# How to Build a Text-Video Retrieval Engine

@Towhee.io

6/22/2022

# How to Build a Text-Video Retrieval Engine

## Introduction

duration: 1

This codelab illustrates how to build a text-video retrieval engine from scratch using Milvus and Towhee.

**What is Text-Video Retrieval?**

In simple words, text-video retrieval is: given a text query and a pool of candidate videos, select the video which corresponds to the text query.

**What are Milvus & Towhee?**

- Milvus is the most advanced open-source vector database built for AI applications and supports nearest neighbor embedding search across tens of millions of entries.
- Towhee is a framework that provides ETL for unstructured data using SoTA machine learning models.

We'll go through video retrieval procedures and evaluate the performance. Moreover, we managed to make the core functionality as simple as few lines of code, with which you can start hacking your own video retrieval engine.

## Preparation

duration: 2

### Install Dependencies

First please make sure you have installed required python packages:

```
$ python -m pip install -q pymilvus towhee towhee.models pillow ipython
    gradio
```

### Prepare the data

MSR-VTT (Microsoft Research Video to Text) is a dataset for the open domain video captioning, which consists of 10,000 video clips.

Download the MSR-VTT-1kA test set from google drive and unzip it, which contains just 1k videos. And the video captions text sentence information is in ./MSRVTT_JSFUSION_test.csv.

The data is organized as follows:

- **test_1k_compress:** 1k compressed test videos in MSR-VTT-1kA.
- **MSRVTT_JSFUSION_test.csv:** a csv file containing an ***key,vid_key,video_id,sentence***, for each video and caption text.

First to download the dataset and unzip it:

```
$ curl -L https://github.com/towhee-io/examples/releases/download/data/text_video_search.zip -O
$ unzip -q -o text_video_search.zip
```

Let's take a quick look:

```
import pandas as pd
import os

raw_video_path = './test_1k_compress' # 1k test video path.
test_csv_path = './MSRVTT_JSFUSION_test.csv' # 1k video caption csv.

test_sample_csv_path = './MSRVTT_JSFUSION_test_sample.csv'

sample_num = 1000 # you can change this sample_num to be smaller, so that this notebook will be faster.
test_df = pd.read_csv(test_csv_path)
print('length of all test set is {}'.format(len(test_df)))
sample_df = test_df.sample(sample_num, random_state=42)

sample_df['video_path'] = sample_df.apply(lambda x:os.path.join(raw_video_path, x['video_id']) + '.mp4', axis=1)

sample_df.to_csv(test_sample_csv_path)
print('random sample {} examples'.format(sample_num))

df = pd.read_csv(test_sample_csv_path)

df[['video_id', 'video_path', 'sentence']].head()
```

```
length of all test set is 1000
random sample 1000 examples
```

| | video_id | video_path | sentence |
|---|---|---|---|
| 0 | video7579 | ./test_1k_compress/video7579.mp4 | a girl wearing red top and black trouser is pu... |
| 1 | video7725 | ./test_1k_compress/video7725.mp4 | young people sit around the edges of a room cl... |
| 2 | video9258 | ./test_1k_compress/video9258.mp4 | a person is using a phone |
| 3 | video7365 | ./test_1k_compress/video7365.mp4 | cartoon people are eating at a restaurant |
| 4 | video8068 | ./test_1k_compress/video8068.mp4 | a woman on a couch talks to a a man |

Define some helper function to convert video to gif so that we can have a look at these video-text pairs.

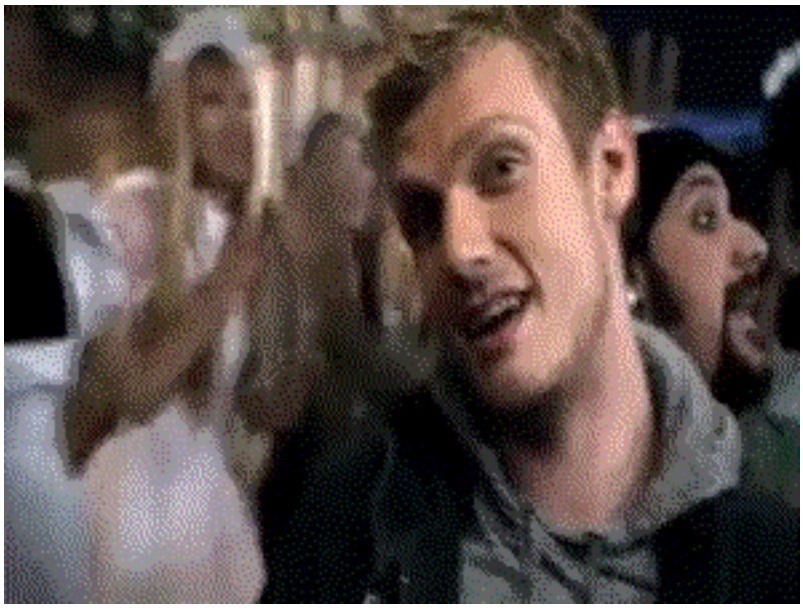
```python
from IPython import display
from pathlib import Path
import towhee
from PIL import Image

def display_gif(video_path_list, text_list):
    html = ''
    for video_path, text in zip(video_path_list, text_list):
        html_line = '<img src="{}"> {} <br/>'.format(video_path, text)
        html += html_line
    return display.HTML(html)


def convert_video2gif(video_path, output_gif_path, num_samples=16):
    frames = (
        towhee.glob(video_path)
              .video_decode.ffmpeg(sample_type='
                 uniform_temporal_subsample', args={'num_samples':
                 num_samples})
              .to_list()[0]
    )
    imgs = [Image.fromarray(frame) for frame in frames]
    imgs[0].save(fp=output_gif_path, format='GIF', append_images=imgs
        [1:], save_all=True, loop=0)


def display_gifs_from_video(video_path_list, text_list, tmpdirname = '
    ./tmp_gifs'):
    Path(tmpdirname).mkdir(exist_ok=True)
    gif_path_list = []
    for video_path in video_path_list:
        video_name = str(Path(video_path).name).split('.')[0]
        gif_path = Path(tmpdirname) / (video_name + '.gif')
        convert_video2gif(video_path, gif_path)
        gif_path_list.append(gif_path)
```

```
    return display_gif(gif_path_list, text_list)
```

Take a look at the ground-truth video-text pairs.

```
sample_show_df = sample_df[:3]
video_path_list = sample_show_df['video_path'].to_list()
text_list = sample_show_df['sentence'].to_list()
tmpdirname = './tmp_gifs'
display_gifs_from_video(video_path_list, text_list, tmpdirname=
    tmpdirname)
```

a girl wearing red top and black trouser is putting a sweater on a dog



young people sit around the edges of a room clapping and raising their arms while others dance in the center during a party

cartoon people are eating at a restaurant

### Create a Milvus Collection

Before getting started, please make sure you have installed milvus. Let's first create a `text_video_retrieval` collection that uses the L2 distance metric and an IVF_FLAT index.

```
from pymilvus import connections, FieldSchema, CollectionSchema,
    DataType, Collection, utility

connections.connect(host='127.0.0.1', port='19530')

def create_milvus_collection(collection_name, dim):
    if utility.has_collection(collection_name):
        utility.drop_collection(collection_name)

    fields = [
    FieldSchema(name='id', dtype=DataType.INT64, descrition='ids',
        is_primary=True, auto_id=False),
    FieldSchema(name='embedding', dtype=DataType.FLOAT_VECTOR,
        descrition='embedding vectors', dim=dim)
    ]
    schema = CollectionSchema(fields=fields, description='video
        retrieval')
    collection = Collection(name=collection_name, schema=schema)

    # create IVF_FLAT index for collection.
```

```
    index_params = {
        'metric_type':'L2', #IP
        'index_type':"IVF_FLAT",
        'params':{"nlist":2048}
    }
    collection.create_index(field_name="embedding", index_params=
        index_params)
    return collection

collection = create_milvus_collection('text_video_retrieval', 512)
```

## Text-Video retrieval

duration: 3

### Load Video Embeddings into Milvus

We first extract embeddings from images with `CLIP4Clip` model and insert the embeddings into Milvus for indexing. Towhee provides a method-chaining style API so that users can assemble a data processing pipeline with operators.

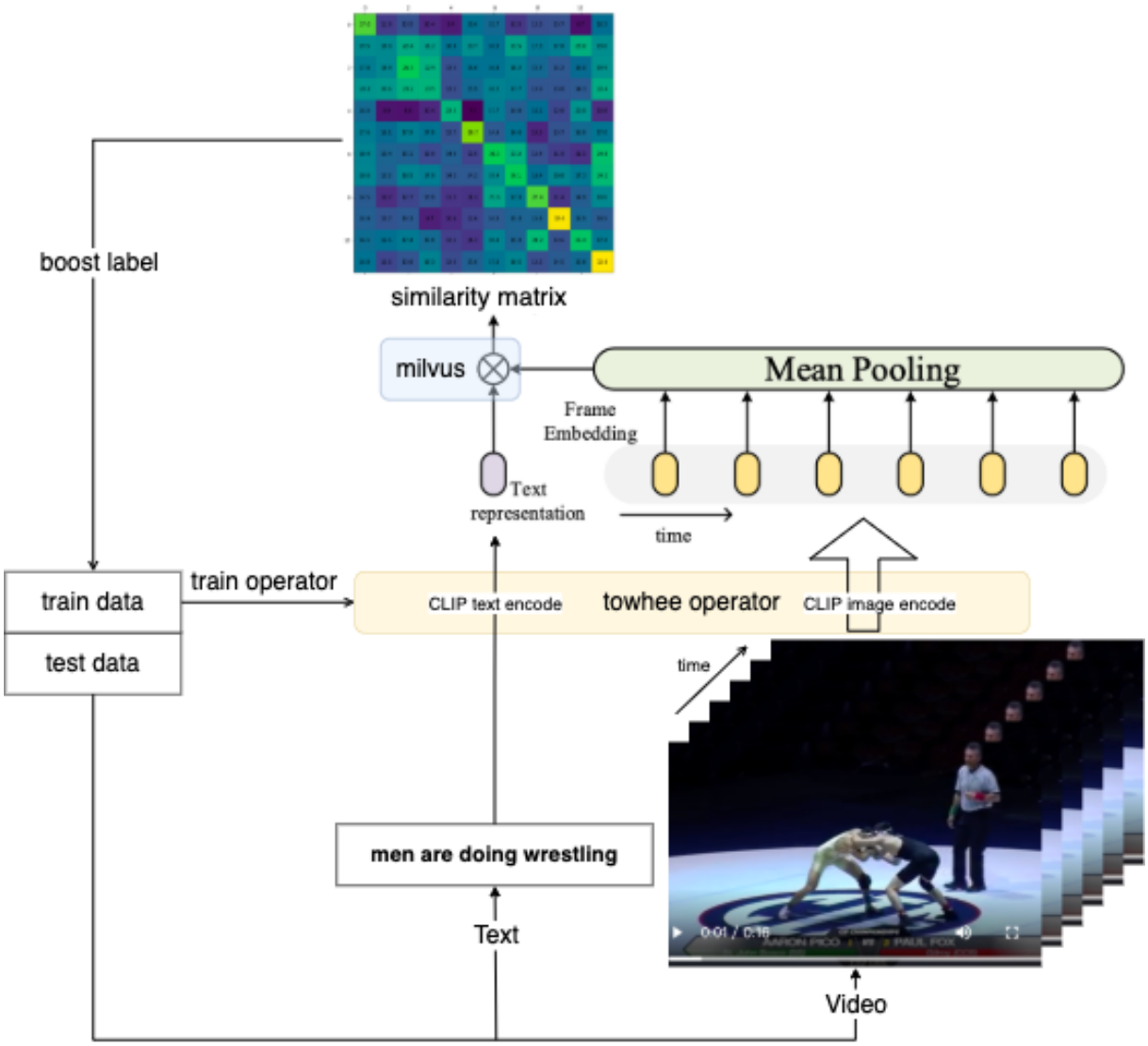
Before you start running the clip4clip operator, you should make sure you have make git and git-lfs installed. For git(If you have installed, just skip it):

```
$ sudo apt-get install git
```

For git-lfs(You must install it for downloading checkpoint weights on backend):

```
$ sudo apt-get install git-lfs
$ git lfs install
```

CLIP4Clip is a video-text retrieval model based on CLIP (ViT-B). The towhee clip4clip operator with pretrained weights can easily extract video embedding and text embedding by a few codes.




```python
import os
import towhee

device = 'cuda:2'
# device = 'cpu'

# For the first time you run this line,
# it will take some time
# because towhee will download operator with weights on backend.
dc = (
    towhee.read_csv(test_sample_csv_path)
      .runas_op['video_id', 'id'](func=lambda x: int(x[-4:]))
      .video_decode.ffmpeg['video_path', 'frames'](sample_type='
         uniform_temporal_subsample', args={'num_samples': 12})
      .runas_op['frames', 'frames'](func=lambda x: [y for y in x])
```

```
        .video_text_embedding.clip4clip['frames', 'vec'](model_name='
            clip_vit_b32', modality='video', device=device)
        .to_milvus['id', 'vec'](collection=collection, batch=30)
)
print('Total number of inserted data is {}.'.format(collection.
    num_entities))
```

Total number of inserted data is 1000.

Here is detailed explanation for each line of the code:

- `towhee.read_csv(test_sample_csv_path)`: read tabular data from csv file;
- `.runas_op['video_id', 'id'](func=lambda x: int(x[-4:]))`: for each row from the data, convert the data type of the column `id` from last 4 number of `video_id`;
- `.video_decode.ffmpeg` and `runas_op`: subsample the video uniformly, and then get a list of images in the video, which are the input of the clip4clip model;
- `.video_text_embedding.clip4clip['frames', 'vec'](model_name='clip_vit_b32', modality='video')`: extract embedding feature from the images subsampled from video, and then mean pool them in the temporal dimension, which repre.
- `.to_milvus['id', 'vec'](collection=collection, batch=30)`: insert video embedding features in to Milvus;

#### Search in Milvus with Towhee

```
dc = (
    towhee.read_csv(test_sample_csv_path).unstream()
      .video_text_embedding.clip4clip['sentence','text_vec'](model_name
          ='clip_vit_b32', modality='text', device=device)
      .milvus_search['text_vec', 'top10_raw_res'](collection=collection
          , limit=10)
      .runas_op['video_id', 'ground_truth'](func=lambda x : [int(x
          [-4:])])
      .runas_op['top10_raw_res', 'top1'](func=lambda res: [x.id for i,
          x in enumerate(res) if i < 1])
      .runas_op['top10_raw_res', 'top5'](func=lambda res: [x.id for i,
          x in enumerate(res) if i < 5])
      .runas_op['top10_raw_res', 'top10'](func=lambda res: [x.id for i,
           x in enumerate(res) if i < 10])
)

dc.select['video_id', 'sentence', 'ground_truth', 'top10_raw_res', '
    top1', 'top5', 'top10']().show()
```

| video_id | sentence | ground_truth | top10_raw_res | top1 | top5 | top10 |
|---|---|---|---|---|---|---|
| video7579 | a girl wearing red top and black... | [7579] len=1 | [{"id": 7579, "score": 1.4152562618255615},{"id": 9969, "score": 1.479762315750122},{"id": 8837, "score": 1.4900273084640503},{"id": 9347, "score": 1.4925360679626465},...] len=10 | [7579] len=1 | [7579,9969,8837,9347,...] len=5 | [7579,9969,8837,9347,...] len=10 |
| video7725 | young people sit around the edge... | [7725] len=1 | [{"id": 7725, "score": 1.3606964349746704},{"id": 8014, "score": 1.490865707397461},{"id": 8339, "score": 1.4913759231567383},{"id": 8442, "score": 1.503324270248413},...] len=10 | [7725] len=1 | [7725,8014,8339,8442,...] len=5 | [7725,8014,8339,8442,...] len=10 |
| video9258 | a person is using a phone | [9258] len=1 | [{"id": 9258, "score": 1.401209831237793},{"id": 9257, "score": 1.4216816425323486},{"id": 9697, "score": 1.4404691457748413},{"id": 7910, "score": 1.4957969188690186},...] len=10 | [9258] len=1 | [9258,9257,9697,7910,...] len=5 | [9258,9257,9697,7910,...] len=10 |
| video7365 | cartoon people are eating at a r... | [7365] len=1 | [{"id": 7365, "score": 1.4048435688018799},{"id": 8781, "score": 1.4607740640640259},{"id": 9537, "score": 1.4721850156784058},{"id": 7831, "score": 1.5039441585540771},...] len=10 | [7365] len=1 | [7365,8781,9537,7831,...] len=5 | [7365,8781,9537,7831,...] len=10 |
| video8068 | a woman on a couch talks to a a ... | [8068] len=1 | [{"id": 7162, "score": 1.4739655256271362},{"id": 8304, "score": 1.4785505533218384},{"id": 8068, "score": 1.4936964511871338},{"id": 7724, "score": 1.495842456817627},...] len=10 | [7162] len=1 | [7162,8304,8068,7724,...] len=5 | [7162,8304,8068,7724,...] len=10 |

## Evaluation

duration: 2

We have finished the core functionality of the text-video retrieval engine. However, we don't know whether it achieves a reasonable performance. We need to evaluate the retrieval engine against the ground truth so that we know if there is any room to improve it.

In this section, we'll evaluate the strength of our text-video retrieval using recall@topk: `Recall@topk` is the proportion of relevant items found in the top-k recommendations. Suppose that we computed recall at 10 and found it is 40% in our top-10 recommendation system. This means that 40% of the total number of the relevant items appear in the top-k results.

```
benchmark = (
    dc.with_metrics(['mean_hit_ratio',])
        .evaluate['ground_truth', 'top1'](name='recall_at_1')
        .evaluate['ground_truth', 'top5'](name='recall_at_5')
        .evaluate['ground_truth', 'top10'](name='recall_at_10')
        .report()
)
```

| | mean_hit_ratio |
|---|---|
| recall_at_1 | 0.421 |
| recall_at_5 | 0.712 |
| recall_at_10 | 0.813 |

This result is almost identical to the recall metrics represented in the paper. You can find more detail about metrics in paperwithcode.

### Release a Showcase

duration: 2

We've learnt how to build a reverse video search engine. Now it's time to add some interface and release a showcase. Towhee provides `towhee.api()` to wrap the data processing pipeline as a function with `.as_function()`. So we can build a quick demo with this `milvus_search_function` with Gradio.

```
import gradio

show_num = 3
with towhee.api() as api:
    milvus_search_function = (
         api.clip4clip(model_name='clip_vit_b32', modality='text',
            device=device)
            .milvus_search(collection=collection, limit=show_num)
            .runas_op(func=lambda res: [os.path.join(raw_video_path, '
               video' + str(x.id) + '.mp4') for x in res])
            .as_function()
    )

interface = gradio.Interface(milvus_search_function,
                             inputs=[gradio.Textbox()],
                             outputs=[gradio.Video(format='mp4') for _
                                in range(show_num)]
                            )

interface.launch(inline=True, share=True)
```