Instructions

The extraction of the videos was done using the scrapper developed by drawrowfly. We performed a match on the audio of the videos in order to detect the exact time each prompt. We also extracted frames and faces from these videos and used similarity metrics in order to detect duplicate faces and group them into individuals. Details of these processes can be found in the paper and in the documentation below.

Prerequisites

We extracted the conda environment .yml file. Make sure to create a conda environment containing Python 3 and install the requirements by running: conda env create --file tiktokfer.yml. Other requirements:

- Install ffmpeg.
- tiktok-scrapper with npm i -g tiktok-scraper. Make sure it worked with tiktok-scraper --help.
- In order to run the face detection model, you'll need a machine with cuDNN. Indeed, we run import torch.backends.cudnn as cudnn.
- Download the weights.zip for the Resnet model this folder and put the two files in a folder called weights at the root.
- You need to have the aws cli set up if you wish to export the results to your s3 bucket (in a venv: pip install --upgrade awscli).

Before running the code, also make sure to add your own SQL information in the variables_and_constants/passwords.py file.

Data Extraction and Processing

If you would like to run the *main.py* file, remember to indicate the face challenge you would like to run it on (*emoji_challenge* or *face_challenge*) by running python main.py face_challenge for example. Also remember to first switch the RUNTYPE constant in the variables_and_constants/constants.py file to test before running with prod. Please also remember to change the BUCKET_NAME to your own AWS S3 bucket in variables_and_constants/constants.py.
The main.py file calls the following substeps:

- main_videos.py first check if the tiktok-scrapper is installed. Then it fetches the videos from the challenge indicated in the run. For the prod version, it fetches 5000 videos and for the test version it only fetches 4 videos. Then it exports the metadata file generated automatically by the tiktok-scrapper to SQL. Finally, it covers the TikTok trademark in the videos before uploading them to the S3 bucket and uploading the videos table to SQL.
- main_subvideos.py first downloads the baseline video for the challenge considered (i.e. the video of the user that launched the challenge) and splits its audio into prompts based on the timestamps specific to that baseline video. Then, it fetches the list of all video that

we have and, for all videos, runs the following process. First, it downloads the video and extracts its audio. Then, using the split_to_prompts.check_subaudio function, it uses convolution to find the location of each original prompt audio. Having extracted the timestamps of each prompt in the video, it uploads the subvideos table to SQL.

- main_frames.py first gets the list of all the videos fetched previously and for each video, runs the following process. First it downloads the video, then it extracts one frame every 0.5 seconds of the video using the extract_frames.extract_frames function. Finally, it uploads each frame to S3 as well as the frames table to SQL.
- main_faces.py first gets the list of all frames extracted previously. Then, for every frame, it checks if a face is present using the extract_faces.extract_faces function that relies on the RetinaFace algorithm with the MobileNet-0.25 backbone and rotates the image if no face is found in case it is in landscape mode. Finally, it uploads the faces to S3 and, if no face is detected, it adds the frame to the nofaces table, and uploads the final faces and nofaces tables to SQL.
- main_individuals.py gets the list of all the faces extracted previously. First, it uses the cluster_faces.intravideo_coordinate_clustering function to use coordinate tracking to run a first version of the clustering and exports it to SQL as faces_with_intra_ind. The coordinate tracking groups faces from the same video together if there is only one face per frame in the entire video and, if there are multiple, it compares the coordinates of the faces between frames and associates each face with the closest to its coordinates in the subsequent frame. This creates a first pass of clustering "intra-video". Then, using embedding2cosineSimilarityMatrix, we generate a similarity matrix using the mean features of each "intra-video" cluster and exports it as similarities in SQL. Finally, using the threshold defined in the constants.py file, it groups "intra-video" clusters together if they are "similar enough" and manually breaks some "intra-video" clusters that we have identified as erroneous. This creates a final clustering corresponding to individuals and is exported to SQL as faces_with_final_ind.
- main_duplicates.py fetches all the faces extracted previously and, using imagededup's perceptual hashing PHash, it removes any duplicated image. It then uploads the list of all the unique images in SQL in unique_faces.

Transfer Learning

Models and performances can be found in the deep learning folder.