Build a Video Classification System in 5 Lines

@Milvus.io

6/13/2022

Build a Video Classification System in 5 Lines

Introduction

Video tagging means adding proper tags for videos. Tags can be various from different aspects. For example, object detection, action recognition, place identification can all contribute to video tagging.

This codelab will show you how to build a basic video classification system with sample data (of 20 human activities), visualize predicted labels, and measure the system with performance metrics. Moreover, you can try optimization methods for accuracy and efficiency. At the end, you are able to build up a playable video classification system with 5 lines of code.

Preparation

Install Dependencies

First make sure you have installed required python packages:

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

Prepare the data

This tutorial will use a small data extracted from validation data of Kinetics400. You can download the subset from Github. This tutorial will just use 200 videos under `train` as example.

The data is organized as follows:

- **train:** 20 classes, 10 videos per class (200 in total)
- **reverse_video_search.csv:** a csv file containing an ***id****, **path***, and ***label*** for each video in train directory

First to download the dataset and unzip it:

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

Let’s take a quick look:

```
import pandas as pd

df = pd.read_csv('./reverse_video_search.csv')
print(df.head(3))
print(df.label.value_counts())
```

```
   id                                         path                label
0   0  ./train/country_line_dancing/bTbC3w_NIvM.mp4  country_line_dancing
1   1  ./train/country_line_dancing/n2dWtEmNn5c.mp4  country_line_dancing
2   2  ./train/country_line_dancing/zta-Iv-xK7I.mp4  country_line_dancing
country_line_dancing     10
pumping_fist             10
playing_trombone         10
shuffling_cards          10
tap_dancing              10
clay_pottery_making      10
eating_hotdog            10
eating_carrots           10
juggling_soccer_ball     10
juggling_fire            10
javelin_throw            10
dunking_basketball       10
chopping_wood            10
trimming_trees           10
using_segway             10
pushing_cart             10
dancing_gangnam_style    10
riding_mule              10
drop_kicking             10
doing_aerobics           10
Name: label, dtype: int64
```

For later steps to easier get videos & measure results, we build some helpful functions in advance:

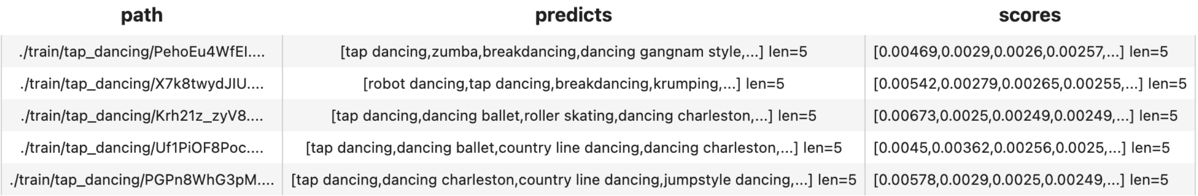
- **ground_truth:** get ground-truth label for the video by its path

```
def ground_truth(path):
    label = df.set_index('path').at[path, 'label']
    return [label.replace('_', ' ')]import cv2
from towhee._types.image import Image

id_img = df.set_index('id')['path'].to_dict()
def read_images(results):
    imgs = []
    for re in results:
        path = id_img[re.id]
        imgs.append(Image(cv2.imread(path), 'BGR'))
    return imgs
```

Predict labels

Let's take some 'tap_dancing' videos as example to see how to predict labels for videos within 5 lines. By default, the system will predict top 5 labels sorting by scores (of possibility) from high to low. You can control the number of labels returnbed by change `topk`. Please note that the first time run will take some time to download model.

```
import towhee

(
    towhee.glob['path']('./train/tap_dancing/*.mp4')
          .video_decode.ffmpeg['path', 'frames'](sample_type='uniform_temporal_subsample', args={'num_samples': 16})
          .action_classification['frames', ('predicts', 'scores', 'features')].pytorchvideo(
             model_name='x3d_m', skip_preprocess=True, topk=5)
          .select['path', 'predicts', 'scores']()
          .show()
)
```



path	predicts	scores
./train/tap_dancing/PehoEu4WfEl....	[tap dancing,zumba,breakdancing,dancing gangnam style,...] len=5	[0.00469,0.0029,0.0026,0.00257,...] len=5
./train/tap_dancing/X7k8twydJIU....	[robot dancing,tap dancing,breakdancing,krumping,...] len=5	[0.00542,0.00279,0.00265,0.00255,...] len=5
./train/tap_dancing/Krh21z_zyV8....	[tap dancing,dancing ballet,roller skating,dancing charleston,...] len=5	[0.00673,0.0025,0.00249,0.00249,...] len=5
./train/tap_dancing/Uf1PiOF8Poc....	[tap dancing,dancing ballet,country line dancing,dancing charleston,...] len=5	[0.0045,0.00362,0.00256,0.0025,...] len=5
./train/tap_dancing/PGPn8WhG3pM....	[tap dancing,dancing charleston,country line dancing,jumpstyle dancing,...] len=5	[0.00578,0.0029,0.0025,0.00249,...] len=5

Here are some details for each line of the assemble pipeline:

- `towhee.read_csv()`: read tabular data from csv file
- `.video_decode.ffmpeg()`: an embeded Towhee operator reading video as frames with specified sample method and number of samples. learn more
- `.action_classification.pytorchvideo()`: an embeded Towhee operator applying specified model to video frames, which can be used to predict labels and extract features for video. learn more

Evaluation

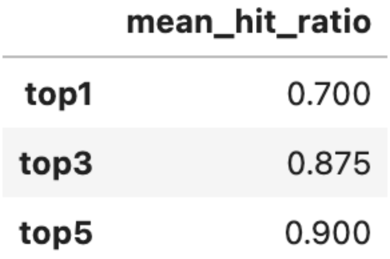
We have just showed how to classify video, but how's its performance? Towhee has provided different options for metrics to evaluate predicted results against ground truths.

In this section, we'll measure the performance with the average metric value:

- mHR (recall@K):

- Mean Hit Ratio describes how many actual relevant results are returned out of all ground truths.
- Since we predict top K labels while only 1 ground truth for each entity, the mean hit ratio is equivalent to recall@topk.

```
import time

start = time.time()
dc = (
    towhee.read_csv('reverse_video_search.csv').unstream()
          .video_decode.ffmpeg['path', 'frames'](sample_type='uniform_temporal_subsample', args={'num_samples': 16})
          .action_classification['frames', ('predicts', 'scores', 'features')].pytorchvideo(
              model_name='x3d_m', skip_preprocess=True, topk=5)
)
end = time.time()
print(f'Total time: {end-start}')

benchmark = (
    dc.runas_op['path', 'ground_truth'](func=ground_truth)
      .runas_op['predicts', 'top1'](func=lambda x: x[:1])
      .runas_op['predicts', 'top3'](func=lambda x: x[:3])
      .with_metrics(['mean_hit_ratio'])
      .evaluate['ground_truth', 'top1'](name='top1')
      .evaluate['ground_truth', 'top3'](name='top3')
      .evaluate['ground_truth', 'predicts'](name='top5')
      .report()
)
```



	mean_hit_ratio
top1	0.700
top3	0.875
top5	0.900

Optimization

You're always encouraged to play around with the tutorial. We present some optimization options here to make improvements in accuracy, latency, and resource usage. With these methods, you can make the classification system better in performance and more feasible in production.

Change model

There are more video models using different networks. Normally a more complicated or larger model will show better results while cost more. You can always try more models to tradeoff among accuracy, latency, and resource usage. Here I show the performance of video classification using a SOTA model with multiscale vision transformer as backbone.

```
benchmark = (
    towhee.read_csv('reverse_video_search.csv').unstream()
          .video_decode.ffmpeg['path', 'frames'](sample_type='uniform_temporal_subsample', args={'num_samples': 32})
          .action_classification['frames', ('predicts', 'scores', 'features')].pytorchvideo(
              model_name='mvit_base_32x3', skip_preprocess=True, topk=5)
          .runas_op['path', 'ground_truth'](func=ground_truth)
          .runas_op['predicts', 'top1'](func=lambda x: x[:1])
          .runas_op['predicts', 'top3'](func=lambda x: x[:3])
          .with_metrics(['mean_hit_ratio'])
          .evaluate['ground_truth', 'top1'](name='top1')
          .evaluate['ground_truth', 'top3'](name='top3')
          .evaluate['ground_truth', 'predicts'](name='top5')
          .report()
)
```

	mean_hit_ratio
top1	0.745
top3	0.900
top5	0.920

Deploy with parallel and exception safe

Parallel Execution

We are able to enable parallel execution by simply calling set_parallel within the pipeline. It tells Towhee to process the data in parallel. The code below enables parallel execution on the above example. It shows that it finishes the classification of 200 videos within 2 seconds with 5 parallel executions.

```
dc = (
    towhee.read_csv('reverse_video_search.csv')
          .set_parallel(5)
```

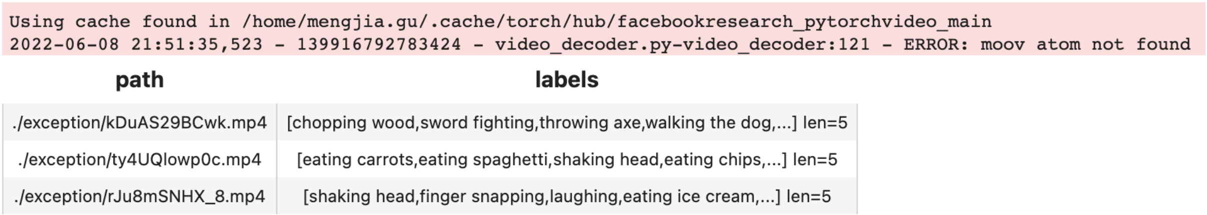
```
            .video_decode.ffmpeg['path', 'frames'](sample_type='uniform_temporal_subsample', args={'num_samples': 16})
            .action_classification['frames', ('predicts', 'scores', 'features')].pytorchvideo(
                model_name='x3d_m', skip_preprocess=True, topk=5)
)
```

Exception Safe

When we have large-scale data, there may be some bad data that will cause errors. Typically, the users don't want such errors to break the system in production. Therefore, the pipeline should continue to process the rest of the videos and report broken ones.

Towhee supports an `exception-safe` execution mode that allows the pipeline to continue on exceptions and represent the exceptions with Empty values. The user can choose how to deal with the empty values at the end of the pipeline. During the query below, there are 4 files in total under the exception folder, one of them is broken. With `exception-safe`, it will print the ERROR but NOT terminate the process. As you can see from results, `drop_empty` deletes empty data.

```
(
    towhee.glob['path']('./exception/*')
            .exception_safe()
            .video_decode.ffmpeg['path', 'frames'](sample_type='uniform_temporal_subsample', args={'num_samples': 16})
            .action_classification['frames', ('labels', 'scores', 'vec')].pytorchvideo(
                model_name='x3d_m', skip_preprocess=True)
            .drop_empty()
            .select['path', 'labels']()
            .show()
)
```



```
Using cache found in /home/mengjia.gu/.cache/torch/hub/facebookresearch_pytorchvideo_main
2022-06-08 21:51:35,523 - 139916792783424 - video_decoder.py-video_decoder:121 - ERROR: moov atom not found
```

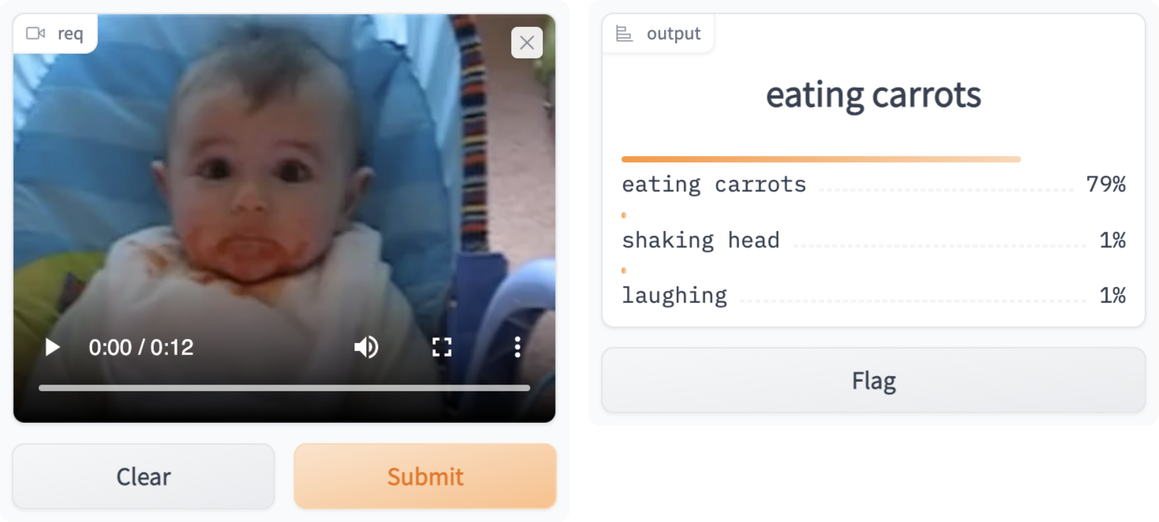
path	labels
./exception/kDuAS29BCwk.mp4	[chopping wood,sword fighting,throwing axe,walking the dog,...] len=5
./exception/ty4UQlowp0c.mp4	[eating carrots,eating spaghetti,shaking head,eating chips,...] len=5
./exception/rJu8mSNHX_8.mp4	[shaking head,finger snapping,laughing,eating ice cream,...] len=5

Release a Showcase

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 `action_classification_function` with Gradio.

```python
import gradio

topk = 3
with towhee.api() as api:
    action_classification_function = (
         api.video_decode.ffmpeg(
                sample_type='uniform_temporal_subsample', args={'num_samples': 32})
            .action_classification.pytorchvideo(model_name='mvit_base_32x3', skip_preprocess=True, topk=topk)
            .runas_op(func=lambda res: {res[0][i]: res[1][i] for i in range(len(res[0]))})
            .as_function()
    )


interface = gradio.Interface(action_classification_function,
                             inputs=gradio.Video(source='upload'),
                             outputs=[gradio.Label(num_top_classes=topk)]
                            )


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



eating carrots

```
eating carrots    79%
shaking head       1%
laughing           1%
```