First Order Motion Model Report

Brain and Cognitive Science Club

Shruti Sekhar, Siddhant Shekhar

Contents

1 Introduction

Image animation involves generating a video sequence where an object in a source image moves according to the motion of a driving video. Traditional methods rely on landmark-based facial tracking, but here a keypoint-based approach is used for greater flexibility across object categories. The method is based on the framework proposed by Siarohin et al. The model is trained on a dataset of videos without manual annotations or predefined keypoints. It learns a set of keypoints and their local affine transformations in a self-supervised manner. The motion is then transferred from the driving video to the source image using dense motion estimation. Finally, a generator network synthesizes the animated image by combining extracted motion information and appearance details from the source. This approach enables realistic motion transfer across different object categories.

2 Dataset Generation

This repository was used for dataset generation and preprocessing. To train and evaluate the model, a dataset of approximately 7,000 videos was collected and preprocessed. The dataset generation process involved the following steps:

2.1 Dataset Collection

We used the VoxCeleb dataset, which consists of a large collection of talking-head videos. The dataset was obtained from publicly available YouTube videos using an automated script.

2.2 Metadata Processing

A metadata file was used to extract relevant information about each video, including:

- Video ID
- Frame rate (FPS)
- Bounding box coordinates for cropping
- Start and end timestamps
- Partitioning into training and testing sets

2.3 Video Download and Preprocessing

Videos were downloaded using the `youtube-dl` tool and converted into frame sequences. Faces were cropped based on bounding box metadata and frames resized to 256x256 resolution. The pixel values were then normalized and the dataset was split into train/test partitions.

3 Model Architecture

The FOMM pipeline consists of three main components:

3.1 Keypoint Detection

The keypoint detector identifies a set of keypoints in both the source image and driving video. These keypoints serve as motion anchors and guide the animation process.

- Passing the input image through a stacked hourglass network to obtain feature maps.
- Converting feature maps into heatmaps using a convolutional layer.Each heatmap corresponds to one keypoint and represents the probability of a keypoint at each pixel.
- Normalizing the raw heatmap values with a softmax function.
- The most likely keypoint positions are identified using a Gaussian weighted mean.
- The final output is a tensor of shape $(N, K, 2)$, where N is the batch size, K is the number of keypoints, and each keypoint has (x, y) coordinates.
- Along with the keypoints, the network also estimates local affine transformations (scaling, rotation, and translation) around each keypoint, allowing better motion modeling.

3.2 Dense Motion Estimation

Dense motion estimation is used to understand the required motion by computing the difference between the driving keypoints and the source keypoints. The process includes:

- Predicting a dense motion field that describes how each pixel of the source image should move to match the driving motion.
- Generating an occlusion map, which highlights regions of the source image that become occluded due to motion.

3.3 Generator Network

The generator synthesizes the final animation:

- The source image is passed through a convolutional encoder to extract multi-scale feature maps.
- The feature maps from the source image are then warped using the dense motion field, meaning they are adjusted to match the motion inferred from the driving video.
- Occlusion maps are applied to prevent artifacts by ensuring that only visible and relevant features contribute to the reconstruction.
- The warped features are processed by a decoder to generate the final animated image.

4 Training

The training of the proposed first-order motion model was carried out using an adversarial framework that jointly optimizes a generator, a discriminator, and a keypoint detector. The overall training procedure is designed to ensure stable convergence while capturing the complex motion dynamics required by the task.

4.1 Model Initialization and Optimizers

The generator, discriminator, and keypoint detector are initialized and optimized using the Adam optimizer. For each network, distinct learning rates were assigned, with the generator and discriminator learning rates denoted as lr_{gen} and lr_{disc}, respectively, and a separate learning rate lr_{kp} for the keypoint detector.

The optimizers are configured with betas set to $(0.5, 0.999)$, a common choice to balance the momentum terms during adversarial training.

In order to leverage prior training, if a checkpoint was available, the model weights and optimizer states were restored; otherwise, training commenced from scratch.

4.2 Learning Rate Scheduling

A `MultiStepLR` scheduler was employed for each optimizer, with predefined epoch milestones to reduce the learning rates by a factor of 0.1. This step-wise decay helps in fine-tuning the model parameters during later stages of training and contributes to the overall stability of the adversarial training process.

4.3 Data Loading and Augmentation

The training data were fed into the network using a PyTorch `DataLoader`. To enhance the effective training dataset size and ensure robust model performance, the dataset was optionally repeated multiple times using a dataset repeater wrapper. Data batches were shuffled and processed in parallel using multiple workers, ensuring that the model received diverse mini-batches throughout training.

4.4 Adversarial Training Loop

During each epoch, the training loop iterates over the mini-batches of data. The generator is first used to produce synthetic outputs, and its associated loss is computed by averaging over several loss components. The summed loss is then backpropagated through the generator and the keypoint detector, and their parameters are updated accordingly.

If the loss weight corresponding to the adversarial component is non-zero, a similar backpropagation and optimization step is performed for the discriminator using its specific loss function computed over both real and generated samples. This alternating update strategy is crucial to maintain the balance between the generator and the discriminator during training.

4.5 Logging and Checkpointing

Throughout training, a dedicated logger records loss values at each iteration, providing real-time feedback on the model's performance. At the end of each epoch, the current

state of the model parameters and optimizer states is checkpointed. This not only aids in monitoring the training progress but also facilitates potential recovery from any unforeseen interruptions.

5 Results

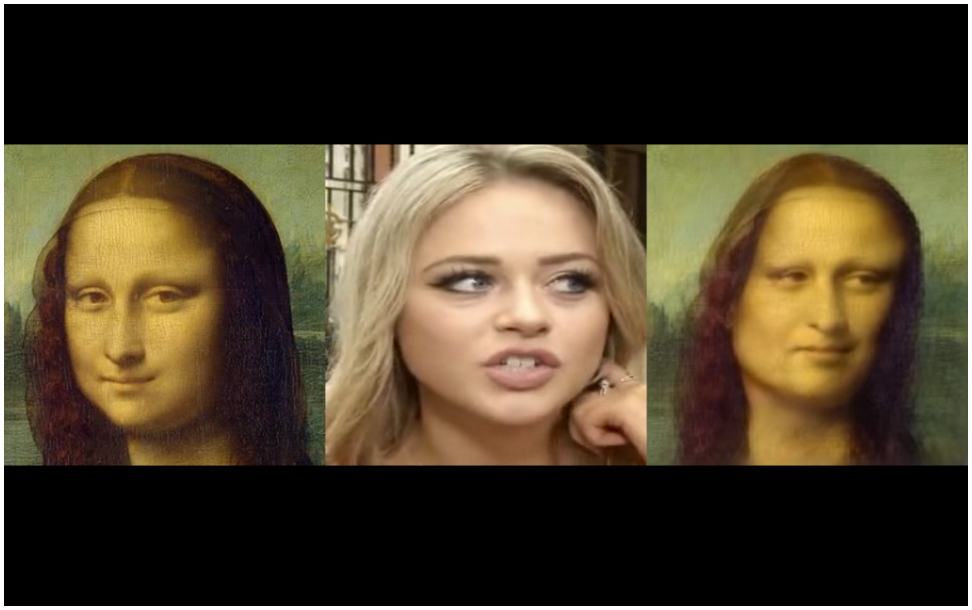

Figure 1: Video 1

Figure 2: Video 2

6 Challenges and Mitigations

During the development and training process, several significant challenges were encountered and addressed as follows:

- **Outdated Repository Requirements** The initial repository provided legacy requirements that were incompatible with the current system environment. Rather than attempting to adapt the outdated dependencies, we build everything from scratch. This approach not only resolved compatibility issues but also allowed for a cleaner, more optimized implementation tailored to the specific needs of the project.
- **Large Dataset Management** The training dataset comprised a vast collection of over 7,000 VoxCeleb videos, which posed significant challenges in terms of storage and processing. To efficiently acquire the dataset, a custom downloading pipeline was implemented using the `yt-dlp` tool. This method streamlined the retrieval process and ensured that the dataset was readily available for subsequent preprocessing and training stages.
- **Computational Constraints** The computational demands of training the first-order motion model were substantial. To address these challenges, training was conducted on the Ola Krutrim. Utilizing this resource not only alleviated the computational bottlenecks but also significantly reduced the training time.