Final Project: Action Recognition with Deep Learning

ECSE 4965/6965 Deep Learning

1 Overview

In the final project, we are going to apply deep learning techniques to recognize human actions, which has a variety of applications such as human-computer interaction, surveillance, health care, entertainment, etc. There are different modalities of data that can be used for action recognition. Commonly used modalities include images and videos captured by color or depth camera. Other time series data such as skeleton pose and kinematic measurements are also used. For this project, we use color videos. The goal is to determine the category of a video from a pre-defined list of actions. In other words, we treat action recognition as a classification task. The following instructions will provide you the information about the data and choice of deep models, followed by evaluation criteria and submission instructions.

2 Dataset

Figure 1: Example images from UCF11 dataset [5].

The dataset we are going to use is UCF YouTube Action Dataset (UCF11) developed by UCF CRCV [1]. The dataset contains 11 action categories: *basketball shooting,*

[1] http://crcv.ucf.edu/data/UCF_YouTube_Action.php

biking/cycling, diving, golf swinging, horse back riding, soccer juggling, swinging, tennis swinging, trampoline jumping, volleyball spiking, and *walking with a dog*. Each action category contains about 100 video with a total number of 1168 videos for the whole dataset. The dataset contains large variations in background, viewpoint, camera motion, subject appearance, illumination conditions, etc. The resolution of the videos is 320x240 and the length varies.

2.1 Pre-processing

The pre-processing contains 3 steps. We will performed the first two steps for you. You **only** need to perform the third step so that the storage consuming part is done locally. The pre-processed dataset can be downloaded here [2].

- Resize the video resolution from 320x240 to 64x64
- Divide each video into clips of length 30 frames using sliding window
- Normalize the pixel between 0 and 1 by dividing 255 (you may need to convert pixel value from uint to float before division)

Here is the sample code to load the data we provided. You can split a subset from training data for validation and tuning hyperparameters. We will withhold a subset as testing data, which is used for final evaluation and is not released to students.

```
import pickle
# load the dataset into memory
data_file = open('youtube_action_train_data.pkl', 'rb')
train_data, train_labels = pickle.load(data_file)
data_file.close()
```

3 Models

In the final project, you should use deep model to perform the task. But the specific choice and architecture is not restricted. Some popular architectures include C3D [1], Two-stream CNN [2], LRCN [3], HRNN [4]. Since we are working with video data, which by nature contains temporal dependencies, we do recommend using models that explicitly model the temporal dependencies such as LSTM. You can build your own model or modify existing models. The training process must be performed by yourself and any use of existing models must be clearly cited. Whichever model you develop, it should take the input as sequences of images (same format as provided training data) and output an array of prediction labels (same format as provided training labels).

[2] https://drive.google.com/drive/u/1/folders/17Ul1bps7ONxQ3Ktt_QlJHnNWfl1tqHSq

3.1 Evaluation

To evaluate model performance, we consider two sets of criteria. The first set is class-wise accuracy and overall accuracy. The accuracy is defined as

$$\text{accuracy} = \frac{\text{number of correctly predicted videos}}{\text{number of videos to be predicted}} \tag{1}$$

Apply Eq. (1) to data of each individual class to get class-wise accuracy. Apply Eq. (1) to the entire validation dataset to get overall accuracy. You should also report confusion matrix as a way to visualize the performance of the model. An example is shown in Figure 2. The cross-entropy loss is often used to train classification model. We

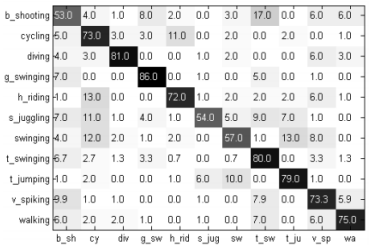

Figure 2: Confusion matrix of UCF11 dataset reported in [5]

recommend using it as a starting point. Feel free to use more sophisticated loss functions as you feel more confident. The second criterion is speed in terms of number of recognition frames per second. We will evaluate your model on the our machine measuring both overall accuracy the speed. The final performance is determined with the combination of accuracy and speed. The specific evaluation process will be released later.

3.2 Fair comparison

We do have considered the case for students with less computation powers, which restrict them from using a complex model and getting better accuracy. For fair comparison, we use small scale dataset like UCF11, to penalize complex models which leads to over-fitting. Secondly, the grading rubric makes sure simple models (certain prediction error) can get at least 70% model evaluation credits.

4 Submission

The following materials should be submitted to TA.

- Your source code
- Your model (see next section for details)
- Report

- Model architecture description
- Explanation and justification of your model design
- Loss and overall accuracy plots versus training iterations
- Final class-wise accuracy and confusion matrix

4.1 Submit your model

We will still use Tensorflow Graph Collection to store some operations in your model, these nodes should be stored in a collection called "validation-nodes", the following code may help you

```
x = tf.placeholder(shape=[batch_size, seq_len, 64, 64, 3],
    dtype=tf.float32)
y = tf.placeholder(shape=[batch_size], dtype=tf.int64)
predict_op = ...
# Save the model
tf.get_collection('validation_nodes')

# Add opts to the collection
tf.add_to_collection('validation_nodes', x)
tf.add_to_collection('validation_nodes', y)
tf.add_to_collection('validation_nodes', predict_op)
```

"x" is the placeholder for your input video sequences, "y" is the placeholder for the labels. "predict-op" is your prediction operation. This operation should return a vector of integers, where each integer is the predicted labels(value between 0-10), i.e. apply tf.argmax for your output.

When exporting your model, you still use the built-in Saver(). You can refer to the following sample code,

```
# start training
saver = tf.train.Saver()
with tf.Session() as sess:
   sess.run(tf.global_variables_initializer())
   ...
   for i in np.arange(max_iteration):
     # training your model
     ...
     sess.run(your_train_step, feed_dict={x: batch_x, y:
         batch_y})
   ...
   # this saver.save() should be within the same tf.Session()
       after the training is done
   save_path = saver.save(sess, "./my_model")
```

References

[1] Ji, Shuiwang, Wei Xu, Ming Yang, and Kai Yu. "3D convolutional neural networks for human action recognition." IEEE transactions on pattern analysis and machine intelligence 35, no. 1 (2013): 221-231.
[2] Simonyan, Karen, and Andrew Zisserman. "Two-stream convolutional networks for action recognition in videos." In Advances in neural information processing systems, pp. 568-576. 2014.
[3] Donahue, Jeffrey, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. "Long-term recurrent convolutional networks for visual recognition and description." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 2625-2634. 2015.
[4] Du, Yong, Wei Wang, and Liang Wang. "Hierarchical recurrent neural network for skeleton based action recognition." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 1110-1118. 2015.
[5] Liu, Jingen, Jiebo Luo, and Mubarak Shah. "Recognizing realistic actions from videos in the wild." In Computer vision and pattern recognition, 2009. CVPR 2009. IEEE conference on, pp. 1996-2003. IEEE, 2009.