Инженерный вестник Дона, №11 (2022)
ivdon.ru/ru/magazine/archive/n11y2022/8006

Реализация алгоритма переноса стилей изображений с использованием Keras и TensorFlow

А.Н. Мирошниченко[1], *В.И.Шиян*[1,2]

[1]*Кубанский государственный университет, Краснодар*
[2]*Кубанский государственный технологический университет, Краснодар*

Аннотация: В данной статье рассматривается вариант решения задачи переноса стилей изображений с помощью сверточной нейронной сети. Вначале рассматривается общая математическая концепция алгоритма переноса стилей изображений, а затем - реализация данного алгоритма с использованием Keras и TensorFlow – библиотек для языка программирования Python, а также сверточной нейронной сети VGG19, входящей в состав данных библиотек.
Ключевые слова: машинное творчество, перенос стиля изображения, сверточная нейронная сеть, библиотека Keras, библиотека TensorFlow, язык программирования Python, сеть VGG19.

В настоящее время нейронные сети [1, 2] стали очень популярны [3]. Они позволяют решать сложные задачи [4, 5], улучшать решения, которые уже существуют [6 – 8]. Они нашли очень широкое применение в так называемом машинном творчестве [9, 10], в частности, в переносе стилей. Перенос стилей – одно из современных и креативных приложений сверточных нейронных сетей.

Задачу переноса стилей изображений можно сформулировать, как перевод одного изображения в другое, то есть, перевод одного представления какого-нибудь объекта x в другое – y. В данной задаче входными данными являются два изображения: контентное $x_{content}$, с которого заимствуется содержание, второе – стилевое x_{style}, с которого берется стиль – набор признаков, которые характеризуют данное изображение. Выходным данным является стилизованное изображение y, которое объединено из двух – контентного и стилевого.

Рассмотрим вариант решения задачи переноса стилей изображений с помощью сверточной нейронной сети.

Общая математическая концепция алгоритма переноса стилей изображений

Общая идея алгоритма переноса стилей изображений заключается в том, что нужно взять исходное (контентное) изображение, далее рассматривать его пиксели, как настраиваемые параметры в алгоритме градиентного спуска. Критерий качества стилизованного изображения J нужно выбрать таким образом, чтобы он уменьшался по мере приближения контентного изображения к стилевому.

Стоит отметить, что в целом можно обойтись и без нейронной сети. Но именно нейронная сеть позволит «понимать»:

1. Степень соответствия преобразованного изображения исходному.

2. Степень стилизации преобразованного изображения.

Данные критерии можно вычислять независимо друг от друга с помощью нейронной сети и затем сложить с определенными весами для вычисления общего критерия качества стилизованного изображения J. Фактически критерий качества стилизованного изображения J – ключевой элемент алгоритма.

Покажем, как можно вычислить степень соответствия преобразованного изображения исходному. В классических алгоритмах обработки изображений данная оценка рассчитывается по формуле (1):

$$E = \frac{1}{2}\sum_{i,j}\left(x_{i,j} - y_{i,j}\right)^2. \quad (1)$$

Но, стоит отметить, что для решения поставленной задачи приведенная формула не подходит, так как в таком случае не будет учитываться структура изображения, например, линии, овалы и тому подобное. Нейронная сеть как раз и будет использоваться для того, чтобы можно было оценивать стилизованные изображения на более абстрактном уровне. В рамках поставленной задачи можно воспользоваться уже готовой сверточной

нейронной сетью VGG19. Она уже обучена на базе изображений ImageNet, которая состоит примерно из 10 миллионов различных полноцветных изображений. Слои данной нейронной сети как раз описывают изображение набором признаков. Далее рассмотрим один из последних слоев нейронной сети VGG19 (рис. 1) и вычислим степень соответствия стилизованного изображения и исходного по формуле (2):

$$J_C = \frac{1}{4 \cdot n_H \cdot n_W \cdot n_C} \cdot \sum_{i=1}^{n_H} \sum_{j=1}^{n_W} \sum_{k=1}^{n_C} \left(t_{i,j,k} - p_{i,j,k} \right)^2, \quad (2)$$

где $\{t_{i,j,k}\}$ – набор признаков исходного изображения, $\{p_{i,j,k}\}$ – набор признаков стилизованного изображения, n_H, n_W – высота и ширина карты признаков, n_C – число каналов, J_C – степень соответствия стилизованного изображения и исходного. Множитель перед суммой используется для нормирования показателя качества.

Через нейронную сеть VGG19 пропускается исходное изображение и на последнем слое получается набор признаков $\{t_{i,j,k}\}$. Затем пропускается стилизованное изображение и получается набор признаков $\{p_{i,j,k}\}$.

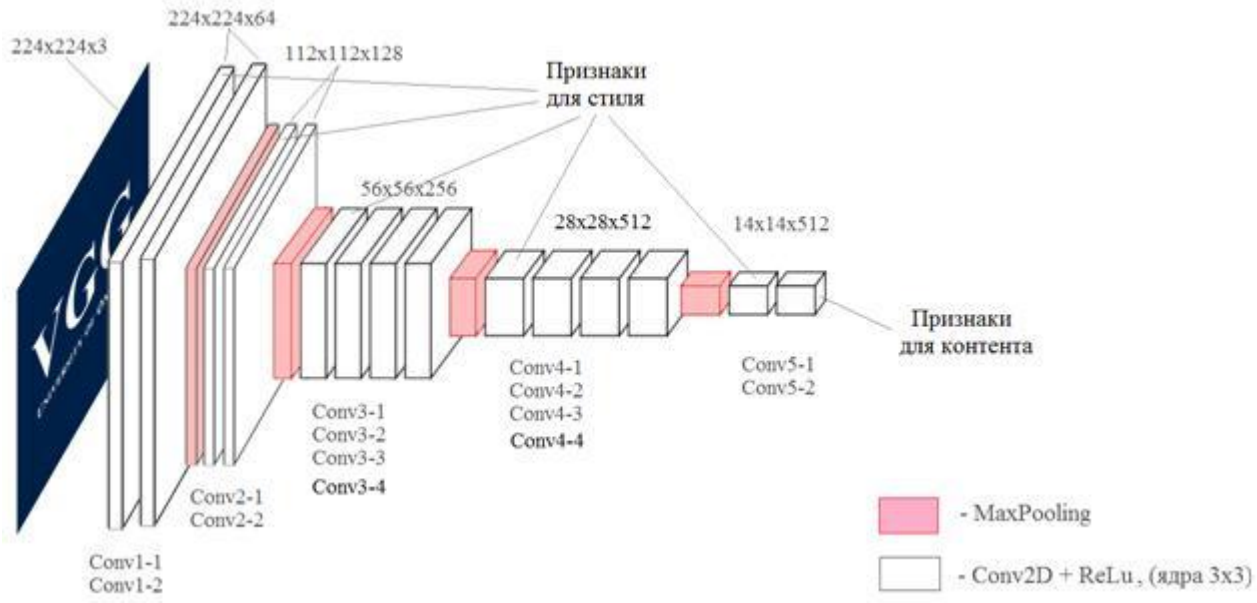

Рис. 1. – Нейронная сеть VGG19

Теперь покажем, как можно вычислить степень стилизации стилизованного изображения. Стоит отметить, что это было непосильной задачей до появления нейронных сетей, поэтому с помощью нейронной сети VGG19 вычислим еще и степень стилизации преобразованного изображения.

Через нейронную сеть нужно пропустить стилевое изображение. Затем рассмотрим тензор предпоследнего слоя Conv5-1. Для удобства дальнейших вычислений рассмотрим его в виде матрицы (рис. 2):

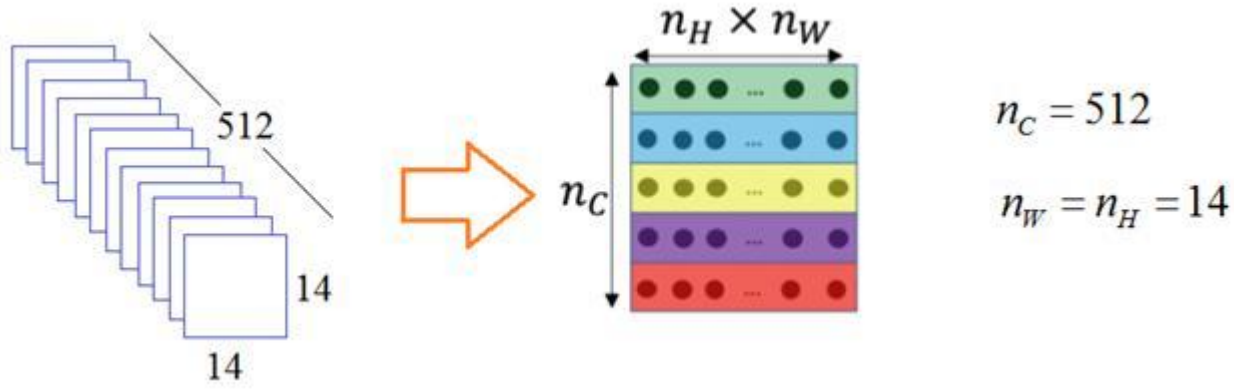

Рис. 2. – Предпоследний слой Conv5-1 в виде матрицы

Далее данную матрицу нужно умножить на саму себя в транспонированном виде. Получим так называемую матрицу Грама (рис. 3).

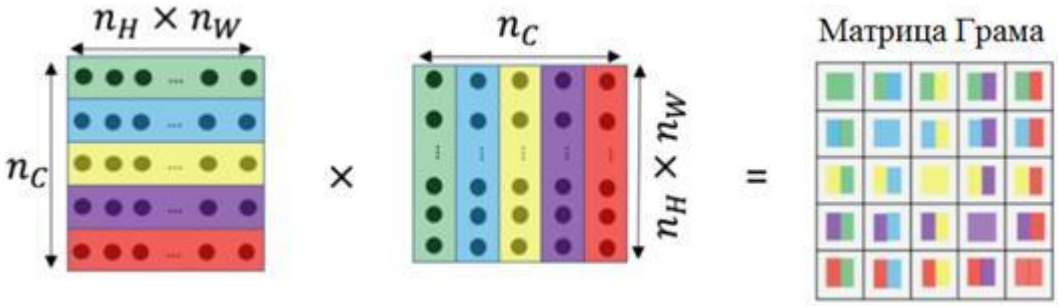

Рис. 3. – Матрица Грама

Данная матрица содержит скалярные произведения признаков для разных каналов. Элементы главной диагонали – скалярные произведения векторов с самими собой, а остальные элементы – скалярные произведения векторов для разных каналов.

Стоит отметить, что скалярное произведение в данном случае будет выступать в качестве меры схожести. То есть, матрица Грама будет отражать меру схожести признаков между разными каналами. Это и будет выступать в роли степени стилизации стилизованного изображения на текущем слое нейронной сети.

Для того, чтобы вычислить величину рассогласования стилей между стилевым и стилизованным изображениями, пропустим через нейронную сеть стилизованное изображение и для него также найдем матрицу Грама.

Таким образом, получатся:

1. G^s – матрица Грама для стилевого изображения.
2. G^p – матрица Грама для стилизованного изображения.

С помощью данных матриц можно вычислить величину рассогласования стилей между стилевым и стилизованным изображениями на текущем слое нейронной сети, можно по формуле (3):

$$J_S^{(5)} = \frac{1}{n_C^2 \cdot n_H^2 \cdot n_W^2} \cdot \sum_{i=1}^{n_C} \sum_{j=1}^{n_C} \left(G_{i,j}^p - G_{i,j}^s \right)^2 . \quad (3)$$

Здесь вычисляется квадрат евклидового расстояния между матрицами Грама для стилевого и стилизованного изображений. Верхний индекс (5) означает, что данное вычисление нужно выполнять на слое Conv5-1.

Стоит отметить, что данной величины для вычисления величины рассогласования стилей между изображениями не достаточно. Стиль, в отличие от объектов (линий, овалов и тому подобного), на разных уровнях абстракции имеет важные детали. Таким образом, нужно повторить упомянутые ранее операции и для слоев Conv1-1, Conv2-1, Conv3-1 и Conv4-1. Так получим следующие величины:

$$J_S^{(1)}, J_S^{(2)}, J_S^{(3)}, J_S^{(4)}, J_S^{(5)}.$$

Для вычисления степени стилизации стилизованного изображения, сложим данные величины с определенными весами, формула (4):

$$J_S = \sum_{i=1}^{5} \lambda_i \cdot J_S^{(i)}. \quad (4)$$

Таким образом, общий критерий качества стилизованного изображения можно вычислить по формуле (5):

$$J = \alpha J_C + \beta J_S, \quad (5)$$

где $\alpha, \beta \in [0; 1]$ – веса, показывающие, насколько важно учитывать содержимое или стиль.

Определив общий критерий качества стилизованного изображения, можно воспользоваться, например, алгоритмом градиентного спуска, чтобы минимизировать данный показатель J.

Реализация алгоритма переноса стилей изображений

Вначале нужно импортировать необходимые библиотеки (рис. 4).

```
# импортируем необходимые библиотеки
import numpy as np
import matplotlib.pyplot as plt
from io import BytesIO
from PIL import Image
import tensorflow as tf
from keras import layers
from tensorflow import keras
```

Рис. 4. – Импорт необходимых библиотек

Затем нужно загрузить два изображения – контентное и стилевое (рис. 5). Пусть первое хранится в файле content_image.jpg, а второе – в файле style_image.jpg.

```
# загружаем два изображения – контентное и стилевое
with open('content_image.jpg', 'rb') as f:
    content_image = Image.open(BytesIO(f.read()))
with open('style_image.jpg', 'rb') as f:
    style_image = Image.open(BytesIO(f.read()))
```

Рис. 5. – Загрузка контентного и стилевого изображений

Чтобы убедиться, что данные изображения были загружены, отобразим их (рис. 6).

```
# чтобы убедиться, что данные изображения были загружены верно, отобразим их
plt.subplot(1, 2, 1)
plt.imshow(content_image)
plt.subplot(1, 2, 2)
plt.imshow(style_image)
plt.show()
```

Рис. 6. – Отображение контентного и стилевого изображений

Затем, как уже упоминалось ранее, будет использоваться нейронная сеть VGG19, поэтому данные изображения нужно преобразовать во входной формат данной нейронной сети (рис. 7).

```
# затем будет использоваться сеть VGG19,
# поэтому данные изображения нужно преобразовать во
# входной формат данной нейронной сети
x_content = keras.applications.vgg19.preprocess_input(
    np.expand_dims(content_image, axis=0)
)
x_style = keras.applications.vgg19.preprocess_input(
    np.expand_dims(style_image, axis=0)
)
```

Рис. 7. – Преобразование контентного и стилевого изображений во входной формат нейронной сети VGG19

Метод preprocess_input устроен таким образом, что изображение из формата RGB преобразовывается в формат BGR и, кроме того, значение каждого цветового канала уменьшается на следующие величины: синий – на 103,939, зеленый – на 116,779, красный – на 123,68. Поэтому определим функцию, которая преобразовывает изображение из формата BGR в формат RGB (рис. 8).

Затем загрузим обученную нейронную сеть VGG19 (рис. 9).

Параметр weights='imagenet' позволяет загрузить веса модели, уже обученной примерной на 10 миллионах изображениях базы изображений ImageNet. Следующая строка запрещает проводить обучение данной нейронной сети.

```
# определим функцию, которая преобразовывает изображение
# из формата BGR в формат RGB
def deprocess_image(processed_image):
    x = processed_image.copy()
    if len(x.shape) == 4:
        x = np.squeeze(x, 0)
    assert len(x.shape) == 3, ("Input to deprocess image must be an image of"
                               "dimension [1, height, width, channel] or "
                               "[height, width, channel]")
    if len(x.shape) != 3:
        raise ValueError("Invalid input to deprocessing image")
    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
```

Рис. 8. – Функция, которая преобразовывает изображение из формата BGR в формат RGB

```
# загрузим обученную нейронную сеть VGG19
vgg = keras.applications.vgg19.VGG19(include_top=False, weights='imagenet')
vgg.trainable = False
```

Рис. 9. – Загрузка обученной нейронной сети VGG19

Для дальнейшей работы с данной нейронной сетью (подавать на вход изображения, брать на выходах определенных слоев вычисленные карты признаков), создадим новую нейронную сеть на основе VGG19 (рис. 10).

```
# создадим новую нейронную сеть на основе VGG19
inputs = keras.Input(shape=(784,), name='img')
x = layers.Dense(64, activation='relu')(inputs)
x = layers.Dense(64, activation='relu')(x)
outputs = layers.Dense(10, activation='softmax')(x)
model = keras.models.Model(inputs, outputs)
```

Рис. 10. – Создание новой нейронной сети на основе VGG19

Выделим из нейронной сети VGG19 выходы слоев с именами (рис. 11).

Имена данных слоев можно узнать, выполнив команду вывода структуры нейронной сети (рис. 12).

Далее вычислим количество данных слоев (рис. 13).

Выделим выходы слоев из нейронной сети VGG19 (рис. 14).

```
# слой контента, на котором будут отображаться карты признаков
content_layers = ['block5_conv2']

# слои стиля
style_layers = [
    'block1_conv1',
    'block2_conv1',
    'block3_conv1',
    'block4_conv1',
    'block5_conv1'
]
```

Рис. 11. – Имена слоев нейронной сети VGG19

```
print(vgg.summary())  # вывод структуры нейронной сети
```

Рис. 12. – Вывод структуры нейронной сети

```
# вычислим количество данных слоев
num_content_layers = len(content_layers)
num_style_layers = len(style_layers)
```

Рис. 13. – Вычисление количества слоев

```
# выделим выходы слоев из нейронной сети VGG19
style_outputs = [vgg.get_layer(name).output for name in style_layers]
content_outputs = [vgg.get_layer(name).output for name in content_layers]
```

Рис. 14. – Выделение слоев из нейронной сети VGG19 слоев

Затем сформируем общий список выходных слоев и отобразим их (рис. 15).

```
# сформируем общий список выходных слоев
model_outputs = style_outputs + content_outputs

# отобразим их
print(vgg.input)
for m in model_outputs:
    print(m)
```

Рис. 15. – Формирование общего списка выходных слоев и их отображение

Затем сформируем нейронную сеть на основе VGG19 (рис. 16).

```
# сформируем нейронную сеть на основе VGG19
model = keras.models.Model(vgg.input, model_outputs)
for layer in model.layers:
    layer.trainable = False
print(model.summary())  # вывод структуры нейронной сети
```

Рис. 16. – Формирование нейронной сети на основе VGG19

Помимо этого, здесь указывается, что веса изменять нельзя, так как нейронная сеть уже обучена и ее изменять не нужно. Пропуская через данную нейронную сеть какое-нибудь изображение, получим тензор выходных значений.

```
outs = model(x_content)  # получение тензора выходных значений
```

Рис. 17. – Получение тензора выходных значений

Определим функцию для выделения необходимых признаков для стилевого и контентного изображений (рис. 18).

```
# функция для выделения необходимых признаков для
# стилевого и контентного изображений
def get_feature_representations(model):
    style_outputs = model(x_style)
    content_outputs = model(x_content)
    style_features = [
        style_layer[0] for style_layer in style_outputs[:num_style_layers]
    ]
    content_features = [
        content_layer[0] for content_layer in
content_outputs[num_style_layers:]
    ]
    return style_features, content_features
```

Рис. 18. – Функция для выделения необходимых признаков для стилевого и контентного изображений

Реализуем функцию для вычисления степени соответствия стилизованного изображения и исходного по формуле (2) (рис. 19).

```
# функция для вычисления степени соответствия
# стилизованного изображения и исходного
def get_content_loss(base_content, target):
    return tf.reduce_mean(tf.square(base_content - target))
```

Рис. 19. – Функция для вычисления степени соответствия стилизованного изображения и исходного по формуле (2)

Реализуем функцию для вычисления матрицы Грама для переданного ей тензора (рис. 20).

```
# функция для вычисления матрицы Грама для переданного ей тензора
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)
```

Рис. 20. – Функция для вычисления матрицы Грама для переданного ей тензора

Определим функцию для вычисления величины рассогласования стилей между стилевым и стилизованным изображениями по формуле (3) (рис. 21).

```
# функция для вычисления величину рассогласования стилей между
# стилевым и стилизованным изображениями
def get_style_loss(base_style, gram_target):
    gram_style = gram_matrix(base_style)
    return tf.reduce_mean(tf.square(gram_style - gram_target))
```

Рис. 21. – Функция для вычисления величины рассогласования стилей между стилевым и стилизованным изображениями по формуле (3)

Реализуем функцию для определения общего критерия качества стилизованного изображения по формуле (5) (рис. 22:

```
# функция для определения общего критерия качества стилизованного изображения
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
```

Рис. 22. – Функция для определения общего критерия качества стилизованного изображения

Здесь style_weight и content_weight являются параметрами α, β в формуле (5).

Затем определим число итераций и параметры α, β (рис. 23).

```
# определим число итераций и параметры α, β
num_iterations = 100
style_weight = 1e-2
content_weight = 1e3
```

Рис. 23. – Определение числа итераций и параметров α, β

Далее определим карты стилей и контента и вычислим матрицы Грама для стилевого изображения (рис. 24).

```
style_features, content_features = get_feature_representations(model)
gram_style_features = [
    gram_matrix(style_feature) for style_feature in style_features
]
```

Рис. 24. – Определение карты стилей и контента и вычисление матрицы Грама для стилевого изображения

Формируем начальное изображение как копию контентного (рис. 25).

```
# формируем начальное изображение как копию контентного
init_image = np.copy(x_content)
init_image = tf.Variable(init_image, dtype=tf.float32)
```

Рис. 25. – Формирование начального изображения как копии контентного

Далее укажем вспомогательные переменные (рис. 26). Укажем в качестве оптимизатора алгоритма градиентного спуска Adam, номер текущей итерации, переменные для хранения потерь и лучшего стилизованного изображения, кортеж параметров α, β, сформируем словарь конфигурации, а также укажем вспомогательные переменные, которые нужны для преобразования формируемых изображений в формат RGB, а также список для хранения изображений, которые формируются на каждой итерации алгоритма.

Наконец запускаем алгоритм градиентного спуска, который сформирует стилизованное изображение (рис. 27).

```
# укажем вспомогательные переменные
opt = tf.compat.v1.train.AdamOptimizer(
    learning_rate=2, beta1=0.99, epsilon=1e-1
)
iter_count = 1
best_loss, best_img = float('inf'), None
loss_weights = (style_weight, content_weight)
cfg = {
    'model': model,
    'loss_weights': loss_weights,
    'init_image': init_image,
    'gram_style_features': gram_style_features,
    'content_features': content_features
}
norm_means = np.array([103.939, 116.779, 123.68])
min_vals = -norm_means
max_vals = 255 - norm_means
imgs = []
```

Рис. 26. – Указание вспомогательных переменных

```
# алгоритм градиентного спуска, который формирует стилизованное изображение
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_image(init_image.numpy())
        plot_img = deprocess_image(init_image.numpy())
        imgs.append(plot_img)
        print('Iteration: {}'.format(i))
```

Рис. 27. – Алгоритм градиентного спуска, который формирует стилизованное изображение

На рис. 28 приведен пример входных данных – контентного и стилевого изображений.

Для указанных входных данных на рис. 29 приведены выходные данные – стилизованное изображение, которое объединено из двух – контентного и стилевого.

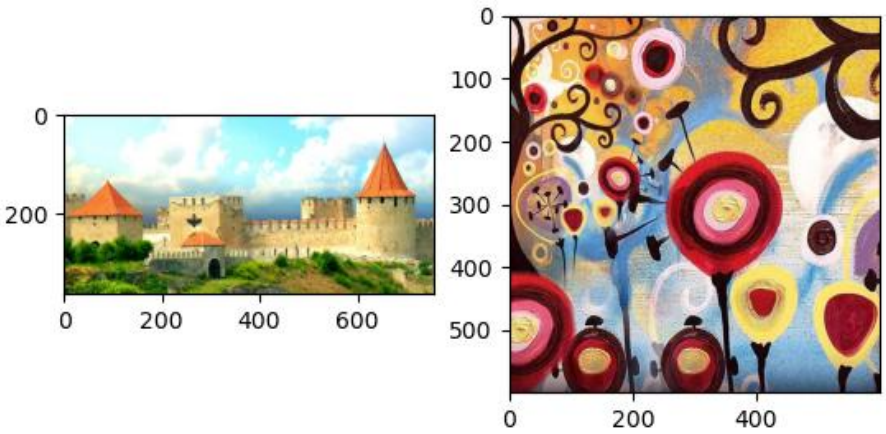

Рис. 28. – Пример контентного и стилевого изображений

Рис. 29. – Стилизованное изображение

Таким образом, в ходе работы реализован алгоритм переноса стилей изображений с использованием Keras и TensorFlow.

Литература

1. Аггарвал Чару. Нейронные сети и глубокое обучение. СПб.: ООО «Диалектика», 2020. 752 с.

2. Гафаров Ф. М., Галимянов Ф. М. Искусственные нейронные сети и приложения. Казань: Изд-во Казан. ун-та, 2018. 121 с.

3. Николенко С. И., Кадурин А. А., Архангельская Е. О. Глубокое обучение. СПб.: Питер, 2018. 480 с.

4. Мирошниченко А. Н., Шиян В. И. Автоматическая колоризация изображений на основе модели искусственной нейронной сети, состоящей из классификатора и колоризатора // Прикладная математика: современные проблемы математики, информатики и моделирования: Материалы IV Всероссийской научно-практической конференции молодых ученых, Краснодар, 18-23 апреля 2022 года. Краснодар: ФГБУ «Российское энергетическое агентство» Минэнерго России Краснодарский ЦНТИ – филиал ФГБУ «РЭА» Минэнерго России, 2022. С. 157-162.

5. Белоусов И. С., Рогачев А. Ф. Разработка глубокой нейронной сети для сегментации проблемных участков сельскохозяйственных полей // Инженерный вестник Дона, 2022. № 8. URL: ivdon.ru/ru/magazine/archive/n8y2022/7864

6. Шиян В. И. Распределение товаров на категории с помощью нейронной сети BERT // Государство. Бизнес. Общество. Цифровая среда: траектория взаимодействия от теории к практике: сборник научных статей по итогам международной научно-практической конференции., Санкт-Петербург, 29-30 апреля 2021 года. Санкт-Петербург: Санкт-Петербургский государственный экономический университет, 2021. С. 38-42.

7. Жданов Н. С., Лапина О. Н. Применение нейронной сети Хопфилда для решения задачи коммивояжера // Прикладная математика XXI века: современные проблемы математики, информатики и моделирования: Материалы всероссийской научно-практической конференции, Краснодар, 24-26 апреля 2019 года. Краснодар: ФГБУ «Российское энергетическое

агентство» Минэнерго России Краснодарский ЦНТИ – филиал ФГБУ «РЭА» Минэнерго России, 2019. С. 201-206.

8. Корнаухова М. А., Основин С. С., Баскаков Е. В., Савин Е. С., Полусмакова Н. С. Информационная система помощи выбора специальности для абитуриентов на основе нейронной сети // Инженерный вестник Дона, 2022. № 11. URL: ivdon.ru/ru/magazine/archive/n11y2022/7965

9. Gatys L. A., Ecker A. S., Bethge M. A Neural Algorithm of Artistic Style (2015) // URL: arxiv.org/abs/1508.06576 (дата обращения: 16.11.2022).

10. Gatys L. A., Ecker A. S., Bethge M. Texture and art with deep neural networks (2017) // URL: doi.org/10.1016/j.conb.2017.08.019 (дата обращения: 16.11.2022).

References

1. Aggarval Charu. Nejronnye seti i glubokoe obuchenie [Neural networks and deep learning]. SPb.: OOO «Dialektika», 2020. 752 p.

2. Gafarov F. M., Galimyanov F. M. Iskusstvennye nejronnye seti i prilozheniya [Artificial neural networks and applications]. Kazan: Izd-vo Kazan. un-ta, 2018. 121 p.

3. Nikolenko S. I., Kadurin A. A., Arxangelskaya E. O. Glubokoe obuchenie [Deep learning]. SPb.: Piter, 2018. 480 p.

4. Miroshnichenko A. N., Shiyan V. I. Prikladnaya matematika: sovremennye problemy matematiki, informatiki i modelirovaniya: Materialy IV Vserossijskoj nauchno-prakticheskoj konferencii molodyx uchenyx, Krasnodar, 18-23 aprelya 2022 goda. Krasnodar: FGBU «Rossijskoe energeticheskoe agentstvo» Minenergo Rossii Krasnodarskij CzNTI – filial FGBU «REA» Minenergo Rossii, 2022. pp. 157-162.

5. Belousov I. S., Rogachev A. F. Inzhenernyj vestnik Dona, 2022. № 8. URL: ivdon.ru/ru/magazine/archive/n8y2022/7864

6. Shiyan V. I. Gosudarstvo. Biznes. Obshhestvo. Cifrovaya sreda: traektoriya vzaimodejstviya ot teorii k praktike: sbornik nauchnyx statej po itogam mezhdunarodnoj nauchno-prakticheskoj Konferencii, Sankt Peterburg, 29-30 aprelya 2021 goda. Sankt-Peterburg: Sankt Peterburgskij gosudarstvennyj ekonomicheskij universitet, 2021. pp. 38-42.

7. Zhdanov N. S., Lapina O. N. Prikladnaya matematika XXI veka: sovremennye problemy matematiki, informatiki i modelirovaniya: Materialy vserossijskoj nauchno-prakticheskoj konferencii, Krasnodar, 24 26 aprelya 2019 goda. Krasnodar: FGBU «Rossijskoe energeticheskoe agentstvo» Minenergo Rossii Krasnodarskij CzNTI – filial FGBU «REA» Minenergo Rossii, 2019. pp. 201-206.

8. Kornauxova M. A., Osnovin S. S., Baskakov E. V., Savin E. S., Polusmakova N. S. Inzhenernyj vestnik Dona, 2022. № 11. URL: ivdon.ru/ru/magazine/archive/n11y2022/7965

9. Gatys L. A., Ecker A. S., Bethge M. A Neural Algorithm of Artistic Style (2015). URL: arxiv.org/abs/1508.06576 (accessed: 11/16/2022).

10. Gatys L. A., Ecker A. S., Bethge M. Texture and art with deep neural networks (2017). URL: doi.org/10.1016/j.conb.2017.08.019 (accessed: 11/16/2022).