REAL-TIME IMAGE ANIMATION USING DEEP LEARNING

A project report submitted in partial fulfillment of the requirements for

the award of the degree of

Bachelor of Technology

by

T. Varshitha : 2111CS020624

Under the guidance of

Siva Kumar

Assistant Professor

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING
MALLA REDDY UNIVERSITY
(As per Telangana State Private Universities Act No.13 of 2020 and G.O.Ms.No.14, Higher Education (UE) Department)
HYDERABAD – 500043
TELANAGANA
INDIA
2023-24

MALLA REDDY UNIVERSITY

(As per Telangana State Private Universities Act No.13 of 2020 and

G.O.Ms.No.14, Higher Education (UE) Department)

HYDERABAD – 500043

TELANAGANA

Certificate

This is to certify that this is the bonafide record of the application development entitled,**" REAL-TIME IMAGE ANIMATION USING DEEP LEARNING"** submitted by **T.Varshitha (2111CS020624)** of B.Tech III year II semester, Department of CSE (AI&ML) during the year 2023- 24.The results embodied in the report have not been submitted to any other university or institute for the awardof any degree or diploma.

INTERNAL GUIDE
A. Siva Kumar
Assistant Professor

HEAD OF THE DEPARTMENT
Dr. Thayyaba Khatoon
Professor & HoD

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

We would like to express our gratitude to all those who extended their support and suggestions tocome up with this application. Special Thanks to our guide Siva Kumar whose help and stimulating suggestions and encouragement helped us all time in the due course of project development.

We sincerely thank our HOD Dr. Thayyaba Khatoon for her constant support and motivation all the time. A special acknowledgement goes to a friend who enthused us from the back stage. Last but not the least our sincere appreciation goes to our family who has been tolerant understanding our moods, and extending timely support.

ABSTRACT

This project delves into deep learning-based image animation, employing conditional generative models like Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs). Trained on datasets comprising image-sequence pairs, these models transform single input images into coherent and novel animations, simulating natural movements and transformations. An interactive image animation system is introduced, implemented in a Jupyter notebook environment using TensorFlow for deep learning capabilities. Leveraging OpenCV, FFmpeg, ImageIO, PIL, and scikit-image for image and video processing, the system incorporates IPython widgets for enhanced user interaction. The technology also plays a crucial role in live video streaming, providing dynamic visual content without the need for manual frame-by-frame animation. This project harnesses the power of deep learning to eliminate manual efforts, opening new possibilities for efficient and realistic content creation in diverse domains.

CONTENTS

1. INTRODUCTION

This cutting-edge project pioneers the realm of image animation through the utilization of advanced deep learning techniques, specifically focusing on conditional generative models such as Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs). By training on meticulously curated datasets containing pairs of images and their corresponding sequences, these models are adept at transforming static images into dynamic and cohesive animations, imbuing them with lifelike movements and transformations.

Central to this project is the introduction of an interactive image animation system, ingeniously implemented within a Jupyter notebook environment using TensorFlow for its robust deep learning capabilities. This system seamlessly integrates a host of powerful image and video processing libraries including OpenCV, FFmpeg, ImageIO, PIL, and scikit-image. Furthermore, it enhances user engagement and control through the incorporation of IPython widgets, fostering a fluid and intuitive interaction with the animation creation process.

Beyond its role in static image animation, this technology demonstrates its versatility by facilitating live video streaming with dynamic visual content, all achieved without the arduous task of manual frame-by-frame animation. By harnessing the unparalleled capabilities of deep learning, this project revolutionizes content creation across diverse domains, effectively eliminating manual efforts and ushering in a new era of efficient and hyper-realistic visual storytelling.

1.1 PROBLEM DEFINITION

The project tackles the inefficiencies and constraints of traditional image and video animation methods by harnessing the power of deep learning. It seeks to automate the animation process, ensuring that single input images are transformed into cohesive and lifelike animations without manual intervention. By focusing on realism and fluidity, the system aims to produce animations that exhibit natural movements and transformations. Through an intuitive user interface implemented in a Jupyter notebook environment, users can interactively control and customize the animation process. Additionally, the project aims to optimize computational efficiency, enabling

real-time or near real-time animation generation. Extending its capabilities to live video streaming, the technology promises dynamic visual content without the need for laborious frame-by-frame animation, revolutionizing content creation across various domains.

1.2 OBJECTIVE OF PROJECT

Automated Animation Generation: Develop algorithms and models capable of automatically transforming single input images into coherent and dynamic animations, reducing or eliminating the need for manual frame-by-frame animation.

Realism and Cohesion: Ensure that the generated animations exhibit natural movements, transformations, and visual coherence, enhancing their realism and appeal to viewers.

Interactive User Interface: Implement an intuitive and user-friendly interface enabling users to interactively control and customize the animation process according to their preferences and requirements.

Computational Efficiency: Optimize algorithms and techniques to be computationally efficient and scalable, enabling real-time or near real-time animation generation even with large datasets and complex models.

Application to Live Video Streaming: Extend the capabilities of the system to support live video streaming, enabling the creation of dynamic visual content without the need for manual intervention, and thus opening up new possibilities for interactive and engaging multimedia experiences.

1.3 LIMITATIONS

Training Data Quality: The quality and diversity of the training data may impact the realism and variety of generated animations.

Hardware Requirements: The computational resources required for training and inference may be substantial, potentially limiting accessibility to users with limited computing power.

Generalization: The models developed may exhibit limitations in generalizing to unseen data or scenarios, potentially leading to unrealistic or unexpected animation outputs.

Complexity of Animations: Generating complex animations with multiple objects or intricate movements may pose challenges for the models, potentially leading to less satisfactory results.

2. ANALYSIS

2.1 LITERATURE SURVEY

1. First-order Motion Model for Image Animation" by Aliaksandr Siarohin et al. (2019):

This paper introduces the first-order motion model, a deep learning-based approach for image animation. The model learns to transfer the motion from a driving video to a target image, generating realistic animations. The authors demonstrate the effectiveness of the model on various tasks, including face animation, object manipulation, and character animation.

2. Liquid Warping GAN: A Unified Framework for Human Motion Imitation, Appearance Transfer and Novel View Synthesis" by Mingyu Liang et al. (2019):

This paper presents the Liquid Warping GAN (LWGAN), a deep learning framework for human motion imitation, appearance transfer, and novel view synthesis. The LWGAN combines geometric warping with generative adversarial networks (GANs) to achieve high-quality image animation results across different domains, such as face animation and human motion imitation.

3. Few-Shot Adversarial Learning of Realistic Neural Talking Head Models" by Egor Zakharov et al. (2019):

In this paper, the authors propose a few-shot adversarial learning approach for generating realistic neural talking head models from a small number of input images. The method leverages deep learning techniques, including generative adversarial networks (GANs) and few-shot learning, to synthesize expressive talking head animations that closely resemble the input subject.

4. Deep Video Portraits by Justus Thies et al. (2019):

This paper introduces the concept of deep video portraits, where deep learning models are used to animate static portraits by transferring the motion from a source video. The authors demonstrate the capability of deep video portraits to generate high-quality animations of static images, including facial expressions and head movements, using a single input video.

5. Liquid Warping GAN++: A Unified Framework for Human Motion Imitation, Appearance Transfer and Novel View Synthesis with Improved Consistency and Quality" by Mingyu Liang et al. (2020):

Building upon their previous work, the authors propose an enhanced version of the Liquid Warping GAN (LWGAN++) framework for human motion imitation, appearance transfer, and novel view

synthesis. The LWGAN++ improves consistency and quality in image animation tasks by incorporating additional loss functions and refinement mechanisms into the model architecture.

2.2 SOFTWARE REQUIREMENT SPECIFICATION

The purpose of this software is to develop an image animation system that utilizes deep learning techniques to generate animations from single input images.

2.2.1 SOFTWARE REQUIREMENT

- NVIDIA GPU with CUDA support for accelerated deep learning computations (e.g., NVIDIA GeForce RTX series or Tesla GPUs).
- Minimum 16GB RAM for handling large datasets and model training.
- ffmpeg-python==0.2.0
- imageio==2.22.0
- imageio-ffmpeg==0.4.7
- matplotlib==3.6.0
- numpy==1.23.3
- pandas==1.5.0
- python-dateutil==2.8.2
- pytz==2022.2.1
- PyYAML==6.0
- scikit-image==0.19.3
- scikit-learn==1.1.2
- scipy==1.9.1
- torch==1.12.1
- torchvision==0.13.1
- tqdm==4.64.1

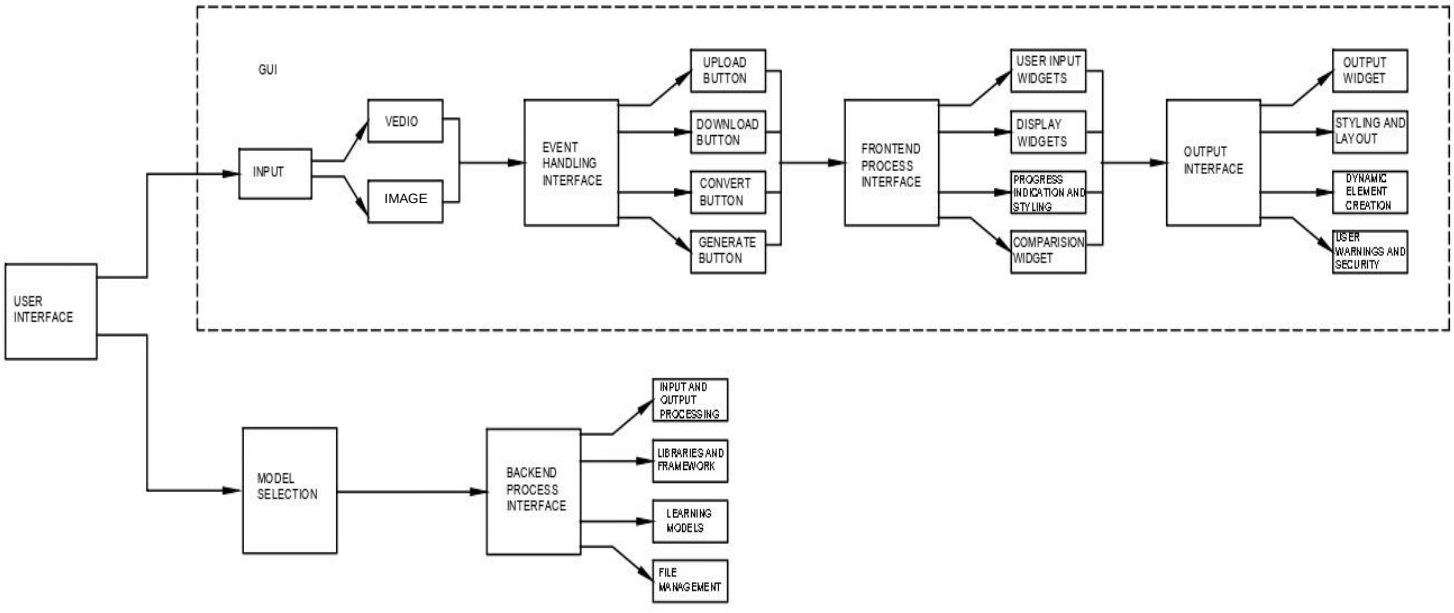
2.2.2 HARDWARE REQUIREMENT

- High-resolution monitor (1920x1080 or higher) for visualizing images, videos, and animation outputs.

- The system should be compatible with major operating systems including Windows 10 or later
- Jupyter notebook environment for interactive development and experimentation.

2.2 MODEL SELECTION AND ARCHITECTURE

The selection of model architecture for the Image Animation System involves choosing between conditional generative models like Generative Adversarial Networks (GANs) or Variational Autoencoders (VAEs), each tailored to the task of image-to-image translation and animation generation. GANs consist of a generator network mapping input images to outputs and a discriminator network distinguishing real from generated images, while VAEs encode input images into latent representations and decode them back to the original space. Hybrid approaches, such as VAE-GANs, combine the strengths of both models. Design considerations include capturing spatial and temporal dependencies, scalability, and memory efficiency, with pre-trained models and transfer learning techniques often employed for improved performance and efficiency, ultimately requiring experimentation to determine the most suitable architecture for a given task.

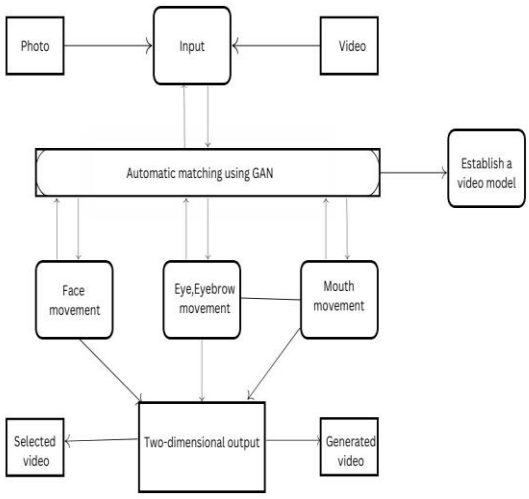


3. DESIGN

3.1 INTRODUCTION

The design of the Image Animation System encompasses a thoughtful and systematic approach to creating an intuitive, efficient, and flexible platform for generating animations from static images. At its core, the design philosophy prioritizes user experience, computational efficiency, and scalability. User-centric design principles guide the development of an interactive interface, allowing users to seamlessly navigate through the animation creation process while providing ample opportunities for experimentation and customization. The system's architecture is meticulously crafted to harness the power of deep learning, incorporating cutting-edge techniques such as conditional generative models and transfer learning to ensure high-quality animation outputs. Emphasis is placed on modularity and extensibility, enabling easy integration of new features, algorithms, and datasets as the system evolves. Furthermore, attention is given to optimization strategies to maximize computational resources and minimize training time, thereby enhancing productivity and accessibility for users across various domains. Overall, the design of the Image Animation System reflects a commitment to innovation, usability, and performance, empowering users to unleash their creativity and bring their ideas to life with unparalleled ease and efficiency.

3.2 BLOCK DIAGRAM



3.3 DATASET DESCRIPTION

1. **BAIR Dataset:**
    - The BAIR (Berkeley AI Research) dataset is a collection of videos created for research purposes, particularly in the field of artificial intelligence and computer vision.
    - It consists of various types of videos, including human actions, object interactions, and dynamic scenes, captured under different lighting conditions and backgrounds.
    - The dataset is commonly used for tasks such as action recognition, video prediction, and generative modeling.
2. **MGIF Dataset:**
    - The MGIF (Moving GIF) dataset comprises a diverse collection of animated GIFs sourced from the web.
    - It contains a wide range of animated content, including short loops, cartoon animations, and reaction GIFs, covering various themes and styles.
    - The dataset serves as a valuable resource for training models for GIF generation, animation synthesis, and related tasks.
3. **Fashion Dataset:**
    - The Fashion dataset typically refers to a collection of fashion-related images or videos curated for research and development in the fashion industry and computer vision applications.
    - It includes images and videos of clothing items, accessories, fashion shows, and street style, encompassing different trends, styles, and aesthetics.
    - Researchers and practitioners use the Fashion dataset for tasks such as clothing recognition, style transfer, recommendation systems, and trend analysis.
4. **Taichi Dataset:**
    - The Taichi dataset is a dataset focusing on the martial art form of Tai Chi (Taiji), which encompasses a series of slow and controlled movements performed with precision and mindfulness.
    - It consists of videos capturing practitioners performing Tai Chi routines, sequences, or forms, filmed from various angles and perspectives.

- The dataset is utilized for research in action recognition, motion analysis, human pose estimation, and related fields, particularly in understanding the dynamics and nuances of Tai Chi movements.

5. **Nemo Dataset:**
    - The Nemo dataset is not a widely recognized dataset in the context provided. It could potentially refer to a specific dataset named Nemo, which might be related to underwater imagery, marine life, or oceanography. However, without further context, it's challenging to provide a precise description.
6. **VoxCeleb Dataset:**
    - The VoxCeleb dataset is a large-scale audio-visual dataset designed for speaker recognition and verification tasks.
    - It contains videos extracted from YouTube, featuring celebrities and public figures speaking in various contexts, such as interviews, speeches, and presentations.
    - Each video clip is accompanied by corresponding audio recordings, enabling researchers to explore multimodal approaches for speaker identification and verification.

3.4 DATA PREPROCESSING TECHNIQUES

Pre-processing is a crucial stage in preparing the dataset for training the deep learning models in the image animation system. It involves several key steps aimed at ensuring data consistency, quality, and compatibility with the chosen deep learning framework. The pre-processing steps include:

Data Collection: Gather a diverse and representative dataset comprising image-sequence pairs that cover a wide range of motions, actions, and transformations. Ensure that the dataset includes high-resolution images and videos captured under various lighting conditions, viewpoints, and backgrounds.

Data Annotation: Annotate the dataset with relevant metadata, such as object labels, action descriptions, and temporal information. This metadata facilitates training and evaluation by

providing additional context and semantics to the input data. Additionally, annotate key points or keypoints in the images to assist in motion tracking and alignment during training.

Data Cleaning: Perform data cleaning to remove any corrupted or irrelevant images and videos from the dataset. Ensure that the remaining data is consistent and free from artifacts, noise, or distortions that could negatively impact model training and performance.

Data Augmentation: Apply data augmentation techniques to increase the diversity and robustness of the dataset. Common augmentation techniques include random rotations, translations, scaling, cropping, flipping, and color jittering. Data augmentation helps prevent overfitting and improves the generalization ability of the trained models by simulating variations in the input data.

Image and Video Formatting: Convert the images and videos in the dataset to standard formats compatible with the chosen deep learning framework and libraries. Common formats include JPEG or PNG for images and MP4 or AVI for videos. Ensure consistency in format to facilitate smooth preprocessing and compatibility during training and inference stages.

Image and Video Resizing: Resize the images and videos in the dataset to a uniform size or resolution suitable for training the deep learning models. Resizing helps standardize the input dimensions and reduces computational complexity during model training and inference. Use interpolation techniques such as Lanczos or bilinear interpolation to preserve image quality during resizing.

Normalization: Normalize the pixel values of the images to a common scale or range to improve training stability and convergence speed. Common normalization techniques include rescaling pixel values to the range [0, 1] or [-1, 1]. Normalization helps mitigate the effects of varying brightness and contrast levels across different images in the dataset.

Data Splitting: Split the dataset into training, validation, and testing subsets to evaluate the performance of the trained models accurately. The training set is used to optimize model parameters, while the validation set is used to tune hyperparameters and prevent overfitting. The testing set is kept separate and used only for final evaluation to assess the generalization performance of the models on unseen data.

3.5 METHODS AND ALGORITHMS

The primary deep learning algorithm used in the provided script is the First Order Model (FOM) for image animation. The FOM is based on conditional generative models and employs a combination of neural networks to generate realistic animations from single input images and driving videos.

First Order Model (FOM): The core algorithm used for image animation in the script is the First Order Model. FOM is a generative model that operates in a conditional manner, meaning it generates animations conditioned on both an input image and a driving video. It consists of two main components:

- Generator Network: This network generates the frames of the animated sequence based on the input image and the motion information extracted from the driving video.
- Keypoint Detector Network: This network detects keypoints or landmarks in both the input image and the driving video. These keypoints capture the spatial information necessary to align the generated frames with the motion dynamics of the video.

Conditional Generative Models: FOM belongs to the family of conditional generative models, where the generation process is conditioned on additional information (in this case, the input image and driving video). By conditioning the generation process, FOM can produce animations that are coherent and consistent with both the appearance and motion characteristics of the input data.

Neural Network Architectures: The generator and keypoint detector networks in FOM are typically based on deep neural network architectures. These architectures may include convolutional neural networks (CNNs) for feature extraction and transformation, recurrent neural networks (RNNs) for temporal modeling, and other components designed to capture complex dependencies between the input image and the driving video.

4. DEPLOYMENT AND RESULTS

4.1 INTRODUCTION

The deployment of models and the presentation of results in the context of the Image Animation System represent pivotal stages that bridge the gap between research and practical application. This phase involves the integration of trained deep learning models into a production environment, enabling users to interact with the system and generate animations from static images effectively. The introduction of model deployment encompasses the setup of infrastructure, configuration of software dependencies, and implementation of user interfaces to facilitate seamless interaction. Additionally, the presentation of results involves showcasing the performance and efficacy of the deployed models through visual demonstrations, quantitative metrics, and user feedback. By providing a comprehensive overview of the deployment process and demonstrating the capabilities of the system through compelling results, this stage aims to validate the effectiveness of the developed solution and foster user engagement and adoption.

4.2 SOURCE CODE

```
import IPython.display
import PIL.Image
import cv2
import ffmpeg
import imageio
import io
import ipywidgets
import numpy
import os.path
import requests
import skimage.transform
import warnings
from base64 import b64encode
from demo import load_checkpoints, make_animation  # type: ignore (local file)
from google.colab import files, output
from IPython.display import HTML, Javascript
from shutil import copyfileobj
from skimage import img_as_ubyte
from tempfile import NamedTemporaryFile
```

```
from tqdm.auto import tqdm
warnings.filterwarnings("ignore")
os.makedirs("user", exist_ok=True)

display(HTML("""

"""))

def thumbnail(file):
        return imageio.get_reader(file, mode='I', format='FFMPEG').get_next_data()

def create_image(i, j):
        image_widget = ipywidgets.Image.from_file('demo/images/%d%d.png' % (i, j))
        image_widget.add_class('resource')
        image_widget.add_class('resource-image')
        image_widget.add_class('resource-image%d%d' % (i, j))
        return image_widget

def create_video(i):
        video_widget = ipywidgets.Image(
                value=cv2.imencode('.png', cv2.cvtColor(thumbnail('demo/videos/%d.mp4' % i),
cv2.COLOR_RGB2BGR))[1].tostring(),
                format='png'
        )
        video_widget.add_class('resource')
        video_widget.add_class('resource-video')
        video_widget.add_class('resource-video%d' % i)
        return video_widget

def create_title(title):
        title_widget = ipywidgets.Label(title)
        title_widget.add_class('title')
        return title_widget

def download_output(button):
        complete.layout.display = 'none'
        loading.layout.display = ''
        files.download('output.mp4')
        loading.layout.display = 'none'
        complete.layout.display = ''

def convert_output(button):
        complete.layout.display = 'none'
        loading.layout.display = ''
        ffmpeg.input('output.mp4').output('scaled.mp4',
vf='scale=1080x1080:flags=lanczos,pad=1920:1080:420:0').overwrite_output().run()
```

```
        files.download('scaled.mp4')
        loading.layout.display = 'none'
        complete.layout.display = ''

def back_to_main(button):
        complete.layout.display = 'none'
        main.layout.display = ''

label_or = ipywidgets.Label('or')
label_or.add_class('label-or')

image_titles = ['Peoples', 'Cartoons', 'Dolls', 'Game of Thrones', 'Statues']
image_lengths = [8, 4, 8, 9, 4]

image_tab = ipywidgets.Tab()
image_tab.children = [ipywidgets.HBox([create_image(i, j) for j in range(length)]) for i, length
in enumerate(image_lengths)]
for i, title in enumerate(image_titles):
        image_tab.set_title(i, title)

input_image_widget = ipywidgets.Output()
input_image_widget.add_class('input-widget')
upload_input_image_button = ipywidgets.FileUpload(accept='image/*', button_style='primary')
upload_input_image_button.add_class('input-button')
image_part = ipywidgets.HBox([
        ipywidgets.VBox([input_image_widget, upload_input_image_button]),
        label_or,
        image_tab
])

video_tab = ipywidgets.Tab()
video_tab.children = [ipywidgets.HBox([create_video(i) for i in range(5)])]
video_tab.set_title(0, 'All Videos')

input_video_widget = ipywidgets.Output()
input_video_widget.add_class('input-widget')
upload_input_video_button = ipywidgets.FileUpload(accept='video/*', button_style='primary')
upload_input_video_button.add_class('input-button')
video_part = ipywidgets.HBox([
        ipywidgets.VBox([input_video_widget, upload_input_video_button]),
        label_or,
        video_tab
])

model = ipywidgets.Dropdown(
        description="Model:",
```

```
        options=[
                'vox',
                'vox-adv',
                'taichi',
                'taichi-adv',
                'nemo',
                'mgif',
                'fashion',
                'bair'
        ]
)
warning = ipywidgets.HTML('Warning: Upload your own images and videos (see README)')
warning.add_class('warning')
model_part = ipywidgets.HBox([model, warning])

relative = ipywidgets.Checkbox(description="Relative keypoint displacement (Inherit object proporions from the video)", value=True)
adapt_movement_scale = ipywidgets.Checkbox(description="Adapt movement scale (Don’t touch unless you know want you are doing)", value=True)
generate_button = ipywidgets.Button(description="Generate", button_style='primary')
main = ipywidgets.VBox([
        create_title('Choose Image'),
        image_part,
        create_title('Choose Video'),
        video_part,
        create_title('Settings'),
        model_part,
        relative,
        adapt_movement_scale,
        generate_button
])

loader = ipywidgets.Label()
loader.add_class("loader")
loading_label = ipywidgets.Label("This may take several minutes to process…")
loading_label.add_class("loading-label")
progress_bar = ipywidgets.Output()
loading = ipywidgets.VBox([loader, loading_label, progress_bar])
loading.add_class('loading')

output_widget = ipywidgets.Output()
output_widget.add_class('output-widget')
download = ipywidgets.Button(description='Download', button_style='primary')
download.add_class('output-button')
download.on_click(download_output)
convert = ipywidgets.Button(description='Convert to 1920×1080', button_style='primary')
```

```
convert.add_class('output-button')
convert.on_click(convert_output)
back = ipywidgets.Button(description='Back', button_style='primary')
back.add_class('output-button')
back.on_click(back_to_main)

comparison_widget = ipywidgets.Output()
comparison_widget.add_class('comparison-widget')
comparison_label = ipywidgets.Label('Comparison')
comparison_label.add_class('comparison-label')
complete = ipywidgets.HBox([
        ipywidgets.VBox([output_widget, download, convert, back]),
        ipywidgets.VBox([comparison_widget, comparison_label])
])

display(ipywidgets.VBox([main, loading, complete]))
display(Javascript("""
var images, videos;
function deselectImages() {
        images.forEach(function(item) {
                item.classList.remove("selected");
        });
}
function deselectVideos() {
        videos.forEach(function(item) {
                item.classList.remove("selected");
        });
}
function invokePython(func) {
        google.colab.kernel.invokeFunction("notebook." + func, [].slice.call(arguments, 1), {});
}
setTimeout(function() {
        (images = [].slice.call(document.getElementsByClassName("resource-
image"))).forEach(function(item) {
                item.addEventListener("click", function() {
                        deselectImages();
                        item.classList.add("selected");
                        invokePython("select_image", item.className.match(/resource-
image(\d\d)/)[1]);
                });
        });
        images[0].classList.add("selected");
        (videos = [].slice.call(document.getElementsByClassName("resource-
video"))).forEach(function(item) {
                item.addEventListener("click", function() {
                        deselectVideos();
```

```
                        item.classList.add("selected");
                        invokePython("select_video", item.className.match(/resource-
video(\d)/)[1]);
                    });
            });
            videos[0].classList.add("selected");
}, 1000);
"""))

selected_image = None
def select_image(filename):
        global selected_image
        selected_image = resize(PIL.Image.open('demo/images/%s.png' %
filename).convert("RGB"))
        input_image_widget.clear_output(wait=True)
        with input_image_widget:
                display(HTML('Image'))
        input_image_widget.remove_class('uploaded')
output.register_callback("notebook.select_image", select_image)

selected_video = None
def select_video(filename):
        global selected_video
        selected_video = 'demo/videos/%s.mp4' % filename
        input_video_widget.clear_output(wait=True)
        with input_video_widget:
                display(HTML('Video'))
        input_video_widget.remove_class('uploaded')
output.register_callback("notebook.select_video", select_video)

def resize(image, size=(256, 256)):
        w, h = image.size
        d = min(w, h)
        r = ((w - d) // 2, (h - d) // 2, (w + d) // 2, (h + d) // 2)
        return image.resize(size, resample=PIL.Image.LANCZOS, box=r)

def upload_image(change):
        global selected_image
        for name, file_info in upload_input_image_button.value.items():
                content = file_info['content']
        if content is not None:
                selected_image = resize(PIL.Image.open(io.BytesIO(content)).convert("RGB"))
                input_image_widget.clear_output(wait=True)
                with input_image_widget:
                        display(selected_image)
                input_image_widget.add_class('uploaded')
```

```
                display(Javascript('deselectImages()'))
upload_input_image_button.observe(upload_image, names='value')

def upload_video(change):
        global selected_video
        for name, file_info in upload_input_video_button.value.items():
                content = file_info['content']
        if content is not None:
                selected_video = 'user/' + name
                with open(selected_video, 'wb') as video:
                        video.write(content)
                preview =
resize(PIL.Image.fromarray(thumbnail(selected_video)).convert("RGB"))
                input_video_widget.clear_output(wait=True)
                with input_video_widget:
                        display(preview)
                input_video_widget.add_class('uploaded')
                display(Javascript('deselectVideos()'))
upload_input_video_button.observe(upload_video, names='value')

def change_model(change):
        if model.value.startswith('vox'):
                warning.remove_class('warn')
        else:
                warning.add_class('warn')
model.observe(change_model, names='value')

def generate(button):
        main.layout.display = 'none'
        loading.layout.display = ''
        filename = model.value + ('' if model.value == 'fashion' else '-cpk') + '.pth.tar'
        if not os.path.isfile(filename):
                response = requests.get('https://github.com/graphemecluster/first-order-model-
demo/releases/download/checkpoints/' + filename, stream=True)
                with progress_bar:
                        with tqdm.wrapattr(response.raw, 'read',
total=int(response.headers.get('Content-Length', 0)), unit='B', unit_scale=True,
unit_divisor=1024) as raw:
                                with open(filename, 'wb') as file:
                                        copyfileobj(raw, file)
                progress_bar.clear_output()
        reader = imageio.get_reader(selected_video, mode='I', format='FFMPEG')
        fps = reader.get_meta_data()['fps']
        driving_video = []
        for frame in reader:
                driving_video.append(frame)
```

```
        generator, kp_detector = load_checkpoints(config_path='config/%s-256.yaml' %
model.value, checkpoint_path=filename)
        with progress_bar:
                predictions = make_animation(
                        skimage.transform.resize(numpy.asarray(selected_image), (256, 256)),
                        [skimage.transform.resize(frame, (256, 256)) for frame in driving_video],
                        generator,
                        kp_detector,
                        relative=relative.value,
                        adapt_movement_scale=adapt_movement_scale.value
                )
        progress_bar.clear_output()
        imageio.mimsave('output.mp4', [img_as_ubyte(frame) for frame in predictions], fps=fps)
        try:
                with NamedTemporaryFile(suffix='.mp4') as output:
                        ffmpeg.output(ffmpeg.input('output.mp4').video,
ffmpeg.input(selected_video).audio, output.name, c='copy').run()
                        with open('output.mp4', 'wb') as result:
                                copyfileobj(output, result)
        except ffmpeg.Error:
                pass
        output_widget.clear_output(True)
        with output_widget:
                video_widget = ipywidgets.Video.from_file('output.mp4', autoplay=False,
loop=False)
                video_widget.add_class('video')
                video_widget.add_class('video-left')
                display(video_widget)
        comparison_widget.clear_output(True)
        with comparison_widget:
                video_widget = ipywidgets.Video.from_file(selected_video, autoplay=False,
loop=False, controls=False)
                video_widget.add_class('video')
                video_widget.add_class('video-right')
                display(video_widget)
        display(Javascript("""
        setTimeout(function() {
                (function(left, right) {
                        left.addEventListener("play", function() {
                                right.play();
                        });
                        left.addEventListener("pause", function() {
                                right.pause();
                        });
                        left.addEventListener("seeking", function() {
                                right.currentTime = left.currentTime;
```

```
                    });
                    right.muted = true;
                })(document.getElementsByClassName("video-left")[0],
document.getElementsByClassName("video-right")[0]);
        }, 1000);
        """))
        loading.layout.display = 'none'
        complete.layout.display = ''

generate_button.on_click(generate)

loading.layout.display = 'none'
complete.layout.display = 'none'
select_image('00')
select_video('0')
```

4.2 MODEL IMPLEMENTATION AND TRAINING

There are 2 different ways of performing animation: by using **absolute** key point locations or by using **relative** key point locations.

- ***Animation using absolute coordinates:*** the animation is performed using the absolute positions of the driving video and appearance of the source image. In this way there are no specific requirements for the driving video and source appearance that is used. However this usually leads to poor performance since irrelevant details such as shape is transferred. Check animate parameters in taichi-256.yaml to enable this mode.
- ***Animation using relative coordinates:*** from the driving video we first estimate the relative movement of each key point, then we add this movement to the absolute position of key points in the source image. This key point along with source image is used for animation. This usually leads to better performance, however this requires that the object in the first frame of the video and in the source image have the same pose.

Training on your own dataset

1. Resize all the videos to the same size e.g 256x256, the videos can be in '.gif', '.mp4' or folder with images. We recommend the later, for each video make a separate folder with all the frames in '.png' format. This format is loss-less, and it has better i/o performance.

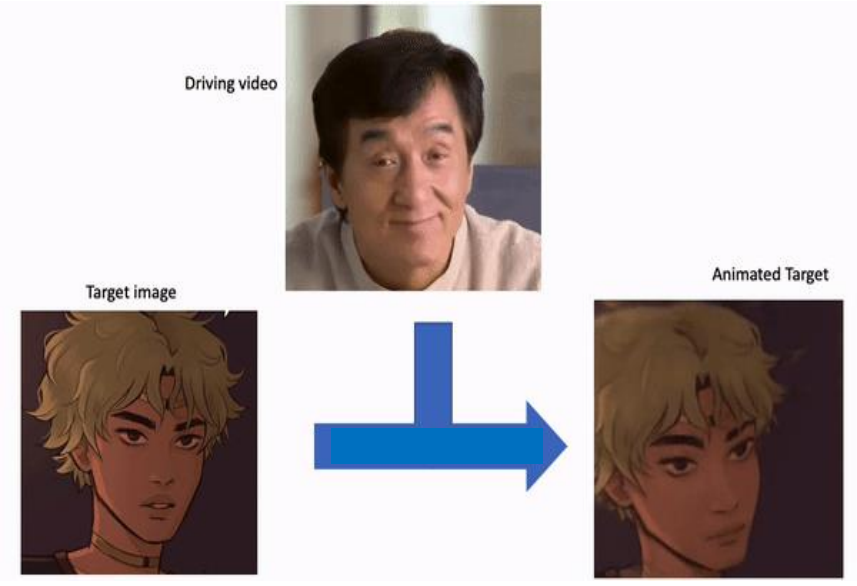
2. Create a folder data/dataset_name with 2 subfolders train and test, put training videos in the train and testing in the test.

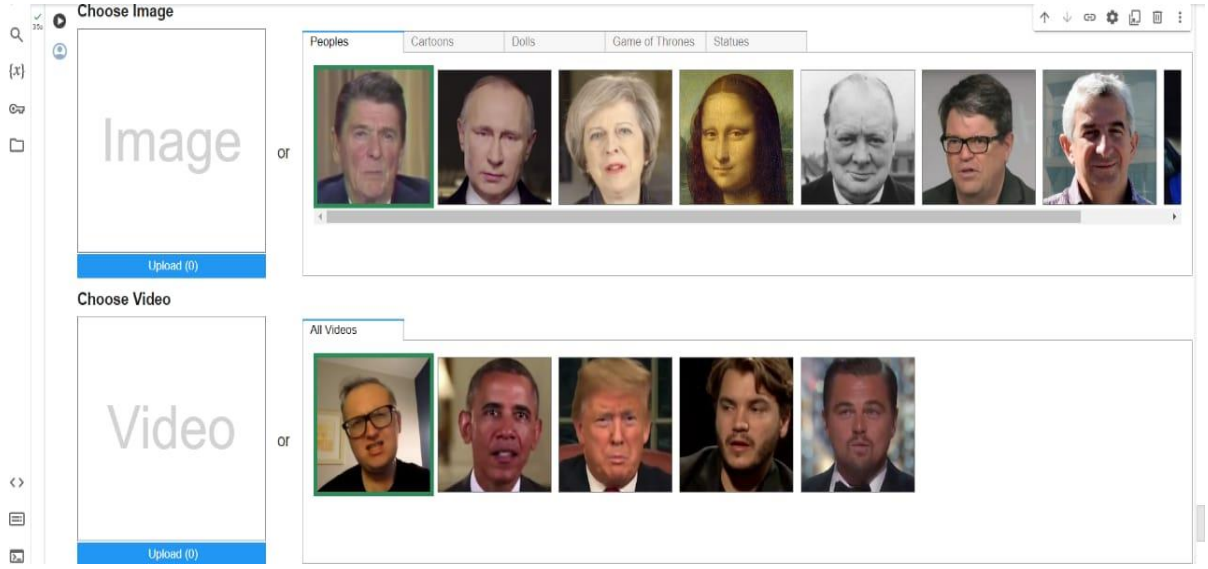
3. Create a config config/dataset_name.yaml, in dataset_params specify the root dir the root_dir: data/dataset_name. Also adjust the number of epoch in train_params.

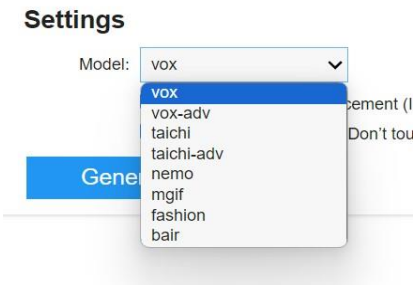
Model implementation and training in the Image Animation System involve several key steps aimed at effectively deploying deep learning models for animation generation from static images. The process begins with the selection of an appropriate model architecture, such as a Generative Adversarial Network (GAN) or a Variational Autoencoder (VAE), tailored to the task of image-to-image translation and animation synthesis. Once the architecture is chosen, the model is implemented using deep learning frameworks like TensorFlow or PyTorch, with careful attention to network design, hyperparameter tuning, and optimization techniques. Training data, consisting of image-sequence pairs, is prepared and preprocessed to ensure compatibility with the chosen model architecture. The training process involves feeding the model with input images and corresponding animation sequences, iteratively updating its parameters to minimize a defined loss function and improve animation quality. Techniques such as transfer learning, data augmentation, and regularization may be employed to enhance model performance and generalization. Throughout the training phase, model convergence and performance are monitored, and adjustments are made as necessary to achieve desired results. Once training is complete, the trained model is saved and ready for deployment within the Image Animation System, where users can interactively generate animations from their input images.

4.4 RESULTS

INPUT:




Choose Image
Image
Upload (0)
or
Peoples
Cartoons
Dolls
Game of Thrones
Statues
Choose Video
Video
Upload (0)
or
All Videos


Settings
Model:
vox
vox
vox-adv
taichi
taichi-adv
nemo
mgif
fashion
bair

OUPUT:


0:08

5. CONCLUSION

5.1 PROJECT CONCLUSION

The Image Animation System represents a significant advancement in the field of computer vision and deep learning, offering a powerful platform for generating animations from static images with remarkable realism and efficiency. Through the integration of cutting-edge technologies such as Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs), coupled with innovative software engineering techniques, the system enables users to unleash their creativity and bring their ideas to life in ways never before possible. The project's user-centric design, intuitive interface, and robust architecture make it accessible to a wide range of users, from artists and content creators to researchers and developers. By automating the laborious process of manual frame-by-frame animation and providing dynamic visual content in real-time, the system significantly enhances productivity and opens up new avenues for content creation and storytelling across various domains. Moving forward, the project holds immense potential for further innovation and expansion, with opportunities for integrating advanced features, enhancing model capabilities, and addressing emerging challenges in the field. As the boundaries of AI and creativity continue to be pushed, projects like the Image Animation System will undoubtedly play a pivotal role in shaping the future of visual media and storytelling.

5.2 FUTURE SCOPE

The Image Animation System exhibits promising avenues for future development and expansion, with several key areas of focus for further enhancement and innovation:

- **Advanced Model Architectures:** Continuously explore and integrate state-of-the-art model architectures and techniques for image animation, such as attention mechanisms, transformer networks, and self-supervised learning approaches. These advancements can lead to improved animation quality, realism, and diversity.
- **Fine-grained Control:** Develop mechanisms to provide users with finer control over animation attributes, such as motion style, speed, and direction. Implementing interactive interfaces or control parameters can empower users to tailor animations to their specific preferences and creative visions.
- **Multi-modal Fusion:** Investigate techniques for integrating additional modalities, such as audio or text descriptions, to enrich the animation generation process. Multi-modal

approaches can enable more contextually aware animations and support novel applications in interactive storytelling and immersive experiences.

- **Domain-specific Applications:** Explore the adaptation of the Image Animation System for domain-specific applications, such as educational content creation, virtual try-on experiences in e-commerce, or personalized avatar generation for social media platforms. Tailoring the system to specific domains can unlock new opportunities for innovation and user engagement.
- **Real-time Performance:** Optimize model inference and animation generation pipelines for real-time performance, enabling interactive and responsive user experiences. Leveraging techniques like model quantization, hardware acceleration, and parallel processing can minimize latency and enhance usability.
- **Collaborative Tools:** Develop collaborative features that enable multiple users to collaboratively create and edit animations in real-time. Such tools can facilitate teamwork among artists, designers, and content creators, fostering creativity and enabling collaborative storytelling projects.
- Ethical Considerations: Address ethical considerations related to the use of AI-generated content, including issues such as privacy, bias, and misuse. Implement safeguards and transparency mechanisms to ensure responsible and ethical use of the Image Animation System in diverse contexts.