|  |  |
| --- | --- |
| **СОГЛАСОВАНО**  Доцент кафедры  ИАНИ ННГУ, к.ф.-м.н.  \_\_\_\_\_\_\_\_\_\_\_\_\_ Д.А. Яшунин  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_2019 г. | **УТВЕРЖДАЮ**  Профессор кафедры  ИАНИ ННГУ, д.т.н.  \_\_\_\_\_\_\_\_\_\_\_\_\_ Н.В. Старостин  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_2019 г. |

**Пояснительная записка № 5**

**«Разработка архитектуры ПО «FaceRecognition»»**

**Этап 1. Исследование предметной области и разработка способа решения задачи**

**НИР «Разработка и реализация программного обеспечения   
для распознавания лиц на групповых фотографиях»**

**(Шифр ПО «FaceRecognition»)**

Ответственный исполнитель

\_\_\_\_\_\_\_\_\_\_\_\_ A. Филатова

\_\_\_\_\_\_\_\_\_\_\_\_ A. Пронина

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_2019 г.

**Н. Новгород 2019**

**Аннотация**

В пояснительной записке «Разработка архитектуры ПО «FaceRecognition»» в рамках этапа исследования предметной области и разработка способа решения задачи описан технологический стек: представлены Data Flow диаграмма системы, UML диаграммы описания работы системы, описаны технологии, которые будут использоваться для разработки.

Также представлены пояснительные записки по выбранным методам решения задачи, а также по выбранным фреймворкам.

**Оглавление**

[«Data Flow диаграмма Face Recognition системы» 4](#_Toc5052326)

[UML Диаграмма деятельности «Клиент-серверные взаимодействия внутри Face Recognition системы» 5](#_Toc5052327)

[UML Диаграмма деятельности «Face Recognition двигатель» 6](#_Toc5052328)

[Pipeline проекта 7](#_Toc5052329)

[Описание технологического стека 8](#_Toc5052330)

[Диаграмма технологического стека 8](#_Toc5052331)

[Пояснительная записка к фреймворку OpenVINO 9](#_Toc5052332)

[Пояснительная записка к фреймворку PyTorch 12](#_Toc5052333)

[Пояснительная записка к фреймворку SphereFace 17](#_Toc5052334)

[Пояснительная записка к сети Faster R-CNN 20](#_Toc5052335)

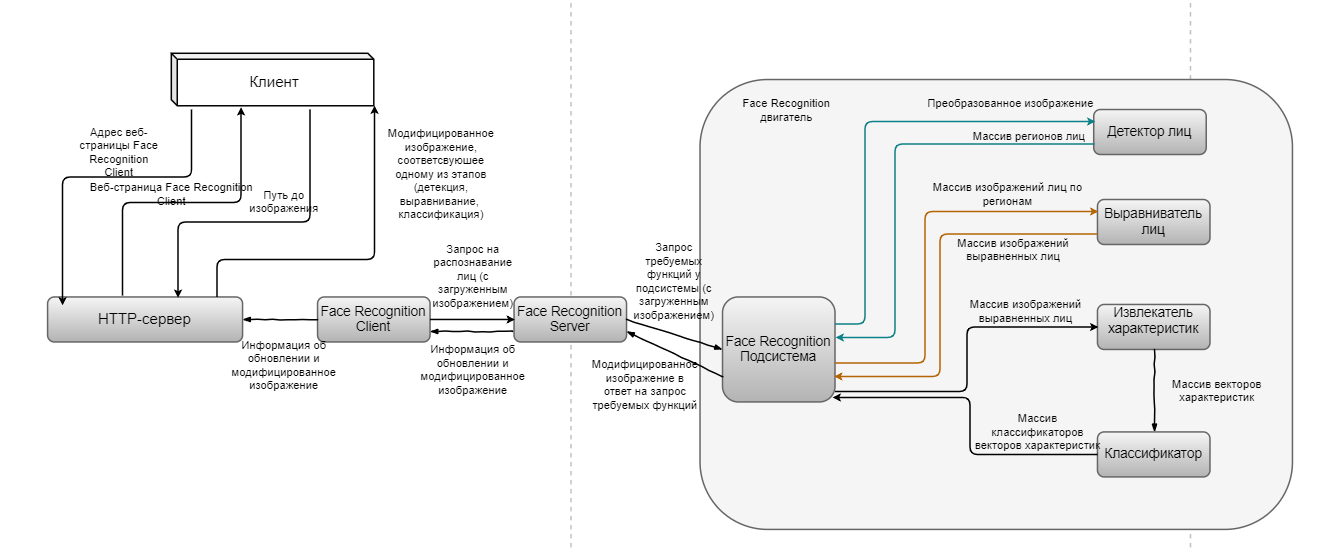
[Пояснительная записка к сети face-detection-adas-0001 из OpenVINO Toolkit для детекции лиц 23](#_Toc5052336)

[Пояснительная записка к фреймворку OpenCV для выравнивания лиц и нахождения ключевых точек 23](#_Toc5052337)

[Пояснительная записка сети facial-landmarks-35-adas-0001 для нахождения ключевых точек 24](#_Toc5052338)

[Список используемых источников 25](#_Toc5052340)

# «Data Flow диаграмма Face Recognition системы»



# UML Диаграмма деятельности «Клиент-серверные взаимодействия внутри Face Recognition системы»



# UML Диаграмма деятельности «Face Recognition двигатель»



# Pipeline проекта

Общий конвейер работы системы:

1. *Пользователь* заходит в веб-браузер и набирает доменное имя сайта с проектом распознавания лиц.
2. *HTTP сервер* отсылает требуемую HTML + JS страничку (*Face Recognition клиент*) обратно пользователю.
3. *Пользователь* загружает на страницу изображение и нажимает «Recognize Face» кнопку.
4. *Face Recognition клиент* отсылает запрос *Face Recognition серверу* на обработку загруженного пользователем изображения.
5. *Face Recognition сервер* передает управление *Face Recognition подсистеме*, делая запрос на распознавание.
6. *Face Recognition подсистема* производит предварительную обработку изображения и посылает результат *модулю Детектор лиц.*
7. *Детектор лиц* определяет ограничивающие прямоугольники лиц на изображении. Результат (массив регионов лиц) и управление возвращаются *Face Recognition подсистеме*.
8. *Face Recognition подсистема* обрамляет каждое задетектированное лицо на входящем исходном изображении, сохраняет полученное изображение и делает его “DT копию” для дальнейшей работы. Также *Face Recognition подсистема* для каждого из регионов лиц создает отдельное изображение, содержащее только соответствующий регион пикселей. Эти изображения в виде массива передаются *модулю Выравниватель лиц*.
9. *Выравниватель лиц* обрабатывает все регионы интереса, центрируя лица, если это требуется. Результат (массив изображений выровненных лиц) и управление возвращаются *Face Recognition подсистеме*.
10. *Face Recognition подсистема* собирает полученный массив изображений в одно изображение и сохраняет этот промежуточный результат. Также *Face Recognition подсистема* посылает полученный на предыдущем шаге массив *модулю Извлекатель характеристик.*
11. *Извлекатель характеристик* обрабатывает все регионы интереса, детектируя ключевые точки на лицах и сохраняя последние как характеристические вектора. Характеристические вектора и управление возвращается *Face Recognition подсистеме*.
12. *Face Recognition подсистема* отправляет полученные характеристические вектора *модуль Классификатор*. *Классификатор* сравнивает каждый характеристический вектор с подобными векторами в предопределенном отклассифицированном наборе данных. Находятся ближайшие по расстоянию вектора из набора к входящим характеристическим векторам и возвращаются их идентификаторы. Для каждого ограничивающего прямоугольника в сохраненном изображении добавляется идентификатор лица (Имя человека или Неизвестно). Результат (массив идентификаторов векторов) и управление возвращается *Face Recognition подсистеме.*
13. *Face Recognition подсистема* на основе результата работы классификатора добавляет на “DT копию” изображения надпись – Имя человека либо Неизвестно и сохраняет полученное изображение. После этого создается ассоциативный контейнер, куда для каждого этапа сохраняется полученные промежуточные изображения. *Face Recognition подсистема* уведомляет *Face Recognition сервер* о готовности результатов запроса и передает ему заполненный ассоциативный контейнер.
14. *Face Recognition сервер* передает ассоциативный контейнер с изображениями *Face Recognition клиенту*.
15. *Face Recognition клиент* отрисовывает изображения с названиями этапов пользователю.

# Описание технологического стека

Cтек технологий разбит на подкатегории по цели использования:

*Frontend*: HTML+Js ? **|** Python ? **|** HTML5 ? **|** Ruby-on-Rails  
*Backend*: HTTP server on Python   
 TCP Face Recognition server on C++.  
*Face Recognition Subsystem:*

* Для задачи распознавания лиц без тренировки сети для классификации:
  + Детекция лиц: OpenVINO Deep Learning Framework + Faster R-CNN / face-detection-adas-0001.
  + Выравнивание лиц: OpenCV (является часть дистрибьюции OpenVINO).
  + Извлечение характеристик: OpenVINO Deep Learning Framework + SphereFace (может быть загружена из Caffe).
  + Классификатор: написанный вручную классификатор векторов характеристик.
* Для задачи распознавания лиц с тренировкой сети для классификации:
  + Для детекции лиц, выравнивания лиц, извлечения характеристик *стек технологий остается неизменным.*
  + Классификатор: натренированная своя классификационная сеть с помощью PyTorch Deep Learning Framework.

# Диаграмма технологического стека



# Пояснительная записка к фреймворку OpenVINO

Intel® Distribution of **OpenVINO ™** toolkit [1] помогает быстро развертывать решения для эмуляции человеческого зрения. Основанный на сверточных сетях (CNN), данный инструмент позволяет запускать алгоритмы компьютерного зрения (CV) на аппаратном обеспечении Intel®, с максимальной производительностью. OpenVINO включает в себя Intel® Deep Learning Deployment Toolkit (Intel® DLDT).

**Преимущества и недостатки**

**+** Позволяет запускать натренированные сверточные нейронные сети эффективно прямо на целевом устройстве.

**+** Ускоряет конечную разработку благодаря простоте эксплуатации библиотеки компьютерного зрения и предварительно оптимизированным ядрам.

**+** Поддерживает гетерогенное выполнение на процессорах Intel®, интегрированных графических процессорах Intel®, Intel® Movidius ™ Neural Compute Stick (NCS), Intel® Neural Compute Stick 2 и Intel® Vision Accelerator Design с Intel® Movidius ™ VPU.

**+** Включает оптимизированные вызовы для стандартов компьютерного зрения, включая OpenCV \*, OpenCL ™ и OpenVX.

**+** Продукт сам по себе не тяжелый и имеет практически нулевое количество зависимостей.

**+** Производительность продукта при вычислении сетей на платформах Intel в разы выше по сравнению с популярными фреймворками.

**+** Значительно ниже требования по используемой памяти, что актуально для ряда приложений.

**+** Поддерживает запуск сетей популярных фреймворков на всех платформах Intel и под управлением различных операционных систем, например, полноценный запуск сетей на платформах Интел под управлением Windows OS. Запуск под управлением Windows – отдельная история. Не все фреймворки поддерживают такой запуск «из коробки», к примеру, запуск caffe не очень прост. TensorFlow поставляет бинарники для Windows, но, если нужно внести модификации и перестроить – может быть проблема. В то же самое время, заметно, что запуск на Windows часто востребован.

**+** Поставляются натренированные оптимизированные модели.

**+** Поставляется сразу несколько натренированных моделей для решения одной задачи (например, детектирования лиц). Некоторые модели могут работать значительно быстрее, за счёт незначительной потери в качестве. В зависимости от требований приложения можно выбрать то или другое.  
Также модели могут отличаться по условиям постановки задачи: к примеру, угол съемки человека может влиять на качество детектирования. OpenVINO предоставляет 2 модели для разных случаев.  
Следует отметить что для моделей существует файл с описанием, там можно посмотреть цифры производительности, аккуратность модели и примеры изображений, которые ожидаются на вход, т.е. описание сценария.

**-** Отсутствие тренировки сетей.

**-** Натренированные модели решают узкую задачу.

**-** Интеграция с Web.

**Установка**

Следующие компоненты устанавливаются по умолчанию:

| **Компонент** | **Описание** |
| --- | --- |
| [Model Optimizer](https://software.intel.com/en-us/articles/OpenVINO-ModelOptimizer) | Этот инструмент импортирует, конвертирует и оптимизирует модели, которые были обучены на популярных платформах, в формат, используемый инструментами Intel, особенно Inference Engine.  Примечание. Популярные платформы включают Caffe \*, TensorFlow \*, MXNet \* и ONNX \*. |
| [Inference Engine](https://software.intel.com/en-us/articles/OpenVINO-InferEngine) | Это двигатель, который запускает модель глубокого обучения. Он включает в себя набор библиотек для простой интеграции в ваши приложения. |
| Drivers and runtimes for OpenCL™ version 2.1 | Включает OpenCL на GPU / CPU для Intel® процессоров. |
| Intel® Media SDK | Предоставляет доступ к аппаратно ускоренным видеокодекам и обработке кадров. |
| [OpenCV\*](https://docs.opencv.org/) | Community версия OpenCV \*, скомпилированная для оборудования Intel®. Включает библиотеки PVL для компьютерного зрения. |
| OpenVX\* | Intel реализация OpenVX \* оптимизированная для работы на оборудовании Intel® (CPU, GPU, IPU). |
| Pre-trained models | Набор предварительно обученных моделей Intel для учебных и демонстрационных целей или для разработки ПО для глубокого обучения. |
| Sample Applications | Набор простых консольных приложений, демонстрирующих, как использовать Inference Engine в ваших приложениях. Для получения дополнительной информации о построении и запуске примеров, обратитесь к [Руководство к разработке на Inference Engine](https://software.intel.com/en-us/articles/OpenVINO-InferEngine#Using_Inference_Engine_Samples). |

Установка OpenVINO на Linux и Windows – [2] и [3] соответственно.

**Inference Engine**

**Рабочий процесс развертывания**

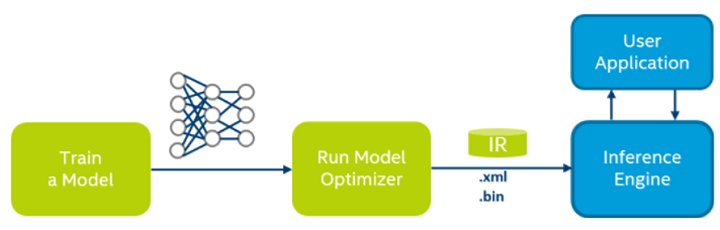
Процесс развертывания Inference Engine предполагает, что использование Model Optimizer, чтобы преобразовать обученную модель в промежуточное представление (IR - Intermediate Representation). Схема ниже иллюстрирует типичный рабочий процесс для развертывания натренированной модели глубокого обучения.

Рис.1 Развертывание натренированной модели глубокого обучения

Краткое изложение шагов по оптимизации и развертыванию обученной модели:

* Настройте Model Optimizer для вашей платформы.
* Преобразуйте обученную модель для получения оптимизированного промежуточного представления (IR) модели на основе топологии обученной сети, значений весов и смещений.
* Протестируйте модель в формате промежуточного представления используя Inference Engine в целевой среде с помощью приложения проверки или приложений-примеров.
* Интегрируйте Inference Engine в свое приложение, чтобы развернуть модель в целевой среде.

**Вводная в Inference Engine**

Inference Engine – это C++ библиотека с набором C++ классов чтобы запусить сети на входных данных (изображениях) и получить результат. C++ библиотека предоставляет некий API чтобы считывать промежуточное представление (IR), устанавливать входящие и выходящие форматы и запускать модели на устройствах.

Inference Engine использует архитектуру плагинов. Плагин Inference Engine - это программный компонент, который содержит полную реализацию для инференса на определенном аппаратном устройстве Intel®: например, CPU, GPU, VPU, FPGA. Каждый плагин реализует унифицированный API и предоставляет дополнительные API-интерфейсы, специализированные к конкретному аппаратному устройству.

**Общий рабочий процесс c Inference Engine**

1. *Прочитайте IR* - Используя класс InferenceEngine::CNNNetReader, прочтите файл IR в класс CNNNetwork. Этот класс представляет сеть в памяти хоста.
2. *Подготовьте формат входов и выходов*. После загрузки сети укажите точность и схему входа и выхода сети. Для этих спецификаций используйте CNNNetwork::getInputInfo() и CNNNetwork::getOutputInfo().
3. *Выберите плагин* - выберите плагин для загрузки вашей сети. Создайте плагин с помощью вспомогательного класса загрузки InferenceEngine::PluginDispatcher. Передайте по-девайсные конфигурации загрузки, специфичные для данного устройства, и зарегистрируйте расширения для данного устройства.
4. *Постройте и загрузите* - используйте класс-оболочку над интерфейсом плагина InferenceEngine::InferencePlugin для вызова LoadNetwork(), чтобы скомпилировать и загрузить сеть на устройстве. Передайте по-целевую конфигурацию загрузки для данной операции построения и загрузки.
5. *Задать входные данные* – После загрузки сети у вас появляется объект класса ExecutableNetwork. Используйте этот объект для создания InferRequest, в котором вы указываете входные буферы для в и вывода. Укажите выделенную устройству память и скопируйте ее непосредственно в память устройства или попросите устройство использовать память приложения для сохранения копии.
6. *Выполнить* - Теперь, когда память ввода и вывода определена, выберите режим выполнения:

* Синхронно - метод Infer(). Блокирует до завершения инференса.
* Асинхронно - метод StartAsync(). Cтатус проверяется с помощью метода wait() (0 timeout), ждите или укажите обратный вызов для завершения.

1. *Забрать результат* - после завершения инференса прочитайте выходную память или память, предоставленную ранее. Сделайте это с помощью InferRequest GetBlob API.

Интеграция IE API в приложение – [4].

# Пояснительная записка к фреймворку PyTorch

PyTorch [5] - основанная на Torch библиотека машинного обучения для Python. Она похожа на NumPy, но с мощной поддержкой графического процессора. Библиотека была разработана AI Research Group из Facebook в 2016 году.

**Преимущества и недостатки PyTorch:**

(+) Процесс моделирования прост и прозрачен благодаря архитектурному стилю фреймворка;

(+) Режим определения по умолчанию больше похож на традиционное программирование, и вы можете использовать общие средства отладки, такие как отладчик pdb, ipdb или PyCharm;

(+) Имеет декларативный параллелизм данных;

(+) Легко писать свои собственные типы слоев и работать на GPU;

(+) Имеет много предварительно обученных моделей и модульных частей, которые легко объединяются;

(-) Не хватает модельной порции;

(-) Отсутствуют интерфейсы для мониторинга и визуализации, такие как Tensorboard, хотя вы можете подключиться к Tensorboard извне.

**Основы PyTorch**

Инструкции по установкe PyTorch – [6, 7].

Так как в дальнейшей работе мы будем использовать PyTorch для тренировки сетки, то детальнее разберем как обучать и тренировать сетки на PyTorch.

Весь процесс делится на следующие этапы:

1. Загрузка данных

2. Определение CNN

3. Тренировка модели

4. Оценка производительности обученной модели на наборе данных

**Пример 1**

В качестве примера рассмотрим тренировку сетки модели ResNet 50, сетка включена в подпакет PyTorch уже обученных моделей. С полной реализацией можно ознакомиться в [9], далее будет рассмотрена лишь некоторые моменты связанные с тренировки сети.

Организация своего учебного набора данных:PyTorch ожидает, что данные будут организованы по папкам, для каждого класса определена своя папка. Есть несколько вариантов - часто предполагается, что есть «верхние» папки – для обучения, тестирования и валидации, а внутри них расположены «нижние» папки – папки, соответствующие классам. В примере используется другой способ разбить набор данных на обучающие и тестовые наборы - на лету, т.е. часть данных случайным образом определяется как тренировочные, а другие – как проверочные (описание метода – [8]).

Загрузка предварительно обученной модели ResNet 50 осуществляется следующим образом:

device = torch.device("cuda" if torch.cuda.is\_available()   
 else "cpu")  
model = models.resnet50(pretrained=True)  
print(model)

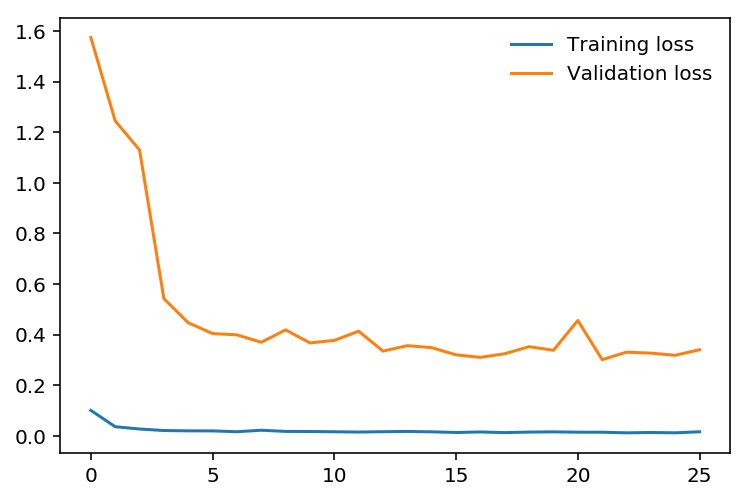
Далее «замораживаются» предварительно обученные слои, чтобы не пропустить их обратно в обучение. Затем переопределяется последний полностью связанный слой, который будет тренироваться с изображениями. Также создается критерий (функция потерь) и подбирается оптимизатор (в данном случае Адама) и скорость обучения.

for param in model.parameters():  
 param.requires\_grad = False  
   
model.fc = nn.Sequential(nn.Linear(2048, 512),  
 nn.ReLU(),  
 nn.Dropout(0.2),  
 nn.Linear(512, 10),  
 nn.LogSoftmax(dim=1))  
criterion = nn.NLLLoss()  
optimizer = optim.Adam(model.fc.parameters(), lr=0.003)  
model.to(device)

В этом примере есть только одна эпоха, но в большинстве случаев понадобится больше. Базовый процесс довольно интуитивно понятен из кода: загружаются пакеты изображений и выполняется цикл прямой связи. Затем рассчитывается функция потерь и используется оптимизатор для применения градиентного спуска при обратном распространении – в PyTorch это делается просто. Большая часть приведенного ниже кода имеет дело с отображением потерь и вычислением точности каждых 10 пакетов, поэтому происходит обновление во время обучения. Во время проверки необходимо установить модель в режим eval (), а затем вернуть к train ().

epochs = 1  
steps = 0  
running\_loss = 0  
print\_every = 10  
train\_losses, test\_losses = [], []

for epoch in range(epochs):  
 for inputs, labels in trainloader:  
 steps += 1  
 inputs, labels = inputs.to(device), labels.to(device)  
 optimizer.zero\_grad()  
 logps = model.forward(inputs)  
 loss = criterion(logps, labels)  
 loss.backward()  
 optimizer.step()  
 running\_loss += loss.item()  
   
 if steps % print\_every == 0:  
 test\_loss = 0  
 accuracy = 0  
 model.eval()  
 with torch.no\_grad():  
 for inputs, labels in testloader:  
 inputs, labels = inputs.to(device),  
 labels.to(device)  
 logps = model.forward(inputs)  
 batch\_loss = criterion(logps, labels)  
 test\_loss += batch\_loss.item()  
   
 ps = torch.exp(logps)  
 top\_p, top\_class = ps.topk(1, dim=1)  
 equals =   
 top\_class == labels.view(\*top\_class.shape)  
 accuracy +=  
 torch.mean(equals.type(torch.FloatTensor)).item()  
 train\_losses.append(running\_loss/len(trainloader))  
 test\_losses.append(test\_loss/len(testloader))   
 print(f"Epoch {epoch+1}/{epochs}.. "  
 f"Train loss: {running\_loss/print\_every:.3f}.. "  
 f"Test loss: {test\_loss/len(testloader):.3f}.. "  
 f"Test accuracy: {accuracy/len(testloader):.3f}")  
 running\_loss = 0  
 model.train()  
torch.save(model, 'aerialmodel.pth')



После этого модель будет сохранена для последующих прогнозов.

Есть еще одна вещь, которую можно сделать - это отразить потери обучения и проверки:

plt.plot(train\_losses, label='Training loss')  
plt.plot(test\_losses, label='Validation loss')  
plt.legend(frameon=False)  
plt.show()

Рис.2 Графики функций потерь обучения и проверки для примера 1

**Пример 2**

В данном примере тренировки сетки подробно рассмотрены процессы создания, обучения и оценки сети с помощью PyTorch. С полной реализацией можно ознакомиться в [10], далее будет рассмотрена лишь некоторые моменты связанные с тренировки сети.

Пример инициализации CNN:

class Net(nn.Module):  
 def \_\_init\_\_(self):  
 super(Net, self).\_\_init\_\_()  
 # convolutional layer  
 self.conv1 = nn.Conv2d(3, 16, 5)  
 # max pooling layer  
 self.pool = nn.MaxPool2d(2, 2)  
 self.conv2 = nn.Conv2d(16, 32, 5)  
 self.dropout = nn.Dropout(0.2)  
 self.fc1 = nn.Linear(32\*53\*53, 256)  
 self.fc2 = nn.Linear(256, 84)  
 self.fc3 = nn.Linear(84, 2)  
 self.softmax = nn.LogSoftmax(dim=1)  
   
 def forward(self, x):  
 # add sequence of convolutional and max pooling layers  
 x = self.pool(F.relu(self.conv1(x)))  
 x = self.pool(F.relu(self.conv2(x)))  
 x = self.dropout(x)  
 x = x.view(-1, 32 \* 53 \* 53)  
 x = F.relu(self.fc1(x))  
 x = self.dropout(F.relu(self.fc2(x)))  
 x = self.softmax(self.fc3(x))  
 return x

# create a complete CNN  
model = Net()

Также нужно инициализировать функцию потерь и оптимизатор. Функция потерь поможет рассчитать потери, сравнив прогноз и исходную метку. Оптимизатор минимизирует потери, обновляя параметры модели после каждой эпохи. Они могут быть инициализированы:

# Loss function  
criterion = torch.nn.CrossEntropyLoss()

# Optimizer  
optimizer = torch.optim.SGD(model.parameters(), lr = 0.003, momentum= 0.9)

Для обучения необходимо выполнить следующие шаги:

1. Очистить градиенты всех оптимизированных переменных:

Могут быть градиенты от предыдущих пакетов (batch), поэтому необходимо очищать градиент после каждой эпохи

1. Прямой проход:

Этот шаг вычисляет прогнозируемые результаты путем передачи входных данных в сверточную модель нейронной сети.

1. Рассчитать потери:

Когда модель тренируется, функция потерь вычисляет потери после каждой эпохи, а затем она используется оптимизатором.

1. Обратный проход:

Этот шаг вычисляет градиент потерь относительно параметров модели

1. Оптимизация

Выполняется один шаг оптимизации / обновление параметров для модели.

1. Обновление средней потери обучения

Ниже приведен код для обучения модели (для одной эпохи):

# number of epochs to train the model

n\_epochs = 5

valid\_loss\_min = np.Inf # track change in validation loss

for epoch in range(1, n\_epochs+1):

# keep track of training and validation loss

train\_loss = 0.0

valid\_loss = 0.0

# train the model #

model.train()

for data, target in train\_loader:

# move tensors to GPU if CUDA is available

if train\_on\_gpu:

data, target = data.cuda(), target.cuda()

# clear the gradients of all optimized variables

optimizer.zero\_grad()

# forward pass: compute predicted outputs by passing inputs to the model

output = model(data)

# calculate the batch loss

loss = criterion(output, target)

# backward pass: compute gradient of the loss with respect to model parameters

loss.backward()

# perform a single optimization step (parameter update)

optimizer.step()

# update training loss

train\_loss += loss.item()\*data.size(0)

# Validating the model

model.eval()

for data, target in valid\_loader:

if train\_on\_gpu:

data, target = data.cuda(), target.cuda()

# forward pass: compute predicted outputs by passing inputs to the model

output = model(data)

# calculate the batch loss

loss = criterion(output, target)

# update average validation loss

valid\_loss += loss.item()\*data.size(0)

# calculate average losses

train\_loss = train\_loss/len(train\_loader.dataset)

valid\_loss = valid\_loss/len(valid\_loader.dataset)

# print training/validation statistics

print('Epoch: {} \tTraining Loss: {:.6f} \tValidation Loss: {:.6f}'.format(

epoch, train\_loss, valid\_loss))

# save model if validation loss has decreased

if valid\_loss <= valid\_loss\_min:

print('Validation loss decreased ({:.6f} --> {:.6f}). Saving model ...'.format(

valid\_loss\_min,

valid\_loss))

torch.save(model.state\_dict(), 'model\_cifar.pt')

valid\_loss\_min = valid\_loss

Чтобы оценить модель, она должна быть изменена с model.train () на model.eval ():

model.eval()  
# iterate over test data  
len(test\_loader)  
for data, target in test\_loader:  
 # move tensors to GPU if CUDA is available  
 if train\_on\_gpu:  
 data, target = data.cuda(), target.cuda()  
 # forward pass  
 output = model(data)  
 # calculate the batch loss  
 loss = criterion(output, target)  
 # update test loss   
 test\_loss += loss.item()\*data.size(0)  
 # convert output probabilities to predicted class  
 \_, pred = torch.max(output, 1)   
 # compare predictions to true label  
 correct\_tensor = pred.eq(target.data.view\_as(pred))  
 correct = np.squeeze(correct\_tensor.numpy()) if not train\_on\_gpu else np.squeeze(correct\_tensor.cpu().numpy())  
 # calculate test accuracy for each object class  
 for i in range(batch\_size):   
 label = target.data[i]  
 class\_correct[label] += correct[i].item()  
 class\_total[label] += 1

# average test loss  
test\_loss = test\_loss/len(test\_loader.dataset)  
print('Test Loss: {:.6f}\n'.format(test\_loss))

for i in range(2):  
 if class\_total[i] > 0:  
 print('Test Accuracy of %5s: %2d%% (%2d/%2d)' % (  
 classes[i], 100 \* class\_correct[i] / class\_total[i],  
 np.sum(class\_correct[i]), np.sum(class\_total[i])))  
 else:  
 print('Test Accuracy of %5s: N/A (no training examples)' % (classes[i]))

print('\nTest Accuracy (Overall): %2d%% (%2d/%2d)' % (  
 100. \* np.sum(class\_correct) / np.sum(class\_total),  
 np.sum(class\_correct), np.sum(class\_total)))

# Пояснительная записка к фреймворку SphereFace

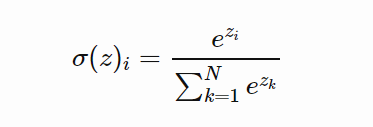
SphereFace [11, 12] – фреймворк для глубокого распознавания лиц. В общем случае пайплайн содержит три этапа - обнаружение, выравнивание и распознавание лиц. Для нашей дальнейшей работы нас будет интересовать только этап распознавания лиц.

Данный метод основан на сопоставлении вектора в пространстве гиперсферы объекту (в нашем случае – изображению лица).

Этот метод предлагает функцию A-Softmax (angular softmax), которая позволяет сверточным нейронным сетям исследовать угловые отличительные особенности. Геометрически функция A-Softmax может рассматриваться как наложение дискриминационных ограничений на гиперсферное многообразие, которое по сути совпадает с тем, что изображения также лежат на многообразии.

**Математическое обоснование функции потерь**

**Применение функции SoftMax в задаче классификации**

Функция SoftMax, функция мягкого максимума, часто используется в нейронных сетях в качестве функции активации при решении задачи классификации. SoftMax задается следующей формулой:

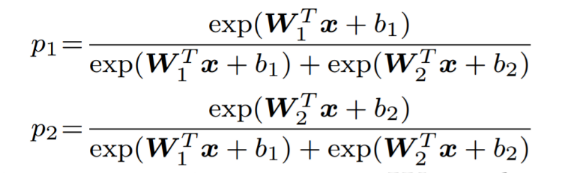
где zi – значение на выходе из i-го нейрона до активации, а N – общее количество нейронов в слое.

Эта функция очень часто используется для задач классификации.

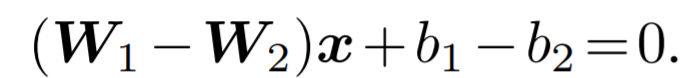
Рассмотрим задачу классификации, в которой объект принадлежит только одному классу. Значения на выходе из слоя могут трактоваться как вероятность принадлежности объекта заданному классу. Это означает, что каждое значение должно быть в диапазоне от 0 до 1, и сумма всех значений должна равняться единице. Также функция должна быть дифференцируема, чтобы можно было применять градиентные методы обучения нейронной сети.

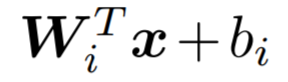
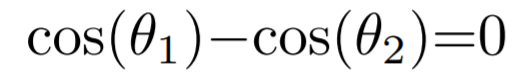
Достоинство функции SoftMax в том, что она использует экспоненту. За счет этого решается проблема с отрицательными выходными значениями, т.к. экспонента всегда положительна. Кроме того, экспонента значительно увеличивает большие значения. Если на выходе из нейрона 1, то экспонента равна 2.7, если 2 – экспонента равна 4.7, а если на выходе 4, то экспонента получается 54.6. Именно поэтому функции SoftMax удобно применять для задач классификации, т.к. она позволяет трактовать выходные значения нейронов как вероятность принадлежности данному классу, а также обеспечивает, чтобы только одно выходное значение было близко к единице за счет применения экспоненты.

**Пересмотр функции SoftMax loss**

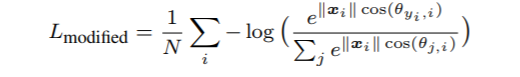
В случае двух классов апостериорные вероятности, полученные при Softmax loss равны:

где х - вектор изученных признаков, Wi и bi - веса и смещения последнего полностью связного слоя, соответствующего классу i, соответственно.

Предсказанная метка будет присвоена первому классу, если p1> p2, и второму классу, если p1<p2. Тогда функция, отвечающая за разделение между двумя классами будет иметь вид: 

можно представить в виде  , где θi - это угол между Wi и х. Нормализуем веса и занулим смещения слоев, тогда границей решения становится (угловая биссектриса вектора W1 иW2)

Хотя приведенный выше анализ построен на основе двоичной классификации, обобщить анализ для случая нескольких классов тривиально.

Функция Softmax loss может быть записана как:

где θj, i (0≤θj, i ≤π) - угол между вектором Wj и xi.

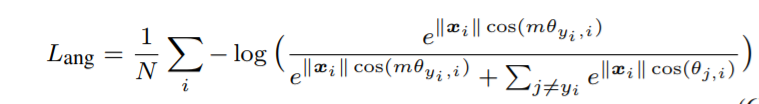
**Введение углового запаса в SoftMac loss**

Из предыдущего анализа потерь Softmax известно, что границы принятия решений могут сильно повлиять на распределение признаков, поэтому основная идея состоит в том, чтобы манипулировать границами принятия решений, чтобы получить угловой запас.

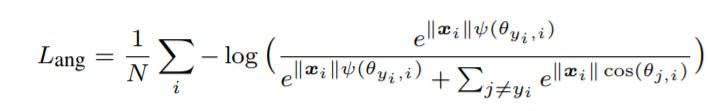
Пусть x – вектор признаков из первого класса, тогда чтобы правильно классифицировать х необходимо, чтобы .

Если вместо этого потребовать , то это сделает решение более строгим. Границей решения будет являться .

Таким образом измененная функция Softmax loss будет иметь следующий вид:



, где (θyi, i) должен находиться в диапазоне [0,π/m].

Чтобы избавиться от этого ограничения и сделать его оптимизируемым в CNN, расширяется диапазон определения cos (θyi, i), обобщив его на монотонно убывающую функцию угла ψ (θyi, i), которая должна быть равна cos (θyi , i) в [0,π/m]. Итоговая функция A-Softmax loss имеет вид:

, где  , 

Следовательно, A-Softmax Loss - это угловое расстояние между категориями выборки, что делает функцию принятия решений более строгой и более различимой. С увеличением m угловое расстояние также увеличивается.

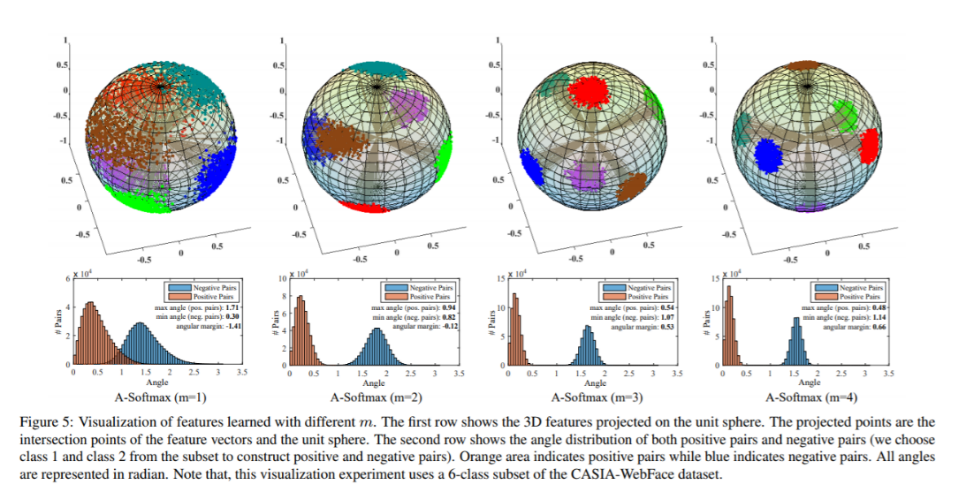
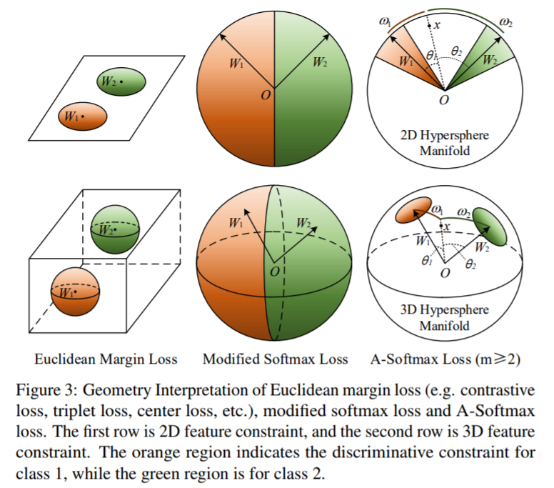
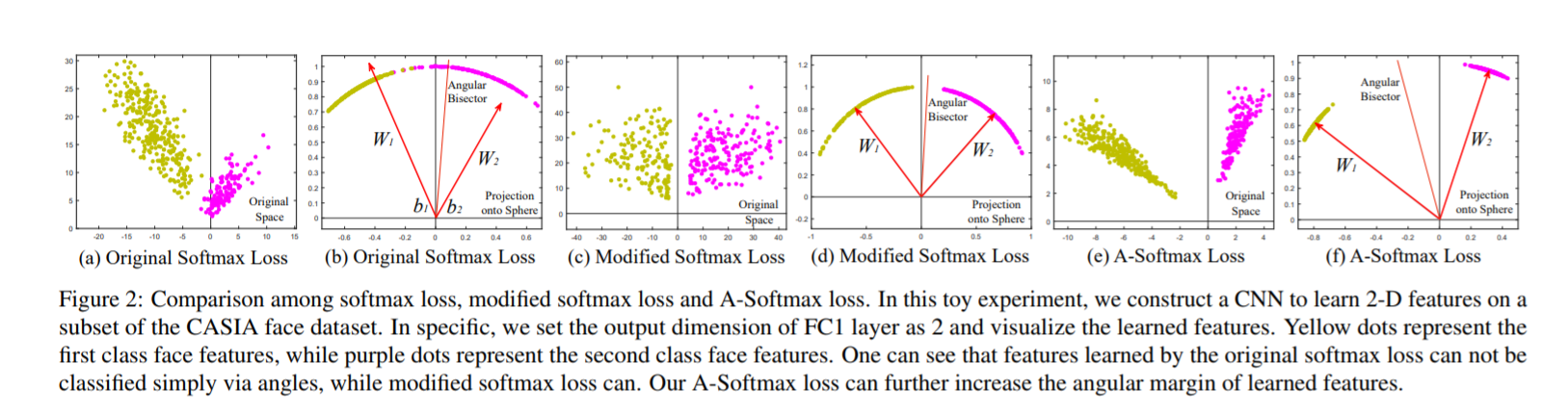
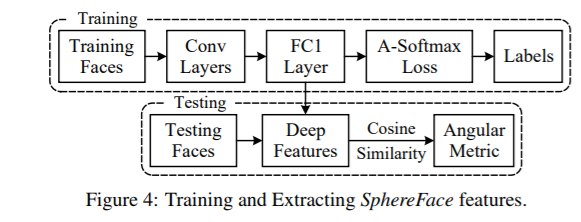
Контролируемые функцией A-Softmax, CNN изучают черты лица с геометрически интерпретируемым угловым запасом. Поскольку потери ASoftmax требуют Wi = 1, bi = 0, классификация зависит только от углов между выборкой x и Wi. Так что х можно отнести к тождеству с наименьшим углом. Параметр m добавляется с целью изучения углового поля между различными идентификаторами.

Рис.3 Графическая интерпретация функций потерь

**Обучение и тренировка**

Caffe используется для реализации функции потерь A-Softmax и CNN. Общая структура для обучения и извлечения функций SphereFace показана на рисунке:

Глубокие векторы признаков извлекаются из вывода слоя FC1. Для всех экспериментов окончательное представление тестируемого лица получается путем объединения ее исходных векторов признаков и горизонтальных отраженных элементов. Оценка (метрика) вычисляется по косинусному расстоянию двух признаков. Классификатор ближайшего соседа и пороговое значение используются для идентификации и проверки лица, соответственно.

# Пояснительная записка к сети Faster R-CNN

**Архитектура сети Faster R-CNN**

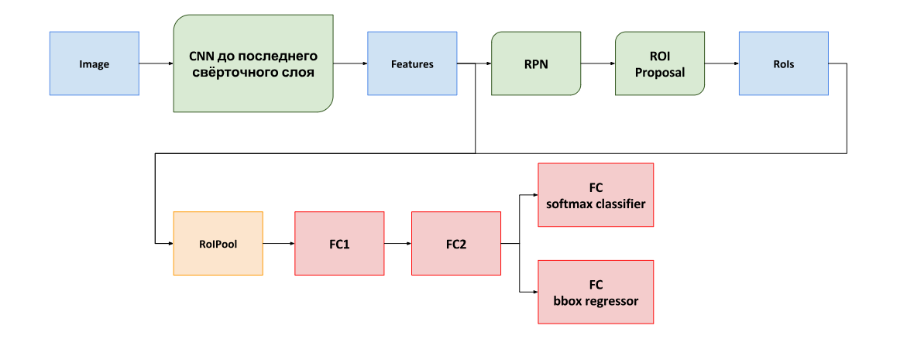
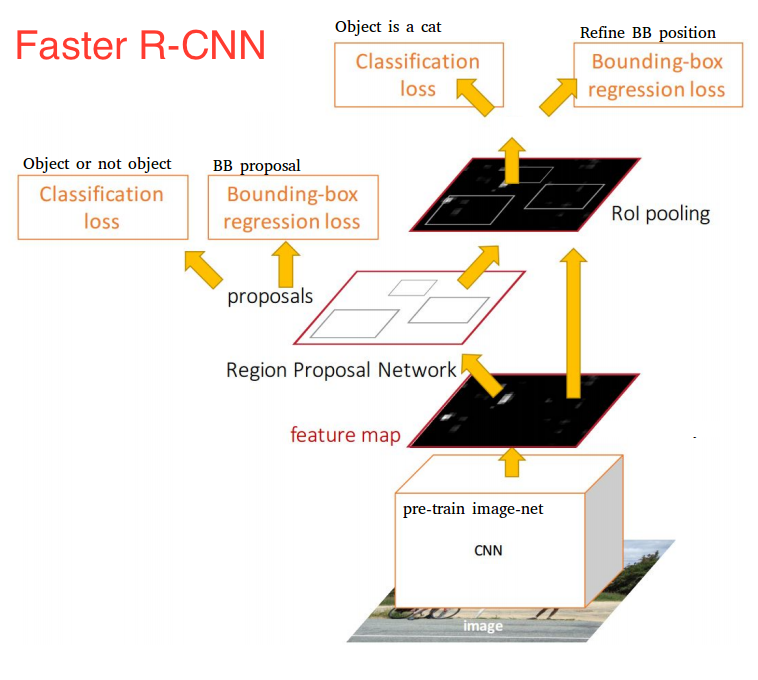
Сеть Faster R-CNN [13] состоит из двух модулей. Первый модуль представляет собой глубокую полностью сверточную сеть, которая предлагает регионы (замена процедуры генерации претендентов selective search в других моделях R-CNN, так как это занимало достаточно много времени), а второй модуль представляет собой Fast R-CNN детектор, который использует предложенные регионы. Вся система представляет собой единую унифицированную сеть для обнаружения объектов.

Рис.5 Архитектура сети Faster R-CNN

В качестве свёрточной сети авторы предлагают использовать либо VGG16 (Simonyan and Zisserman model), либо ZF (Zeiler and Fergus model). В дальнейшем при рассмотрении для определенности будем рассматривать случай VGG16 (для ZF ситуация будет аналогична). Для изображения размера W1×H1 на выходе последнего свёрточного слоя (conv5\_3) сеть VGG16 выдает карту особенностей с пространственными размерами W1/16×H1/16, вектор особенностей для каждой точки будет размерности 512. При этом в вектор особенностей в точке (xf,yf) вносят вклад точки изображения лежащие внутри квадрата с центром в (16xf,16yf) и размера 196×196.

Для каждой точки карты особенностей (xf,yf) будем проверять k претендетов разных размеров на изображении в регионах с центром в (16xf,16yf). В статье авторы предлагают рассматривать 9 претендентов, варьируя три масштаба и три отношения сторон (1:1, 1:2, 2:1). Для этой задачи используется Region Proposal Network (RPN). Модуль RPN сообщает модулю Fast R-CNN, где искать.

В рамках RPN по извлечённым CNN признакам скользят «мини-нейросетью» с небольшим (3х3) окном. Полученные с её помощью значения передаются в два параллельных полносвязанных слоя: box-regression layer (reg) и box-classification layer (cls).

Карта особенностей, полученная от свёрточной сети, подается на свёрточный слой с ядром размера 3×3. А выход этого свёрточного слоя параллельно подается на два свёрточных слоя с ядром размера 1×1. Первый слой rpn\_cls\_score выдает k пар - вероятности наличия или отсутствия объекта в соответствующем региона (фактически мы имеем обычный полносвязный классификатор, который применяется для каждого вектора особенностей сформированного слоем rpn\_conv

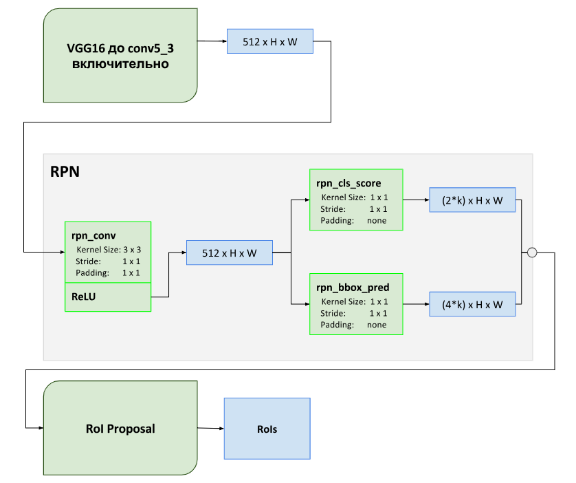
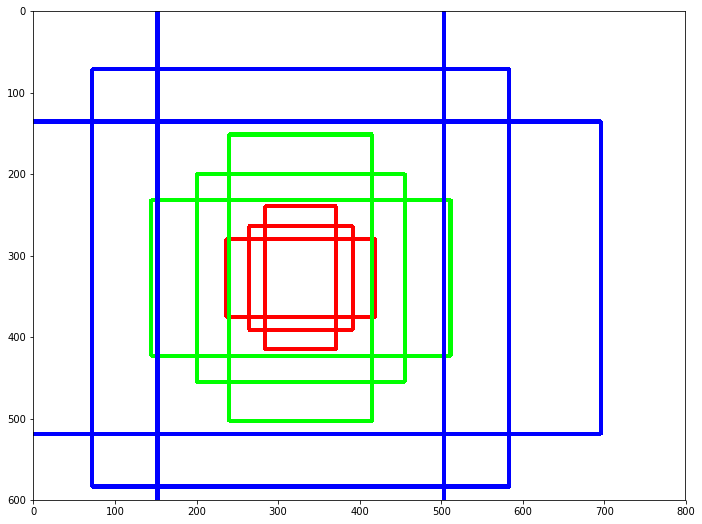
Слой rpn\_bbox\_pred выдает k четверок - поправки для координат центра и размеров соответствующего региона претендента (получается полносвязный регрессор, который опять применяется к векторам особенностей). Данный генератор претендентов, в силу применяемой схемы, инвариантен к сдвигам объектов на изображении. Авторы называют четвёрку - две координаты центра, масштаб и отношение сторон - анкер (anchor). Анкер полностью определяет регион на изображении. Количество всевозможных анкеров для изображения размеров W1×H1 будет равно 9⋅⌊W1/ 16⌋⋅⌊H1 / 16⌋.

Рис.5 Анкеры RPN

Рис.5 Структура сети Faster R-CNN

Вкратце, RPN ранжирует области регионов (анкеры) и предлагает те, которые наиболее вероятно содержат объекты. Архитектура выглядит следующим образом. В стандартной конфигурации Faster R-CNN в позиции изображения имеется 9 анкеров. На следующем графике (рис. 4) показано 9 анкеров в положении (320, 320) изображения с размером (600, 800). Три цвета представляют три шкалы или размеры: 128x128, 256x256, 512x512. Каждые три анкера одного цвета имеют соотношения высоты по ширине 1: 1, 1: 2 и 2: 1 соответственно.

Процессы тренировки сети Faster-RCNN, а также тренировки RPN описаны более подробно в пояснительной записке «ПЗ детекция с использованием OpenVINO + Faster R-CNN» [15]

**Демонстрация примера обнаружения предметов на фотографии с помощью Faster R-CNN и OpenVINO Toolkit**

В этом разделе показано, как запустить демонстрационное приложение Object Detection [15], которое делает вывод, используя сети обнаружения объектов, такие как Faster R-CNN на процессорах Intel® и Intel® HD Graphics.

После запуска демонстрационное приложение считывает параметры командной строки и загружает сеть и изображение в плагин Inference Engine. Когда вывод сделан, приложение создает выходное изображение и выводит данные в стандартный поток вывода.

Загрузка и преобразование Caffe\* Model

VGG16-Faster-RCNN является общедоступной CNN, которую можно легко взять на GitHub:

1. Скачать test.prototxt из <https://raw.githubusercontent.com/rbgirshick/py-faster-rcnn/master/models/pascal_voc/VGG16/faster_rcnn_end2end/test.prototxt>
2. Скачать предварительно обученные модели сиз <https://dl.dropboxusercontent.com/s/o6ii098bu51d139/faster_rcnn_models.tgz?dl=0>
3. Распакавать архив. Необходим файл VGG16\_faster\_rcnn\_final.caffemodel.

Для правильного преобразования исходной модели необходимо запустить Model Optimizer. Можно использовать следующую команду для преобразования исходной модели:

python3 ${MO\_ROOT\_PATH}/mo\_caffe.py

--input\_model <path\_to\_model>/VGG16\_faster\_rcnn\_final.caffemodel

--input\_proto <path\_to\_model>/deploy.prototxt

**Запуск**

Запуск приложения с параметром -h выдает следующее сообщение об использовании:

./object\_detection\_demo -h

InferenceEngine:

API version ............ <version>

Build .................. <number>

object\_detection\_demo [OPTION]

Options:

-h Print a usage message.

-i "<path>" Required. Path to an .bmp image.

-m "<path>" Required. Path to an .xml file with a trained model.

-l "<absolute\_path>" Required for MKLDNN (CPU)-targeted custom layers. Absolute path to a shared library with the kernels impl.

-c "<absolute\_path>" Required for clDNN (GPU)-targeted custom kernels. Absolute path to the xml file with the kernels desc.

-pp "<path>" Path to a plugin folder.

-d "<device>" Specify the target device to infer on; CPU, GPU, FPGA or MYRIAD is acceptable. The demo will look for a suitable plugin for a specified device.

-pc Enables per-layer performance report

-ni "<integer>" Number of iterations (default 1)

-bbox\_name "<string>" The name of output box prediction layer (default: bbox\_pred)

-proposal\_name "<string>" The name of output proposal layer (default: proposal)

-prob\_name "<string>" The name of output probability layer (default: cls\_prob)

-p\_msg Enables messages from a plugin

Запуск приложения с пустым списком опций выдает приведенное выше сообщение об использовании и сообщение об ошибке. Используйте следующую команду, чтобы сделать вывод о процессорах Intel® для образа, использующего обученную сеть Faster R-CNN:

$ ./object\_detection\_demo -i <path\_to\_image>/inputImage.bmp -m <path\_to\_model>/faster-rcnn.xml -d CPU

**Выходные данные**

Приложение выводит изображение с именем out\_0.bmp с обнаруженными объектами, заключенными в прямоугольники. Он выводит список классов обнаруженных объектов вместе с соответствующими значениями достоверности и координатами прямоугольников в стандартный поток вывода.

# Пояснительная записка к сети face-detection-adas-0001 из OpenVINO Toolkit для детекции лиц

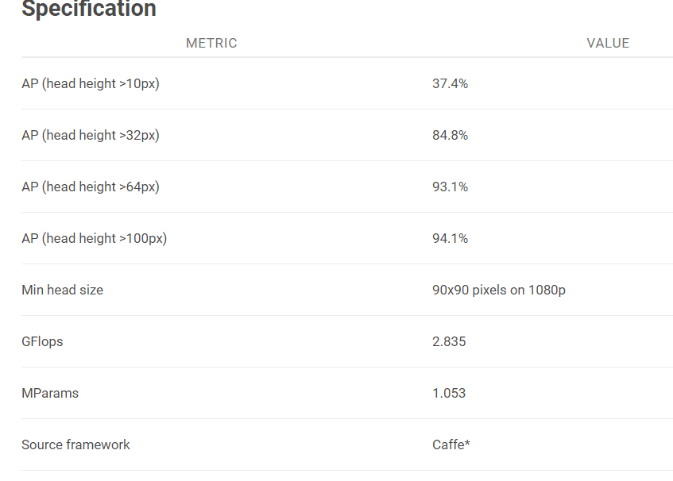
Сеть face-detection-adas-0001 [16,17] имеет магистраль MobileNet по умолчанию, которая включает в себя глубинные свертки, чтобы уменьшить объем вычислений для сверточного блока 3x3.

Рис.7 Детекция с помощью face-detection-adas-0001

Средняя точность (Average Precision, AP) определяется как область под кривой точности. Номера указаны в подмножестве проверки WiderFace.

Входные данные: name: "input", shape: [1x3x384x672] – входное изображение в формате [BxCxHxW], где: B - размер batch, C - количество каналов, H - высота изображения, W - ширина изображения. Ожидаемый порядок цвета - BGR.

Выходные данные имеют следующий вид: [1, 1, N, 7], где N - количество обнаруженных ограничивающих прямоугольников. Для каждого обнаружения описание имеет формат: [image\_id, label, conf, x\_min, y\_min, x\_max, y\_max], где image\_id - идентификатор изображения в пакете; label - прогнозируемый идентификатор класса; conf - уверенность в предсказанном классе; (x\_min, y\_min) - координаты верхнего левого угла ограничительной рамки; (x\_max, y\_max) - координаты нижнего правого угла ограничительной рамки.

# Пояснительная записка к фреймворку OpenCV для выравнивания лиц и нахождения ключевых точек

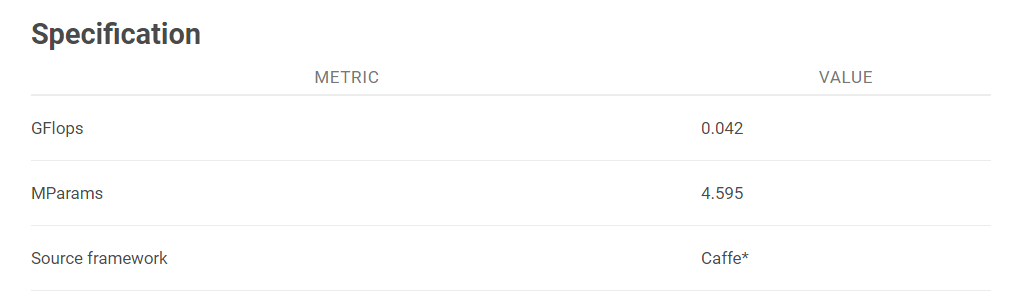
OpenCV [18] является самой мощной библиотекой компьютерного зрения.

Алгоритм работы:

1. Нахождение ключевых точек на лице.
2. Поворот изображения, чтобы центральные точки глаз находились на одной горизонтали.
3. Аффиные преобразования фотографии, что бы ключевые точки лица наиболее были наиболее близки к образцу.

# Пояснительная записка к сети facial-landmarks-35-adas-0001 для нахождения ключевых точек

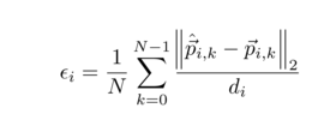
facial-landmarks-35-adas-0001 [20] – это сверточная нейронная сеть с пользовательской архитектурой для оценки 35 лицевых ориентиров.



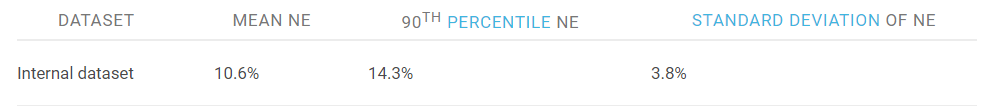
# 

Рис.8 Выделение ключевых точек с помощью facial-landmarks-35-adas-0001

Набор данных проверки - случайное подмножество из 1000 выборок большого внутреннего набора данных, содержащего изображения 300 человек с различными выражениями лица.

Результаты валидации - качество прогнозирования позиций ориентиров оценивается с помощью нормированной ошибки (NE). Ошибка для i-го образца имеет вид:

где N - число ориентиров, p-hat и p - соответственно векторы предсказания и наземной истинности k-го ориентира i-й выборки, а di - межглазное расстояние для i-й выборки.



Входные данные - формат [BxCxHxW], где: B - размер batch, C - количество каналов, H - высота изображения, W - ширина изображения

с данными имени и формы [1x3x60x60].

BLOB-объект состоит из изображения BGR размером 60x60 пикселей с вычитанным средним значением (120, 110, 104) и масштабным коэффициентом 0,0039.

Выходные данные - сеть выводит BLOB-объект в форме: [1, 70], содержащий вектор-строку из 70 значений с плавающей запятой для нормированных координат 35 ориентиров в форме (x0, y0, x1, y1, ..., x34, y34). Имя выходного слоя в формате Inference Engine: align\_fc3;

Имя выходного слоя в формате Caffe: align\_fc

# Список используемых источников

1. https://software.intel.com/en-us/openvino-toolkit

2. https://software.intel.com/en-us/articles/OpenVINO-Install-Linux

3. https://software.intel.com/en-us/articles/OpenVINO-Install-Windows

4. https://software.intel.com/en-us/articles/OpenVINO-InferEngine#integrate-with-api

5. https://pytorch.org/

6.https://github.com/deepmipt/dlschl/wiki/%D0%98%D0%BD%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%86%D0%B8%D1%8F-%D0%BF%D0%BE-%D1%83%D1%81%D1%82%D0%B0%D0%BD%D0%BE%D0%B2%D0%BA%D0%B5-PyTorch

7. https://pytorch.org/get-started/locally/#anaconda-1%23anaconda-1

8.https://medium.com/@contactsunny/how-to-split-your-dataset-to-train-and-test-datasets-using-scikit-learn-e7cf6eb5e0d

9. - https://github.com/cfotache/pytorch\_imageclassifier

10. https://github.com/jayrodge/Binary-Image-Classifier-PyTorch

11. SphereFace: Deep Hypersphere Embedding for Face Recognition - Weiyang Liu, Yandong Wen, Zhiding Yu, Ming Li, Bhiksha Raj, Le Song

12. https://github.com/wy1iu/sphereface

13.Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun

14. ПЗ детекция с использованием OpenVINO + Faster R-CNN, https://github.com/daryafret/faceRecognition/tree/dev/documents/%D0%9F%D0%97

15.https://docs.openvinotoolkit.org/2018\_R5/\_samples\_object\_detection\_demo\_README.html

16.https://docs.openvinotoolkit.org/2018\_R5/\_docs\_Transportation\_object\_detection\_face\_pruned\_mobilenet\_reduced\_ssd\_shared\_weights\_caffe\_desc\_face\_detection\_adas\_0001.html

17. https://github.com/opencv/open\_model\_zoo/tree/master/intel\_models/face-detection-adas-0001

18. https://opencv.org/

19.https://docs.openvinotoolkit.org/2018\_R5/\_docs\_Transportation\_object\_attributes\_facial\_landmarks\_custom\_35\_facial\_landmarks\_caffe\_desc\_facial\_landmarks\_35\_adas\_0001.html

20. https://github.com/opencv/open\_model\_zoo/tree/2018/intel\_models/facial-landmarks-35-adas-0001