УДК 004

Разработка системы замены стиля одного изображения стилем другого изображения

Эрдман Александр Алексеевич
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В статье рассмотрен процесс создания программы, способной заменять стиль одного изображения стилем другого изображения. Программа написана на языке программирования Python с использованием нейронной сети и библиотеки TensorFlow. Результатом исследования будет являться программа с возможностью замены стилей изображения и описание её работы.
Ключевые слова: python, нейронная сеть, tensorflow

Development of a system for replacing the style of one image with the style of another image

Erdman Alexander Alekseevich
Sholom-Aleichem Priamursky State University
Student

Abstract
The article describes the process of creating a program capable of replacing the style of one image with the style of another image. The program is written in the Python programming language using a neural network and TensorFlow library. The result of the study will be a program with the ability to replace image styles and a description of its work.
Keywords: python, neural network, tensorflow

1 Введение

1.1 Актуальность

В области искусственного интеллекта и компьютерного зрения в последние годы значительное внимание уделяется передаче стиля изображения. Возможность заменить стиль одного изображения стилем другого изображения имеет множество применений в различных областях, включая моду, дизайн интерьера и рекламу. Однако процесс редактирования стиля изображения вручную — трудоемкая и сложная задача. Следовательно, разработка автоматизированной системы, которая может точно перенести стиль одного изображения на другое изображение, стала важной областью исследований. В этой статье мы обсудим разработку такой системы, которая сможет плавно заменить стиль одного изображения стилем другого изображения.

1.2 Обзор исследований

П.М. Горбунов, Ю.А. Мацкевич и А.В. Чубарь описали автоматизацию выбора модели машинного обучения с использованием методов машинного обучения; обсуждали, как этот процесс можно оптимизировать и сделать более эффективным, чтобы сократить время и затраты, необходимые для выбора лучшей модели для данной задачи [1]. С. А. Сохина, С. А. Немченко раскрыли понятие машинного обучения, а также раскрыли его типы и виды; объяснили принципы функционирования различных видов машинного обучения [2]. С.И.Билал рассмотрел наиболее известные методы, используемые для анализа настроений в твитах в Твиттере, включая методы машинного обучения, глубокого обучения и обработки естественного языка [3]. И.А. Попова рассмотрела рассмотрен ряд систем автоматизированного машинного обучения, описаны их ключевые особенности, решаемые задачи [4]. С. Рокхая представила обзор распространенных методов классификации изображений в машинном обучении, т.е. методов отнесения объекта к одной из категорий на основании его признаков [5]. В.Е. Садовников рассмотрел алгоритмы искусственного интеллекта в качестве основополагающего функции обработки изображений с видеонаблюдения [6]. А.В. Юдин и Самб Рокхая изучили и продемонстрировали в практике различные методы искусственного интеллекта и машинного обучения, используемые для решения задач распознавания образов и интеллектуальной обработки изображений [7].

1.3 Цель исследования

Целью исследования является создание программы на языке программирования Python, способной заменять стили изображения, при помощи нейронной сети и библиотеки TensorFlow.

2 Материалы и методы

Для создания программы используется язык программирования Python и библиотека TensorFlow. В качестве IDE используется Google Colab.

3 Результаты и обсуждения

Для начала работы нужно импортировать необходимые библиотеки такие, как numpy, matplotlib.pyplot, tensorflow, keras, которые являются основными библиотеками, а также вспомогательные библиотеки для работы с файлами и изображениями google.colab files, BytesIO и Image. После импорта модулей и библиотек, необходимо загрузить изображения, с которыми предстоит работать. Для этого используется команда «upl = files.upload()» для загрузки изображений с компьютера. Данные изображения имеют конкретные названия. Каждое изображение имеет свое уникальное название, которое будет использоваться для сохранения его в соответствующую переменную "picture" или "style_picture" Для этого используются две соответствующие команды (рис. 1)

```
[17] upl = files.upload()
     picture = Image.open(BytesIO(upl['car.jpg']))
     style_picture = Image.open(BytesIO(upl['style.jpg']))
```

Рисунок 1. Команды загрузки изображений

В процессе работы программы будет совершена обработка фотографии автомобиля и картины в стиле футуризма, чтобы в результате получить изображение автомобиля, оформленное в футуристическом стиле.

Для создания функционала замены стиля применяется нейронная сеть vgg19. Нейронная сеть vgg19 была разработана компанией Visual Geometry Group в Университете Оксфорда и состоит из 19 слоев. Эта сеть широко используется в задачах компьютерного зрения, таких как классификация изображений, распознавание объектов и детектирование объектов на изображениях. В данном проекте сеть vgg19 используется для извлечения характеристик изображений, которые затем будут использоваться для создания изображения автомобиля в выбранном стиле. Vgg19 работает с изображениями в формате BGR, в то время как выбранные изображения имеют формат RGB. Следовательно, для корректной работы необходимо преобразовать формат RGB в BGR. Это выполняется с помощью функции preprocess_input (рис. 2), которая заменяет среднее значение у цветовых компонентов: (B) 103.939, (G) 116.779 и (R) 123.68.

```
[6] x_picture = keras.applications.vgg19.preprocess_input( np.expand_dims(picture, axis=0) )
    x_style_picture = keras.applications.vgg19.preprocess_input(np.expand_dims(style_picture, axis=0))
```

Рисунок 2. Функция преобразования формата RGB в BGR

Добавление оси с индексом 0 с помощью параметра axis=0 позволяет изменить форму массива таким образом, чтобы он мог быть обработан сетью vgg19. В дополнение к функции preprocess_input, реализована обратная функция, которая преобразует изображение из формата BGR обратно в формат RGB для удобного просмотра полученного результата (рис. 3). Функция обратного назначения используется для преобразования изображений из формата BGR, необходимого для работы сети vgg19, в формат RGB, который используется для визуализации результатов. В начале функции с помощью метода np.squeeze(x, 0) удаляется нулевая ось, которая была добавлена в функции preprocess_input. Затем проверяется количество осей - должно быть три, используя assert len(x.shape) == 3. Далее к соответствующим цветовым компонентам добавляются средние значения, которые ранее были вычтены в функции preprocess_input. После этого цветовые компоненты меняют местами, чтобы из формата BGR получить формат RGB. Чтобы убедиться, что цветовые компоненты не выходят за допустимый диапазон, используется метод np.clip(x, 0, 255).astype('uint8'), который обрезает значения, меньшие 0 и большие 255, и преобразует массив в формат uint8.

```
[7]  def deprocess_img(processed_img):
       x = processed_img.copy()
       if len(x.shape) == 4:
         x = np.squeeze(x, 0)
       assert len(x.shape) == 3, ("Предупреждение")
       if len(x.shape) != 3:
         raise ValueError("Error")

       # perform the inverse of the preprocessing step
       x[:, :, 0] += 103.939
       x[:, :, 1] += 116.779
       x[:, :, 2] += 123.68
       x = x[:, :, ::-1]

       x = np.clip(x, 0, 255).astype('uint8')
       return x
```

Рисунок 3. Функция преобразования BGR в RGB

Наконец, функция возвращает изображение в формате RGB. После загрузки сети vgg19 следует ее предварительная обработка и настройка перед использованием (рис. 4).

```
[8]  vgg = keras.applications.vgg19.VGG19(include_top=False, weights='imagenet')
     vgg.trainable = False
```

Рисунок 4. Загрузка сети VGG19

Параметр include_top=False отключает полносвязный слой нейронной сети на конце, а параметр weights='imagenet' задает использование весов, обученных на наборе данных 'imagenet' из 10 миллионов изображений. Команда vgg.trainable = False делает веса неизменяемыми (необучаемыми). В дальнейшем будет использоваться данная сеть, на ее вход будет подаваться изображение, а на выходе будут получаться карты признаков на определенных слоях. Затем создаются вспомогательные коллекции, которые содержат имена слоев, выделенных в предыдущем шаге с помощью сети vgg19 (рис. 5).

Были выбраны имена слоев в соответствии со структурой сети vgg19. Цифра после слова "block" указывает на уровень слоя. Для вычисления контента будет использоваться последний сверточный слой "block5_conv2". Для вычисления потерь стиля будут использоваться пять сверточных слоев, определенных в переменной "style_layers": "block1_conv1" - "block5_conv1". Последние две команды на рисунке 7 указывают количество данных слоев. Эти переменные "content_layers" и "style_layers" будут использоваться далее в программе в качестве вспомогательных.

```
[ ]  content_layers = ['block5_conv2']

     style_layers = ['block1_conv1',
                     'block2_conv1',
                     'block3_conv1',
                     'block4_conv1',
                     'block5_conv1'
                    ]

     num_content_layers = len(content_layers)
     num_style_layers = len(style_layers)
```

Рисунок 5. Создание слоёв для работы нейронной сети

Для построения модели нейронной сети используется класс keras.models.Model(inputs, outputs), который принимает на вход коллекцию входов и коллекцию выходов и на основе связей между слоями строит модель сети. В данном случае, модель будет построена на основе иерархии слоев, включая полно связные слои и сверточные слои, и будет использоваться архитектура VGG19. Используя переменные content_layers и style_layers будут выделены соответствующие слои (рис. 6).

```
style_outputs = [vgg.get_layer(name).output for name in style_layers]
content_outputs = [vgg.get_layer(name).output for name in content_layers]
```

Рисунок 6. Выделение слоев style и content

В обоих переменных происходит обращение к сети vgg19 и из неё происходит выделение выходов (output), которые соответствуют именам block5_conv2, block1_conv1-block5_conv1. Для переменной style_outputs выделяются слои block1_conv1 - block5_conv1, которые находятся в коллекции style_layers. Для переменной content_outputs выделяется только один слой block5_conv2. Затем используется операция объединения выходов с помощью команды model_outputs = style_outputs + content_outputs. Для вывода на экран соответствующих слоев используется следующий код: print(vgg.input), for m in model_outputs:, print(m). Затем создается модель сети, используя класс Model: model = keras.models.Model(vgg.input, model_outputs), где указывается один вход и необходимые выходы, полученные ранее с помощью объединения (рис. 7).

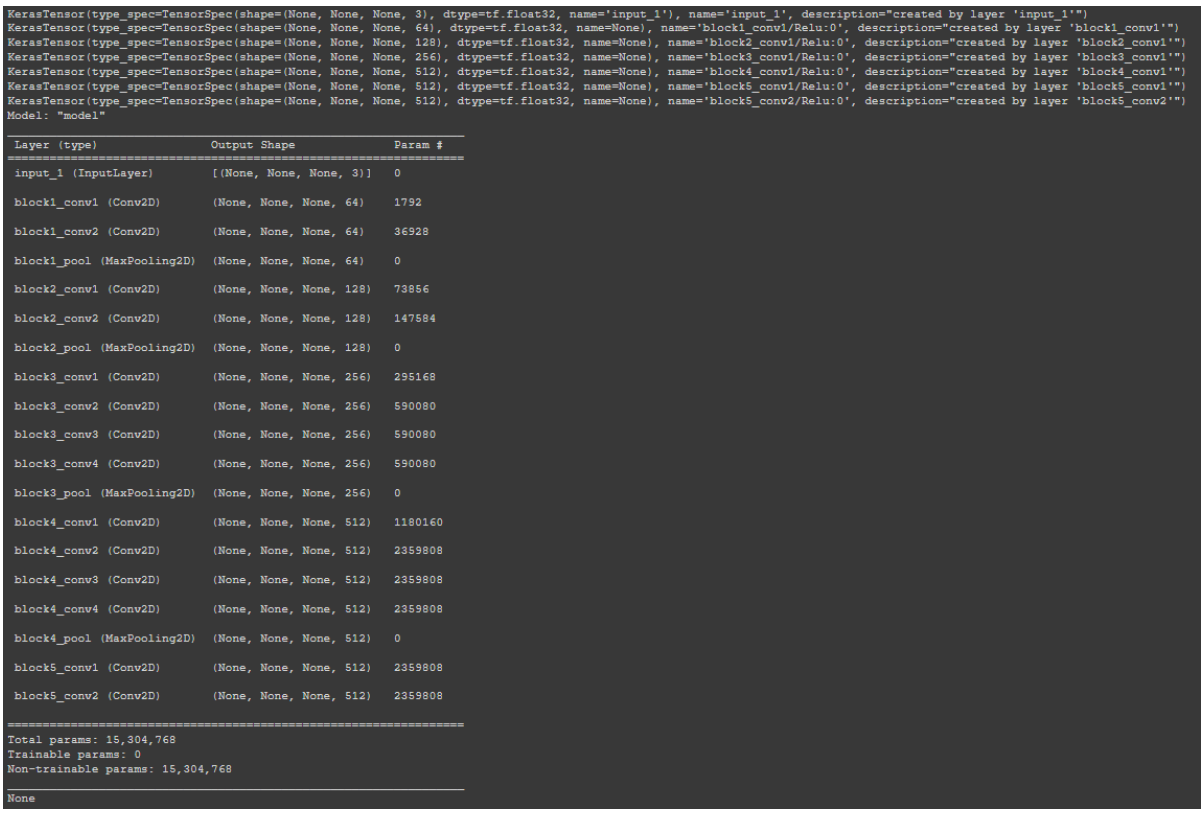

Рисунок 7. Вывод слоёв и структуры нейронной сети

После выполнения операций на экран были выведены тензоры - выходные слои и структура построенной нейронной сети.

После создания сети необходимо создать функцию, которая будет вычислять потери. Сначала создаётся функция get_content_loss, которая будет определять потери, связанные с контентом (8).

```
def get_content_loss(base_content, target):
  return tf.reduce_mean(tf.square(base_content - target))
```

Рисунок 8. Функция потерь по контенту

После пропуска исходного изображения с автомобилем через сверточную сеть получается коллекция тензоров, хранящихся в переменной base_content. После пропуска преобразованного изображения через эту же сеть получается вторая коллекция тензоров target. С помощью операции tf.square из TensorFlow вычисляется квадрат разницы между двумя коллекциями, а затем используется функция tf.reduce_mean для получения среднего значения. Таким образом, определяются потери по контенту.

Для вычисления потерь по стилям используется функция матрицы Грамма (9).

```
def gram_matrix(input_tensor):
  channels = int(input_tensor.shape[-1])
  a = tf.reshape(input_tensor, [-1, channels])
  n = tf.shape(a)[0]
  gram = tf.matmul(a, a, transpose_a=True)
  return gram / tf.cast(n, tf.float32)
```

Рисунок 9. Матрица Грамма

Данная функция вычисляет матрицу грамма, которая используется для вычисления сходства или корреляции между векторами признаков изображения. В качестве аргумента используется тензор input_tensor. Входной тензор представлен трёхмерным, содержащий количество каналов, которые являются картами признаков. Для выбора каналов используется операция channels = int(input_tensor.shape[-1]). В ходе операции tf.reshape происходит преобразование трёхмерного тензора в двумерный. Преобразование 3D-тензора в 2D-тензор часто необходимо, когда нам нужно передать данные в модель машинного обучения, которая ожидает двумерный ввод. Трёхмерный тензор содержит три измерения (длина, ширина и глубина), последнее из которых является картами признаков. Третье измерение записано в переменной channels. В преобразованном двумерном тензоре будет два измерения, одно из которых – это произведение длины и ширины трехмерного тензора, а вторым измерением является глубина трехмерного тензора (карта признаков). Переменная gram включает в себя операцию умножение транспонированной переменной a на исходную a. Операция gram / tf.cast(n, tf.float32) разделяет полученное выше произведение в переменной gram для того, чтобы усреднить вычисления. В результате выполнение данных операций получается матрица Грамма.

Следующая функция вычисляет стиль для строго определённого слоя нейронной сети (10).

```
def get_style_loss(base_style, gram_target):
  gram_style = gram_matrix(base_style)
  return tf.reduce_mean(tf.square(gram_style - gram_target))
```

Рисунок 10. Функция вычисления стиля

Для разных слоёв нейронной сети будет происходить вычисление рассогласование по стилям. Параметр base_style – это карта стилей формируемого изображения, параметр gram_target – матрица Грамма для соответствующего слоя целевого изображения. В переменной gram_style происходит вычисление матрицы Грама для формируемого изображения. Последняя операция данной функции вычисляет среднее арифметическое квадрата рассогласований.

Общей функцией, вычисляющей все потери, является функция compute_loss (рис. 11).

```
def compute_loss(model, loss_weights, init_image, gram_style_features, content_features):
  style_weight, content_weight = loss_weights

  model_outputs = model(init_image)

  style_output_features = model_outputs[:num_style_layers]
  content_output_features = model_outputs[num_style_layers:]

  style_score = 0
  content_score = 0


  weight_per_style_layer = 1.0 / float(num_style_layers)
  for target_style, comb_style in zip(gram_style_features, style_output_features):
    style_score += weight_per_style_layer * get_style_loss(comb_style[0], target_style)


  weight_per_content_layer = 1.0 / float(num_content_layers)
  for target_content, comb_content in zip(content_features, content_output_features):
    content_score += weight_per_content_layer* get_content_loss(comb_content[0], target_content)

  style_score *= style_weight
  content_score *= content_weight


  loss = style_score + content_score
  return loss, style_score, content_score
```

Рисунок 11. Общая функция потерь

В качестве параметров, данная функция принимает модель нейронной сети (model), los_weights – содержит в себе два элемента content_weight и style_weight. Параметр init_image – это формируемое изображение, которое пропускается через нейронную сеть с помощью операции model(init_image), в результате которой получаются значения на каждом сверточном слое (model_outputs).

Из полученных результатов происходит выделение из коллекции model_outputs карты признаков для стилей (style_output_features) и для контента (content_output_features). Переменные style_score и content_score хранят величины потерь для стилей и контента. Далее происходит определение весов для суммирования потерь стилей каждого слоя (weight_per_style_layer). В цикле происходит перебор матриц Грама для стилевого изображения, которые заранее вычисленные (параметр gram_style_features в функции compute_loss), и карт признаков, которые были вычислены с помощью нейронной сети. Внутри цикла происходит суммирования квадрата рассогласований для каждого слоя.

Для контентного слоя происходит тоже самое. Так как слой в данном случае один, следовательно цикл сработает только один раз. В переменных style_score и content_score происходит умножение на вычисленные переменные на величины style_weight и content_weight. Далее следует операция суммирования. Loss – это общий критерий вычисленный критерий качества. В результате функция compute_loss возвращает общие потери, потери по стилю и контенту.

После определения функций потерь в коде программы, задаются значения для нескольких переменных, которые будут использоваться в дальнейшем. Например, переменная num_iterations, определяющая количество итераций, которые будут использоваться в процессе формирования нового изображения. Значение этой переменной равно 100. Также определены переменные content_weight и style_weight, которые являются коэффициентами, отвечающими за вклад контента и стиля в формируемое

изображение соответственно. Значение content_weight равно 1000, что означает, что на изображении будет сохранено большинство контента из исходного изображения, содержащего автомобиль. Значение style_weight равно 0.01, что указывает на то, что стиль второго изображения будет учтен в небольшой степени при формировании нового изображения.

С помощью метода get_feature_representations(model) два исходных изображения (изображение с автомобилем (контент) и с футуристической картиной (стиль)) проходят через нейронную сеть. На выходе данной нейронной сети получатся карты признаков по стилю и контенту. После завершения данной операции, следует вычисление матрицы Грама для стилевого изображения.

Переменная init_image содержит начальное изображение, которое соответствует изображению с автомобилем. Это изображение будет изменяться с помощью библиотеки TensorFlow. Для корректной работы с изображением, оно должно быть преобразовано в переменную типа tf.Variable. Для выполнения алгоритма градиентного спуска, создаются оптимизаторы. В данном случае используется оптимизатор AdamOptimizer из TensorFlow с заданным learning_rate=2, beta1=0.99, epsilon=1e-1. Оптимизатор позволяет настроить параметры обучения модели, такие как скорость обучения и коэффициенты регуляризации.

Параметр learning_rate — это гиперпараметр, который управляет размером шага на каждой итерации при движении к минимуму функции потерь. Другими словами, он определяет скорость обновления параметров модели во время обучения.

Параметр beta1 — это гиперпараметр, который управляет скоростью экспоненциального затухания для оценок первого момента. Это значение от 0 до 1, которое определяет, какой вес следует придавать прошлым градиентам при вычислении текущей оценки градиента. Более высокое значение beta1 означает, что прошлым градиентам придается больший вес, что может помочь сгладить оценки градиента и сделать оптимизацию более стабильной.

Параметр epsilon — это гиперпараметр, который предотвращает деление на ноль при вычислении правила обновления. Это небольшая величина, добавленная к знаменателю для обеспечения числовой стабильности.

Переменная iter_count - это счетчик итераций, который изначально равен 1. Каждая итерация представляет собой один проход через оптимизируемую функцию потерь. В данном случае, на каждой итерации происходит вычисление градиента потерь по отношению к переменной init_image, после чего переменная init_image обновляется в направлении уменьшения потерь с помощью оптимизатора. Таким образом, с каждой итерацией изображение становится все более похожим на искомое. Количество итераций определяется заранее и является гиперпараметром, который может быть подобран экспериментальным путем.

Переменные best_loss и best_img являются вспомогательными в рамках данного алгоритма. Первая переменная хранит наименьшие потери, которые достигаются в процессе вычислений сети, а вторая переменная содержит

лучшее сформированное изображение, соответствующее минимальным потерям. Коллекция loss_weights содержит два параметра: style_weight и content_weight, описанные ранее. Чтобы удобно работать с параметрами, они сведены в словарь, что позволяет легко получать доступ к нужным параметрам для выполнения определенных.

Метрикой является переменная loss, которая используется для хранения значения потери (loss) при стилизации изображения и вычисляется с помощью функции compute_loss. Метрика loss представляет собой суммарную потерю (style_score + content_score) и используется для оценки качества стилизации изображений. Переменная loss является числовым значением, которое позволяет оценить, насколько хорошо текущая модель стилизации выполняет свою задачу. Чем меньше значение loss, тем ближе стилизованное изображение к желаемому результату.

Часть кода, которая отвечает за преобразование изображения из BGR формата в RGB, а также массив img, который будет содержать все изображения, которые будут сформированы в процессе работы нейронной сети (рис. 12).

```
norm_means = np.array([103.939, 116.779, 123.68])
min_vals = -norm_means
max_vals = 255 - norm_means
imgs = []
```

Рисунок 12. Преобразование изображения из BGR в RGB

В конце нейронной сети прописывается запуск алгоритма градиентного спуска (рис.13).

Функция compute_loss обрабатывает формируемое изображение через нейронную сеть и возвращает значения потерь. Вызов функции осуществляется в области видимости объекта GradientTape, который записывает все необходимые величины для последующего вычисления градиента для изменения пикселей формируемого изображения в соответствии с функцией потерь. С помощью объекта GradientTape происходит автоматическое вычисление всех необходимых производных, с помощью которых для подстраиваемых параметров (пикселей изображения) будут вычисляться градиенты, относительно пикселей изображения. С помощью команды opt.apply_gradients([(grads, init_image)]) происходит применение вычисленного градиента к пикселем формируемого изображения. С помощью метода tf.clip_by_value происходит ограничение пикселя изображения минимальными (min_vals) и максимальными (max_vals) значениями, так как каждая цветовая компонента может находиться в строго определённых числовых значениях, и соответственно выход за данные числовые значения не допускается. С помощью условия If происходит проверка изображений, для которых получились минимальные потери.

```
for i in range(num_iterations):
    with tf.GradientTape() as tape:
       all_loss = compute_loss(**cfg)

    loss, style_score, content_score = all_loss
    grads = tape.gradient(loss, init_image)

    opt.apply_gradients([(grads, init_image)])
    clipped = tf.clip_by_value(init_image, min_vals, max_vals)
    init_image.assign(clipped)

    if loss < best_loss:
      best_loss = loss
      best_img = deprocess_img(init_image.numpy())

      plot_img = deprocess_img(init_image.numpy())
      imgs.append(plot_img)
      print('Iteration: {}'.format(i))
```

```
Iteration: 5
Iteration: 6
Iteration: 7
Iteration: 8
Iteration: 9
Iteration: 10
Iteration: 11
Iteration: 12
```

Рисунок 13. Алгоритм градиентного спуска

После проверки и выбора изображения происходит его сохранение. Вывод изображения осуществляется с помощью команд plt.imshow и print (рис. 14).

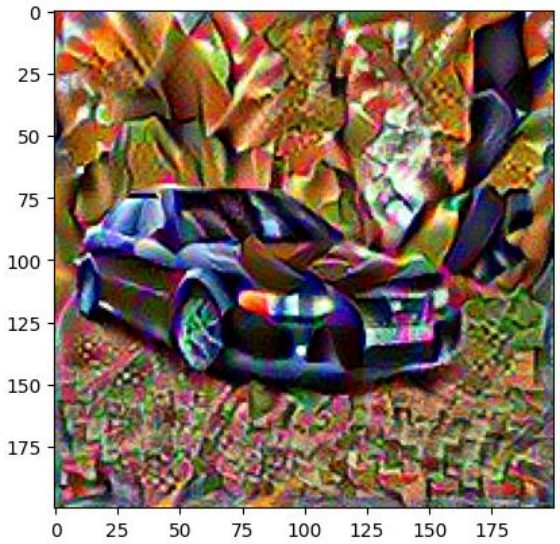

Рисунок 14. Результат работы сети

Для тестов были взяты несколько изображений, для проверки работы сети (рис. 15, 16 и 17)

Рисунок 15. Результат работы сети (снизу) на основе двух изображений (сверху: слева исходное изображение, справа изображение стиля)

Рисунок 16. Результат работы сети (снизу) на основе двух изображений (сверху: слева исходное изображение, справа изображение стиля)

Рисунок 17. Результат работы сети (снизу) на основе двух изображений (сверху: слева исходное изображение, справа изображение стиля)

В результате исследования была построена система с нейронной сетью, которая способна заменять стили двух изображений с сохранением содержимого изменяемого изображения. Для проверки работоспособности сети были проведены несколько тестов с несколькими изображениями, в результате которых были получены удовлетворительные результаты работы системы.

Библиографический список

1. Горбунов П. М., Мацкевич Ю. А., Чубарь А. В. Машинное обучение. Автоматизация подбора модели машинного обучения // Робототехника и искусственный интеллект. 2021. С. 155-160.
2. Сохина С. А., Немченко С. А. Машинное обучение. Методы машинного

обучения // Современная наука в условиях модернизационных процессов: проблемы, реалии, перспективы. 2021. С. 165-168.
3. Билал С. И. Обзор анализа настроений с использованием методов машинного обучения и глубокого обучения // Инновационные технологии, экономика и менеджмент в промышленности. 2022. С. 195-199.
4. Попова И. А. Системы automl как современный инструмент для построения моделей машинного обучения // StudNet. 2022. Т. 5. №. 1. С. 815-825.
5. Рокхая С. Различные типы обработки изображений с помощью искусственного интеллекта // Интернаука. 2021. №. 7-1. С. 17-21.
6. Садовников В. Е. Возможности видеонаблюдения с последующей обработкой изображения на основе алгоритмов искусственного интеллекта // Россия молодая. 2021. С. 31521.1-31521.4.
7. Юдин А. В., Рокхая С. Различные типы обработки изображений с помощью искусственного интеллекта // Общество-наука-инновации: сборник статей Международной научно-практической. 2021. С. 8.