ОЛИМПИАДА ШКОЛЬНИКОВ “ШАГ В БУДУЩЕЕ” ПО ПРОФИЛЮ “ИНЖЕНЕРНОЕ ДЕЛО”

№ 44854

Секция: Информационная безопасность и цифровая криминалистика

Использование технологий компьютерного зрения для повышения узнаваемости фотороботов

Автор: Чекуева Алима Ахмедовна
ЧУ СОШ Олимп-плюс, 10-Б

Москва-2024

АННОТАЦИЯ

В ходе работы изучались возможные решения проблемы низкой узнаваемости фотороботов с помощью компьютерного зрения. Целью являлось найти механизм, исправляющий недостатки используемого в настоящее время программного обеспечения.

Из двух возможных вариантов был реализован один - с использованием технологии стилизации изображений. Подбирались разные образцы стиля, но лучший результат показала модель лица, имеющая усредненные признаки обоих полов. Входные изображения были смещены влево относительно фона, переформатированы на размер 128x128 пикселей. Количество итераций - 300. Программа написана на python с фреймворком PyTorch, использовалась готовая модель сверточной нейросети.

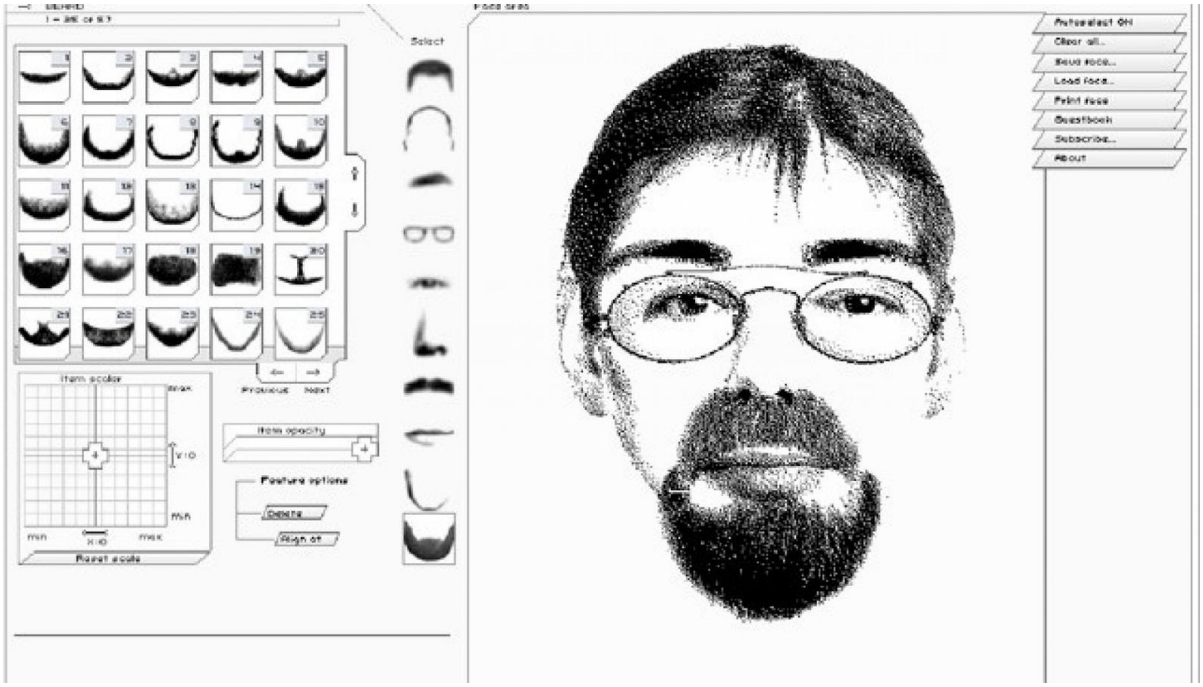
В ходе работы были проведены два опроса. Один - для оценки проблемы, другой - для оценки результатов. В первом предлагалось оценить 6 фотороботов 1-5/5. Во втором имитировать опознание по портрету. Из 4 “подозреваемых” выбирали самого похожего.

В итоге, решение удовлетворило часть поставленных целей. Результаты положительные, удалось повысить узнаваемость фотороботов, добиться частично целого, но хорошо прорисованного изображения. В будущем возможно продолжение работы в виде написания программы с использованием глубокого обучения.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Самый популярный метод в криминалистике - это, пожалуй, опознание по фотороботу. Художник, быстро рисующий портрет преступника по тревожным описаниям свидетеля - незаменимый элемент любого фильма-детектива. Хотя сам процесс сейчас выглядит по другому, фотороботы до сих пор остаются актуально применяемым способом опознания преступника.



(0.1) -пример ПО "фоторобот"

Фоторобот - портрет, составленный из отдельных фрагментов черт лица человека. В России сейчас такие используют в основном в маленьких городах, поселках и регионах. Для составления таких портретов используют специальное программное обеспечение, предоставляющее каталог черт лица человека, из которых механическим подбором составляется фоторобот. Примером такого программного обеспечения является иностранный комплекс "Faces", включающий в себя 2800 фрагментов на выбор.

Несмотря на, казалось бы, обширный набор плюсов составления фоторобота используя подобное программное обеспечение, их узнаваемость выходит достаточно низкой. Это показал социальный опрос, проведенный в рамках данной научной работы. Соответственно, требуется дополнение к применяемой технике, которое бы повысило результаты.

Итак, почему было решено использовать компьютерное зрение? Начнем с самого определения. Под это понятие входят технологии и методики,

распознающие, классифицирующие и в общем проводящие какую-либо работу с изображениями, используя машинное обучение, в том числе нейросети. Технологии компьютерного зрения позволяют проводить быструю, качественную работу над изображением, при этом давая уникальную возможность обучаться на прошлом опыте. Нынешнее ПО не имеет возможности обеспечивать эти функции. Такое решение также уникально: на настоящий момент подобное в данной сфере не используется.
Почему так важно решить проблему низкой узнаваемости? Уже упоминалось, что фоторобот используют маленькие федеральные территории. Причиной тому является отсутствие стабильной системы видеонаблюдения. Соответственно, там, где, скажем, Москва имеет возможность просмотреть по базе данных видеозаписей камер наблюдения, малые города полностью опираются на корректность портретов преступников. Тогда повысив их точность, удалось бы достичь понижения криминального риска в подобных субъектах.
Целью следующей работы являлось исследование возможностей повышения узнаваемости фотороботов с помощью технологий компьютерного зрения.

1. АНАЛИЗ

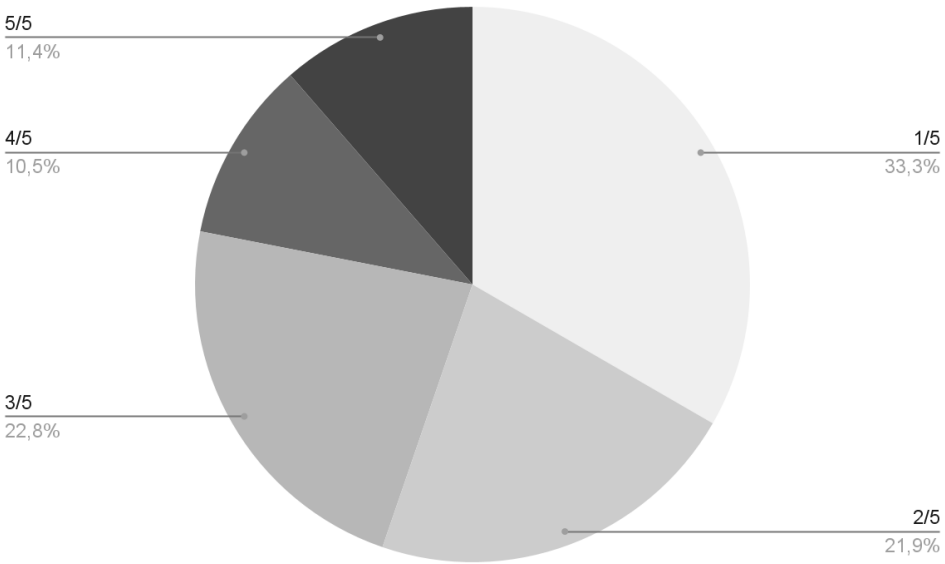
1.1 Оценка ситуации

В качестве оценки ситуации был собран некоторый список фактов о настоящем положении в данной сфере:

1. Основной метод составления фотороботов - программное обеспечение;
2. Целевой потребитель - федеральные территории России с малым бюджетом;
3. На настоящий момент составление фоторобота происходит полностью механически;
4. Для составления фоторобота используются показания жертвы/свидетеля;
5. Примерное количество фрагментов в каталоге комплекса: около 2000-3000.

1.2 Выявление проблемы, социальный опрос

Чтобы оценить корректность фоторобота, был составлен социальный опрос. Участником предлагалось 6 сравнений фотороботов разных типов с лицами преступников. Необходимо было оценить их схожесть 1-5/5 (от одного до пяти из пяти). Изображения были получены из интернета. Результаты на диаграмме:



(1.1)

Как можно увидеть, всего лишь 11.4% оценили схожесть в 5/5. При этом, большинство оценили ее в самый низкий показатель, 1/5. Подобные результаты являются крайне неудовлетворительными. Значительный процент низкой узнаваемости означает, что увидев

преступника в обыденной ситуации, человек будет скорее склоняться к мнению что он не является разыскиваемым.

1.3 Причины проблемы и возможные решения

Итак, почему же результаты опроса оказались такими низкими? Чтобы понять причину, был проведен обширное исследование литературы по теме. Нашлось две главные проблемы нынешнего решения.

Первая из них заключается в том, что каждое лицо абсолютно уникально. Все разнообразие изогнутости носов, форм губ, густоты бровей и величины ушей никак не уместить в набор из 2000 фрагментов. Это, пожалуй, главный минус фоторобота. Слишком малое количество фрагментов ограничивает в возможных комбинациях лиц. При этом, увеличить их количество тоже не получится: сотрудники будут элементарно путаться среди обширного выбора. Время составления фоторобота значительно увеличиться.

Вторая причина - человеческий фактор свидетеля. Он играет не менее важную роль, ведь описывать внешность преступника приходится людям, прошедшим травмирующие события. Людям, на которых было совершенно нападение, попытка грабежа или похищения. Оказалось, что такие события напрямую влияют на человеческую возможность распознавать лица. Она ухудшается и искажается, заставляя свидетеля путаться в показаниях. От такой погрешности трудно избавиться при составлении фоторобота. А с нынешней технологией - невозможно. Хорошо было бы вычислять среднюю “ошибку” искажения лица в памяти человека, чтобы учитывать ее впредь.

В итоге было составлено 4 основных критерия, которым должно удовлетворять решение. Они будут на протяжении всей последующей работы являться целевыми. Критерии:

1. Будущее решение должно иметь возможность предоставлять целое, реалистично прорисованное изображение человека. Такое требование является первым и самым очевидным. Для успешного опознания качество картинки должно быть максимальным. При этом, предпочтение будет отдаваться тому результату, который будет лучше выполнять второе условие. Как выяснилось, наш мозг лучше распознает обрезанную, но четкую картинку, чем наоборот. Настоящее решение

удовлетворяет этому критерию только частично. Скомпонованные фрагменты лиц часто создают прерывистое изображение. Несмотря на реалистичный стиль, нынешний фоторобот все равно больше походит на карандашный набросок, чем на фото. В таком формате мозгу достаточно трудно воспринимать лицо.

2. Решение должно учитывать среднюю погрешность ввиду искаженной возможности запоминания лиц у свидетелей. Это условие - самое трудное для исполнения. Его можно добиться только при обработке обширного прошлого опыта. Ни одно программное обеспечение для составления фоторобота не учитывает такой важный критерий.
3. Минимальное время выполнения. При этом имеется в виду не подготовка и разработка решения, а само использование. Недопустимым временем выполнения считается то, которое занимает больше нескольких минут. В ситуации расследования дела быстрее = лучше.
4. Минимальные затраты на выполнение. Исходит элементарно из целевого потребителя. Ранее мы говорили, что это - федеральные территории, не имеющие материальных средств на качественную систему видеонаблюдения. Соответственно, улучшенное решение должно являться максимально доступным для любого населенного пункта.

Собрав вышеперечисленные условия, можно составить понимание о необходимом решении. Самым удовлетворительным является применение технологий компьютерного зрения. Такой вариант одновременно дешевый (единственные возможные затраты уходят на покупку среды выполнения программы), быстрый (результат, как правило, генерируется не больше минуты), качественный и способный учитывать значения средней ошибки, учиться на прошлом опыте.

Было выделено два возможных типа такой программы:

1. С использованием технологии глубокого обучения (Deep Learning). Такую модель будет необходимо сначала обучить. То есть составить выборку из архива опознанных преступников и их фотороботов и “скормить” их программе. Можно использовать технику обучения с учителем. Разделить выборку на тестовую и тренировочную, где X - фотороботы, а y - лица. Другими словами, программа будет учиться

максимально приближать фоторобота к тому, как будет выглядеть преступник. Такое решение удовлетворяет всем поставленным критериям. Но как и во всем, в нем присутствуют свои нюансы. Во первых, прототип такого решения никак не может быть создан в “любительских” условиях без утрат в точности изображения. Достать несколько десятков тысяч примеров для выборок в домашних условиях - невозможно. Реальным представляется нахождение 50-100 примеров, но в таком случае результаты будут крайне неточными. Можно также на малом количестве примеров сгенерировать еще несколько тысяч похожих, но опять таки, это результирует в утрате корректности работы программы.

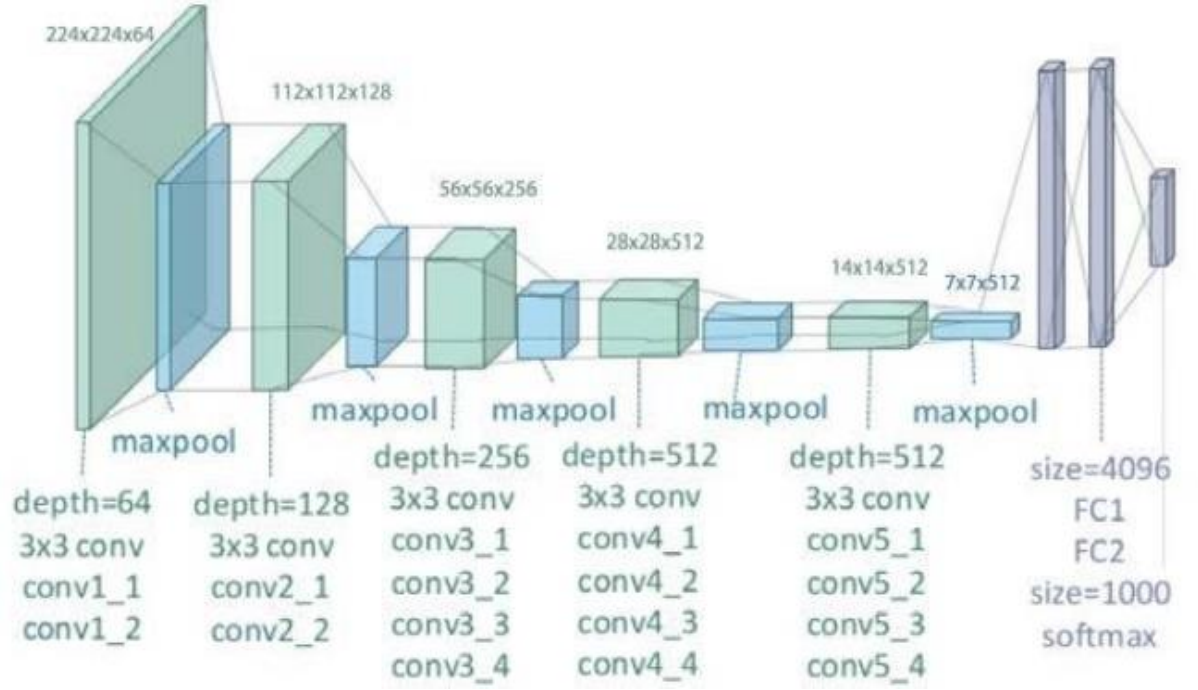
2. С использованием метода стилизации изображений. Такое решение соответствует критериям 1, 3 и 4. Заранее обучать модель не придется: можно взять готовую архитектуру сверточной нейросети. На вход подается фоторобот и образец стиля. Первому тогда присваивается стилистика второго, при этом без потерь в чертах лица. Это решение возможно осуществить не имея доступа к каким либо засекреченным данным, но в нем также есть минус. Такая программа не сможет обучаться на прошлом опыте, соответственно результатов, стремящихся к идеальным, с ней не добьешься. Потерь в остальных критериях не наблюдается: решение быстрое, корректное и бесплатное.

В рамках следующей работы было принято решение воспользоваться вторым методом.

2. ПРОЦЕСС РАБОТЫ

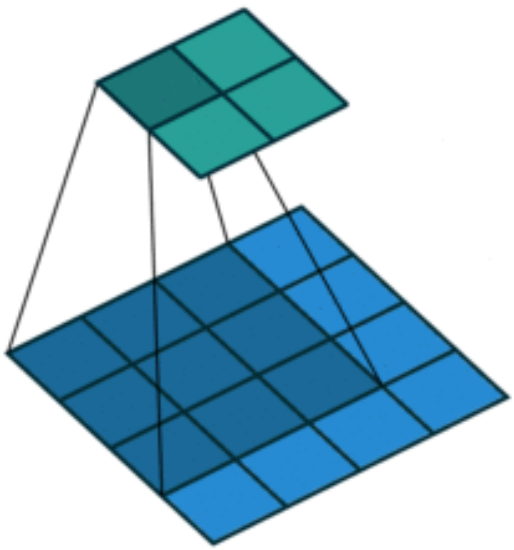
2.1 Про использованные методы

Программа написана на языке Python, фреймворке PyTorch. Фреймворк был выбран ввиду его актуальности и обширного функционала. Использовалась готовая модель сверточной нейросети vgg-19.
Хотелось бы уделить некоторое внимание архитектуре этой нейросети.



(2.1) - архитектура vgg19

На вход подается изображение размером 224 x 224 пикселя. Затем, оно проходит через 2 сверточных слоя conv1_1 и conv1_2 с количеством фильтров 64. Потом maxpool, уменьшающий его линейные размеры. И снова conv, с увеличенной глубиной. Повторив такой цикл несколько раз, результат скармливается в полносвязную нейронную сеть с входным количеством нейронов 4096 на ReLu, а конечным - 1000 на softmax.



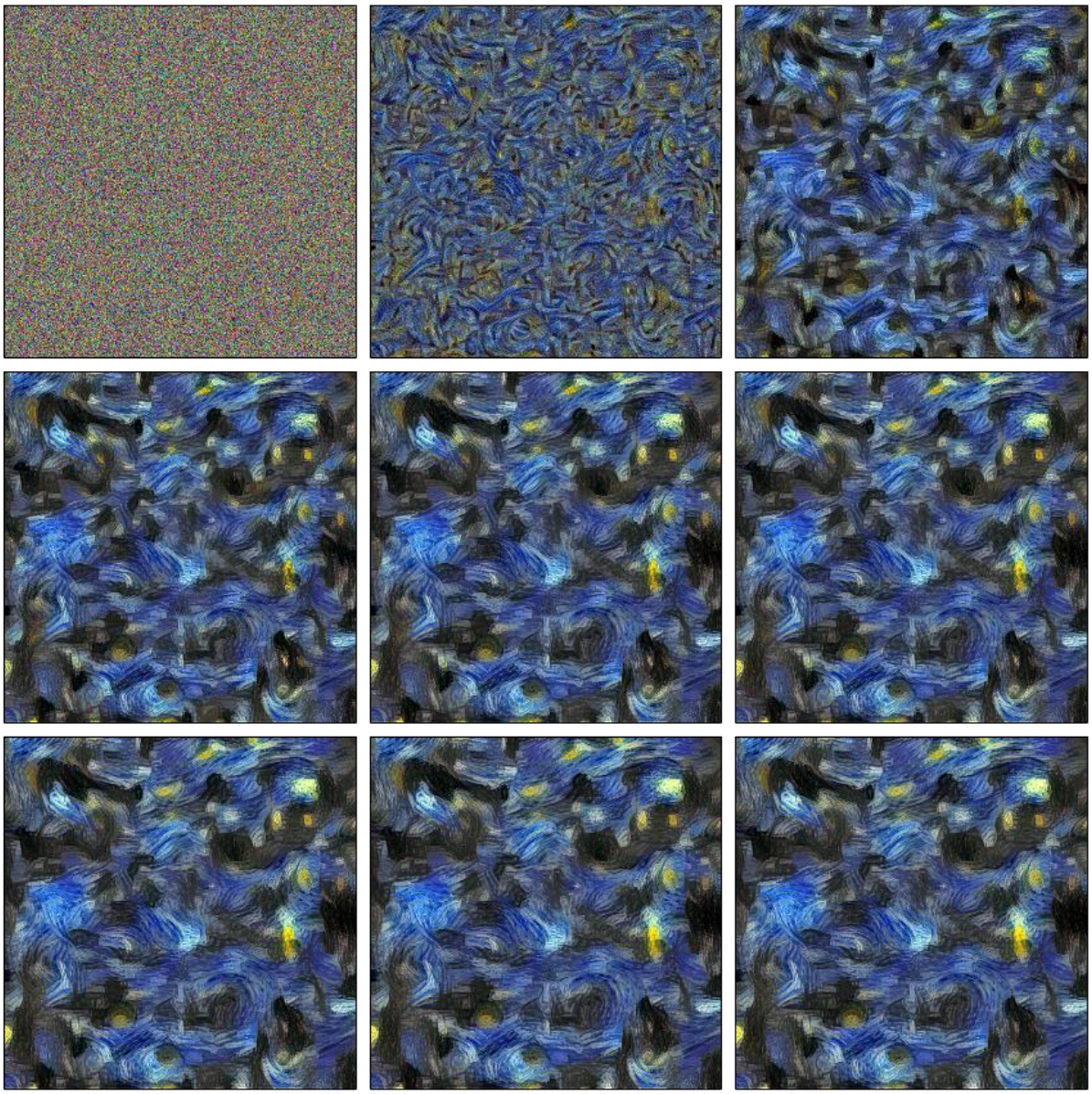
Анимация, демонстрирующая работу сверточного слоя представлена слева.

Данная модель давно показала себя прекрасно работающей в целях

(2.2)

стилизации изображений, за что и была выбрана.

Теперь немного про принцип работы технологии. Оригинальное название - Neural Style Transfer. Идея заключается в следующем. Берем исходное изображение и рассматриваем его пиксели как настраиваемые параметры в алгоритме градиентного спуска. Критерий качества выбираем так, чтобы он уменьшался по мере сближения исходного изображения к стилизованному. Пример на текстуре картин Ван Гога:



(2.3)

Результаты, получаемые таким методом:

(2.4)

2.2 Подбор параметров

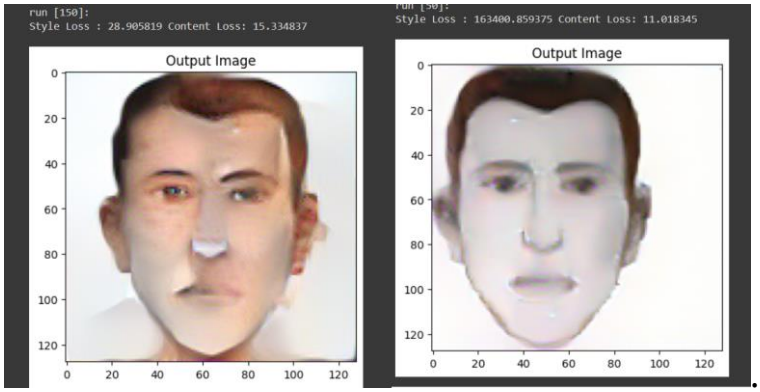
Одним из главных этапов работы являлся подбор параметров. Размер изображения, положение лица относительно фото, количество итераций - все это кардинально меняло работу программы. Необходимо было подобрать такие значения, чтобы результат выходил максимально реалистичным.

Самой важной частью являлся подбор образца стиля. Он должен был быть максимально универсальным. То есть возможные образцы, которые имели редкие черты лица (необычный цвет глаз, шрамы, волосы неестественного цвета, альбиносы и другие) не рассматривались. Было выбрано несколько изображений, с которыми проводились следующие тесты. В общем случае их состоялось около 25. Лучшие результаты показали две модели лица:

(2.5.1)

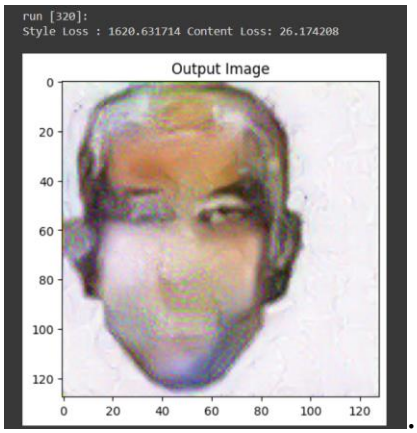
(2.5.2)

Где (2.5.1) - модель лица, имеющая усредненные черты обоих полов. (2.5.2) -модель мужского лица. С изображением (2) выходили результаты с более низким значением ошибки, при этом с (2.5.1) портреты выходили лучше прорисованными, но менее целостными. Примеры результатов генераций (2.5.3).



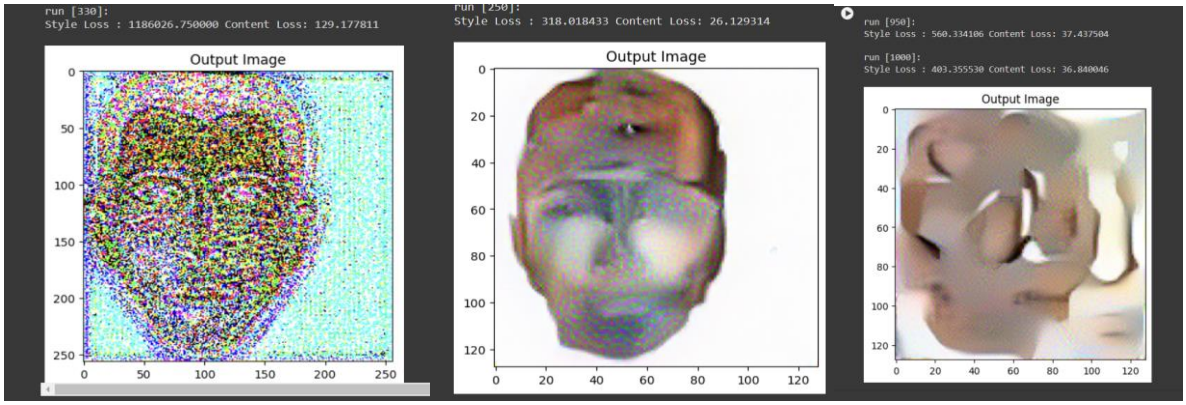
(2.6.1) *(2.6.2)*

Пример (2.5.1):



(2.7)

Некоторые неудачные попытки:



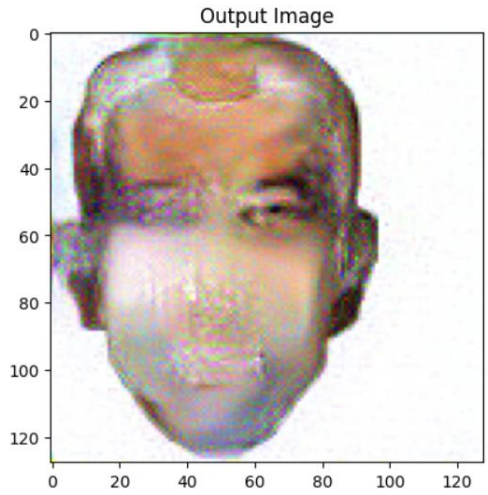
(2.8.1) *(2.8.2)* *(2.8.3)*

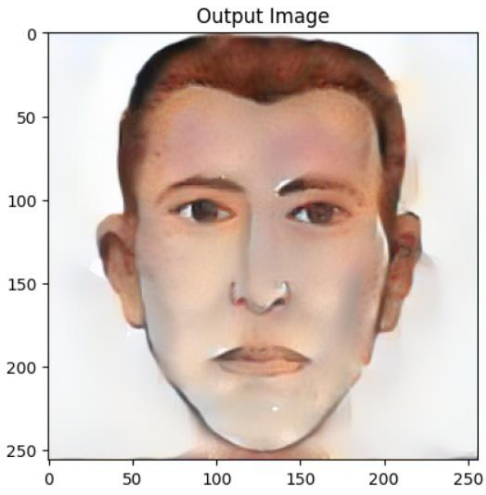
Подобные результаты выходили при критически высоком значении функции ошибки. При совсем низком значении генерировался тот же фоторобот, но уже раскрашенный. Несмотря на аккуратный вид такого результата, он также являлся не подходящим: отсутствовала фактическая стилизация.

Самые низкие показатели ошибки наблюдались при 150 итерациях.

2.3 Результаты, оценка узнаваемости

После многочисленных тестов, было выбрано два лучших результата:





(2.9.1) *(2.9.2)*

Параметры (2.9.1):

128x128 пикселей

300 итераций

образец стиля - (1)

Параметры (2.9.2):

256x256 пикселей

150 итераций

образец стиля - (2)

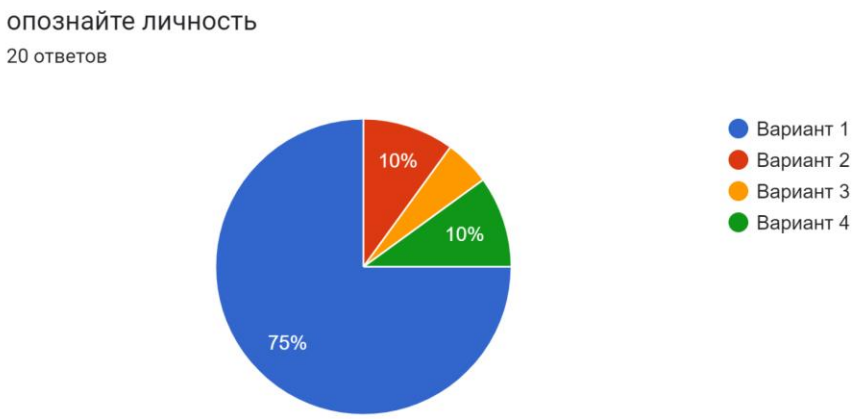
Для оценки результатов был проведен второй социальный опрос, имитирующий процесс опознания преступника. То есть, участникам были представлены результаты генерации и 4 фотографии в одинаковом стиле.

(2.9.3) *(2.9.4)* *(2.9.5)* *(2.9.6)*

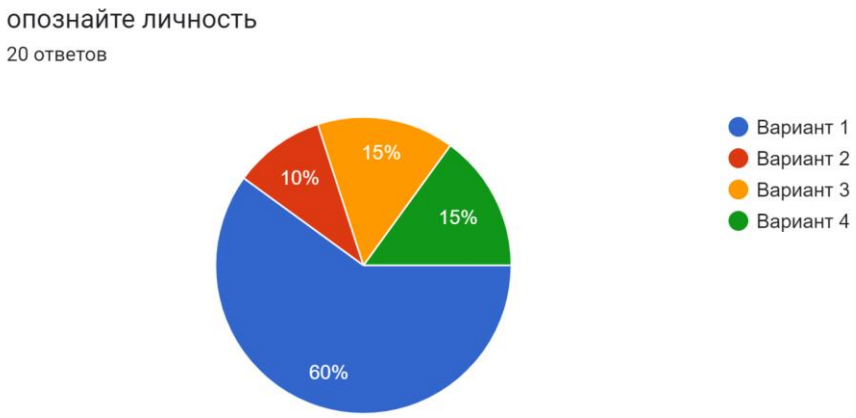
Необходимо было опознать человека по портрету.

Тут (2.9.3) - исходник. Остальные фотографии взяты из сети интернет. Людей специально подбирали из одного временного промежутка, с одним черно- белым фильтром.

Результаты опроса:



(2.9.7)



(2.9.8)

Диаграмма 2.9.7 принадлежит генерации 2.9.1, а 2.9.8 - 2.9.2.

Изображение, являющиеся менее целостным но более реалистичным, показало лучшие результаты.

3. ВЫВОДЫ

3.1 Рекомендации по практическому применению

В рамках данной научной работы удалось создать широкодоступный и быстрый способ повысить узнаваемость портрета преступника с помощью технологии стилизации изображений. Программу можно за минуты запустить на любом компьютере и получить результат, имея на руках только сам фоторобот. Она уже готова к применению в реальной ситуации расследования. Решение значительно повышает процент схожести изображения, дает удовлетворительный результат. Но меньше подходит для обширного ввода в стабильную практику, так как не удовлетворяет всем поставленным в начале работы критериям.

Второе предложенное решение требует большего времени на подготовку и обучение, но дает возможность учитывать сведения из прошлого опыта, что делает его более предпочтительным при варианте его официального дополнения как функции в ПО “Фоторобот”. Именно такой расклад является самым удобным на практике. Сотрудник сначала составляет фоторобот как он это обычно делал, а затем использует возможность обработки с помощью нейросети. В итоге лишь за дополнительную минуту ожидания удается максимально повысить узнаваемость фоторобота, при этом удовлетворяются все поставленные в начале критерии.

3.2 Общий вывод

В итоге, обращаясь к подчеркнутой выше разнице двух решений, их обоих можно назвать удовлетворительными. Они выполняют свой определенный набор функций в контексте поставленных целей и являются значимыми.

При их фактической реализации удасться снизить криминальный риск в малых городах России, сделать нашу страну безопаснее. В будущей работе над данным проектом имеет смысл по возможности воссоздать второй вариант решения с технологией глубокого обучения. Провести такие же опросы, отметить разность процента узнаваемости с ПО и сделать соответствующие выводы.

ЗАКЛЮЧЕНИЕ

После долгой работы, наконец пришло время подводить итоги и делать выводы. Реализуя один из вариантов решения проблемы, удалось повысить узнаваемость фоторобота до 75%. Два из четырех поставленных критериев были удовлетворены полностью, одно частично. Решение дает быструю, доступную возможность редактировать портреты преступников. Изображения выходят реалистично стилизованными, но, к сожалению, не целыми. Тем не менее, такой результат можно считать удовлетворительным.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. В.Н Маркивеч, А.И Дударь, Т.М.Хусяинов Использование субъективного портрета в работе правоохранительных органов: теория и практика // Наука.Мысль - 2016
2. Семенова Мария От выбора носа до поимки жулика. Как в России составляют фотороботы преступника // NGC.ru - 2023
3. Павел Нестеров Стилизация изображений с помощью нейронных сетей: никакой мистики, просто матан // Хабр - 2016
4. Anne Trafton When gender isn’t written all over one’s face // MIT news - 2018
5. Alexis Jacq Neural transfer using torch // PyTorch tutorials
6. Coraline Nehme The effect of post traumatic stress disorders on the ability to recognise facial expressions // Applied psychology opus

ПРИЛОЖЕНИЕ

```
import torch
```

```
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

from PIL import Image
import matplotlib.pyplot as plt

import torchvision.transforms as transforms
from torchvision.models import vgg19, VGG19_Weights

import copy
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
torch.set_default_device(device)
# desired size of the output image
imsize = 128

loader = transforms.Compose([
    transforms.Resize(imsize),  # scale imported image
    transforms.ToTensor()])  # transform it into a torch tensor


def image_loader(image_name):
    image = Image.open(image_name)
    # fake batch dimension required to fit network's input dimensions
    image = loader(image).unsqueeze(0)
    return image.to(device, torch.float)


style_img = image_loader("/content/style.jpg")
content_img = image_loader("/content/input.jpg")

assert style_img.size() == content_img.size(), \
    "we need to import style and content images of the same size"
unloader = transforms.ToPILImage()  # reconvert into PIL image

plt.ion()

def imshow(tensor, title=None):
    image = tensor.cpu().clone()  # we clone the tensor to not do
changes on it
    image = image.squeeze(0)      # remove the fake batch dimension
    image = unloader(image)
    plt.imshow(image)
```

```
    if title is not None:
        plt.title(title)
    plt.pause(0.001) # pause a bit so that plots are updated


plt.figure()
imshow(style_img, title='Style Image')

plt.figure()
imshow(content_img, title='Content Image')
class ContentLoss(nn.Module):

    def __init__(self, target,):
        super(ContentLoss, self).__init__()
        # we 'detach' the target content from the tree used
        # to dynamically compute the gradient: this is a stated value,
        # not a variable. Otherwise the forward method of the criterion
        # will throw an error.
        self.target = target.detach()

    def forward(self, input):
        self.loss = F.mse_loss(input, self.target)
        return input
def gram_matrix(input):
    a, b, c, d = input.size()  # a=batch size(=1)
    # b=number of feature maps
    # (c,d)=dimensions of a f. map (N=c*d)

    features = input.view(a * b, c * d)  # resize F_XL into \hat F_XL

    G = torch.mm(features, features.t())  # compute the gram product

    # we 'normalize' the values of the gram matrix
    # by dividing by the number of element in each feature maps.
    return G.div(a * b * c * d)
class StyleLoss(nn.Module):

    def __init__(self, target_feature):
        super(StyleLoss, self).__init__()
        self.target = gram_matrix(target_feature).detach()

    def forward(self, input):
        G = gram_matrix(input)
```

```
        self.loss = F.mse_loss(G, self.target)
        return input
cnn = vgg19(weights=VGG19_Weights.DEFAULT).features.eval()
cnn_normalization_mean = torch.tensor([0.485, 0.456, 0.406])
cnn_normalization_std = torch.tensor([0.229, 0.224, 0.225])

# create a module to normalize input image so we can easily put it in a
# ``nn.Sequential``
class Normalization(nn.Module):
    def __init__(self, mean, std):
        super(Normalization, self).__init__()
        # .view the mean and std to make them [C x 1 x 1] so that they
can
        # directly work with image Tensor of shape [B x C x H x W].
        # B is batch size. C is number of channels. H is height and W
is width.
        self.mean = torch.tensor(mean).view(-1, 1, 1)
        self.std = torch.tensor(std).view(-1, 1, 1)

    def forward(self, img):
        # normalize ``img``
        return (img - self.mean) / self.std
# desired depth layers to compute style/content losses :
content_layers_default = ['conv_4']
style_layers_default = ['conv_1', 'conv_2', 'conv_3', 'conv_4',
'conv_5']

def get_style_model_and_losses(cnn, normalization_mean,
normalization_std,
                               style_img, content_img,
                               content_layers=content_layers_default,
                               style_layers=style_layers_default):
    # normalization module
    normalization = Normalization(normalization_mean,
normalization_std)

    # just in order to have an iterable access to or list of
content/style
    # losses
    content_losses = []
    style_losses = []
```

```
    # assuming that ``cnn`` is a ``nn.Sequential``, so we make a new
``nn.Sequential``
    # to put in modules that are supposed to be activated sequentially
    model = nn.Sequential(normalization)

    i = 0  # increment every time we see a conv
    for layer in cnn.children():
        if isinstance(layer, nn.Conv2d):
            i += 1
            name = 'conv_{}'.format(i)
        elif isinstance(layer, nn.ReLU):
            name = 'relu_{}'.format(i)
            # The in-place version doesn't play very nicely with the
``ContentLoss``
            # and ``StyleLoss`` we insert below. So we replace with
out-of-place
            # ones here.
            layer = nn.ReLU(inplace=False)
        elif isinstance(layer, nn.MaxPool2d):
            name = 'pool_{}'.format(i)
        elif isinstance(layer, nn.BatchNorm2d):
            name = 'bn_{}'.format(i)
        else:
            raise RuntimeError('Unrecognized layer:
{}'.format(layer.__class__.__name__))

        model.add_module(name, layer)

        if name in content_layers:
            # add content loss:
            target = model(content_img).detach()
            content_loss = ContentLoss(target)
            model.add_module("content_loss_{}".format(i), content_loss)
            content_losses.append(content_loss)

        if name in style_layers:
            # add style loss:
            target_feature = model(style_img).detach()
            style_loss = StyleLoss(target_feature)
            model.add_module("style_loss_{}".format(i), style_loss)
            style_losses.append(style_loss)
```

```
    # now we trim off the layers after the last content and style
losses
    for i in range(len(model) - 1, -1, -1):
        if isinstance(model[i], ContentLoss) or isinstance(model[i],
StyleLoss):
            break

    model = model[:(i + 1)]

    return model, style_losses, content_losses
input_img = content_img.clone()
# if you want to use white noise by using the following code:
#
# ::
#
#    input_img = torch.randn(content_img.data.size())

# add the original input image to the figure:
plt.figure()
imshow(input_img, title='Input Image')
def get_input_optimizer(input_img):
    # this line to show that input is a parameter that requires a
gradient
    optimizer = optim.LBFGS([input_img])
    return optimizer
def run_style_transfer(cnn, normalization_mean, normalization_std,
                       content_img, style_img, input_img,
num_steps=300,
                       style_weight=1000000, content_weight=1):
    """Run the style transfer."""
    print('Building the style transfer model..')
    model, style_losses, content_losses =
get_style_model_and_losses(cnn,
        normalization_mean, normalization_std, style_img, content_img)

    # We want to optimize the input and not the model parameters so we
    # update all the requires_grad fields accordingly
    input_img.requires_grad_(True)
    # We also put the model in evaluation mode, so that specific layers
    # such as dropout or batch normalization layers behave correctly.
    model.eval()
    model.requires_grad_(False)
```

```
    optimizer = get_input_optimizer(input_img)

    print('Optimizing..')
    run = [0]
    while run[0] <= num_steps:

        def closure():
            # correct the values of updated input image
            with torch.no_grad():
                input_img.clamp_(0, 1)

            optimizer.zero_grad()
            model(input_img)
            style_score = 0
            content_score = 0

            for sl in style_losses:
                style_score += sl.loss
            for cl in content_losses:
                content_score += cl.loss

            style_score *= style_weight
            content_score *= content_weight

            loss = style_score + content_score
            loss.backward()

            run[0] += 1
            if run[0] %  50 == 0:
                print("run {}:".format(run))
                print('Style Loss : {:4f} Content Loss: {:4f}'.format(
                    style_score.item(), content_score.item()))
                print()

            return style_score + content_score

        optimizer.step(closure)

    # a last correction...
    with torch.no_grad():
        input_img.clamp_(0, 1)

    return input_img
```

```
output = run_style_transfer(cnn, cnn_normalization_mean,
cnn_normalization_std,
                            content_img, style_img, input_img)

plt.figure()
imshow(output, title='Output Image')

# sphinx_gallery_thumbnail_number = 4
plt.ioff()
plt.show()
```