Homework 2: Video-based Multimedia Event Detection

11-775 Large-Scale Multimedia Analysis (Spring 2022)

Due on: Wednesday March 2, 2022 4:30 PM

1 Overview

The goal of homework 2 is to perform multimedia event detection (MED) with visual features from videos. This homework contains two parts:

- In the first part, you will extract visual features to classify the same data as HW1.
- In the second part, you will scale to a much larger test video collection and have more fun.

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 here. 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.

2 Part I (80%)

In this part, 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 two kinds of visual features:

- Hand-crafted Scale-Invariant Feature Transform (SIFT) [4] features. You can use OpenCV to extract the SIFT features.
- Learning-based Deep Convolutional Neural Network (CNN) features, e.g. ResNet [3]. You can use the models pre-trained on ImageNet available in TorchVision.

2.1 SIFT Features (40%)

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

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

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 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_p1.csv**. You should also submit this prediction file to Kaggle and include its Public test accuracy in your report.

2.2 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.

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, 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.

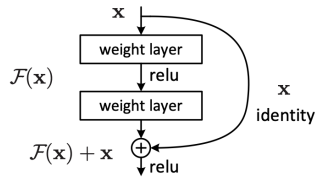

Figure 2: A Building Block of Residual Network. The weight layers are some convolutional filters.

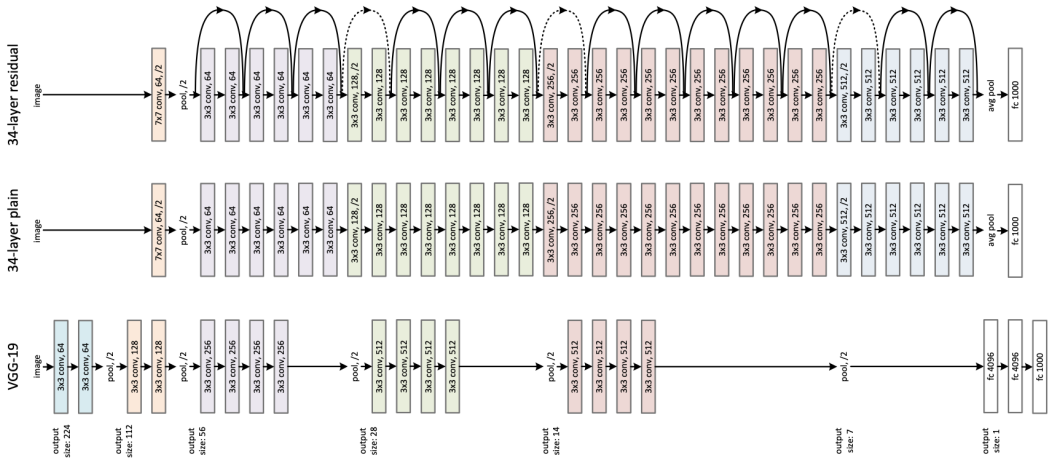

Figure 3: 34-layer ResNet with Skip / Shortcut Connection (Top), 34-layer Plain Network (Middle), 19-layer VGG-19 (Bottom).

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 may also try some more advanced models that directly take video clips as input instead of operating over independent RGB frames. However, to prevent the information leak of our test set, 3D CNNs pre-trained on Kinetics, AVA, and ActivityNet datasets are NOT allowed in this homework. Some exceptions of IG-65M pre-trained models are available here and here, but ONLY use those pre-trained on IG-65M without anything else. You can try training a CNN 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 CNN features, with a handin prediction file named **cnn_p1.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 *cnn_baseline*.

3 Part II (20% + 30% bonus)

WARNING: DO NOT START Part II before you have finished Part I.

In Part I, you have processed a "large" video dataset, which contained 7500 training/validation videos and 749 test videos from 15 classes. In fact, this is only about 1/40 of a much LARGER dataset. So in Part II you will scale up your pipeline to handle that. You should only start doing this part when you have finished Part I and have a reasonable classification accuracy, as you can only rerun for a few times.

3.1 More Data (20%)

As you can see, training takes significantly longer time than testing, and your AWS budget is fairly limited. Therefore, you will only be asked to process a large test set with your previous **best** model

without using additional training data. You should continue use the previous label mapping of 15 classes.

The new test set consists of 29694 videos. You should measure the entire processing time for the Part I test set, from feature extraction to classifier inference, to get an estimate of how long this set will need. It is 40 times larger than the previous test set, but don't panic, it is only 4 times of the previous training set. A reasonably implemented pipeline should finish within 16 hours on `g4dn.4xlarge`, and a well optimized one can get much faster than that. You can iteratively optimize and test on the small test before you go for the large set.

Things become different quickly when you scale up, as many negligible cost could take a lot of time. Think about what takes the most time in your pipeline. Is it the CNN model or is it the MLP? Is your GPU mostly idle waiting for data from CPU, or is it fully utilized? Keep in mind that GPU processes data much faster than CPU, and also much more expensive. Therefore, it is not cost-effective if your pipeline keeps the GPU at a low utilization. You can increase the batch size, run multiple models in parallel, or interleave the pipeline stages. The Pyturbo framework provides much flexibility for these to maximize your efficiency.

You are required to report your design choices, implementation details, and execution time for optimizing to process this large test set, with a handin prediction file named **data_p2.csv**. You should potentially identify a few bottlenecks that you have addressed or should address if further optimization is desired. There is no accuracy requirement for this section. To avoid unexpected charges, you should only proceed when your estimated processing cost is below what is remaining in your account, or you should contact TAs to see if additional credit is available for you.

3.2 More Classes (30% bonus)

In this large test set, there are not only the 15 classes present, but a total of 599 classes as defined in the new label mapping file. However, for those new classes, no training examples are provided but only a text label. Your task is to find a way to distinguish between them without examples. This is often called zero-shot classification. It sounds challenging, right?

A good starting point is to map text labels into feature vectors, for example with BERT [2]. Then you can train an MLP to map the visual features into the text feature of their corresponding classes. You may also get some ideas from this paper [1]. You can only train on the 15-class training data, but hopefully the model will learn some semantic information to generalize.

In this section, you are allowed to use any models and methods including pre-trained ones. We are eager to see your creativity in solving this real problem. And feel free to bring your TAs with crazy ideas on Piazza or at office hours.

If you decide to do this part, you are required to report your design choices, implementation details, execution time, and result analysis for the zero-shot classification task, with a handin prediction file named **class_p2.csv**. You should also submit this prediction file to Kaggle and include its Public test accuracy in your report. The competition for this section is here.

4 Submission

4.1 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_p1.csv**, **cnn_p1.csv**: The classification results for each testing video in Part I.
4. **data_p2.csv**, **class_p2.csv**: The classification results for each testing video in Part II. Note **data_p2.csv** should use the previous 15-class label mapping, while **class_p2.csv** should use the full label mapping.

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

```
Id,Category
LTExODM2MzcOODQyOTc1ODE4NDM=,7
LTUwNDU3NzgyNjE2MzkOOTU1NjQ=,1
ODU3OTE0MDU5NzM5NDI2MDQ2,0
...
```

In the report, please describe the detailed steps and parameters in your MED pipeline for extracting both video features: SIFT and CNN, 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.

4.2 Kaggle

For Part I, please submit your **sift_p1.csv** and **cnn_p1.csv** files to this Kaggle competition and report their Public test set accuracy in your report. You can submit up to 5 times/day. The final performance on the whole test set will be revealed on March 2.

For Part II, please submit your **class_p2.csv** file to this Kaggle competition and report its Public test set accuracy in your report. You can submit up to 5 times/day. The final performance on the whole test set will be revealed on March 2.

References

[1] Biagio Brattoli, Joseph Tighe, Fedor Zhdanov, Pietro Perona, and Krzysztof Chalupka. Rethinking zero-shot video classification: End-to-end training for realistic applications. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4613–4623, 2020.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[3] 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.

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