AutoVideo: An Automated Video Action Recognition System

Daochen Zha1*, Zaid Pervaiz Bhat2*, Yi-Wei Chen2*, Yicheng Wang2*, Sirui Ding2*, Jiaben Chen3*, Kwei-Herng Lai1*, Mohammad Qazim Bhat1*, Anmoll Kumar Jain2, Alfredo Costilla Reyes1, Na Zou1 and Xia Hu1

1 Department of Computer Science, Rice University
2 Department of Computer Science and Engineering, Texas A&M University
3 School of Information Science and Technology, ShanghaiTech University
4 Department of Engineering Technology and Industrial Distribution, Texas A&M University

{dazh@rice.edu, chenjb1@shanghaitech.edu.cn
{zaid.bhat134, yiwei.chen, wangyc, siruiding, jain.amollkumar, nzou1} @tamu.edu

Abstract

Action recognition is an important task for video understanding with broad applications. However, developing an effective action recognition solution often requires extensive engineering efforts in building and testing different combinations of the modules and their hyperparameters. In this demo, we present AutoVideo, a Python system for automated video action recognition. AutoVideo is featured for 1) highly modular and extendable infrastructure following the standard pipeline language, 2) an exhaustive list of primitives for pipeline construction, 3) data-driven tuners to save the efforts of pipeline tuning, and 4) easy-to-use Graphical User Interface (GUI). AutoVideo is released under MIT license at https://github.com/datamllab/autovideo.

1 Introduction

Action recognition is one of the most important tasks in video understanding [Herath et al., 2017]. Given a video clip, it aims to identify the human actions in the video, such as brush hair, cartwheel, catch, chew, clap, climb, etc. It has broad applications, such as security [Meng et al., 2007], healthcare [Gao et al., 2018] and behavior analysis [Poppe, 2010].

Deep learning has achieved promising performance in action recognition [Tran et al., 2015; Wang et al., 2016; Carreira and Zisserman, 2017; Zolfaghari et al., 2018; Hou et al., 2019]. However, developing a deep learning solution heavily relies on human efforts. First, we often need a very complex training pipeline, including but not limited to data loading, frame extraction, video cropping/scaling, video augmentation, model training, etc., which requires huge engineering efforts. Second, to achieve a good performance, a practitioner often needs extensive laborious trials on different combinations of the modules and their hyperparameters.

To bridge this gap, in this demo, we present an automated video action recognition system named AutoVideo, which has several desirable features. First, it is highly modular and extendable. It is designed using the standard pipeline language under D3M infrastructure [Milutinovic et al., 2020; Lai et al., 2021a], which defines primitives as basic building blocks and uses Directed Acyclic Graph (DAG) to describe pipelines. The inputs, the outputs, and the hyperparameters of each module are well formatted so that we can easily develop and integrate new modules into the system with minimum engineering costs. Second, it supports an exhaustive list of primitives for pipeline construction. Specifically, we have so far supported 188 primitives from video processing to recognition models. Users can create various training pipelines via combining these primitives in different ways. Third, it provides data-driven tuners to save the efforts of pipeline tuning. We have implemented two commonly used AutoML tuners, including random search and Hyperopt [Bergstra et al., 2013]. They can automatically explore different primitive combinations and tune the hyperparameters to identify the best pipeline design. Fourth, it provides a Graphical User Interface (GUI). In the GUI, users can manually construct pipelines in a drag-and-drop fashion, launch the training or search, and monitor the training progress.
Table 1: The number of primitives implemented in each module of AutoVideo with some example primitives.

<table>
<thead>
<tr>
<th>Module</th>
<th>Number</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Processing</td>
<td>5</td>
<td>DatasetToDataFrame</td>
</tr>
<tr>
<td>Video Processing</td>
<td>1</td>
<td>FrameExtraction</td>
</tr>
<tr>
<td>Video Transformation</td>
<td>3</td>
<td>GroupScale, GroupCenterCrop</td>
</tr>
<tr>
<td>Video Augmentation</td>
<td>170</td>
<td>AdditiveGaussianNoise, Rotate</td>
</tr>
<tr>
<td>Action Recognition</td>
<td>9</td>
<td>TSN, TSM, C3D, ECO</td>
</tr>
<tr>
<td>Total</td>
<td>188</td>
<td></td>
</tr>
</tbody>
</table>

While there are some other open-sourced video understanding libraries out there [Yue Zhao, 2019; Team, 2021], they only provide a model zoo for users. In contrast, we provide an exhaustive list of primitives to allow users to create various pipelines or search with AutoML tuners. Unlike some other research-oriented AutoML studies for videos [Piergiovanni et al., 2019; Ryoo et al., 2019], we target an easy-to-use and open-sourced package with unified interfaces for users. Thus, AutoVideo complements the existing efforts.

2 AutoVideo System

Figure 1 shows an overview of the AutoVideo system. It implements a complete training pipeline consisting of data processing, video processing, video transformation, video augmentation, and action recognition, where each of the pipeline steps can be instantiated with customized choices of primitives, which leads to a large number of possible pipelines. Then two automated tuners are provided to automatically discover the best pipelines based on the performance. This section first introduces the standard pipeline language and then elaborates on our programming interface and GUI.

2.1 Primitives and Pipelines

We build AutoVideo upon D3M infrastructure [Milutinovic et al., 2020; Lai et al., 2021a], which provides generic and extendable descriptions for modules and pipelines. The interface can accommodate various modules. Here, we provide an overview of the basic concepts. More details of the pipeline language can be found in [Milutinovic et al., 2020].

Primitive is the basic build block. It is an implementation of a function with some hyperparameters. A pipeline is a Directed Acyclic Graph (DAG) consisting of several primitive steps. Data types, such as DataFrame and NumPy Ndarrays, can be passed between steps in a pipeline. Each of the above concepts is associated with metadata to describe parameters, hyper-parameters, etc. Following this pipeline language, we wrap each component in AutoVideo (e.g., an action recognition algorithm or an augmentation module) as a primitive with some associated hyperparameters. We have so far supported 188 primitives, which are summarized in Table 1, where data processing is designed for reading videos and the labels, video processing converts videos into frames, video transformation transforms the videos into the target size through cropping or scaling, video augmentation augments the videos in training, and action recognition trains deep learning models. Various pipelines can be constructed by combining these primitives in different ways. The genericity of the pipeline language allows AutoVideo to be easily extended to support other video-related tasks in the future.

2.2 Programming Interface

From the programming view, AutoVideo enables users to easily train any manually designed pipelines and also use automated tuners to help discover the best pipelines. A minimum example of fitting a pipeline is given below.

Example 1: Code snippet of automated tuning.

```
search_space = {
    "augmentation": ["aug_0", "aug_1"],
    "multi_aug": ["meta_Sometimes", "meta_Sequentially"],
    "algorithm": ["tsn"],
    "learning_rate": tune.uniform(0.0001, 0.001),
    "momentum": tune.uniform(0.9, 0.99),
    "weight_decay": tune.uniform(0, 0.001),
    "num_segments": tune.choice([8, 16, 32]),
}

config = {
    "searching_algorithms": ["hyperopt"],
    "num_samples": 100,
}

searcher = RaySearcher(train_dataset, train_media_dir, target_index=target_index)  
best_config = searcher.search(search_space=search_space, config=config)
```

Example 2: Code snippet of automated tuning.

```
from autovideo import fit, build_pipeline
pipeline = build_pipeline(
    "transformation": [("Scale", {'scale_size': (128,128)})],
    "augmentation": [("arithmetic_AdditiveGaussianNoise", "arithmetic_AdditiveLaplaceNoise"),
                     "geometric_Shear", "geometric_Jigsaw"],
    "multi_aug": ["meta_Sometimes", "meta_Sequentially"],
    "algorithm": ["tsn"],
    "learning_rate": tune.uniform(0.0001, 0.001),
    "momentum": tune.uniform(0.9, 0.99),
    "weight_decay": tune.uniform(0, 0.001),
    "num_segments": tune.choice([8, 16, 32]),
)
```

Example 3: Code snippet of automated tuning.

```
from ray import tune
from autovideo.searcher import RaySearcher
search_space = {
    "augmentation": ["aug_0", "aug_1"],
    "multi_aug": ["meta_Sometimes", "meta_Sequentially"],
    "algorithm": ["tsn"],
    "learning_rate": tune.uniform(0.0001, 0.001),
    "momentum": tune.uniform(0.9, 0.99),
    "weight_decay": tune.uniform(0, 0.001),
    "num_segments": tune.choice([8, 16, 32]),
}

config = {
    "searching_algorithms": ["hyperopt"],
    "num_samples": 100,
}

searcher = RaySearcher(train_dataset, train_media_dir, valid_dataset=valid_dataset, valid_media_dir=valid_media_dir)  
best_config = searcher.search(search_space=search_space, config=config)
```
Here, valid_dataset is the DataFrame associated with the validation data. search_space is a dictionary specifying the augmentation methods, recognition models, and the ranges of hyperparameters the tuner will cover. We follow the search space definitions in Ray [Moritz et al., 2018], which allows the users to flexibly design the search space. config specifies some tuning configurations, such as the type of the tuner and the search budget. The automated tuner will search for the best pipeline designs within the search space and return the best-discovered pipeline configuration in best_config, which can be then used to re-fit the pipeline.

2.3 GUI

Figure 2 presents our GUI to help users construct pipelines, fit the constructed pipelines, and make predictions. The GUI is implemented based on Orange [Demšar et al., 2004].

The left-hand side of Figure 2 illustrates the canvas for pipeline building. Users can construct a pipeline in a drag-and-drop fashion by dragging icons (primitives in the left) to the canvas and connecting them with lines. For each of the primitives, we can double-click the icon to modify the default hyperparameters of the primitive. Once a pipeline is built, the GUI will convert the constructed pipeline into the configuration dictionary as in Example 1 and the backend will instantiate the pipeline. In addition, users can save a pipeline and load it later just like editing a document in Microsoft Word.

The middle and right-hand sides of Figure 2 show the pop-up windows for pipeline fitting and training progress monitoring, respectively. Specifically, the users can fit the pipeline, make predictions with a fitted pipeline, or perform automated searching by simply clicking a button in the control panel. For example, by clicking the “fit” button, the backend will train the model and visualize the training curve once the training is finished. By clicking the “produce” button, the backend will make predictions on the testing data. The “search” button will launch the automated tuners for pipeline search and save the best pipeline. The prediction results will be displayed at the top of the window, where the users can watch each of the testing videos and check its predictions and labels. In addition, we also provide an interactive console, where the users can interact with the backend with command lines. This design provides flexibility to meet different needs of the users.

3 Effectiveness of Pipeline Search

We conduct a preliminary experiment to showcase the effectiveness of pipeline search. We define a search space that allows choosing 3 augmentation primitives from the categories of arithmetic, geometric, and color, and tuning the hyperparameters of the TSN model with learning rate [0.001, 0.0001], momentum [0.9, 0.99], weight decay [5e-4, 1e-3], number of segments 8, 16, 32. We separate 5% of the training data for validation purposes. We apply the random search and Hyperopt tuners to search the pipeline configurations. We run the search with 50 samples and evaluate the best configuration discovered in the search.

In Table 2, we compare the two tuners as well as the default pipeline configuration on HMDB-51, a task that aims to identify 51 human actions, and a subset of it with the first six actions named HMDB-6. Both the tuners outperform the default pipeline by a large margin, which verifies the effectiveness of pipeline search and hyperparameter tuning. We defer a more comprehensive evaluation to our future work since it is out of the scope of our demo presentation.

4 Conclusions and Future Work

We present AutoVideo, an automated video action recognition system. It provides a highly modular implementation of 188 primitives, on which users can flexibly create pipelines. It also supports automated tuners and an easy-to-use GUI to help researchers/practitioners develop prototypes. We will actively maintain AutoVideo and develop more features based on our previous research. We plan to wrap more primitives beyond action recognition from our research codes in object detection [Wang et al., 2022] and outlier detection [Zha et al., 2020; Li et al., 2020; Lai et al., 2021b], and tuners with reinforcement learning [Zha et al., 2021c; Zha et al., 2021a; Zha et al., 2019; Zha et al., 2021b].

---

1https://serre-lab.clps.brown.edu/resource/hmdb-a-large-human-motion-database/
References


