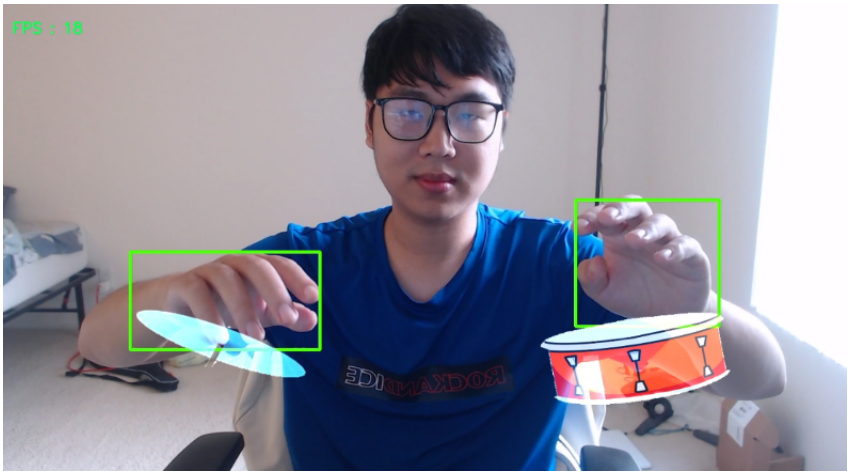
CS216 Project: Air Drum

Zezhong Pan



1 Project Overview

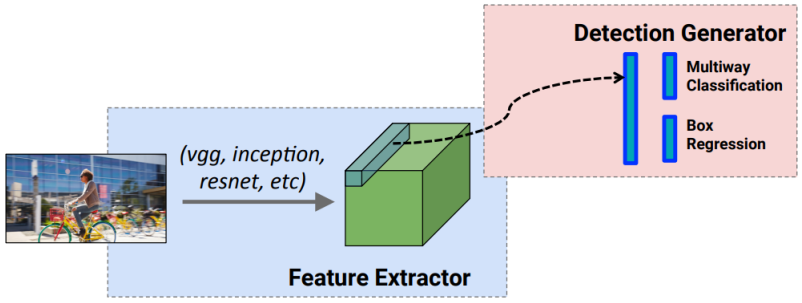
This project is a simple game built on real time hand detector. By reading frame information captured by web camera, the hand detector tries to locate the position of human hands in order to interact with game objects (drums) in the screen. When the location of hands satisfy certain conditions, the drum will be played. This process is usually referred to as region of interest (ROI) analysis in game development. To make the game run smoothly and fun to play, the hand detection must be fast enough to maintain a high frame rate while having a high detection accuracy. The goal of this project is to develop such a good hand detector that can be used to build a simple real time application.

2 Algorithms

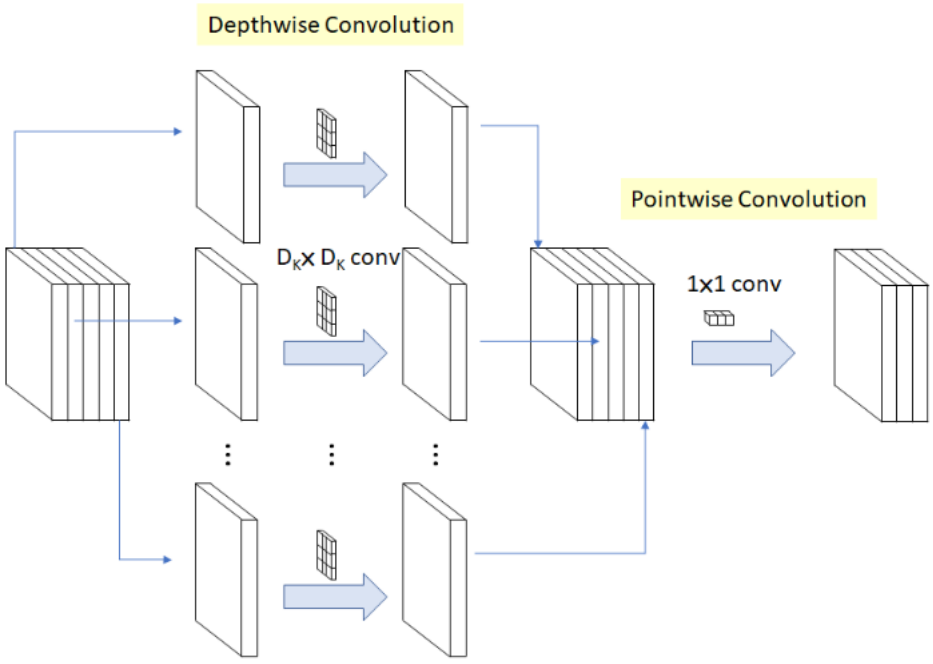
2.1 Hand Detector

There are multiple ways to detect hands from single image or frame. The original air drum project, from which I got my idea for this project, adopted a color based

detector. He used an API from OpenCV which can give the pixels of an image with colors that fall into a certain RGB range, and then use a pair of red drumsticks to hit the drum. This method is fast and accurate as long as the target object such as a red drumstick looks greatly different from background colors. However it does not work well for detecting hands. Hands has similar color as arms and face (sometimes even walls), which make the detection not very consistent. Other rule based detectors, extracting background based on texture and boundary features and distinguishing between hands and background using color histograms and HOG classifiers, also share the same problem of not being robust. So for this project, I decide to build a hand detector by training a neural network (NN).



Overview of Single Shot Detector (SSD)



Stucture of MobileNet (Feature Extractor)

The model I used is referred to as Single Shot Detector (SSD). It contains a convolution based feature extractor and neural network for box regression. Since we are only

interested in detecting hands there is only 1 class and thus classification is not needed. The feature extractor architecture I used is called MobileNet and it comes from a work by Google. MobileNets are based on a streamlined architecture that uses depth-wise separable convolutions to build light weight deep neural networks. With very little sacrifice on accuracy, this architecture is known for being efficient and is very suitable for real time object detection tasks.

I did not implement the neural networks from scratch, thanks to the object detection API from TensorFlow. The package allows developer to build their own DNNs by writing a configuration file, and it can automate the the training process given the training and development datasets in TensorFlow record format. After the training process, it will save the trained model as a inference graph. When we run the actual application, we can load the file for image inference.

2.2 Air Drum

The main body of Air Drum is a while loop where web camera frame is read in each iteration for processing. Each frame is first fed to image inference graph to get the positions and sizes (defined by 4 sides) of boxes that have the highest scores. Ideally the boxes should be bounding the region of hands in the image. After having the information the of bounding boxes, the program then need to decide whether they interact with the fixed game objects. This is called region of interest (ROI) analysis.

```
model = load_model()
while true:
    frame = Camera.capture()
    boxes = model.inference(frame)
    frame.draw(game_objects)
    ROI_analysis(boxes, game_objects)
    show(frame)
```

Pseudo code for single-threaded version of Air Drum

When implementing the main body of the air drum game, there is one more concern: performance. To make sure that the game runs smoothly, the application must maintain a high enough frame rate, which means each iteration of the loop has to run adequately fast. In practice, the single-threaded version only has frame rate at around 5 frames per second, which makes the game look very laggy. I also noticed that most of the time is consumed by model inference. To solve this problem, I came up with an idea to have multiple threads handle the inference process. In each iteration, a frame is added to an input queue. Multiple works wait on the queue to do image inference, and put the results on the output queue. Then for every iteration, we grab one result from the queue for ROI analysis.

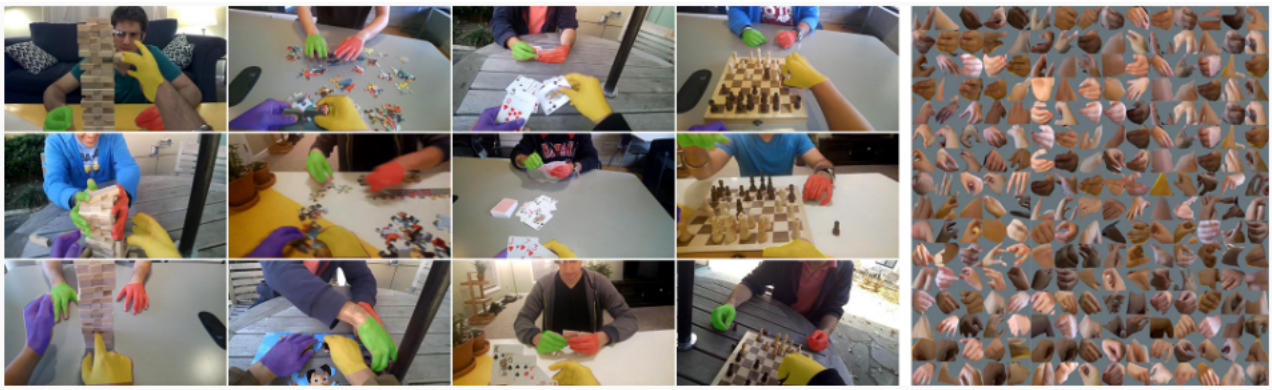
```
worker(in_queue, out_queue):
    model = load_model()
    while true;
        frame = in_queue.get()
        boxes = model.inference(frame)
        out_queue.put(boxes)

main():
    Pool(worker, num_threads, in_queue, out_queue)
    while true:
        frame = Camera.capture()
        in_queue.add(frame)
        boxes = out_queue.get()
        frame.draw(game_objects)
        ROI_analysis(boxes, game_objects)
        show(frame)
```

Pseudo code for multi-threaded version of Air Drum

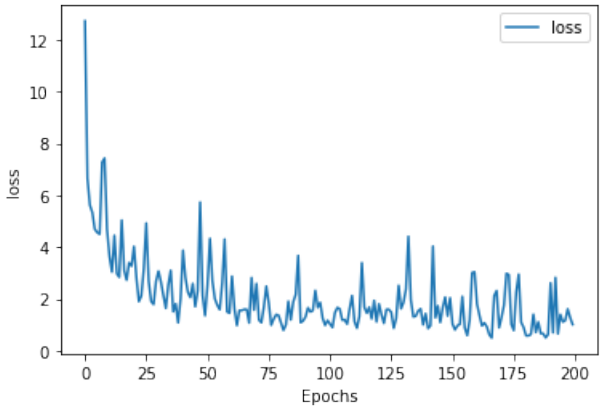
The multi-threaded version brings the frame rate to about 20 fps. This is an acceptable frame rate for playing the game. In practice, I set the number of workers to be 4. Although theoretically the more workers we have the faster the algorithm is, there is trade-off here. We can see from the pseudo code that each worker needs to load its own inference graph. Since the model is rather big size, we may end up using too much memory if we have too many threads. Also notice that ROI analysis does not run in parallel. This is because, unlike the inference graph, each game object can only have one instance, or we may hear the drum played multiple times even though we play it only once. This means that, if we handle the game object interaction in parallel, we will run into concurrency issues, and dealing with them will complicate the whole story, which is not worth it.

3 Data Sets



The dataset I used was collected from a project by IU Computer Vision Lab at Indiana University called EgoHands. The EgoHands dataset contains 48 Google Glass videos of complex, first-person interactions between two people. They also provide labeled data that are labeled frames as JPEG files (720x1280px). There are 100 labeled frames for each of the 48 videos for a total of 4,800 frames. The ground-truth labels consist of pixel-level masks for each hand-type and are provided as Matlab files. Python script is available online to convert the labels to CSV file. I wrote my own script to convert the CSV files to Tensorflow records to feed into Object Detection API.

4 Results



Training Loss vs. Epoch

Ground Truth vs. Inference. We can see that inferred bounding boxes are a little different from ground truth labels, but still capture both hands pretty well.

5 Assessment and Evaluation

In this project, I developed a simple air drum game that uses convolution neural networks for hand detection. The project focuses on the application of computer vision models. The whole idea is to experience a simplified but whole process of building a complete computer vision application. The most challenging part of the project is learn from different tutorials and read documentation and papers on each individual part

of the whole picture, and try to assemble them together. Finding public datasets and converting the format, training a object detection network using TensorFlow APIs, and developing the game using OpenCV and PyGame packages, the biggest success of this project is managing to link these separate parts. I learned a lot during the process and I am pretty happy that I managed to build something that actually works.

There are some improvements to this project that I wish I could do if I have had the time. One of them is the optimization of the hand detector model. Since the project focuses on application of computer vision rather than design, I did not conduct much comparison and hyper-parameter search on the neural networks to get a even better model for inference. There could be different networks that are more accurate and faster than the one I am currently using. Considering the fact that re-training a network takes hours, I had to proceed to developing the remaining part of the project once I obtained something that worked fine.

Another improvement I want to make is about refining the interaction of game objects. Currently the logic of ROI analysis is very simple. The audio is played whenever a detected hand enter a certain region above the game objects. Each game object has a simple state machine that switches off the object once it has been played and switches on after leaving the region to avoid the audio from being played repeatedly. A fancier ROI analysis may consider the motion of the hands, and play the audio if and as long as a complete motion of hitting from above is conducted. It is also a good idea to adjust the volume based on the speed of hitting.

6 Reference

1. Wei Liu, et al. SSD: Single Shot MultiBox Detector. 2016. arXiv:1512.02325v5.

2. Andrew G. Howard, et al. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. 2017. arXiv:1704.04861.

3. AIR Drums. `https://github.com/kaustubh-sadekar/AIR_Drums`. (The image and audio files come from this project)

4. Real-time Hand-Detection using Neural Networks (SSD) on Tensorflow. `https://github.com/victordibia/handtracking`

5. Tensorflow Object Detection API Tutorial. `https://pythonprogramming.net/introduction-use-tensorflow-object-detection-api-tutorial/`

7 Appendix

airdrum.py and game_object.py is completely writen by myself. csv_to_tfrecord.py and detector_utils.py are modified from some existing online source files. Other codes, such as codes for the ML models, are copied from existing packages.