Final Project: Action Recognition with Deep Learning

Xiao Jin
Naiyu Yin

ECSE 4850/6850
Due 11:59 PM, May 12nd

1 Project Task Overview

For the final project of this course, you are required to train a deep neural network model and perform a human action recognition task. Human action recognition has a variety of applications in human-computer interaction(HCI), surveillance, healthcare, entertainment areas. There are different modalities of data that are used for action recognition. Commonly used modalities include images and videos captured by color or depth cameras. Skeleton poses and kinematic measurements are also used.

In this project, color videos are given as the training and testing data. The action recognition task is treated as a classification task. We will provide more detailed instructions about data, the choices of models, evaluation criteria, and submission requirements.

1.1 Action Recognition

The human action recognition task can be interpreted as a multi-class video classification task. The input of the model is a multi-frame video and the model output is the predicted classification probability vector for the corresponding multi-frame video.

1.2 UCF YouTube Action Dataset (UCF11)

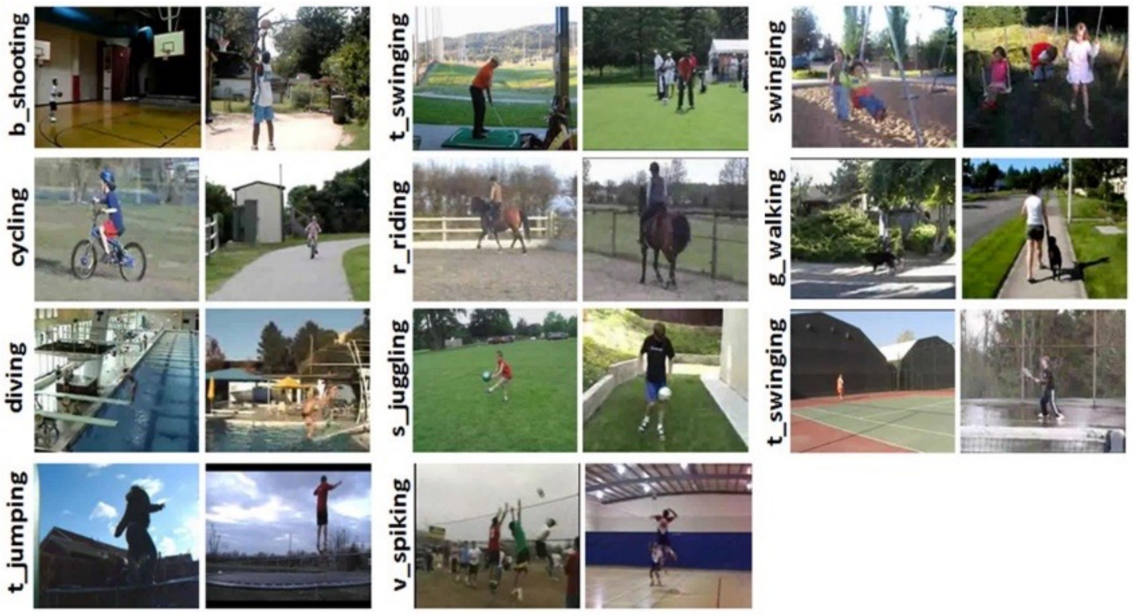


Figure 1: Example images from UCF11 dataset

The dataset for this final project is UCF YouTube Action Dataset[8] (Denoted as UCF11).

The dataset has a total number of 1168 videos within 11 action categories: basketball shooting, biking, diving, golf swing, horseback riding, soccer juggling, swinging, tennis swinging, trampoline jumping, volleyball spiking, and walking with a dog. For each video, the resolution is 320×240 and the length varies. More detailed information can be found here.

1.2.1 Load data and labels

We have already pre-processed the data for your convenience. The pre-processing procedures include:

- Reshape the resolution from 320×240 to 64×64.
- Divide each video into a clip of 30-frame length.

You can download the pre-processed data here. The following example code is able to load the data we provided:

```
import pickle
data_file = open('youtube_action_train_data.pkl', 'rb')
train_data, train_labels = pickle.load(data_file)
data_file.close()
```

You are required to separate the provided data proportionally into two parts for training and validation respectively. You are free to choose the ratio between training dataset size and validation dataset size.

1.3 Loss function

The output tensor of the model, which has the same dimension as the label during training. You can use the same cross-entropy function in previous programming assignments in this project.

1.4 The model

In this project, you are required to construct a deep model to perform the task. In principle, there is no limitation on the choice of model. You are free to choose any model you want. Popular model structures include C3D[5], Two-stream CNN[7], LRCN[1], HRNN[2], ResNet[4], VGG[6] and Action Transformer Network[3]. Since you are given video data with temporal dependencies, we strongly recommend you to use models that can explicitly capture the temporal dependencies such as RNN, LSTM, GRU, or even transformer. If you decide to use other's models, you are required to cite the sources of the models. All the models need to be trained from scratch. Fine-tuning on a pre-trained model is not allowed .

1.5 Backpropagation

In the final project, you are free to use tf.GradientTape to help you compute the gradients. The example code of using tf.GradientTape to compute gradients and optimize your model is given below:

```
with tf.GradientTape(persistent = True) as T:
    output = your_model.forward(input_data)
    loss = compute_loss_function(output, input_labels)
model_gradient = T.gradient(loss, your_model.trainable_variables)
your_optimizer.apply_gradients(zip(model_gradient, your_model.trainable_variables))
```

1.6 Save Your Model

You may use functions in Tensorflow2.X like

```
model.save_weights()
model.save()
```

to save your model and you may use functions in Tensorflow 2.X like

```
tf.keras.models.load_model()
model.load_weights()
```

to load your model.

2 Evaluation Criterion and Submission Requirements

In this section, we provide more detailed instructions on the final project. You can use the configured environment as before and we still use Tensorflow2.X this time.

2.1 Code

For this assignment, you are allowed to use any functions in Tensorflow2.X to help you train the model. You can also use the example code in the document. However, you still need to follow certain rules below:

- **Document your code well.** You should provide detailed comments in your code. This will allow TAs to have a better understanding of your implementation and give reasonable partial points when your implementation is wrong.

- **Provide a script to show your model performance.** For the final project, you are required to submit a python script named 'run_file.py'. Your script should be able to load your trained model and evaluated its performance on validation data. The script should also output all the evaluation results. For example, print the final accuracies/losses or plot the curves.
- **Provide a README File.** You are also required to attach a README file in the submission to let the grading TAs know how to correctly run your python script.
- **Do not plagiarize.** For the final project, you are allowed to use any functions in Tensorflow. However, copying code segments from others(your classmates, online projects on Github/Gitlab, students who previously took the course, etc.) are not allowed and will be considered as committing plagiarism.

2.2 Report

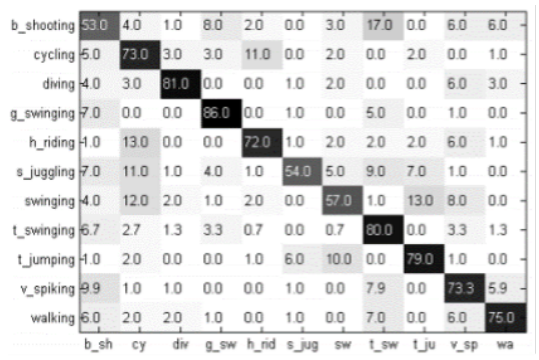
Besides submitting the well-documented code, you are also required to submit a report. Your grade on the final project largely depends on this report and you will suffer a penalty if the report is not well-written. We suggest you using Latex, Markdown, or Microsoft Word to write the report. For the final project, we don't accept the late submission. PLEASE submit it before the deadline.

Here is a list of components that MUST be included in the report:

- Problem statement: A brief introduction to the problem you have solved. Don't copy any sentence in this document. Please write it yourself.
- Model description: Explicitly describe your model. The description should include but are not limited to architecture, activation functions/filters settings, and hyperparameters settings.
- Pseudo-code(optional).
- Training and validation loss curves.
- Training and validation accuracy curves (class-wise and overall accuracies): The accuracy is defined as:

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

- Confusion Matrix: The confusion matrix allows the visualization of the model performance. It summarizes the prediction results of the classification task. Usually, the rows represent the instances of predicted labels and the columns represent the instances of true labels. From Figure 2, we can infer from the first row that the model classified a total of 100 videos as basketball shooting action. The number 53 in the first column indicates that 53 of the 100 videos are actually basketball shooting action and are correctly predicted by the model. The number 4 in the second column indicates that 4 videos are cycling actions and are wrongly predicted as basketball shooting actions by the model.



	b_sh	cy	div	g_sw	h_rid	s_jug	sw	t_sw	t_ju	v_sp	wa
b_shooting	53.0	4.0	1.0	8.0	2.0	0.0	3.0	17.0	0.0	6.0	6.0
cycling	5.0	73.0	3.0	3.0	11.0	0.0	2.0	0.0	2.0	0.0	1.0
diving	4.0	3.0	81.0	0.0	0.0	1.0	2.0	0.0	0.0	6.0	3.0
g_swinging	7.0	0.0	0.0	86.0	0.0	1.0	0.0	5.0	0.0	1.0	0.0
h_riding	1.0	13.0	0.0	0.0	72.0	1.0	2.0	2.0	2.0	6.0	1.0
s_juggling	7.0	11.0	1.0	4.0	1.0	54.0	5.0	9.0	7.0	1.0	0.0
swinging	4.0	12.0	2.0	1.0	2.0	0.0	57.0	1.0	13.0	8.0	0.0
t_swinging	6.7	2.7	1.3	3.3	0.7	0.0	0.7	80.0	0.0	3.3	1.3
t_jumping	1.0	2.0	0.0	0.0	1.0	6.0	10.0	0.0	79.0	1.0	0.0
v_spiking	9.9	1.0	1.0	0.0	0.0	1.0	0.0	7.9	0.0	73.3	5.9
walking	6.0	2.0	2.0	1.0	0.0	1.0	0.0	7.0	0.0	6.0	75.0

Figure 2: Example Confusion matrix of UCF11 dataset

- A brief discussion of the settings (Tensroflow version and environment settings) and the hyper-parameters you use: You can also describe the impact of changing model structure on model performance.
- Clearly state your device and environment settings in the report: We will grade the model performance based on both your computing resource and your model performance.

2.3 Submission

The following items should be submitted by **11:59 PM May 12nd, 2021**:

- The report in pdf format.
- The well-documented code.
- The saved models.
- The 'run_file.py' script.
- The README file.

3 Tips

- Some structures in the classic neural network such as residual block may be useful when you design your network.
- Since you have done several PAs before, you must be familiar with Tensorflow and deep learning. We encourage you to try some skills such as learning rate decay, data augmentation, batch normalization, and optimizer selection to enhance your model performance.
- The training process may take several hours if you have limited computational power. Try to start early and use as few loops as possible in your code.

4 Contact Information

This final project will be graded by both TA Xiao Jin and TA Naiyu Yin. If you still have confusion about the instructions, feel free to contact the TA (jinx2@rpi.edu, yinn2@rpi.edu) or come to his/her office hour.

References

[1] J. Donahue, L. Hendricks, M. Rohrbach, S. Venugopalan, S. Guadarrama, K. Saenko, and T. Darrell. Long-term recurrent convolutional networks for visual recognition and description. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(04):677–691, apr 2017.

[2] Y. Du, W. Wang, and L. Wang. Hierarchical recurrent neural network for skeleton based action recognition. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1110–1118, 06 2015.

[3] R. Girdhar, J. a. Carreira, C. Doersch, and A. Zisserman. Video Action Transformer Network. In *CVPR*, 2019.

[4] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[5] S. Ji, W. Xu, M. Yang, and K. Yu. 3d convolutional neural networks for human action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1):221–231, 2013.

[6] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings.*

[7] K. Simonyan and A. Zisserman. Two-stream convolutional networks for action recognition in videos. In *Advances in Neural Information Processing Systems*, volume 27, 2014.

[8] K. Soomro, A. Zamir, and M. Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. In *ArXiv*, 12 2012.