Video Classification

Project 2

Introduction to Deep Learning in Computer Vision

October 2025

In this project, you will be classifying videos according to their actions. You will work on a subset of the ufc-101 dataset for video action recognition derived from this Kaggle version of UFC-101. You can find the designated dataset, as well as the file `datasets.py` on Learn.

Dataset and tools

The provided dataset includes 720 videos (500/120/120 for train/val/test) of 10 balanced classes related to workout, mode precisely including the classes ['BodyWeightSquats', 'HandstandPushups', 'HandstandWalking', 'JumpingJack', 'JumpRope', 'Lunges', 'PullUps', 'PushUps', 'TrampolineJumping', 'WallPushups']

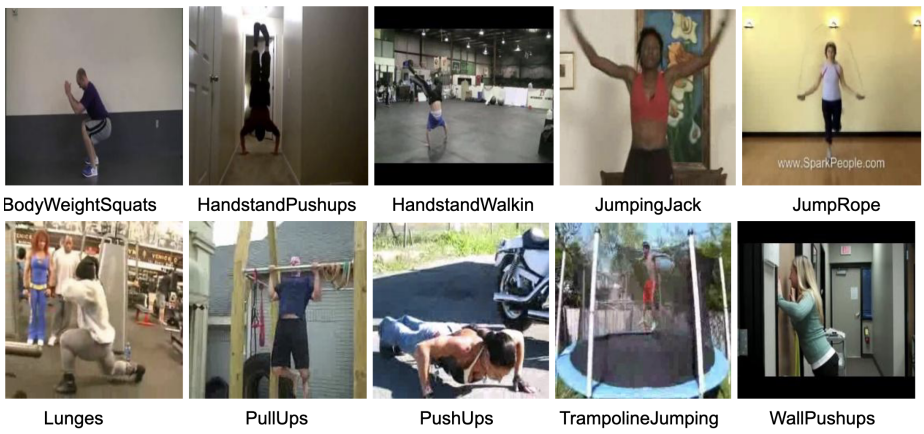

Figure 1: Examples from the UCF-101 dataset.

For each video, we provide:

- 10 uniformly sampled frames
- csv files that include the video name, relative path, actions, and labels.

Moreover, we provide example dataset classes and dataloaders in datasets.py to help you start with both 2D and 3D modeling. You may find the dataset of frames useful for training per-frame models, and the videos dataset useful either for testing on videos or training early/late fusion models (if $\text{stack}_frames = True returns the sampled frames are stacked as [C, T, H, W], else as a list$).

Project 4.1

In this first part of the project, your task is to explore different models for video classification, namely:

- aggregation of per-frame models
- late fusion
- early fusion

... all for 2D CNN models, along with 3D CNNs.

Project 4.2

Task 4.2.1: Information leakage between train/val/test splits

Your first task will address the problem of **information leakage between training/validation/test splits**. As you already know, you should not repeat images between splits, and you should only use the test set for final testing, not for model selection.

However, what might be less evident is that you **need to sample your splits independently** from the data distribution. In particular, when your data comes from humans, different instances of the same human are often closely correlated, and you therefore should take care **not to include the same subject in multiple splits**. Splits must be created at individual-level to ensure accurate evaluation.

The dataset you analyzed in 4.1 was retrieved, splits included, from Kaggle. However, Kaggle splits were different from the original UCF101 splits, and in particular, there was leakage between the Kaggle version's splits: Different splits included videos of the same person carrying out a given action. In the first part of project 4.2, we therefore ask you to select one of your well-performing models from project 4.1, and retrain the model using the updated dataset `ucf101noleakage`, which you can now find on `/dtu/datasets1/02516` both as a directory and as a zip file (in case you want to download it elsewhere).

Task 4.2.2: Dual-stream networks

Now that you have most likely observed a drop in performance when evaluating on splits without leakage, let's try a more powerful model: Dual-stream networks that include optical flow as input:

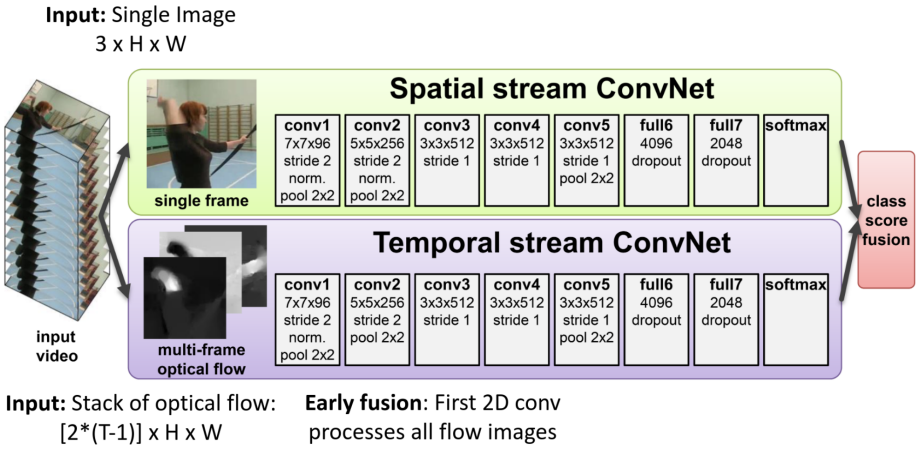

Figure 2: Two-stream architecture for video classification, Simonyan and Andrew Zisserman, NeurIPS 2014.

To support you in this endeavor, the updated dataset `ucf101noleakage` found on `/dtu/datasets1/02516` contains pre-computed optical flows between the 10 uniformly sampled frames. This optical flow has been generated using the RAFT model[1]. You can inspect the optical flows in the flows-png folder. An example is also shown below.

Hand-in

Your process, performance evaluation, and results should be documented and discussed in a PDF report to be uploaded on DTU Learn. All tasks of this project should be described in the same document.

Please submit a 3-page report on your project using the CVPR paper template we provide. These 3 pages:

- should include all tables and figures,
- do not have to include references and the mandatory statement on use of generative AI.

The project report should be uploaded as a single pdf per group, and will be graded as pass/fail.

[1] RAFT: paper, pytorch implementation