Homework 2: Video-based Multimedia Event Detection

11-775 Large-Scale Multimedia Analysis (Spring 2021)

Due on: Monday, March 22, 2021 11:59 PM

1 Task

The goal of homework 2 is to perform multimedia event detection (MED) with visual features from videos. You will continue to extend and refine the MED pipeline you built in HW1. Please **START EARLY**! It takes time to understand different modules in the pipeline. Employing visual features is more **time-consuming** than that of audio features in homework 1.

2 MED Pipeline Overview

With the MED pipeline you built in homework 1, you have already finished a great portion for this assignment. In the audio-based MED pipeline, you first extract audio features from the audio, pack them into fixed-length representations for each video, and you apply a classification model to get the prediction for each video. In this homework, instead of using the audio part of videos, you will learn to process the visual part of videos and extract two kinds of visual features (SURF/ResNet) to build video representations. You could re-use the MLP and SVM modules you built in homework 1 or try to further improve them in homework 2. As a reminder, it is a good idea to speed up your MED pipeline and keep the interfaces clean and easy to integrate so that you have less work for homework 3 where you will learn how to fuse audio and visual models.

To help you get started, we provide you an exemplary framework. Please refer to **`spring2021/hw2/`** in the GitHub repository of this course: `https://github.com/11775website/11775-hws.git`. Please note that the provided code is just an example that helps you to understand the basic components in the MED system. Also, the original parameters may not perform well. Therefore, you are NOT required to follow the provided example pipeline. We encourage you to create your own pipeline and try different tools, features, and configurations to get better results. The only requirement is to write a **`README.md`** in your GitHub repository to explain how to run your pipeline.

3 Dataset

The dataset will be shared for all three homework assignments. Recall the audio-visual dataset explained in homework 1 that contains 7,942 10-sec videos (around 22 hours of video in total). There are 5,662 videos for training/validation and 2,280 videos for testing. There are 10 classes/events in this dataset: C00: *dribbling basketball*, C01: *mowing lawn*, C02: *playing guitar*, C03: *playing piano*, C04: *playing drums*, C05: *tapping pen*, C06: *blowing out candles*, C07: *singing*, C08: *tickling*, and C09: *shoveling snow*. You may simply reuse the training and validation split you used

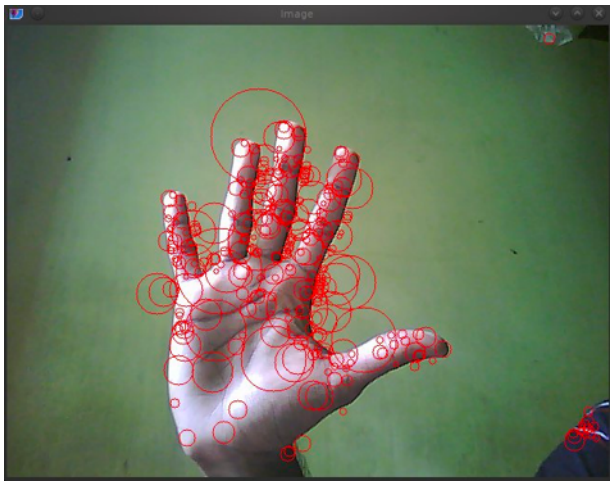

Figure 1: Visualization of key-points detection from the SURF descriptor.

in the previous homework. In this homework, we ask you to report the testing results based on the visual features of videos.

4 Visual Features

A video can be viewed as a sequence of RGB frames. To represent a video, the simplest ways is to extract the features of its individual frames and use some pooling functions to aggregate these frame-level visual features to a fixed-length representation. In this homework you will investigate two types of visual features: 1) The hand-crafted Speeded Up Robust Features (SURF) feature and 2) the Convolutional Nerual Network (CNN) features.

4.1 Video Pre-Processing

To extract the RGB frames from a video, you can simply use the `ffmpeg` tool introduced in homework 1 with the following command:

```
ffmpeg -y -i HW00006645.mp4 temp/frame%04d.jpg
```

The video frames of HW00006645.mp4 will then be stored in the `temp` folder. You may check ffmpeg manual for more options. As you learn in homework 1-2, ffmpeg is a versatile tool for audio and video pre-processing.

In practice, you may not have enough storage to store these RGB frames (you may notice that they consume much more space then the videos), an alternative way to extract RGB frames on-the-fly then process them. To do this, you don't need to re-invent the wheels. You can use `cv2.VideoCapture` in the OpenCV python toolkit[1] to dynamically load and decode videos. With the frames of a video at hand, you can then extract the visual features of each frame and aggregate (e.g., averaging across frames) them to generate video representations. You can choose to use extracted frame images (with ffmpeg or other tools) or integrate OpenCV to decode images on-the-fly in your MED pipeline.

[1] https://anaconda.org/conda-forge/opencv

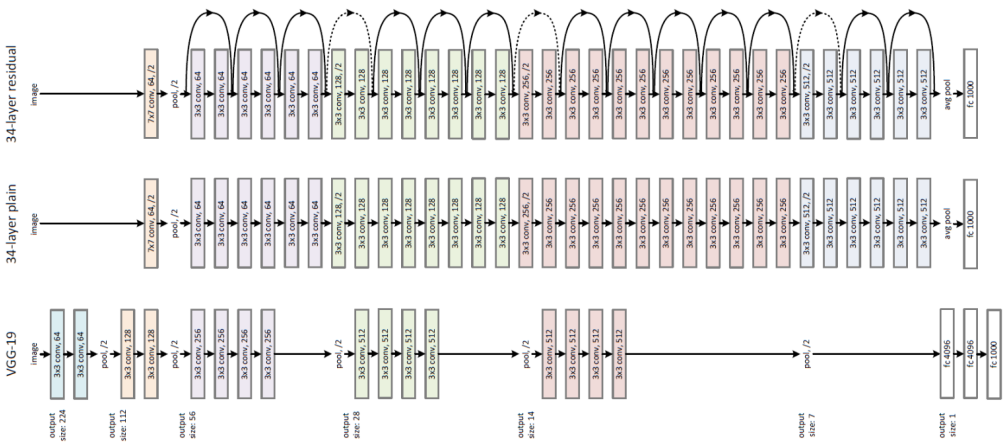

Figure 2: 34-layer ResNet with Skip / Shortcut Connection (Top), 34-layer Plain Network (Middle), 19-layer VGG-19 (Bottom).

4.2 SURF Features

SURF is a kind of hand-crafted local feature detector and descriptor, which was widely applied to tasks such as object recognition and image classification. The SURF descriptor could be viewed as an extension of the Scale-Invariant Feature Transform (SIFT) descriptor. The SURF algorithm first extracts key locations (interest points) of an image and then extracts visual attributes around these locations. Key locations are defined as maxima and minima of the Hessian matrices on a series of smoothed and re-sampled images. Sum of Haar wavelet responses in horizontal and vertical directions are then being assigned to these key-points as feature descriptors. A visualization result for the SURF descriptor is depicted in Figure 1. Please refer to the original paper for more details[2].

To build video representations based on SURF, the typical approach is to pre-process a video into key-frames, extract features, and then pack these extracted key-frame features into one video-level representation. Specifically, you may implement the following steps:

1. **Extract SURF:** You can use the OpenCV python toolkit to extract SURF features over the key-frames of down-sampled videos. There are many examples online [3] you may like to reference. A simple way to select key-frames is to use a fixed interval at 5 frames. In this way, the 0^{th}, 5^{th}, 10^{th}, 15^{th}... frames of the down-sampled videos are utilized for SURF feature extraction.

2. **Perform clustering** to train and generate the Bag-of-Words representation of each key-frame. This is similar to what you have done in homework 1. However, this is still not the video level representation, since the number of key frames in each video varies.

3. **Average/Max-Pooling** over the Bag-of-Words SURF feature representations of frames then perform normalization to aggregate the final video-level SURF representation.

Once you get the video representations with SURF features, you can use the SVM and MLP pipeline for training and testing.

[2] https://people.ee.ethz.ch/ surf/eccv06.pdf

[3] https://enumap.wordpress.com/2012/10/30/python-opencv-surf-use-cv2/

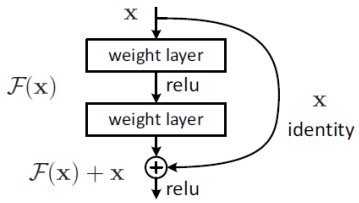

Figure 3: A Building Block of Residual Network. The weight layers are some convolutional filters.

4.3 CNN features

In recent years convolutional neural networks gain popularity as it provides robust feature representations pre-trained on some large-scale dataset such as ImageNet. In this homework you will get a taste of these CNN feature and find the answer for the following question: Are the learned features better than hand-crafted features?

Feature learning with CNNs has become a hot topic in computer vision and many other fields. One of the widely used CNNs for various computer vision tasks is the deep Residual Network (ResNet)[4]. As shown in Fig 3, the basic building block in ResNet is to add residual links to CNNs. Empirically, ResNet achieved state-of-the-art performance on ImageNet classification and became the backbone of many models for different computer vision tasks. Fig. 2 shows a comparison of ResNet to other CNNs such as the VGG network.

In this part, please select your favorite neural network tools (e.g., PyTorch or Tensorflow) to extract the image-level ResNet features for the extracted video frames. You may choose a 34-layer or a 101-layer ResNet to extract visual features. What's the size of the the 2D feature map in the layer you choose to extract features from? How do you aggregate them? There are lots of tutorials and resources available online (e.g., img2vec[5]) you may find useful. You may also try more advanced models (e.g., I3D[6]) that directly take video clips as input instead of operating over independent RGB frames.

Please try to explore features from different layers, you may find the results quite interesting. You may skip the clustering process for CNN features as they are much condense then hand-crafted features. Please note that there is no limitation for the CNN tools and models. You are free to use any other models beyond ResNet. Please note that a CPU instance would be sufficient for feature extraction in this homework. You can use GPU instance to speed up for your final submission but please keep the AWS computation cost in mind. Once you have the video representations with CNN features, you can use the original training/testing pipeline to get the results.

4.4 VLAD Encoding (30% Bonus)

This part is fully optional and is for the brave! Recall the Bag-of-Words for image representation, it involves simply counting the number of descriptors associated with each cluster in a codebook (vocabulary). Then, it creates a histogram for each set of descriptors from an image. In this way, it represents the information in an image in a compact vector. The Vector of Locally Aggregated Descriptors (VLAD) is an extension of this concept. We accumulate the residual of each descriptor with respect to its assigned cluster. In simpler terms, we first match a descriptor to its closest cluster as in Bag-of-Words. Later, for each cluster, we store the sum of the differences of the descriptors assigned to the cluster and the centroid of the cluster. The mathematical formulation for VLAD can

[4]https://arxiv.org/abs/1512.03385
[5]https://becominghuman.ai/extract-a-feature-vector-for-any-image-with-pytorch-9717561d1d4c
[6]https://deepmind.com/research/open-source/i3d-model

be derived as following.

As with Bag-of-Words, we train a codebook from the descriptors collected in our datasets, denoted as $C = \{c_1, c_2, \ldots, c_k\}$, where k is the number of cluster centers specified in the K-means algorithm. We then associate each d-dimensional local descriptor, x_i from an image to its nearest neighbor in the codebook:

$$c_i = NN(x_i) = argmin_{c \in C} \|x_i - c\|$$

The idea behind VLAD feature quantization is simple. For each cluster centroid, we accumulate the difference of $x_i - c_i$. Therefore, the VLAD representation for an image V is a $(k \times d)$-dimensional matrix:

$$V_{i,j} = \sum_{x|NN(x_i)=c_i} (x_j - c_{ij})$$

The matrix V is subsequently flattened into a vector v, followed by normalization with its L2-norm:

$$v = \frac{v}{\|v\|_2}$$

The primary advantage of VLAD over Bag-of-Words is that we have more discriminative property in our feature vector by adding the difference of each descriptor from the mean in its Voronoi cell. This first order statistic gives more information in our feature vector and hence tends to exhibit us better discrimination for our classifier.

As a bonus, in this assignment, you can implement the VLAD encoding strategy for the visual features (SURF/CNN) in the previous section. Please compare the performances with and without VLAD encoding. If you choose to do the bonus, please describe your implementation in your report. Your work on VLAD encoding will give you up to 30% extra credits.

5 Model Training and Testing

As in homework 1, you may use SVM or MLP for training and testing the model. You may need to experiment more to tune hyperparameters that yield the best performance. Beside SVM and MLP, you can also try out fancier idea such as adding residual links to MLP and/or using different regularizers to improve the robustness and generalizability of your classification model.

6 What to submit

In Canvas: Please compress your submission into **`ANDREWID_HW2.zip`** and submit it through the turn-in link in Canvas. The contents of your zip file should be organized as the following:

1. **report.pdf**: Your pdf report for homework 2.
2. **github_kaggle.txt**: A txt file with the link to your GitHub repository and your Kaggle account. Please be sure to add a README.md explaining how to run your code or script.
3. **surf.csv**, **cnn.csv**, and **best.csv**: The classification results for each testing video. The csv format is:

Example 1

```
Id,Category
HW00002897,7
HW00001276,1
HW00000794,0
```

In your report, please:

1. Describe the steps and parameters in your MED pipeline with two types of features: SURF and CNN. If you do the VLAD bonus, please also provide explanation about your VLAD implementation.
2. Report the **confusion matrix**[7] for multi-class classification in your validation set. Which feature is better? Which class(es) is harder and with which it is confused? Do you have insight or explanation for the difference in accuracy?
3. Report the time your MED system takes (CPU time) for feature extraction and classification on the testing set. Please also tell us the amount of credits left on your AWS account.

In Kaggle: Please submit your **best.csv** to the Kaggle leaderboard on https://www.kaggle.com/c/11775-hw2 and briefly explain your method. You can submit up to 5 times/day. The final performance on the whole testing set will be revealed on March 22.

[7] https://towardsdatascience.com/multi-class-metrics-made-simple-part-i-precision-and-recall-9250280bddc2