Homework 2: Video-based Multimedia Event Detection

11-775 Large-Scale Multimedia Analysis (Spring 2024)

Due on: Monday March 11, 2024 11:59 PM

1 Overview

The goal of homework 2 is to perform multimedia event detection (MED) with visual features from videos. Please **START EARLY**! Employing visual features is more time-consuming than that of audio features in HW1.

To help you get started, template code with instructions is provided. A Conda environment provides the dependencies you will need for this homework. The template uses PyTorch and Pyturbo. You are NOT required to follow the template. But if you find Pyturbo helpful or have suggestions, feel free to STAR or open pull requests on Github.

In this homework, you will use the same dataset as HW1. There is also a Kaggle competition for you to participate. Instead of extracting audio features, in this homework you will learn to process the visual part of videos.

A video can be viewed as a sequence of frames, where each frame is just an image. To represent a video, the simplest way is to separately extract the features of each frame and aggregate them into a fixed-length representation. There are also more advanced algorithms or models that could directly extract features from a video clip. In this homework, you will investigate three kinds of visual features:

- Hand-crafted Scale-Invariant Feature Transform (SIFT) [2] features. You can use OpenCV to extract the SIFT features.
- Learning-based Deep Convolutional Neural Network (CNN) features, e.g. ResNet [1]. You can use the models pre-trained on ImageNet available in TorchVision.
- Learning-based video-level features, e.g., 3D ResNet.

2 Part I: SIFT Features (30%)

The SIFT algorithm first extracts key locations of an image and then extracts visual attributes around these locations, as shown in Figure 1. For each frame, you will get a $N \times D$ feature matrix, where N is the number of key locations and D is the feature dimensions. As N varies across different images, the size of the feature matrix for each image is not fixed. Furthermore, the number of frames in each video also varies. Therefore, to get a fixed-length representation, you will learn a bag-of-visual-words representation, similar to what you have done in HW1.

Applying SIFT to each frame is not computationally cheap. And adjacent frames often have similar appearance, so it would be a wise choice to downsample the frame rate (frame-per-second, fps) before extracting features. There is also a speeded-up version of SIFT called SURF. However, it is a bit difficult to install with OpenCV in Python due to legal issues. If you are interested and feel confident about building it from source, you can try it here.

You will cluster the feature vectors with the K-Means algorithm, where each sample corresponds to one key location in one frame. To speed up the clustering, only a subset of the key locations will be used. It is up to you how to select them: a subset of frames from each video, a subset of locations from each selected frame, etc. The number of clusters is also a design choice for you.

Now that you have trained the K-Means clusters, you can assign each feature vector to a cluster to form the Bag-of-Words representation for each frame. As the number of frames varies for different videos, you will further need to average or max-pooling over the Bag-of-Words vectors of each frame

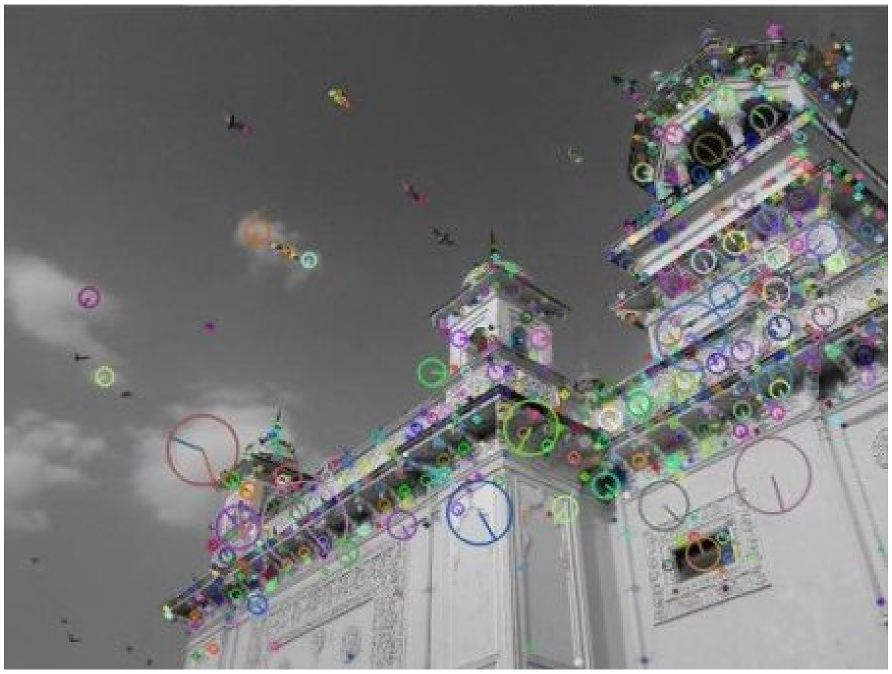

Figure 1: Visualization of key-points detection from the SIFT descriptor.

to get the final video-level feature. Then you can train a classifier that you prefer (e.g., SVM or MLP used in HW1) to classify the videos.

You are required to report your design choices, implementation details, execution time, and result analysis based on the SIFT features and Bag-of-Words representation, with a handin prediction file named **sift.csv**. You should also submit this prediction file to Kaggle and include its Public test accuracy in your report. To qualify for a full score of this part, your Public test accuracy should be **no less than 0.02** below the submission marked as *sift_baseline*.

3 Part II: CNN Features (40%)

In recent years convolutional neural networks gain popularity as it provides robust feature representations pre-trained on some large-scale datasets 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). As shown in Fig 2, 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 3 shows a comparison of ResNet to other CNNs such as the VGG network.

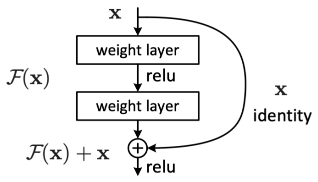

Figure 2: A Building Block of Residual Network. The weight layers are some convolutional filters.

In this part, PyTorch is recommended and well supported by TAs but feel free to use other neural network tools you like. You will use it to extract the image-level CNN features for selected frames,

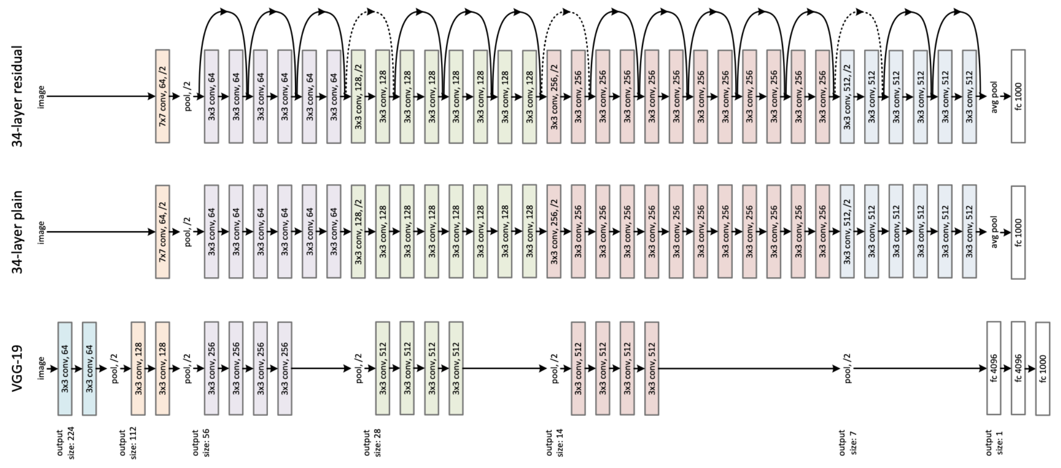

Figure 3: 34-layer ResNet with Skip / Shortcut Connection (Top), 34-layer Plain Network (Middle), 19-layer VGG-19 (Bottom).

instead of the hand-crafted SIFT feature. There is a large set of models here with ImageNet pre-trained weights, where ResNet-18 could be a good starting point. These models are built for the ImageNet classification task, but you can use this tool to extract features from intermediate layers.

You will need to consider some of these questions when extracting the features. What's the shape of the 2D feature map in the layer you choose to extract features from? How do you aggregate them? 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 than hand-crafted features. So you can directly use these features for classification.

You are required to report your design choices, implementation details, execution time, and result analysis based on the CNN features, with a handin prediction file named **cnn.csv**. You should also submit this prediction file to Kaggle and include its Public test accuracy in your report. To qualify for a full score of this part, your Public test accuracy should be **no less than 0.02** below the submission marked as *cnn_baseline*.

4 Part III: Video-level Features (30%+20%)

In this part, you will try some more advanced models that directly take video clips as input instead of operating over independent RGB frames. These models include 3D CNNs, (2+1)D CNNs, transformers, etc. You can feel free to explore any architectures or pre-trained weights. You can also try training a model end-to-end, which would also be fun.

You are required to report your design choices, implementation details, execution time, and result analysis based on the video features, with a handin prediction file named **video.csv**. You should also submit this prediction file to Kaggle and include its Public test accuracy in your report. To qualify for a full score of this part, your **Private** test accuracy should be **no less than 0.02** below the submission marked as *video_baseline*. If your **Private** test accuracy can surpass the submission marked as *video_baseline*, you can earn a bonus of additional 20% points.

5 Submission

5.1 Canvas

Please compress your submission into a zip file named as `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 with your pipeline design, findings, results, and analysis.

2. **code.zip**: A .zip file with your code only. Please be sure to add a `README.md` with the instructions on how to run your code to reproduce the results.

3. **sift.csv**, **cnn.csv**, **video.csv**: The classification results for each testing videos.

For the CSV files, the following format should be used:

```
Id,Category
LTExODM2MzcOODQyOTc1ODE4NDM=,7
LTUwNDU3NzgyNjE2MzkOOTU1NjQ=,1
ODU3OTEOMDU5NzM5NDI2MDQ2,0
...
```

In the report, please describe the detailed steps and parameters in your MED pipeline for extracting the three types of features, as well as the feature aggregation methods and hyper-parameters of the classification models during training. Then, specify the size of your training and validation sets, as well as the validation metric you used for validation, e.g. top-1 accuracy, top-5 accuracy, average precision, etc. Please report your best model's performance on the validation set. For the report, we also ask you to add the confusion matrix, and make an analysis that describes which classes are harder to detect and with which other classes those are being confused. Then, please report the time your MED pipeline takes and the hardware platform.

5.2 Kaggle

Please submit your **sift.csv**, **cnn.csv**, and **video.csv** files to this Kaggle competition and report their Public test set accuracy in your report. You can submit up to 10 times/day. The final performance on the Private test set will be revealed after the due date.

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[2] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.