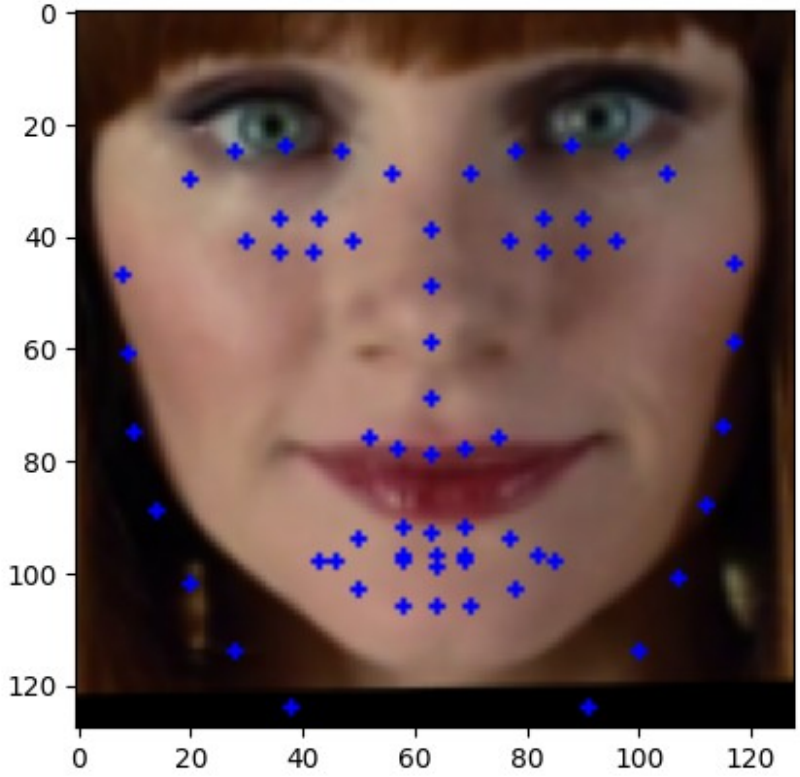

0
20
40
60
80
100
120
0
20
40
60
80
100
120

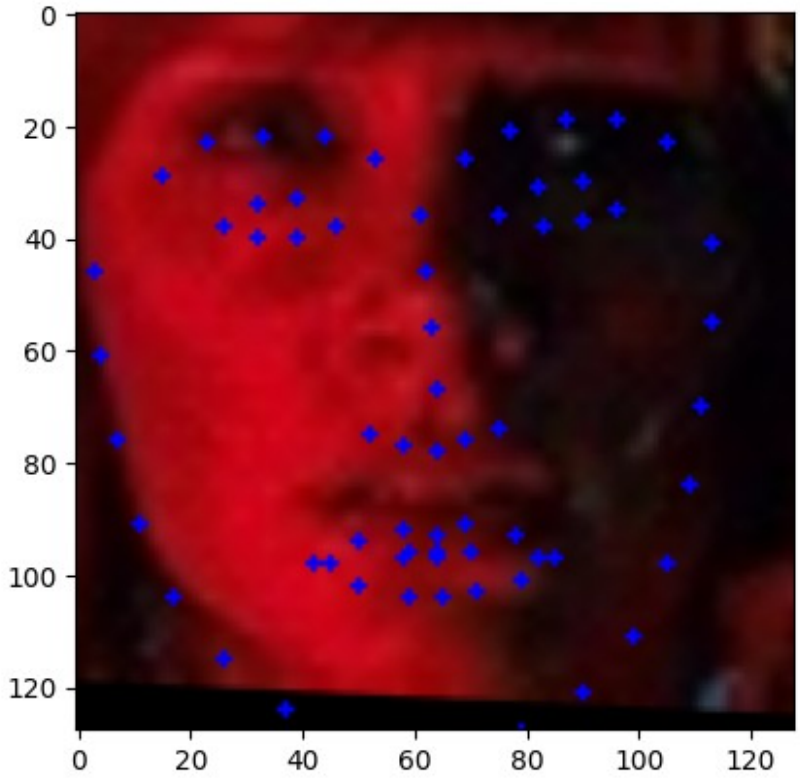

0
20
40
60
80
100
120
0
20
40
60
80
100
120

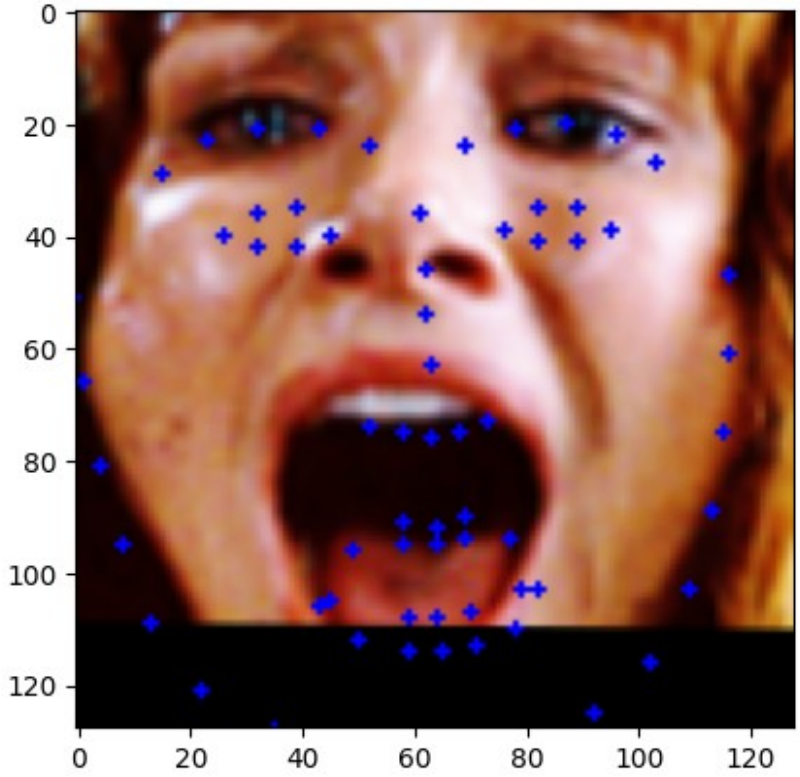

0
20
40
60
80
100
120
0
20
40
60
80
100
120

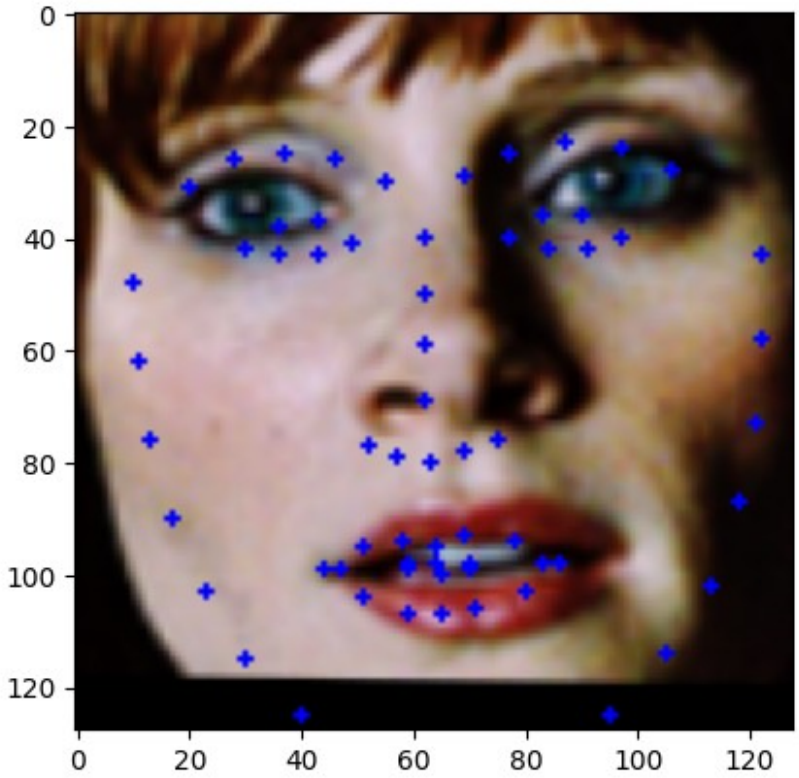

0
20
40
60
80
100
120
0
20
40
60
80
100
120

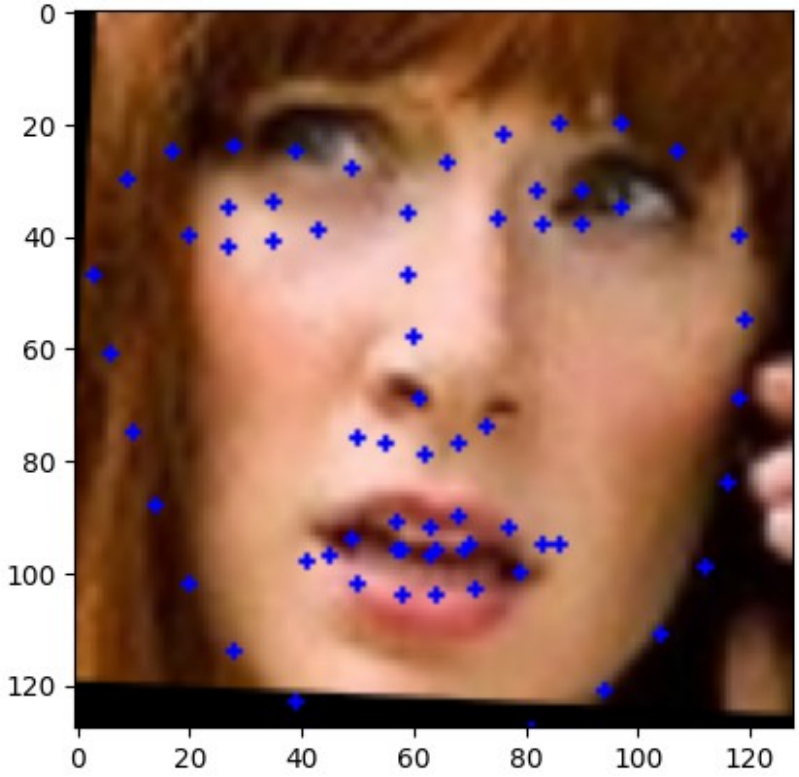

0
20
40
60
80
100
120
0
20
40
60
80
100
120

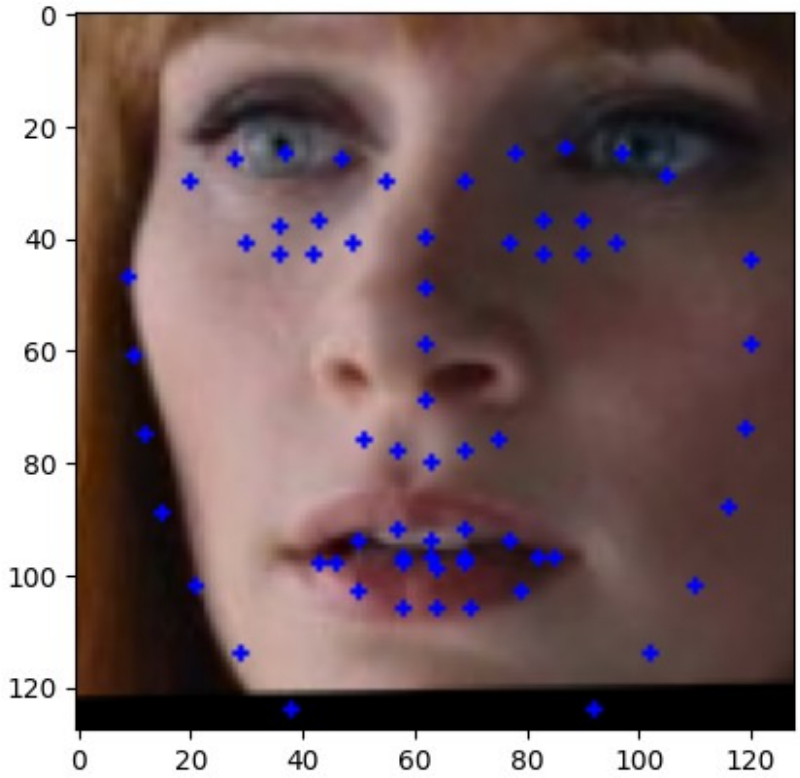

0
20
40
60
80
100
120
0
20
40
60
80
100
120

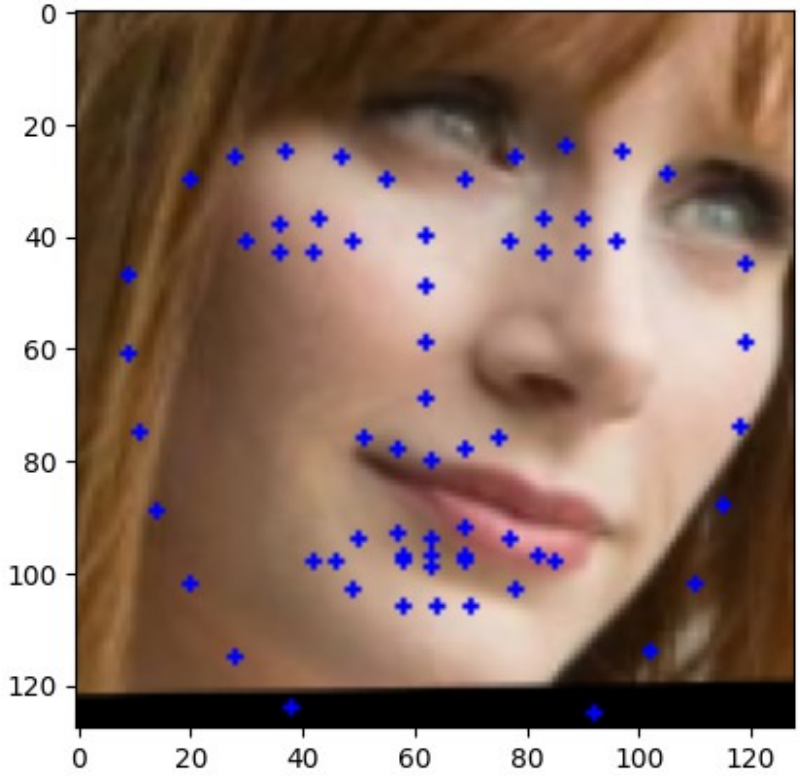

0
20
40
60
80
100
120
0
20
40
60
80
100
120

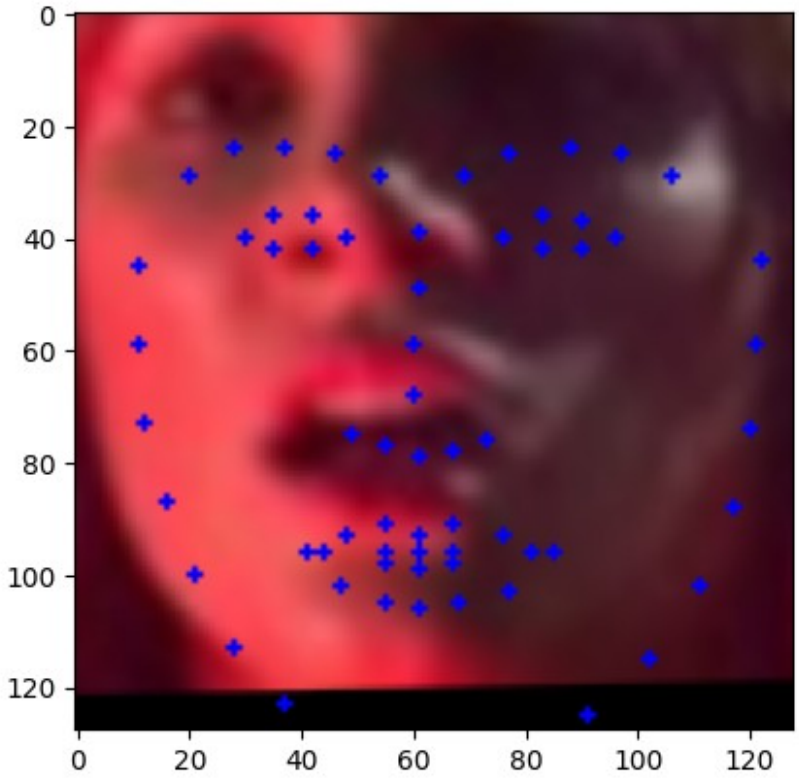

0
20
40
60
80
100
120
0
20
40
60
80
100
120

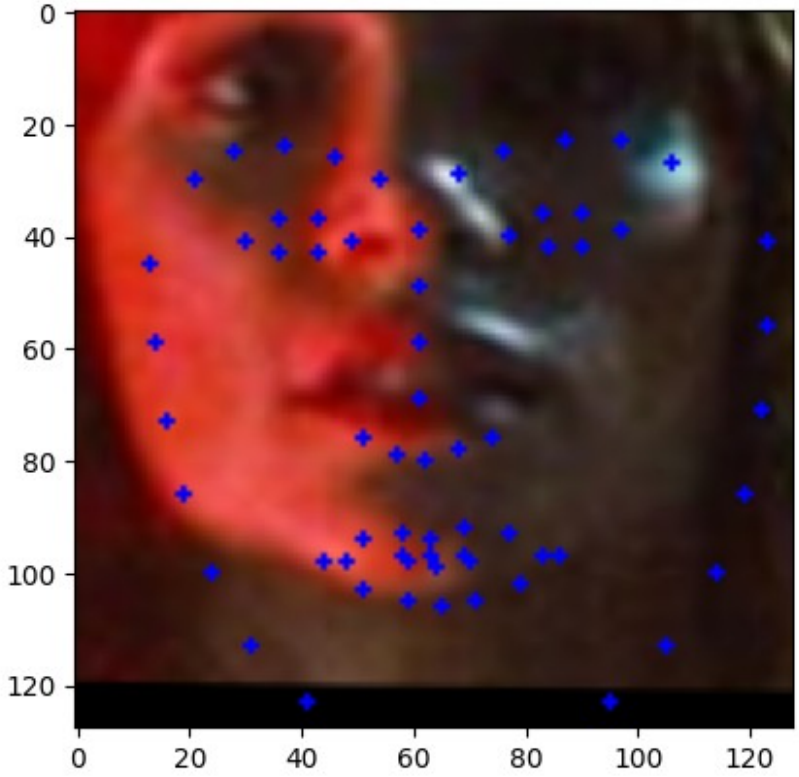

0
20
40
60
80
100
120
0
20
40
60
80
100
120

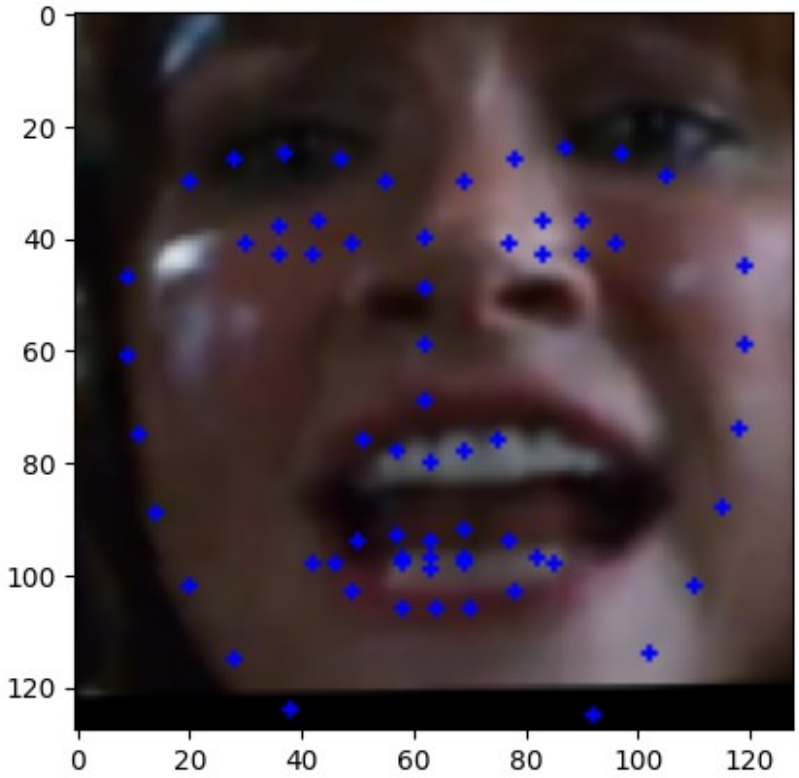

0
20
40
60
80
100
120
0
20
40
60
80
100
120

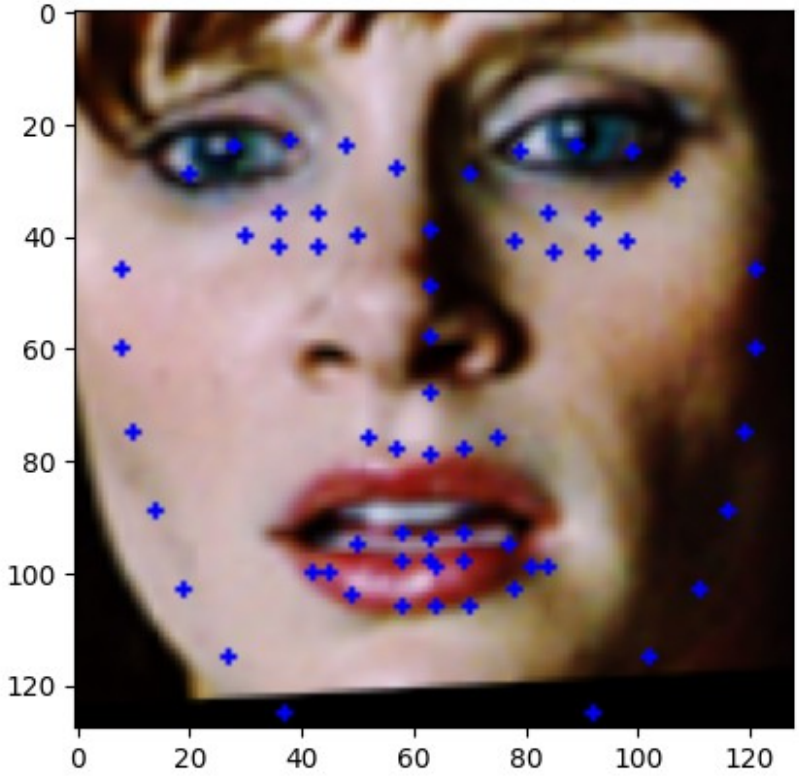

0
20
40
60
80
100
120
0
20
40
60
80
100
120

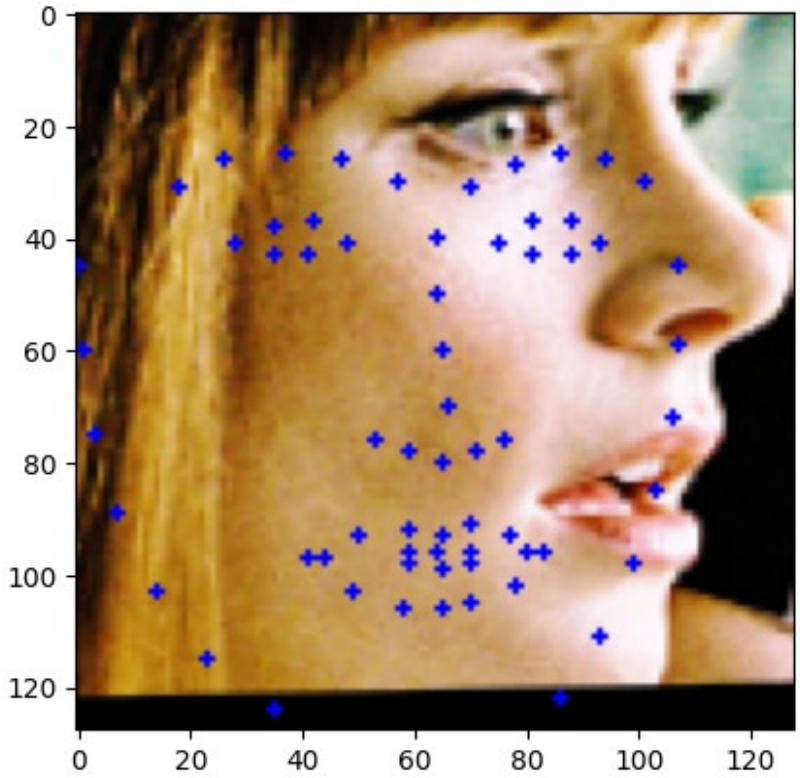

0
20
40
60
80
100
120
0
20
40
60
80
100
120

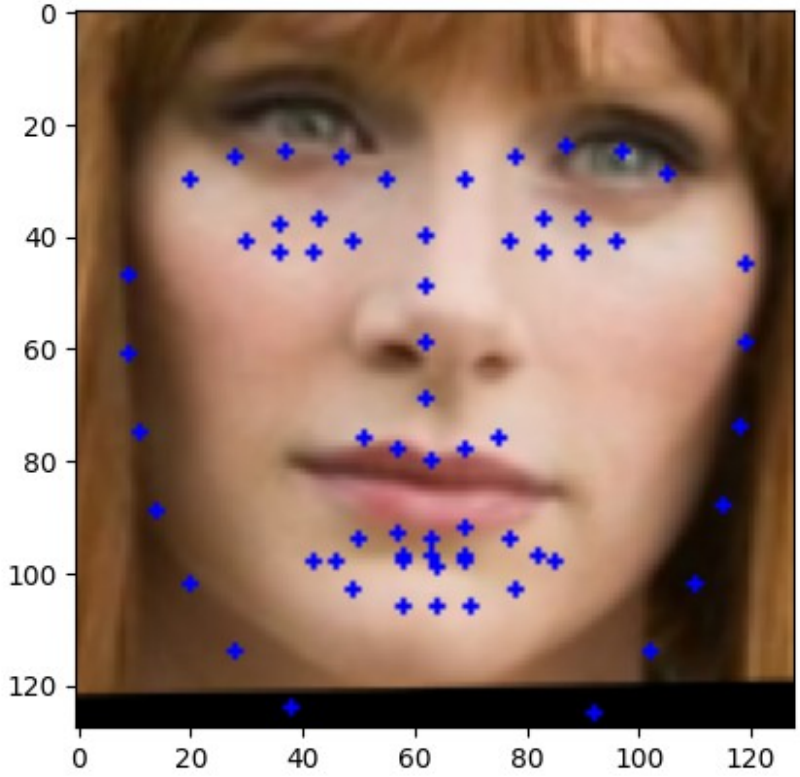

0
20
40
60
80
100
120
0
20
40
60
80
100
120

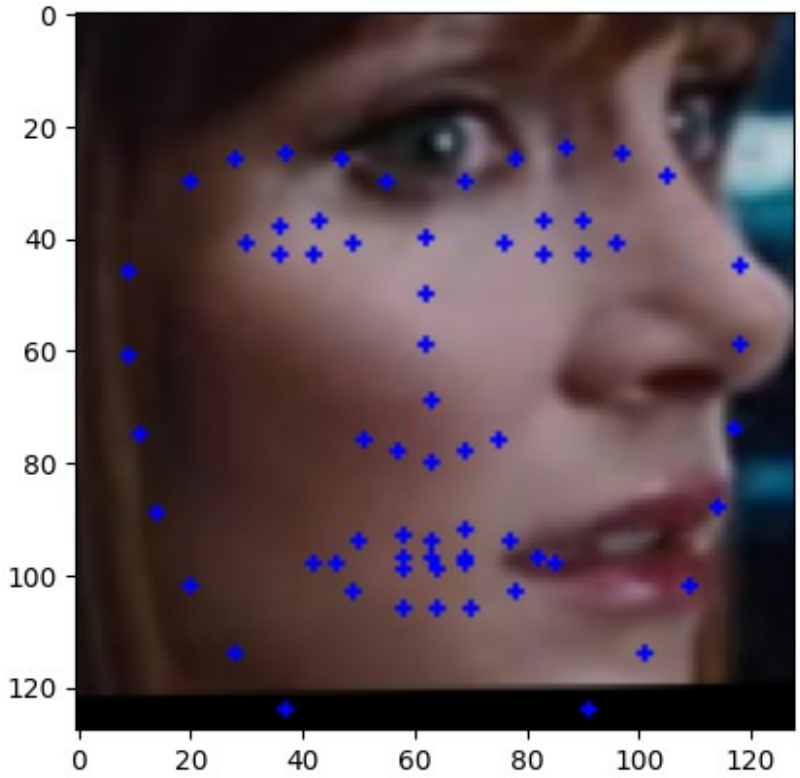

0
20
40
60
80
100
120
0
20
40
60
80
100
120

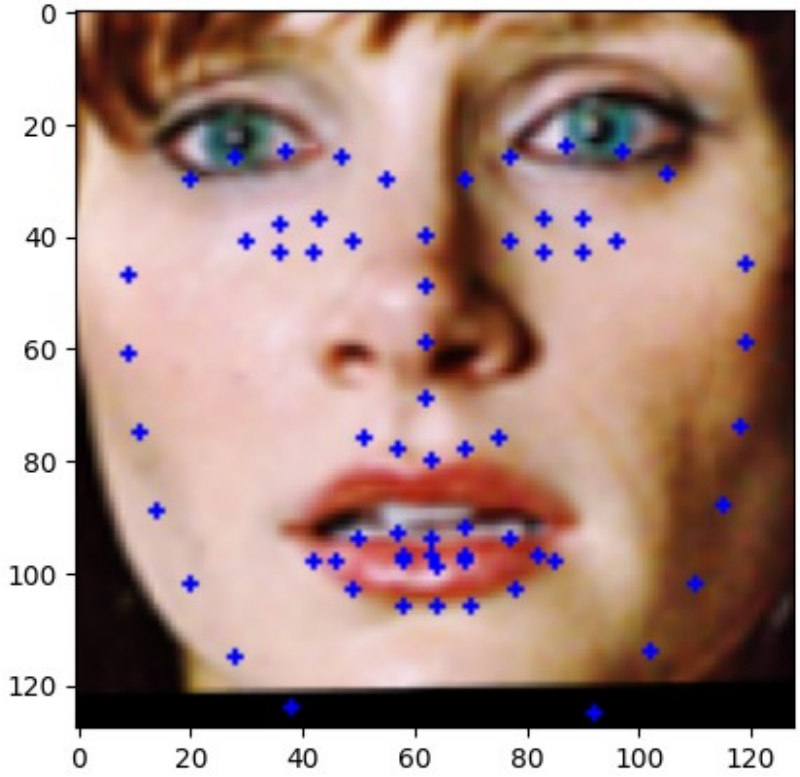

0
20
40
60
80
100
120
0
20
40
60
80
100
120

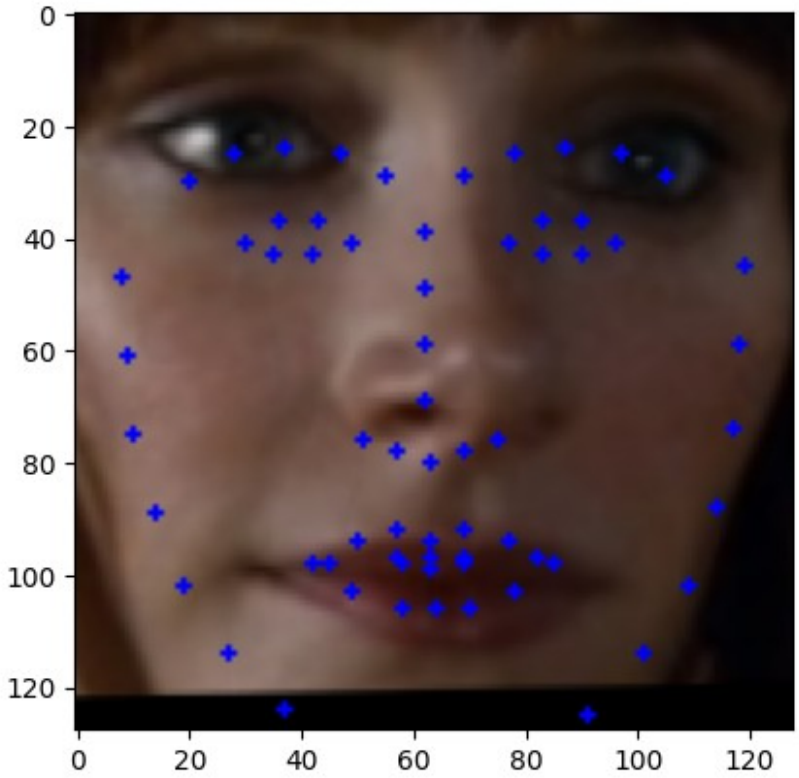

0
20
40
60
80
100
120
0
20
40
60
80
100
120

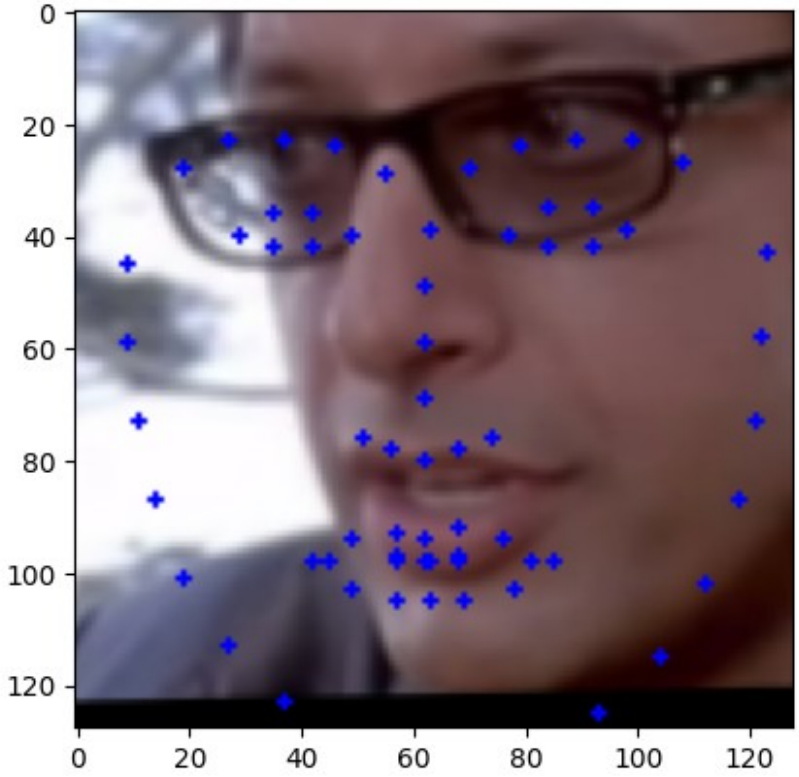

0
20
40
60
80
100
120
0
20
40
60
80
100
120

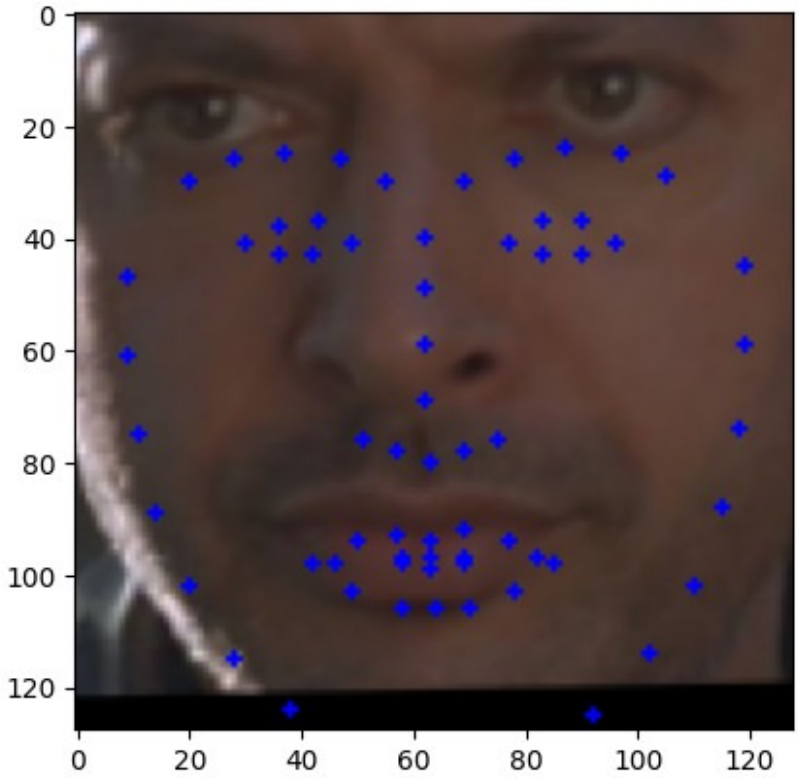

0
20
40
60
80
100
120
0
20
40
60
80
100
120

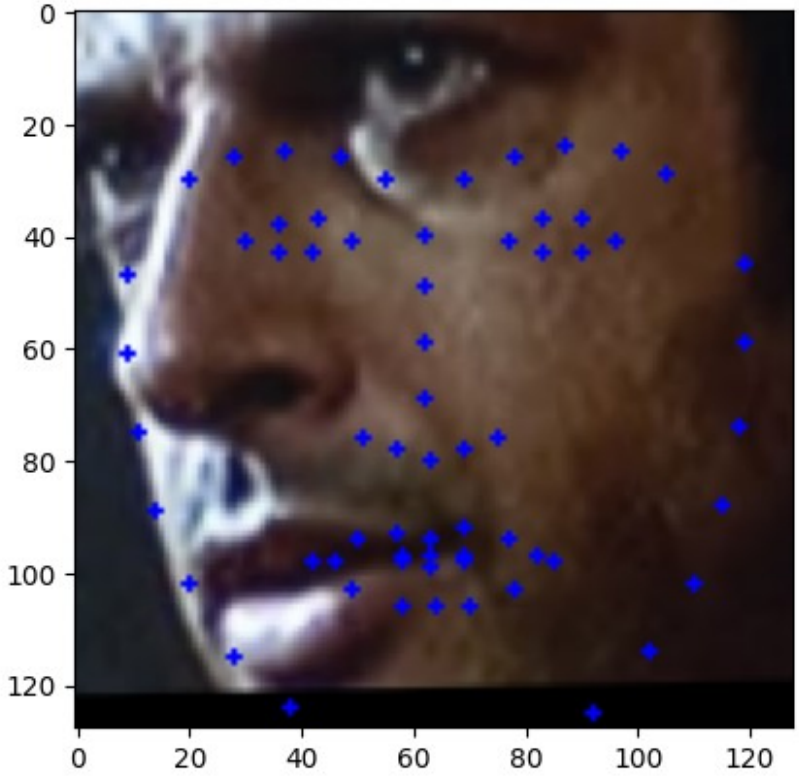

0
20
40
60
80
100
120
0
20
40
60
80
100
120

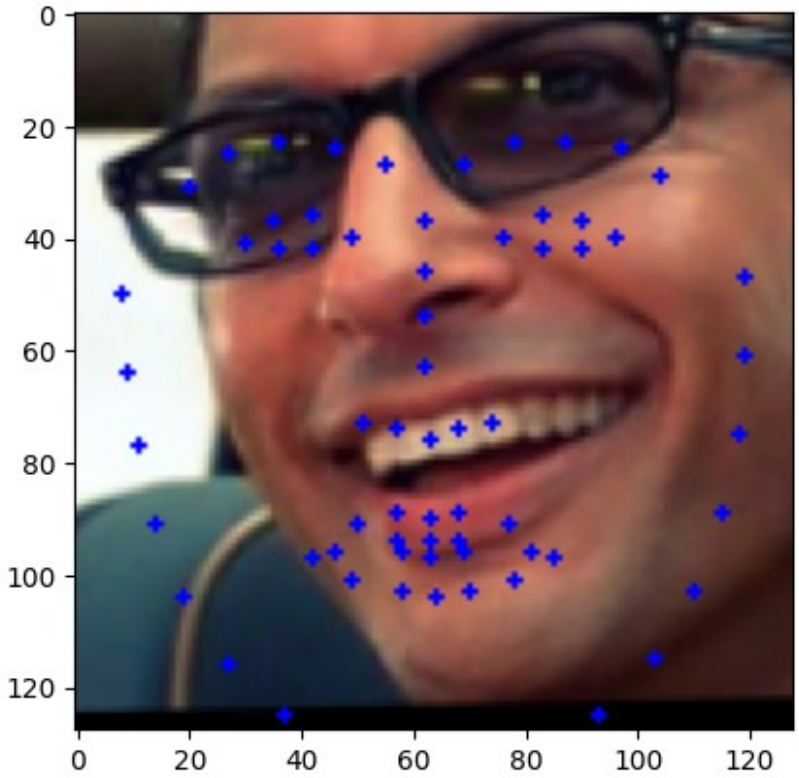

0
20
40
60
80
100
120
0
20
40
60
80
100
120

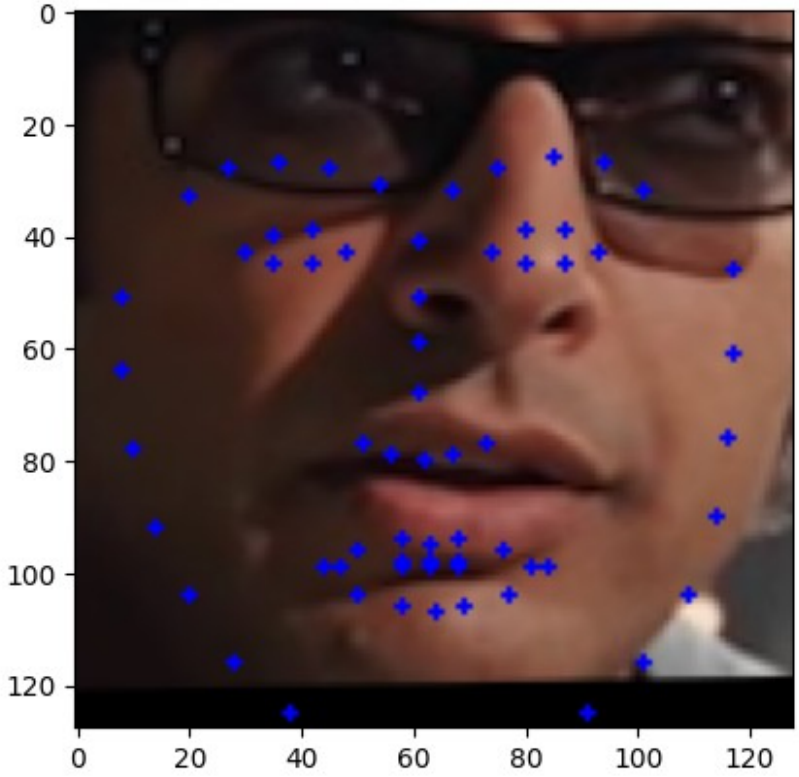

0
20
40
60
80
100
120
0
20
40
60
80
100
120

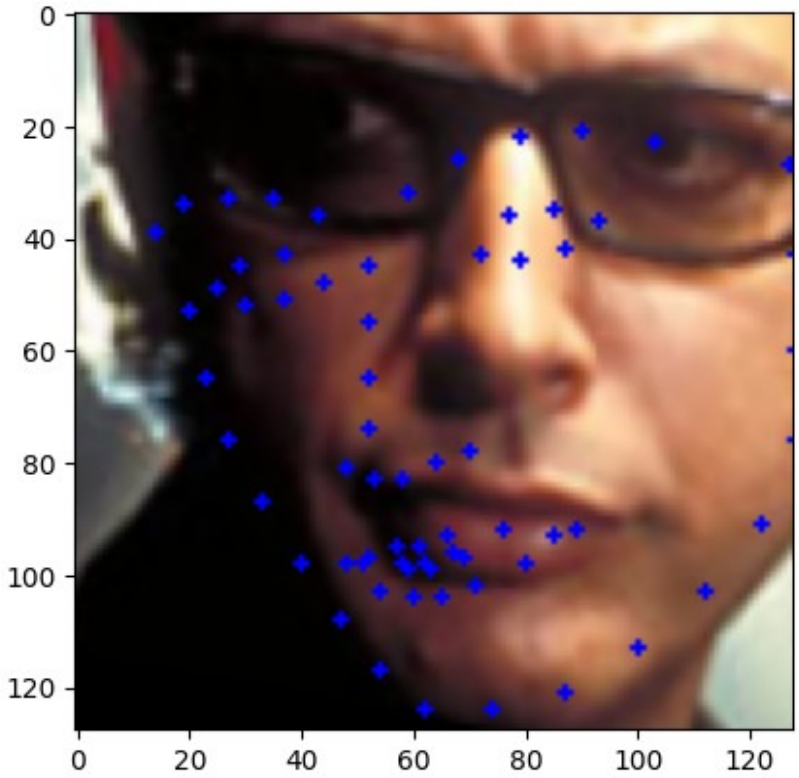

0
20
40
60
80
100
120
0
20
40
60
80
100
120

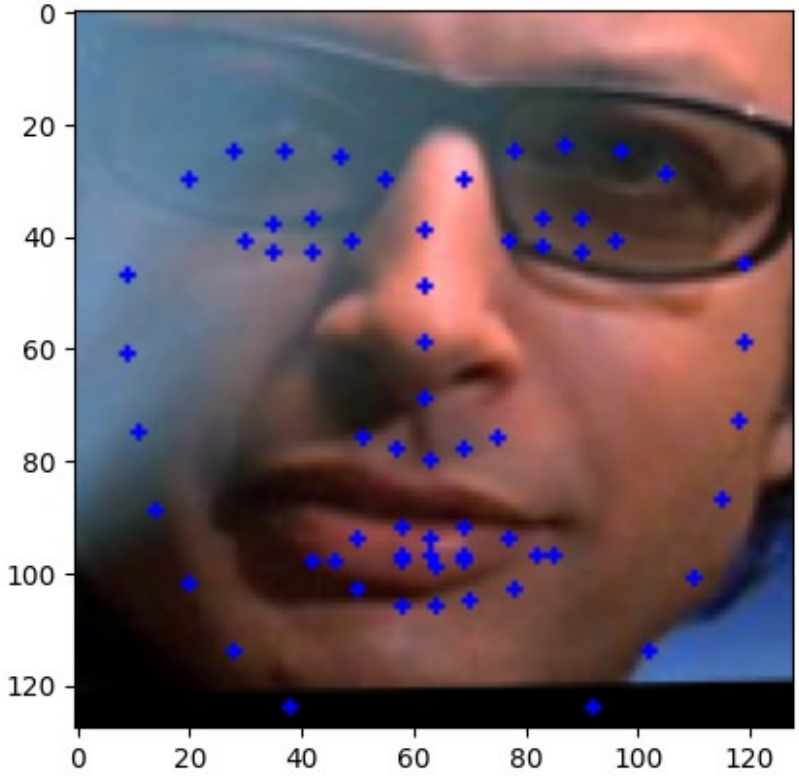

0
20
40
60
80
100
120
0
20
40
60
80
100
120

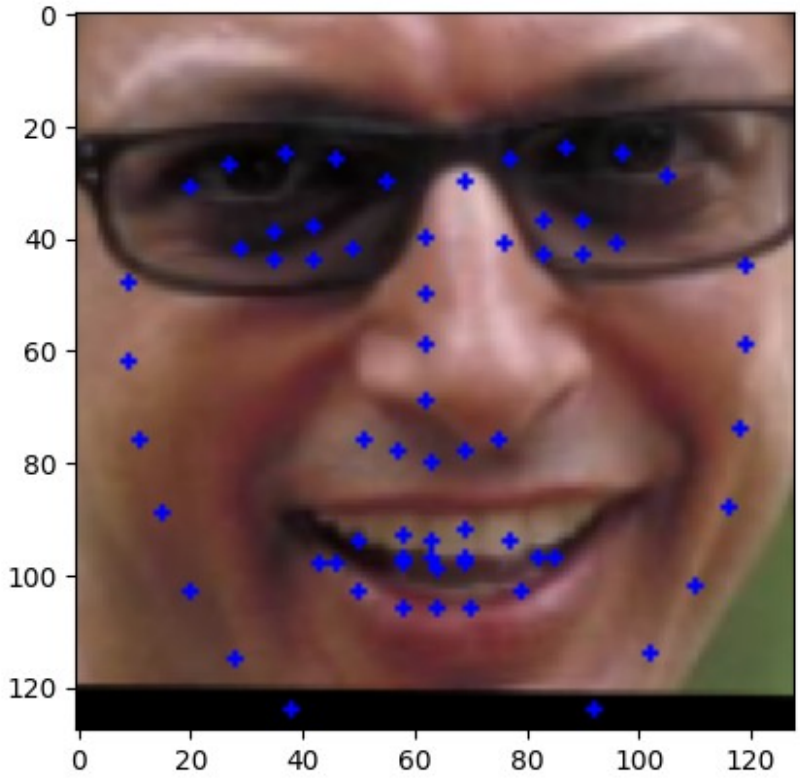

0
20
40
60
80
100
120
0
20
40
60
80
100
120

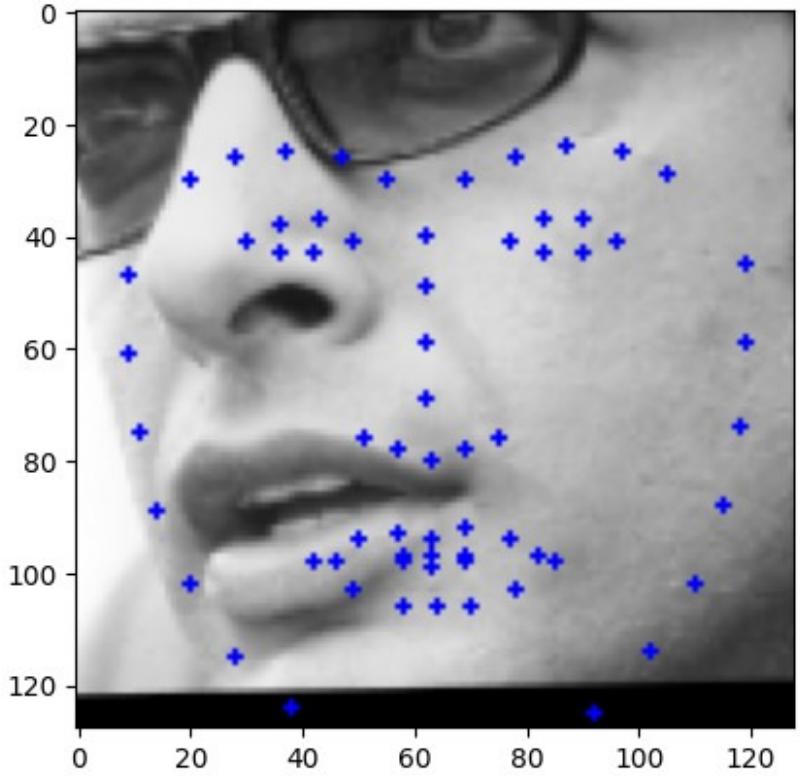

0
20
40
60
80
100
120
0
20
40
60
80
100
120

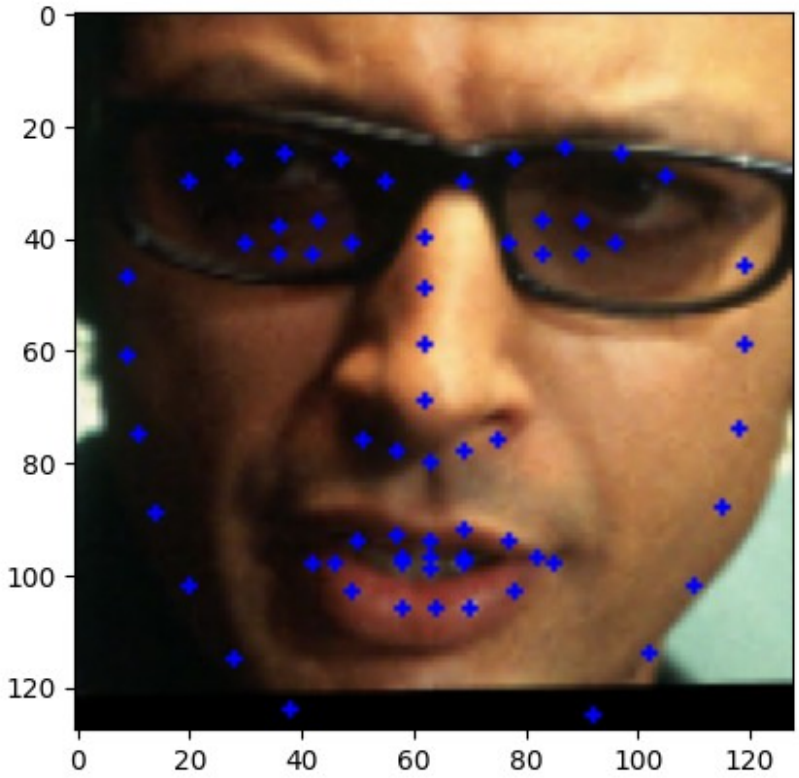

0
20
40
60
80
100
120
0
20
40
60
80
100
120

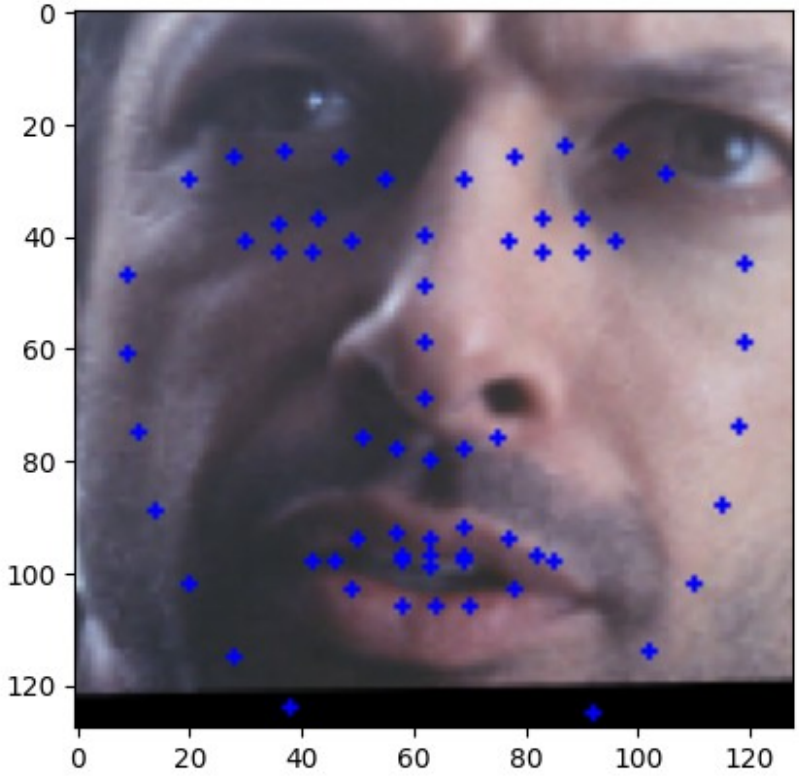

0
20
40
60
80
100
120
0
20
40
60
80
100
120

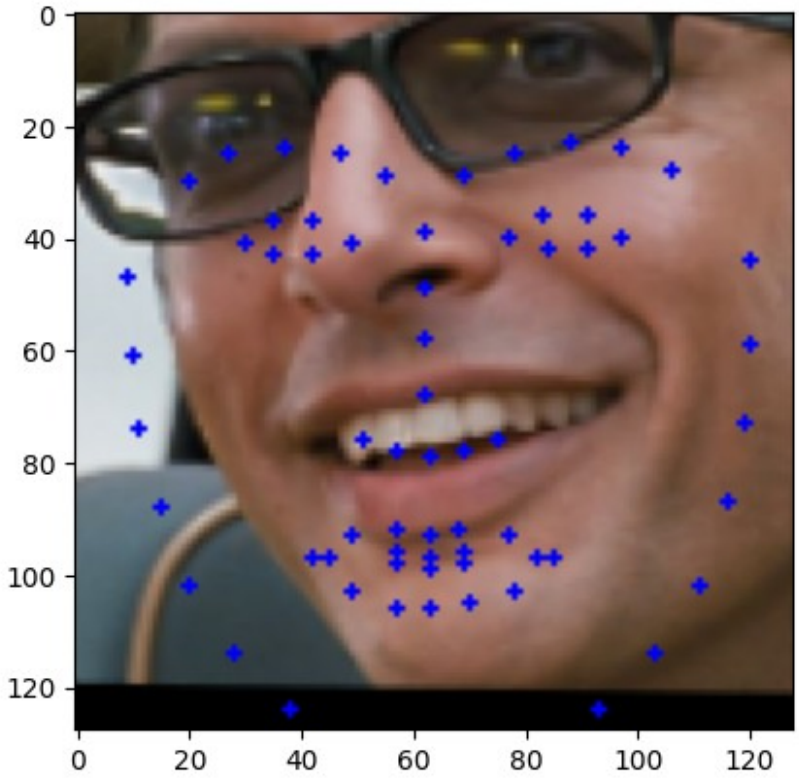

0
20
40
60
80
100
120
0
20
40
60
80
100
120

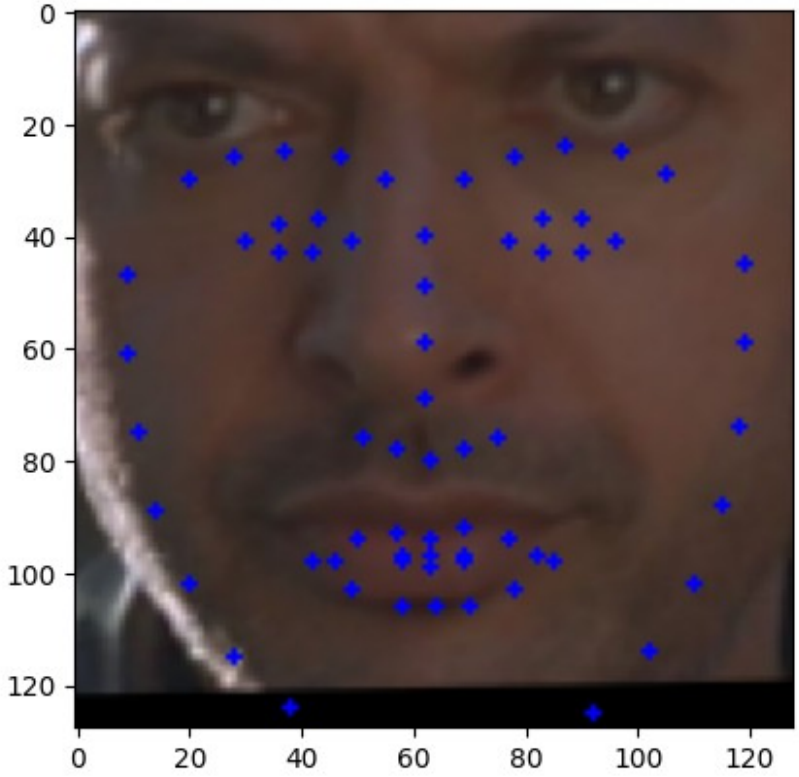

0
20
40
60
80
100
120
0
20
40
60
80
100
120

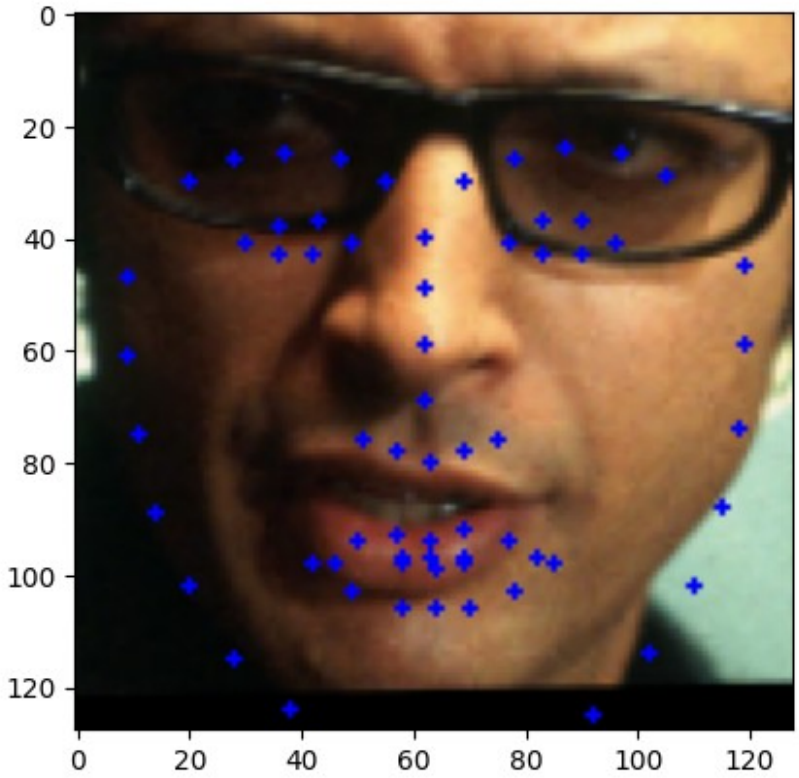

0
20
40
60
80
100
120
0
20
40
60
80
100
120

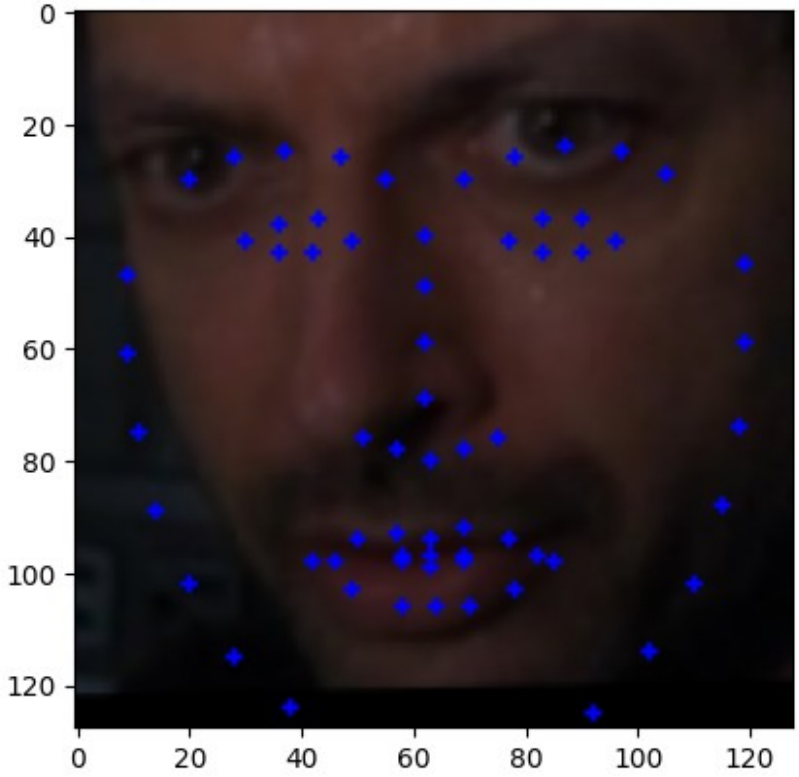

0
20
40
60
80
100
120
0
20
40
60
80
100
120

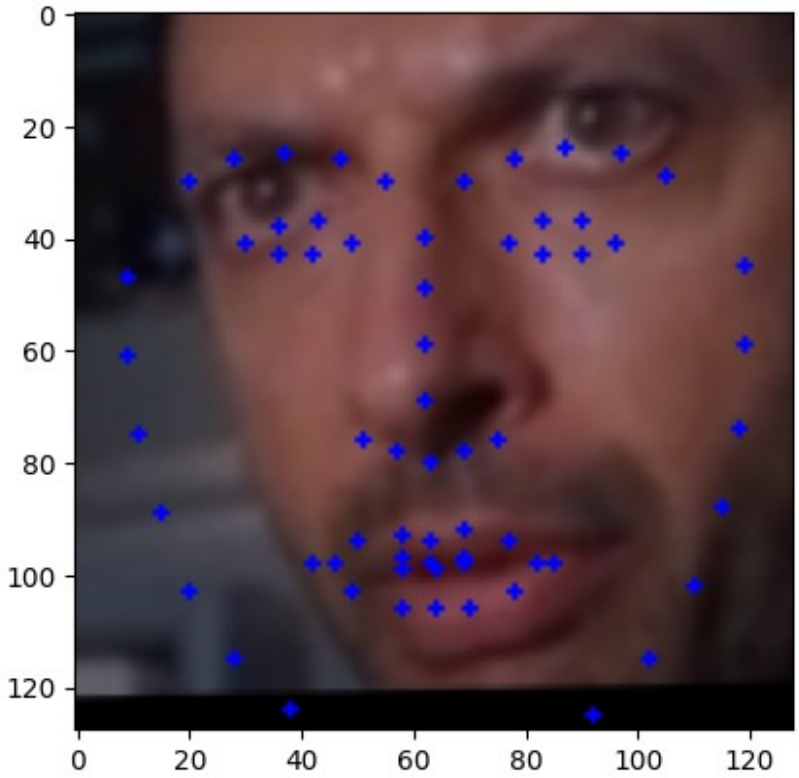

0
20
40
60
80
100
120
0
20
40
60
80
100
120

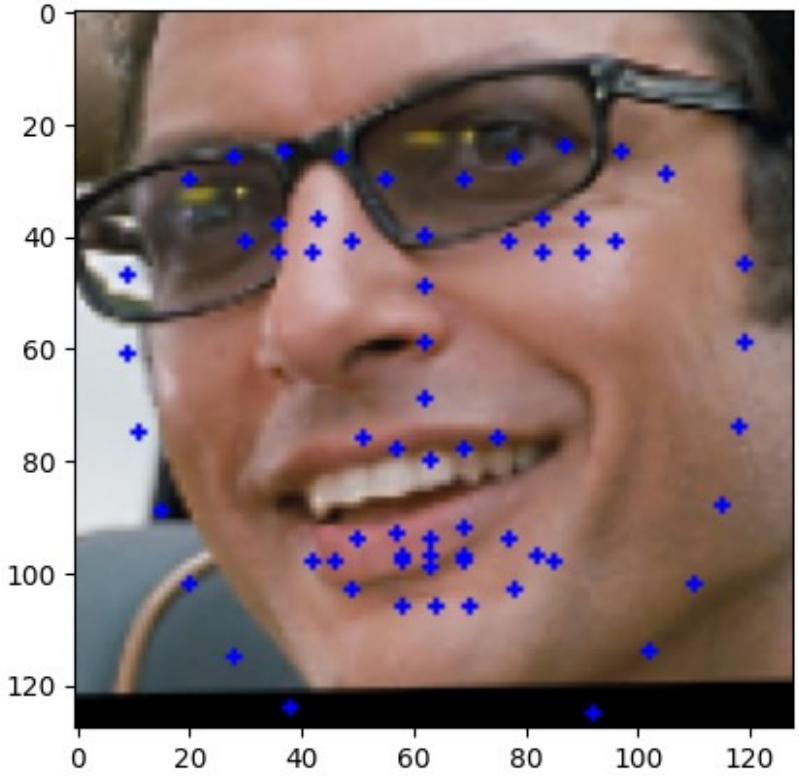

0
20
40
60
80
100
120
0
20
40
60
80
100
120

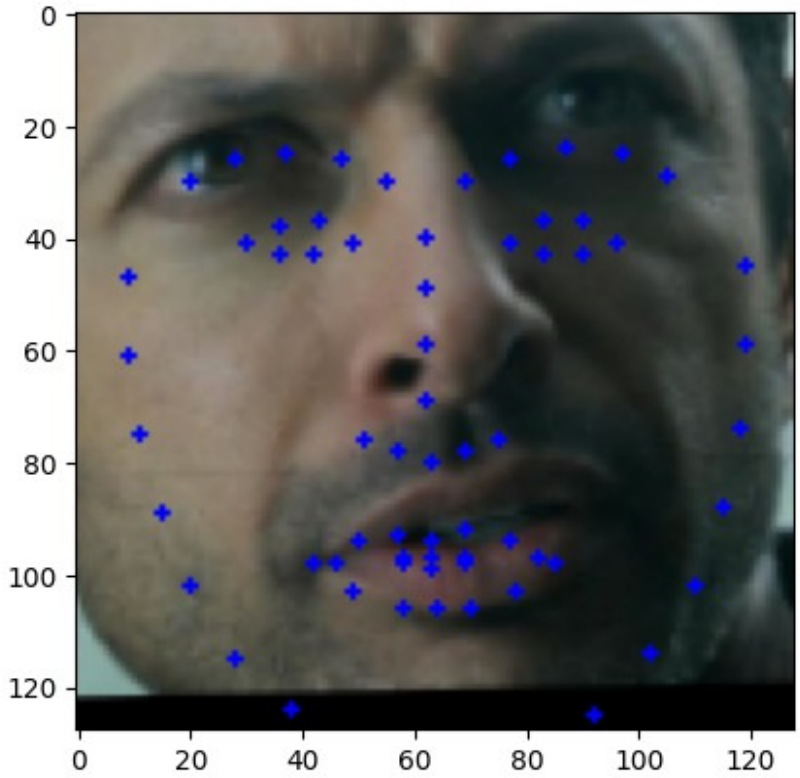

0
20
40
60
80
100
120
0
20
40
60
80
100
120

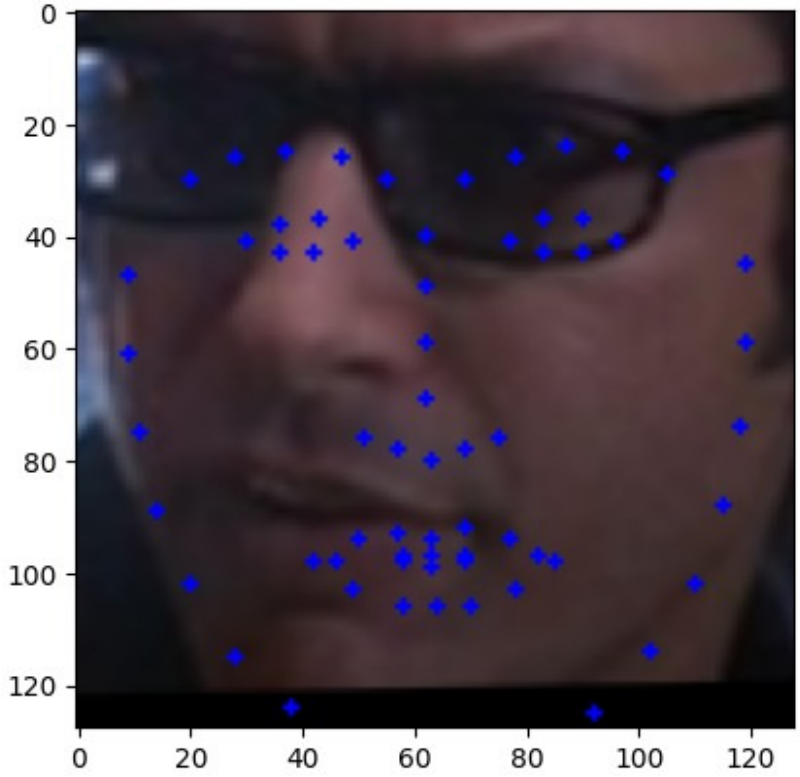

0
20
40
60
80
100
120
0
20
40
60
80
100
120

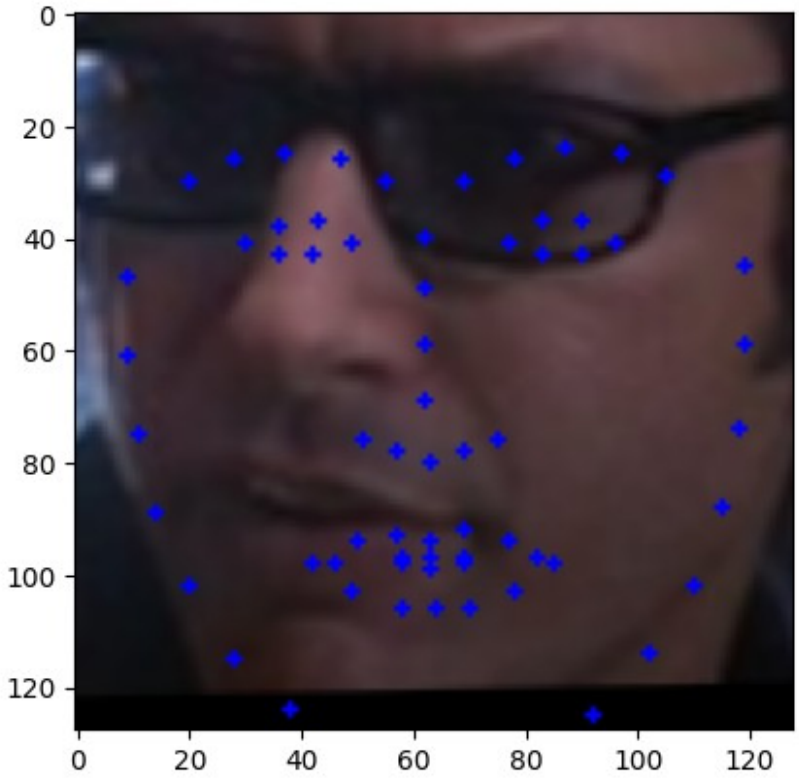

0
20
40
60
80
100
120
0
20
40
60
80
100
120

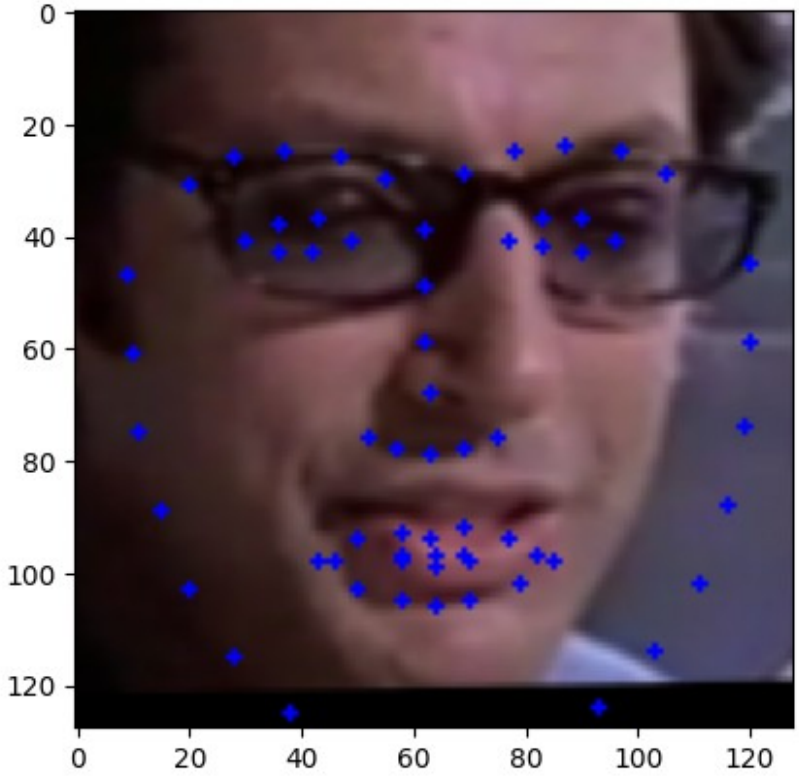

0
20
40
60
80
100
120
0
20
40
60
80
100
120

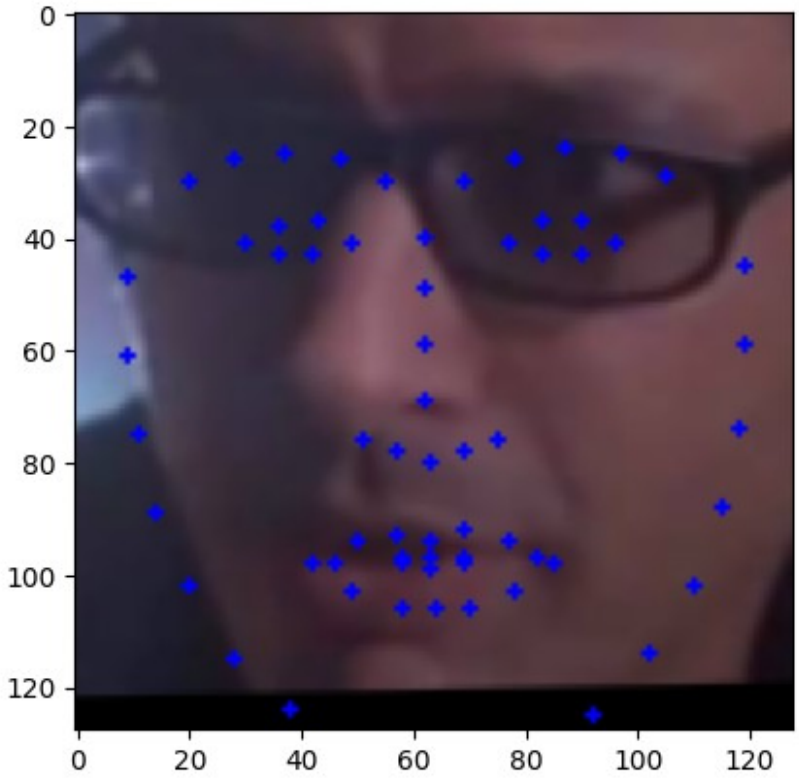

0
20
40
60
80
100
120
0
20
40
60
80
100
120

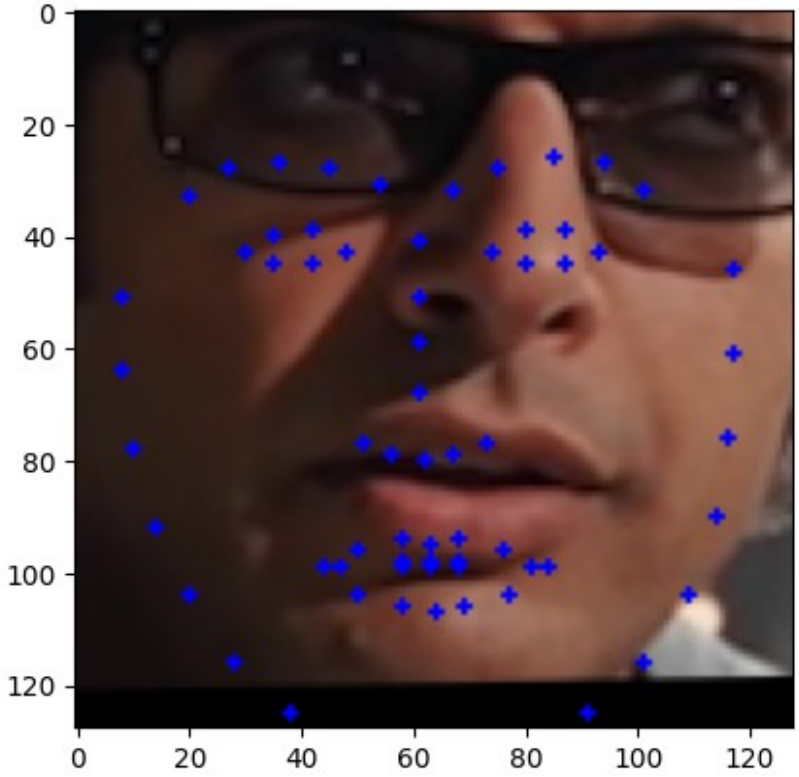

0
20
40
60
80
100
120
0
20
40
60
80
100
120

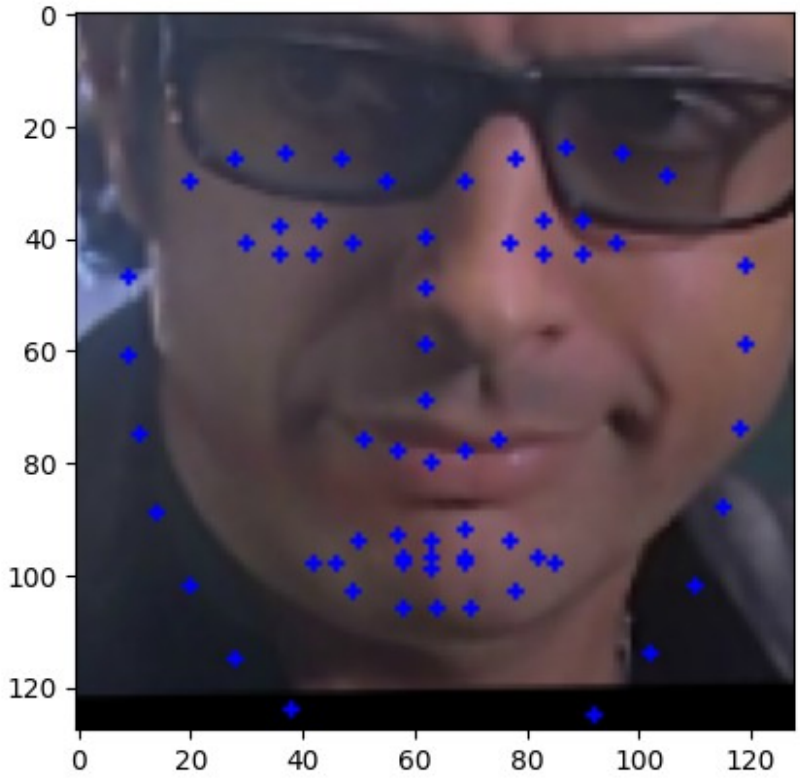

0
20
40
60
80
100
120
0
20
40
60
80
100
120

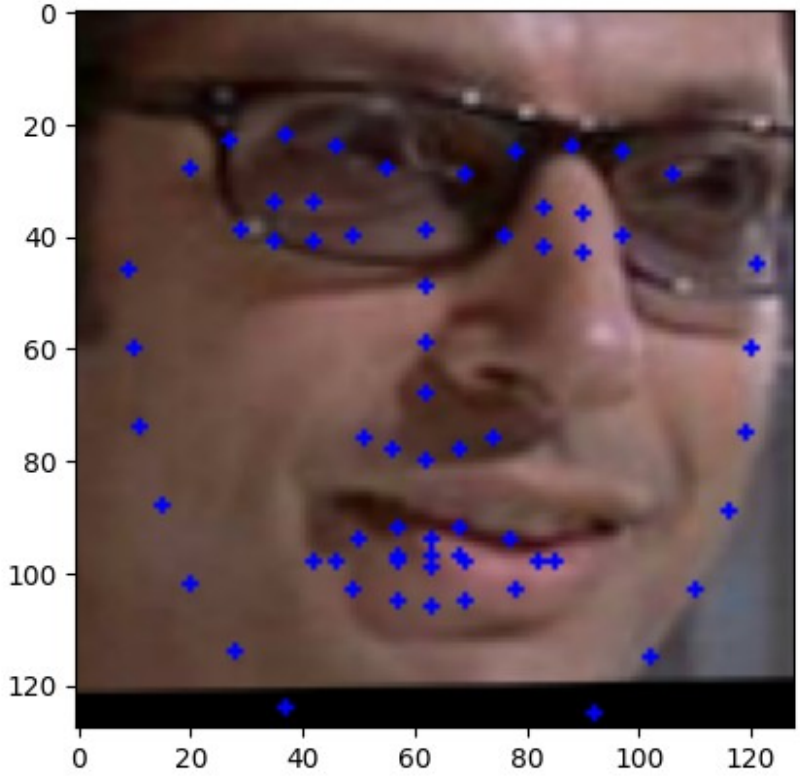

0
20
40
60
80
100
120
0
20
40
60
80
100
120

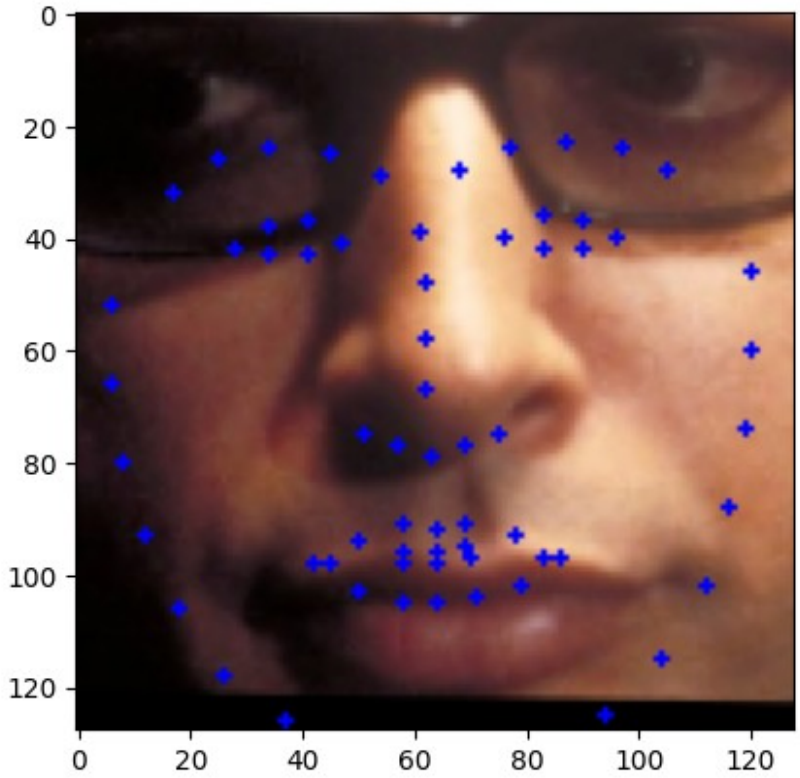

0
20
40
60
80
100
120
0
20
40
60
80
100
120

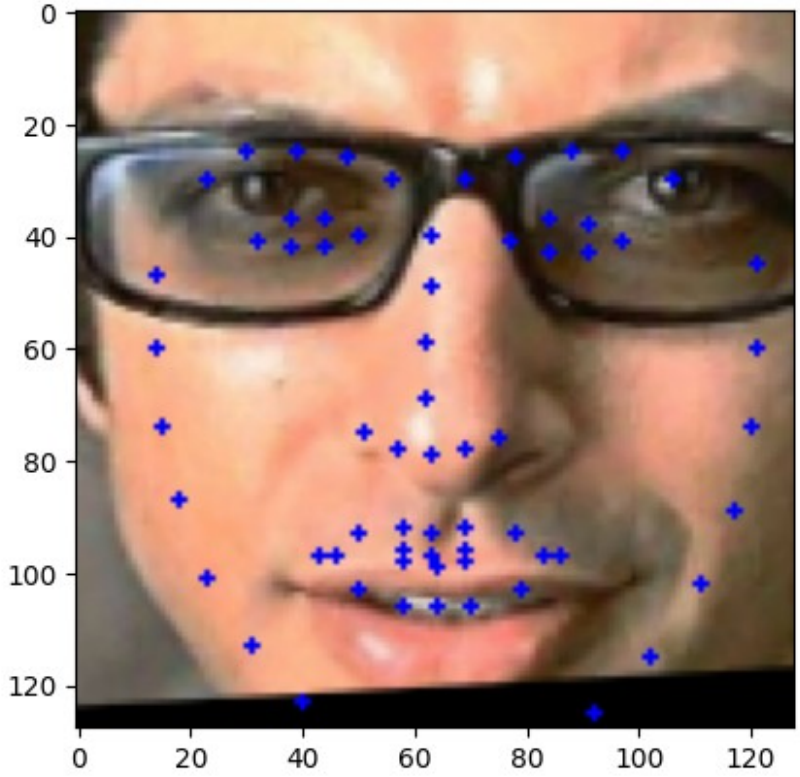

0
20
40
60
80
100
120
0
20
40
60
80
100
120

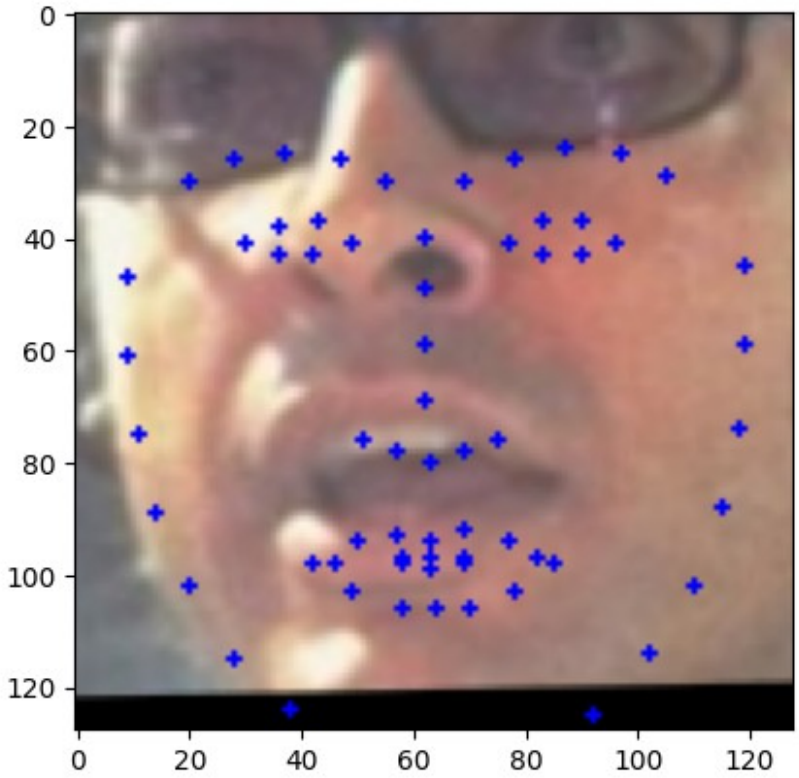

0
20
40
60
80
100
120
0
20
40
60
80
100
120

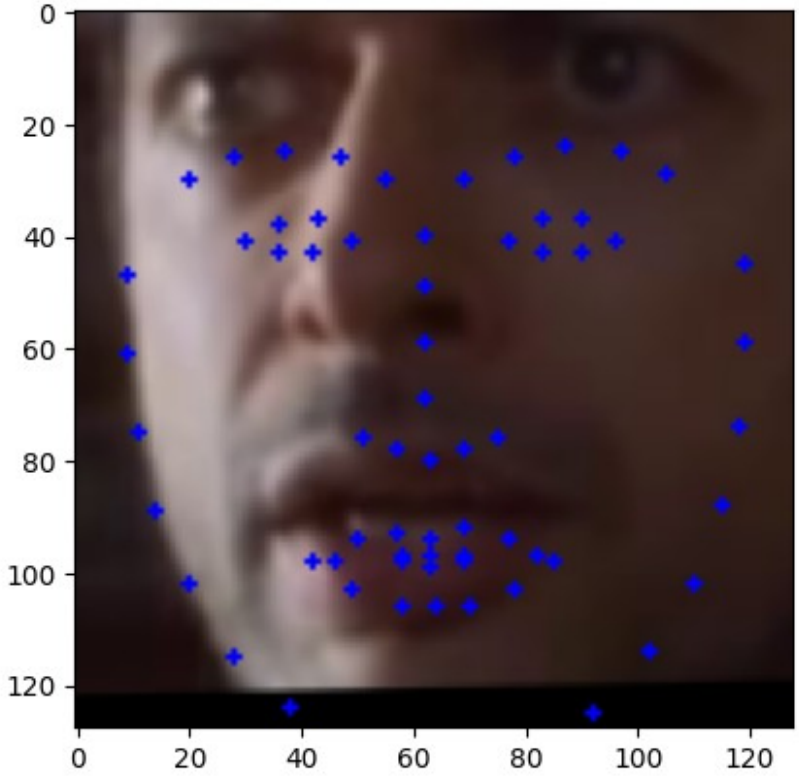

0
20
40
60
80
100
120
0
20
40
60
80
100
120

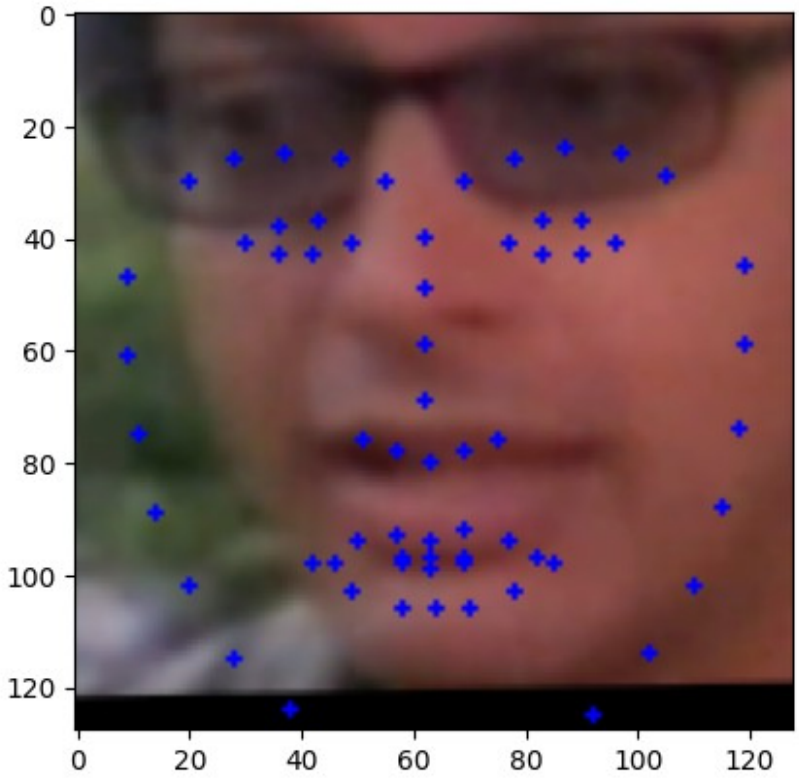

0
20
40
60
80
100
120
0
20
40
60
80
100
120

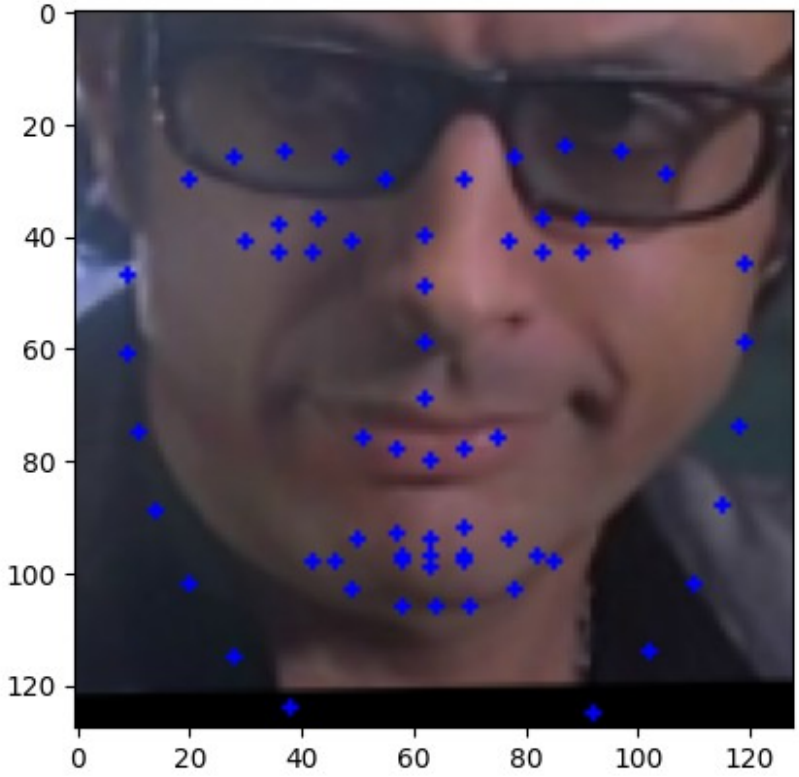

0
20
40
60
80
100
120
0
20
40
60
80
100
120

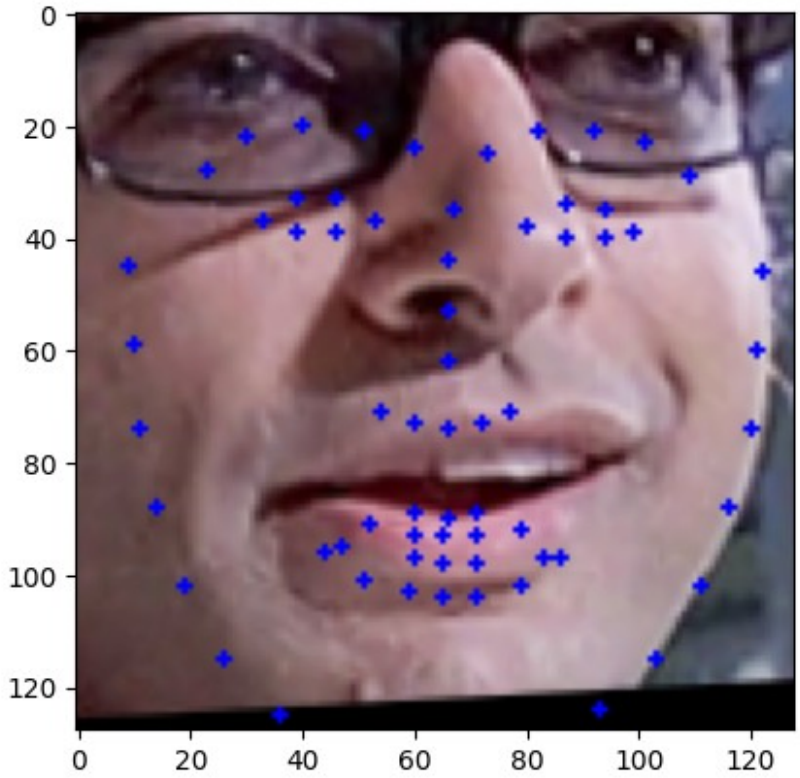

0
20
40
60
80
100
120
0
20
40
60
80
100
120

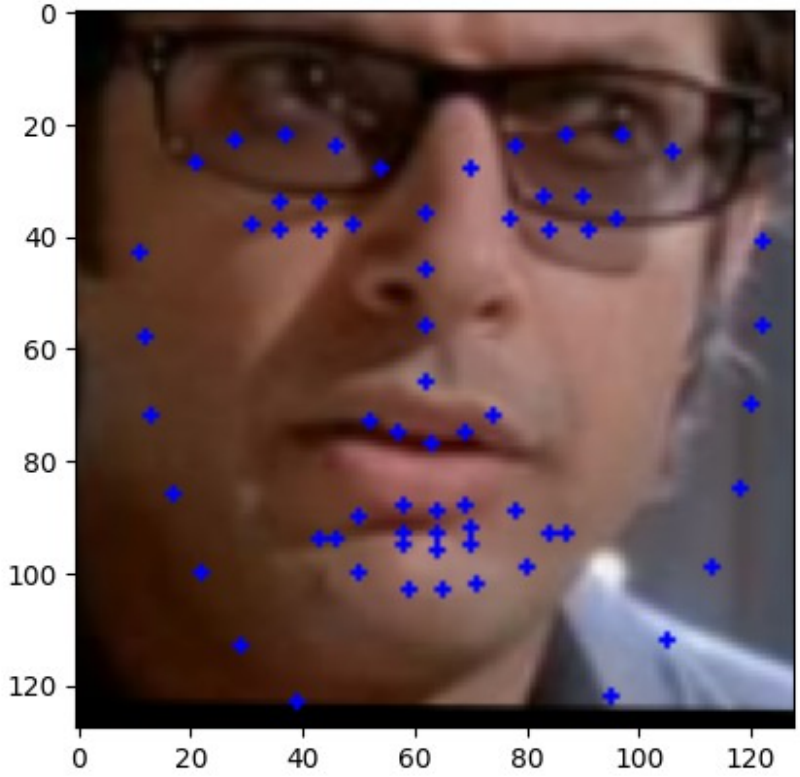

0
20
40
60
80
100
120
0
20
40
60
80
100
120

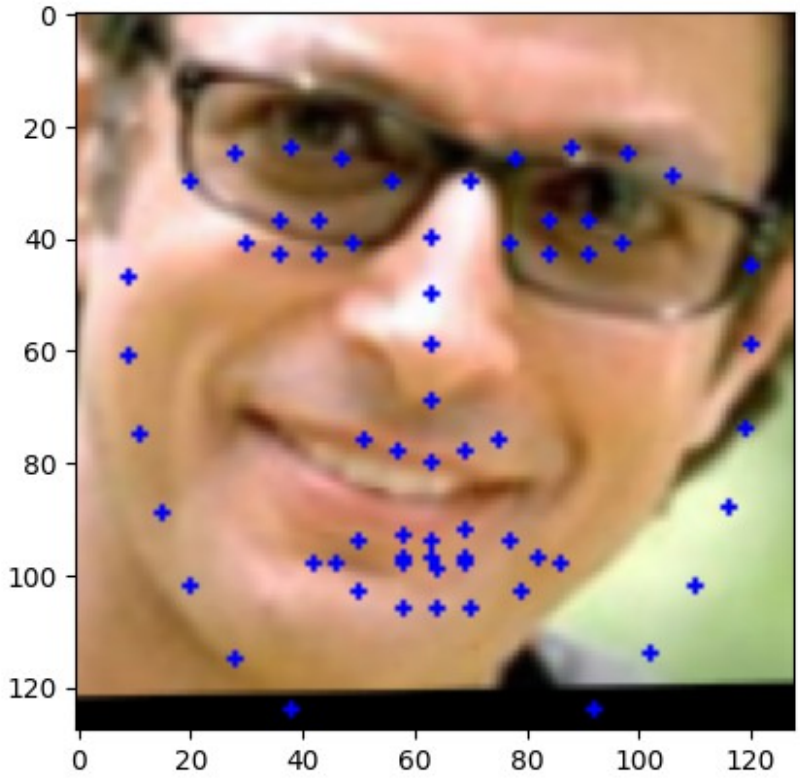

0
20
40
60
80
100
120
0
20
40
60
80
100
120

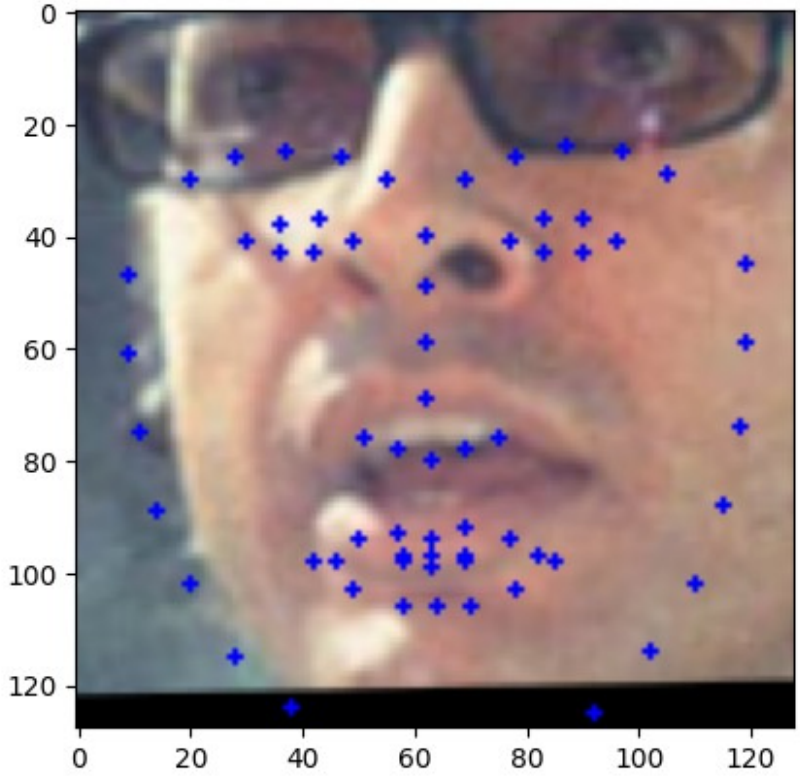

0
20
40
60
80
100
120
0
20
40
60
80
100
120

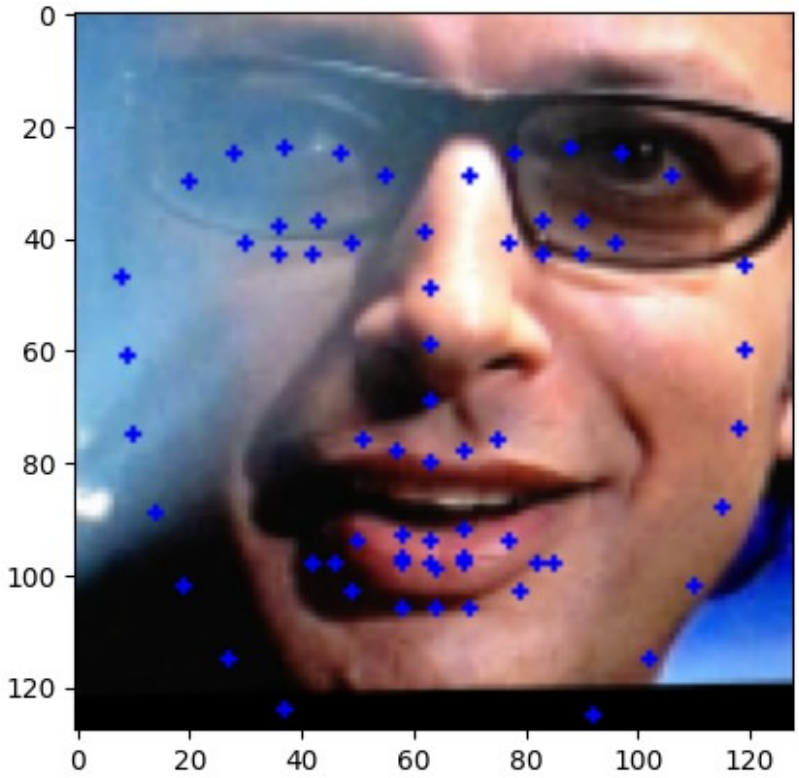

0
20
40
60
80
100
120
0
20
40
60
80
100
120

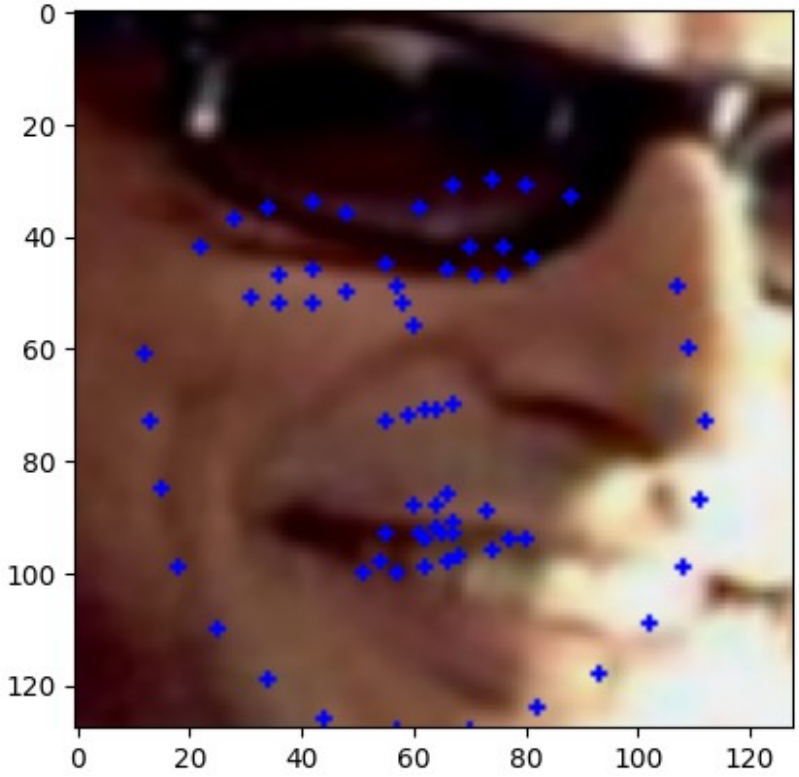

0
20
40
60
80
100
120
0
20
40
60
80
100
120

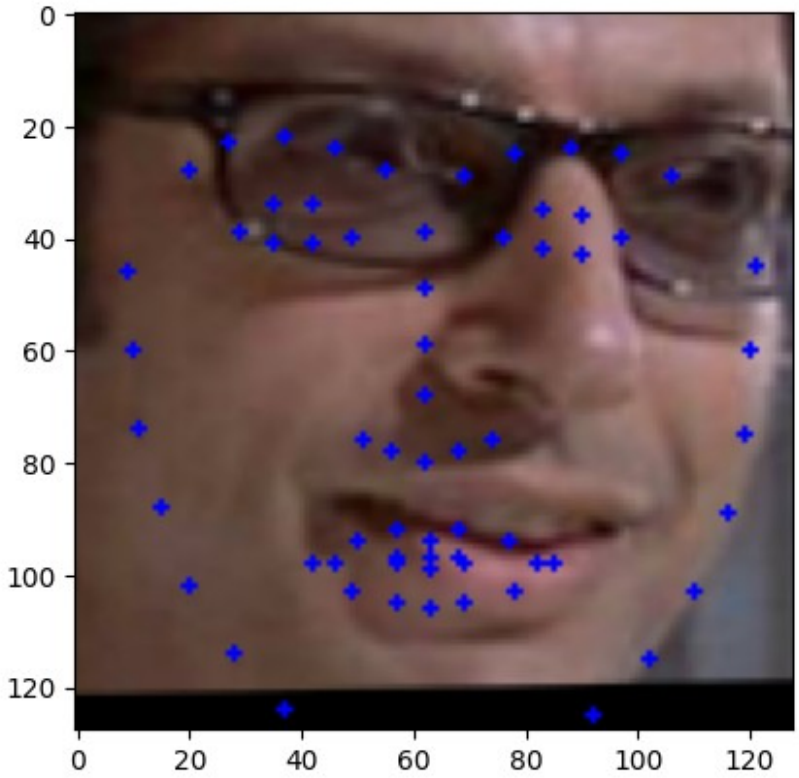

0
20
40
60
80
100
120
0
20
40
60
80
100
120

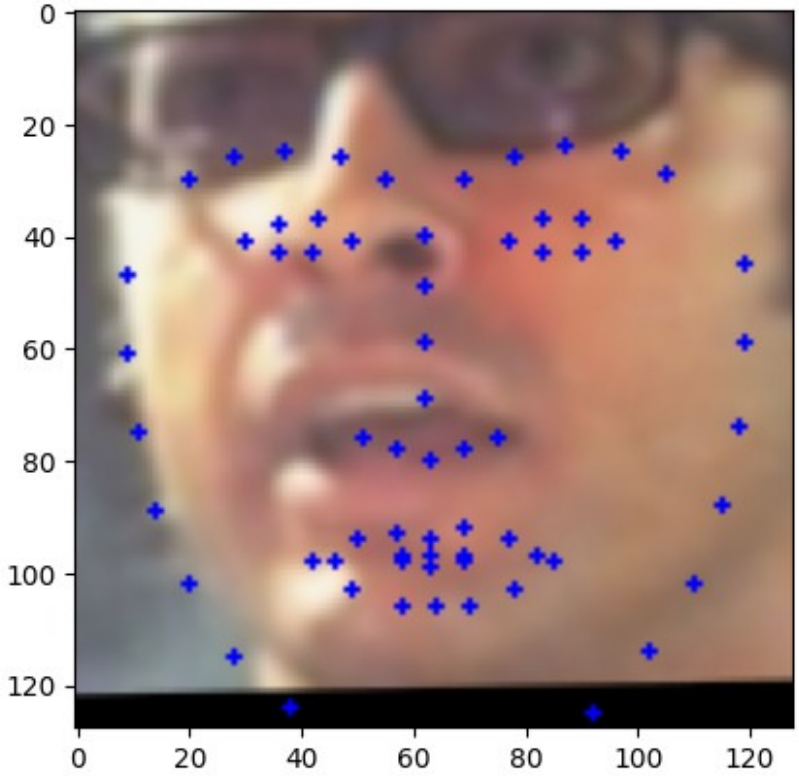

0
20
40
60
80
100
120
0
20
40
60
80
100
120

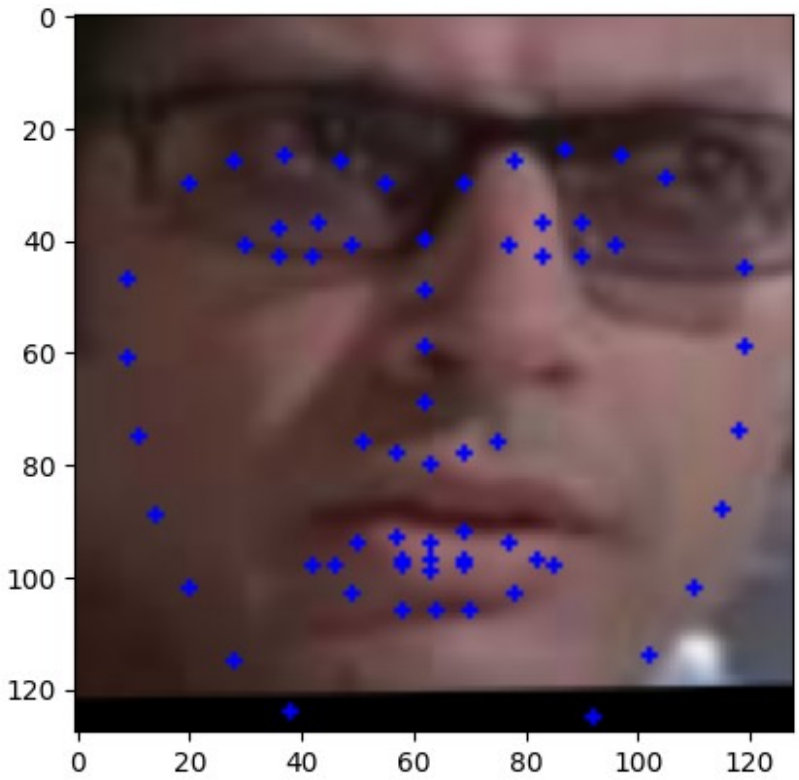

0
20
40
60
80
100
120
0
20
40
60
80
100
120


0
20
40
60
80
100
120
0
20
40
60
80
100
120


0
20
40
60
80
100
120
0
20
40
60
80
100
120


0
20
40
60
80
100
120
0
20
40
60
80
100
120


0
20
40
60
80
100
120
0
20
40
60
80
100
120




```
X_train_aligned, X_validation_aligned, Y_train_aligned,
Y_validation_aligned = train_test_split(aligned_faces, list_labels,
test_size=0.3, random_state=42)

# Création d'une instance de LabelBinarizer pour l'encodage
encoder = LabelBinarizer()

# Encodage des étiquettes d'entraînement en format binaire
# La méthode fit_transform ajuste le transformateur sur Y_train et
encode simultanément les étiquettes
Y_train_encoded_aligned = encoder.fit_transform(Y_train_aligned)

# Encodage des étiquettes de test en format binaire
# La méthode transform utilise les informations d'encodage apprises
sur Y_train pour encoder Y_test
Y_validation_encoded_aligned = encoder.transform(Y_validation_aligned)

train_generator_aligned =
datagen.flow(np.array(X_train_aligned),Y_train_encoded_aligned,
batch_size=32)

# Define the network
model = models.Sequential()

model.add(layers.Conv2D(32, (3, 3), activation='relu',
input_shape=(128, 128, 3)))
```

```python
model.add(layers.MaxPooling2D(2,2))
model.add(layers.Conv2D(64, (3,3), activation = 'relu'))
model.add(layers.MaxPooling2D(2,2))
model.add(layers.Conv2D(128, (3,3), activation = 'relu'))
model.add(layers.MaxPooling2D(2,2))
model.add(layers.Conv2D(128, (3,3), activation = 'relu'))
#model.add(layers.Dropout(0.5))
model.add(layers.Flatten())
model.add(layers.Dense(512, activation = 'relu'))
model.add(layers.Dense(6, activation = 'softmax'))
model.summary()

model.compile(loss='categorical_crossentropy',
optimizer=optimizers.Adam(learning_rate=1e-3), metrics=['acc'])

# Train the network. Hint: use .fit(...)
history = model.fit(train_generator, epochs=70, validation_data =
(np.array(X_validation), Y_validation_encoded))

# Évaluation du modèle sur les données de validation et obtention de
la perte (loss) et de la précision (accuracy)
test_loss, test_acc = model.evaluate(np.array(X_validation),
Y_validation_encoded)

print('Validation accuracy: {:2.2f}%'.format(test_acc*100))
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/
convolutional/base_conv.py:107: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)

Model: "sequential_1"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┓
━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┩
━━━━━━━━━━━━━━━━━┩
│ conv2d_4 (Conv2D)                    │ (None, 126, 126, 32)        │
896 │
├──────────────────────────────────────┼─────────────────────────────┤
─────────────────┤
│ max_pooling2d_3 (MaxPooling2D)       │ (None, 63, 63, 32)          │
0 │
├──────────────────────────────────────┼─────────────────────────────┤
─────────────────┤
```

```
| conv2d_5 (Conv2D)                    | (None, 61, 61, 64)          |
18,496 |
├──────────────────────────────────────┼─────────────────────────────┼
──────────────┤
| max_pooling2d_4 (MaxPooling2D)       | (None, 30, 30, 64)          |
0 |
├──────────────────────────────────────┼─────────────────────────────┼
──────────────┤
| conv2d_6 (Conv2D)                    | (None, 28, 28, 128)         |
73,856 |
├──────────────────────────────────────┼─────────────────────────────┼
──────────────┤
| max_pooling2d_5 (MaxPooling2D)       | (None, 14, 14, 128)         |
0 |
├──────────────────────────────────────┼─────────────────────────────┼
──────────────┤
| conv2d_7 (Conv2D)                    | (None, 12, 12, 128)         |
147,584 |
├──────────────────────────────────────┼─────────────────────────────┼
──────────────┤
| flatten_1 (Flatten)                  | (None, 18432)               |
0 |
├──────────────────────────────────────┼─────────────────────────────┼
──────────────┤
| dense_2 (Dense)                      | (None, 512)                 |
9,437,696 |
├──────────────────────────────────────┼─────────────────────────────┼
──────────────┤
| dense_3 (Dense)                      | (None, 6)                   |
3,078 |
└──────────────────────────────────────┴─────────────────────────────┴
──────────────┘

 Total params: 9,681,606 (36.93 MB)

 Trainable params: 9,681,606 (36.93 MB)

 Non-trainable params: 0 (0.00 B)

Epoch 1/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 855ms/step - acc: 0.1405 - loss: 1.9285 -
val_acc: 0.2857 - val_loss: 1.7100
Epoch 2/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 5s 48ms/step - acc: 0.2231 - loss: 1.7969 -
val_acc: 0.2857 - val_loss: 1.7677
Epoch 3/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 47ms/step - acc: 0.2388 - loss: 1.7688 -
val_acc: 0.2857 - val_loss: 1.7027
Epoch 4/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 46ms/step - acc: 0.2611 - loss: 1.7188 -
```

```
val_acc: 0.2857 - val_loss: 1.7194
Epoch 5/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - acc: 0.2288 - loss: 1.7350 -
val_acc: 0.2857 - val_loss: 1.6711
Epoch 6/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - acc: 0.2835 - loss: 1.6590 -
val_acc: 0.3968 - val_loss: 1.6350
Epoch 7/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - acc: 0.3411 - loss: 1.6563 -
val_acc: 0.3175 - val_loss: 1.5879
Epoch 8/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - acc: 0.3314 - loss: 1.5725 -
val_acc: 0.3968 - val_loss: 1.5016
Epoch 9/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - acc: 0.4454 - loss: 1.5488 -
val_acc: 0.4762 - val_loss: 1.4659
Epoch 10/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - acc: 0.4079 - loss: 1.5108 -
val_acc: 0.4127 - val_loss: 1.4307
Epoch 11/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - acc: 0.4074 - loss: 1.5246 -
val_acc: 0.5556 - val_loss: 1.2673
Epoch 12/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 47ms/step - acc: 0.4472 - loss: 1.4199 -
val_acc: 0.5556 - val_loss: 1.1283
Epoch 13/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - acc: 0.5266 - loss: 1.2425 -
val_acc: 0.6032 - val_loss: 1.1277
Epoch 14/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 53ms/step - acc: 0.5077 - loss: 1.4065 -
val_acc: 0.5873 - val_loss: 1.1938
Epoch 15/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - acc: 0.4723 - loss: 1.3602 -
val_acc: 0.4444 - val_loss: 1.2110
Epoch 16/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - acc: 0.4195 - loss: 1.3826 -
val_acc: 0.5079 - val_loss: 1.2535
Epoch 17/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - acc: 0.4232 - loss: 1.3780 -
val_acc: 0.5238 - val_loss: 1.1929
Epoch 18/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - acc: 0.4744 - loss: 1.3443 -
val_acc: 0.5873 - val_loss: 1.1706
Epoch 19/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - acc: 0.5035 - loss: 1.2684 -
val_acc: 0.5397 - val_loss: 1.0937
Epoch 20/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - acc: 0.5227 - loss: 1.1975 -
val_acc: 0.6190 - val_loss: 1.0807
```

```
Epoch 21/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - acc: 0.5826 - loss: 1.1286 -
val_acc: 0.6349 - val_loss: 0.9794
Epoch 22/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - acc: 0.6464 - loss: 1.0594 -
val_acc: 0.5714 - val_loss: 1.0142
Epoch 23/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - acc: 0.5544 - loss: 1.2309 -
val_acc: 0.6508 - val_loss: 1.0189
Epoch 24/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - acc: 0.5973 - loss: 1.0663 -
val_acc: 0.5714 - val_loss: 1.1205
Epoch 25/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - acc: 0.5093 - loss: 1.1748 -
val_acc: 0.6190 - val_loss: 0.9809
Epoch 26/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 51ms/step - acc: 0.5895 - loss: 1.1544 -
val_acc: 0.6508 - val_loss: 0.9909
Epoch 27/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 41ms/step - acc: 0.5747 - loss: 1.0897 -
val_acc: 0.6508 - val_loss: 1.0311
Epoch 28/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - acc: 0.5578 - loss: 1.1845 -
val_acc: 0.5714 - val_loss: 1.0244
Epoch 29/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - acc: 0.5685 - loss: 1.0589 -
val_acc: 0.6032 - val_loss: 0.9930
Epoch 30/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - acc: 0.6155 - loss: 0.9403 -
val_acc: 0.6825 - val_loss: 0.9374
Epoch 31/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - acc: 0.6104 - loss: 1.0931 -
val_acc: 0.6825 - val_loss: 0.9264
Epoch 32/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - acc: 0.6781 - loss: 1.0404 -
val_acc: 0.6349 - val_loss: 0.9241
Epoch 33/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - acc: 0.6504 - loss: 0.9259 -
val_acc: 0.6032 - val_loss: 0.8890
Epoch 34/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - acc: 0.6326 - loss: 0.9021 -
val_acc: 0.6190 - val_loss: 1.1219
Epoch 35/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 55ms/step - acc: 0.5994 - loss: 0.9783 -
val_acc: 0.6825 - val_loss: 0.9032
Epoch 36/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 41ms/step - acc: 0.6072 - loss: 0.9448 -
val_acc: 0.6032 - val_loss: 0.9379
Epoch 37/70
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - acc: 0.6923 - loss: 0.8522 -
val_acc: 0.6825 - val_loss: 0.8419
Epoch 38/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - acc: 0.6362 - loss: 0.9010 -
val_acc: 0.6508 - val_loss: 0.9442
Epoch 39/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 47ms/step - acc: 0.6745 - loss: 0.9666 -
val_acc: 0.6984 - val_loss: 0.8582
Epoch 40/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - acc: 0.7227 - loss: 0.7845 -
val_acc: 0.6667 - val_loss: 0.9560
Epoch 41/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - acc: 0.7325 - loss: 0.9079 -
val_acc: 0.6984 - val_loss: 0.9224
Epoch 42/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - acc: 0.7436 - loss: 0.7628 -
val_acc: 0.7143 - val_loss: 0.8839
Epoch 43/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - acc: 0.7274 - loss: 0.7696 -
val_acc: 0.6825 - val_loss: 0.8408
Epoch 44/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - acc: 0.7428 - loss: 0.7621 -
val_acc: 0.6984 - val_loss: 0.8865
Epoch 45/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - acc: 0.7150 - loss: 0.8224 -
val_acc: 0.6667 - val_loss: 0.9824
Epoch 46/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - acc: 0.7000 - loss: 0.8273 -
val_acc: 0.6825 - val_loss: 0.9510
Epoch 47/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 29ms/step - acc: 0.7537 - loss: 0.7684 -
val_acc: 0.6825 - val_loss: 0.9953
Epoch 48/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - acc: 0.7399 - loss: 0.6982 -
val_acc: 0.6190 - val_loss: 1.1180
Epoch 49/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 46ms/step - acc: 0.6767 - loss: 0.8689 -
val_acc: 0.7143 - val_loss: 0.8479
Epoch 50/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - acc: 0.6422 - loss: 1.0001 -
val_acc: 0.6984 - val_loss: 0.8490
Epoch 51/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - acc: 0.7469 - loss: 0.6892 -
val_acc: 0.6190 - val_loss: 0.9663
Epoch 52/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - acc: 0.6824 - loss: 0.7690 -
val_acc: 0.6508 - val_loss: 0.8512
Epoch 53/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - acc: 0.7298 - loss: 0.7782 -
```

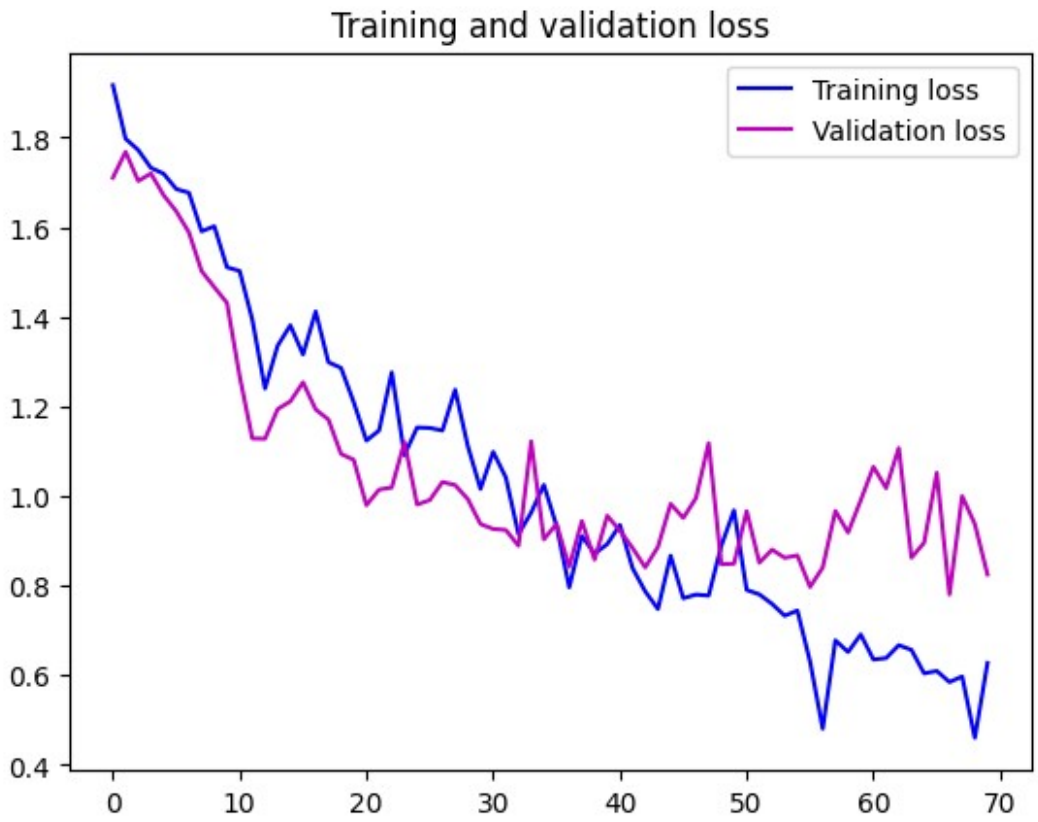
```
val_acc: 0.6667 - val_loss: 0.8802
Epoch 54/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 47ms/step - acc: 0.7403 - loss: 0.7119 -
val_acc: 0.6825 - val_loss: 0.8621
Epoch 55/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 46ms/step - acc: 0.7109 - loss: 0.7824 -
val_acc: 0.6825 - val_loss: 0.8675
Epoch 56/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - acc: 0.7797 - loss: 0.6105 -
val_acc: 0.6825 - val_loss: 0.7971
Epoch 57/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - acc: 0.8211 - loss: 0.5231 -
val_acc: 0.6984 - val_loss: 0.8406
Epoch 58/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - acc: 0.7264 - loss: 0.7225 -
val_acc: 0.7143 - val_loss: 0.9666
Epoch 59/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - acc: 0.7492 - loss: 0.6488 -
val_acc: 0.7460 - val_loss: 0.9184
Epoch 60/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - acc: 0.7750 - loss: 0.6989 -
val_acc: 0.6984 - val_loss: 0.9906
Epoch 61/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - acc: 0.7255 - loss: 0.5793 -
val_acc: 0.7619 - val_loss: 1.0652
Epoch 62/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - acc: 0.7883 - loss: 0.5481 -
val_acc: 0.6508 - val_loss: 1.0175
Epoch 63/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - acc: 0.7641 - loss: 0.7106 -
val_acc: 0.6825 - val_loss: 1.1071
Epoch 64/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - acc: 0.7591 - loss: 0.7092 -
val_acc: 0.7302 - val_loss: 0.8624
Epoch 65/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - acc: 0.8380 - loss: 0.5440 -
val_acc: 0.6825 - val_loss: 0.8959
Epoch 66/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - acc: 0.8160 - loss: 0.5694 -
val_acc: 0.6667 - val_loss: 1.0516
Epoch 67/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - acc: 0.7798 - loss: 0.5108 -
val_acc: 0.7143 - val_loss: 0.7804
Epoch 68/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 28ms/step - acc: 0.7751 - loss: 0.6190 -
val_acc: 0.6984 - val_loss: 0.9997
Epoch 69/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step - acc: 0.8416 - loss: 0.4900 -
val_acc: 0.6984 - val_loss: 0.9373
Epoch 70/70
```

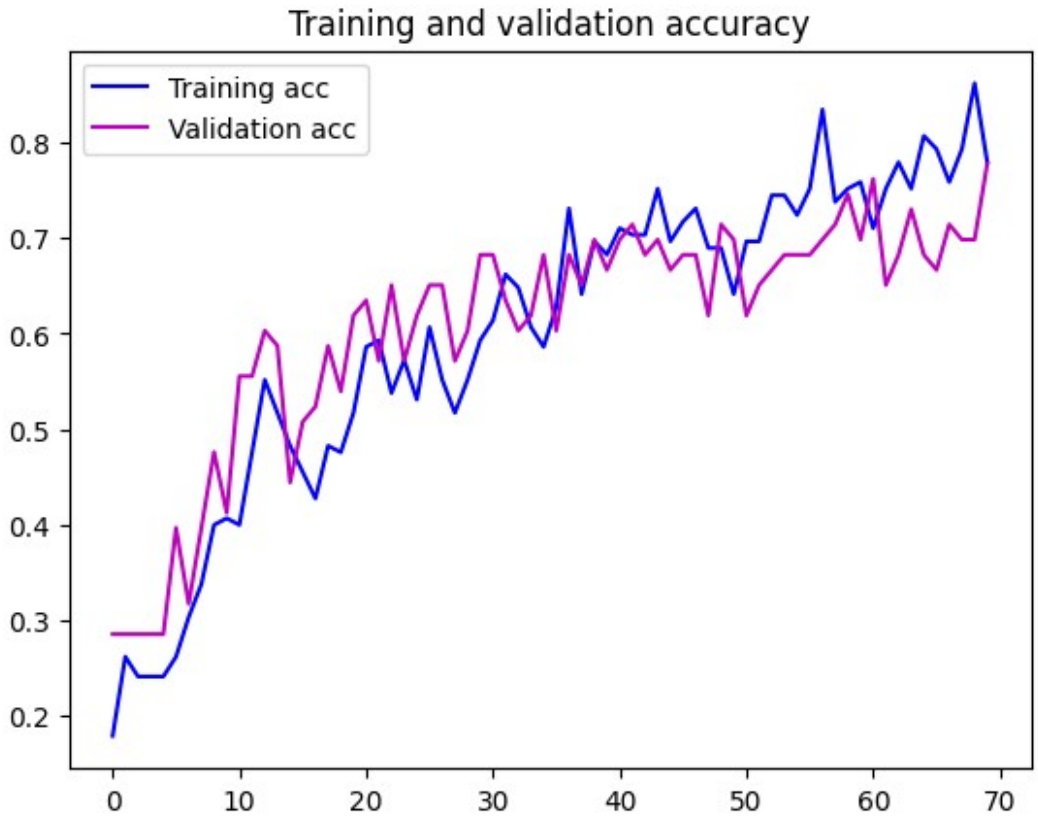
```
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 46ms/step - acc: 0.7509 - loss: 0.6466 -
val_acc: 0.7778 - val_loss: 0.8248
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - acc: 0.7685 - loss: 0.8589
Validation accuracy: 77.78%
```

```
print('Validation accuracy: {:2.2f}%'.format(test_acc*100))
```

```
Validation accuracy: 77.78%
```

```
# Get the training info
loss     = history.history['loss']
val_loss = history.history['val_loss']
acc      = history.history['acc']
val_acc  = history.history['val_acc']

# Visualize the history plots
plt.figure()
plt.plot(loss, 'b', label='Training loss')
plt.plot(val_loss, 'm', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
plt.figure()
plt.plot(acc, 'b', label='Training acc')
plt.plot(val_acc, 'm', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.show()
```


Training and validation loss
Training loss
Validation loss
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0
10
20
30
40
50
60
70


Training and validation accuracy
Training acc
Validation acc
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
10
20
30
40
50
60
70

3. Face encoding

The simplest approach to face recognition is to directly classify an unknown face with a convnet trained on your database of tagged people. Seems like a pretty good idea, right? There's actually a huge problem with that approach. A site like Facebook with billions of users and a trillion photos can't possibly train such a big convnet. That would take way too long. What you need is a way to extract a few basic measurements from each face, which you can then use to quickly compare the unknown face with your database. For example, you might measure the size of each ear, the spacing between the eyes, the length of the nose, etc. However, it turns out that the measurements that seem obvious to us humans (like eye color) don't really make sense to a computer looking at individual pixels in an image. Researchers have discovered that the most accurate approach is to let the computer figure out the measurements to collect itself. Deep learning does a better job than humans at figuring out which parts of a face are important to measure.

The solution is to train a convnet. But instead of training the network to classify pictures, it is trained to generate 128 measurements for each face. The training process works by looking at 3 face images at a time: the picture of a known person, another picture of the same known person, and a picture of a totally different person. Then, the algorithm looks at the measurements currently generated for each of those three images. It tweaks the neural network slightly to make sure that the measurements generated for the same person are slightly closer, and the measurements for different persons are slightly further apart. After repeating this step millions of times for millions of images of thousands of different people, the convnet learns to generate 128 measurements for each person.

This process of training a convnet to output face encodings requires a lot of data and computer power. Even with an expensive GPU, it takes about 24 hours of continuous training to get good accuracy. But once the network has been trained, it can generate measurements for any face, even ones it has never seen before! So this step only needs to be done once. Fortunately, the people at OpenFace already did this and they published several trained networks which you can directly use. So all you need to do is run your face images through their pre-trained network to get the 128 measurements for each face.

Assignment

Here's what you are required to do for this part of the assignment.

- Preprocess the cropped faces by encoding them. You should now have a dataset of cropped and encoded faces.
- Train a neural network on the modified dataset. Since the encoded faces are just 128-length vectors, **you don't need a convnet**. Use a regular neural network with a series of fully-connected layers.
- Evaluate the performance on the test set, and compare it to the scores obtained with your previously trained convnets.

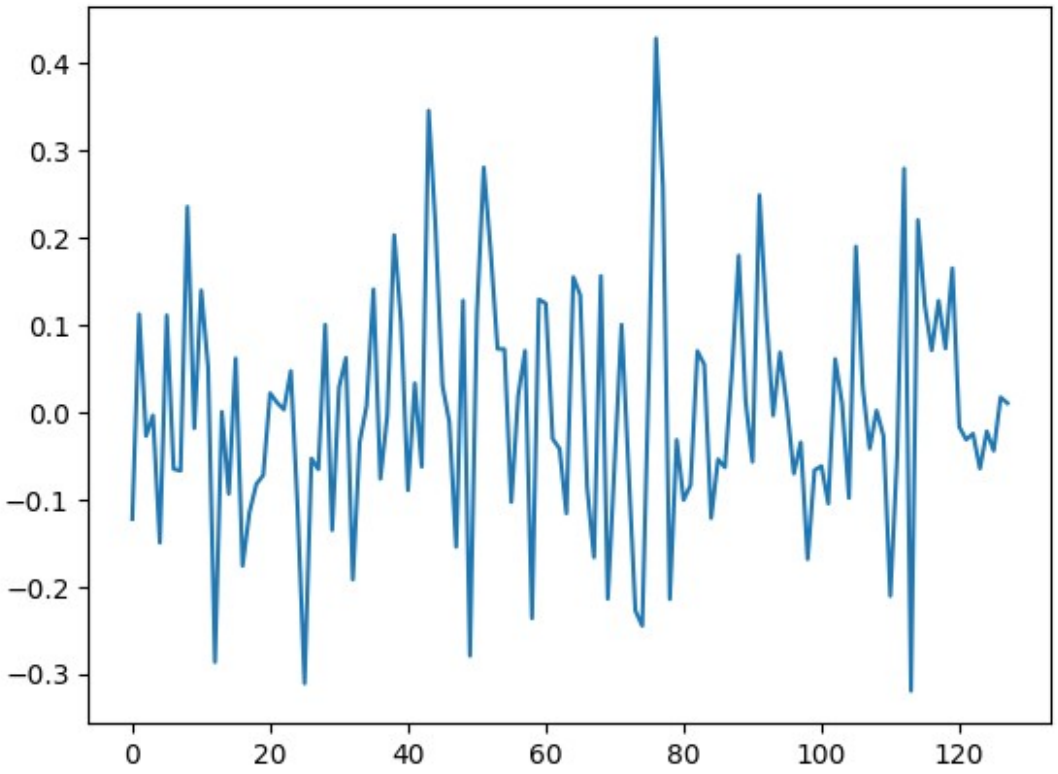
Provided functions

```
cnn_encoder =
dlib.face_recognition_model_v1('models/dlib_face_recognition_resnet_mo
del_v1.dat')

def face_encoder(faces):

    landmarks = face_landmarks(faces)

    if not isinstance(faces, list):
        return
np.array(cnn_encoder.compute_face_descriptor(faces,landmarks))
    else:
        return np.array([cnn_encoder.compute_face_descriptor(f,l) for
f,l in zip(faces,landmarks)])
```

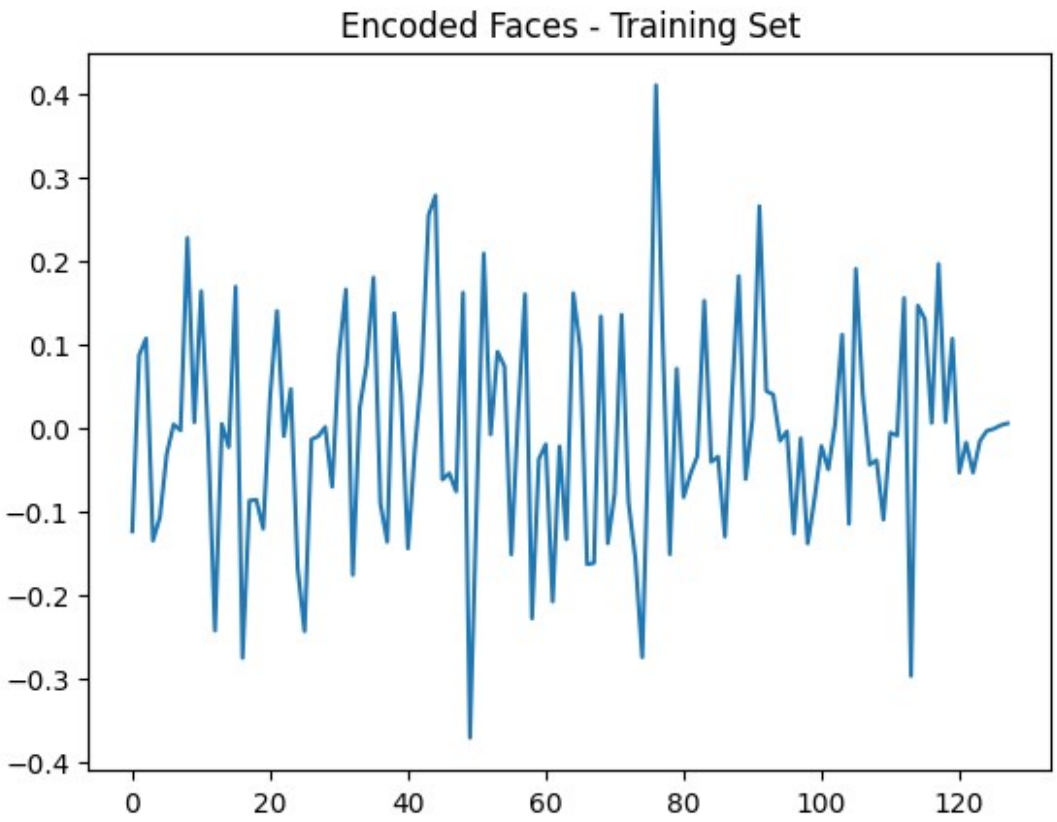
Hints

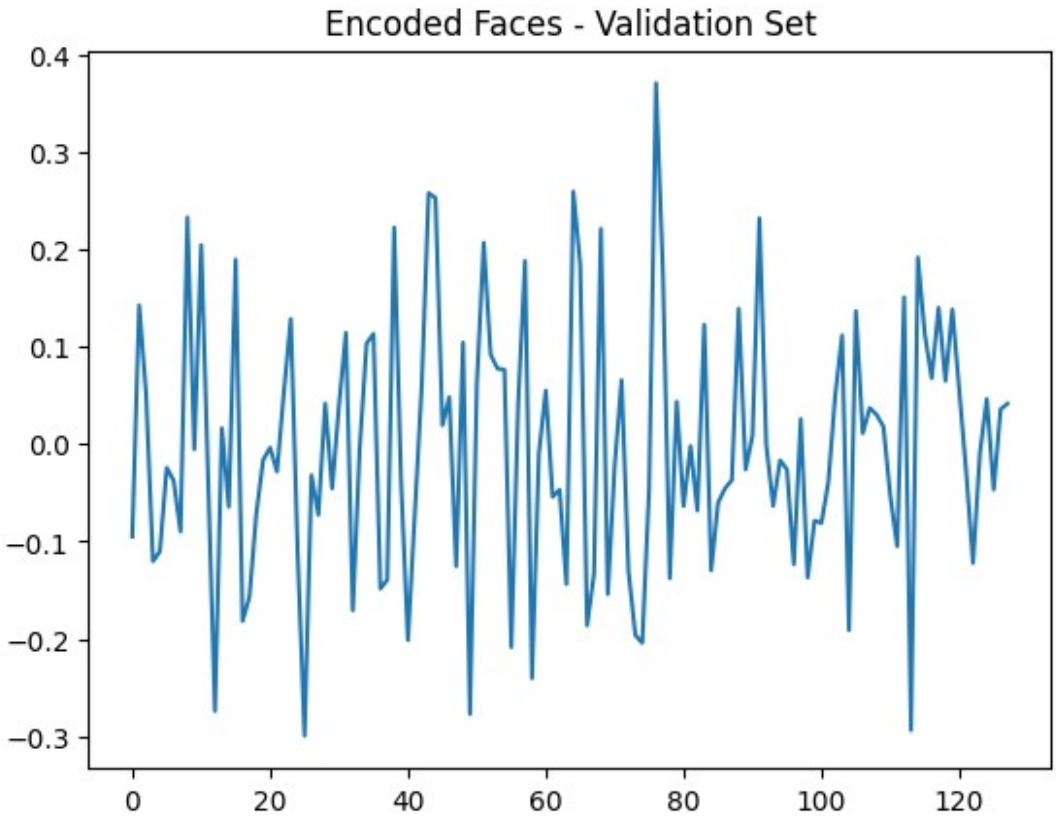
The provided function `face_encoder()` computes the encodings for a list of cropped faces. Alignment and normalization are handled internally.

```
encoded_faces = face_encoder(faces)

plt.plot(encoded_faces[0])

[<matplotlib.lines.Line2D at 0x7f83a3fc6800>]
```




```python
# Encoder les visages de l'ensemble d'entraînement
encoded_faces_train = face_encoder(X_train_aligned)
# Encoder les visages de l'ensemble de validation
encoded_faces_validation = face_encoder(X_validation_aligned)
# Créer une nouvelle figure pour le premier graphique
plt.figure()
plt.plot(encoded_faces_train[0])
plt.title("Encoded Faces - Training Set")

plt.figure()
plt.plot(encoded_faces_validation[0])
plt.title("Encoded Faces - Validation Set")

plt.show()
```


Encoded Faces - Training Set
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
0
20
40
60
80
100
120


Encoded Faces - Validation Set
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
0
20
40
60
80
100
120

```
from tensorflow.keras.layers import Dropout
from tensorflow.keras.callbacks import EarlyStopping
from keras import regularizers


model = models.Sequential()
model.add(layers.Dense(128, activation='relu', input_shape=(128,)))



#model.add(layers.Dense(16, kernel_regularizer=regularizers.l2(0.001),
activation='relu'))
model.add(layers.Dense(6, activation='softmax'))

model.summary()

model.compile(loss='categorical_crossentropy',
optimizer=optimizers.Adam(learning_rate=1e-2), metrics=['acc'])
early_stopping = EarlyStopping(monitor='val_loss', patience=10)

# Train the network
history = model.fit(encoded_faces_train, Y_train_encoded_aligned,
epochs=70, validation_data=(encoded_faces_validation,
Y_validation_encoded_aligned),callbacks=[early_stopping])

# Evaluate the model on the validation set
test_loss, test_acc = model.evaluate(encoded_faces_validation,
Y_validation_encoded_aligned)
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/
dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim`
argument to a layer. When using Sequential models, prefer using an
`Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)
```

```
Model: "sequential_2"
```

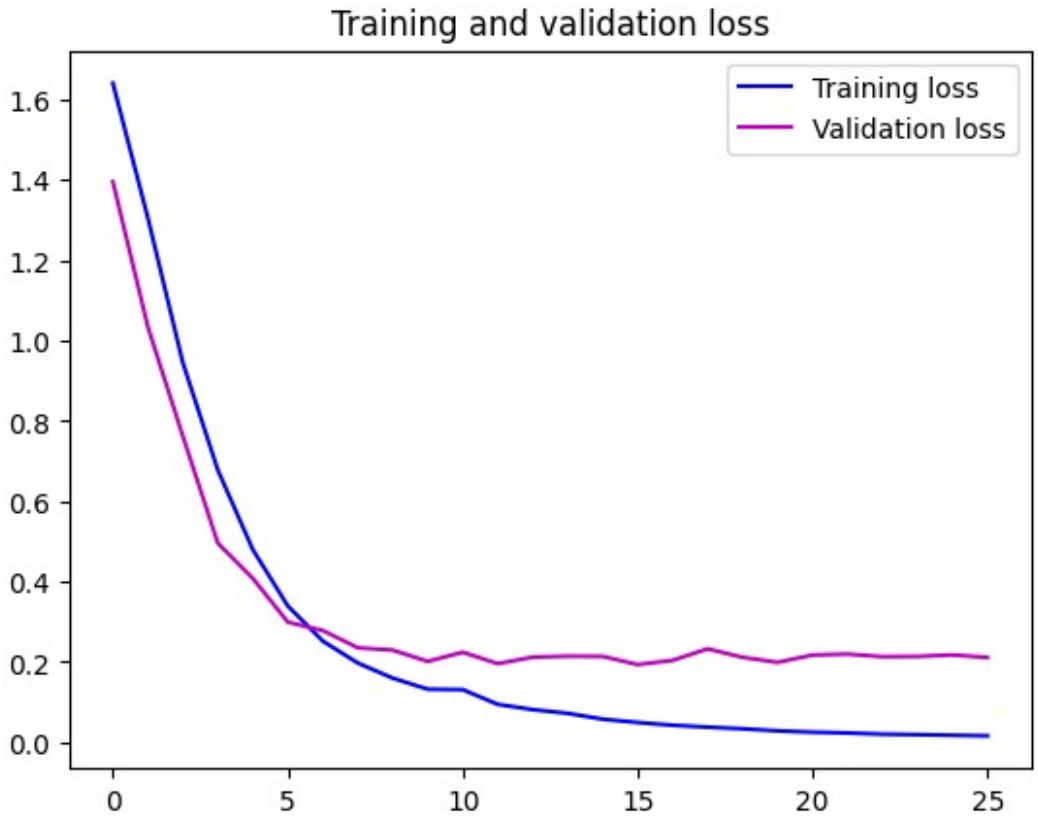
Layer (type)	Output Shape	Param #
dense_4 (Dense)	(None, 128)	16,512
dense_5 (Dense)	(None, 6)	774

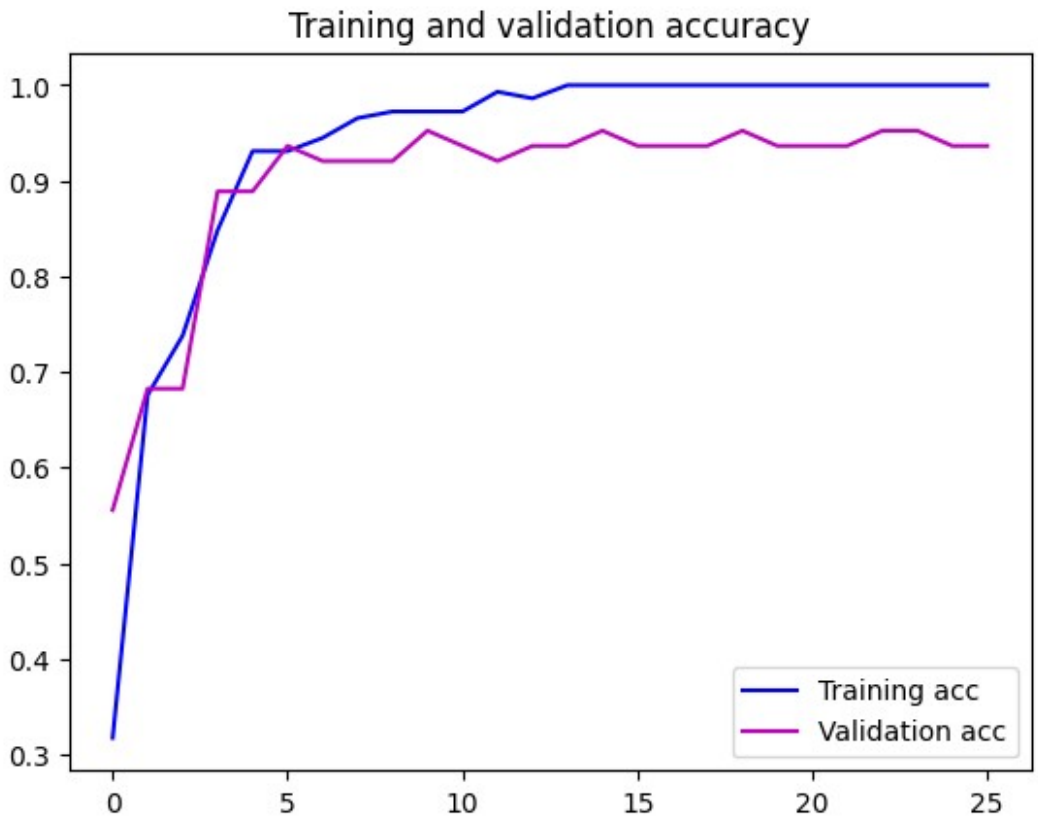
```
└───────────────────────────────────────┴──────────────────────────────┴
─────────────────┘

 Total params: 17,286 (67.52 KB)

 Trainable params: 17,286 (67.52 KB)

 Non-trainable params: 0 (0.00 B)

Epoch 1/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 311ms/step - acc: 0.2603 - loss: 1.6972 -
val_acc: 0.5556 - val_loss: 1.3958
Epoch 2/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - acc: 0.6433 - loss: 1.3447 -
val_acc: 0.6825 - val_loss: 1.0342
Epoch 3/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - acc: 0.7777 - loss: 0.9581 -
val_acc: 0.6825 - val_loss: 0.7620
Epoch 4/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - acc: 0.8392 - loss: 0.7005 -
val_acc: 0.8889 - val_loss: 0.4957
Epoch 5/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - acc: 0.9280 - loss: 0.4750 -
val_acc: 0.8889 - val_loss: 0.4079
Epoch 6/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - acc: 0.9510 - loss: 0.3339 -
val_acc: 0.9365 - val_loss: 0.2990
Epoch 7/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - acc: 0.9495 - loss: 0.2303 -
val_acc: 0.9206 - val_loss: 0.2780
Epoch 8/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - acc: 0.9724 - loss: 0.1758 -
val_acc: 0.9206 - val_loss: 0.2351
Epoch 9/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - acc: 0.9756 - loss: 0.1461 -
val_acc: 0.9206 - val_loss: 0.2291
Epoch 10/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - acc: 0.9865 - loss: 0.1198 -
val_acc: 0.9524 - val_loss: 0.2012
Epoch 11/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - acc: 0.9665 - loss: 0.1491 -
val_acc: 0.9365 - val_loss: 0.2239
Epoch 12/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - acc: 0.9977 - loss: 0.0908 -
val_acc: 0.9206 - val_loss: 0.1955
Epoch 13/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - acc: 0.9815 - loss: 0.0838 -
val_acc: 0.9365 - val_loss: 0.2117
Epoch 14/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - acc: 1.0000 - loss: 0.0625 -
```

```
val_acc: 0.9365 - val_loss: 0.2141
Epoch 15/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - acc: 1.0000 - loss: 0.0559 -
val_acc: 0.9524 - val_loss: 0.2136
Epoch 16/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - acc: 1.0000 - loss: 0.0514 -
val_acc: 0.9365 - val_loss: 0.1931
Epoch 17/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - acc: 1.0000 - loss: 0.0425 -
val_acc: 0.9365 - val_loss: 0.2037
Epoch 18/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - acc: 1.0000 - loss: 0.0398 -
val_acc: 0.9365 - val_loss: 0.2325
Epoch 19/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - acc: 1.0000 - loss: 0.0340 -
val_acc: 0.9524 - val_loss: 0.2113
Epoch 20/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - acc: 1.0000 - loss: 0.0283 -
val_acc: 0.9365 - val_loss: 0.1988
Epoch 21/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step - acc: 1.0000 - loss: 0.0237 -
val_acc: 0.9365 - val_loss: 0.2170
Epoch 22/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - acc: 1.0000 - loss: 0.0239 -
val_acc: 0.9365 - val_loss: 0.2193
Epoch 23/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - acc: 1.0000 - loss: 0.0175 -
val_acc: 0.9524 - val_loss: 0.2129
Epoch 24/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - acc: 1.0000 - loss: 0.0169 -
val_acc: 0.9524 - val_loss: 0.2135
Epoch 25/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - acc: 1.0000 - loss: 0.0188 -
val_acc: 0.9365 - val_loss: 0.2173
Epoch 26/70
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - acc: 1.0000 - loss: 0.0136 -
val_acc: 0.9365 - val_loss: 0.2108
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - acc: 0.9473 - loss: 0.1731
```

```python
# Get the training info
loss     = history.history['loss']
val_loss = history.history['val_loss']
acc      = history.history['acc']
val_acc  = history.history['val_acc']

# Visualize the history plots
plt.figure()
plt.plot(loss, 'b', label='Training loss')
plt.plot(val_loss, 'm', label='Validation loss')
plt.title('Training and validation loss')
```

```
plt.legend()
plt.show()
plt.figure()
plt.plot(acc, 'b', label='Training acc')
plt.plot(val_acc, 'm', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.show()
```





4. Face recognition

This last step is actually the easiest one in the whole process. All you have to do is find the person in your database of known people who has the closest measurements to some test image. You can do that by using any machine learning classification algorithm, such as neaural network (as you did in the previous section), logistic regression, SVM, nearest neighbours, etc. All you need to do is training a classifier that can take in the measurements from a new test image, and tells which known person is the closest match. Running this classifier must only take milliseconds, so that you can apply it to video sequences.

Assignment

Here's what you are required to do for this part of the assignment.

- Train several classifiers (logistic regression, SVM, kNN, neural network) on the dataset of encoded faces (you can use the package `scikit-learn`).
- Evaluate their performance on the test set, in terms of accuracy and speed.
- Finally, run your best classifier on the test images and video available in the `test` folder.

Provided functions

```
def process_frame(image, mode="fast"):

    # face detection
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    if mode == "fast":
        matches = hog_detector(gray,1)
    else:
        matches = cnn_detector(gray,1)
        matches = [m.rect for m in matches]

    for rect in matches:

        # face landmarks
        landmarks = pose68(gray, rect)

        # face encoding
        encoding = cnn_encoder.compute_face_descriptor(image,
landmarks)

        # face classification
        label = "label"

        # draw box
        cv2.rectangle(image, (rect.left(), rect.top()), (rect.right(),
rect.bottom()), (0, 255, 0), 2)
        y = rect.top() - 15 if rect.top() - 15 > 15 else rect.bottom()
+ 25
        cv2.putText(image, label, (rect.left(), y),
cv2.FONT_HERSHEY_SIMPLEX, 0.75, (0, 255, 0), 2)

    return image
```

Note: cv2.VideoCapture does not work in Google Colab. You can use https://colab.research.google.com/notebooks/snippets/advanced_outputs.ipynb#scrollTo=2viqYx97hPMi to capture video 'on the fly' with Google Colab. The following function has to be modified accordingly

```
from google.colab.patches import cv2_imshow

def process_movie(video_name, mode="fast"):
    video = cv2.VideoCapture(video_name)

    try:
        while True:
            # Grab a single frame of video
            ret, frame = video.read()

            # Resize frame of video for faster processing
            frame = cv2.resize(frame, (0, 0), fx=0.5, fy=0.5)
```

```
            # Quit when the input video file ends or key "Q" is
pressed
            key = cv2.waitKey(1) & 0xFF
            if not ret or key == ord("q"):
                break

            # Process frame
            image = process_frame(frame, mode)

            # Display the resulting image
            cv2_imshow(image)

    finally:
        video.release()
        cv2.destroyAllWindows()
        print("Video released")
```

Hints

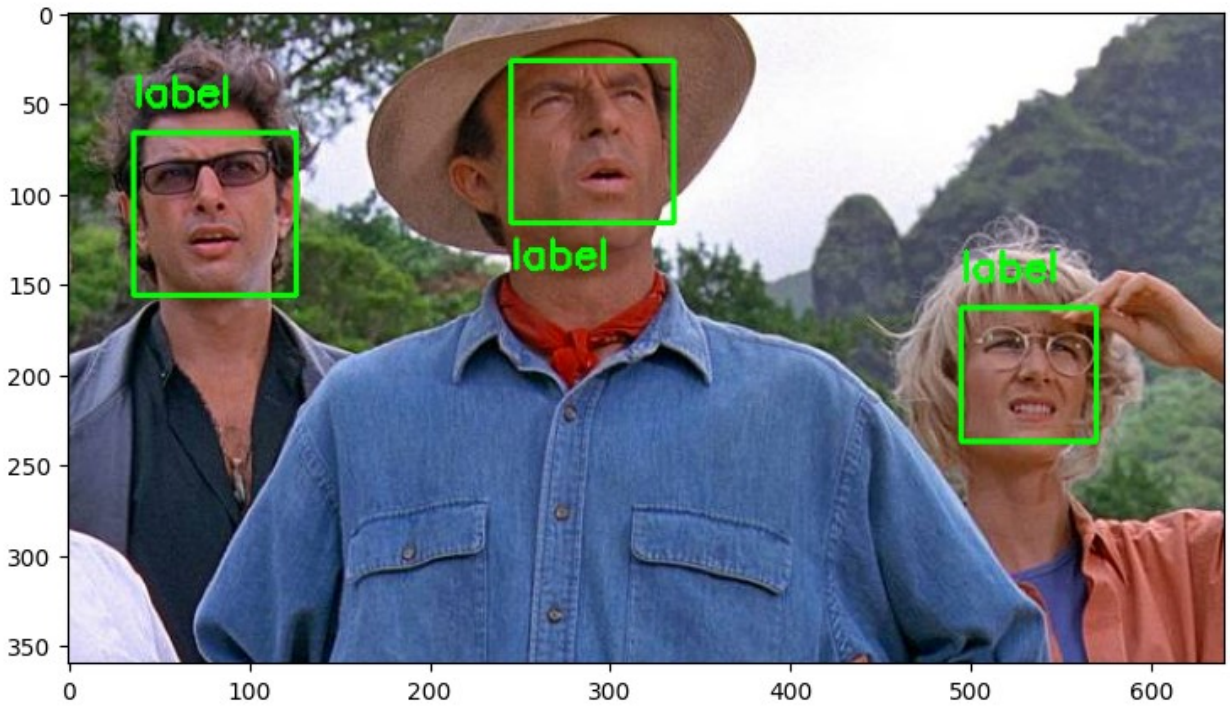
The provided function `process_frame()` detects and encodes all the faces in the input image.

```
!wget https://perso.esiee.fr/~najmanl/FaceRecognition/test.zip
!unzip test.zip
```

```
--2024-12-12 16:39:52--
https://perso.esiee.fr/~najmanl/FaceRecognition/test.zip
Resolving perso.esiee.fr (perso.esiee.fr)... 147.215.150.8
Connecting to perso.esiee.fr (perso.esiee.fr)|147.215.150.8|:443...
connected.
HTTP request sent, awaiting response... 200 OK
Length: 12530126 (12M) [application/zip]
Saving to: ‘test.zip.1’

test.zip.1          100%[===================>]  11.95M  8.38MB/s    in
1.4s

2024-12-12 16:39:54 (8.38 MB/s) - ‘test.zip.1’ saved
[12530126/12530126]

Archive:  test.zip
replace test/lunch_scene.mp4? [y]es, [n]o, [A]ll, [N]one, [r]ename:
```

```
image = cv2.imread("test/example_03.png")

processed = process_frame(image.copy())
processed = cv2.cvtColor(processed, cv2.COLOR_BGR2RGB)

plt.figure(figsize=(15,5))
plt.imshow(processed)
```

```
<matplotlib.image.AxesImage at 0x7f83dc36b2e0>
```



The provided function `process_frame()` detects and encodes the faces in the input video.

```
#process_movie("test/lunch_scene.mp4", "fast")
```

The special input `0` can be used to access the webcam.

```
#process_movie(0)

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score

# Convert one-hot encoded labels to class labels
y_train_classifiers = np.argmax(Y_train_encoded_aligned, axis=1)

# Split the dataset
X_train_classifiers, X_test_classifiers, y_train_classifiers,
y_test_classifiers = train_test_split(encoded_faces_train,
y_train_classifiers, test_size=0.2, random_state=42)


# 3. Train classifiers
```

```
# Logistic Regression
logistic_reg = LogisticRegression()
logistic_reg.fit(X_train_classifiers, y_train_classifiers)

# SVM
svm_classifier = SVC()
svm_classifier.fit(X_train_classifiers, y_train_classifiers)

# kNN
knn_classifier = KNeighborsClassifier()
knn_classifier.fit(X_train_classifiers, y_train_classifiers)

# Neural Network
nn_classifier = MLPClassifier()
nn_classifier.fit(X_train_classifiers, y_train_classifiers)

# 4. Evaluate classifiers
# Logistic Regression
logistic_reg_pred = logistic_reg.predict(X_test_classifiers)
logistic_reg_acc = accuracy_score(y_test_classifiers,
logistic_reg_pred)

# SVM
svm_pred = svm_classifier.predict(X_test_classifiers)
svm_acc = accuracy_score(y_test_classifiers, svm_pred)

# kNN
knn_pred = knn_classifier.predict(X_test_classifiers)
knn_acc = accuracy_score(y_test_classifiers, knn_pred)

# Neural Network
nn_pred = nn_classifier.predict(X_test_classifiers)
nn_acc = accuracy_score(y_test_classifiers, nn_pred)

# Print accuracy of each classifier
print("Logistic Regression Accuracy:", logistic_reg_acc)
print("SVM Accuracy:", svm_acc)
print("kNN Accuracy:", knn_acc)
print("Neural Network Accuracy:", nn_acc)
```

```
Logistic Regression Accuracy: 0.9655172413793104
SVM Accuracy: 0.9655172413793104
kNN Accuracy: 0.896551724137931
Neural Network Accuracy: 0.9310344827586207

/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/
_multilayer_perceptron.py:690: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization
hasn't converged yet.
  warnings.warn(
```

5. Build a custom dataset

So far, you have used a pre-curated dataset, where somebody did the hard work of gathering and labeling the images for you. Now, you will tackle the problem of recognizing faces of yourselves, friends, family members, colleagues, etc. To accomplish this, you need to gather examples of faces you want to recognize. You can enroll facial pictures via a webcam attached to your computer.

Assignment

Here's what you are required to do for this part of the assignment.

- Use your webcam to enroll face pictures of yourself, your friends, etc. To do so, you need to open the webcam, detect faces in the video stream, and save the captured face images to disk.
- Build a dataset of reasonable size: a group of 10-15 people with 50-100 face pictures each, taken in different conditions of light, angle, emotion, etc.
- Apply the previously developed pipeline to build your own personalized face recognition system.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

```
!ls /content/drive/
```

```
MyDrive  Shareddrives
```

```
# Copier le fichier zip depuis Google Drive vers le répertoire actuel
!cp "/content/drive/MyDrive/drive_photo_perso_2.zip" .

# Dézipper le fichier
!unzip drive_photo_perso_2.zip -d drive_photo_perso_2
```

```
cp: cannot stat '/content/drive/MyDrive/drive_photo_perso_2.zip': No
such file or directory
unzip:  cannot find or open drive_photo_perso_2.zip,
drive_photo_perso_2.zip.zip or drive_photo_perso_2.zip.ZIP.
```

```
imagePaths_perso = list_images("drive_photo_perso_2")
imagePaths_perso

list_names_perso = []

for imagePath in imagePaths_perso :
```

```
    separate = imagePath.split('/')
    name_images = separate[1]
    list_names_perso.append(name_images)

print(list_names_perso)
print(len(list_names_perso))
```

```
[]
0
```

```
imagePaths_perso
```

```
[]
```

```
from PIL import Image
import os

dossier_images = imagePaths_perso

for fichier in imagePaths_perso:
    if fichier.endswith(".heic") or fichier.endswith(".JPG"):
        # Ouvrir l'image HEIC
        chemin_heic = os.path.join(dossier_images, fichier)
        img = Image.open(chemin_heic)

        # Convertir en JPG
        chemin_jpg = os.path.splitext(chemin_heic)[0] + ".jpg"
        img.convert("RGB").save(chemin_jpg, "JPEG")

        # Fermer l'image
        img.close()
```

```
imagePaths_perso
```

```
[]
```

```
import cv2

# Initialisation des listes pour stocker les visages extraits et les
noms des images
cropped_image_perso = []
list_labels_perso = []

try:
    image_perso = cv2.imread(i)
    if image_perso is None:
        print("Erreur: Impossible de lire l'image à", i)
except Exception as e:
    print("Erreur OpenCV:", e)

# Parcours des chemins d'accès des images
```

```
for i in imagePaths_perso:
    # Lecture de l'image à l'aide de OpenCV
    image_perso = cv2.imread(i)

    # Conversion de l'espace de couleur de BGR à RGB
    image_perso = cv2.cvtColor(image_perso, cv2.COLOR_BGR2RGB)

    # Appel de la fonction pour extraire les visages de l'image
    image_perso = extract_faces(image_perso, "cnn")

    # Vérification si la fonction extract_faces() a renvoyé au moins
un visage
    if image_perso:
        # Ajout du premier visage extrait à la liste cropped_image
        cropped_image_perso.append(image_perso[0])

        # Extraction du nom de l'image à partir du chemin d'accès
        filepath = i
        name = filepath.split('/')[1]

        # Ajout du nom de l'image à la liste list_labels
        list_labels_perso.append(name)
```

```
Erreur OpenCV: OpenCV(4.10.0) :-1: error: (-5:Bad argument) in
function 'imread'
> Overload resolution failed:
>  - Expected 'filename' to be a str or path-like object
>  - Expected 'filename' to be a str or path-like object
>  - Expected 'filename' to be a str or path-like object
```

```
# Initialisation d'une liste pour stocker les images normalisées
normalized_images_perso = []

# Parcours des images extraites
for image in cropped_image_perso:

    # Normalisation de l'image en divisant chaque pixel par 255.0
    normalized_image_perso = image / 255.0

    # Ajout de l'image normalisée à la liste normalized_images
    normalized_images_perso.append(normalized_image_perso)

landmarks_cropped_images_perso = [face_landmarks(image_perso) for
image_perso in cropped_image_perso]
# Aligner le visage
aligned_face_perso = align_faces(cropped_image_perso,
landmarks_cropped_images_perso)

for image, landmarks in zip(aligned_face_perso,
landmarks_cropped_images_perso):
```

```
    canvas = image.copy()
    coords = shape_to_coords(landmarks)
    for p in coords:
        cv2.circle(canvas, (int(p[0]), int(p[1])), 1, (0, 0, 255), -1)
    # Afficher ou sauvegarder chaque image modifiée
    plt.imshow(canvas)
    plt.show()

X_train_aligned_perso, X_validation_aligned_perso,
Y_train_aligned_perso, Y_validation_aligned_perso =
train_test_split(aligned_face_perso, list_labels_perso, test_size=0.3,
random_state=42)

# Sauvegarde de l'ensemble d'entraînement dans un fichier pickle
'train_data.pkl'
with open('train_data_perso.pkl', 'wb') as f:
    # Utilisation de pickle.dump() pour sauvegarder les données dans
le fichier
    # Les données sont sauvegardées sous forme de tuple (X_train,
Y_train)
    pickle.dump((X_train_aligned_perso, Y_train_aligned_perso), f)

# Sauvegarde de l'ensemble de test dans un fichier pickle
'validation_data.pkl'
with open('validation_data_perso.pkl', 'wb') as f:
    # Utilisation de pickle.dump() pour sauvegarder les données dans
le fichier
    # Les données sont sauvegardées sous forme de tuple (X_test,
Y_test)
    pickle.dump((X_validation_aligned_perso,
Y_validation_aligned_perso), f)

---------------------------------------------------------------------
-----
ValueError                                Traceback (most recent call
last)
<ipython-input-76-392ac5408a36> in <cell line: 1>()
----> 1 X_train_aligned_perso, X_validation_aligned_perso,
Y_train_aligned_perso, Y_validation_aligned_perso =
train_test_split(aligned_face_perso, list_labels_perso, test_size=0.3,
random_state=42)
      2
      3 # Sauvegarde de l'ensemble d'entraînement dans un fichier
pickle 'train_data.pkl'
      4 with open('train_data_perso.pkl', 'wb') as f:
      5     # Utilisation de pickle.dump() pour sauvegarder les
données dans le fichier

/usr/local/lib/python3.10/dist-packages/sklearn/utils/_param_validatio
n.py in wrapper(*args, **kwargs)
```

```
    211                         )
    212                 ):
--> 213                     return func(*args, **kwargs)
    214             except InvalidParameterError as e:
    215                 # When the function is just a wrapper around
an estimator, we allow

/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split
.py in train_test_split(test_size, train_size, random_state, shuffle,
stratify, *arrays)
   2783
   2784     n_samples = _num_samples(arrays[0])
-> 2785     n_train, n_test = _validate_shuffle_split(
   2786         n_samples, test_size, train_size,
default_test_size=0.25
   2787     )

/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split
.py in _validate_shuffle_split(n_samples, test_size, train_size,
default_test_size)
   2413
   2414     if n_train == 0:
-> 2415         raise ValueError(
   2416             "With n_samples={}, test_size={} and
train_size={}, the "
   2417             "resulting train set will be empty. Adjust any of
the "

ValueError: With n_samples=0, test_size=0.3 and train_size=None, the
resulting train set will be empty. Adjust any of the aforementioned
parameters.
```

```
# Création d'une instance de LabelBinarizer pour l'encodage
encoder = LabelBinarizer()

# Encodage des étiquettes d'entraînement en format binaire
# La méthode fit_transform ajuste le transformateur sur Y_train et
encode simultanément les étiquettes
Y_train_encoded_aligned_perso =
encoder.fit_transform(Y_train_aligned_perso)

# Encodage des étiquettes de test en format binaire
# La méthode transform utilise les informations d'encodage apprises
sur Y_train pour encoder Y_test
Y_validation_encoded_aligned_perso =
encoder.transform(Y_validation_aligned_perso)

train_generator_aligned_perso =
datagen.flow(np.array(X_train_aligned_perso),Y_train_encoded_aligned_p
erso, batch_size=32)
```

```
# Define the network
model = models.Sequential()

model.add(layers.Conv2D(32, (3, 3), activation='relu',
input_shape=(128, 128, 3)))
model.add(layers.MaxPooling2D(2,2))
model.add(layers.Conv2D(64, (3,3), activation = 'relu'))
model.add(layers.MaxPooling2D(2,2))
model.add(layers.Conv2D(128, (3,3), activation = 'relu'))
model.add(layers.MaxPooling2D(2,2))
model.add(layers.Conv2D(128, (3,3), activation = 'relu'))
#model.add(layers.Dropout(0.5))
model.add(layers.Flatten())
model.add(layers.Dense(128, activation = 'relu'))
model.add(layers.Dense(6, activation = 'softmax'))
model.summary()

model.compile(loss='categorical_crossentropy',
optimizer=optimizers.Adam(learning_rate=1e-4), metrics=['acc'])

# Train the network. Hint: use .fit(...)
history = model.fit(train_generator_aligned_perso, epochs=50,
validation_data = (np.array(X_validation_aligned_perso),
Y_validation_encoded_aligned_perso))

# Évaluation du modèle sur les données de validation et obtention de
la perte (loss) et de la précision (accuracy)
test_loss, test_acc =
model.evaluate(np.array(X_validation_aligned_perso),
Y_validation_encoded_aligned_perso)
print('Validation accuracy: {:2.2f}%'.format(test_acc*100))

# Get the training info
loss     = history.history['loss']
val_loss = history.history['val_loss']
acc      = history.history['acc']
val_acc  = history.history['val_acc']

# Visualize the history plots
plt.figure()
plt.plot(loss, 'b', label='Training loss')
plt.plot(val_loss, 'm', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
plt.figure()
plt.plot(acc, 'b', label='Training acc')
plt.plot(val_acc, 'm', label='Validation acc')
plt.title('Training and validation accuracy')
```

```
plt.legend()
plt.show()
```

Credits

This assignment is based on Adam Geitgey's post.
