Encompass Oculus

Purpose

The purpose of this project is to develop a model for broadcast tv content identification for Project Oculus for client Encompass. The proof-of-principle pipeline contains the code for data preprocessing and tv show identification. The codebase was developed and tested on an AWS P3 instance.

Table of Contents

1. Data Preparation

The user should first clone the code repository, then download 3 zip files from the provided S3 URL and then unzip them to the code repository: - Download the sample tv-shows to be trained (download `s3://encompass-dev-ai-ml-content/sflscientific-encompass/raw.zip`) and unzip to `<path-to-code-repository>/data/raw` - Download the sample tv-shows to be predicted (download `s3://encompass-dev-ai-ml-content/sflscientific-encompass/demo_` and unzip to `<path-to-code-repository>/demo_files` - Download the pre-trained models (download `s3://encompass-dev-ai-ml-content/sflscientific-encompass/vosk-model-en` and unzip to `<path-to-code-repository>/models/vosk-model-en-us-0.22`

Note: The sample tv-shows to be trained is a very small dataset to ensure the user is able to run the code, ideally the user will need much more data to observe good results.

Finally, the user can navigate to the code repository directory via `cd Encompass_Oculus`, and the result code structure will look like the following:

```
+-- LICENSE
+-- README.md                              <- Top-level README
+-- requirements.txt                       <- Python libraries
+-- build.sh                               <- Bash script to build image
+-- run.sh                                 <- Bash script to run container
+-- sync.sh                                <- Bash script to download required files fo
+-- config
|   \-- config.toml                        <- Config file
+-- dockerfiles
|   \-- Dockerfile                         <- Dockerfile
+-- imgs
+-- reference                              <- reference bank, exists only after buildin
|   +-- reference_emb                      <- embeddings for reference frames
|   +-- reference_emb_face                 <- embeddings for reference faces
|   +-- reference_frames                   <- reference frames
|   +-- reference_faces                    <- reference faces
|   +-- reference_name_dict.json           <- actor names for shows
|   \-- reference_prob_dict.json           <- conditional probability for shows
+-- models
|   \-- vosk-model-en-us-0.22              <- pretrained speech to text model weight
+-- src                                    <- Src code folder
|   +-- __init__.py
|   +-- main.py                            <- main script for training/reference buildi
|   +-- train.py                           <- training functions
|   +-- datsets.py                         <- dataset classes
|   +-- losses.py                          <- loss functions
|   +-- preprocess.py                      <- preprocessing tools
|   +-- utils.py                           <- helper functions
|   \-- s3_sync.py                         <- functions to sync files from S3
+-- data
|   +-- raw                                <- raw video for training
|   \-- processed                          <- processed videos for training, exists onl
\--demo_files                              <- demo files (used for demo inference)
```

2. Installation

To set up the environment, first, install Docker version 19.03 or later. The required Linux packages and python packages are specified in Dockerfile (with Python 3.8 in the base image) and requirements.txt, and will be automatically installed when building the docker image.

The user needs to install the required packages to run the code. This is done by navigating to the code directory, building a docker image, and running the code in the docker container. The recommended approach is to execute the `build.sh` script. The bash script is provided in the code repository which will build the

Docker image, including all dependencies.

```
bash build.sh
```

3. Usage

There is a config file (`config/config.toml`) where users are able to define the input, output, and hyperparameters for the model. The user can define the input and output paths of the tv shows in `[DATA]`. For all other parameters in the config file, it is recommended that the user uses the default values and only change the values based on the following training and inference instructions.

Several parameters can be configured in `config/config.toml`.

- `TRAIN_EPOCHS`: Total training epochs
- `BATCH_SIZE`: Batch size for model training, evaluation, and inference.
- `LR`: Learning rate for model training
- `DECAY_STEP`: Steps to decay learning rate.
- `DECAY_RATE`: Decay rate for learning rate.
- `BF_THRES`: Threshold value for skipping black screens
- `NUM_REF`: Number of reference frames to establish from training data.

Training and Building reference 1. The user should set the following fields in the `config/config.toml` file: - TASK = "Train" <– put model in train/build reference model - RETRAIN_BACKBONE = "True" <– allow model to train Siamese model 2. Running the bash script `bash run.sh`, the user will now be located inside the docker container. 3. Running `python src/main.py` will start the training process. A model weight will be generated at `models/SIAMESE_NEG/checkpoint_latest.pt`. Processed frames and videos will be generated under the `processed` folder. 4. The user should set the following fields in the `config/config.toml` file: - TASK = "Train" <– put model in train/build reference model - RETRAIN_BACKBONE = "FALSE" <– avoid Siamese model training - WEIGHTS = "models/SIAMESE_NEG/checkpoint_latest.pt" <– fetch the latest trained weight or user specified weight for reference building 5. Running `python src/main.py` will start the reference building process. A reference folder will be created at `reference`. 6. To exit the application, run `exit` inside the docker container.

Inference 1. The user should set the following fields in the `config/config.toml` file: - TASK = "Inference" <– put model in inference model - EXPECTED_SHOW = "Big Bang Theory"<– the show expected to be on air - WEIGHTS = "models/SIAMESE_NEG/checkpoint_latest.pt" <– specified the trained model weights from the training phase 2. Running the bash script `bash run.sh`, the user will now be located inside the docker container. 3. Running `python src/main.py` will start the inference process. 4. `demo_files/final_results.json` will be the results of the model prediction. 5. `demo_files/explanations` will contain the explanation results of the model prediction. 6. To exit the application, run `exit` inside the docker container.

Note: In the training and building reference section, a very small dataset(`raw.zip`) is used so the user is able to run the code. The user is not expected to see multiple matches in `demo_files/final_results.json`, this is because the sample training set is very small. To see the results for the model being training on a regular-sized dataset, please refer to the Demo section.

4. Demo

To simulate the code in production (Inference only), a pretrained model and a prebuilt reference are provided. 1. The user should download 3 zip files from the provided dropbox link and then unzip them to the repository: - Download the sample tv-shows to be predicted (`s3://encompass-dev-ai-ml-content/sflscientific-encompass/demo_files.zip`) and unzip to `<path-to-repo>/demo_files` - Download the show reference (`s3://encompass-dev-ai-ml-content/sflscientific-encompass/reference.zip`) and unzip to `<path-to-repo>/reference` - Download the pretrained speech-to-text model weights (`s3://encompass-dev-ai-ml-content/sflscientific-encompass/vosk-model-en-us-0`) and unzip to `<path-to-repo>/models/vosk-model-en-us-0.22` - Download the Siamese model weights (`s3://encompass-dev-ai-ml-content/sflscientific-encompass/SIAMESE_NEG.`) and unzip to `<path-to-repo>/models/SIAMESE_NEG` 2. The user can then follow the `Inference` section in `Usage`.

5. Output Format

The example JSON output will be:


```
[
    {
        "clip_name": "demo_files/raw/BBANGC051022-01_1.mxf",
        "match_type": "frame_match",
        "match_ref": "BBANGB051322-01_frame55.jpg",
        "relative_time_stamp": "12.0s",
        "match_certainty": "1.0",
        "current_certainty": "1.0"
    },
    {
        "clip_name": "demo_files/raw/BBANGC051022-01_1.mxf",
        "match_type": "face_match",
        "match_ref": "face_3.jpg",
        "relative_time_stamp": "6.0s",
        "match_certainty": "0.9032258064516129",
        "current_certainty": "1.0"
    },
    ...
]
```

6. Contacts

- Developers
    - Andrew Tseng (atseng@sflscientific.com)
    - Dave McNamee (dmcnamee@sflscientific.com)
- SFL Scientific founders
    - Mike Luk (mluk@sflscientific.com)
    - Mike Segala (msegala@sflscientific.com)
    - Dan Ferrante (danieldf@sflscientific.com)

7. License

To the extent possible under the law, and under our agreements, SFL Scientific retains all copyright and related or neighboring rights to this work.