```
# IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES
# TO THE CORRECT LOCATION (/kaggle/input) IN YOUR NOTEBOOK,
# THEN FEEL FREE TO DELETE THIS CELL.
# NOTE: THIS NOTEBOOK ENVIRONMENT DIFFERS FROM KAGGLE'S PYTHON
# ENVIRONMENT SO THERE MAY BE MISSING LIBRARIES USED BY YOUR
# NOTEBOOK.

import os
import sys
from tempfile import NamedTemporaryFile
from urllib.request import urlopen
from urllib.parse import unquote, urlparse
from urllib.error import HTTPError
from zipfile import ZipFile
import tarfile
import shutil

CHUNK_SIZE = 40960
DATA_SOURCE_MAPPING = 'first-order-motion:https%3A%2F%2Fstorage.googleapis.com%2Fkaggle-data-sets%2F1453154%2F2402876%2Fbundle%2Farchive.zip%

KAGGLE_INPUT_PATH='/kaggle/input'
KAGGLE_WORKING_PATH='/kaggle/working'
KAGGLE_SYMLINK='kaggle'

!umount /kaggle/input/ 2> /dev/null
shutil.rmtree('/kaggle/input', ignore_errors=True)
os.makedirs(KAGGLE_INPUT_PATH, 0o777, exist_ok=True)
os.makedirs(KAGGLE_WORKING_PATH, 0o777, exist_ok=True)

try:
  os.symlink(KAGGLE_INPUT_PATH, os.path.join("..", 'input'), target_is_directory=True)
except FileExistsError:
  pass
try:
  os.symlink(KAGGLE_WORKING_PATH, os.path.join("..", 'working'), target_is_directory=True)
except FileExistsError:
  pass

for data_source_mapping in DATA_SOURCE_MAPPING.split(','):
    directory, download_url_encoded = data_source_mapping.split(':')
    download_url = unquote(download_url_encoded)
    filename = urlparse(download_url).path
    destination_path = os.path.join(KAGGLE_INPUT_PATH, directory)
    try:
        with urlopen(download_url) as fileres, NamedTemporaryFile() as tfile:
            total_length = fileres.headers['content-length']
            print(f'Downloading {directory}, {total_length} bytes compressed')
            dl = 0
            data = fileres.read(CHUNK_SIZE)
            while len(data) > 0:
                dl += len(data)
                tfile.write(data)
                done = int(50 * dl / int(total_length))
                sys.stdout.write(f"\r[{'=' * done}{' ' * (50-done)}] {dl} bytes downloaded")
                sys.stdout.flush()
                data = fileres.read(CHUNK_SIZE)
            if filename.endswith('.zip'):
              with ZipFile(tfile) as zfile:
                zfile.extractall(destination_path)
            else:
              with tarfile.open(tfile.name) as tarfile:
                tarfile.extractall(destination_path)
            print(f'\nDownloaded and uncompressed: {directory}')
    except HTTPError as e:
        print(f'Failed to load (likely expired) {download_url} to path {destination_path}')
        continue
    except OSError as e:
        print(f'Failed to load {download_url} to path {destination_path}')
        continue

print('Data source import complete.')
```

```
Downloading first-order-motion, 1623427942 bytes compressed
[==================================================] 1623427942 bytes downloaded
```

```
Downloaded and uncompressed: first-order-motion
Data source import complete.
```

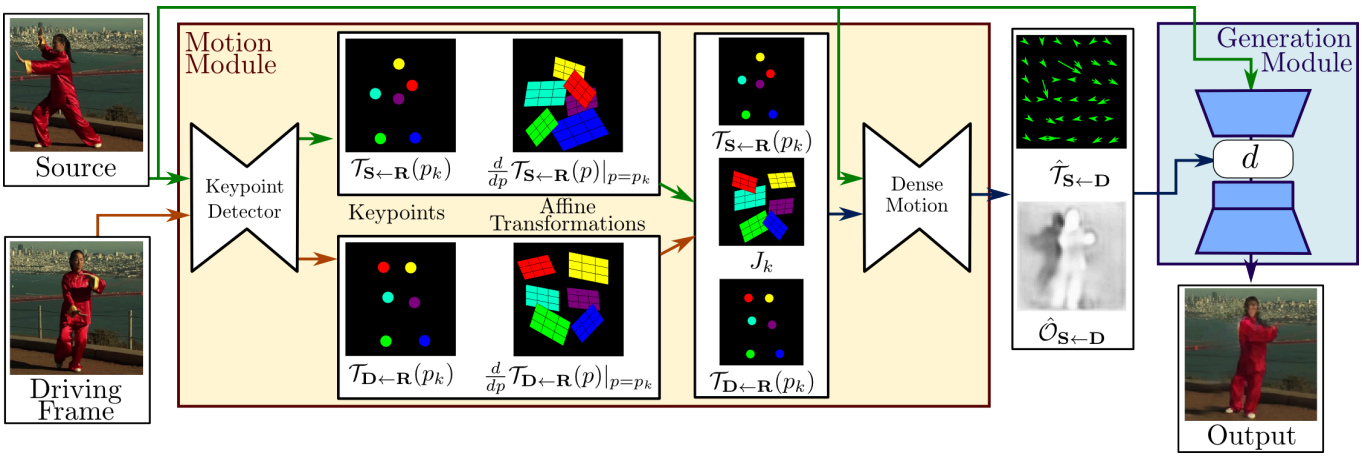
First Order Motion Model for Image Animation

by Aliaksandr Siarohin, Stéphane Lathuilière, Sergey Tulyakov, Elisa Ricci and Nicu Sebe

Image animation consists of generating a video sequence so that an object in a source image is animated according to the motion of a driving video This Framework addresses this problem without using any annotation or prior information about the specific object to animate.Once trained on a set of videos depicting objects of the same category (e.g. faces, human bodies), this can be applied to any object of this class.

The model consist of two parts:

- Motion estimation model : predict dense motion fields
- Generation model : use source image and results from motion estimation model to generate frame.



Installing Dependencies

- OpenCV : for reading images, video and face detection using cascade classifier to crop faces from images
- First Motion Model: cloning first order motion from github
- numpy, matplotlib : for array manipulation and plotting

```
!pip install opencv-contrib-python
```

```
Requirement already satisfied: opencv-contrib-python in /usr/local/lib/python3.10/dist-packages (4.8.0.76)
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.10/dist-packages (from opencv-contrib-python) (1.25.2)
```

```
!git clone https://github.com/GTGaganReddy/Projectmldl.git
```

```
Cloning into 'Projectmldl'...
remote: Enumerating objects: 71, done.
remote: Total 71 (delta 0), reused 0 (delta 0), pack-reused 71
Receiving objects: 100% (71/71), 34.47 MiB | 19.83 MiB/s, done.
Resolving deltas: 100% (15/15), done.
```

```
pip install --upgrade opencv-python
```

```
Requirement already satisfied: opencv-python in /usr/local/lib/python3.10/dist-packages (4.8.0.76)
Collecting opencv-python
  Downloading opencv_python-4.9.0.80-cp37-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (62.2 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.2/62.2 MB 1.3 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.10/dist-packages (from opencv-python) (1.25.2)
Installing collected packages: opencv-python
  Attempting uninstall: opencv-python
    Found existing installation: opencv-python 4.8.0.76
    Uninstalling opencv-python-4.8.0.76:
      Successfully uninstalled opencv-python-4.8.0.76
Successfully installed opencv-python-4.9.0.80
```

```
import cv2
import os
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import warnings
import urllib.request
warnings.filterwarnings("ignore")
from IPython.display import HTML


%matplotlib inline
```

Helper Functions

- imread : read and load image from path or url in numpy array.
- vidread : read and load video from path or url as list of frames in numpy array.
- vidsave : saving video file
- display : display video and images as html
- display_image_grid : display images as grid

```
'''
    params:
        img_path : path or url to image
        size : size of image to resize, default: None (do not resize)
        scale : scale image between 0 and 1 by dividing with 255.0, default: True
    return:
        image as numpy array
'''
def imread(img_path, size=None, scale=True):
    if img_path.startswith("http://") or img_path.startswith("https://") or img_path.startswith("www."):
        resp = urllib.request.urlopen(img_path)
        img = np.asarray(bytearray(resp.read()), dtype="uint8")
        img = cv2.imdecode(img, cv2.IMREAD_COLOR)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        if size is not None:
            img = cv2.resize(img, size)
        if scale:
            img = np.array(img/255.0)
        return img
    img = cv2.cvtColor(cv2.imread(img_path), cv2.COLOR_BGR2RGB)
    img = cv2.resize(img, size)
    if scale:
        img = np.array(img/255.0)
    return img


'''
    params:
        video_path : path or url to video
        size : size of frame to resize, default: None (do not resize)
        scale : scale image between 0 and 1 by dividing with 255.0, default: True
    return:
        list of video frames as numpy array
'''
def vidread(video_path, size=None, scale=True):
    vc = cv2.VideoCapture(video_path)
    vid = []
    while vc.isOpened():
        ret, img = vc.read()
        if not ret:
            break
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        if size is not None:
            img = cv2.resize(img, size)
        if scale:
            img = img/255.0
        vid.append(img)
    vc.release()
    return vid
```

```
'''
    params:
        save_path : path to save video
        frames : list of video frames as numpy array
        fps : framerate of video to save, default: 20
        size : size of video frames to save, default: (256,256)
    return:
        None
'''
def vidsave(save_path, frames, fps=20, size=(256,256)):
    #revert scaling factor of image and convert to uint8
    frames = np.array(frames)*255.0
    frames = frames.astype(np.uint8)
    writer = cv2.VideoWriter(save_path,
                     cv2.VideoWriter_fourcc(*'MJPG'),
                     fps, size)
    for frame in frames:
        writer.write(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
    writer.release()


'''
    params:
        source : source image
        driving : frames of driving video
        generated : frames of generated video
    return:
        animation to display in html
'''
def display(source, driving, generated=None):
    fig = plt.figure(figsize=(8 + 4 * (generated is not None), 6))

    ims = []
    for i in range(len(driving)):
        cols = [source]
        cols.append(driving[i])
        if generated is not None:
            cols.append(generated[i])
        im = plt.imshow(np.concatenate(cols, axis=1), animated=True)
        plt.axis('off')
        ims.append([im])

    ani = animation.ArtistAnimation(fig, ims, interval=50, repeat_delay=1000)
    plt.close()
    return ani


'''
    params:
        images : list of images to display
    return:
        None
'''
def display_image_grid(images):
    plt.title("Plot Images")
    plt.axis('off')
    plt.imshow(np.concatenate(images, axis=1))


'''
    changing current working path to first-order-model repo.
'''
prev_path = os.getcwd()
os.chdir("Projectmldl")
```

Loading pretrained model

- model is created using config file and model checkpoints are loaded.
- We have two models:
    1. keypoint detector
    2. generator model

```
config_path = "config/vox-256.yaml"
checkpoint_path = "../../input/checkpointaftertraining/checkpointFinal.pth.tar"
```

```
!pip install ffmpeg-python
```

```
Collecting ffmpeg-python
  Downloading ffmpeg_python-0.2.0-py3-none-any.whl (25 kB)
Requirement already satisfied: future in /usr/local/lib/python3.10/dist-packages (from ffmpeg-python) (0.18.3)
Installing collected packages: ffmpeg-python
Successfully installed ffmpeg-python-0.2.0
```

```
from demo import load_checkpoints, make_animation

generator, kp_detector = load_checkpoints(
    config_path=config_path,
    checkpoint_path=checkpoint_path
)
```

Source image

- source image is loaded from path in system
- using a driving video we animate source image

```
source_image_path = "/kaggle/input/gagang/WhatsApp Image 2024-03-03 at 20.18.34.jpeg"
driving_video_path = "../../input/first-order-motion/10.mp4"
```

here we generate animated video of source image using motion of driving image

Face detection to crop faces from image

- It uses faces haarcascde to detect faces which is fast and give decent results.
- We detect all faces and return list of crops of these faces.

```
'''
    params:
        image : image to crop faces from
        haarcascade_path : path to haarcascde xml file
        size : size of cropped face images
        margin_around : margin around detect box around face.
            this is used to include some parts around faces like hair, neck soulders etc. value is tuned based on output needed.
    return:
        list of cropped images of faces
'''
def crop_faces(image, haarcascade_path, size=(256,256), margin_around=50):
    gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
    detector = cv2.CascadeClassifier(haarcascade_path)
    rects = detector.detectMultiScale(gray, scaleFactor=1.05, minNeighbors=10, minSize=(30,30), flags=cv2.CASCADE_SCALE_IMAGE)
    crop_images = []
    for x, y, w, h in rects:
        a = 0 if y - margin_around < 0 else y - margin_around
        b = 0 if x - margin_around < 0 else x - margin_around
        img = image[a:y + h + margin_around, b:x + w + margin_around]
        img = cv2.resize(img, size)
        crop_images.append(img)
    return crop_images
```

downloading face haarcascade xml file from opencv github

```
!wget https://raw.githubusercontent.com/opencv/opencv/master/data/haarcascades/haarcascade_frontalface_default.xml
```

```
--2024-04-02 16:55:44--  https://raw.githubusercontent.com/opencv/opencv/master/data/haarcascades/haarcascade_frontalface_default.xml
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 930127 (908K) [text/plain]
```

```
Saving to: ‘haarcascade_frontalface_default.xml’

haarcascade_frontal 100%[===================>] 908.33K  --.-KB/s    in 0.05s

2024-04-02 16:55:45 (19.7 MB/s) - ‘haarcascade_frontalface_default.xml’ saved [930127/930127]
```

Source Image from URL

- loading image from a internet using its url.

```
source_image_url = "https://tse1.mm.bing.net/th?id=OIP.Y2JUsTgVo9B5FlgxAxytoQAAAA&pid=Api&P=0&h=180"
driving_video_path = "../../input/first-order-motion/00.mp4"


source_image = imread(source_image_url, size=None, scale=False)
driving_video = vidread(driving_video_path, size=(256,256))


display_image_grid([source_image])
```


```
haarcascade_path = "haarcascade_frontalface_default.xml"
margin_around = 20
```

Get all cropped face images from source image to transfer motion from driving video.

```
crop_images = crop_faces(source_image, haarcascade_path=haarcascade_path, size=(256,256), margin_around=margin_around)
crop_images = np.array(crop_images)/255.0


print("Number of Faces in image: ", len(crop_images))
display_image_grid(crop_images)
```

```
Number of Faces in image:  1
```

Select a cropped image from all cropped faces we get using its index

```
face_index = 0
```

```
HTML(display(crop_images[face_index], driving_video).to_html5_video())
```

creating and saving animated video based on motion from driving video

```
os.chdir(prev_path)
!rm -rf first-order-model
```