МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН

Казахский национальный исследовательский технический университет имени К. И. Сатпаева

Институт информационных и телекоммуникационных технологий

Кафедра «Программной инженерии»

Адылханов Данияр Жаныбекулы

Методы машинного обучения для анализа изображении

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

Специальность 7M06101 – Software Engineering

Алматы 2022

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН

Казахский национальный исследовательский технический университет имени К.И. Сатпаева

Институт автоматики и информационных технологий

Кафедра Программная инженерия

Адылханов Данияр Жаныбекулы

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

На соискание академической степени магистра

Название диссертации	Методы машинного обучения для анализа изображении
Направление подготовки	7M06101 – Software Engineering

Научный руководитель
PhD, ассоциированный профессор
_____ Н.К. Мукажанов
«__» _____ 2022г.

Рецензент

Профессор, заведующий
кафедрой «Искусственный
интеллект и Big data»
_____
(ученая степень, звание)
_____ М. Е. Мансурова
«__» _____ 2022 г.

ДОПУЩЕН К ЗАЩИТЕ
Заведующий кафедрой ПИ
канд.физ.-мат.наук,
ассоциированный профессор
_____ А.Н. Молдагулова
«01» 06 2022 г.

Алматы 2022

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН
Казахский национальный исследовательский технический университет имени К.И. Сатпаева

Институт информационных и телекоммуникационных технологий
Кафедра Информационные технологии
Специальность: 6М070400 – Software engineering

УТВЕРЖДАЮ
Заведующий кафедрой ПИ
канд.физ.-мат. наук,
ассоциированный профессор
_____ А.Н. Молдагулова
«_01_» _06_ 2022г.

ЗАДАНИЕ
на выполнение магистерской диссертации

магистранту Адылханов Данияр Жаныбекулы
Тема: «Методы машинного обучения для анализа изображении»
Утверждена *приказом Ректора Университета №1938-М от "07" декабря 2021 г.*
Срок сдачи законченной диссертации *«06» июня 2022 г.*

Исходные данные к магистерской диссертации. Был произведен анализ существующих алгоритмов и множеств изображении. Поставлены цели выполнения диссертационной работы, методологии исследования для получения в результате онлайн классификатора для распознавания медицинских масок надетых на людях во время пандемии COVID-19.

Перечень подлежащих разработке в магистерской диссертации вопросов или краткое содержание магистерской диссертации:
а) обзор существующих алгоритмов и методов для классификации и выбор самого быстрого метода для работы классификатора в режиме онлайн;
б) поиск решения задачи для использования полученной модели;
в) выбор метода и решения использования системы и обоснование выбора;
г) разработка системы распознавания медицинских масок надетых на людях во время пандемии COVID-19.

Рекомендуемая основная литература: Wang Q.J. LPP-HOG: A New Local Image Descriptor for Fast Human Detection / IEEE International Symposium on Knowledge Acquisition and Modeling Workshop. – № 1. – С. 640-643.

ГРАФИК

подготовки магистерской диссертации

Наименование разделов, перечень разрабатываемых вопросов	Сроки представления научному руководителю	Примечание
Сбор множества данных	21.10.2021 г.	Выполнено
Реализация метода машинного обучения	18.11.2021 г.	Выполнено
Реализация полученного результата на серверной части	05.12.2021 г.	Выполнено

Подписи

Консультантов и нормаконтролера на законченную магистерскую диссертацию с указанием относящихся к ним разделов диссертации

Раздел	Консультанты, Ф.И.О. (уч. степень, звание)	Дата подписания	Подпись
Стандартизация/нормоконтроль	Ахмедиярова Айнур Танатаровна – Ассистент проф.	06.05.2022-20.05.2022	
Программное обеспечение/нормоконтроль	Мукажанов Нуржан Какенович – Ассоц. профессор	06.05.2022-20.05.2022	
Антиплагиат	Әубәкіров Б. С. – Ассистент	09.05.2022 - 20.05.2022	

Дата выдачи задания "____"_____ 2022г.

Заведующий кафедрой _____ А.Н. Молдагулова

Научный руководитель _____ Н.К. Мукажанов

Задание принял к исполнению магистрант _____ Д. Ж. Адылханов

Дата "____"_____ 2022 г.

АҢДАТПА

Магистрлік диссертацияда зерттеу нәтижелері баяндалған және қолданыстағы әдістерді салыстыру жүргізілген. Негізгі жұмыс әдістерді, модельдер кітапханасын салыстыру арқылы жүргізілді. Машиналарды оқыту құралдарын қолдана отырып, әрі қарай жұмыс істеудің негізгі әдісі таңдалды және модельдер жетілдірілді. Бұл ғылыми зерттеудің негізгі гипотезасы-бұлтты технологиялар мен серверлік кескінді JavaScript арқылы өңдеуді қолданған кезде Машиналық оқыту арқылы күнделікті қажеттіліктерді цифрландыру тиімдірек болатыны баяндалады. Осылайша, таратушы кескін құрылғысының өндірістік қуаты маңызды болмайды. Осылайша, бұл шешім кез-келген бизнеске қол жетімді болады және карантин кезеңінде жұмысты жеңілдетеді.

АННОТАЦИЯ

В магистерской диссертации изложены результаты исследования и проведено сравнение уже имеющихся методов. Основная работа была произведена посредством сравнения методов, библиотек и моделей. Используя инструменты машинного обучения, был выбран основной метод для дальнейшей работы и проведены улучшения моделей. Основная гипотеза данного научного исследования в том, что цифровизация ежедневных нужд с помощью машинного обучения станет более эффективной, если использовать облачные технологии и обработку выходного изображения на стороне сервера посредством Javascript. Таким образом производственная мощность для устройства передающего изображения будет не важна. И тем самым данное решение будет доступно любому бизнесу и облегчит работу в период карантина.

ANNOTATION

The master's thesis presents the results of the study and compares the existing methods. The main work was done by comparing methods, model libraries. Using machine learning tools, the main method was chosen for further work and model improvements were carried out. The main hypothesis of this scientific study is that the digitalization of daily needs with the help of machine learning will become more effective if cloud technologies and server-side processing of the output image via Javascript are used. Thus, the production capacity for the device transmitting the image will not be important. And thus, this solution will be available to any business and thereby facilitate work during the quarantine period.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Машинное обучение является основой в области массовой обработки изображении. Технология обработки изображении, широко используется для классификации, сегментации и распознавания изображении. Машинное обучение в качестве целевой технологии классификации изображении используется в различных отраслях. Таких как: медицина, агрономия, астрология, бизнес и.т.д. Таким образом, применение машинного обучения в качестве компьютерного зрения стало достаточно важной темой для исследования. Основой технологии компьютерного зарения является компьютеры и информация. Данные технологии наиболее развиты во всем мире. Компьютер является основой технологии. Он берет на себя функцию обработчика и анализатора изображении. Так-как острой проблемой являлся новый вирус, проект был создан для предотвращения распространения данного вируса.

Актуальность темы исследования заключается в том, что во время вспышки COVID-19, принося различные серьезные угрозы миру, показывает нам о том, что нужно быть более осторожным и применять меры предосторожности для предотвращения передачи вируса. У людей был впервые обнаружен новый штамм вируса, который был известен как коронавирус (nCoV) и он ранее не был обнаружен у людей. Первый инфицированный пациент с коронавирусом был обнаружен в декабре 2019 года. Из-за эпидемии короновируса людям стало обязательно носить медицинские маски, чтобы защитить не только себя, а так же чтобы защитить людей которые слабее и не смогут противостоять вирусу. Ведь, нашлись люди которые заболев и зная о своей болезни, не переставали посещать работу, общественные места и даже не пользовались масками и тем самым были одной из причин превращения COVID-19 в глобальную эпидемию. Самое эффективное медицинское вмешательство для предотвращения распространения вируса - ношение масок. Маски для лица стали средством для снижения угрозы вируса и применение медицинских масок для лица в настроящее время является одним из самых необходимых средств защиты в общественном транспорте, густонаселенных районах, крупных предприятиях для обеспечения безопасности. Так-как вирус распространялся при тесном контакте и местах скопления людей, людям нужно было соблюдать личную гигиену, соблюдение безопасной дистанции, воздержание от прикосновения к лицу. Люди не соблюдали данные правила должным образом и в итоге вирус быстро распространился среди людей. Являясь одним из самых эффективных методов защиты, появилась необходимость для автоматического обнаружения масок. Распознавание медицинской маски – способ найти нарушителя порядка и наказания. Что в дальнейшем, поможет людям быть более ответственными и поможет снизить количество больных. Ношение масок уже доказанная, немедикаментозная мера которая помогает удобно и дешево снизить эффективность инфекции COVID-19 [1]. Особенно в мегаполисах,

взаимодействия между людьми в ежедневных поездках и на работе очень опасны. Поэтому крайне важно разработать метод автоматического обнаружения ношения масок, который поможем в борьбе с болезнью. Данный программный продукт может быть использован во всех сферах и использоваться повсеместно. В тех же автобусах использование данной программы может быть незаменимо.

За последние несколько лет, воздействие облачных технологии на мир стало заметно увеличиваться. И поэтому большинство продуктов на данный момент используют данную технологию. Облачные вычисления — информационно-технологическая концепция, подразумевающая обеспечение удаленного доступа к вычислительным ресурсам: сетям передачи данных, серверам, устройствам хранения данных, приложениям и сервисам [2]. Причиной того, почему будут использоваться облачные технологии в том, что устроиству передачи изображения не потребуются огромные вычислительные мощности. Устроиству лишь нужно будет стабильное подключение к интернету и камера. Данный продукт на начальной стадии будет установлен на одном из крупных облачных сервисов. Таких как - Microsoft Azure или Amazon AWS. В дальнейшем, учитывая популяризацию данной услуги, продукт будет поставляться как SaaS (англ. Software-as-a-Service) - модель, в которой пользователь использует в качестве услуги прикладное программное обеспечение, доступного с тонкого клиента. Управление виртуальной средой (вычислительными ресурсами) осуществляется провайдером, предоставляющим облачную услугу [2].

Но, уже на частных или государственных серверах. Так же и в данном продукте будут использоваться облачные технологии для выявления нарушителей и передачи фотографии властям. Так-как камер “Сергек” станет больше, это означает что будет обширная база жителей города и не составит труда определить личность жителя. Президентом Республики Казахстан было сказано: “Дело не в том, что мы собираемся установить какой-то тотальный мониторинг или слежение за действиями наших граждан. Это вопрос их безопасности” [3]. Поэтому, можно в любой момент так же интегрировать и систему идентификации медицинских масок в качестве дополнения к основному функционалу системы “Сергек”.

Цель исследования:

- Создание программного продукта для идентификации людей не носящих медицинские маски в условиях карантина;
- Уменьшение распространения вируса среди людей;
- Помощь крупному бизнесу и государству в уменьшении затрат на работников для идентификации и предупреждению людей не носящих медицинские маски;
- Пополнение государственной казны за счет нарушителей карантина.

Задачи исследования:

- проведение анализа существующих методов для анализа и классификации изображении людей в медицинских масках;

- Выбор среди существующих методов наиболее точного и самого быстрого;

- программная реализация метода и инструмента в рамках его использования в задании идентификации людей в масках.

- Повышение точности и скорости за счет улучшения выбранного метода;

- разработка документации к применению метода и его использования;

Научная новизна исследования заключается в том, что прежде, не использовались никакие сервисы для онлайн обнаружения носителей медицинских масок. В торговых центрах работали люди, которые при входе обязывали надевать медицинские маски. Сейчас это можно осуществить значительно дешевле и более автоматизировано. Так-как устройству-передатчику нужно будет всего лишь иметь соединение с интернетом и камеру. Этого достаточно для того, чтобы показывать имеется ли на посетителе медицинская маска или нет.

Практическая значимость диссертационной работы заключается в том, что была создана модель, которая была использована в связке с Javascript. и обработка изображении происходит в режиме реального времени.

Теоретическая значимость диссертационной работы в том, что велись исследования по уже имеющимся методом для работы моделей формата .pt, .weights, .onnx, .h5, pb в связке с Javascript. Также проводились исследования по улучшению результатов тестирования для повышения точности модели и увеличению скорости обработки.

Апробация работы. Диссертационные исследования были опубликованы и апробированы на международной научно-практической конференции «XXI Сатпаевские чтения», тема статьи «Современные подходы к алгоритмам сжатия гиперспектральных аэрокосмических изображений» в 2022 году.

Структура диссертации. Диссертационная работа состоит из трех глав, введения, заключения, списка использованной литературы и приложения (листинг программы). Материал изложен на 57 страницах, включает 4 таблицы, 36 рисунков, а также одно приложение. Список использованной литературы содержит 40 наименований.

1. Обзор методов для разработки

1.1 Мотивация для разработки

Широко распространенной проблемой с момента появления и распространения COVID-19 стал не только сам вирус, но и его распрострения воздушно-капельным путем. Борьбой с которым является именно использование медицинских масок [4]. Главной проблемой борьбы с новым вирусом является ношение медицинских масок на всю открытую часть лица (не включая область выше носа). Так-как сейчас век высоких технологии, недопустимо проводить идентификацию и сортировку людей в масках привычным, старым способом. Поэтому во многих странах все чаще В последние годы искусственный интеллект и методы машинного обучения активно применяются для решения практических задач. [5] Так-как машинное обучение и искусственный интеллект - это идеальное решение для подобной задачи, такой как пандемия. Поэтому в данной диссертационной работы за основу было взято машинное обучение и оно поможет людям стать более осознаннами в проблеме распространения COVID-19 и будет принимать решения о том, носят ли люди маски или же все же нет. Так-как данное ПО будет использоваться во многих сферах, можно подстраивать его под разные условия. Например, если это вход в торговый центр, можно установить открытие дверей на посетителя в маске. Если же посетитель заходит без медицинской маски, двери открываться не будут. Или же выводить напоминание посетителю надеть маску и что при повторной попытке зайти без маски, его фотография будет отправлена в государственные органы для выписки штрафа. Т.е. доработка данной системы для каждой организации может быть индивидуальна и возможности доработки могут быть безграничны. Поэтому использовав машинное обучение можно достичь огромных высот в здравоохранении, эпидемиологии, исследовании поведения посетителей и.т.д. Минусы привычной идентификации масок использующихся на сегодняшний день:

- Человеческий фактор. Человек может не уследить за всеми посетителями и пропустить людей без масок. Именно среди этих людей может быть зараженный;
- Сотрудник не может преследовать всех посетителей одновременно и следить за соблюдением масочного режима. Идея проекта в том, чтобы установить данное ПО во всем торговом центре. Это требуется для того, чтобы посетители не снимали свои маски после входа в помещение. На сегодняшний день, многие посетители снимают маски как только попадают в нужное помещение, такого быть не должно. Программное обеспечение будет следить за всеми посетителями и в случае нарушения, будет сохранять фотографии нарушителей и выводить на экран с предупреждением.

1.2 Способы и методы работы

Так-как основой работы является распознавание лиц, было выделено несколько существующих методов распознавания лиц [6]. Так-как тут используется в основном обнаружение некого объекта на фотографии в виде

маски, нам требуется также и обнаружение лица. Для обнаружения лица на фотографиии будут рассмотрены такие методы и фреймворки как: YOLO, OpenCV + Tensorflow, DeepFace.

1.2.1 Yolo – You Only Look Once (“Стоит только раз взглянуть”)

Yolo является полностью настраиваемым алгоритмом глубокого обучения. Базовая модель YOLO обрабатывает изображения в режиме реального времени со скоростью 45 кадров в секунду. Уменьшенная версия сети, Fast YOLO, обрабатывает поразительные 155 кадров в секунду, сохраняя при этом вдвое большую карту, чем у других детекторов реального времени [7]. На рисунке 1 изображено то, как именно Yolo отмечает на изображении нужные объекты в рамках классификации.

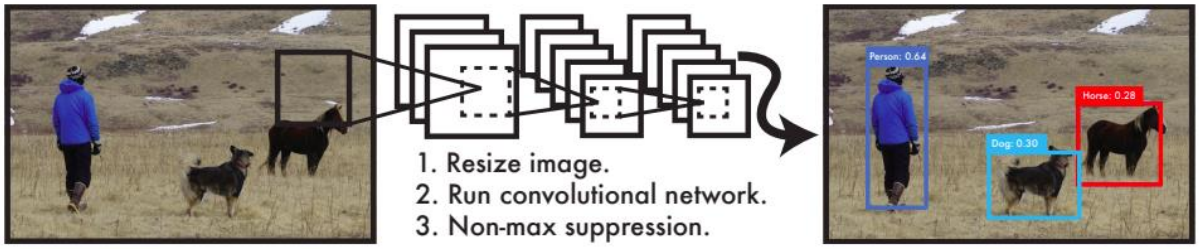

Рисунок 1 – Система Обнаружения YOLO. Обработка изображений с помощью YOLO проста и понятна. Система YOLO

1. изменяет размер входного изображения до 448 × 448,
2. запускает единую сверточную сеть на изображении и
3. определяет пороговые значения результирующих обнаружений по достоверности модели [7].

Основные преимущества YOLO – YOLO имеет преимущества перед другими системами на основе классификаторов. Он просматривает все изображение во время тестирования, поэтому его прогнозы основаны на глобальном контексте изображения. Он также делает прогнозы с помощью одной оценки сети, в отличие от таких систем, как R-CNN , которым требуются тысячи для одного изображения. Это делает его чрезвычайно быстрым, более чем в 1000 раз быстрее, чем R-CNN, и в 100 раз быстрее, чем Fast R-CNN [8].

Как отмечено в [9], YOLO – Очень популярная архитектура CNN и оно может быть использовано в качестве компьютерного зрения, для распознования объектов. И главной особенностью является то, что в своей основе CNN не может использоваться сразу ко всему изображению. А YOLO наоборот, используется и применяется ко всему изображению сразу. Следуя механизму привязки, представленному в YOLO9000, YOLOv3 делает прогнозы на основе карт объектов на трех разных уровнях. Карты объектов разделены на сетки, а ячейки сеток размещены с привязками разных размеров и соотношений сторон, которые получены путем выполнения K-средних значений в наборе данных. Привязки помечаются как передние, если их долговая расписка с любым полем истинности больше порогового значения 0,3. Центральные координаты и размеры блоков регрессируют на основе положений и размеров привязок. По сравнению с использованием полностью связанного

слоя для непосредственного прогнозирования координат ограничивающих рамок, механизм на основе привязки может значительно улучшить отзыв модели на объектах небольших размеров. YOLO выполняет классификации с помощью двоичных классификаторов C, где C – количество классов набора данных. Это изменение наделяет YOLO способностью выполнять классификацию с несколькими метками. Следуя идее остаточной сети, в YOLO была принята более глубокая сеть Darknet-53. YOLO конкурентоспособен как по точности, так и по скорости, и он надежен в обнаружении различных типов объектов. В результате YOLO получил широкое применение в таких отраслях промышленности, как производство и военные. Было предложено много расширенных работ над YOLO, которые в основном были сосредоточены на скорости, точности и размере модели. Слой точечной свертки в MobileNet является эффективной заменой остаточным слоям в Darknet-53 с меньшей вычислительной сложностью. Смешанный YOLOv3-LITE использует мелкую основу YOLO-LITE для замены Darknet-53 и добавляет остаточную структуру и параллельные подсети с высоким и низким разрешением для достижения слияния мелких и глубоких функций.

Блок глобального контекста может создавать отношения зависимости на большие расстояния между всеми пикселями объектов на карте объектов, чтобы модель могла фокусироваться на разных регионах. Обучаемый компонент семантического слияния использует выходные данные сети извлечения, повышая способность головной сети лучше распознавать объекты на картах объектов.

YOLO следует идее блока привязки, устанавливая 9 блоков привязки, относящихся к площади и соотношению сторон.

Кластеризация k-средних по данным, чтобы получить предварительные значения ограничивающей рамки для задачи. K-means – это широко используемый алгоритм кластеризации, основанный на евклидовом расстоянии. Гипотеза K-средних состоит в том, что данные генерируются из k точных центров и некоторого гауссова шума. Сначала он выбирает k случайных точек из центроидов данных, а затем присваивает все точки ближайшим центроидам кластера. Центроиды пересчитываются во вновь сформированных кластерах после присвоений. Присваивание и вычисление центроидов повторяются до тех пор, пока не будет достигнуто максимальное количество итераций или центроиды вновь сформированного кластера не изменятся. Конечный размер привязки – это центр 9 кластеров (например, P8, P2), показанных на рисунке 3. Значения ограничительной рамки показаны в таблице 1. Таким образом, это может снизить сложность обучения модели и значительно улучшить производительность модели при обнаружении целевого ограничивающего прямоугольника. На рисунке 2 приведена визуализация кластеризации k-средних.

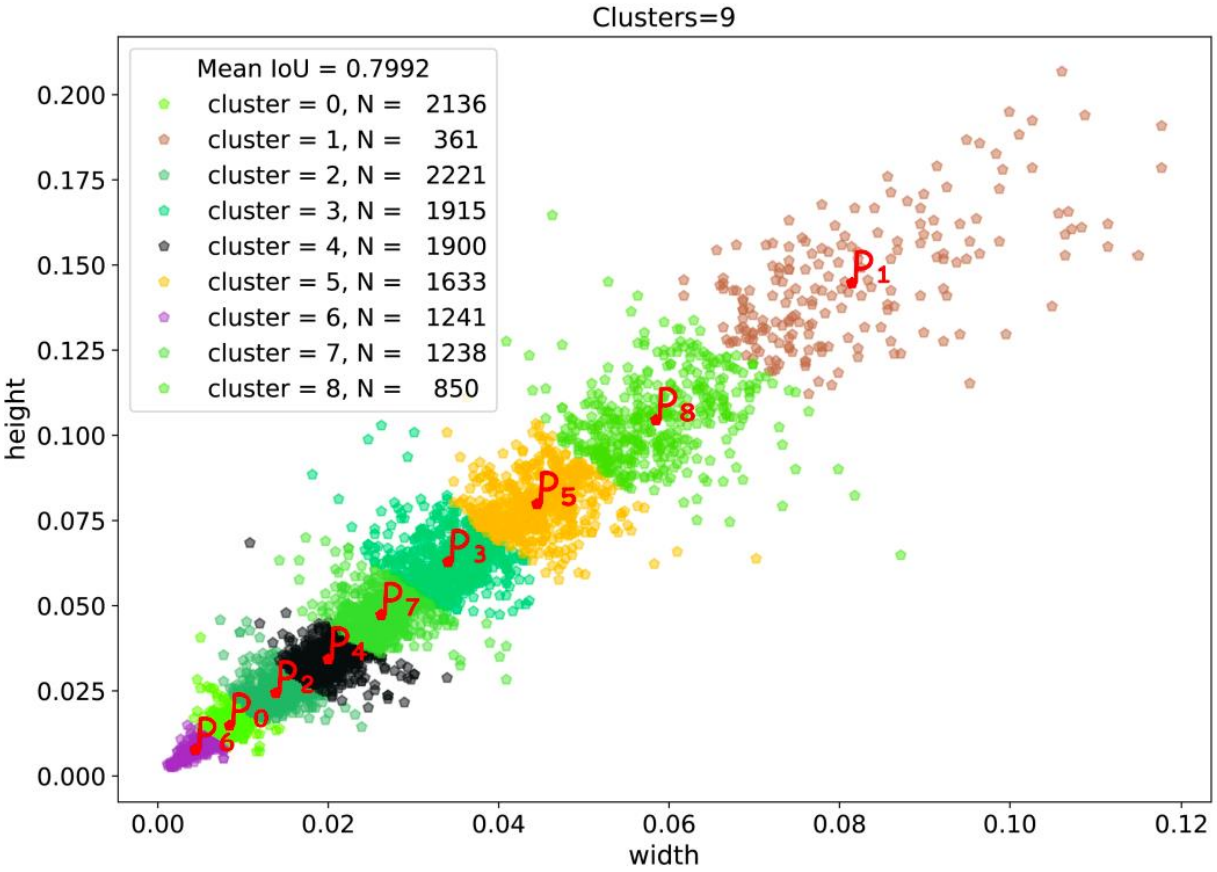

Рисунок 2 - Визуализация кластеризации k-средних. Прогнозируя ширину и высоту прямоугольника как смещения от центроидов кластера.

1.2.2 OpenCV + Tensorflow

OpenCV - open source библиотека компьютерного зрения, которая предназначена для анализа, классификации и обработки изображений[10]. Возможности OpenCV:

- Кадрирование;
- Изменение размера ;
- Поворот изображения;
- Перевод цветного изображения в черно-белое;
- Размытие ;
- Рисование линии;
- Надписи;
- Распознавание лиц [11].

В OpenCV главное - обнаружение лиц на фотографии. После обнаружения лица, это лицо извлекается для дальнейшей обработки и работы с ним.

После извлечения лица, из него извлекаются определенные черты, который были даны в качестве входных данных. Когда мы обучаем нейронную сеть, она учится находить похожие вектора лиц, которые выглядят одинаково.

OpenCV использует в своей основе алгоритм Виолы - Джонса. Данный метод используется в основе для поиска лиц на изображениях. Этот алгоритм разделен на четыре этапа:

- Выбор функции Хаара
- Интегральные изображения
- АдаБуст
- Каскадный классификатор

В библиотеке OpenCV используются дополнительные признаки помимо основных в алгоритме Виолы - Джонса. (Рисунок 3)

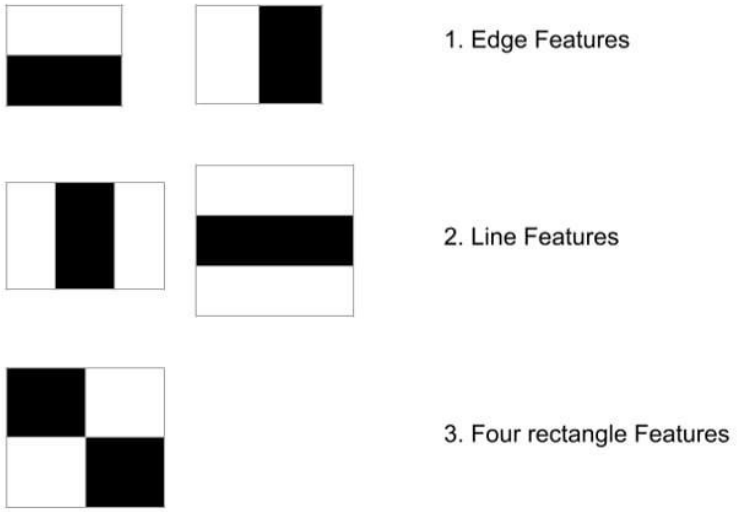

Рисунок 3 - Признаки в методе Виолы-Джонса [12].

Алгоритм в своей основе работает подобным образом:

- Получено изображение для работы;
- Далее окно выбранное по размеру двигается по фотографии/изображению с 1 шагом;
- В каждом окне просматривается больше 200000 вариантов сканирования за счет масштабирования признаков и размера ячейки;
- Каждый найденный признак передается классификатору и классификатор дает решение.

Пример работы на рисунке 4.

В современном мире, есть множество систем для распознования лиц и классов на изображениях. Но, так-как сравнение проводилось между основными моделями и библиотеками YOLO (преимущественно V3-V5) и OpenCV (Keras + Tensorflow), был выбран более быстрая и легкая к обучению модель YOLO. Самым главным преимуществом данного метода стало то, что YOLO может с легкостью работать в режиме онлайн и не требовать от

передающего устроиства больших производственных мощностей. В то время как OpenCV в купе с Keras и Tensorflow может классифицировать объекты только на фотографиях. Работать с захватом видеоизображения с камеры устроиства данная библиотека не может.

Рисунок 4 - Пример работы алгоритма сканирования окна с признаками.

Из всех функции, для данного проекта является важной - функция распознавания лиц. Для поиска лиц используется принцип “скользящего окна” (метод Виолы — Джонса). Для каждой области изображения, над которой проходит окно, рассчитывается признак Хаара [13]. То-есть изображение каждый раз увеличивается и с каждым проходом ищет лицо большего размера.

Признак Хаара состоит из смежных прямоугольных областей. Они позиционируются на изображении, далее суммируются интенсивности пикселей в областях, после чего вычисляется разница между суммами. Эта модель довольно сложная и долгая в обучении, но она работает гораздо быстрее, чем нейронные сети с использованием Tensorflow. И тут даже не нужен графический процессор. [13].

Основные модули OpenCV:

cxcore — ядро

* содержит базовые структуры данных и алгоритмы:

— базовые операции над многомерными числовыми массивами

— матричная алгебра, математические ф-ции, генераторы случайных чисел

— Запись/восстановление структур данных в/из XML

— базовые функции 2D графики

CV — модуль обработки изображений и компьютерного зрения

— базовые операции над изображениями (фильтрация, геометрические преобразования, преобразование цветовых пространств и т. д.)

— анализ изображений (выбор отличительных признаков, морфология, поиск контуров, гистограммы)

— анализ движения, слежение за объектами

— обнаружение объектов, в частности лиц

— калибровка камер, элементы восстановления пространственной структуры

Highgui — модуль для ввода/вывода изображений и видео, создания пользовательского интерфейса

— захват видео с камер и из видео файлов, чтение/запись статических изображений.

— функции для организации простого UI (все демо приложения используют HighGUI)

Cvaux — экспериментальные и устаревшие функции

— пространств. зрение: стерео калибрация, само калибрация

— поиск стерео-соответствия, клики в графах

— нахождение и описание черт лица

CvCam — захват видео

— позволяет осуществлять захват видео с цифровых видео-камер. [14]

OpenCV в совокупности с Tensorflow так же используется для классификации.

1.2.3 DeepFace

DeepFace - это фреймворк для распознавания лиц и анализа атрибутов лица (возраст, пол, эмоции). Эксперименты в рамках тестирования самой системы показывают, что люди имеют точность распознавания лиц 97,53%, DeepFace уже преодолели этот уровень точности. Если брать общий процесс распознавания лиц, он состоит из 5 этапов:

- Обнаружение;
- Выравнивание;
- Нормализация;
- Представление;
- Проверка. [15]

Эта модель как и многие подобные модели использует сверточные нейронные сети и сверточные нейронные сети представляют лица в виде векторов [15]. На рисунке 5 видно то, как можно использовать DeepFace.

Рисунок 5 - Методы использования DeepFace для идентификации личности [16]

Deepface - это облегченная платформа для анализа лиц, включающая распознавание лиц и демографию (возраст, пол, эмоции и расу) для Python. Может так же применить анализ лица с помощью нескольких строк кода [16].

DeepFace чаще всего используется в развлекательных целях. Используется для так называемых DeepFake. Данная техника дает возможность замены лица на видео или фотографиях без монтажа и используя только глубокое обучение. Принцип работы DeepFace на рисунке 6 и рисунке 7.

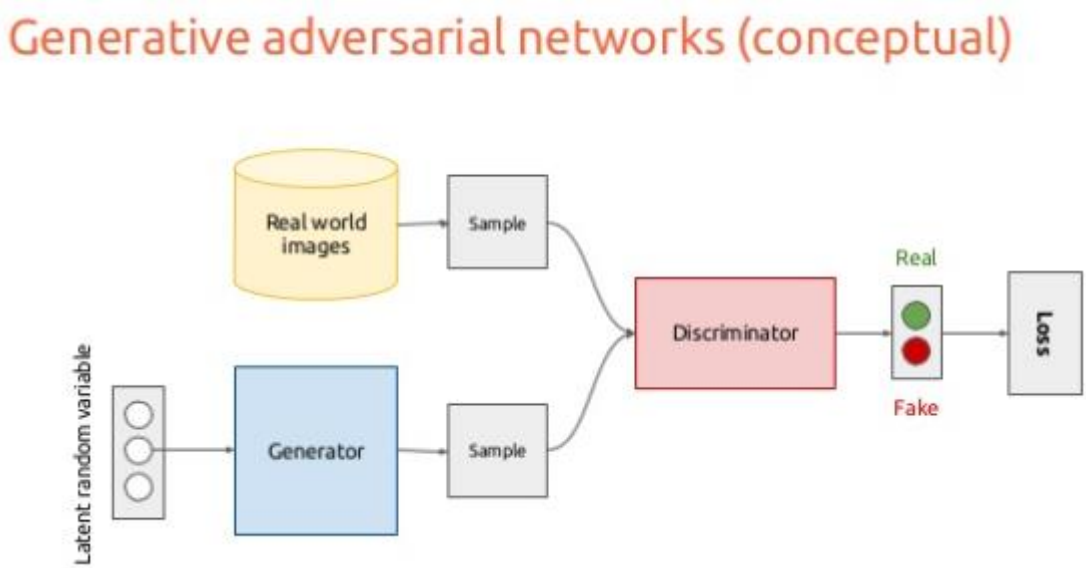

Рисунок 6 – Визуализация работы DeepFace [17]

Рисунок 7 – Использование энкодера в DeepFace [18]

2. Выбор метода и методологии

Выбор метода для идентификации или же классификации жителей города на “Маска присутствует” и “Маска отсутствует” на персональном компьютере с веб-камерой или недорогом устройстве оборудованном камерой для данной программы не имеет значения. Так-как данный проект выполнен с использованием облачных технологии и вычислительные мощности устройства, на котором будет запущена данная программа не важны. Одно из самых важных аспектов в работе данной программы является связь с облачным сервером.

Для получения начальных тестовых данных был выгружен готовый набор данных с Kaggle и так же для лучших результатов была использована выборка людей без масок. Так-как запись нового массива данных и переразметка этих данных вручную было бы нецелесообразно. В данном массиве данных уже хранятся более 500 изображении которые на данный момент будут достаточны для работы. Исключениями могут быть только те случаи, когда люди могут носить маски с разными рисунками (человеческие лица, усы и бороды, животные морды и.т.д.) (в соответствии с рисунком 8). Так-как работа с такими масками еще не была протестирована.

Рисунок 8 - Пример маски с изображенным принтом

При выборе метода классификации было выбрано 2 основных и быстрых метода. Так-как все-таки данное программное обеспечение является облачным, скорость здесь являет немаловажную роль.

Первым и более предпочтительным является YOLO и вторым был выбран OpenCV+Tensorflow. В чем же все же преимущество YOLO, другие системы предварительного обнаружения переназначают классификаторы или локализаторы для выполнения обнаружения. Они применяют модель к изображению в различных местах и масштабах. Области изображения с высокой оценкой считаются обнаружениями. YOLO использует совершенно другой подход. YOLO применяет одну нейронную сеть к полному изображению. Эта сеть делит изображение на области и прогнозирует ограничивающие рамки и вероятности для каждой области. Эти ограничивающие рамки взвешиваются

по предсказанным вероятностям. Это делает его быстрее R-CNN в 1000 раз и в 100 раз быстрее Fast R-CNN. Пример кода (см. Приложение А)

Вторым, но не менее важным является OpenCV. OpenCV берет за основу классификатор Хаара. Данный классификатор основывается на машинном обучении. Он работает за счет того, что извлекает Хаара из каждого изображения. Каждое окно размещается на картинке для расчета одного из признаков. Эта характеристика представляет собой единичное значение путем вычитания суммы пикселей под белой частью окна из суммы пикселей под черной частью окна. Все возможные размеры каждого окна размещены во всех возможных местах каждого изображения для расчета огромного набора функции. Для обнаружения лиц используя OpenCV и классификатор Хаара используется следующий код (см. Приложение Б)

Рассмотрев эти два варианта можно увидеть что основа у YOLO так же состоит из OpenCV. Поэтому выбор падает на более прогрессивный, более быстрый, но в то же время более сложный метод YOLO. Одним из самых стабильных версии является YOLO v3 (В соответствии с рисунком 9). Но, минусом YOLO v3 является точность и скорость. Они гораздо выше у YOLO v4, чем у исходного YOLO v3.

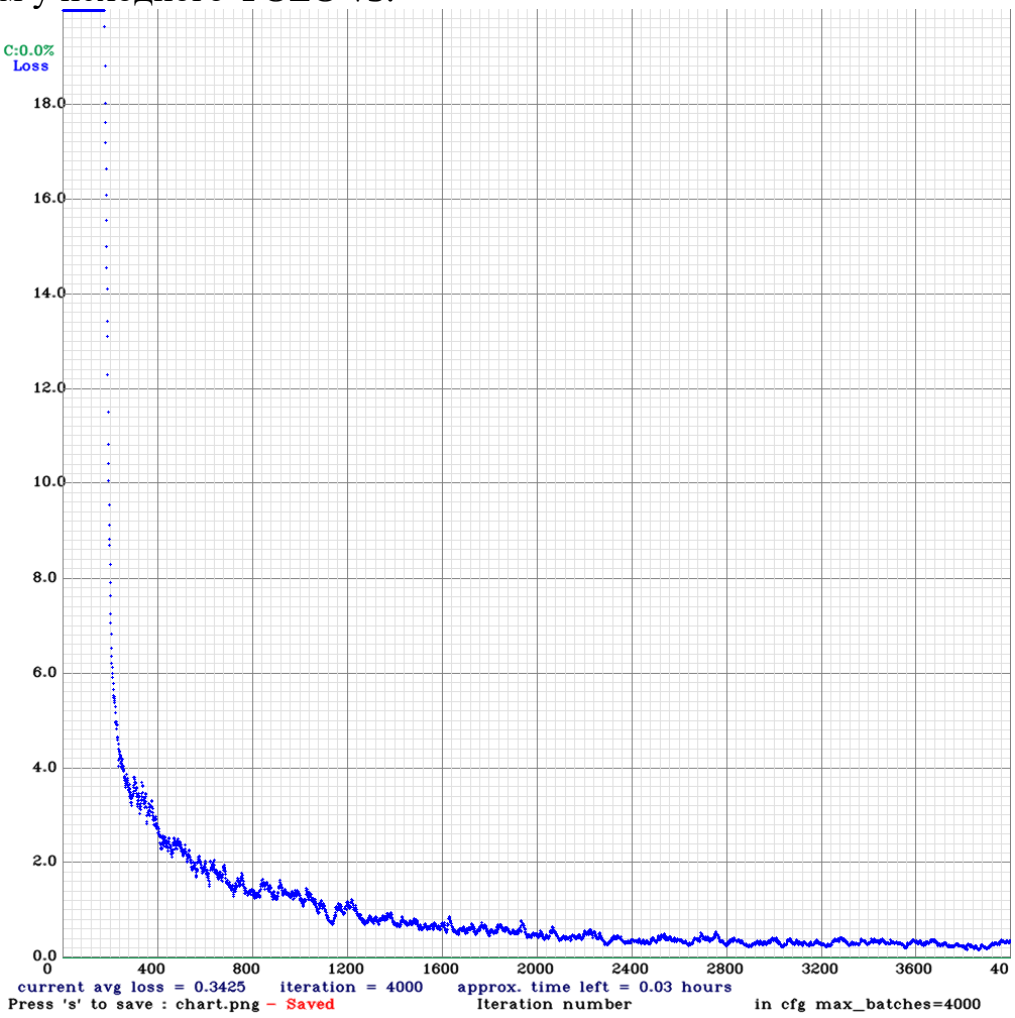

Рисунок 9 - График тестового обучения YOLO v3

2.1. Описание методологии и их недостатков.

В этом разделе разбора YOLO можно представить справочную информацию. Он разрабатывает наиболее представительные и новаторские двухэтапные методы обнаружения объектов с их значительным вкладом в обнаружение объектов. Сначала будут описаны их методологии, а затем разобраны их недостатки.

2.1.1 HOG

HOG - это дескриптор признаков, который широко применяется в инвариантных доменах для различения объектов путем идентификации их формы и структуры. Локальная структура объекта, шаблон, аспект и представление обычно могут быть охарактеризованы расположением градиентов локальной интенсивности или способами ребер. В методе обнаружения HOG [19] первым шагом является разбиение исходного изображения на блоки, а затем распределение каждого блока по небольшим областям. Эти области называются клетками. Обычно блоки изображения перекрывают друг друга, из-за этого соответствующая ячейка может быть частью многих блоков. Для каждого пикселя внутри ячейки он вычисляет градиенты по вертикали и горизонтали. Недостатки метода HOG - Из-за появления глубокого обучения и его значительных применений разумным мнением было заменить классификаторы, развернутые по методологии HOG [19], классификатором, основанным на сверточной нейронной сети [20] из-за его сравнительно более высокой точности. Но были и некоторые проблемы. Вычислительная стоимость сверточных нейронных сетей была высокой, а скорость - слишком низкой. Поэтому было сложно запустить классификатор на основе CNN на многочисленных исправлениях, созданных методом обнаружения скользящего окна (В соответствии с рисунками 10, 11).

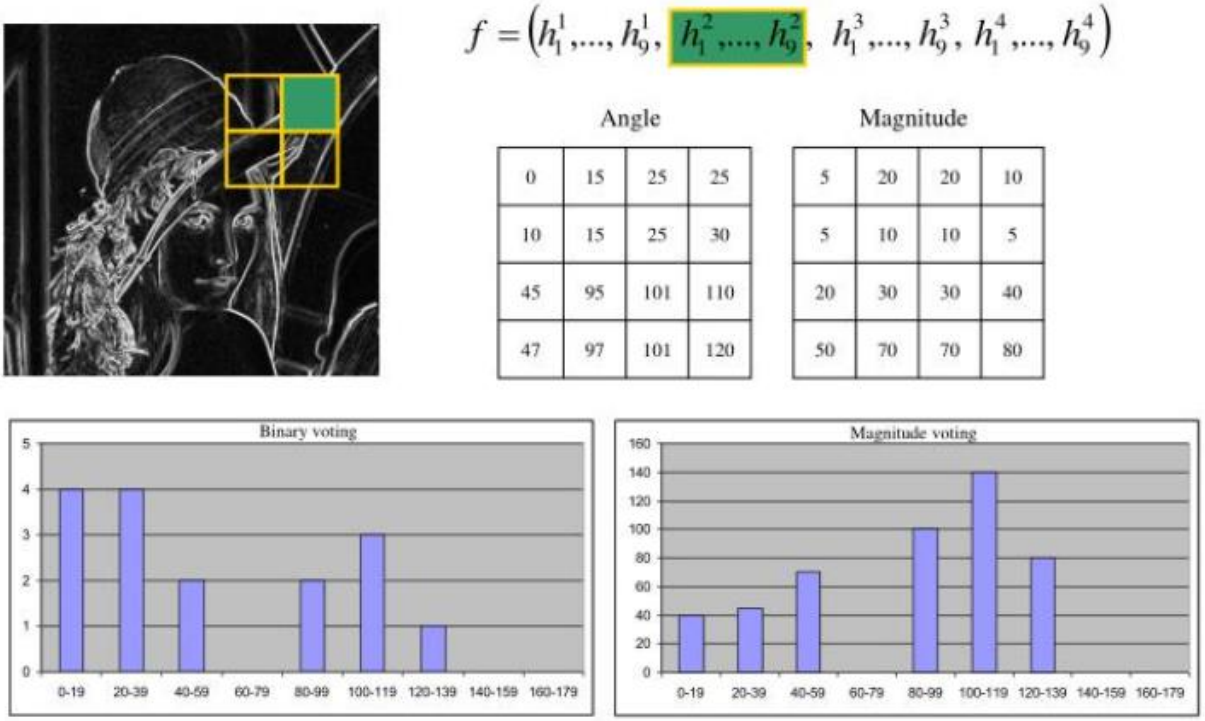

Рисунок 10 – Вектор признаков HOG для одного блока [21]

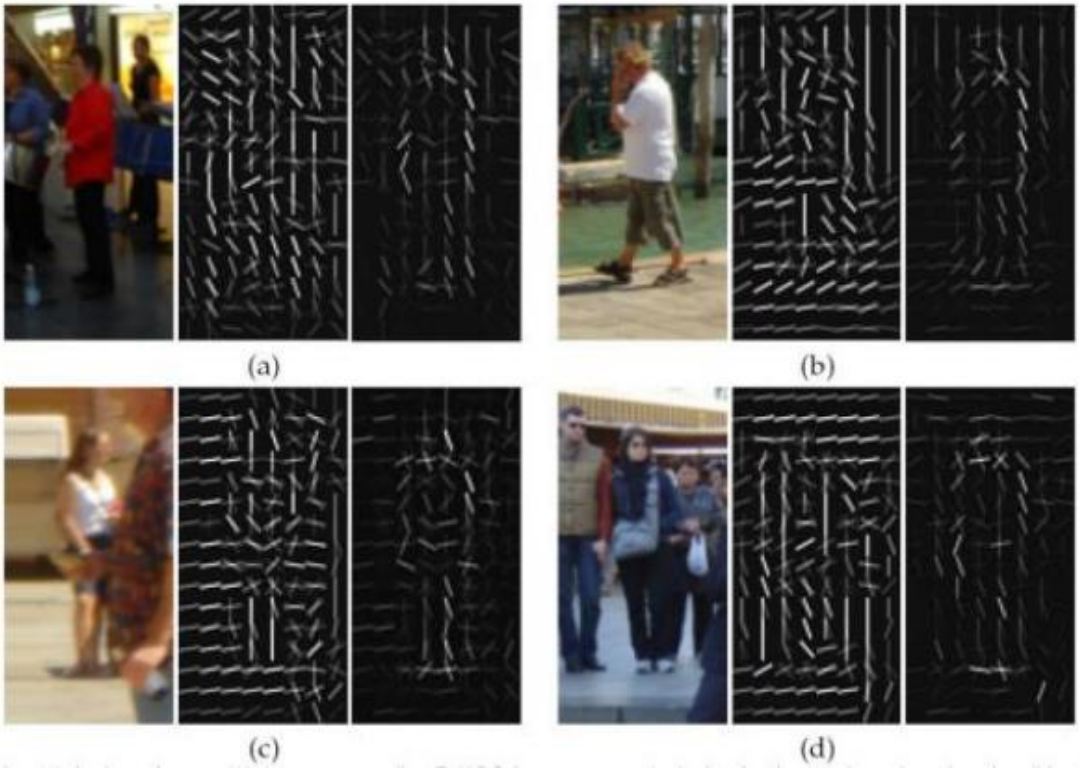

Рисунок 11 - Пример HOG [21]

Эта трудность была решена в R-CNN [22].

В нем используется алгоритм, основанный на предложениях объектов, называемый методом выборочного поиска [23]. Этот подход уменьшает количество ограничивающих рамок до 2000 региональных предложений, которые были отправлены в классификатор R-CNN.

2.1.2 R-CNN

Алгоритм сверточных нейронных сетей на основе регионов (R-CNN) [22], использует группу блоков для изображения, а затем анализирует в каждом блоке, содержит ли какой-либо из блоков цель. Он использует метод выборочного поиска, чтобы выбрать эти разделы из изображения (В соответствии с рисунком 12).

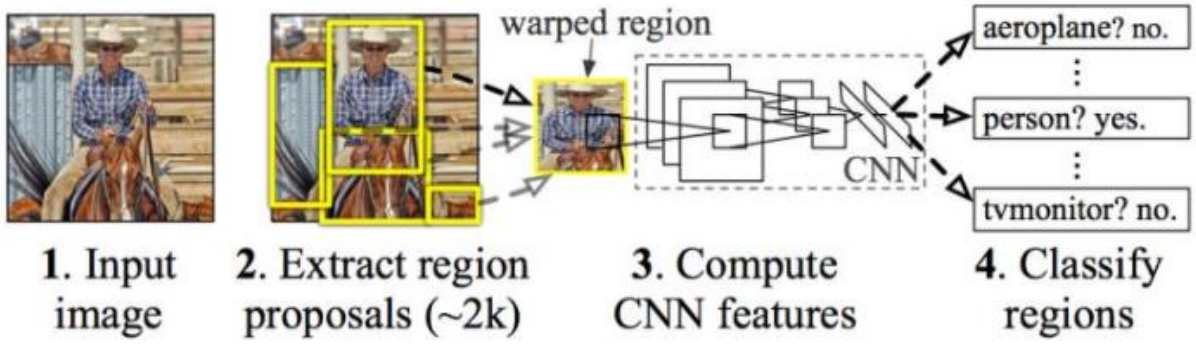

Рисунок 12 - Регионы с функцией R-CNN [24]

В объекте используются четыре области. Это различные масштабы, цвета, текстуры и корпуса. Недостатки метода R-CNN - Основанный на выборочном поиске [23], на одно изображение выделяется 2000 секций. Для каждой области или части изображения мы должны выбрать объекты с помощью CNN. Для этого, если у нас есть "i" количество изображений, то выбранные области станут i × 2000. Весь метод идентификации цели с помощью R-CNN использует следующие три модели: линейный классификатор SVM для идентификации объектов, CNN используется для извлечения характеристик, а регрессионная модель требуется для ужесточения ограничивающих рамок. Все эти три процесса в совокупности занимают значительное количество времени. Это увеличивает время выполнения метода R-CNN. Таким образом, R-CNN требуется почти от 40 до 50 секунд, чтобы предсказать результат для нескольких новых изображений [25].

2.1.3 FAST R-CNN

Вместо использования трех разных моделей R-CNN, Fast R-CNN [25] использует одну модель для извлечения характеристик из разных регионов. Затем он распределяет регионы по нескольким категориям на основе выделенных объектов, и границы распознанных подразделений возвращаются вместе. Fast R-CNN использует метод объединения пространственных пирамид [25] для вычисления только одного представления CNN для всего изображения. Он передает одну область для каждого изображения в конкретную модель сверточной сети, заменяя три отдельные модели для извлечения характеристик, распределения по подразделениям и создания ограничивающих рамок. (В соответствии с рисунком 13).

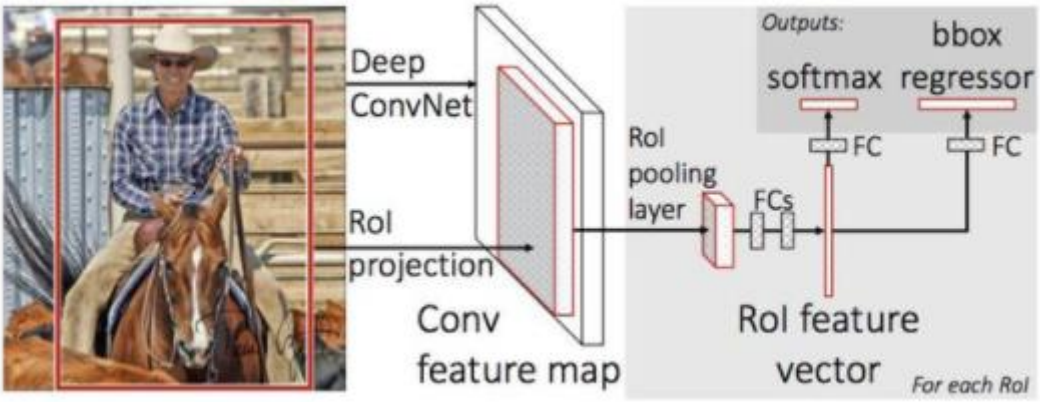

Рисунок 13: Быстрая архитектура R-CNN. Входное изображение и несколько областей интереса (ROI) вводятся в полностью сверточную сеть. Каждый показатель рентабельности инвестиций объединяется в карту объектов фиксированного размера, а затем сопоставляется с вектором объектов с помощью полностью связанных слоев.
[27].

Недостатки метода Fast R-CNN - В Fast R-CNN также используется метод выборочного поиска [23] для обнаружения соответствующих областей.

Этот метод длительный и требует много времени. Обычно для обнаружения объектов на эту полную процедуру требуется почти две секунды для каждого снимка. Поэтому его скорость довольно хороша в отличие от R-CNN. Однако, если мы рассматриваем обширные реальные наборы данных, то быстродействию быстрого подхода R-CNN по-прежнему не хватает скорости.

2.1.4 FASTER R-CNN

Более быстрый R-CNN [28] - это преобразованный вариант быстрого R-CNN. Существенное различие между ними заключается в том, что faster R-CNN реализует сеть региональных предложений (RPN) [29], но fast R-CNN использует метод выборочного поиска для получения соответствующих регионов. (В соответствии с рисунком 14).

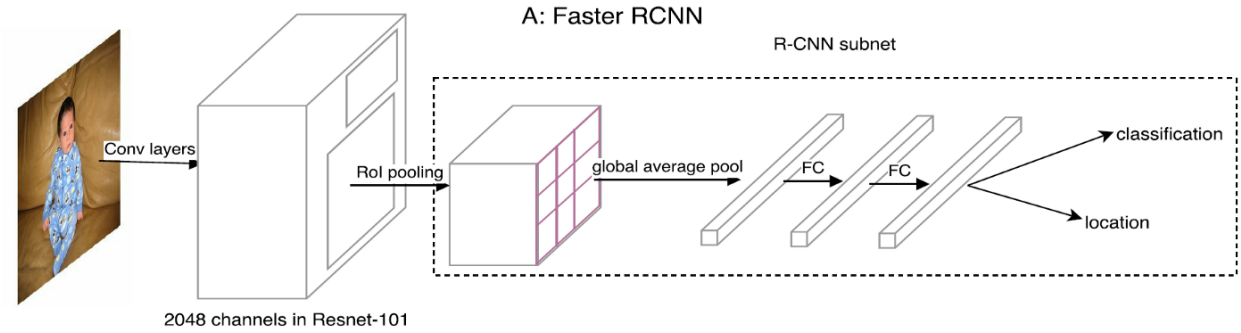

Рисунок 14: Архитектура Faster R-CNN [30].

На входе RPN принимает карты объектов изображения и выдает набор рекомендаций по объектам и оценку объектности для каждой рекомендации на выходе. Обычно этот подход занимает в десять раз меньше времени в отличие от быстрого подхода R-CNN из-за RPN. Недостатки более быстрого метода R-CNN - Чтобы выделить все цели на данном изображении, для этой процедуры требуется несколько проходов для этого конкретного изображения. Различные системы работают последовательно, поэтому выполнение предстоящей операции основано на выполнении предыдущих операций. Этот подход использует региональные сети предложений для локализации и идентификации объектов на изображении. Но RPN не рассматривает полную картину, поскольку использует только те части картины, которые имеют высокую вероятность присутствия целей [25].Сравнение методов приведено в таблице 1.

Таблица 1 - Сравнение методов для распознавания

Тип	Основная характеристика	Время обработки	Недостатки
R-CNN	Для создания регионов он использует выборочный поиск. Из	40-50 секунд	Время, затрачиваемое на прогнозирование, велико, потому что через CNN

	каждого изображения он извлекает около 2000 областей.		определенно проходит несколько регионов, и в нем используются три различные модели для обнаружения целей
Fast R-CNN	Чтобы выделить особенности, каждое изображение проходит один раз через CNN. Все различные модели, применяемые в R-CNN, объединяются вместе, образуя единую модель. Он использует метод выборочного поиска на картах объектов для получения результата распознавания цели.	2 секунды	Используемый метод является длительным и трудоемким, поэтому время вычислений по-прежнему велико.
Faster R-CNN	Предыдущий подход заменяется сетями региональных предложений. Поэтому эта процедура работает намного быстрее по	0.2 секунды	Предложение области объекта отнимает много времени. Различные типы систем работают последовательно. Таким образом, выполнение всей процедуры

	сравнению с предыдущими методами.		основано на выполнении предыдущих операций.

Двухступенчатые детекторы обеспечивают достаточную точность, но время, затрачиваемое на вычисления, велико. Поэтому для обработки за меньшее время при обеспечении достаточной точности предлагаются одноступенчатые детекторы. Некоторые алгоритмы в одноступенчатой модели - это SSD и варианты YOLO. Улучшив архитектуру двухступенчатых моделей и внеся некоторые изменения, такие как устранение конвейера, одноступенчатая модель достигла превосходной скорости. Но в то же время он не достиг достаточной точности. Следовательно, исследователи находятся в процессе постоянных изменений.

2.2. Традиционная задача и проблемы обнаружения и классификации

В большинстве случаев, подобные задачи обнаружения и классификации лиц решаются R-CNN. Но, как на сегодняшний, так и прежде, данная модель является дорогостоящей в вычислительном отношении и она не совсем предназначена для работы в режиме реального времени. Поэтому, основной задачей является то, чтобы данное создаваемое решение имело большую точность и большую скорость в работе в реальном времени.

Первым экспериментом является то, что была произведена переразметка данных для уменьшения площади распознавания. Но, данная теория оказалась неверна. Так-как YOLO перестал смотреть на остальную часть лица и смотрел только на область ниже носа и тем самым подвергался большим ошибкам.

Основная задача была ясна и была извесна цель того, что именно предстоит находить, нам не требовалось размечать полностью все лицо людей. Требовалось лишь указать на маски и выделять только их. Поэтому первым делом были переразмечены массивы данных и были выделены только маски (В соответствии с рисунком 15).

Рисунок 15 - Переразметка фотографии в связи с увеличением точности распознавания

Вторая проблема была в ограниченном объеме обучающих данных. Данная проблема была решена способом редактирования уже существующих данных. То-есть, картинки были - перевернуты, масштабированы, подвержены вращению на случайный градус, увеличена яркость, насыщенность, убраны цвета, подвержены размытию.

Третья проблема заключалась в том, что модель имеет низкую точность классификации. После того как была создан набор данных для классификации, было обнаружено то, что набор данных не очень сбалансирован. То-есть очень много изображении с правильно надетыми масками и людьми которые прямо смотрят в камеру. А фотографии людей без масок или же с неправильно надетыми масками не хватало (рис. 7). По этой причине пришлось фотографировать окружающих и делать новые фотографии людей, чтобы как то сбалансировать набор данных. Был так же найден дополнительный набор данных и так же интегрирован в имеющийся набор.

YOLO-v3, YOLO-v4, все предсказываются модулем прогнозирования после извлечения трех слоев объектов. Для эффективного слоя объектов 13 × 13 × 24 это эквивалентно разделению входного изображения на сетки 13 × 13, и каждая сетка будет отвечать за обнаружение объекта в области, соответствующей этой сетке. Когдацентр объекта попадает в эту область, необходимо использовать эту сетку, чтобы взять на себя ответственность заобнаружение объекта. Каждая сетка будет предварительно настроена на три предыдущих поля и результаты прогнозирования сети скорректирует параметры положения трех предыдущих блоков для получения окончательных результатов прогнозирования. Аналогично, процесс прогнозирования эффективных слоев

объектов размером 26 × 26 × 24 и 52 × 52 × 24 - это то же самое, что и у слоев объектов размером 13 × 13 × 24.

Слой объектов разделен на сетки 13 × 13, чтобы проиллюстрировать процесс определения местоположения объекта и прогнозирования. (В соответствии с рисунком 16). На рисунке 18a представляется исходное входное изображение трехканального цвета с размером 416 × 416 × 3. На рисунке 18b получен в результате извлечения объектов входного изображения через сеть, которое представляет эффективный слой объектов размером 13 × 13 × 24 в модуле прогнозирования. Слой объектов разделен на сетки 13 × 13, и каждая сетка имеет три предыдущих поля, которые представлены зелеными полями.

Их центральные точки - cx и cy, ширина и высота - pw и ph соответственно. Поле окончательного прогноза представляет собой синее поле с центральными точками tx и ty, шириной и высотой bw и bh соответственно. На рисунке 18A представляется собой входное изображение, сопоставленное с рисунком 18B, что означает, что размер предыдущего блока, точки сетки, блока прогнозирования, высота и ширина на рисунке 18B в 32 раза больше, чем на рисунке 18C. Поэтому, когда центр лица в маске нерегулярно попадает в оранжевую рамку, эта сетка отвечает за распознавание лиц. Результаты прогнозирования сети скорректируют позиции трех предыдущих блоков, а затем окончательного блока прогнозирования будут отсеяны путем ранжирования уровня достоверности и NMS для получения рисунка 18C в качестве результата обнаружения сети.

Если брать в качестве основы YOLO-v4 - это улучшенная версия, основанная на YOLO-v3, которая решает многомасштабные проблемы объектов и улучшает эффект обнаружения сети на объектах малого масштаба.

При обучении передаче мы обычно стремимся построить модель таким образом, чтобы удалить последний слой, чтобы использовать его в качестве средства извлечения объектов. Архитектуры, в которых не существует уровня объединения, называются полностью сверточными сетями (FCN). Архитектура, которая используется в YOLO v3, называется DarkNet-53. Он также упоминается как магистральная сеть для YOLO v3. Его основная задача - выполнять извлечение объектов. Он имеет 53 слоя извилин. Здесь нет максимального объединения. Для каждой операции свертки у нас есть свертка, за которой следует пакетная нормализация и утечка RELU. В более ранних версиях YOLO у нас не было пакетной нормализации и мы использовали максимальное объединение, из-за чего результаты были невелики. Причина в том, что пакетная нормализация гарантирует, что даже когда вы находитесь глубоко в сети, ваши входные данные нормализуются. Кроме того, эти свертки являются свертками stride-2. Поскольку максимальное объединение не работало хорошо, и нам нужно было что-то для уменьшения выборки наших карт фильтров, поэтому используются пошаговые свертки. См. Упомянутый размер 3 x 3/2, здесь / 2 представляет собой свертки stride-2. Шаг будет направлять пространство между каждым образцом в операции с пиксельной сеткой.

Например, если шаг равен двум, между каждым последующим образцом будет два пикселя.

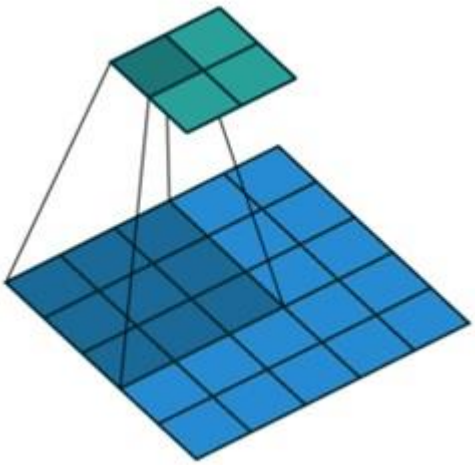

Рисунок 16 – Принцип работы слоев объектов [31]

Свертки Stride-2 уменьшают ввод вдвое, т.е. если input_size = 256 x 256, то output_size = 128 x 128. Итак, fun / 2 представляет собой свертки stride-2, в результате чего размер изображения уменьшается вдвое. Если вы посмотрите на всю архитектуру, то за сверточными слоями здесь следует остаточное соединение или сеть. По сути, это означает, что из предыдущих блоков у меня здесь есть остаточные соединения (остаточные соединения поступают из ресетей). Когда у меня очень глубокая нейронная сеть, остаточные или пропущенные соединения помогают мне избежать переобучения. Кроме того, 1x, 2x, 3x означают, сколько раз этот конкретный блок повторялся в реальной архитектуре. Из-за этих повторяющихся блоков общее количество сверточных слоев в архитектуре составляет 53 [32]. А также, для каждого блока у нас есть остаточное или пропущенное соединение, которое происходит из выходного сверточного слоя предыдущего блока. Ранее авторы использовали DarkNet-19 в YOLOv3, который работал не так хорошо, как мы хотели. И в конце у нас есть полностью подключенный слой со средним пулом / средним пулом и слой softmax. Что сделали авторы, так это то, что они предварительно обучили всю эту модель на наборе данных Imagenet, чтобы все эти веса были хорошо настроены для распознавания объектов [32] Мы используем Imagenet только для того, чтобы он мог сказать, есть ли объект на изображении или нет. Следовательно, во-первых, они ищут только объекты на изображении, а не для создания ограничивающих рамок или чего-то еще. Позже они используют некоторые хаки, чтобы выполнить свою работу. (В соответствии с рисунком 17)

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Рисунок 17 - Darknet-53 [8]

В то же время YOLO-v3 использует двоичную перекрестную энтропию в качестве функции потерь, так что сеть может реализовывать многокатегорийное предсказание с помощью одного граничного блока. YOLO-v3 и Методы прогнозирования YOLO-v4 используются в процессе прогнозирования в настоящей статье, как показано на рисунке 18B, a tx, ty, tw и th - это четыре параметра, которые необходимо изучить сети, а именно

$$b_x = \sigma(t_x) + c_x \quad (1)$$

$$b_y = \sigma(t_y) + c_y \quad (2)$$

$$b_w = p_w e^{tw} \quad (3)$$

$$b_h = p_h e^{th} \quad (4)$$

В процессе обучения сеть постоянно изучает четыре параметра t_x, t_h, t_yи t_w, таким образом, постоянно корректируя положение предыдущего блока, чтобы приблизиться к положению блока прогнозирования, и, наконец, получая окончательный результат прогнозирования. $\sigma(t_x)$ и $\sigma(t_y)$, соответственно, представляют, что t_x и t_y ограничены сигмоидной функцией, чтобы гарантировать, что центр блока прогнозирования попадает в сетку.

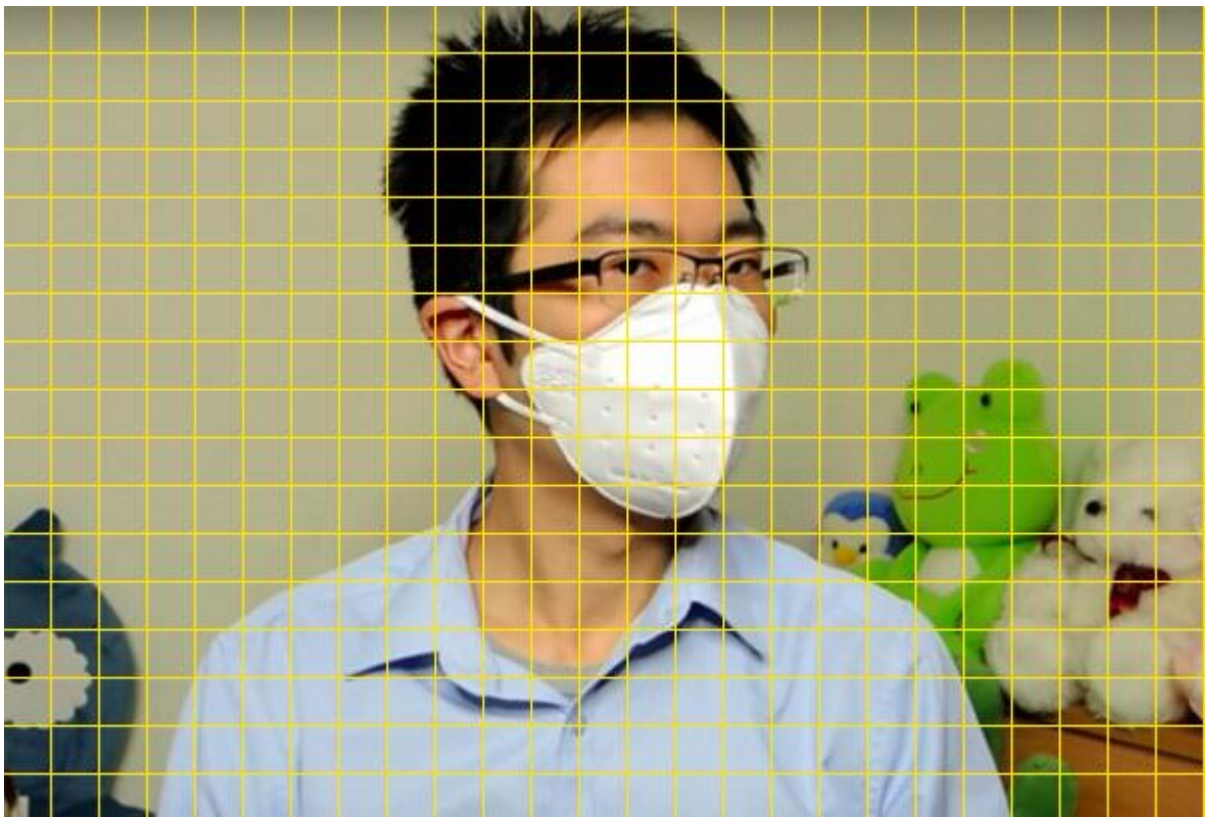

Рисунок 18A – Исходное изображение

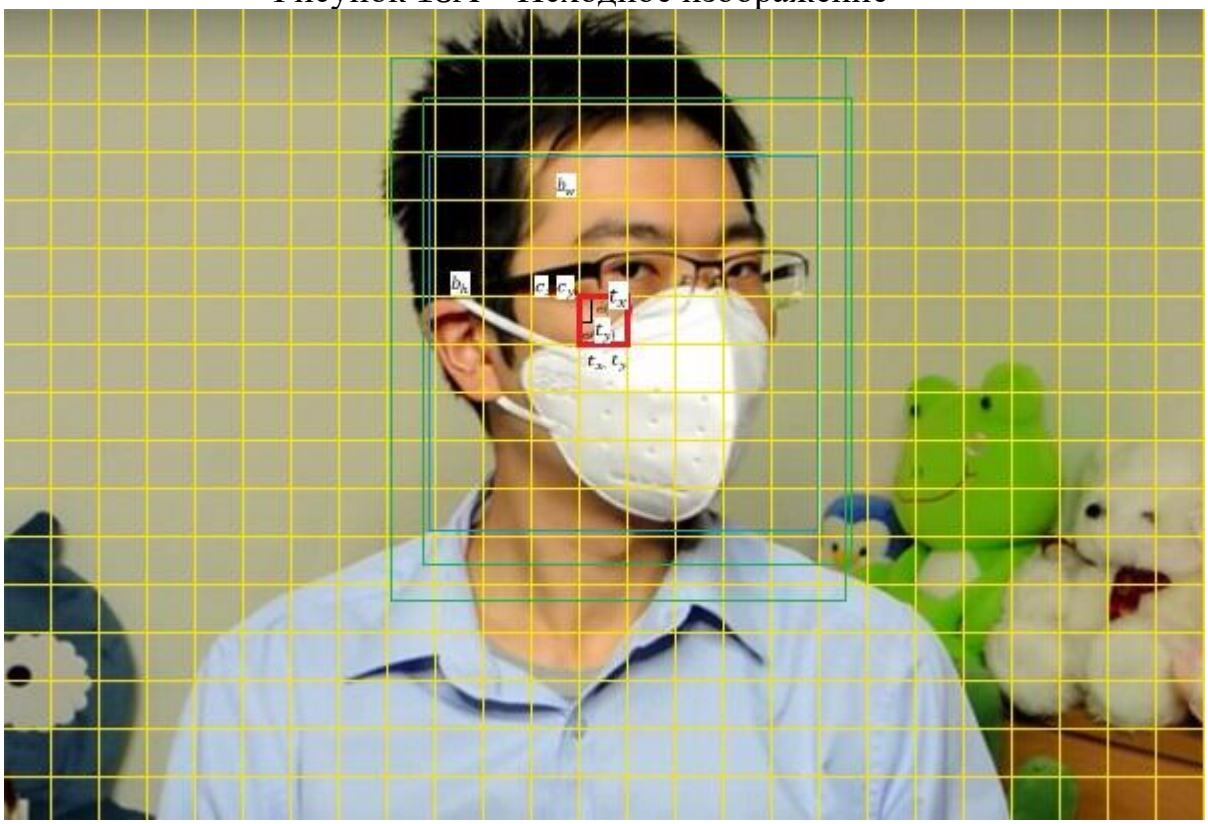

Рисунок 18B – Размеченное изображение

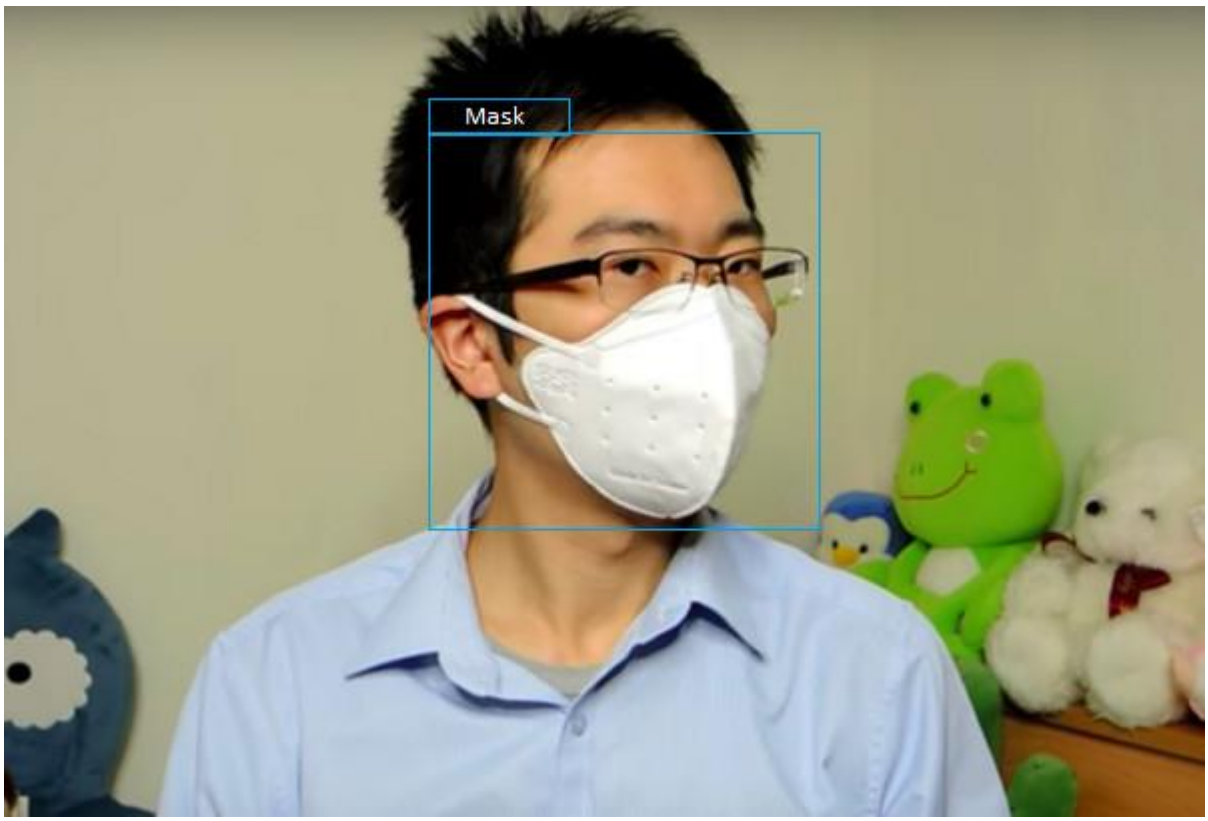

Рисунок 18С – Результат

2.3. Работа с YOLO v3 после исправления проблем классификации

Мы использовали многомасштабное обучение, чтобы сделать нашу модель более надежной при различных входных масштабах. Как показано на рисунке 8, сеть случайным образом выбирает шкалу m каждые 10 партий на этапе обучения, m ∈ [320, 352, 384, 416, 448, 480, 512, 544, 576, 608], и изменяет размер входного изображения на m × m. Затем изображение делится на сетки S × S. Сетки отвечают за обнаружение объектов, центр ограничивающей рамки которых расположен в них. После этого извлекаются карты объектов из входного изображения. Модуль Feature Piramid Network увеличивает выборку отображений выходных объектов и объединяет их с выводом из предыдущего сверточного слоя. Наконец, размеры карты объектов, выводимой моделью, составляют 13 × 13, 26 × 26 и 52 × 52 (В соответствии с рисунком 19).

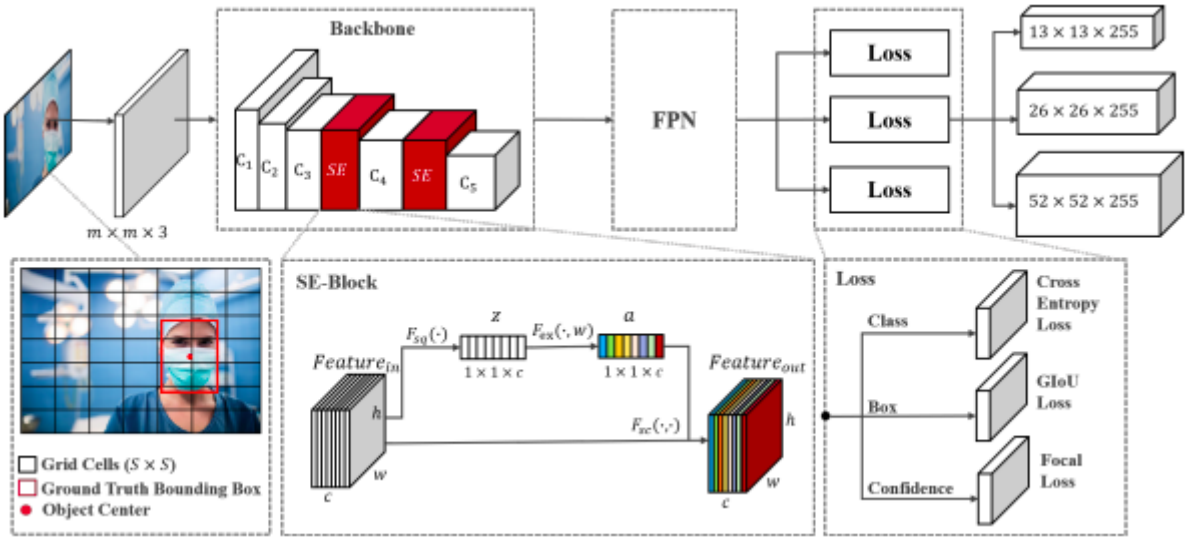

Рисунок 19 - Сетевая архитектура SE-YOLOv3. Каждый сверточный блок Darknet 53 определяется как C1-C5

2.3.1. Увеличение объема данных

Методы увеличения данных также, по-видимому, улучшают модели обнаружения объектов, хотя они улучшают одноступенчатые детекторы больше, чем многоступенчатые детекторы. Согласно (1), причина этого заключается в том, что в многоступенчатых детекторах, таких как Faster-R-CNN, где определенное количество предложений объектов-кандидатов отбирается из большого пула сгенерированных ROI, результаты обнаружения получаются путем многократного обрезания соответствующих областей на картах объектов. Благодаря этой операции обрезки многоступенчатые модели заменяют операцию случайной обрезки входных изображений, следовательно, эти сети не требуют обширных геометрических дополнений, применяемых на этапе обучения.

Эмпирически методы увеличения, такие как случайная обрезка (с ограничениями), расширение, горизонтальный поворот, изменение размера (со случайной интерполяцией) и дрожание цвета (включая яркость, оттенок, насыщенность и контраст), работают лучше во время обучения. Во время тестирования размер изображений просто изменяется путем случайного выбора одного из популярных методов интерполяции, а затем нормализации. Чтобы предотвратить переоснащение модели, важно улучшить обобщаемость за счет использования крупномасштабных выборок в соответствии с принципом минимизации вицинального риска (VRM) [33]. Поскольку сбор и аннотирование данных требуют значительных затрат ручного труда, мы проанализировали данные с двух сторон на основе существующих данных.

Для увеличения изображения мы начали с самого отдельного изображения. В соответствии с характеристиками данных о лице мы выполнили горизонтальное переворачивание, произвольную обрезку и корректировку контрастности изображения, чтобы увеличить преобразование пространственного положения цели.

Мы применили mixup [34], метод увеличения данных, первоначально применявшийся в задаче классификации. В этой задаче обнаружения были объединены метки (т.е. ограничительные рамки базовой истинности) двух обучающих выборок напрямую вместо того, чтобы устанавливать для них соотношение. Были обозначены x_i и x_j как два изображения из обучающего набора данных. y_i и y_j являются основными рамками истины, ограничивающими эти два изображения.

где $\tilde{x}$ относится к дополненному изображению, смешанному из x_i и x_j. $\tilde{y}$ ссылается на список основных ограничивающих рамок истинности объектов, объединенных из них. λ является параметром, полученным из Beta(α, β) распределения, $\lambda \in [0, 1]$. Согласно эксперименту в [35] α и β были установлены на 1,5. (В соответствии с рисунком 20)

Для расчета общих потерь (L) для смешанной выборки мы использовали взвешенную сумму потерь при обнаружении объектов на двух изображениях. Вес потери - это исходное число от 0 до 1 в соответствии с коэффициентом смешивания изображений, к которым изначально принадлежат объекты. Были смешаны два изображения с соотношением от Бета (1,5, 1,5), чтобы улучшить способность сети к помехам в пространственном измерении. Расчет взвешенных потерь может быть сформулирован в виде уравнения (11).

$$L = \lambda L_i + (1 - \lambda) L_j \quad (5)$$

Рисунок 20 – Увеличение объема обучающих данных

2.3.2. График темпа изображения

Большинство популярных сетей обнаружения объектов (Faster R-CNN, YOLO и т.д.) Используют планировщик скорости обучения. Согласно (5), результирующий резкий переход скорости обучения может привести к тому, что оптимизатор повторно стабилизирует импульс обучения на следующих итерациях. Использование косинусного планировщика (где скорость обучения уменьшается медленно) с надлежащим прогревом (две эпохи) может обеспечить даже лучшую точность проверки, чем использование пошагового планировщика, показанного ниже. (В соответствии с рисунком 21)

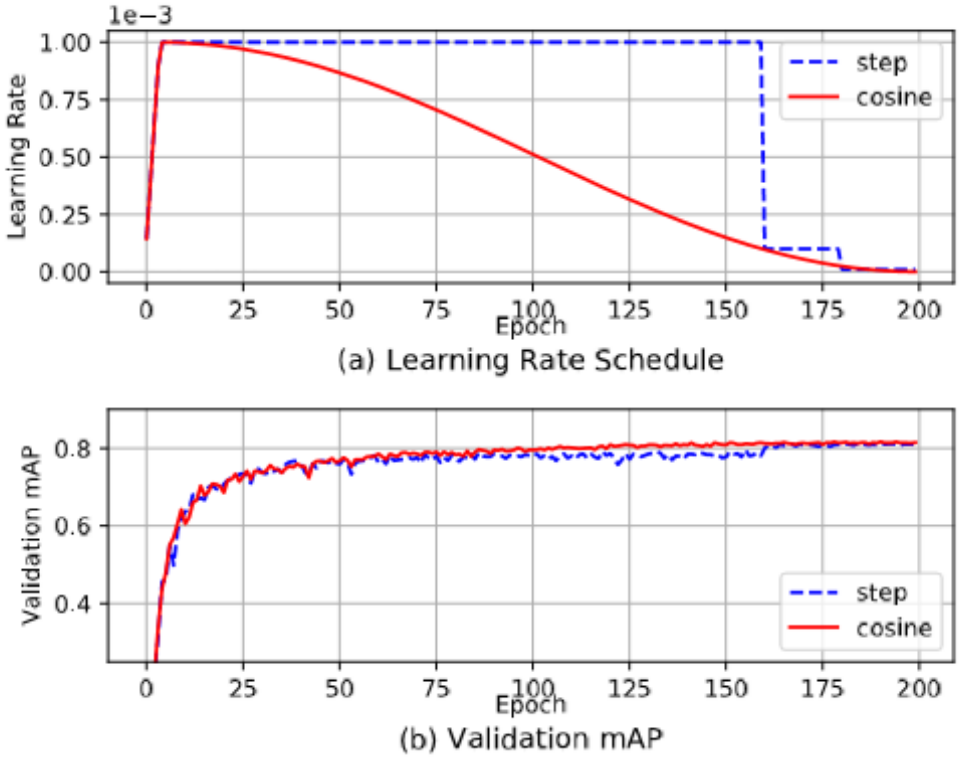

Рисунок 21 – Использование конусного планировщика в задачах улучшения точности обнаружения объектов

2.3.3. Синхронизированная Пакетная Нормализация

В современных архитектурах с глубокой сверточной обработкой пакетная нормализация считается важным уровнем. Он отвечает за ускорение процесса обучения и делает сеть менее чувствительной к инициализации веса за счет нормализации активаций скрытых слоев. Из-за большого размера входного изображения, наличия архитектуры feature pyramid и большого количества предложений объектов-кандидатов (в случае многоступенчатых сетей) размеры пакетов, которые можно разместить на одном графическом процессоре, становятся очень маленькими (т.Е. менее 8 или около того изображений на пакет).

Пакетная нормализация данных является на сегодняшний день одной из самых важных задач в сфере компьютерного зрения. Так-как пакетная нормализация или Batch Normalization будет использоваться для ускорения

обучения нейросети. Для данной задачи были использованы инструменты ускорения обучения нейронной сети.

При использовании Batch normalization было выявлено что сеть в среднем обучилась быстрее в 15 раз. Сравнение моделей типов нормализации приведено в таблице 2.

Таблица 2 - Для начального и пакетного нормализованных вариантов, количество шагов обучения, необходимых для достижения максимальной точности начального (72,2%). и максимальная точность, достигнутая с помощью сетей нормализации.

Модель	Шаги до 72,2%	Максимальная точность
Inception	$31.0 * 10^6$	72.2%
BN-Baseline	$13.3 * 10^6$	72.7%
BN-x5	$2.1 * 10^6$	73.0%
BN-x30	$2.7 * 10^6$	74.8%
BN-x5-Sigmoid		69.8%

Для более глубокого понимания пакетной нормализации нейросети, можно взять за основу нейросеть с несколько большими слоями.

В итоге, результатом имеется средняя дисперсия. Данный параметр является элементом обратного распространения ошибки.

В своей основе, оригинальная сеть выглядит как 8 сверточных слоев, имеющие размерность по стандарту 3x3. После использования каждой свертки, используется ReLU: max(x,0) после данных ReLU используется max-pooling размерности 2x2. Конечный слой pooling размером 7x7 усредняет значения, но не берет максимум. Итогом является массив 1x1x320. (В соответствии с рисунком 22)

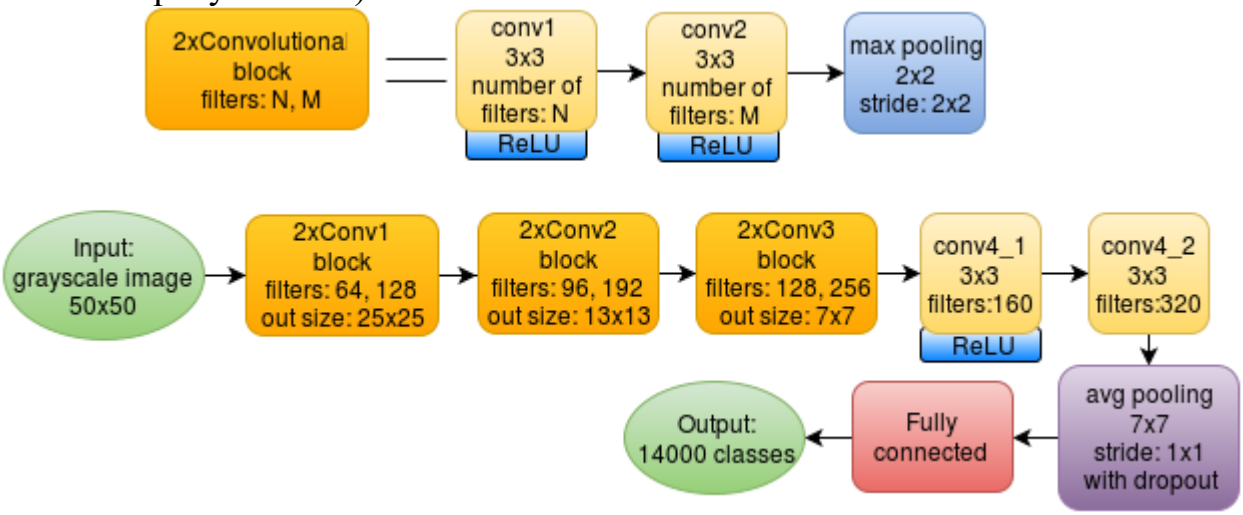

Рисунок 22 - Оригинальная сеть [36]

Сеть пакетной нормализации более усложнена. В данной архитектуре между каждыми свертками есть слой пакетной нормализации. (В соответствии с рисунком 23)

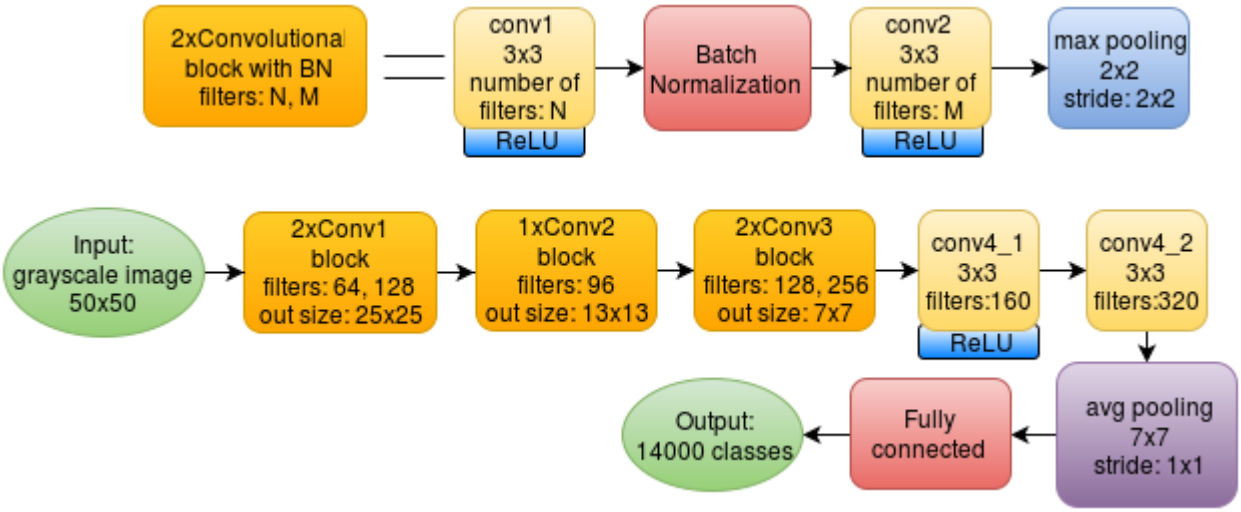

Рисунок 23 - Сеть пакетной нормализации [36]

В парадигме распределенного обучения скрытые активации нормализуются внутри каждого графического процессора. Это приводит к вычислению зашумленных оценок среднего значения и дисперсии, что затрудняет весь процесс пакетной нормализации. Поэтому было предложено синхронизировать нормализацию пакетов, чтобы помочь увеличить размер пакета за счет учета активаций на нескольких графических процессорах для вычисления статистических оценок. В результате это делает вычисления менее шумными.

Синхронизированная пакетная нормализация может быть легко достигнута с помощью библиотеки Apex от NVIDIA для смешанной точности и распределенного обучения в PyTorch. Мы также можем преобразовать любой стандартный модуль BatchNorm в PyTorch в SyncBatchNorm, используя метод convert_syncbn_model, который рекурсивно обходит переданный модуль и его дочерние элементы, чтобы заменить все экземпляры torch.nn.modules.batchnorm._BatchNorm с вершиной.параллельный.SyncBatchNorm, где вершина.параллель.SyncBatchNorm - это модуль PyTorch для выполнения синхронизированной пакетной обработки на графических процессорах NVIDIA.

2.3.4. Адаптивное пространственное слияние пирамид объектов

Сети обнаружения объектов, использующие пирамиды объектов, делают прогнозы в разных масштабах объектов или путем объединения различных масштабов объектов. Например, YOLOv3 делает прогнозы в трех разных масштабах с шагом 32, 16 и 8. Другими словами, если задано входное изображение размером 416 x 416, оно делает прогнозы в масштабах 13 x 13, 26 x 26 и 52 x 52.

Объекты с низким разрешением имеют высокую семантическую ценность, в то время как объекты с высоким разрешением имеют семантически низкую ценность. Карты объектов с низким разрешением также содержат ячейки сетки, которые охватывают большие области изображения и, следовательно, более подходят для обнаружения более крупных объектов. Напротив, ячейки

сетки на картах объектов с более высоким разрешением лучше подходят для обнаружения объектов меньшего размера. Это означает, что обнаружение объектов разного масштаба с использованием объектов только одного масштаба затруднено. Чтобы устранить эту проблему, обнаружение может выполняться в разных масштабах по отдельности для обнаружения объектов разного масштаба, как в архитектуре single shot detector (SSD). Однако, хотя этот подход требует небольших дополнительных вычислительных затрат, он по-прежнему неоптимален, поскольку карты объектов с высоким разрешением не могут в достаточной степени получить семантические признаки из изображений. Архитектуры, такие как RetinaNet, YOLOv3 и т.д. поэтому объедините как объекты с высокой, так и с низкой семантической ценностью, чтобы создать семантически и пространственно сильный объект. Выполнение обнаружения по этим функциям обеспечивает лучший компромисс между скоростью и точностью.

Объединение различных функций разрешения осуществляется путем объединения или добавления их по элементам. Некоторые предложили подход к объединению этих карт объектов таким образом, чтобы для объединения сохранялась только соответствующая информация из каждой карты объектов масштаба (2). На приведенном ниже рисунке это суммируется. Короче говоря, вместо того, чтобы делать прогнозы для объектов на каждом уровне, как в стандартном YOLOv3, объекты с трех уровней сначала масштабируются, а затем адаптивно объединяются на каждом уровне, а затем выполняется прогнозирование / обнаружение для этих новых объектов. (В соответствии с рисунком 24)

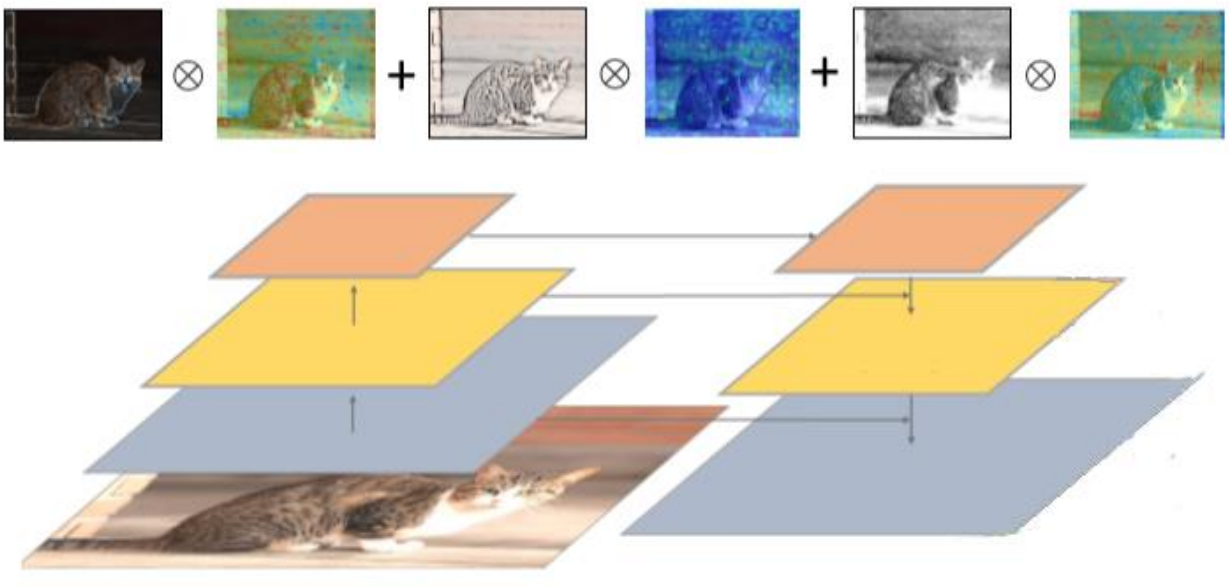

Рисунок 24 – Слияние пирамид объектов

2.4. Работа с безсерверными вычислениями

Разработка модели глубокого обучения в браузере заключается в использовании TensorFlow.js или ONNX.js. Эти методы используют специализированную библиотеку JavaScript для чтения файла модели и завершения вывода путем вызова вычислительных операций, написанных на JavaScript.

Однако JavaScript не является типичным языком программирования в области глубокого обучения и по этой причине сообщество для поиска информации очень небольшое.

- ONNX.js : Для использования ONNX.js , сначала нам нужно преобразовать существующую модель PyTorch в модель ONNX (Open Neural Net- work Exchange). ONNX определяет различные стандартные операторы в машинном обучении и глубоком обучении как открытый формат, чтобы облегчить разработчикам использование ONNX в качестве ретранслятора для преобразования моделей из одной среды в другую. ONNX поддерживает PyTorch, TensorFlow, Caffe2, NCNN и другие распространенные фреймворки глубокого обучения. ONNX.js это библиотека JavaScript, которая может напрямую считывать модель ONNX в среде JavaScript для вывода. Однако существуют ограничения для завершения развертывания в браузере с помощью ONNX.js : (1) Он не поддерживает переменные формата INT64 в модели. С тех пор, как ONNX.js выполняется в среде JavaScript, а JavaScript не поддерживает переменные формата INT64. Модель ONNX, непосредственно экспортируемая PyTorch, содержит множество переменных в формате INT64. Поэтому нам необходимо преобразовать компонент INT64 в формат INT32 в модели ONNX. Тем не менее, даже после замены всех переменных на формат INT32 модель ONNX по-прежнему работает некорректно. Некоторые собственные операторы ONNX поддерживают только INT64 в качестве входных данных для узла, например ConstantOfShape. Официальный интерпретатор модели ONNX больше не поддерживает модифицированный оператор [37]. Многие операторы не поддерживаются. ONNX.js необходимо прочитать модель и вызвать ее встроенные операции JavaScript на основе содержимого модели, что означает, что ONNX.js библиотека должна была оштрафовать все операторы перед их выполнением. Потому что ONNX.js это новая ниша, поддерживающих ее операторов относительно немного. Например, операция изменения размера не поддерживается. Учитывая быстрое появление новых моделей и идей в настоящее время, это является существенным ограничением для развертывания моделей. (В соответствии с рисунком 25)

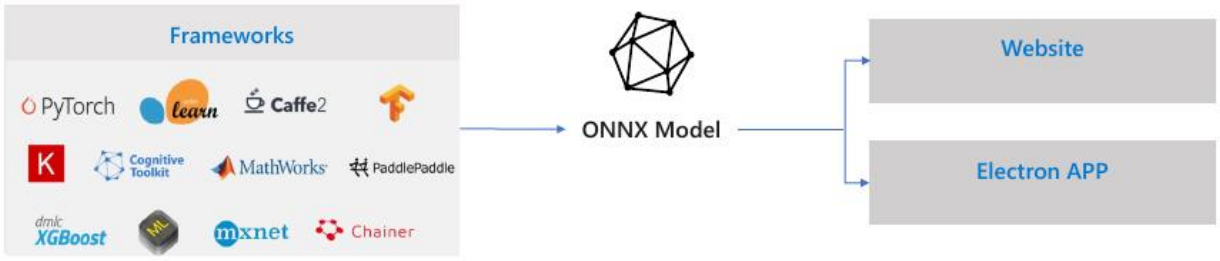

Рисунок 25 – Поддерживаемые фреймворки ONNX.js [37]

2) TensorFlow.js : По сравнению с ONNX.js , TensorFlow.js более широко используется и поддерживает более богатый набор операторов. Для использования TensorFlow.js , мы сначала преобразуем модель в формат TensorFlow SavedModel. Поскольку нет официального способа преобразовать модель PyTorch в TensorFlow, мы адаптируем ее к ONNX модель, а затем будут проводится преобразования ее в сохраненную модель TensorFlow. Затем нужно преобразовать получившуюся модель в конкретную читаемую модель веб-формата, которая может быть прочитана TensorFlow.js в среде JavaScript. Сложные цепочки преобразования означают более низкую надежность и больше ограничений. При его преобразовании он, вероятно, введет в модель некоторые необычные операторы. (В соответствии с рисунком 26)

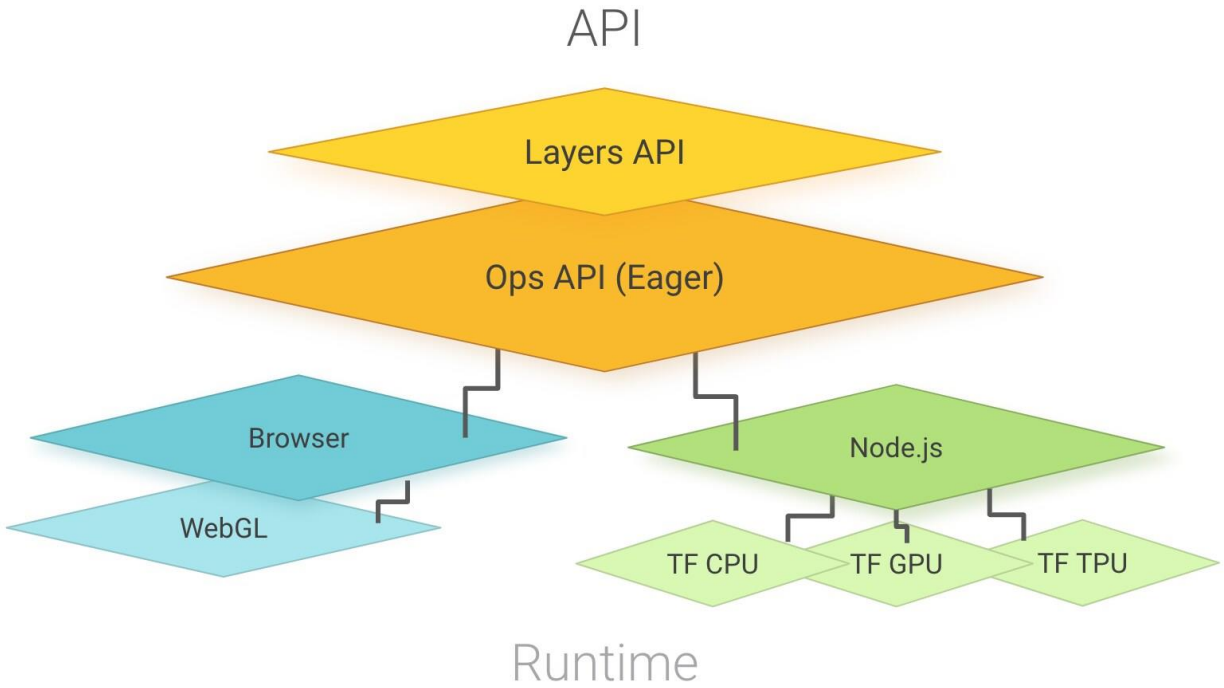

Рисунок 26 - Архитектура Tensorflow.js [38]

Основные проблемы с Tensorflow.js:

Разные условия. Одна из проблем JS заключается в том, что он работает в разных средах. Вычисления могут выполняться на стороне клиента в браузере, на стороне сервера, в первую очередь как часть Node.js фреймворк, а в последнее время и на рабочем столе с помощью таких фреймворков, как

Electron. TensorFlow.js предназначен для работы во всех этих настройках, хотя большая часть нашей работы на сегодняшний день заключалась в настройке его для разработки на стороне клиента в веб-браузере.

Представление. Второй ключевой проблемой, специфичной для среды браузера, является производительность. JS - это интерпретируемый язык, поэтому он обычно не соответствует скорости скомпилированного языка, такого как C++ или Java, для численных вычислений[38]. В отличие от Python, который может привязываться к библиотекам C++, браузеры не предоставляют эту возможность. По соображениям безопасности браузерные приложения не имеют прямого доступа к графическому процессору, который обычно используется для численных вычислений в современных системах глубокого обучения.

Для решения этих проблем с производительностью появляется несколько новых стандартов JS. Одним из примечательных решений является WebAssembly Haas и др. (2017), метод компиляции программ на C++ в байт-код, который может интерпретироваться и выполняться непосредственно в браузере. Для определенных задач WebAssembly может превзойти обычный JS. Большинство современных браузеров также поддерживают WebGL Kronos, API, который предоставляет OpenGL для JS. OpenGL - это межъязыковой кроссплатформенный API для рендеринга 2D и 3D векторной графики, позволяющий выполнять игры и другие высокопроизводительные задачи рендеринга непосредственно на веб-странице [38]. На стороне сервера библиотеки JS могут связываться с существующими собственными модулями, написанными на C и C++, с помощью Node.js интерфейс N-API

Кроссбраузерная совместимость. JS разработан как кроссплатформенный язык, поддерживаемый всеми основными браузерами со стандартизированными веб-API, которые упрощают написание приложений, работающих на всех платформах. На практике браузеры создаются несколькими разными поставщиками с немного разными реализациями и приоритетами. Например, в то время как Chrome и Firefox поддерживают WebGL 2.0, Safari от Apple остановилась на WebGL 1.0 [38]. Разработчикам веб-приложений приходится много работать, чтобы скрыть это несоответствие в своих приложениях, часто требуя обширной инфраструктуры тестирования для тестирования на большом количестве платформ.

3. Проведение экспериментов

Были проведены сравнения 5 моделей YOLO в таблице 3.

Таблица 3 – Сравнение моделей YOLO

Модель	Время обучения
YOLOv4-tiny	2 часа
YOLOv5s	5 минут и 22 секунды
YOLOv5m	15 минут и 39 секунды

YOLOv5l	15 минут и 46 секунды
YOLOv5x	14 минут и 52 секунды

В таблице 4 указано то, что время обучение у модели YOLOv5s гораздо ниже чем у остальных.

Таблица 4 – Сравнение моделей YOLO

Модель	Точность	Скорость	FPS
YOLOv4-tiny	Высокая	4.0	271
YOLOv5s	Высокая	2.1	476
YOLOv5m	Средняя	3.0	333
YOLOv5l	Средняя	3.9	256
YOLOv5x	Средняя	6.1	164

Разница между моделью YOLOv4-tiny и оригинальным YOLOv4. YOLOv4-tiny — это сжатая версия YOLOv4. Предлагается на базе YOLOv4 упростить структуру сети и уменьшить параметры, чтобы она стала возможной для разработки на мобильных и встроенных устройствах. Архитектура и принцип работы YOLOv4-tiny на рисунках 27, 28.

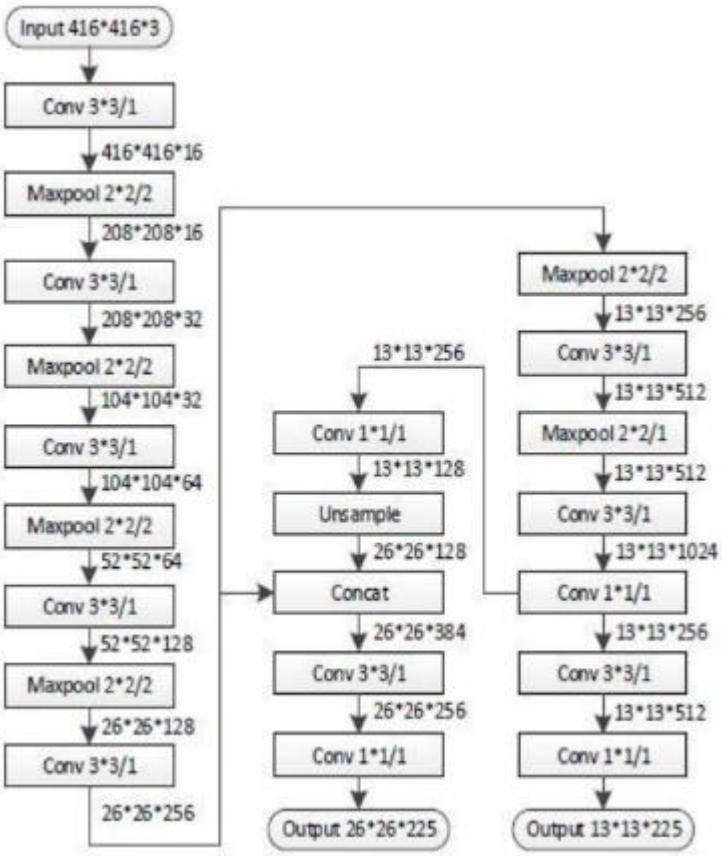

Рисунок 27 - Архитектура работы YOLO v4-tiny [38]

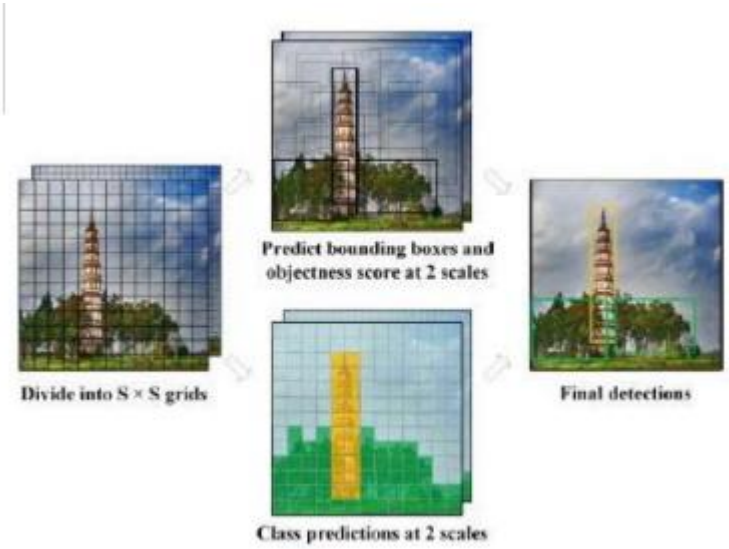

Рисунок 28 - Основной процесс работы YOLO v4-tiny [39]

Мы можем использовать YOLOv4-tiny для более быстрого обучения и более быстрого обнаружения. У него только две головки YOLO, а не три как в YOLOv4, и он был обучен на 29 предварительно обученных сверточных слоях, в отличие от YOLOv4, который был обучен на 137 предварительно обученных сверточных слоях. FPS (кадров в секунду) в YOLOv4-tiny примерно в восемь раз больше, чем в YOLOv4. Однако точность YOLOv4-tiny составляет 2/3 точности YOLOv4 при тестировании на наборе данных MS COCO. Модель YOLOv4-tiny достигает 22,0% AP (42,0% AP50) при скорости 443 FPS на RTX 2080Ti, а при использовании TensorRT, размера партии = 4 и точности FP16 YOLOv4-tiny достигает 1774 FPS.

Для обнаружения объектов в реальном времени YOLOv4-tiny является лучшим вариантом по сравнению с YOLOv4, поскольку более быстрое время вывода важнее, чем точность или аккуратность при работе со средой обнаружения объектов в реальном времени.

После обучения YOLOv4 и YOLOv4-tiny на одном и том же наборе данных из 1500 масок изображений, где средние потери YOLOv4 достигли примерно 0,68 после 6000 итераций, а средние потери YOLOv4-tiny достигли примерно 0,15 после 6000 итераций. Так-как в данном случае, при использовании веб-камеры устройства более важно быстродействие и не самым важным является точность, можно использовать модели которые весят меньше чем полная модель. Точность является не самым важным по причине того, что используется лишь два класса. И в процентном соотношении эти два класса представляют из себя массив чисел от 0 – 1. Если исходный процент меньше 0.5, засчитывается то, что маска не надета. Если больше, то маска сейчас на человеке. Результат обучения модели с помощью YOLOv5s на рисунке 29.

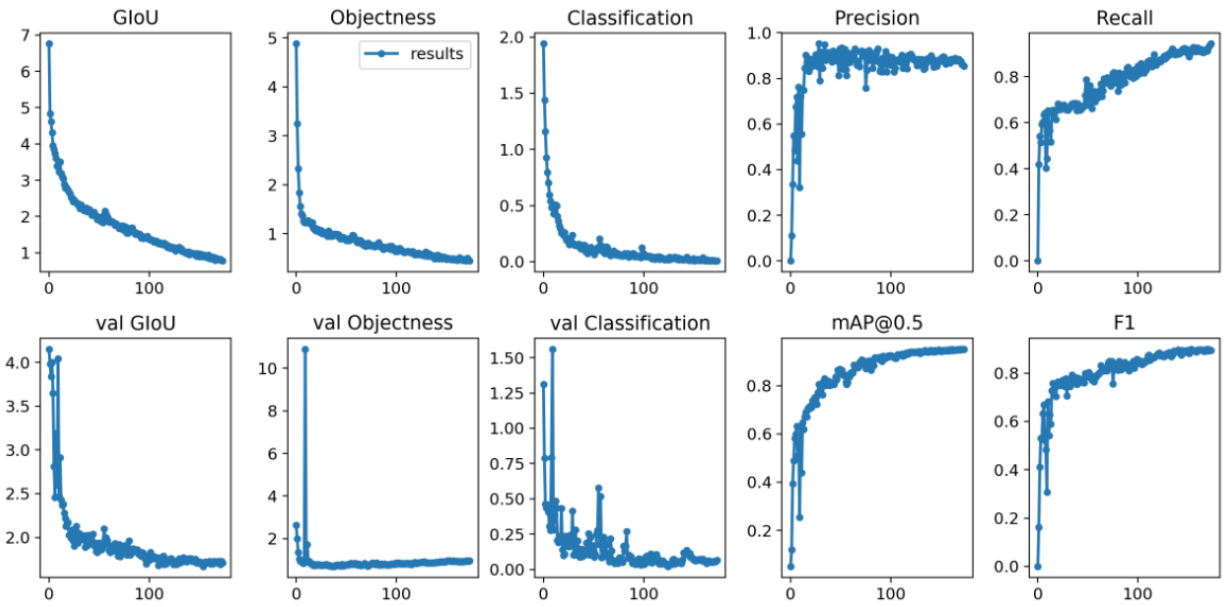

Рисунок 29 - Результат выданный с помощью YOLOv5s

Текущее множество данных было загружено на портал Roboflow для облегчения взаимодействия с ним и автоматической разметкой данных на тестовые и обучающие множества. Еще одной причиной использования Roboflow и создания своей обучающей выборки стало то, что в готовых выборках найденных на просторах интернета, недостаточно много данных. Собрав выборку из найденных в интернете изображении и переразметив многие изображения вручную была получена большая выборка данных для дальнейшего использования. Большое количество данных способствовало увеличению точности метода. В тестовом множестве находится 643 изображения, в наборе для проверки 181 изображение и в тестовом множестве 91 изображение. Количество изображении увеличилось за счет того, что была произведена обрезка на 25%, горизональный переворот изображении, замена цвета от -36º до 36º, размытость на 1px (В соответствии с рисунками 30, 31)

Рисунок 30 - Дополнительные настройки изображении в выборках

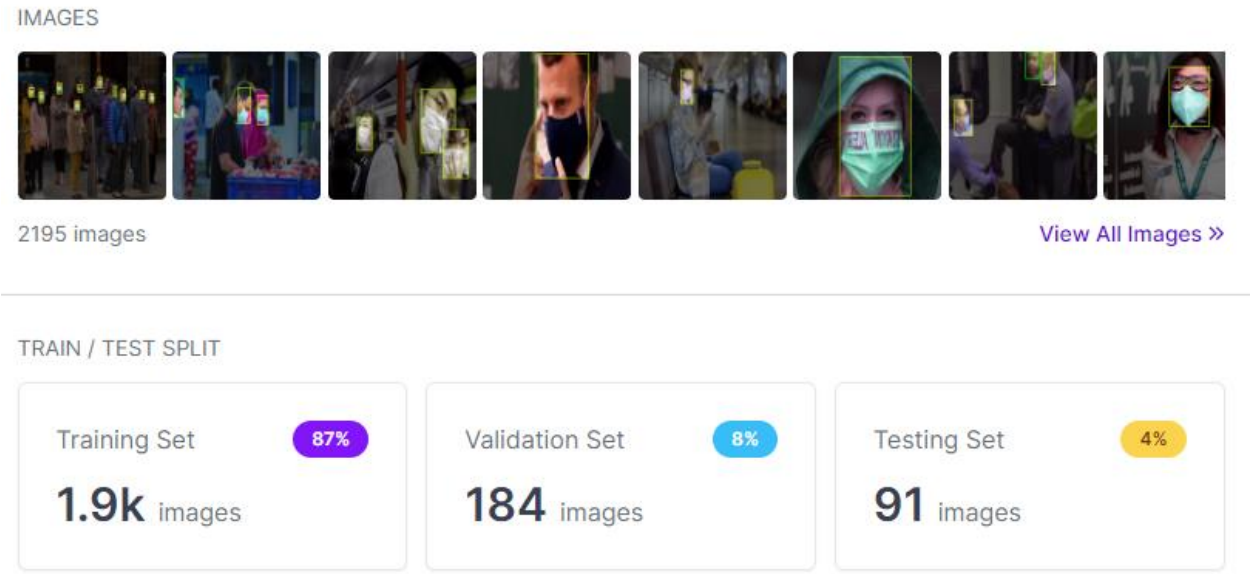

Рисунок 31 - Итоговый размер выборки составил 2195 изображении

Обработка тестовой выборки в 1900 изображении заняло 2 часа 56 минут. В данном проекте использовался Roboflow для удобства загрузки изображении и тренировки моделей. Не требовалось подстраивать отдельно множество изображении в формат Yolo v3, Yolo v4 или в формат Yolo v5s. (В соответствии с рисунком 32)

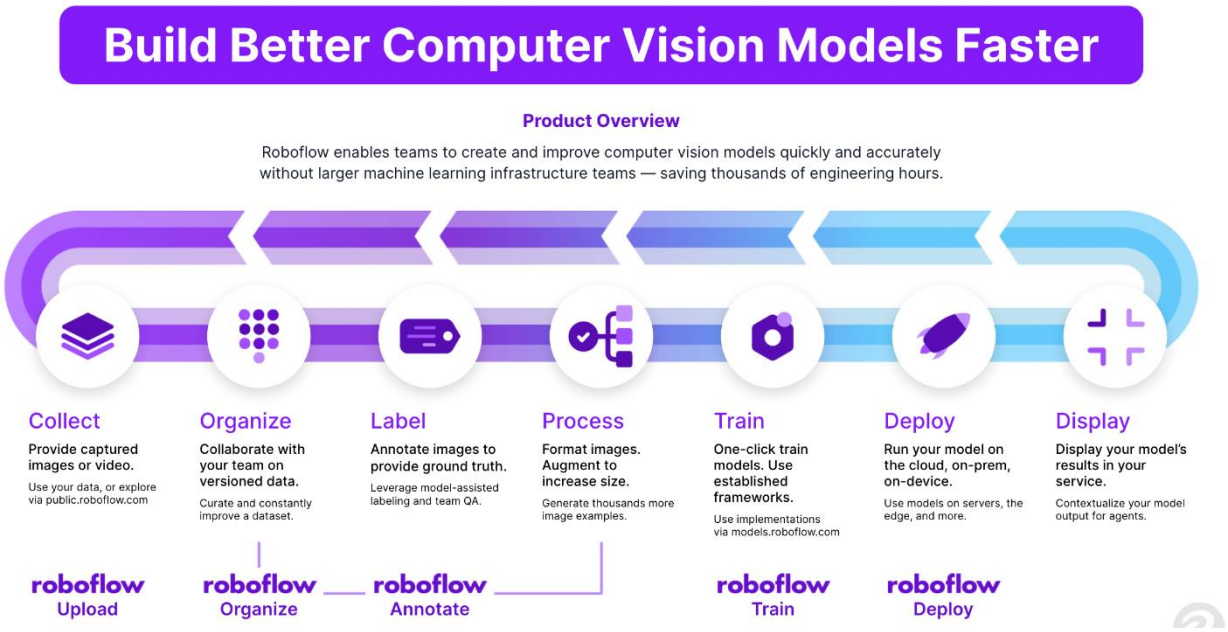

Рисунок 32 – Возможности Roboflow [40]

На Рисунке 33 – Функция потерь YOLO состоит из трех частей:

box_loss — потеря регрессии ограничивающего прямоугольника (среднеквадратичная ошибка).

obj_loss — уверенность в наличии объекта есть потеря объектности

cls_loss — потеря классификации.

Поскольку наши данные имеют только один класс, неверная идентификация классов отсутствует, а ошибка классификации постоянно равна нулю.

Точность измеряет, какая часть предсказаний bbox верна (Истинно положительные / (Истинно положительные + Ложноположительные)), а Отзыв измеряет, какая часть истинных bbox была правильно предсказана (Истинно положительные / (Истинно положительные + Ложноотрицательные)). «mAP_0,5» — это средняя средняя точность (mAP) при пороговом значении IoU (пересечение над объединением), равном 0,5. «mAP_0,5:0,95» — это среднее значение mAP по различным пороговым значениям IoU в диапазоне от 0,5 до 0,95..

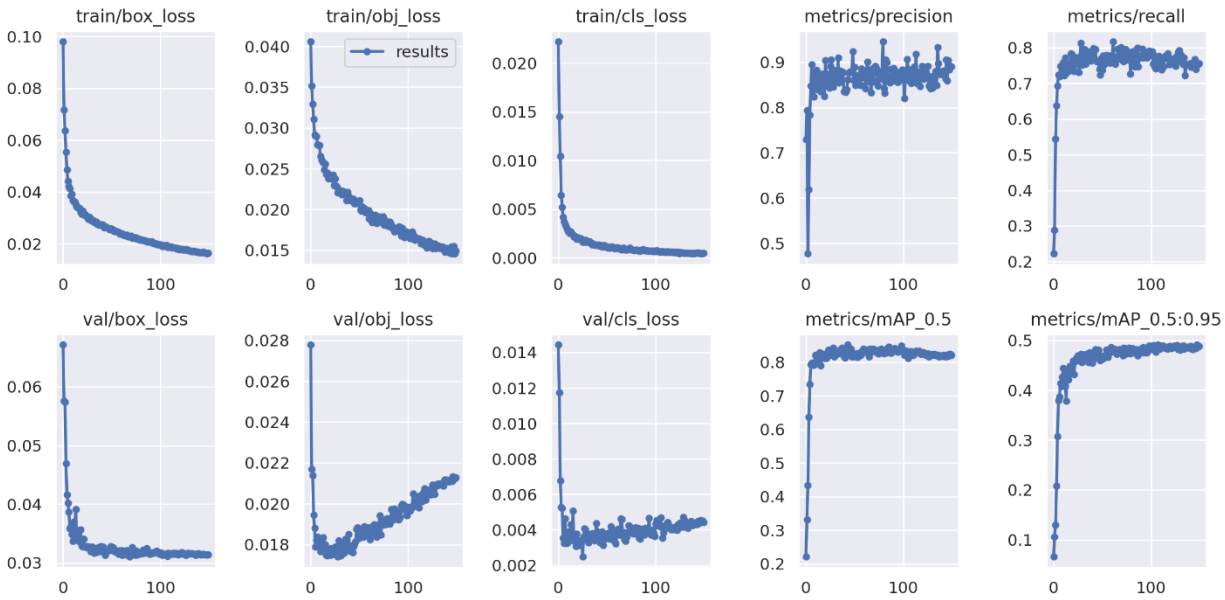

Рисунок 33 – Метрики оценки модели Yolo v5s

На рисунке 34 Recall демонстрирует способность алгоритма обнаруживать данный класс.

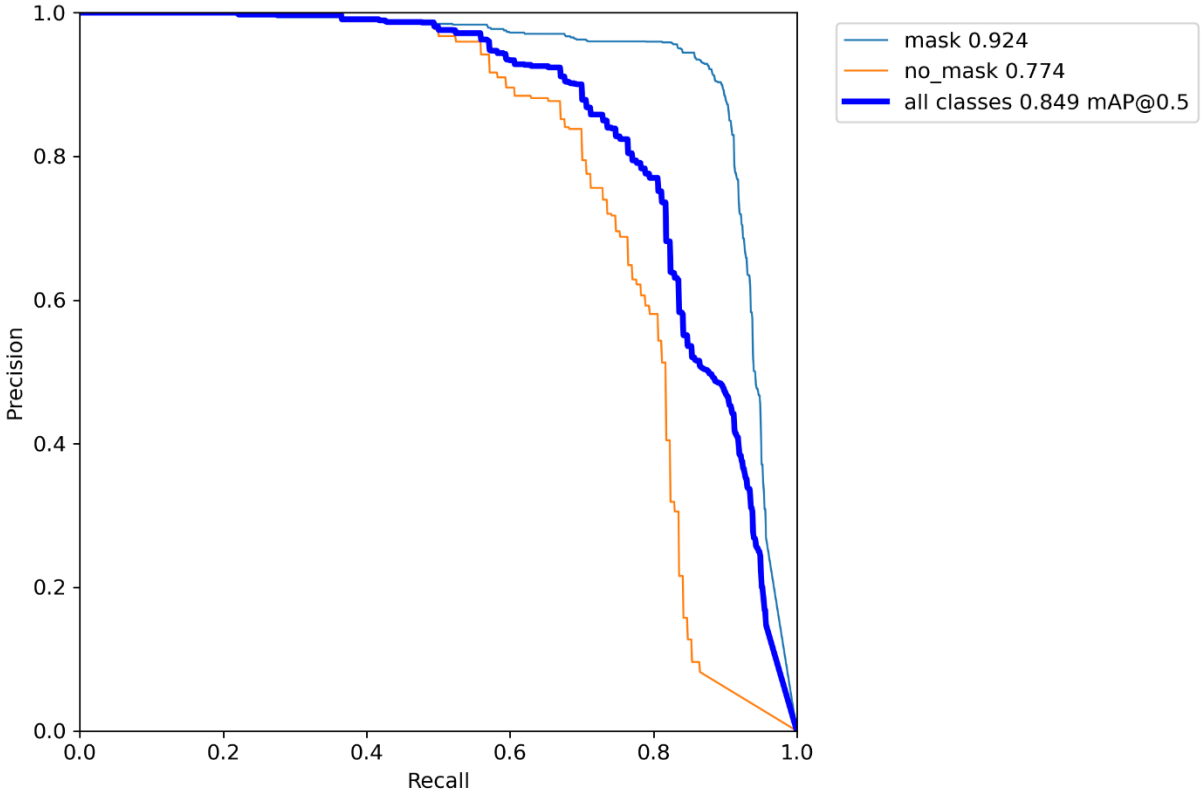

Рисунок 34 - Recall показывает, какую долю объектов положительного класса из всех объектов положительного класса нашел алгоритм.

На рисунке 35 тестирования работы YOLO v5s в рамках тестовых изображении.

Рисунок 35 – Результат работы на Yolo V5 используя модель Yolov5s

На рисунке 36 тестирование конечного продукта на локальном сервере.

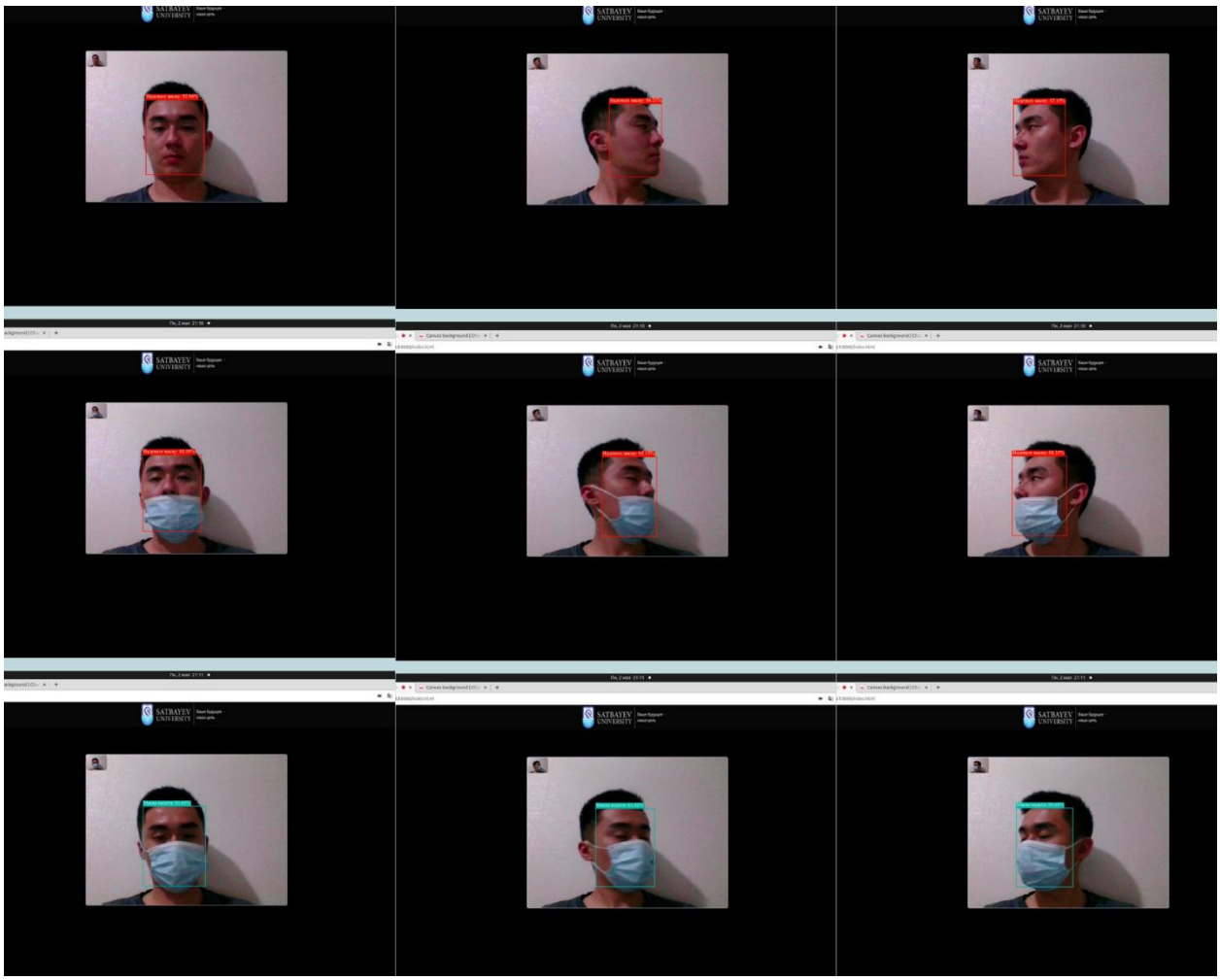

Рисунок 36 – Работа классификатора в режиме реального времени

ЗАКЛЮЧЕНИЕ

По итогу была использована модель YOLOv5s архитектуры YOLO V5. Главным критерием по выбору модели стало то, что скорость обучения была самой быстрой из всех предыдущих и архитектур конкурирующих с YOLO. Благодаря тому, что YOLO является быстрым, получилось сделать классификатор с обработкой на стороне сервера без зависании. Данной архитектуре не понадобилось больших производственных мощностей и самым большим преимуществом стало то, что для работы получившейся модели не требуется GPU. Программа отлично отрабатывает на стороне CPU. Так-как пришлось в итоге создать два визуализатора квадрата для классификации. Один – в случае если на сервере имеется GPU, FPS конечно же выше в данном случае. Но, в случае если имеется только CPU, он так же будет работать.

Получилось сделать это благодаря использованию NCNN. Именно данный фреймворк дал возможность для использования стандартного PyTorch, в режиме быстродействия. Также плюсом стало то, что благодаря использованию NCNN, получилось использовать данную модель на мобильных устроиствах и устройствах без GPU. Стандартный ONXX не мог дать такую возможность.

Так-как сам NCNN не имел возможности для связи с Javascript, потребовалось провести последнюю манипуляцию с моделью. Конвертация с формата NCNN в WASM. По итогу, это облегчило использование модели и классификатор полностью заработал. Для улучшения модели в конечном итоге так же было использование разворота изображения на небольшой градус. Тем самым это нам дало возможность еще больше увеличить точность распознавания.

Перечень принятых сокращений, терминов

ML – Machine Learning (Машинное обучение).
nCov – Временное название вируса Covid-19, Также называется 2019-nCov.
YOLO – Современная система обнаружения объектов в реальном времени.
OpenCV – Один из главных инструментов компьютерного зрения, является библиотекой и содержит в себе алгоритмы для распознавания и работы с изображениями, видео.
Tensorflow – Библиотека машинного обучения. Используется для решения целого ряда задач, но особое внимание уделяется обучению и выводу глубоких нейронных сетей.
DeepFace – Нейросетевая программа меняющая лица в видеороликах.
DL – Deep Learning (Глубокое обучение)
WebGL - кроссплатформенный API для 3D-графики в браузере, разрабатываемый некоммерческой организацией Khronos Group.
JS – Javascript, мультипарадигменный язык программирования. Поддерживает объектно-ориентированный, императивный и функциональный стили.

Список использованных источников

1. Feng S. Rational use of face masks in the COVID-19 pandemic / S. Feng,C. Shen, X. Nan, W. Song, M. Fan, B. J. Cowling. // The Lancet. Respiratory Medicine : электронный журнал. – URL: https://www.thelancet.com/journals/lanres/article/PIIS2213-2600(20)30134-X/fulltext/. – Дата публикации: 20.03.2020. (дата обращения: 11.03.2022)
2. Никульчев Е.В. Облачные технологии / Е.В. Никульчев, О. Лукьянчиков, Д. Ильин. – Российский технологический университет. – Москва : МИРЭА, 2019. – 77 с.
3. Ондирис А. Президент Токаев поручил увеличить количество камер в Алматы / А. Ондирис // Tengrinews.kz. – 2022. – 01 фев. – URL: https://tengrinews.kz/kazakhstan_news/prezident-tokaev-poruchil-uvelichit-kolichestvo-kamer-460793/ (дата обращения: 11.03.2022)
4. Lyu W. Community Use Of Face Masks And COVID-19: Evidence From A Natural Experiment Of State Mandates In The US / W. Lyu, G.L. Wehby // HealthAffairs : электронный журнал. – URL: https://www.healthaffairs.org/doi/10.1377/hlthaff.2020.00818/. – Дата публикации: 16.08.2020. (дата обращения: 11.03.2022)
5. Хадавимогаддам Ф. ПРИМЕНЕНИЕ МЕТОДОВ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА В ПРОГНОЗИРОВАНИИ ОСНОВНЫХ СВОЙСТВ НЕФТИ / Ф. Хадавимогаддам, И.Т. Мищенко, М. Мостаджеран // Нефтегазовые проекты: Взгляд в будущее : электронный журнал. – URL: https://neftegas.info/upload/iblock/b93/b935fb07546371086e84138ae9d47634.pdf. – Дата публикации: 2019. (дата обращения: 11.03.2022)
6. Данилина Е. Ю. РАСПОЗНАВАНИЕ ЛИЦ С ПОМОЩЬЮ МАШИННОГО ОБУЧЕНИЯ / Е. Ю. Данилина, А. А. Ситникова // Издательство Тюменского государственного университета . – Томск, 2018. – С. 162-170.
7. Redmon J. You Only Look Once: Unified, Real-Time Object Detection / J. Redmon, S. Divvala, A. Farhadi // Computer Vision and Pattern Recognition : электронный журнал. – URL: https://arxiv.org/pdf/1506.02640.pdf. – Дата публикации: 09.05.2016. (дата обращения: 11.03.2022)
8. Redmon J. YOLOv3: An Incremental Improvement / J. Redmon, A. Farhadi // University of Washington : электронный журнал. – URL: https://pjreddie.com/media/files/papers/YOLOv3.pdf
9. M00nL1ght A. Распознавание объектов в режиме реального времени на iOS с помощью YOLOv3 / A. M00nL1ght // Habr.com : электронный

журнал. – URL: https://habr.com/ru/post/460869/. – Дата публикации: 25.07.2019.
10. wadik69 B. OpenCV в Python. Часть 1 / B. wadik69 // Habr.com : электронный журнал. – URL: https://habr.com/ru/post/519454/. – Дата публикации: 17.09.2020.
11. Рыбалко С. Использование OpenCV и Face Recognition в системах распознавания лиц на одноплатных компьютерах типа Raspberry Pi / С. Рыбалко // Evergreens.com.ua : электронный журнал. – URL: https://evergreens.com.ua/ru/articles/open-cv-face-recognition.html. – Дата публикации: 25.01.2021.
12. Алгоритм Виолы Джонса и каскадный классификатор Хаара // Ichi.pro : электронный журнал. – URL: https://ichi.pro/ru/algoritm-violy-dzonsa-i-kaskadnyj-klassifikator-haara-261941383264216
13. Smola A. Introduction to Machine Learning / A. Smola, S.V.N. Vishwanathan // Yahoo! Labs : электронный журнал. – URL: https://alex.smola.org/drafts/thebook.pdf
14. Bradski G. Learning OpenCV / G. Bradski, A. Kaehler. – USA : O'Reilly Media, 2008. – 571 с.
15. Github.com, DeepFace: A Lightweight Face Recognition and Facial Attribute Analysis (Age, Gender, Emotion and Race) Library for Python : сайт. – URL: https://github.com/serengil/deepface (дата обращения: 11.03.2022)
16. Pypi.org, Deep Face Analysis Framework for Face Recognition and Demography : сайт. – URL: https://pypi.org/project/deepface/0.0.19/ (дата обращения: 11.03.2022)
17. McGuinness K. Generative models and adversarial training / K. McGuinness // Deep Learning for Computer Vision. – 2016. – № 4. – С. 1-20.
18. Kdnuggets.com, Exploring DeepFakes : сайт. – URL: https://www.kdnuggets.com/2018/03/exploring-deepfakes.html/2 (дата обращения: 11.03.2022)
19. Dalal N. Histograms of Oriented Gradients for Human Detection / N.Dalal, B.Triggs // IEEE CVPR - 2005. - № 1. – С. 886-893.
20. Object Detection Using Convolutional Neural Networks / R.L. Galvez, A.A. Bandala, E.P. Daious [и др.] // IEEE Reg. 10 Annu Int. Conf Proceedings/TENCON. – 2018. – окт.
21. Wang Q.J. LPP-HOG: A New Local Image Descriptor for Fast Human Detection / Q.J. Wang, R.B. Zhang // IEEE International Symposium on Knowledge Acquisition and Modeling Workshop. – 2008. – № 1. – С. 640-643.

22. Rich feature hierarchies for accurate object detection and semantic segmentation / R. Girshick, J. Donahue, T. Darrell, J. Malik // IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.. – 2014. – № 1. – С. 580–587.
23. Selective search for object recognition / J. R. R. Uijlings, K. E. A. Van De Sande, T. Gevers, A.W.M. Smeulders // Int. J. Comput. Vis.. – 2013. – Т. 2, № 104. – С. 154–171.
24. Parthasarathy D. A BRIEF HISTORY OF CNNS IN IMAGE SEGMENTATION: FROM R-CNN TO MASK R-CNN / D. Parthasarathy // Njkhanh.com : электронный журнал. – URL: https://njkhanh.com/a-brief-history-of-cnns-in-image-segmentation-from-r-cnn-to-mask-r-cnn-p5f393438. – Дата публикации: 27.04.2017.
25. Object Detection with Deep Learning: A Review / Z.Q. Zhao, P. Zheng, S.T. Xu, X. Wu // IEEE Trans. Neural Networks Learn. Syst.. – 2019. – Т. 11, № 30. – С. 3212–3232.
26. Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition / K. He, X. Zhang, S. Ren, J. Sun // ECCV. – 2014. – Т. 1, № 1. – С. 346–361.
27. Girshick R. Fast R-CNN / R. Girshick // Computer Vision and Pattern Recognition : электронный журнал. – URL: https://arxiv.org/abs/1504.08083. – Дата публикации: 13.04.2015.
28. Faster R-CNN: Towards RealTime Object Detection with Region Proposal Networks / S. Ren, K. He, R. Girshick, J. Sun // IEEE Trans. Pattern Anal. Mach. Intell.. – 2017. – Т. 39, № 6. – С. 1137–1149.
29. Nabati R. RRPN : RADAR REGION PROPOSAL NETWORK FOR OBJECT DETECTION IN AUTONOMOUS VEHICLES / R. Nabati, H. Qi // IEEE Int. Conf. Image Process.. – 2019. – № 1. – С. 3093–3097.
30. Tsang S. Reading: Light-Head R-CNN — In Defense of Two-Stage Object Detector (Object Detection) / S. Tsang // Medium.com : электронный журнал. – URL: https://sh-tsang.medium.com/reading-light-head-r-cnn-in-defense-of-two-stage-object-detector-object-detection-83929fe5947a. – Дата публикации: 05.10.2020.
31. Github.com, Convolution arithmetic : сайт. – URL: https://github.com/vdumoulin/conv_arithmetic (дата обращения: 11.03.2022)
32. De Palma R. YOLOv3 Architecture: Best Model in Object Detection / R. De Palma // BestInAU.com.au : электронный журнал. – URL: https://bestinau.com.au/yolov3-architecture-best-model-in-object-detection/
33. Vicinal risk minimization. In Advances in Neural Information Processing Systems / O. Chapelle, J. Weston, L. Bottou, V. Vapnik // MIT Press: Cambridge, MA, USA. – 2020. – № 1. – С. 416–422.

34. Object Detection in 20 Years: A Survey / Z. Zou, Z. Shi, Y. Guo, J. Ye // Computer Vision and Pattern Recognition : электронный журнал. – URL: https://arxiv.org/abs/1905.05055
35. Bag of Freebies for Training Object Detection Neural Networks / Z. Zhang, T. He, H. Zhang, J. Xie, M.Li // Computer Vision and Pattern Recognition : электронный журнал. – URL: https://arxiv.org/abs/1902.04103
36. boomer Д. Batch Normalization для ускорения обучения нейронных сетей / Д. boomer // Habr.com : электронный журнал. – URL: https://habr.com/ru/post/309302/. – Дата публикации: 06.09.2016.
37. Ning E. A Javascript library to run ONNX models in browsers and Node.js / E. Ning // W3.org : электронный журнал. – URL: https://www.w3.org/2020/Talks/mlws/en_onnxjs.pdf
38. TENSORFLOW.JS: MACHINE LEARNING FOR THE WEB AND BEYOND / D. Smilkov, N. Thorat, Y. Assogba, A. Yuan, N.Kreeger, P. Yu, K. Zhang, S. Cai, E. Nielsen // Computer Vision and Pattern Recognition : электронный журнал. – URL: https://arxiv.org/pdf/1901.05350v1.pdf
39. A novel YOLOv3-tiny network for unmanned airship obstacle detection / S. Ding, F. Long, H. Fan, L. Liu // IEEE 8th Data Driven Control Learn. Syst. Conf. DDCLS. – 2019. – № 1. – С. 277–281.
40. Roboflow.com : сайт. – URL: https://docs.roboflow.com (дата обращения: 15.04.2022)

Приложение А

```
import cv2
import argparse
import numpy as np

ap = argparse.ArgumentParser()
ap.add_argument('-i', '--image', required=True,
        help = 'path to input image')
ap.add_argument('-c', '--config', required=True,
        help = 'path to yolo config file')
ap.add_argument('-w', '--weights', required=True,
        help = 'path to yolo pre-trained weights')
ap.add_argument('-cl', '--classes', required=True,
        help = 'путь к текстовому файлу, содержащему имена классов')
args = ap.parse_args()
image = cv2.imread(args.image)

Width = image.shape[1]
Height = image.shape[0]
scale = 0.00392

classes = None
with open(args.classes, 'r') as f:
    classes = [line.strip() for line in f.readlines()]

COLORS = np.random.uniform(0, 255, size=(len(classes), 3))
net = cv2.dnn.readNet(args.weights, args.config)

blob = cv2.dnn.blobFromImage(image, scale, (416,416), (0,0,0), True, crop=False)

net.setInput(blob)
def get_output_layers(net):
    layer_names = net.getLayerNames()
    output_layers = [layer_names[i[0] - 1] for i in net.getUnconnectedOutLayers()]
    return output_layers

def draw_bounding_box(img, class_id, confidence, x, y, x_plus_w, y_plus_h):
    label = str(classes[class_id])
    color = COLORS[class_id]
    cv2.rectangle(img, (x,y), (x_plus_w,y_plus_h), color, 2)
    cv2.putText(img, label, (x-10,y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)
outs = net.forward(get_output_layers(net))

class_ids = []
confidences = []
boxes = []
conf_threshold = 0.5
nms_threshold = 0.4
for out in outs:
    for detection in out:
```

```
            scores = detection[5:]
            class_id = np.argmax(scores)
            confidence = scores[class_id]
            if confidence > 0.5:
                center_x = int(detection[0] * Width)
                center_y = int(detection[1] * Height)
                w = int(detection[2] * Width)
                h = int(detection[3] * Height)
                x = center_x - w / 2
                y = center_y - h / 2
                class_ids.append(class_id)
                confidences.append(float(confidence))
                boxes.append([x, y, w, h])
        indices = cv2.dnn.NMSBoxes(boxes, confidences, conf_threshold, nms_threshold)
        for i in indices:
            i = i[0]
            box = boxes[i]
            x = box[0]
            y = box[1]
            w = box[2]
            h = box[3]

            draw_bounding_box(image, class_ids[i], confidences[i], round(x), round(y),
round(x+w), round(y+h))
        cv2.imshow("object detection", image)
        cv2.waitKey()
        cv2.imwrite("object-detection.jpg", image)
        cv2.destroyAllWindows()
```

Приложение Б

```
import cv2
import argparse
import os
parser = argparse.ArgumentParser(description='OpenCV)
parser.add_argument('--src', action='store', default=0, nargs='?', help='Set video source; default is
usb webcam')
parser.add_argument('--w', action='store', default=320, nargs='?', help='Set video width')
parser.add_argument('--h', action='store', default=240, nargs='?', help='Set video height')
args = parser.parse_args()
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcas-
cade_frontalface_default.xml')
cap = cv2.VideoCapture(args.src)
while 1:
ret, img = cap.read()
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
faces = face_cascade.detectMultiScale(gray, 1.3, 5)
for (x,y,w,h) in faces:
cv2.circle(img, (x+w/2,y+h/2),(max(w,h)/2),(255,255,0),2)
roi_gray = gray[y:y+h, x:x+w]
roi_color = img[y:y+h, x:x+w]
cv2.imshow('opencv face detection', img)
if cv2.waitKey(10):
break
cap.release()
cv2.destroyAllWindows()
```

Приложение В

```
#clone YOLOv5 and
!git clone https://github.com/ultralytics/yolov5
%cd yolov5
%pip install -qr requirements.txt
%pip install -q roboflow

import torch
import os
from IPython.display import Image, clear_output

print(f"Setup complete. Using torch {torch.__version__}
({torch.cuda.get_device_properties(0).name if torch.cuda.is_available() else 'CPU'})")
from roboflow import Roboflow
rf = Roboflow(model_format="yolov5", notebook="ultralytics")
# set up environment
os.environ["DATASET_DIRECTORY"] = "/content/datasets"
!python train.py --img 416 --batch 16 --epochs 150 --data {dataset.location}/data.yaml --weights
yolov5s.pt –cache
%load_ext tensorboard
%tensorboard --logdir runs
!python detect.py --weights runs/train/exp/weights/best.pt --img 416 --conf 0.1 --source {da-
taset.location}/test/images
#display inference on ALL test images

import glob
from IPython.display import Image, display

for imageName in glob.glob('/content/yolov5/runs/detect/exp/*.jpg'): #assuming JPG
    display(Image(filename=imageName))
    print("\n")
#export your model's weights for future use
from google.colab import files
files.download('./runs/train/exp/weights/best.pt')

  <script type='text/javascript'>
      var Module = {};

      var has_simd;
      var has_threads;

      var wasmModuleLoaded = false;
      var wasmModuleLoadedCallbacks = [];

      Module.onRuntimeInitialized = function() {
         wasmModuleLoaded = true;
         for (var i = 0; i < wasmModuleLoadedCallbacks.length; i++) {
            wasmModuleLoadedCallbacks[i]();
         }
      }
```

```
wasmFeatureDetect.simd().then(simdSupported => {
   has_simd = simdSupported;

   wasmFeatureDetect.threads().then(threadsSupported => {
      has_threads = threadsSupported;

      if (has_simd & has_threads)
      {
         yolo_module_name = 'yolo';
         console.log('simd&threads enabled.');
      }
      else
      {
         yolo_module_name = 'ios/yolo-ios';
         console.log('cannot enable simd&threads.');
      }

      console.log('load ' + yolo_module_name);

      var yolowasm = yolo_module_name + '.wasm';
      var yolojs = yolo_module_name + '.js';

      fetch(yolowasm)
         .then(response => response.arrayBuffer())
         .then(buffer => {
            Module.wasmBinary = buffer;
            var script = document.createElement('script');
            script.src = yolojs;
            script.onload = function() {
               console.log('Emscripten boilerplate loaded.');
            }
            document.body.appendChild(script);
         });

   });
});


var dst = null;
var resultarray = null;
var resultbuffer = null;
window.addEventListener('DOMContentLoaded', function() {
   var isStreaming = false;
   video = document.getElementById('video');
   canvas = document.getElementById('canvas');
   ctx = canvas.getContext('2d');
   w = 640;
   h = 480;
   var constraints = { audio: false, video: { width: w, height: h } };
   navigator.mediaDevices.getUserMedia(constraints)
      .then(function(mediaStream) {
```

```
            var video = document.querySelector('video');
            video.srcObject = mediaStream;
            video.onloadedmetadata = function(e) {
               video.play();
            };
         })
         .catch(function(err) {
            console.log(err.message);
         });
      // Wait until the video stream canvas play
      video.addEventListener('canplay', function(e) {
         if (!isStreaming) {
            // videoWidth isn't always set correctly in all browsers
            if (video.videoWidth > 0) h = video.videoHeight / (video.videoWidth / w);
            canvas.setAttribute('width', w);
            canvas.setAttribute('height', h);
            isStreaming = true;
         }
      }, false);

      // Wait for the video to start to play
      video.addEventListener('play', function() {
         //Setup image memory
         var id = ctx.getImageData(0, 0, canvas.width, canvas.height);
         var d = id.data;

         if (wasmModuleLoaded) {
            mallocAndCallSFilter();
         } else {
            wasmModuleLoadedCallbacks.push(mallocAndCallSFilter);
         }

         function mallocAndCallSFilter() {
            dst = _malloc(d.length);

            // max 20 objects
            resultarray = new Float32Array(6 * 20);
            resultbuffer = _malloc(6 * 20 * Float32Array.BYTES_PER_ELEMENT);

            HEAPF32.set(resultarray, resultbuffer / Float32Array.BYTES_PER_ELEMENT);

            //console.log("What " + d.length);

            sFilter();
         }
      });

});

var class_names = [
   "background",
```

```
        "Наденьте маску", "Маска надета"
    ];

    var colors = [
        "rgb( 255, 69,   0)",
        "rgb( 32, 178, 170)"
    ];

    function ncnn_yolo() {
        var canvas = document.getElementById('canvas');
        var ctx = canvas.getContext('2d');

        var imageData = ctx.getImageData(0, 0, canvas.width, canvas.height);
        var data = imageData.data;
        ctx.font = '16px comics-sans'

        HEAPU8.set(data, dst);

        _yolo_ncnn(dst, canvas.width, canvas.height, resultbuffer);

        // resultarray
        var qaqarray = HEAPF32.subarray(resultbuffer / Float32Array.BYTES_PER_ELEMENT, resultbuffer / Float32Array.BYTES_PER_ELEMENT + 6 * 20);

        var i;
        var remind_ctx = '';
        for (i = 0; i < 20; i++) {
            var label = qaqarray[i * 6 + 0];
            var prob = qaqarray[i * 6 + 1];
            var bbox_x = qaqarray[i * 6 + 2];
            var bbox_y = qaqarray[i * 6 + 3];
            var bbox_w = qaqarray[i * 6 + 4];
            var bbox_h = qaqarray[i * 6 + 5];

            if (label == -233)
                continue;
            console.log('qaq ' + label + ' = ' + prob);

            //ctx.strokeStyle = colors[i % 19];
            if (label == 1) {
                ctx.strokeStyle = colors[0];
                ctx.fillStyle = colors[0];
            }
            else {
                ctx.strokeStyle = colors[1];
                ctx.fillStyle = colors[1];
            }

            ctx.strokeRect(bbox_x, bbox_y, bbox_w, bbox_h);
            ctx.lineWidth = 2;
```

```
        var text = class_names[label] + ": " + parseFloat(prob * 100).toFixed(2) + "%";

        ctx.textBaseline = 'top';
        var text_width = ctx.measureText(text).width;
        var text_height = parseInt(ctx.font, 10);

        var x = bbox_x;
        var y = bbox_y - text_height;
        if (y < 0)
            y = 0;
        if (x + text_width > canvas.width)
            x = canvas.width - text_width;

        //ctx.fillStyle = "rgb(255,255,255)";
        ctx.fillRect(x-1, y-2, text_width+4, text_height+2);
        ctx.fillStyle = "rgb(255,255,255)";
        ctx.fillText(text, x+1, y-2);
    }
    // console.log(remind_ctx);
    document.getElementById('remind').innerHTML = remind_ctx;
}

//Request Animation Frame function
var sFilter = function() {
    if (video.paused || video.ended) return;

    ctx.fillRect(0, 0, w, h);
    ctx.drawImage(video, 0, 0, w, h);

    ncnn_yolo();

    window.requestAnimationFrame(sFilter);
}
```

Приложение Г

```
var Module = {};

var has_simd;
var has_threads;

var wasmModuleLoaded = false;
var wasmModuleLoadedCallbacks = [];

Module.onRuntimeInitialized = function() {
  wasmModuleLoaded = true;
  for (var i = 0; i < wasmModuleLoadedCallbacks.length; i++) {
    wasmModuleLoadedCallbacks[i]();
  }
}

wasmFeatureDetect.simd().then(simdSupported => {
  has_simd = simdSupported;

  wasmFeatureDetect.threads().then(threadsSupported => {
    has_threads = threadsSupported;

    if (has_simd & has_threads)
    {
      yolo_module_name = 'yolo';
      console.log('simd&threads enabled.');
    }
    else
    {
      yolo_module_name = 'ios/yolo-ios';
      console.log('cannot enable simd&threads.');
    }

    console.log('load ' + yolo_module_name);

    var yolowasm = yolo_module_name + '.wasm';
    var yolojs = yolo_module_name + '.js';

    fetch(yolowasm)
      .then(response => response.arrayBuffer())
      .then(buffer => {
        Module.wasmBinary = buffer;
        var script = document.createElement('script');
        script.src = yolojs;
        script.onload = function() {
          console.log('Emscripten boilerplate loaded.');
        }
        document.body.appendChild(script);
      });

  });
});
```

```
var dst = null;
var resultarray = null;
var resultbuffer = null;
window.addEventListener('DOMContentLoaded', function() {
  var isStreaming = false;
  video = document.getElementById('video');
  canvas = document.getElementById('canvas');
  ctx = canvas.getContext('2d');
  w = 640;
  h = 480;
  var constraints = { audio: false, video: { width: w, height: h } };
  navigator.mediaDevices.getUserMedia(constraints)
    .then(function(mediaStream) {
      var video = document.querySelector('video');
      video.srcObject = mediaStream;
      video.onloadedmetadata = function(e) {
        video.play();
      };
    })
    .catch(function(err) {
      console.log(err.message);
    });
  // Wait until the video stream canvas play
  video.addEventListener('canplay', function(e) {
    if (!isStreaming) {
      // videoWidth isn't always set correctly in all browsers
      if (video.videoWidth > 0) h = video.videoHeight / (video.videoWidth / w);
      canvas.setAttribute('width', w);
      canvas.setAttribute('height', h);
      isStreaming = true;
    }
  }, false);

  // Wait for the video to start to play
  video.addEventListener('play', function() {
    //Setup image memory
    var id = ctx.getImageData(0, 0, canvas.width, canvas.height);
    var d = id.data;

    if (wasmModuleLoaded) {
      mallocAndCallSFilter();
    } else {
      wasmModuleLoadedCallbacks.push(mallocAndCallSFilter);
    }

    function mallocAndCallSFilter() {
      dst = _malloc(d.length);

      // max 20 objects
      resultarray = new Float32Array(6 * 20);
      resultbuffer = _malloc(6 * 20 * Float32Array.BYTES_PER_ELEMENT);
```

```
            HEAPF32.set(resultarray, resultbuffer / Float32Array.BYTES_PER_ELEMENT);

            //console.log("What " + d.length);

            sFilter();
          }
        });

    });

    var class_names = [
      "background",
      "Маска киңіз", "Маска киілген"
    ];

    var colors = [
      "rgb( 255, 69,   0)",
      "rgb( 32, 178, 170)"
    ];

    function ncnn_yolo() {
      var canvas = document.getElementById('canvas');
      var ctx = canvas.getContext('2d');

      var imageData = ctx.getImageData(0, 0, canvas.width, canvas.height);
      var data = imageData.data;
      ctx.font = '14px comics-sans'

      HEAPU8.set(data, dst);

      _yolo_ncnn(dst, canvas.width, canvas.height, resultbuffer);

      // resultarray
      var qaqarray = HEAPF32.subarray(resultbuffer / Float32Array.BYTES_PER_ELEMENT, resultbuffer
/ Float32Array.BYTES_PER_ELEMENT + 6 * 20);

      var i;
      var remind_ctx = '';
      for (i = 0; i < 20; i++) {
        var label = qaqarray[i * 6 + 0];
        var prob = qaqarray[i * 6 + 1];
        var bbox_x = qaqarray[i * 6 + 2];
        var bbox_y = qaqarray[i * 6 + 3];
        var bbox_w = qaqarray[i * 6 + 4];
        var bbox_h = qaqarray[i * 6 + 5];

        if (label == -233)
          continue;

        console.log('qaq ' + label + ' = ' + prob);

        //ctx.strokeStyle = colors[i % 19];
```

```
            if (label == 1) {
                ctx.strokeStyle = colors[0];
                ctx.fillStyle = colors[0];
            }
            else {
                ctx.strokeStyle = colors[1];
                ctx.fillStyle = colors[1];
            }

            ctx.strokeRect(bbox_x, bbox_y, bbox_w, bbox_h);
            ctx.lineWidth = 2;

            var text = class_names[label] + ": " + parseFloat(prob * 100).toFixed(2) + "%";

            ctx.textBaseline = 'top';
            var text_width = ctx.measureText(text).width;
            var text_height = parseInt(ctx.font, 10);

            var x = bbox_x;
            var y = bbox_y - text_height;
            if (y < 0)
                y = 0;
            if (x + text_width > canvas.width)
                x = canvas.width - text_width;

            //ctx.fillStyle = "rgb(255,255,255)";
            ctx.fillRect(x-1, y-2, text_width+4, text_height+2);
            ctx.fillStyle = "rgb(255,255,255)";
            ctx.fillText(text, x+1, y-2);
        }
        // console.log(remind_ctx);
        document.getElementById('remind').innerHTML = remind_ctx;
    }

    //Request Animation Frame function
    var sFilter = function() {
        if (video.paused || video.ended) return;

        ctx.fillRect(0, 0, w, h);
        ctx.drawImage(video, 0, 0, w, h);

        ncnn_yolo();

        window.requestAnimationFrame(sFilter);
    }

        // Tencent is pleased to support the open source community by making ncnn available.
//
// Copyright (C) 2020 THL A29 Limited, a Tencent company. All rights reserved.
//
// Licensed under the BSD 3-Clause License (the "License"); you may not use this file except
// in compliance with the License. You may obtain a copy of the License at
//
```

```
// https://opensource.org/licenses/BSD-3-Clause
//
// Unless required by applicable law or agreed to in writing, software distributed
// under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR
// CONDITIONS OF ANY KIND, either express or implied. See the License for the
// specific language governing permissions and limitations under the License.

#ifndef NCNN_COMMAND_H
#define NCNN_COMMAND_H

#include "platform.h"

#if NCNN_VULKAN

#include "mat.h"

#include <vulkan/vulkan.h>

namespace ncnn {

class Pipeline;
class VkCompute
{
public:
    VkCompute(const VulkanDevice* vkdev);
    virtual ~VkCompute();

public:
    void record_upload(const Mat& src, VkMat& dst, const Option& opt);

    void record_upload(const Mat& src, VkImageMat& dst, const Option& opt);

    void record_download(const VkMat& src, Mat& dst, const Option& opt);

    void record_download(const VkImageMat& src, Mat& dst, const Option& opt);

    void record_buffer_to_image(const VkMat& src, VkImageMat& dst, const Option& opt);

    void record_image_to_buffer(const VkImageMat& src, VkMat& dst, const Option& opt);

    void record_clone(const Mat& src, VkMat& dst, const Option& opt);

    void record_clone(const Mat& src, VkImageMat& dst, const Option& opt);

    void record_clone(const VkMat& src, Mat& dst, const Option& opt);

    void record_clone(const VkImageMat& src, Mat& dst, const Option& opt);

    void record_clone(const VkMat& src, VkMat& dst, const Option& opt);

    void record_clone(const VkImageMat& src, VkImageMat& dst, const Option& opt);

    void record_clone(const VkMat& src, VkImageMat& dst, const Option& opt);
```

```
    void record_clone(const VkImageMat& src, VkMat& dst, const Option& opt);

    void record_pipeline(const Pipeline* pipeline, const std::vector<VkMat>& bindings, const
std::vector<vk_constant_type>& constants, const VkMat& dispatcher);

    void record_pipeline(const Pipeline* pipeline, const std::vector<VkImageMat>& bindings, const
std::vector<vk_constant_type>& constants, const VkImageMat& dispatcher);

    void record_pipeline(const Pipeline* pipeline, const std::vector<VkMat>& buffer_bindings, const
std::vector<VkImageMat>& image_bindings, const std::vector<vk_constant_type>& constants, const
VkMat& dispatcher);
    void record_pipeline(const Pipeline* pipeline, const std::vector<VkMat>& buffer_bindings, const
std::vector<VkImageMat>& image_bindings, const std::vector<vk_constant_type>& constants, const
VkImageMat& dispatcher);
    void record_pipeline(const Pipeline* pipeline, const std::vector<VkMat>& buffer_bindings, const
std::vector<VkImageMat>& image_bindings, const std::vector<vk_constant_type>& constants, const
Mat& dispatcher);

#if NCNN_BENCHMARK
    void record_write_timestamp(uint32_t query);
#endif // NCNN_BENCHMARK

#if __ANDROID_API__ >= 26
    void record_import_android_hardware_buffer(const ImportAndroidHardwareBufferPipeline* pipeline,
const VkImageMat& src, const VkMat& dst);

    void record_import_android_hardware_buffer(const ImportAndroidHardwareBufferPipeline* pipeline,
const VkImageMat& src, const VkImageMat& dst);
#endif // __ANDROID_API__ >= 26

    int submit_and_wait();

    int reset();

#if NCNN_BENCHMARK
    int create_query_pool(uint32_t query_count);

    int get_query_pool_results(uint32_t first_query, uint32_t query_count, std::vector<uint64_t>&
results);
#endif // NCNN_BENCHMARK

protected:
    int init();
    int begin_command_buffer();
    int end_command_buffer();

protected:
    const VulkanDevice* vkdev;

    VkCommandPool compute_command_pool;

    VkCommandBuffer compute_command_buffer;
```

```
    VkFence compute_command_fence;

    std::vector<VkMat> upload_staging_buffers;
    std::vector<VkMat> download_post_buffers;
    std::vector<Mat> download_post_mats_fp16;
    std::vector<Mat> download_post_mats;

    std::vector<VkImageMemory*> image_blocks_to_destroy;

    // the good-old path for device without VK_KHR_push_descriptor
    std::vector<VkDescriptorPool> descriptor_pools;
    std::vector<VkDescriptorSet> descriptorsets;

    struct record
    {
        enum
        {
            TYPE_copy_buffer,
            TYPE_copy_image,
            TYPE_copy_buffer_to_image,
            TYPE_copy_image_to_buffer,
            TYPE_bind_pipeline,
            TYPE_bind_descriptorsets,
            TYPE_push_constants,
            TYPE_dispatch,
            TYPE_memory_barrers,
            TYPE_buffer_barrers,
            TYPE_image_barrers,

#if NCNN_BENCHMARK
            TYPE_write_timestamp,
#endif // NCNN_BENCHMARK

            TYPE_post_download,
            TYPE_post_cast_float16_to_float32,
        };

        int type;
        VkCommandBuffer command_buffer;

        union
        {
            struct
            {
                VkBuffer src;
                VkBuffer dst;
                uint32_t region_count;
                const VkBufferCopy* regions;
            } copy_buffer;
            struct
            {
                VkImage src;
```

```
    VkImageLayout src_layout;
    VkImage dst;
    VkImageLayout dst_layout;
    uint32_t region_count;
    const VkImageCopy* regions;
} copy_image;
struct
{
    VkBuffer src;
    VkImage dst;
    VkImageLayout layout;
    uint32_t region_count;
    const VkBufferImageCopy* regions;
} copy_buffer_to_image;
struct
{
    VkImage src;
    VkImageLayout layout;
    VkBuffer dst;
    uint32_t region_count;
    const VkBufferImageCopy* regions;
} copy_image_to_buffer;

struct
{
    VkPipelineBindPoint bind_point;
    VkPipeline pipeline;
} bind_pipeline;
struct
{
    VkPipelineBindPoint bind_point;
    VkPipelineLayout pipeline_layout;
    uint32_t descriptorset_count;
    uint32_t descriptorset_offset;
} bind_descriptorsets;
struct
{
    VkPipelineLayout pipeline_layout;
    VkShaderStageFlags stage_flags;
    uint32_t size;
    const void* values;
} push_constants;

struct
{
    uint32_t group_count_x;
    uint32_t group_count_y;
    uint32_t group_count_z;
} dispatch;

struct
{
    VkPipelineStageFlags src_stage;
```

```
            VkPipelineStageFlags dst_stage;
            uint32_t barrier_count;
            const VkMemoryBarrier* barriers;
        } memory_barrers;
        struct
        {
            VkPipelineStageFlags src_stage;
            VkPipelineStageFlags dst_stage;
            uint32_t barrier_count;
            const VkBufferMemoryBarrier* barriers;
        } buffer_barrers;
        struct
        {
            VkPipelineStageFlags src_stage;
            VkPipelineStageFlags dst_stage;
            uint32_t barrier_count;
            const VkImageMemoryBarrier* barriers;
        } image_barrers;

#if NCNN_BENCHMARK
        struct
        {
            uint32_t query;
        } write_timestamp;
#endif // NCNN_BENCHMARK

        struct
        {
            uint32_t download_post_buffer_mat_offset;
            uint32_t download_post_mat_fp16_offset;
        } post_download;
        struct
        {
            uint32_t download_post_mat_fp16_offset;
            uint32_t download_post_mat_offset;
        } post_cast_float16_to_float32;
    };
  };

  std::vector<record> delayed_records;

#if NCNN_BENCHMARK
  uint32_t query_count;
  VkQueryPool query_pool;
#endif // NCNN_BENCHMARK
};

class VkTransfer
{
public:
  VkTransfer(const VulkanDevice* vkdev);
  ~VkTransfer();
```

```
public:
    void record_upload(const Mat& src, VkMat& dst, const Option& opt, bool flatten = true);

    void record_upload(const Mat& src, VkImageMat& dst, const Option& opt);

    int submit_and_wait();

protected:
    int init();
    int begin_command_buffer();
    int end_command_buffer();

protected:
    const VulkanDevice* vkdev;

    VkCommandPool compute_command_pool;
    VkCommandPool transfer_command_pool;

    VkCommandBuffer upload_command_buffer;
    VkCommandBuffer compute_command_buffer;

    VkSemaphore upload_compute_semaphore;

    VkFence upload_command_fence;
    VkFence compute_command_fence;

    std::vector<VkMat> upload_staging_buffers;
};

} // namespace ncnn

#endif // NCNN_VULKAN

#endif // NCNN_COMMAND_H
```