ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет информатики, математики и компьютерных наук

Программа подготовки бакалавров по направлению

09.03.04 Программная инженерия

Горелов Евгений Владимирович

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Приложение для управления жестами мультимедиа ПК

A PC Multimedia gesture control application

Рецензент:

Ведущий инженер-разработчик Huawei Technologies,

Казаков Максим Анатольевич

Научный руководитель:

Старший преподаватель кафедры прикладной математики и информатики,

Груздев Алексей Сергеевич

Нижний Новгород, 2021

Содержание

Введение

На заре компьютерных технологий, поиск удобного способа взаимодействия с компьютером был крайне сложной задачей. В самом начале компьютеры представляли собой целые комнаты, полные различной громоздкой электроники. Тогда они были исключительно лабораторной технологией, способом управления которой были единственная заранее написанная программа и огромное количество кнопок и переключателей. Для управления таким компьютером требовались целые команды специалистов.

История современного взаимодействия с компьютером

Позже компьютеры модернизировались и уменьшались в размерах. Когда появилась необходимость набирать текст прямо во время работы компьютера, было создано такое устройство как клавиатура. Клавиатура в своё время избавила специалистов от огромного количества тумблеров для управления, а также необходимости писать программы для этого же компьютера на листке бумаги. Она совместила в себе всё необходимое для управления работой компьютера и сопровождением его времени работы. Программисты отныне могли набирать программы прямо на самом компьютере. А у компьютера в свою очередь отпала необходимость выполнять строго одну единственную заранее записанную программу. Ведь теперь появилась вариативность – изменить поведение компьютера можно было командой с клавиатуры. Ради справедливости стоит сказать, что любая подобная команда – это тоже заранее записанная программа. Однако с помощью таких мини-программ можно уже было серьёзно преобразовывать входные данные.

Позднее, в 1963 году, Дуглас Энгельбарт создал первый прототип всемирно известного устройства – компьютерной мыши. Это устройство

необходимо для управления курсором. Оно особенно удобно при использовании графического интерфейса – пользователь в буквальном смысле видит, куда он производит клик. Компьютерная мышь до сих пор используется при работе с компьютером, а вот в современных ноутбуках её постепенно вытеснила тач-панель. Тач-панель является аналогом компьютерный мыши и выполняет все те же самые функции, однако, перемещение курсора определяется не движением мыши по столу, а движением пальцев по поверхности тач-панели. В большинстве ситуаций тач-панель очень удобна, например её не нужно подключать отдельно – она встроена прямо в ноутбук. Однако в некоторых случаях выигрывает всё-таки мышь – с её помощью можно осуществлять более точные и аккуратные манипуляции – всё-таки она более, так сказать, привычна нашим рукам.

С развитием мобильных устройств и сенсорных экранов, человечество понемногу начало привыкать выполнять почти все действия на смартфонах. И хоть у них и отсутствуют привычные нам клавиатура с мышкой, их роль выполняют пальцы, которые управляют всем смартфоном и выполняют роль мышки, а клавиатура в нужный момент появляется виртуальная прямо на экране.

На сегодняшний день клавиатура и компьютерная мышь являются самыми распространёнными и основными способами взаимодействия с компьютером. Люди уже настолько сильно привыкли к ним, что они стали для нас в буквальном смысле интуитивно понятны.

Однако время не стоит на месте, скорость жизни и скорость нашего развития растут. Компьютеры сегодня являются очень важным звеном в нашей жизни. С их помощью люди могут обмениваться информацией друг с другом находясь на разных концах нашей планеты (и даже не только планеты). Человечеству удалось автоматизировать очень большое количество манипуляций и процессов, которые раньше выполнялись человеком. И сегодня, чем быстрее мы будем взаимодействовать с компьютером, тем

быстрее сможем выполнять как можно больше полезных операций на нём. И в данном случае клавиатура с мышкой хоть и остаются самыми основными способами взаимодействия с компьютерами, но никак не являются быстрыми.

Перспективные решения

Голосовое управление

Для решения этой проблемы сегодня ведутся разработки систем взаимодействия с компьютерами “без посредников”. Чем меньшее количество устройств при этом будет задействовано и чем более естественный способ общения будет для человека, тем быстрее мы будем «общаться» с компьютерами. Из этого очевидно, что можно было бы управлять голосом – это естественно и привычно для человека; кроме того, наш голос не требует дополнительных устройств для его передачи компьютеру (микрофоны не требуют взаимодействия и работают отлично). И действительно, сегодня уже есть хорошие голосовые помощники, например, знакомая на всем Алиса – проект Яндекса, которая очень хорошо себя зарекомендовала. Вот только Алиса пока что не может полноценно управлять компьютером. Кроме того, у голосовых помощников есть несколько очень серьёзных проблем:

1. Язык. Языков у человечества очень много, и для каждого из них приходится разрабатывать свою модель распознавателя. А ведь помимо самих языков существуют ещё их диалекты и варианты произношения, что тоже создаёт проблемы при разработке.
2. Неидеальная техника, шумы вокруг неё, разные голоса у разных людей – всё это создаёт большие технические трудности для распознавания речи в принципе, не говоря уже об управлении таким сложным механизмом как компьютер.

Однако, несмотря на все эти проблемы, разработчики с каждым днём совершенствуют модели распознавателей и с каждым днём становятся всё ближе к голосовому управлению.

Управление жестами

К счастью, голос – это не единственное, что природа подарила человеку для коммуникации. Раз у нас пока что плохо получается извлекать из звуковой волны с приличным количеством шума человеческую речь, то почему бы не использовать то, что нам уже удалось очень хорошо отфильтровать – картинки. Все свои глобальные грандиозные изобретения человечество подсмотрело у природы. Самолёт – это птица, подводная лодка – рыба. Воспользуемся этим преимуществом и сейчас – если человек по каким-то причинам не может разговаривать, то он пользуется жестами. Более того, люди так или иначе используют язык тела и жесты даже при общении голосом, чтобы подчеркнуть свою речь. Благодаря этому, жесты интуитивно понятно и просты для людей. А что насчёт управления компьютером с помощью жестов? Действительно, если мы смогли научить компьютер выделять определённые предметы на картинках, то почему бы не научить его же искать там именно жесты рук? Однако не всё так просто, как кажется на первый взгляд. Обо всех моментах и тонкостях распознавания жестов далее и пойдёт речь.

Цели и задачи этой работы

Целью данной работы является поиск подходящей модели (или архитектуры модели) распознавателя жестов, а также использование её в разрабатываемом приложении для управления мультимедиа ПК.

Для достижения цели необходимо выполнить ряд задач:

- Провести анализ готовых публичных моделей распознавателей жестов и выбрать из них самый подходящий под управление мультимедиа ПК;
- Определить наиболее подходящую платформу и инструменты разработки приложения;
- Разработать приложение для управления мультимедиа ПК;
- Интегрировать в приложение распознаватель жестов;
- Провести анализ проделанной работы, протестировать приложение.

Практическая ценность данной работы определяется повсеместным распространением технологии управления жестами в других сферах. Жесты являются одним из наиболее удобных и интуитивно понятных способов управления устройствами.

Кроме того, некоторые особенности приложения помогут заинтересовать пользователей и облегчить им период перехода с одного способа управления на другой:

- Автозагрузка вместе с ОС;
- Возможность быстро приостановить распознавание жестов;
- Незаметность приложения для пользователя.

Теоретическая часть

Итак, на входе мы имеем изображение с жестом, например полученное с веб-камеры компьютера, а на выходе должны увидеть реакцию компьютера на этот жест. Давайте подробнее разберём весь путь картинки. В этом разделе я опущу такие моменты, как захват изображения и имплементация реакции на жест – они будут рассмотрены в практической части. Сейчас же я сконцентрируюсь именно на обработке самого изображения для получения информации о показанном (или не показанном) жесте.

Машинное обучение

Определение одного из заранее известных жестов на картинке – это классическая задача классификации. К сожалению, выделить жест на картинке напрямую мы не можем. Но мы можем попытаться использовать методы машинного обучения для решения этой задачи. Машинное обучение не стремиться решать задачи напрямую, оно использует множество «готовых» решений сходных задач. Это чем-то похоже на обучение человека, например, решать уравнения. Изначально человек этого не может, но его учат их решать, показывая исходное уравнение и способ его решения. Раз за разом человек обучается находить правильное решение. Похожим образом работает и машинное обучение. Используя средства математики, компьютер раз за разом прогоняет через себя множество уже решённых задач (в нашем случае картинок с жестами) и запоминает для них правильные ответы. Когда же он сталкивается с ещё нерешённым примером, он уже может попытаться найти схожие признаки с предыдущими примерами и дать подходящий ответ. Отсюда и происходит название «*машинное обучение*».

Задачи машинного обучения

Машинное обучение призвано решать две основные задачи:

- Классификацию;
- Регрессию.

В задачи классификации входит определить принадлежность исходных данных к одному из предопределённых классов. Самый типичный пример – «Да/Нет». Допустим мы хотим определять есть ли на картинке яблоко. В таком случае машинное обучение присваивало каждой картинке класс «Да» или класс «Нет». В данном случае количество классов 2, поэтому классификация называется *бинарной*. Но в других задачах классов может быть больше.

Регрессия же заметно отличается от вышеупомянутой классификации. Она не разделяет все данные на классы, а каждым входным данным присваивает какое-то вещественное значение. Типичный пример – прогноз погоды, температура воздуха. В данном случае нет огромного количества классов температуры, скажем, от -40 °C до +40 °C. Модель сама выдаёт вещественное число, которое считает подходящим.

Методы машинного обучения

На заре машинного обучения, люди пытались обработать входящие данные классическими эмпирическими алгоритмами. Например, один из самых простых «*KNN – K nearest neighbors*» («*K ближайших соседей*»). Суть этого алгоритма проста – он запоминает тренировочные данные и при попытке решить задачу опирается на *K* самых похожих данных.

Давайте в качестве примера рассмотрим одну из самых популярных задач в машинном обучении – «Титаник». Суть задачи: нам известен список из 1309 пассажиров Титаника. Для всех этих пассажиров известны некоторые детали: пол, класс, номер билета, каюта проживания и т. д. У некоторых

пассажиров некоторые данные могут быть пропущены. Кроме того, для 891 пассажира известно, выжил ли он/она во время катастрофы или нет. Необходимо по имеющимся данным предсказать эту же характеристику для оставшихся 418 пассажиров.

KNN в данном случае будет искать *K* максимально похожих по характеристикам пассажиров и выдавать наиболее часто встречающуюся среди них метку «выжил/не выжил». Точность такого подхода не всегда будет высокой, но его плюс в том, что он очень простой.

Существуют и другие алгоритмы, так называемого, «*классического машинного обучения*». Всех их объединяют общие плюсы и минусы. Из плюсов стоит отметить:

- Относительную простоту алгоритмов;
- Относительно высокую скорость выполнения при правильном подборе алгоритма и качественной реализации

Однако есть и минусы:

- Жёсткая связь с тренировочными данными. Повышение “сложности” данных может оказаться непосильной задачей для таких алгоритмов.

Обычно такие алгоритмы используются в тех случаях, когда заранее известно, что реальные задачи не будут сильно отличаться по сложности от тех, на которых мы учились. Задача с «Титаником» – как раз этот случай. Но у нас на входе картинки, которые из раза в раз могут сильно меняться. На одних будет чистый фон и хорошее освещение, а другие будут засвечены или наоборот слишком тёмными.

Deep learning (глубокое обучение)

В отличие от классического машинного обучения, глубокое обучение использует нейронные сети для решения задач. *Нейронная сеть* – это сложный механизм обработки входящих данных, суть которого заключается в обработке данных каждым узлом (нейроном) последовательно и передачи своих результатов следующим. Идея нейронной сети тоже была “подсмотрена” у природы, ведь именно так работает наш мозг. С одним отличием – вместо химических реакций внутри нейронов, здесь каждый нейрон выполняет свою математическую функцию. Кроме того, каждый нейрон имеет определённый вес – вклад, в результат. Именно благодаря тонкой корректировке этих весов и происходит тренировка нейронных сетей.

Однако, чем нейросеть лучше классических алгоритмов? Основываясь на исследованиях G. R. S. Murthy, R. s. Jadon, J. Krüger, T. Lien и A. Verl, описанных в их научных публикациях «Hand gesture recognition using neural networks» [1] и «Cooperation of human and machines in assembly lines» [2], я сделал несколько выводов:

1. Благодаря своей структуре, нейросети не ищут заранее предопределённых нами признаков в данных. Во время тренировки они стараются выделить наиболее общие признаки из всех полученных данных, а потом ищут их же в тестовых примерах.
2. Благодаря всё той же структуре, нейросеть может расходовать значительно больше вычислительных ресурсов компьютера, чем классические алгоритмы.

В сухом остатке мы получаем, что в нашем случае глубокое машинное обучение с нейросетями будут лучше справляться с задачей классификации изображений. Однако нам необходимо нивелировать все возможные затраты вычислительных мощностей и времени.

Препроцессинг

Как уже было сказано выше, нейросети – это сложные механизмы, которые могу затрачивать значительное количество вычислительных ресурсов компьютера и времени. В таком случае остаётся только попробовать каким-то образом подготовить наши данные, чтобы нейросети было легче на них обучаться и легче потом искать признаки.

Согласно исследованиям Hogyu L. и Lihui W. в статье «Gesture recognition for interacting with AI: from theory to latest achievements» [3], а также Viola P. и Jones M. в статье «Robust real-time object detection» [4], для лучшего распознавания руки на картинке желательно выделять её контуры. С практической точки зрения это достигается с помощью применения небольшого гауссова размытия и увеличения резкости после. Однако стоит отметить, что в нашем случае эффекты применяются ко всей картинке, а значит и контуры будут выделены у всех объектов.

Также для увеличения скорости тренировки и скорости работы нейросети авторы рекомендуют сжимать изображения до средне-низкого разрешения. Количество всевозможных деталей на изображении будет только мешать нейросети найти руку, поэтому качеством можно пренебречь в угоду производительности. Кроме того, низкое разрешение означает меньшее количество пикселов и как следствие меньшее количество данных для обработки нейросетью, что также увеличивает её производительность.

Ещё одним методом оптимизации может послужить правильный выбор цветовой схемы. Например, если конвертировать стандартное RGB изображение в чёрно-белый формат, мы сэкономим значительное количество времени и памяти на обработку изображения. Однако при конвертации в другие палитры, мы можем добиться более яркого выделения светлых объектов. Авторы вышеупомянутых статей не дают однозначного ответа на этот вопрос. В данном случае требуется проведение дополнительных тестов.

От себя же могу ещё добавить, что если нам удастся сильно повысить производительность нейросети за счёт чего-то другого, то можно будет вернуться к цветному изображению, чтобы конвертировать нашу выигранную производительность в точность распознавания.

Структура нейросети и нейроны

Нейросеть обычно состоит из множества слоёв. Есть два обязательных – вход (input layer) и выход (output layer). На первый подаются исходные данные после препроцессинга, а выходной слой выдаёт нам ответ – результат всей математической работы нейросети. Кроме того, между ними может быть ещё неограниченное число скрытых слоёв. Количество этих слоёв напрямую влияет на производительность нейросети. А вот на точность её ответов будут в первую очередь влиять тип каждого слоя и их порядок следования.

Самый простой для понимания тип слоя – полносвязный слой. В этом типе все нейроны связаны со всеми нейронами следующего слоя. Чаще всего используется как входной и выходной слои нейросети, реже как скрытый слой.

Свёрточные слои (свёртки) используются для уменьшения размерности данных и перехода от конкретных особенностей к абстракциям. Суть её в том, что ядро свёртки (матрица меньшего размера, чем входные данные) поэлементно умножается на входную матрицу и суммируется, а результат записывается в выходную. Весь секрет кроется в том, что так как ядро свёртки меньше исходной матрицы, то после умножения мы получаем не набор чисел, а только одно, которое и записывается в соответствующую позицию выходной матрицы. Таким образом выход получается с меньшей размерностью, чем вход. Такие слои как раз используют при обработке изображений, чтобы, во-первых, сильно увеличить производительность, а во-вторых, избавиться от деталей на изображении. Например, Tompson J., Stein M., Lecun Y., Perlin K. в своей работе “Real-time continues pose recovery of human hands using

convolutional networks” [5] используют в нейросети для фиксирования рук свёрточные слои.

Помимо вышеуказанных слоёв существуют ещё рекуррентные слои. В сетях с такими слоями, вычисления могут повторяться несколько раз, благодаря связям рекуррентных слоёв с предыдущими слоями.

Практическая часть

Модель распознавателя

Для поиска подходящей готовой свободной модели я обратился к ресурсу «GitHub» [14]. К сожалению, подходящих моделей для анализа нашлось не так много. Во-первых, одно из требований – это год разработки не раньше 2015. Это обусловлено тем, что за последние пять лет, прогресс в области машинного обучения многократно вырос. Кроме того, ранние модели могут быть попросту не эффективными с большой вероятностью. Во-вторых, модель должна быть открытой, с доступным исходным кодом. В-третьих, приложение, которое написал автор модели должно запускаться, чтобы была возможность проверить модель. Ещё желательно, чтобы у модели был доступен датасет, на котором она училась, чтобы проанализировать его.

Исходя из этих критериев я нашёл всё несколько репозиториев, которые запускались. Часть из них пришлось немного исправить, чтобы запустить.

Вот список подошедших репозиториев, в дальнейшем я буду упоминать их по порядковому номеру в этом списке:

1. anasmorahhib/3D-CNN-Gesture-recognition [15]
2. Gogul09/gesture-recognition [16]
3. SparshaSaha/Hand-Gesture-Recognition-Using-Background-Ellimination-and-Convolution-Neural-Network [17]
4. Lucamoroz/StaticGestureRecognition [18]

Каждую модель из данного списка я протестировал вручную, показывая жесты в разной последовательности, и записывал, правильно ли мне угадывает модель жест.

Модель №1 распознаёт 3 жеста: Thumb up, Swipe left, Slide fingers. Я провёл эксперимент, в котором показал модели по 20 жестов в случайном порядке. В таблице №1 видно, что модель довольно плохо распознаёт жест Thumb up, а два остальных примерно одинаково. В любом случае, по итогам эксперимента модель показала себя очень плохо, средний процент угадываний за весь эксперимент ≈63%. Это отвратительный показатель для модели. По своей структуре модель является нейронной сетью, классической свёрточной нейронной сетью с чередующимися слоями «Conv3D» и «MaxPool3D». А из препроцессинга только сжатие изображения до 64*64 пикселов.

Модель №2 определяет количество показываемых ей пальцев. Этой модели я показал по 10 комбинаций в случайном порядке. По результатам эксперимента (таблица №2) видно, что модель относительно хорошо справляется только с одним и двумя пальцами, на большем количестве, модель распознаёт их очень плохо. По структуре эта модель представляет собой классическую обработку изображения для выделения контуров ладони. Никакого машинного обучения тут не применяется. Однако здесь есть примеры применения эффектов к изображению, например гауссово размытия в купе с повышением резкости усиливает контуры объектов.

Модель №3 определяет 3 жеста: Palm, Swing, Fist. Этой модели я показал по 10 жестов в случайном порядке. Опираясь на результаты в таблице №3 можно сделать вывод, что эта модель достаточно уверенно распознаёт заданные жесты, однако, она всё ещё далека от идеала – средний процент успехов равен 80%. Кроме того, я заметил, что модель плохо распознаёт жесты левой руки – видимо тренировка происходила преимущественно на правой. По своей структуре эта модель является чистой свёрточной нейросетью с 7 скрытыми свёрточными слоями и 1 полносвязным слоем на выходе.

Модель №4 определяет 6 жестов: Fist, Palm, Pointer, Spok, Thumb up, Thumb down. Я провёл эксперимент, показывая по 10 жестов в случайной последовательности. Модель отлично себя проявила, показав средний процент

успехов 90%. Все результаты сохранены в таблице №4. Однако стоит заметить, что модель очень плохо распознаёт жест Pointer – всего 4/10 успехов. Кроме того, я заметил, что жест распознаётся как Palm только с сомкнутыми пальцами, иначе модель сочтёт ладонь как Spok. По своей структуре представляет собой нейронную сеть *MobileNetV2* с добавленными слоями *GlobalAveragePooling2D* и дропаутом с коэффициентом 0.2. В качестве препроцессинга здесь используется перевод цветового формата и сжатие изображения методом билинейной интерполяции.

Исходя из полученных данных я остановился на модели №4.

Выбор платформы и средств имплементации

Перед началом реализации приложения необходимо определиться с конечной платформой, на которой будет использоваться это приложение. На персональных компьютерах и ноутбуках для домашнего пользования сегодня существует всего одна платформа – x86-64, и 3 самых распространённых и основных операционных системы:

1. Microsoft Windows®;
2. Apple OS X®;
3. Семейство операционных систем Linux®:
    a. Debian дистрибутивы;
    b. RPM дистрибутивы.

Microsoft Windows® – одна из самых распространённых операционных систем во всё мире по данным Operating System market share worldwide [6]. Её доля рынка среди всех доступных ОС на сегодня 32% устройств – это в 5 раз больше, чем *Apple OS X*, и 30 раз больше, чем *Linux*. Ради справедливости подчеркну, что эта статистика касается только устройств личного пользования. В пользу распространённости, я выбрал *Microsoft Windows*®.

Выбор языка программирования

Как я уже говорил в теоретической части, для подобного приложения крайне важна производительность, особенно, когда есть заранее известный риск её потери на этапе работы нейросети. Исходя из такого требования, выбор языка программирования более чем очевиден. Язык «Python» удобен для проектирования и тренировки сети, потому что на этом этапе у нас нет необходимости задействовать вычислительные мощности локальной машины – проще и грамотней использовать облачные вычисления с помощью, например, «Google Colaboratory». А скорость написания кода на этом языке крайне высокая. Кроме того, «Python» имеет отличные модули, с помощью которых можно сильно сократить время разработки.

Однако, конечный пользователь не будет запускать нашу программу на облаке. Ему необходимо именно локальное решение. Учитывая сравнительно невысокую вычислительную мощность домашних компьютеров, язык «Python» перестаёт быть пригодным для загрузки и использования готовой (именно готовой) модели. Для ускорения вычислений и оптимизации по памяти я выбрал язык «C++». Именно на нём нужно производить загрузку и использование модели. Однако для упрощения и ускорения разработки небольшого графического интерфейса пользователя, я не использовал стандартные средства «C++» и классический интерфейс «WinAPI». Вместо этого я использовал «.NET Framework» на среде «Common Language Runtime» (CLR). Исходный код части графического интерфейса по-прежнему написан на C++, однако теперь он имеет автоматическое управление и сборку мусора. Кроме того «.NET Framework» имеет очень удобный интерфейс работы с GUI. Обратите внимание, что среда CLR по-прежнему позволяет пользоваться всеми возможностями «C++» по ручному контролю памяти. Благодаря этому, часть работы с моделью работает быстро.

Использование сторонних библиотек и фреймворков

В первую очередь у меня возник вопрос о захвате изображения с камеры и последующей предварительной обработке этого изображения. Я выбрал «OpenCV» [11], так как он очень распространён в сфере машинного обучения и имеет весь необходимый мне функционал по получению и обработке изображения. Для «C++» он поставляется в виде библиотеки динамической компоновки.

Следом возник вопрос о том, как загрузить готовую модель. Проблема в том, что столь популярные фреймворки как «Tensorflow», «Pytorch», «Keras» имплементированы для языка «Python». Большинство существующих решение представляет собой компиляцию модуля «Python» для использования на «C++». Но кроме вышеуказанных причин невозможности такого метода, сюда добавляется ещё и необходимая зависимость на компьютере пользователя – «Python». Но мне всё-таки удалось найти фреймворк «Frugally-Deep» [7], который способен загружать и использовать готовые модель нейросетей. Правда, как оказалось с несколькими ограничениями.

Во-первых, использование и код данного модуля пришлось обернуть в библиотеку динамической компоновки, потому что один из используемых им файлов стандартной библиотеки «C++» по какой-то причине не компилировался напрямую. Фреймворк поставляется как чистый исходный код, полностью записанный в заголовочных файлах «C++», а все дополнительные зависимости («Eigen» [8], «Functional Plus» [9], «nlohmann JSON» [10]) также поставляются в виде чистого исходного кода.

Во-вторых, данный фреймворк не работает напрямую с моделями формата «hdf5», но при этом он имеет в себе программу, написанную на «Python» для конвертации модели «hdf5» в объект «JSON».

Процесс распознавания и реакции на жесты

Захват изображения

Как я уже сказал ранее, для захвата и обработки изображения с веб-камеры я использовал фреймворк «OpenCV» [11]. Для того чтобы получить изображение с веб-камеры, её необходимо сначала “захватить” и проинициализировать. Для захвата веб-камеры во фреймворке существует класс «cv::VideoCapture». Захватить веб-камеру можно указав её дескриптор сразу в конструкторе. Кроме того, при передаче значения «0», фреймворк попытается захватить веб-камеру, которая обозначена в операционной системе как по умолчанию. Инициализация веб-камеры происходит автоматически в асинхронном режиме. Проверить доступность веб-камеры можно с помощью метода «bool cv::VideoCapture::isOpened() const». Данный метод вернёт значение «true», если камера была успешно захвачена нашим процессом.

Теперь мы можем получать кадр с веб-камеры по запросу. В данном классе перегружен оператор «>>», что позволяет работать с ним как со стандартным потоком ввода «C++». Для получения кадра необходимо объявить переменную типа «cv::Mat». Данный класс представляет ни что иное как многомерный динамический массив. При передаче его в перегруженный оператор, в этот массив запишется изображение в самом примитивном его виде – трёхмерный массив пикселов. Для освобождения веб-камеры достаточно вызвать метод «void cv::VideoCapture::release()».

В данном моменте я столкнулся с одной проблемой. По задумке нам не нужно получать кадры с веб-камеры постоянно с высоким FPS, достаточно опрашивать её приблизительно раз в 500 мс. В таком случае можно было бы освобождать веб-камеру для других приложений в это время. Нам достаточно было бы захватить веб-камеру раз в 500 мс, получить с неё кадр и освободить

её. А уже после заниматься распознаванием жестов на этом кадре. Однако, для того чтобы камера смогла захватывать корректное изображение, ей необходимо небольшое количество времени для инициализации, а так как она проходит асинхронно, то нет никакой возможности отследить завершение инициализации, кроме как вручную посмотреть на полученное изображение. Такой вариант для приложения не подходит.

Кроме того, в следствие постоянного захвата и освобождения веб-камеры, индикатор её работы постоянно мигает, что, как я посчитал, будет раздражать конечного пользователя.

Из-за этих двух факторов, мне пришлось отказаться от этой идеи и захватывать камеру на всё время распознавания. Однако у приложения есть функция приостановки распознавания, в момент которой камера освобождается, но об этом чуть позже.

Кроме того, всю работу с веб-камерой я вынес в отдельный поток, чтобы не блокировать его во время распознавания.

Препроцессинг захваченного изображения

Проанализировав исходный код программы тренировки и тестирования модели на «Python», я выяснил, что автор модели переводит изображение, полученное с веб-камеры, из цветового формата BGR в цветовой формат RGB, сжимает изображение до размера 224*224 пикселов методом билинейной интерполяции, а также трансформирует все значения компонентов цветов из диапазона [0, 255] в диапазон [-1, 1].

Большинство современных веб-камер отдают на выход изображение уже в формате RGB, поэтому преобразование приводит только лишь к увеличению контрастности изображения, что выделяет немного руку на фоне других объектов. Для выполнения изменения цветового формата я использовал функцию «void cv::cvtColor()».

Сжатие изображения необходимо проводить, чтобы увеличить производительность модели, а также чтобы избавиться от мелких лишних деталей на изображение, которые могут помешать модели правильно распознать жест. Для выполнения сжатия я использовал функцию «void cv::resize()».

После этого я преобразовал полученное изображение в тензор – линейный массив данных, который передаётся в нейросеть. Примечателен этот момент тем, что функция преобразования картинки в тензор позволяет ещё и трансформировать диапазоны значений компонентов цветов. Я предполагаю, что разработчики фреймворка «Frugally-Deep» [7] предусмотрели эту возможность, потому что она довольно сильно распространена. Это преобразование несёт важную функцию – ускорение вычислений. В силу архитектурных особенностей процессору легче выполнять базовые математические операции с вещественными числами в диапазоне близком к нулю, чем с целыми числами на границах байтового диапазона. В итоге всё преобразование выполнилось одной функцией «fdeep::tensor fdeep::tensor_from_bytes()».

Распознавание жеста с изображения

Для использования модели и распознавания жестов я использовал класс «fdeep/model». Для загрузки модели в память компьютера достаточно вызвать функцию «fdeep::model fdeep::load_model()», в которую передать путь к файлу модели на накопителе данных. После этого для распознавания жестов на картинке нам достаточно у объекта «fdeep::model» вызвать метод «fdeep::tensors fdeep::model::predict() const», в который передать массив тензоров для проведения операций распознавания. Данный метод возвращает нам тоже массив тензоров, в котором каждый выходной тензор соответствует каждому нашему входному тензору.

Модель построена таким образом, что наш выходной тензор будет содержать вероятности присутствия каждого из возможных жестов на изображении. Максимальная вероятность в этом тензоре – это и есть распознанный нейросетью жест. Но это ещё не всё. В случае отсутствия каких-либо жестов на изображении, вероятности не будут нулевыми. Поэтому нам необходимо найденный максимум из вероятностей сравнить с минимально заданной вероятностью, при которой мы можем с уверенностью сказать, что жест на кадре присутствует. Экспериментальным путём, наблюдая за ответами нейросети я выяснил, что данное минимальное значение должно быть приблизительно равно 0.9995, то есть 99.95%. Именно с таким ответом нейросети я получал правильно распознанные жесты.

Реакция на жесты

Одним из самых простых способов управления системой мультимедиа является имитация нажатия мультимедийных клавиш на клавиатуре. Изначально я планировал написать небольшой драйвер клавиатуры, который должен был принимать сигналы от этого приложения и посылать соответствующие сигналы системе. Но позже я вспомнил, что при разработке драйвера клавиатуры для собственной ОС в рамках курсовой работы третьего курса, я сохранял все нажатые клавиши в специальную очередь, из которой они по планам должны были последовательно передаваться компонентам ОС. Эта схема работы клавиатуры существует уже очень долго, и мне удалось выяснить, что «WinAPI» позволяет поместить в конец очереди событий клавиатуры *Windows* собственное событие. Использовав функцию «UINT SendInput()» я посылаю нужные события клавиш в очередь.

Кроме того, среди всех жестов есть два, которые я отнёс группе «продолжительных». Это жесты «Thumb up» и «Thumb down». Так как задержка между распознаваниями составляет 500 мс, то для жестов управления громкостью это слишком мало. Исходя из этого я, во-первых,

снизил задержку между повторами конкретно этих жестов; а во-вторых, реализовал постепенное увеличение скорости изменения громкости. Суть в том, что чем дольше мы показываем жест, тем быстрее начинает изменяться громкость. Например, мы уже 5 секунд держим жест «Thumb up». В таком случае громкость будет изменяться не на 2% каждый раз, а уже на 4. Ещё через 5 секунд на 8%, и так далее.

Графический интерфейс

В рамках данного модуля была реализована иконка в системном трее. Она появляется сразу после запуска приложения и носит функцию контроля его работы. С помощью контекстного меню иконки можно добавлять/удалять приложение из автозагрузки *Windows*, приостанавливать/возобновлять работу распознавателя, освобождая при это веб-камеру, и полностью выключать приложение.

Итоговая структура и схема работы приложения

В итоге всё приложение состоит из отдельных модулей:

- Графический интерфейс
- Модуль распознавания жестов
- Модуль захвата изображений

Итоговая схема работы приложения изображена на приложении №1. Внимания заслуживает момент с двумя проверками паузы во время распознавания. Это нужно для того, чтобы исключить возможность наступления пуазы распознавания во время самого распознавания.

Анализ и тестирование приложения

После завершения разработки приложение необходимо протестировать и проанализировать его производительность. С тестами справились все модули, кроме одного – после автозагрузки приложение завершалось с ошибкой. Как выяснилось, при автозагрузке рабочая директория приложения не соответствует директории, в которой находится исполняемый файл, из-за чего приложение не могло найти файл с моделью распознавателя. Чтобы решить эту проблему, я запросил у ОС полный путь к исполняемому файлу приложения и заменил в нём имя исполняемого файла на имя файла с моделью распознавателя. После чего режим автозагрузки заработал в штатном режиме. Других ошибок приложения я не обнаружил. Однако заметил интересную особенность, не связанную с самим приложением. Дело в том, что *Windows* отсылает сигнал «Play/Pause» клавиши активному в данный момент проигрывателю. При тестах приложение «AIMP» реагировало на клавишу без исключений, а вот проигрыватель «YouTube», запущенный в «Яндекс.Браузере» не всегда срабатывал. При тех же условиях другие браузеры отрабатывали событие без ошибок.

Что же касается производительности приложения, то по результатам *Монитора ресурсов Windows* приложение загружает мой процессор (Intel Core i5 3550 3.30 GHz) на 11% и использует в пике около 100 МиБ оперативной памяти в режиме распознавания жестов (приложение №7). В режиме паузы приложение не нагружает процессор вообще, благодаря предусмотренным “усыплениям” рабочих потоков приложения, а расход памяти снижается до приблизительно 50 МиБ (приложение №8). Я считаю это хорошими показателями для такого приложения. Конечно же, можно оптимизировать приложение и дальше, например уйти от «.NET Framework», чтобы снизить расход памяти и процессора, но уже сейчас приложение более чем готово к работе с конечными пользователями.

Заключение

Целью данной работы были поиск и анализ готовых распознавателей жестов и разработка приложения для управления жестами мультимедиа ПК на основе найденного распознавателя. В процессе работы был также проведён поиск подходящих инструментов для реализации заданной цели. В процессе тестирования приложения ошибок выявлено не было.

Результатом всей проведённой работы стали приложение для управления жестами мультимедиа ПК, которое полностью готово к работе в реальных условиях, а также выполненный анализ моделей, архитектур и инструментов разработки.

Однако, функционал такого приложения можно расширять, к примеру можно добавить в следующих реализациях:

- Новую модель распознавателя с большим количеством жестов;
- Возможность не блокировать веб-камеру во время распознавания;
- Возможность пере привязывать жесты к определённым действиям;
- Возможность добавления новых действий в качестве реакций на жесты

Список литературы и используемых ресурсов

[1] Murthy G. R. S., Jadon R. s. «Hand gesture recognition using neural networks»

[2] Krüger J., Lien T., Verl A. «Cooperation of human and machines in assembly lines» CIPR Annals-Manufacturing Technology

[3] Hogyi L., Lihui W. «Gesture recognition for interacting with AI: from theory to latest achievements»

[4] Viola P., Jones M. «Robust real-time object detection» International Journal of Computer Vision

[5] Tompson J., Stein M., Lecun Y., Perlin K. «Real-time continues pose recovery of human hands using convolutional networks»

[6] Operating System market share worldwide – https://gs.statcounter.com/os-market-share

[7] Frugally-Deep – https://github.com/Dobiasd/frugally-deep

[8] Eigen – https://eigen.tuxfamily.org/index.php?title=Main_Page

[9] Functional Plus – https://github.com/Dobiasd/FunctionalPlus

[10] nlohmann JSON – https://github.com/nlohmann/json

[11] OpenCV – https://opencv.org/

[12] GestureControl on github – https://github.com/yazimut/GestureControl

[13] GestureControl on github, images – https://github.com/yazimut/GestureControl/tree/master/img

[14] GitHub – https://github.com/

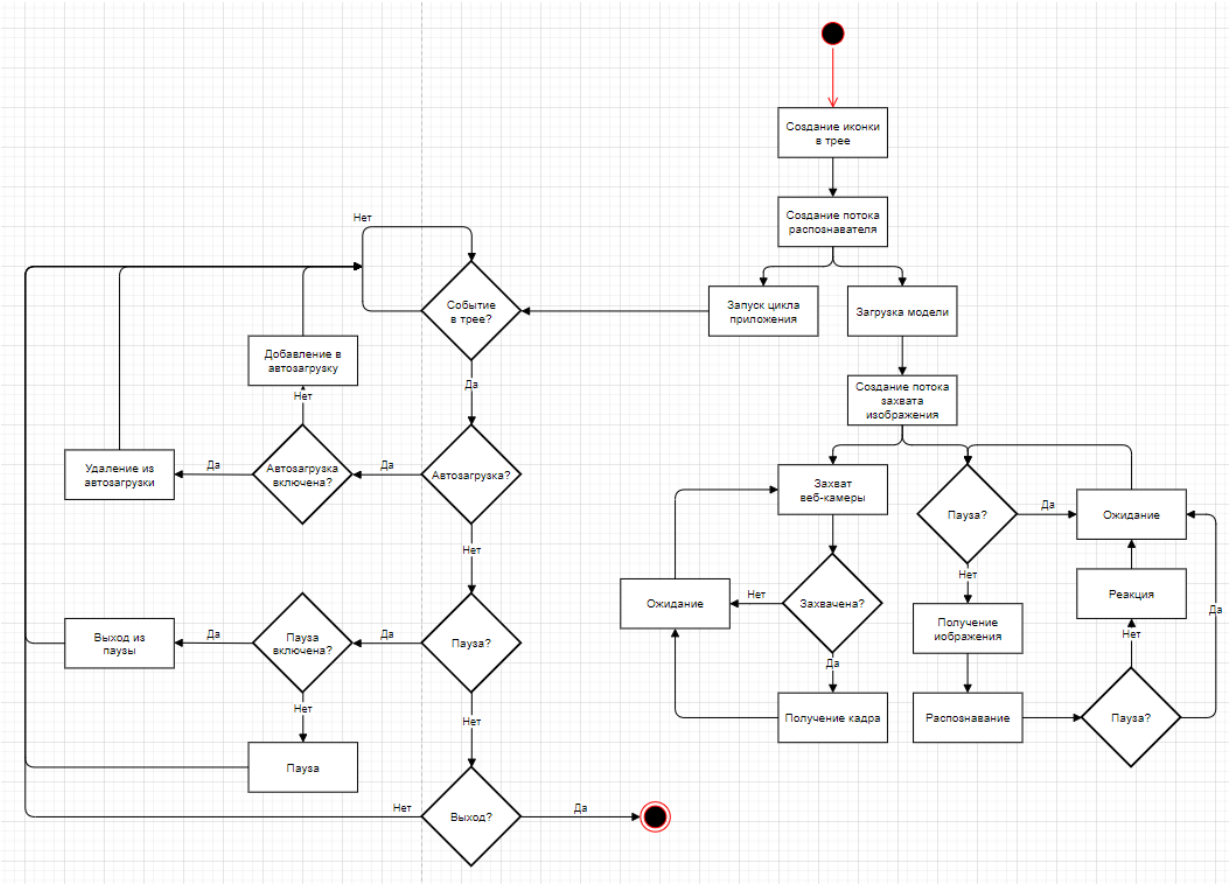
[15] anasmorahhil/3D-CNN-gesture-recognition – https://github.com/anasmorahhib/3D-CNN-Gesture-recognition

[16] Gogul09/gesture-recognition – https://github.com/Gogul09/gesture-recognition

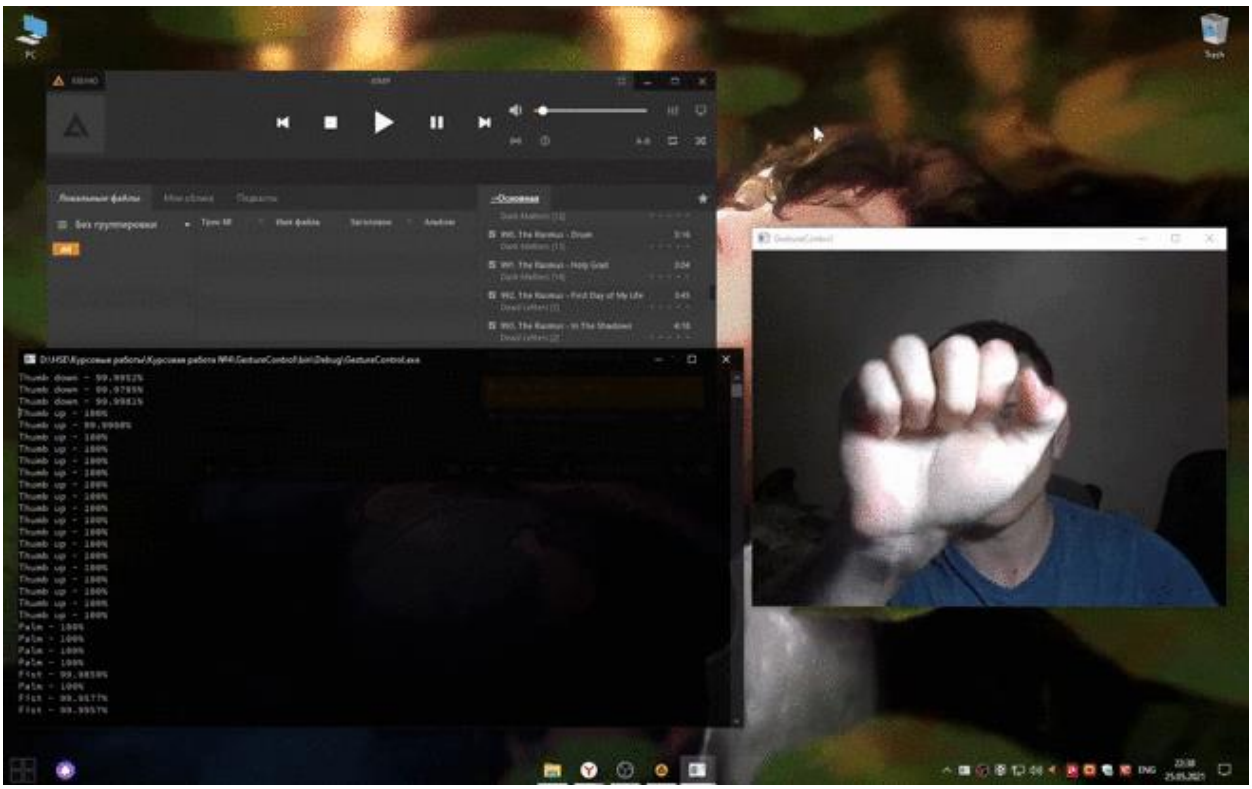
[17] SparshaSaha/Hand-Gesture-Recognition-Using-Ellimination-and-Convolutional-Neural-Network – https://github.com/SparshaSaha/Hand-Gesture-Recognition-Using-Background-Elllimination-and-Convolution-Neural-Network

[18] lucamoroz/StaticGestureRecognition – https://github.com/lucamoroz/StaticGestureRecognition

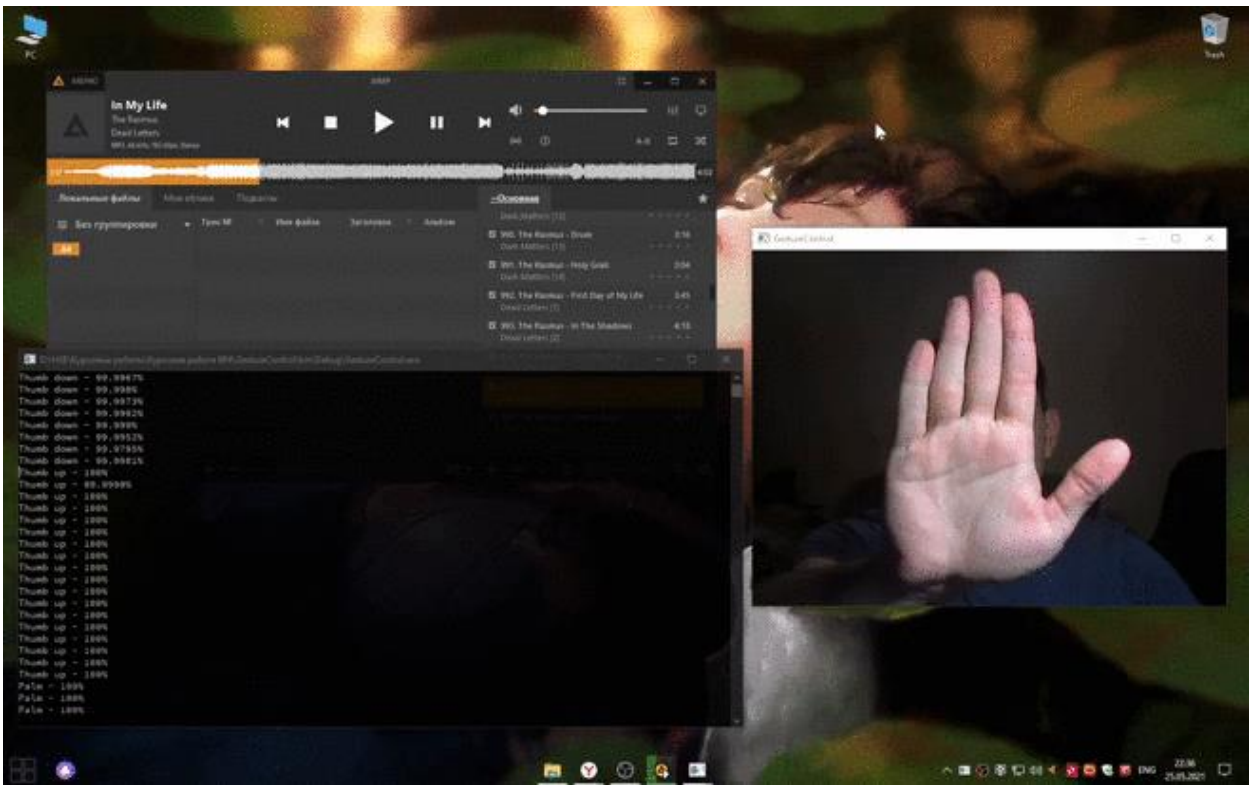
Приложения



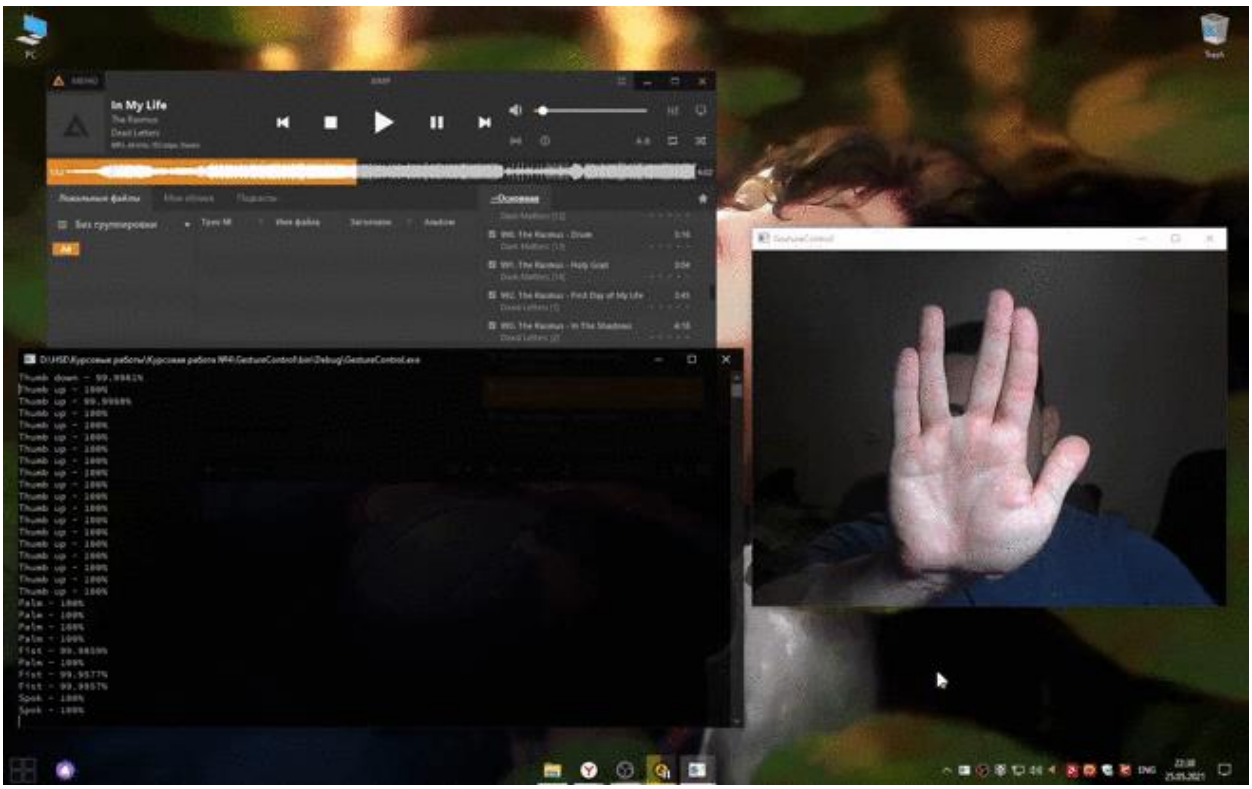
Приложение №1 – схема работы приложения.



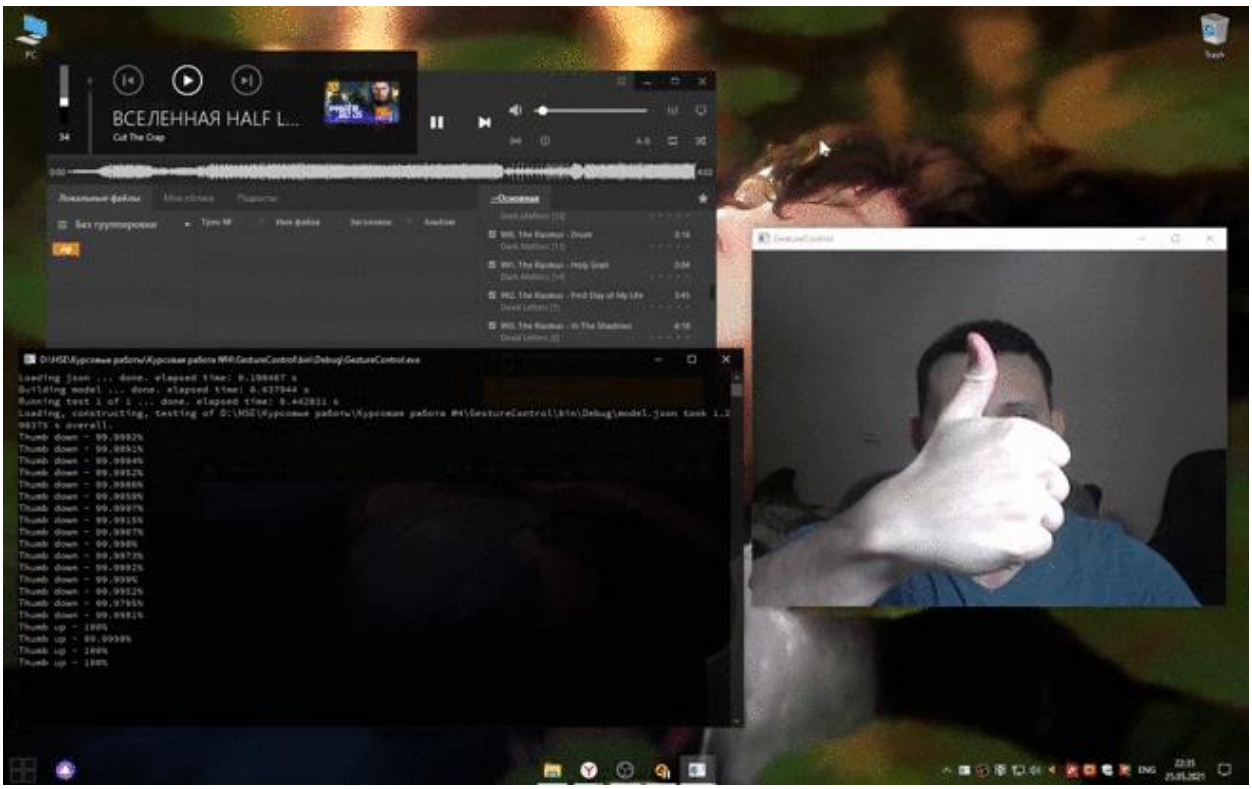
Приложение №2 – пример жеста «Fist».



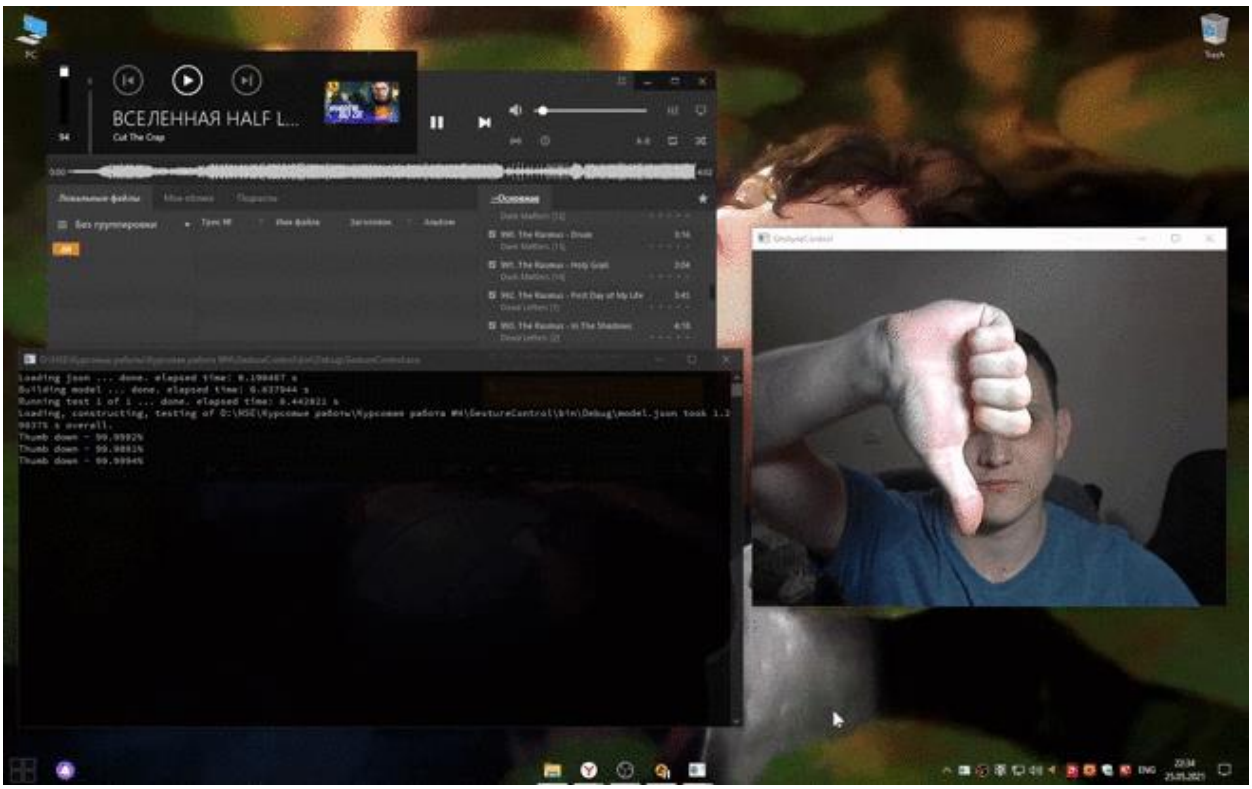
Приложение №3 – пример жеста «Palm».



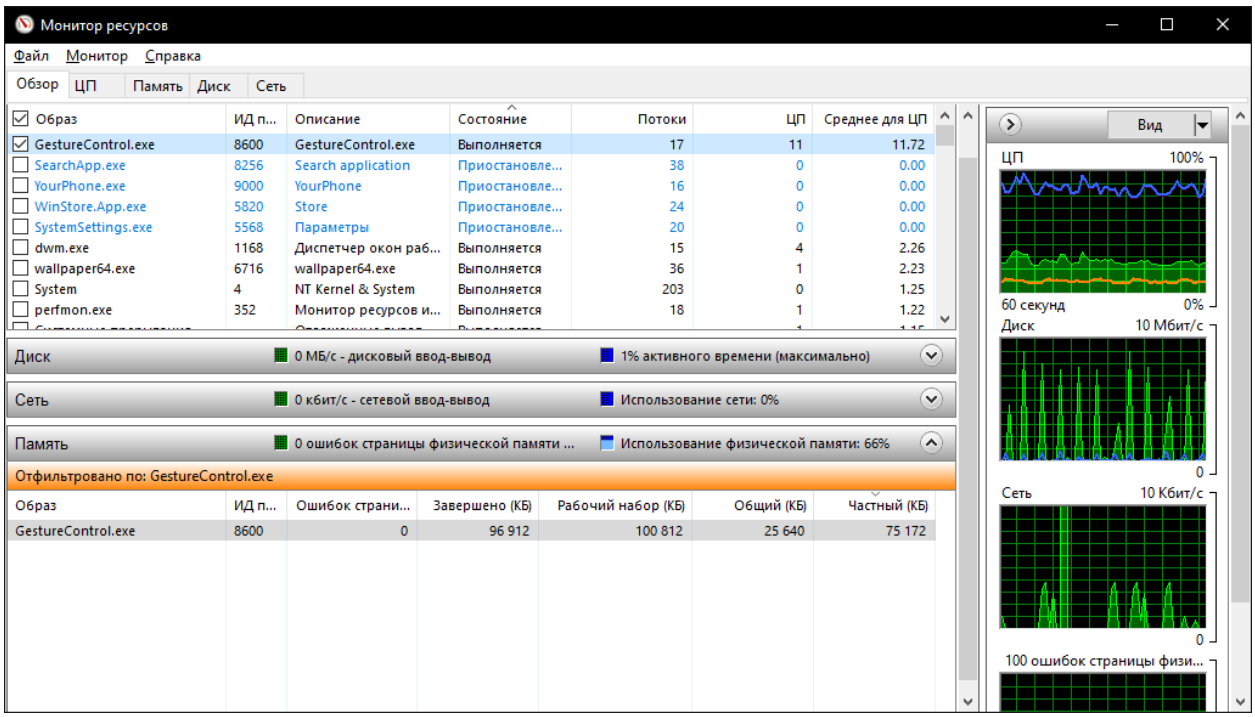
Приложение №4 – пример жеста «Spok».



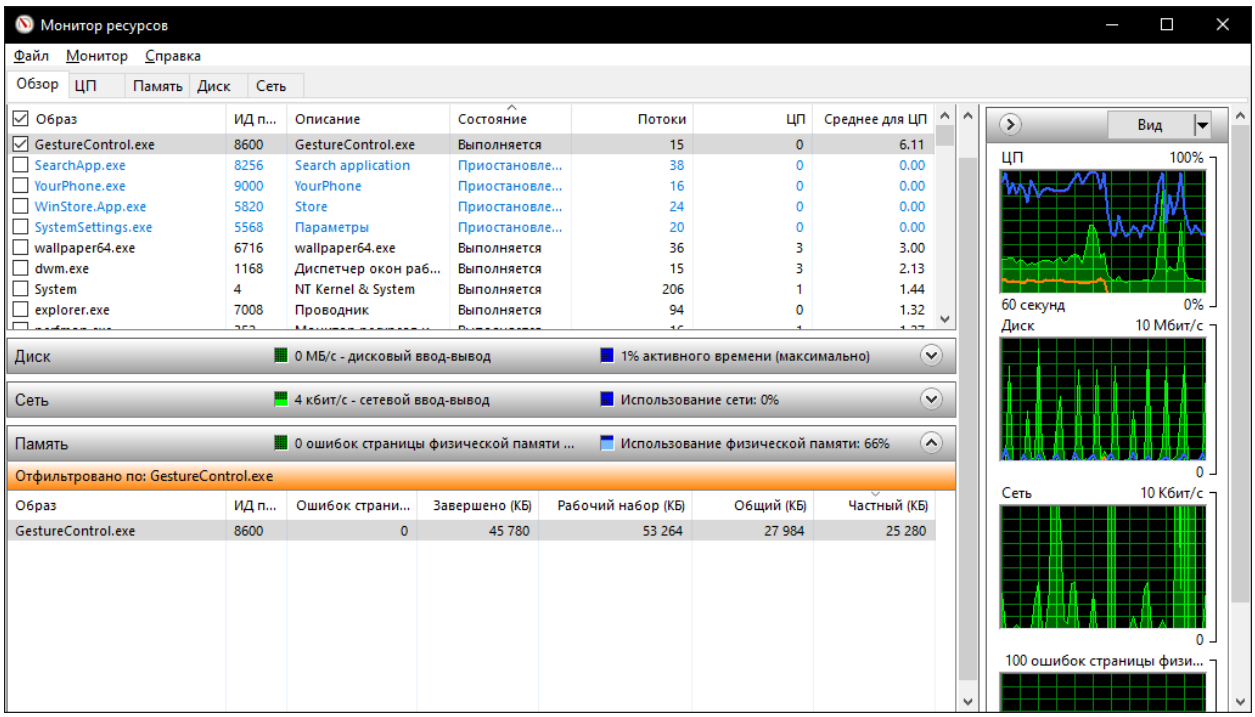
Приложение №5 – пример жеста «Thumb up».



Приложение №6 – пример жеста «Thumb down».



Приложение №7 – монитор ресурсов (активное состояние).



Приложение №8 – монитор ресурсов (во время паузы).

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	Results	Total
Thump up	0	0	0	0	1	0	1	1	1	1	1	0	1	0	0	0	1	0	0	1	45%	63.33%
Swipe right	1	0	1	1	1	0	1	0	1	1	1	1	1	1	1	0	0	1	1	1	75%	
Slide fingers	1	1	1	1	1	1	0	1	1	1	1	1	0	0	0	0	1	1	0	1	70%	

Таблица №1

	1	2	3	4	5	6	7	8	9	10	Results	Total
1	1	1	1	1	1	1	1	0	1	1	90%	62%
2	1	1	1	1	1	1	1	1	1	1	100%	
3	0	0	0	1	1	1	1	0	1	0	50%	
4	0	0	1	1	1	0	0	1	0	0	40%	
5	0	0	0	1	0	0	1	0	1	0	30%	

Таблица №2

	1	2	3	4	5	6	7	8	9	10	Results	Total
Palm	1	1	1	1	1	0	1	1	1	1	90%	80%
Swing	1	0	0	1	1	0	1	1	1	1	70%	
Fist	1	0	1	1	1	1	1	1	0	1	80%	

Таблица №3

	1	2	3	4	5	6	7	8	9	10	Results	Total
Fist	1	1	1	1	1	1	1	1	1	1	100%	90%
Palm	1	1	1	1	1	1	1	1	1	1	100%	
Pointer	0	0	1	0	1	1	0	0	1	0	40%	
Spok	1	1	1	1	1	1	1	1	1	1	100%	
Thumb up	1	1	1	1	1	1	1	1	1	1	100%	
Thumb down	1	1	1	1	1	1	1	1	1	1	100%	

Таблица №4