

Final Project

Advanced Programming

DeepFake and FaceSwap

Milana Aligaziyeva, Dilnaza Maratova

IT-2101

Introduction

Problem.

The goal of the project is to explore machine learning and artificial intelligence techniques for producing or modifying visual content, with the aim of creating different ways of generating social media content.

Literature review with links.

The project is holding both Deepfake and Faceswap programs. Since these programs are considered more complex and complicated than what was taught in this course, they require beforehand research including tutorials, pre-trained models, and some repositories.

The Deepfake project was created with the help of the website article named “Into the Cageverse — Deepfaking with Autoencoders: An Implementation in Keras and Tensorflow” (<https://medium.com/gradientcrescent/deepfaking-nicolas-cage-into-the-mcu-using-autoencoders-an-implementation-in-keras-and-tensorflow-ab47792a042f>). All the while Faceswap project was developed with the help of “SimSwap” named tutorial on GitHub platform (<https://github.com/neuralchen/SimSwap>). Also, there was First Order Motion Model for Image Animation by Aliaksandr Siarohin, Stéphane Lathuilière, Sergey Tulyakov, Elisa Ricci, and Nicu Sebe involved in the creation of the Deepfake program, which was implemented in the code using the cloning function (<https://github.com/AliaksandrSiarohin/first-order-model>).

Current Work

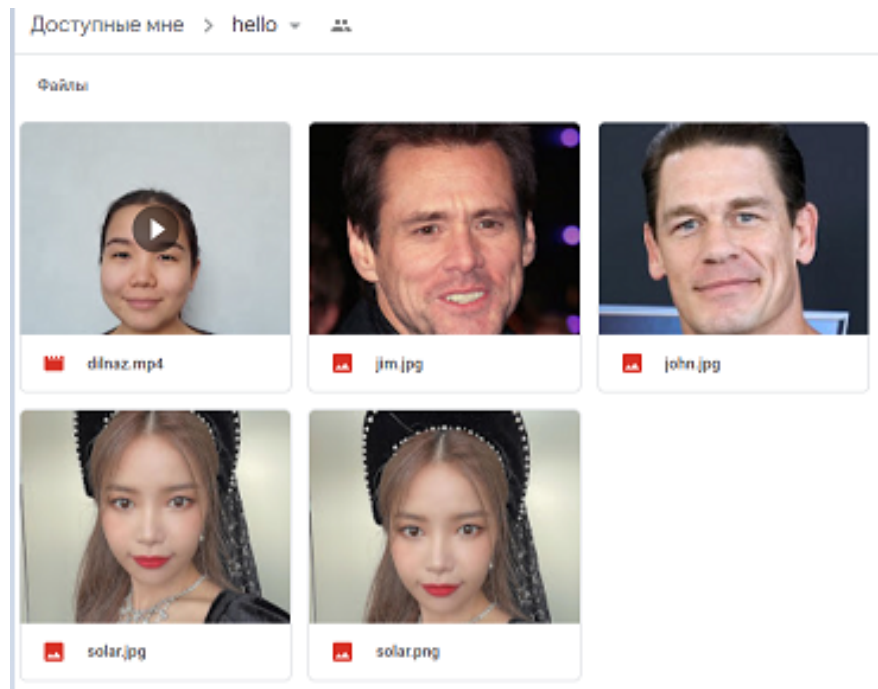
This project is created using OpenCV and Tensorflow libraries with Python as a programming language. As data storage, this project uses Google Drive, where the necessary media files, datasets, and ML model files are kept.

The project consists of a Deepfake program and a Faceswap program. Face swapping is the practice of switching a person's face with that of an animal, another person, or an inanimate object. Deepfakes are fictitious videos produced with the aid of computer software, machine learning, and face swapping. Deepfakes are computer-generated fake videos that portray events or actions that never actually occurred. They are formed by combining photos to create new footage.

Data and Methods

Information about the data.

To develop the project, video and images from the Google Drive folder were used. It was necessary to record a video with different emotions in order to apply facial expressions to the photo. Pictures for DeepFake and FaceSwap were gathered from the internet. The video is saved in .mp4 format and the images are in jpg and png format. The screen of the folder “hello” will be attached below.



Description of ML models.

Models for DeepFake: vox-cpk.pth.tar, demo.py


Checkpoints are created in order to capture every variable of the model (vox-cpk.pth.tar). The generator transforms the source image according to the values of keypoints. This generator follows the Johnson architecture. And demo.py model is created in order to generate the final result and output it. The make_animation function was used from that model.

Model for FaceSwap: shape_predictor_68_face_landmarks.dat

Shape predictors, also known as landmark predictors, are used to forecast the important (x, y)-coordinates of a given "shape." The dlib facial landmark predictor is the most common and well-known shape predictor for determining the precise location of individual facial characteristics.

Results

№	Code	Output
1	<pre> !git clone https://github.com/AliaxsandrSiarohin/first-order-model %cd first-order-model !pip install ffmpeg-python !pip install imageio !pip install imageio-ffmpeg </pre>	<pre> Cloning into 'first-order-model'... remote: Enumerating objects: 337, done. remote: Counting objects: 100% (31/31), done. remote: Compressing objects: 100% (24/24), done. remote: Total 337 (delta 15), reused 19 (delta 7), pack-reused 386 Receiving objects: 100% (337/337), 71.10 MiB 24.32 MiB/s, done. Resolving deltas: 100% (173/173), done. Updating files: 100% (47/47), done. /content/first-order-model Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-ubuntu/public-wheel/ Collecting ffmpeg-python Downloading ffmpeg-python-0.2.0-py3-none-any.whl (25 kB) Requirement already satisfied: future in /usr/local/lib/python3.8/dist-packages (from ffmpeg-python) (0.16.0) Installing collected packages: ffmpeg-python Successfully installed ffmpeg-python-0.2.0 Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-ubuntu/public-wheel/ Requirement already satisfied: imageio in /usr/local/lib/python3.8/dist-packages (2.9.0) Requirement already satisfied: numpy in /usr/local/lib/python3.8/dist-packages (from imageio) (1.21.0) Requirement already satisfied: pillow in /usr/local/lib/python3.8/dist-packages (from imageio) (7.1.2) Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-ubuntu/public-wheel/ Collecting imageio-ffmpeg Downloading imageio_ffmpeg-0.4.8-py3-none-manylinux2010_x86_64.whl (26.9 MB) Installing collected packages: imageio-ffmpeg Successfully installed imageio-ffmpeg-0.4.8 </pre>
2	<pre> from google.colab import drive drive.mount('/content/gdrive') </pre>	Connected to the Google Drive
3	<pre> [] from demo import make_animation from skimage import img_as_ubyte picture = imageio.imread('/content/gdrive/My Drive/'+photo) vid = imageio.get_reader('/content/gdrive/My Drive/'+video) fps = vid.get_meta_data()['fps'] result_video = [] try: for im in vid: result_video.append(im) except RuntimeError: pass vid.close() picture = resize(picture, (256, 256))[..., :3] result_video = [resize(frame, (256, 256))[..., :3] for frame in result_video] predictions = make_animation(picture, result_video, generator, kp_detector, relative=True, adapt_movement_scale=True) </pre>	Final video is resized and saved
4	<pre> [] HTML(show(picture, result_video, predictions).to_html5_video()) </pre>	
5	<pre> pic = detector(img_gr) for face in pic: landmarks = predictor(img_gr, face) lpoints = [] for n in range(0, 68): x = landmarks.part(n).x y = landmarks.part(n).y lpoints.append((x, y)) points = np.array(lpoints, np.int32) convexhull = cv2.convexHull(points) cv2.fillConvexPoly(mask, convexhull, 255) face_image_1 = cv2.bitwise_and(img, img, mask=mask) </pre>	

6	<pre> pic2 = detector(img2_gr) for face in pic2: landmarks = predictor(img2_gr, face) lpoints2 = [] for n in range(0, 68): x = landmarks.part(n).x y = landmarks.part(n).y lpoints2.append((x, y)) points2 = np.array(lpoints2, np.int32) convexhull2 = cv2.convexHull(points2) lsm = np.zeros_like(img_gr) ls_new_face = np.zeros_like(img2) </pre>	
7	<pre> img2_face_mask = np.zeros_like(img2_gr) img2_head_mask = cv2.fillConvexPoly(img2_face_mask, convexhull2, 255) img2_face_mask = cv2.bitwise_not(img2_head_mask) img2_head = cv2.bitwise_and(img2, img2, mask=img2_face_mask) result = cv2.add(img2_head, img2_new_face) (x, y, w, h) = cv2.boundingRect(convexhull2) cntr_face2 = (int((x + x + w) / 2), int((y + y + h) / 2)) </pre>	

Discussion

Critical Review of Results.

At the stage of choosing a project topic, it was decided to create Faceswap and Deepfake programs. However, at the stage of code development, it was discovered that both of these projects are more complicated than was anticipated, which is why a part of this project, which is Deepfake, heavily relies on the repository models mentioned above. Thus, although this Deepfake project is implemented successfully, the results are underwhelming due to the fact that it is not entirely built manually.

Furthermore, it can be noticed that the Face swapping of two pictures is not fully seamless and precise. It is due to the fact the triangulation, and the face reconstruction are not accurate and need further improvements.

Next Steps.

To improve the project and its accuracy, models of FaceSwap and DeepFake should be trained more. In the face swap part, the triangulation function was used to define the facial characteristics of source images. But the result wasn't clear enough, it should be improved to get a more accurate picture of the face swap. The same goes for the Deepfake part. The source image did not portray the same emotions as in the source video, especially the eyes. Also, for Deepfake it will be better to do a code optimization because the project runs more slowly than it is expected to. The models should be developed and supplemented so the accuracy of the project will be high.

References

<https://medium.com/gradientcrescent/deepfaking-nicolas-cage-into-the-mcu-using-autoencoders-an-implementation-in-keras-and-tensorflow-ab47792a042f>
<https://github.com/neuralchen/SimSwap>
<https://github.com/AliaksandrSiarohin/first-order-model>
<https://www.webwise.ie/news/explained-what-are-deepfakes/#:~:text=Deepfakes%20are%20fake%20videos%20created,results%20can%20be%20quite%20convincing.>
<https://www.kaggle.com/datasets/sergiovirahonda/shape-predictor-68-face-landmarksdat>
<https://opencv.org/>