Smart Home Project Part 2

Ryan Johnson

Problem Description and Solution

For the second part of the project we were supposed to create a training data set and compare them against a test dataset using the cosine similarity to determine the labels for the gesture.

In order to solve this, my solution needed to do 3 high level steps

1) Iterate through the training data and extract the feature vector of the gesture of the video.
2) Iterate through the test data and extract the feature vector of the gesture of the video.
3) Iterate through the feature vectors of the test data to calculate and compare the cosine similarity against the training vectors. The highest cosine similarity corresponded to the label.

Solution Explanation

In order to extract and cache the feature vector, I created a function that would iterate through the training data, and store the feature vector in a dictionary. This dictionary is a key value map, where the key is the name of the training video. The processes for creating the cache of the test data was the exact same.

In order to get the feature vector, it required reading and opening the video file using OpenCV. OpenCV is a library that is used for processing imagery, and has a lot of functions that are used for the model prediction functions. Using OpenCVs VideoCapture class, the code is able to find the number of frames in the video and extract the frame in the middle of the video. The middle frame of the video is used because it is the highest probability of containing the gesture that should be tested against.

This frame is then passed into the HandShapeFeatureExtractor class that was provided as part of the project. This class is a utility that loads the pre-trained prediction model into memory and makes predictions based off of a frame input.

Using the cosine similarity, we then can determine if the matching gesture from the test data matches a gesture in the training data. In order to calculate the cosine similarity, the python Numpy package makes this very easy. Numpy makes this very easy since it has both the dot() and norm() functions built into the library.

Once the similarity is calculated, the high score is kept, along with the name of the training video name. Once the highest matching cosine similarity is found, the next step is to find the label corresponding to the training video. In order to do this, I created a function that would translate the name of the file into the correct label. This function is very dependent on the name of the training videos, but it does return a successful label

Lessons Learned

There were a couple things that I needed to figure out for this project to get it done. The first was what the extract_feature() function was returning and how that related to the cosine similarity, and how that determined the relevance. I have used prediction models to label imagery before, but had not used this method to make prediction results and found it to be very useful.

The other thing that I learned, and learned it through doing it wrong, was how to identify the label. In an early approach to the assignment, I was using the cosine similarity to get predictions, but was doing the data comparison backwards. I was using the test data to compare against the training data, and was then using the test data files to determine the predicted label. This did in fact work, and I was getting a good score once submitted to gradescope. I was corrected by the TA, and realized that I was doing my check look backwards. Once that logic was updated, it worked better and I saw better results.

Conclusion

This was a great project that showed the multiple options when using a prediction model to generate labels for video data.