Project 4: Assisquat

Mobile Systems for AI/ML - Towards a brief experience on system design from a top-down (application-driven) perspective

Source code: https://github.com/jmink1896/cs330-project4

Demo video: https://www.youtube.com/watch?v=lXdQTiX2Wa4

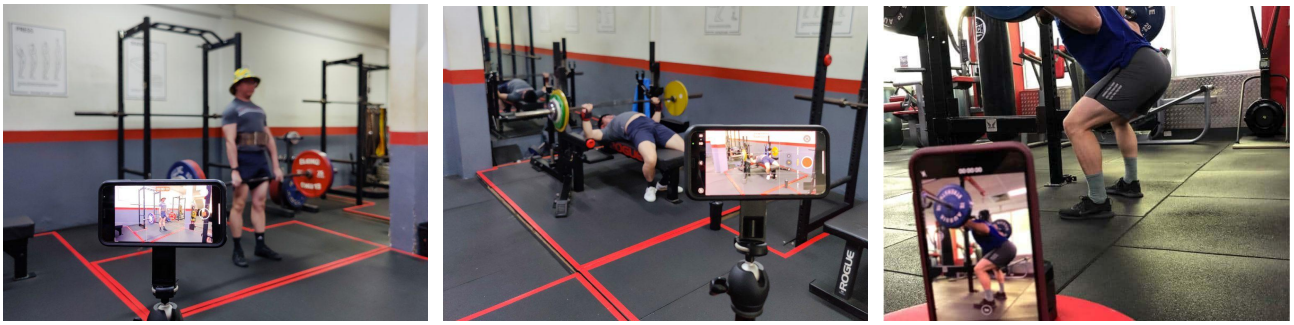
20180896 김재민

20180917 이수빈

Table of Contents

Motivation

During weight training, it is often desired to film yourself during a lifting set, as observing your form while lifting is one of the best ways to identify problems with your lifting form and to improve it. When filming, you often prepare equipment such as a tripod to place your cell phone at a fixed position, then record your sets.



<Filming oneself during free-weight exercises (deadlift, bench press, and squat).>

However, recording with a standard camera application implies a usability issue, as the user has to

1. Approach the cell phone,
2. Press the ‘start recording’ button,
3. Go to the rack and perform the set,
4. Go back to the cell phone,
5. Press the ‘stop recording’ button

For every set one would like to record. If a man, for instance, wishes to record 5 sets of squats, he has to go back and forth between the squat rack and his cell phone 5 times. This movement is unnecessary if we could automate the recording of each set.

Goal of the Application

Assisquat is our sensor-based application to automatically record the user’s exercise sets. The app uses continuous sensor monitoring and ML models to remotely detect the start and end of each set to record the sets.

Usage patterns

The application will be installed on the user's smartphone, which will be held fixed at the position desired by the user. When the front-facing camera detects a person, the smartphone starts recording the set. Once a set is finished, the user claps to stop and save the recording. The user can simply re-enter the rack to start recording a new set.

Models used

1. A Tensorflow object detection model to detect a person in the field of view (the app starts recording upon person detection).
2. A Yamnet audio event classification model to detect the clapping sound (the app stops recording upon clap detection).

Operation example

The application first monitors the front-facing camera input to detect a person, denoted as 'idle mode'. Once a person enters its field of view, the application stops the image detection and enters the 'recording mode'. During recording mode, the application records the video of the person exercising for a set, while monitoring the audio input to detect a clapping sound. Once the set is complete, the user claps to notify the end of the set, in which the app stops the recording and resumes the image detection, waiting for another set.

Development: Fragments

In this project, the same models are used as in the skeleton code. A Yamnet audio classification model detects clapping sounds from 0.975s-long 16kHz audio samples, and a Tensorflow object detection model is used to detect a person from image samples collected by the front-facing camera. Assisquat uses the front-facing camera, such that the user can visually check whether the recording function is running as intended (indicated by the Text views) just before each set.

There are two fragments used in our project:

- `CameraFragment`

  This fragment controls the camera device behavior. We dynamically bind and

unbind three different use cases to the camera. The first one is `Preview`. `Preview` synchronously displays the video captured by the camera device. The second one, `Image Analysis` sends the snapshots to the object detection model to perform object classification. The third use case, `Video Capture`, is newly added for video recording and saving.

- Function `setDetectionOn(on:Boolean)` binds the `Image Analysis` use case to the camera if `on` is true. Otherwise, it unbinds the use case.
- Function `setRecordingOn(on:Boolean)` binds the `Video Capture` use case to the camera if `on==true`. Otherwise, it unbinds the use case.
- The initialization of the recorder object, which is used for video recording, is added to the function `bindCameraUseCases(cameraProvider: ProcessCameraProvider).`
- Function `captureVideo()` sets the format saved in the storage and starts the recording if no recording is in progress. A set of behaviors (finalization event), which includes unbinding `Video Capture`, binding `Image Analysis`, and emitting a beep sound, are performed once the recording finishes. If some recording is already in progress, the function stops the recording.
- Function `onObjectDetectionResults()` defines actions to be done based on the detection results. It adjusts the image analysis frequency and triggers the video recording. We have two different values for the image analysis time interval. The first value, `busyDetectionPeriod`, is a short period temporarily used to quickly confirm that the person is present, just before the recording begins. The second value, `idleDetectionPeriod`, is the longer value, used in idle state to reduce energy consumption during rest periods. Details on the image analysis frequency adjustments are elaborated later.

- `AudioFragment`

  `AudioFragment` is in charge of audio sensing and sound classification. The behaviors triggered by clapping sounds are defined here.

- Function `setAudioInference(on : Boolean)` turns on audio detection if the argument 'on' is true. Otherwise, it turns off the detection.
- Function `onResults(score: Float)` defines the behavior performed when the clapping sound is detected. Once a clap is detected, the video recording is stopped and saved to the gallery.

Development: MainActivity

In `MainActivity`, we added wrappers of some important functions in `AudioFragment` and `CameraFragment`. These functions allow us to call `Fragment` object methods from other `Fragment` classes (e.g. upon clapping detected by `AudioFragment`, call `captureVideo()` defined in `CameraFragment`).

Development: Control Flow

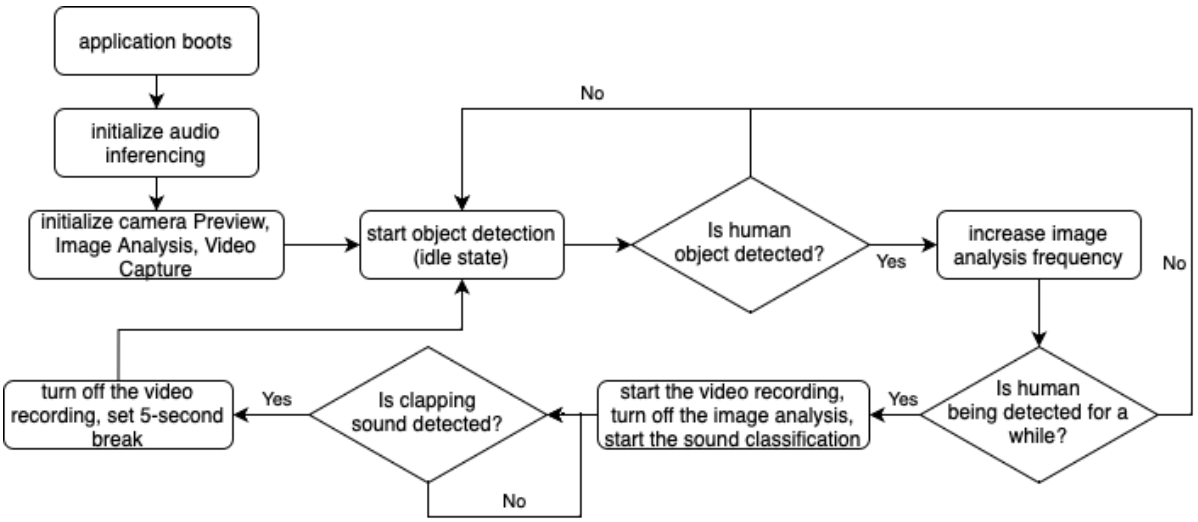
- Hardware Limitation:

We have found that binding `Image Analysis` and `Video Capture` at the same time is allowed only for cameras with `INFO_SUPPORTED_HARDWARE_LEVEL` of `INFO_SUPPORTED_HARDWARE_LEVEL_3`. However, neither Android Studio's virtual devices nor our physical Galaxy S10 device did not support this level. Hence, we dynamically bind / unbind the use cases, such that only one of the two are bound at a time.

- Control Flow:

When the application boots, audio detection is initialized and set to off, and object detection is initialized and turned on. To lower battery consumption, the default frequency of image analysis is set low (`idleDetectionPeriod`). Once a person is detected in a snapshot, the analysis frequency shortly increases. If a human is detected for a while (~0.5s), we first unbind `Image Analysis`, then bind `Video Capture`. We start the video recording by calling the `captureVideo()` function. At the same time, Yamnet starts the sound classification as well. Once the clapping sound is detected, the finalization event of `Video Capture` is triggered. The sound classification is turned off immediately. For usability, we have set a 5-second break after the end of each recording, to avoid detecting the user immediately after

the set. During the break, neither the sound detection nor the image detection should be executed.



<Flowchart representation of Assisquat's control flow.>

Evaluation: Completeness

Our application Assisquat does work as intended. When the application is first opened, it monitors the front-facing camera as input, waiting for the user to enter its view. Once the user enters, the first exercise set begins. The object detection model is deactivated, video recording is turned on, and audio detection model is activated. The app keeps recording, while monitoring the audio input. Once the user claps to denote the end of the set, the app stops recording, saves the video into the gallery, deactivates the audio detection, and reactivates the object detection to record another set.

Assisquat can suffer from excessive power consumption, as it uses two ML-based models to continuously monitor data from two sensors. Naively running these inference models during the entire execution time would quickly drain the smartphone's battery. From our application control flow, we have imposed some mitigations to avoid excessive power usage:

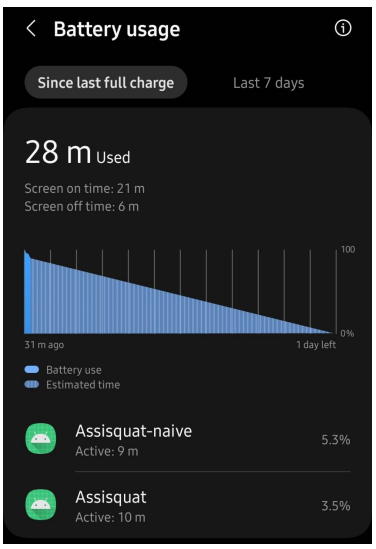
- While the app is waiting for a new set (idle state), the app does not need to monitor the audio input. Since the detection of the user is the only trigger to start recording, the app only needs to monitor the camera input before recording.

- When in idle state, the app continuously runs the object detection model. When the user rests for a few minutes between sets, however, running the model at a high frequency can consume a lot of energy with no useful work. We can keep the frame rate low while the view is empty. Once the user is detected for a frame, we can temporarily raise the detection rate for a better QoS.
- While the app is in recording mode, the app does not need to run the image detection model. Since the clapping sound is the only trigger to stop the recording, the app only needs to infer audio input while recording.

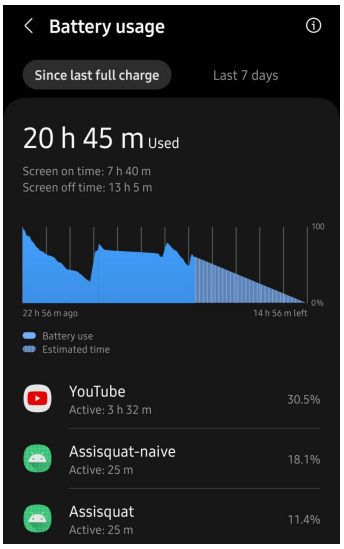
Evaluation: Power Consumption

We have implemented two versions of Assisquat to demonstrate our power optimization. The first version naively runs both ML models during idle state. The object detection model is consistently run at 10ms interval, and audio inference is always turned on. The second version reduces the object detection period to 200ms, and audio detection is turned off when idle, since audio input is required only during the recording.

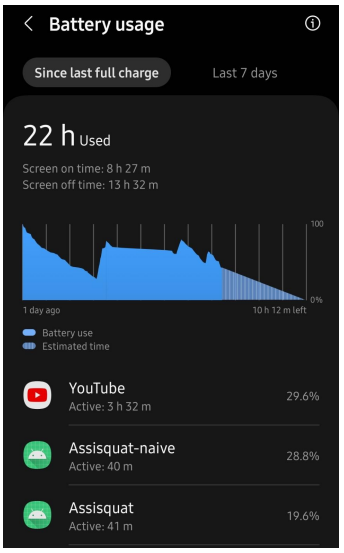
We have compared the idle-state power consumptions between these two versions. We let each version run idle for 10-15 minutes, and checked the battery consumption statistics provided by the Android system application. For accurate results, the smartphone is tested in flight mode (to avoid notifications), and the smartphone is cooled down sufficiently before each test.



<Test 1 (10 mins) >



<Test 2 (15 mins) >



<Test 3 (15 mins)>

The results show that for each test, the naive version of the Assisquat has consumed around 50% or more power, demonstrating that our optimized version is a substantial improvement on idle-state power consumption.

Discussion: Limitations

There exist limitations and potential improvements for Assisquat in multiple aspects:

- In perspective of user:
    - The user must clap at the end of each set to stop the recording. Although better than physically approaching the device after each set, the user still has to perform a slightly unnatural behavior to remotely control the device, so this application is not fully life-immersive.
    - In a crowded gym, when people frequently walk past the scene, unintended recording can occur by triggering the object detection model.
- In perspective of developer:
    - CameraX is limited in binding multiple use cases (`Image Analysis` and `Video Capture`) at the same time. To use camera input in multiple operations, the developer must manually bind / unbind the use cases in a mutually exclusive fashion.
- In perspective of system:
    - Although the object detection frequency is lowered during the idle state for power efficiency, there still exists redundant object detection between sets when the user takes minutes of rest periods. Current implementation mitigates the idle-state battery consumption issue to some degree, but does not fundamentally resolve it.

Discussion: Usefulness and demand for similar applications

Sensor-based monitoring applications can become increasingly more significant as the mobile computing era progresses. For life-immersiveness, an ideal mobile device shall demand less and less deliberate user input, and shall function in an automated fashion based on inputs given implicitly, such as naturally emitted sounds, motions, gestures, etc. If possible, we would want our app to require zero

explicit input throughout its execution cycle; an ideal application would turn on automatically as necessary, perform actions automatically solely based on implicit inputs, then terminate automatically as needed. To automate these steps, continuous monitoring of the surrounding would be necessary.

Continuous monitoring apps would be desired by various groups of users. They could be useful in automated recording apps with repeated patterns, such as Assisquat. They can also be used in security monitoring devices to automatically report or raise security alarms in real-time, when a suspicious scene is observed. Furthermore, this concept can be applied in wearables for real-time health status monitoring of users, even when they are not attached to large, burdensome equipment only available in hospitals. ML models would be a useful tool in this context in general, as it allows the triggering inputs to have complex patterns.

Discussion: System-level support for AI/ML models

Despite our application-level attempt to mitigate excessive power consumption in idle mode, the fixed detection frequency during idle periods still limits the usability. If the idle detection frequency is set too high, the app will suffer excessive energy consumption. However, if the frequency is set too low, once the user re-enters the scene to record another set, seconds of delay could occur for the device to detect the user.

A smarter system-level support that dynamically controls the ML inference frequency can assist application development of this kind. For example, during rest periods, when the scene hardly changes significantly, it is likely to be an overkill to run an entire object detection model again. The majority of inferences shall be done when the scene is significantly changing. Hence, there could be a system-level support that dynamically adjusts the ML inference frequency based on the level of changes in the scene, such that expensive ML inferences are performed to a minimum, while still retaining the reactiveness as soon as the input is first given. With this type of support, developers would be able to easily create more power-efficient apps involving continuous sensing and ML inferences.

Additionally, limitations on the number of camera use cases simultaneously bound prevents us from creating sensing applications with multiple simultaneous subtasks. The Video Capture use case and the Image Analysis use case are some

of the most commonly used functionalities related to camera input. However, even relatively recent android smartphones still cannot simultaneously support image analysis and video recording. For apps involving multiple types of tasks from a single sensor input, it would be desirable to enable binding multiple common use cases at the same time.

API prototype:

```
/*
A prototype function that lazily runs the ML object detection.
Run ML inference only if the new snapshot is significantly different from
the previous snapshots.
*/
fun lazyDetect() {
      // dot product between two image vectors,
      //used as an example similarity metric.
      imageDifference = vector.dot(imageNow, imagePast)
      // snapshots are very similar
      if (imageDifference > differenceThreshold) {
            objectDetector.detect(imageNow)
            increaseFrequency(detectionModel)
      }
      // if two snapshots are very similar, skip ML inference.
      else {
            return
      }
}
```

Conclusion

Developing Assisquat provided an opportunity to observe a platform from an application developer's perspective, enabling us to think of possible system-level support for a more effective application development experience. The project conveys how system design ideas are not motivated by solely studying the system alone, but they come from application development and end-user's demand. System design never operates alone; it is a continuously changing process, requiring both the system knowledge and the insights on the applications served by the system.