МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ «МИСиС»**

Институт ИТКН

Кафедра инженерной кибернетики

Направление подготовки: «01.03.04 Прикладная математика»

Квалификация: бакалавр

Группа: БПМ-17-1

**ОТЧЕТ**

**ПО КУРСОВОЙ РАБОТЕ**

**«ИСКУССТВЕННЫЕ НЕЙРОННЫЕ СЕТИ»**

на тему: «Стилизация изображений с помощью сверточных сетей»

**Студенты** \_\_\_\_\_\_\_\_\_\_/ Артемьева П.И.

Палатова Т.А.

**Руководитель** \_\_\_\_\_\_\_\_\_\_/ старший преподаватель, Кондыбаева А.Б.

**Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Дата защиты: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Москва 2020**

**Содержание**

1. Введение 3

2. Теоретическая часть 5

2.1. Нейронные сети 5

2.2. Сверточные нейронные сети 6

2.3. Модель SqueezeNet 7

3. Реализация модели 12

3.1. Используемые программные средства 12

3.2. Особенности разработанного ПО 12

3.3. Тестирование модели 14

4. Заключение 18

5. Список использованных источников 19

Приложение. 20

1. **Введение**

Глубокие нейронные сети в настоящее время становятся одним из самых популярных методов машинного обучения. Они показывают лучшие результаты по сравнению с альтернативными методами в таких областях, как распознавание речи, обработка естественного языка, компьютерное зрение, медицинская информатика и др. Одна из причин успешного применения глубоких нейронных сетей заключается в том, что сеть автоматически выделяет из данных важные признаки, необходимые для решения задачи. В альтернативных алгоритмах машинного обучения признаки должны выделяться людьми, существует специализированное направление исследований — инженерия признаков (feature engineering). Однако при обработке больших объемов данных нейронная сеть справляется с выделением признаков гораздо лучше, чем человек.

Модель искусственных нейронных сетей была предложена в 1943 году, а сам термин глубокое обучение (deep learning) стал широко использоваться только начиная с 2006 года. До этого применялись термины загрузка глубоких сетей (loading deep networks) и обучение глубокой памяти (learning deep memories). Рост популярности глубоких нейронных сетей, происходящий в последние несколько лет, можно объяснить тремя факторами. Во-первых, произошло существенное увеличение производительности компьютеров, в том числе ускорителей вычислений GPU (Graphics Processing Unit), что позволило обучать глубокие нейронные сети сети значительно быстрее и с более высокой точностью. Ранее имеющихся вычислительных мощностей не хватало для обучения сколько-нибудь сложной сети, пригодной для решения практических задач. Во-вторых, был накоплен большой объем данных, который необходим для обучения глубоких нейронных сетей. В-третьих, разработаны методы обучения нейронных сетей, позволяющие быстро и качественно обучать сети, состоящие из ста и более слоев, что раньше было невозможно из-за проблемы исчезающего градиента и переобучения. Сочетание трех факторов привело к существенному прогрессу в обучении глубоких нейронных сетей и их практическом использовании, что позволило глубоким нейронным сетям занять лидирующую позицию среди методов машинного обучения.

Целью данной курсовой работы является изучение работы сверточной сети SqueezeNet, а также разработка алгоритма стилизации изображений.

**2. Теоретическая часть**

**2.1. Нейронные сети**

Нейронная сеть – это сеть или схема нейронов, или в современном смысле искусственная нейронная сеть, состоящая из искусственных нейронов или узлов. Таким образом, нейронная сеть является либо биологической нейронной сетью, состоящей из реальных биологических нейронов, либо искусственной нейронной сетью, для решения задач искусственного интеллекта (ИИ). Нейронные сети представляют собой набор алгоритмов, свободно смоделированных на подобии человеческого мозга, которые предназначены для распознавания закономерностей. Они интерпретируют сенсорные данные через своего рода машинное восприятие, маркировку или кластеризацию исходных данных. Распознаваемые ими закономерности являются числовыми, содержатся в векторах, в которые должны быть переведены все реальные данные, будь то изображения, звук, текст или временные ряды. Нейронные сети помогают нам группировать и классифицировать. О них можно думать, как о кластеризации и классификации поверх данных, которые хранятся и которыми управляют. Они помогают группировать немаркированные данные в соответствии с сходством между входами примеров, и они классифицируют данные, когда у них есть маркированный набор данных для обучения. Соединения биологического нейрона моделируются в виде весов. Положительный вес отражает возбуждающую связь, отрицательные значения означают ингибирующие (тормозные) связи. Все входные данные модифицируются по весу и суммируются. Эта активность называется линейной комбинацией. Данная комбинация проходит через функцию активации, которая контролирует амплитуду выхода.

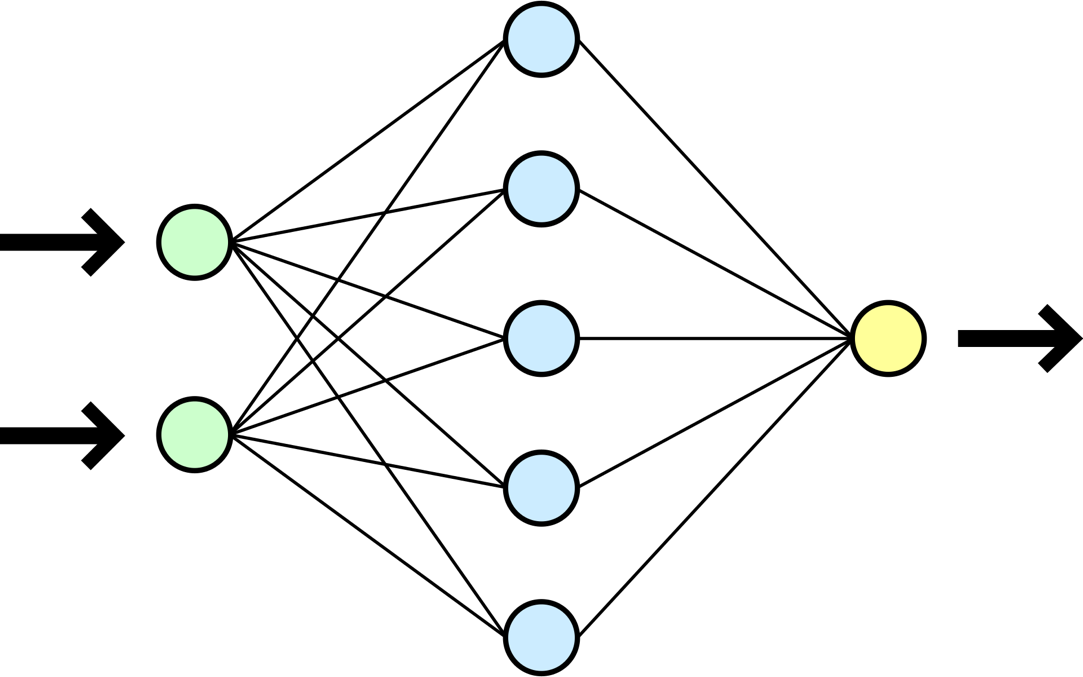


Рисунок 1 – Классический вид нейронной сети

2.2. Сверточные нейронный сети

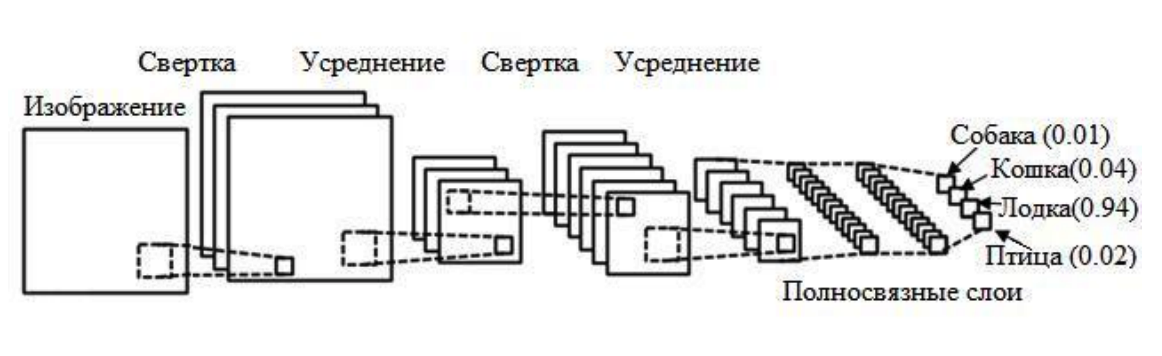
Основной смысл в сверточных нейронных сетях - сверточные слои изучают локальные шаблоны в случае с изображениями — шаблоны в небольших двумерных окнах во входных данных. Поэтому можно сказать, что сверточная нейронная сеть делает сильные и в основном правильные предположения о природе изображений, а именно о локальности пиксельных зависимостей и стационарности статистики. Следовательно, по сравнению со стандартными нейронными сетями с прямой связью, сверточная нейронная сеть имеет гораздо меньше соединений и параметров, что облегчает обучение. Сверточная нейронная сеть состоит из сверточных слоев, за которыми следуют полностью связанные слои (рис. 2). Сверточный слой состоит из фильтра свертки, за которым следуют фильтр объединения и функция активации. 

Рисунок 2 – Типичная архитектура для СНС

Сверточный фильтр имеет несколько фильтров с тем же размером окна, которые растекаются по изображению с заданным шагом. Объединение суммирует выходные данные соседних групп нейронов в одну ячейку – ядро. 23 Фильтр объединения имеет окно с определенным размером, которое перемещается по изображению с заданным шагом. Обычно, функцией объединения является функция максимума, которая выводит максимальное значение в ячейку, т.е. в ядро и используется в нашей модели.

Рассмотрим представления изображений в сверточной нейронной сети (CNN). Заданное входное изображение представлено как набор отфильтрованных изображений на каждом этапе обработки в CNN. В то время как количество различных фильтров увеличивается по иерархии обработки, размер отфильтрованных изображений уменьшается с помощью некоторого механизма понижающей дискретизации (например, максимального объединения), что приводит к уменьшению общего количества слоев сети. Мы можем визуализировать информацию на разных этапах обработки в CNN, реконструируя входные данные, изображение, только зная ответы сети на определенном уровне (рисунок 3).

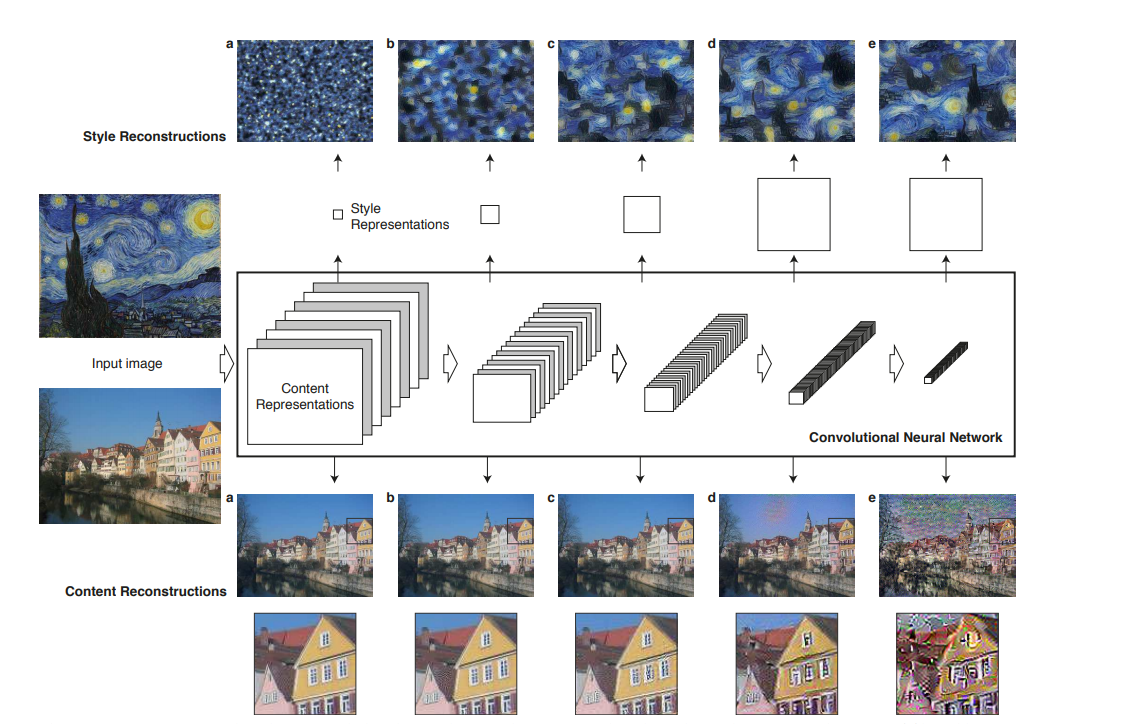


Рисунок 3 – Content Reconstructions

Рассмотрим алгоритм переноса стиля. Первые элементы контента и стиля извлекаются и сохраняются. Образ стиля передается по сети и его стилевое представление на всех включенных слоях вычисляются и сохраняются. Изображение содержимого передается по сети и представление содержания в один слой хранится (справа). Затем случайное изображение белого шума передается по сети и его особенности стиля, и особенности содержания вычислены. На каждом слое, включенном в представление стиля, поэлементная среднеквадратичная разница между Gl и Al вычисляется, чтобы показать потерю стиля Lstyle (слева). Также среднеквадратичная разница между Fl и Pl вычисляется, чтобы увидеть потерю контента Lcontent (справа). В этом случае общие потери Ltotal представляют собой линейную комбинацию между потерями содержимого и стиля. Его производная по значениям пикселей может быть вычислена с использованием метода обратного распространения ошибки (в центре). Этот градиент используется для итеративного обновления изображения, пока оно не будет одновременно соответствовать характеристикам стиля изображения стиля и характеристикам содержимого изображения (посередине, внизу) (рисунок 4).

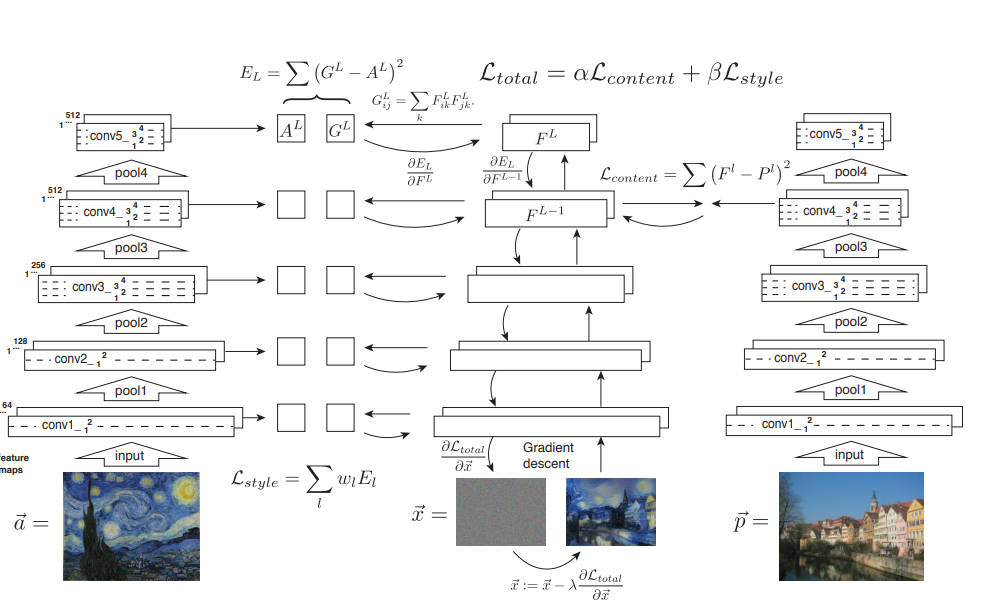


Рисунок 4 – Алгоритм передачи стиля

**2.3. Модель SqueezeNet**

SqueezeNet – [нейронная сеть](https://en.wikipedia.org/wiki/Deep_neural_network) для [компьютерного зрения](https://en.wikipedia.org/wiki/Computer_vision), которая была выпущена в 2016 году. SqueezeNet была разработана исследователями из [DeepScale](https://en.wikipedia.org/wiki/DeepScale" \o "Глубокая шкала), [Калифорнийского университета, Беркли](https://en.wikipedia.org/wiki/University_of_California,_Berkeley)и [Стэнфордского университета](https://en.wikipedia.org/wiki/Stanford_University). При проектировании SqueezeNet целью авторов было создать нейронную сеть с меньшим количеством параметров, которая могла бы легче вписываться в компьютерную память и легче передаваться по компьютерной сети.

**3. Реализация модели**

**3.1. Используемые программные средства**

Реализация модели в рамках данной работы производилась в облачном сервисе Google Colaboratory, на языке Python v3.8. Список используемых библиотек:

* Tensorflow – библиотека для создания нейронных сетей;
* PIL – библиотека, предназначенная для работы с растровой графикой;
* Matplotlib – построение различных графиков и диаграмм;
* NumPy – библиотека, необходимая для работы с векторами и массивами.

**3.2. Особенности разработанного ПО**

Для достижения заданной цели работы была реализована идея, заключающаяся в том, чтобы взять два изображения, и получить новое изображение, с содержанием первого и (артистическим) стилем второго. Метод подразумевает из себя следующее: формулируется функция потерь, которая соответствует содержимому и стилю каждого соответствующего изображения, а затем выполняется градиентный спуск на пикселях самого изображения.

Cетью, используемой в данной работе в качестве экстрактора признаков, является [SqueezeNet](https://arxiv.org/abs/1602.07360" \t "_blank) - небольшая модель, обученная на ImageNet. Мы выбрали SqueezeNet из-за ее небольшого размера и эффективности.

В ходе разработке были произведены:

* импорт библиотек и объявление констант;
* реализованы вспомогательные функции для обработки изображений, представленные в приложении;
* реализована архитектура SqueezeNet;
* вычисление потерь.

Вычисление потерь.

Для того чтобы сгенерировать изображение, которое отражает содержание одного изображения и стиль другого, нужно пенализировать отклонения от содержания изображения-контента и отклонения от стиля изображения-стиля. Затем можно использовать эту гибридную функцию потерь для выполнения градиентного спуска не по параметрам модели, а по значениям пикселей исходного изображения.

Теперь вычислим три компонента нашей функции потерь. Функция потерь представляет собой взвешенную сумму трех членов: потеря контента + потеря стиля + потеря общего отклонения.

Потеря контента(содержимого) определяется по формуле, показанной на рисунке 5.

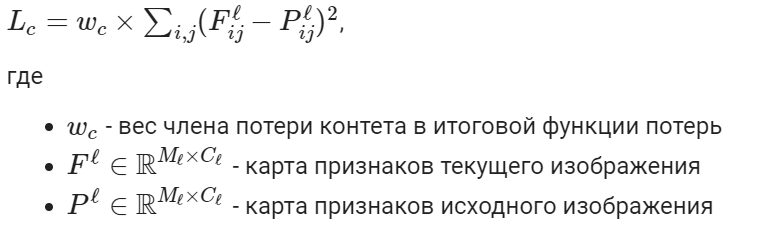


Рисунок 5 – Потеря контента

Далее рассмотрим формулы, с помощью которых вычисляется потеря стиля (рисунок 6).

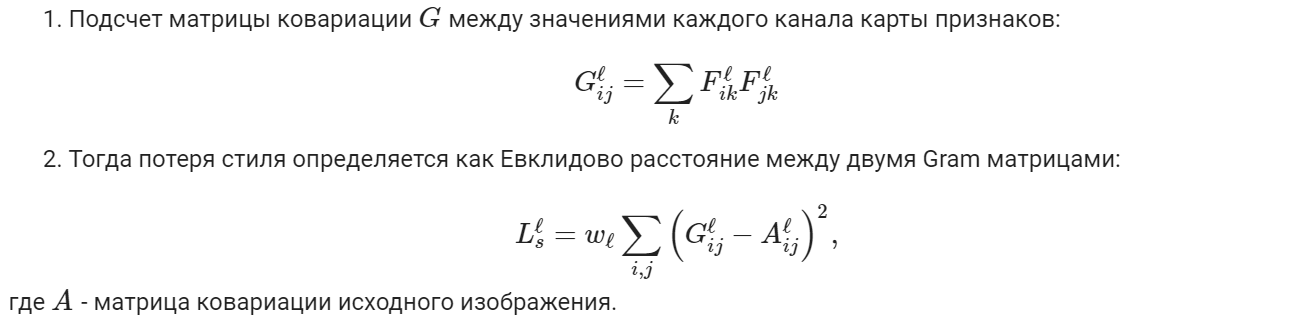


Рисунок 6 – Потеря стиля

Потеря общего отклонения. Вычисления представлены на рисунке 7.

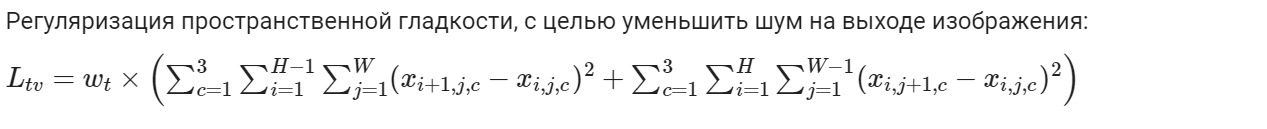


Рисунок 7 – Потеря общего отклонения

Перенос стиля. Функция, представленная ниже, объединяет все функции потерь, которые были описаны ранее, и оптимизирует изображение, которое минимизирует общие потери.

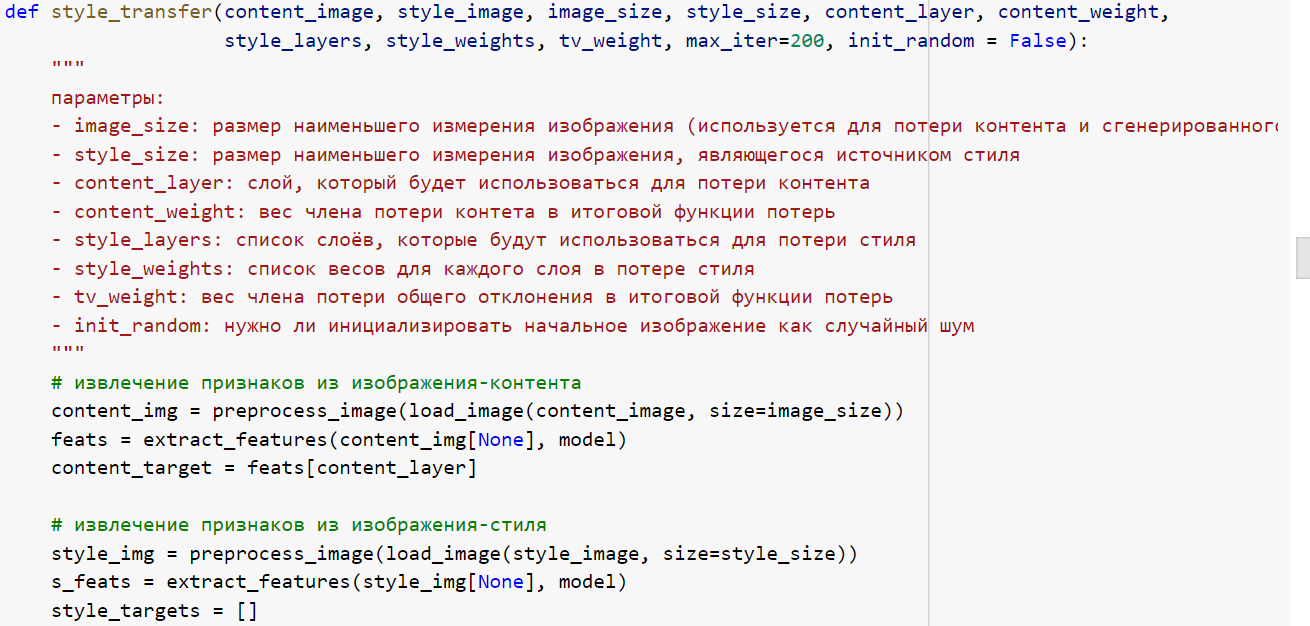


Рисунок 8 – Функция, объединяющая все функции потерь

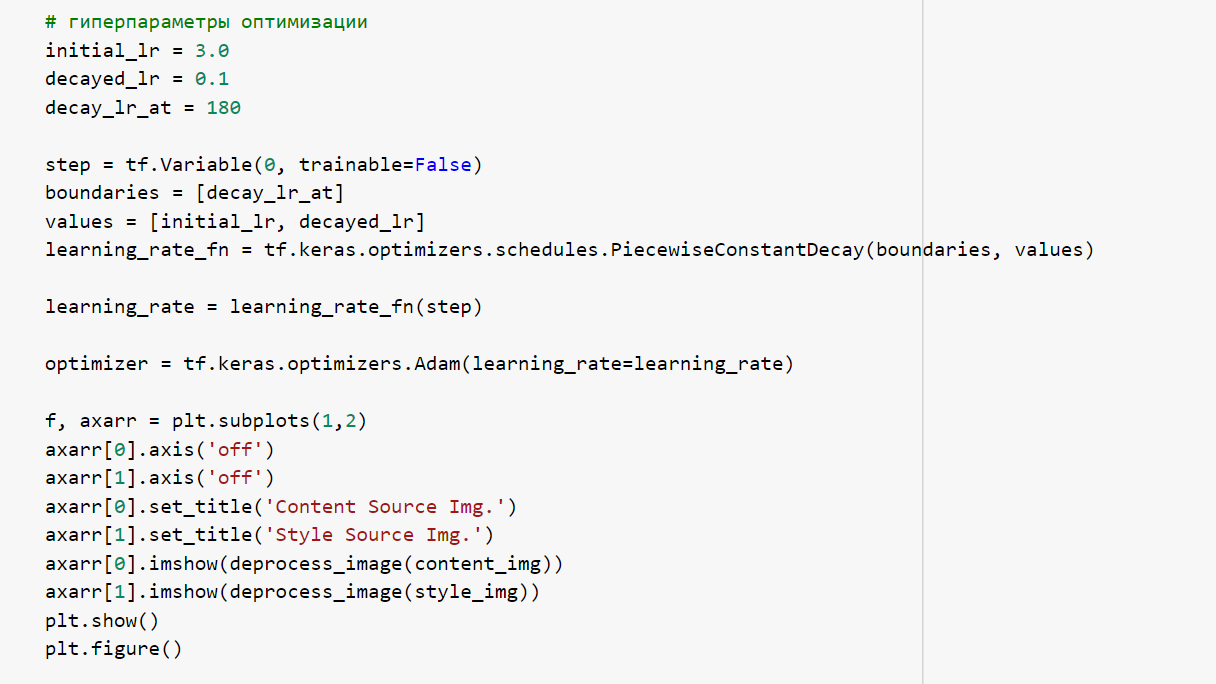


Рисунок 9 – Функция, объединяющая все функции потерь

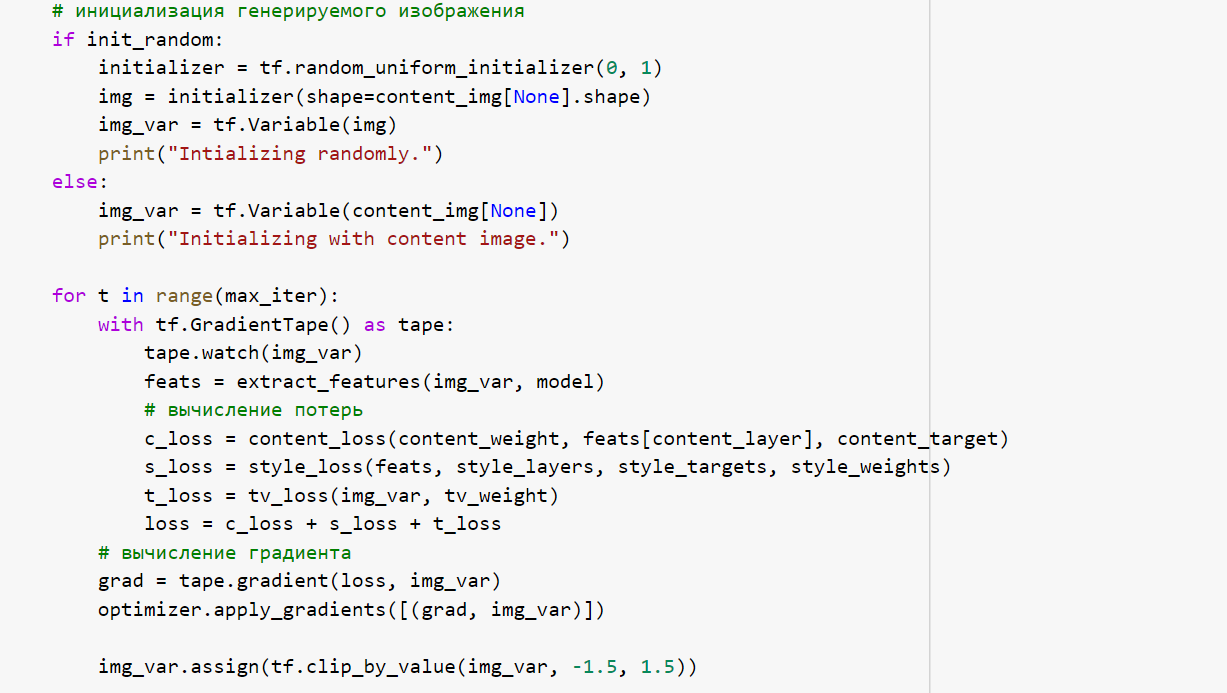


Рисунок 10 – Функция, объединяющая все функции потерь

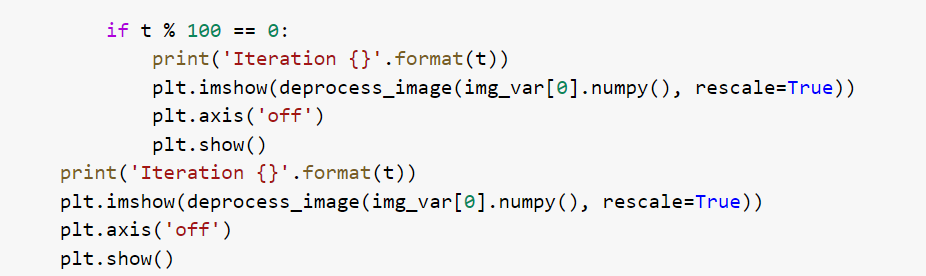


Рисунок 11 – Функция, объединяющая все функции потерь

**3.3. Тестирование модели**

В результате тестирования стилизации изображений были получены следующие результаты.

1. Композиция VII Кандинского + фотография Тюбингена. Представлены на рисунке 12.



Рисунок 12 – Изображения, использованные для тестирования

Полученные результаты на 0, 100 и 199 итерациях представлены на рисунках 13, 14 и 15.

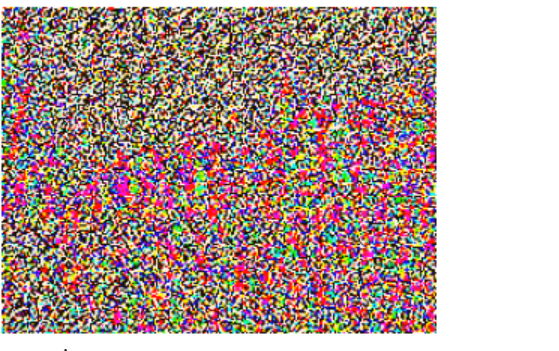


Рисунок 13 – Результаты тестирования: итерация 0



Рисунок 14 – Результаты тестирования: итерация 100



Рисунок 15 – Результаты тестирования: итерация 199

1. Композиция Муза Пикассо + фотография Тюбингена. Представлены на рисунке 16.



Рисунок 16 – Изображения, использованные для тестирования

Полученные результаты на 0, 100 и 199 итерациях представлены на рисунках 17, 18 и 19.



Рисунок 17 – Результаты тестирования: итерация 0



Рисунок 18 – Результаты тестирования: итерация 100



Рисунок 19 – Результаты тестирования: итерация 199

**4. Заключение**

Подводя итоги, можно сделать вывод, что на сегодняшний момент существуют разные архитектуры сверточных нейронных сетей, способные решать задачи обнаружения объектов в реальном времени. Эта возможность появилась благодаря развитию машинного обучения, в частности очень глубоких сетей, и аппаратной части для обучения таких ресурсозатратных нейронных сетей.

В данной курсовой работе была изучена и создана модель нейронной сети SqueezeNet с возможностью стилизовать изображения.

**5. Список использованных источников**

* N. Ashikhmin. Fast texture transfer. IEEE Computer Graphics and Applications, 23(4):38–43, July 2003.
* M. Berning, K. M. Boergens, and M. Helmstaedter. SegEM: Efficient Image Analysis for High-Resolution Connectomics. Neuron, 87(6):1193–1206, Sept. 2015.
* C. F. Cadieu, H. Hong, D. L. K. Yamins, N. Pinto, D. Ardila, E. A. Solomon, N. J. Majaj, and J. J. DiCarlo. Deep Neural Networks Rival the Representation of Primate IT Cor2421 tex for Core Visual Object Recognition. PLoS Comput Biol, 10(12):e1003963, Dec. 2014.
* L. A. Gatys, A. S. Ecker, and M. Bethge .Image Style Transfer Using Convolutional Neural Networks. In Proc. CVPR, 2016.

**Приложение.**

from google.colab import drive

drive.mount('/content/drive')

import os

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

from imageio import imread

from PIL import Image

%matplotlib inline

tf.get\_logger().setLevel('ERROR')

SQUEEZENET\_MEAN = np.array([0.485, 0.456, 0.406], dtype=np.float32)

SQUEEZENET\_STD = np.array([0.229, 0.224, 0.225], dtype=np.float32)

NUM\_CLASSES = 1000

# вспомогательные функции для обработки изображений

def preprocess\_image(img):

    return (img.astype(np.float32) / 255.0 - SQUEEZENET\_MEAN) / SQUEEZENET\_STD

def deprocess\_image(img, rescale=False):

    img = img \* SQUEEZENET\_STD + SQUEEZENET\_MEAN

    if rescale:

        vmin, vmax = img.min(), img.max()

        img = (img - vmin) / (vmax - vmin)

    return np.clip(255 \* img, 0.0, 255.0).astype(np.uint8)

def load\_image(filename, size=None):

    img = imread(filename)

    if size is not None:

        orig\_shape = np.array(img.shape[:2])

        min\_idx = np.argmin(orig\_shape)

        scale\_factor = float(size) / orig\_shape[min\_idx]

        new\_shape = (orig\_shape \* scale\_factor).astype(int)

        img = np.array(Image.fromarray(img).resize((new\_shape[1],new\_shape[0])))

    return img

# реализация архитектуры SqeezeNet

class Fire(tf.keras.Model):

    def \_\_init\_\_(self, inplanes, squeeze\_planes, expand1x1\_planes, expand3x3\_planes,name=None):

        super(Fire, self).\_\_init\_\_(name='%s/fire'%name)

        self.inplanes = inplanes

        self.squeeze = tf.keras.layers.Conv2D(squeeze\_planes, input\_shape=(inplanes,), kernel\_size=1, strides=(1,1), padding="VALID", activation='relu',name='squeeze')

        self.expand1x1 = tf.keras.layers.Conv2D(expand1x1\_planes, kernel\_size=1, padding="VALID", strides=(1,1), activation='relu',name='e11')

        self.expand3x3 = tf.keras.layers.Conv2D(expand3x3\_planes, kernel\_size=3, padding="SAME", strides=(1,1), activation='relu',name='e33')

    def call(self, x):

        x = self.squeeze(x)

        return tf.concat([

            self.expand1x1(x),

            self.expand3x3(x)

        ], axis=3)

class SqueezeNet(tf.keras.Model):

    def \_\_init\_\_(self, num\_classes=NUM\_CLASSES):

        super(SqueezeNet, self).\_\_init\_\_()

        self.num\_classes = num\_classes

        self.net = tf.keras.models.Sequential([

            tf.keras.layers.Conv2D(64, kernel\_size=(3, 3), strides=(2,2), padding="VALID", activation='relu', input\_shape=(224, 224, 3), name='features/layer0'),

            tf.keras.layers.MaxPool2D(pool\_size=3, strides=2, name='features/layer2'),

            Fire(64, 16, 64, 64, name='features/layer3'),

            Fire(128, 16, 64, 64, name='features/layer4'),

            tf.keras.layers.MaxPool2D(pool\_size=3, strides=2, name='features/layer5'),

            Fire(128, 32, 128, 128, name='features/layer6'),

            Fire(256, 32, 128, 128, name='features/layer7'),

            tf.keras.layers.MaxPool2D(pool\_size=3, strides=2, name='features/layer8'),

            Fire(256, 48, 192, 192, name='features/layer9'),

            Fire(384, 48, 192, 192, name='features/layer10'),

            Fire(384, 64, 256, 256, name='features/layer11'),

            Fire(512, 64, 256, 256, name='features/layer12'),

            tf.keras.layers.Conv2D(self.num\_classes, kernel\_size=1, padding="VALID",  activation='relu', name='classifier/layer1'),

            tf.keras.layers.AveragePooling2D(pool\_size=13, strides=13, padding="VALID", name='classifier/layer3')

            ])

    def call(self, x, save\_path=None):

        x = self.net(x)

        scores = tf.reshape(x, (-1, self.num\_classes))

        return scores

# подходит для Unix систем (в том числе Google Colab)

!wget "http://cs231n.stanford.edu/squeezenet\_tf2.zip"

!unzip squeezenet\_tf2.zip

!rm squeezenet\_tf2.zip

model=SqueezeNet()

model.load\_weights('squeezenet.ckpt')

model.trainable=False

def content\_loss(content\_weight, content\_current, content\_original):

    return content\_weight \* tf.reduce\_sum(tf.math.squared\_difference(content\_current, content\_original))

# функиця, извлекающая признаки для каждого слоя изображения

def extract\_features(x, cnn):

    features = []

    prev\_feat = x

    for i, layer in enumerate(cnn.net.layers[:-2]):

        next\_feat = layer(prev\_feat)

        features.append(next\_feat)

        prev\_feat = next\_feat

    return features

# вычисление Gram матрицы из признаков

def gram\_matrix(features, normalize=True):

    \_, H, W, C = tf.shape(features)

    features = tf.reshape(features, [H \* W, C])

    gram = tf.matmul(tf.transpose(features), features)

    if normalize:

        gram /= tf.cast(H \* W \* C, dtype=tf.float32)

    return gram

# вычисление потерь стиля для набора слоев

def style\_loss(feats, style\_layers, style\_targets, style\_weights):

    style\_loss = 0

    for i in range(len(style\_layers)):

        G = gram\_matrix(feats[style\_layers[i]])

        style\_loss += style\_weights[i] \* tf.reduce\_sum(tf.math.squared\_difference(G, style\_targets[i]))

    return style\_loss

def tv\_loss(img, tv\_weight):

    \_, H, W, C = tf.shape(img)

    diff\_vertical = img[:,:H-1,:,:] - img[:,1:H,:,:]

    diff\_horizontal = img[:,:,:W-1,:] - img[:,:,1:W,:]

    return tv\_weight \* (tf.reduce\_sum(diff\_vertical\*\*2) + tf.reduce\_sum(diff\_horizontal\*\*2))

def style\_transfer(content\_image, style\_image, image\_size, style\_size, content\_layer, content\_weight,

                   style\_layers, style\_weights, tv\_weight, max\_iter=200, init\_random = False):

    """

    параметры:

    - image\_size: размер наименьшего измерения изображения (используется для потери контента и сгенерированного изображения)

    - style\_size: размер наименьшего измерения изображения, являющегося источником стиля

    - content\_layer: слой, который будет использоваться для потери контента

    - content\_weight: вес члена потери контета в итоговой функции потерь

    - style\_layers: список слоёв, которые будут использоваться для потери стиля

    - style\_weights: список весов для каждого слоя в потере стиля

    - tv\_weight: вес члена потери общего отклонения в итоговой функции потерь

    - init\_random: нужно ли инициализировать начальное изображение как случайный шум

    """

    # извлечение признаков из изображения-контента

    content\_img = preprocess\_image(load\_image(content\_image, size=image\_size))

    feats = extract\_features(content\_img[None], model)

    content\_target = feats[content\_layer]

    # извлечение признаков из изображения-стиля

    style\_img = preprocess\_image(load\_image(style\_image, size=style\_size))

    s\_feats = extract\_features(style\_img[None], model)

    style\_targets = []

    # список Gram матриц

    for idx in style\_layers:

        style\_targets.append(gram\_matrix(s\_feats[idx]))

    # гиперпараметры оптимизации

    initial\_lr = 3.0

    decayed\_lr = 0.1

    decay\_lr\_at = 180

    step = tf.Variable(0, trainable=False)

    boundaries = [decay\_lr\_at]

    values = [initial\_lr, decayed\_lr]

    learning\_rate\_fn = tf.keras.optimizers.schedules.PiecewiseConstantDecay(boundaries, values)

    learning\_rate = learning\_rate\_fn(step)

    optimizer = tf.keras.optimizers.Adam(learning\_rate=learning\_rate)

    f, axarr = plt.subplots(1,2)

    axarr[0].axis('off')

    axarr[1].axis('off')

    axarr[0].set\_title('Content Source Img.')

    axarr[1].set\_title('Style Source Img.')

    axarr[0].imshow(deprocess\_image(content\_img))

    axarr[1].imshow(deprocess\_image(style\_img))

    plt.show()

plt.figure()

    # инициализация генерируемого изображения

    if init\_random:

        initializer = tf.random\_uniform\_initializer(0, 1)

        img = initializer(shape=content\_img[None].shape)

        img\_var = tf.Variable(img)

        print("Intializing randomly.")

    else:

        img\_var = tf.Variable(content\_img[None])

        print("Initializing with content image.")

    for t in range(max\_iter):

        with tf.GradientTape() as tape:

            tape.watch(img\_var)

            feats = extract\_features(img\_var, model)

            # вычисление потерь

            c\_loss = content\_loss(content\_weight, feats[content\_layer], content\_target)

            s\_loss = style\_loss(feats, style\_layers, style\_targets, style\_weights)

            t\_loss = tv\_loss(img\_var, tv\_weight)

            loss = c\_loss + s\_loss + t\_loss

        # вычисление градиента

        grad = tape.gradient(loss, img\_var)

        optimizer.apply\_gradients([(grad, img\_var)])

        img\_var.assign(tf.clip\_by\_value(img\_var, -1.5, 1.5))

        if t % 100 == 0:

            print('Iteration {}'.format(t))

            plt.imshow(deprocess\_image(img\_var[0].numpy(), rescale=True))

            plt.axis('off')

            plt.show()

    print('Iteration {}'.format(t))

    plt.imshow(deprocess\_image(img\_var[0].numpy(), rescale=True))

    plt.axis('off')

    plt.show()

FOLDERNAME = '/content/drive/My Drive/Курсовая\_Артемьева\_БПМ-17-1'

# Composition VII + Tubingen

params1 = {

    'content\_image' : '%s/styles/tubingen.jpg' % (FOLDERNAME),

    'style\_image' : '%s/styles/composition\_vii.jpg' % (FOLDERNAME),

    'image\_size' : 192,

    'style\_size' : 512,

    'content\_layer' : 2,

    'content\_weight' : 5e-2,

    'style\_layers' : (0, 3, 5, 6),

    'style\_weights' : (20000, 500, 12, 1),

    'tv\_weight' : 5e-2

}

style\_transfer(\*\*params1)

# Scream + Tubingen

params2 = {

    'content\_image':'%s/styles/tubingen.jpg' % (FOLDERNAME),

    'style\_image':'%s/styles/the\_scream.jpg' % (FOLDERNAME),

    'image\_size':192,

    'style\_size':224,

    'content\_layer':2,

    'content\_weight':3e-2,

    'style\_layers':[0, 3, 5, 6],

    'style\_weights':[200000, 800, 12, 1],

    'tv\_weight':2e-2

}

style\_transfer(\*\*params2)

# Starry Night + Tubingen

params3 = {

    'content\_image' : '%s/styles/tubingen.jpg' % (FOLDERNAME),

    'style\_image' : '%s/styles/starry\_night.jpg' % (FOLDERNAME),

    'image\_size' : 192,

    'style\_size' : 192,

    'content\_layer' : 2,

    'content\_weight' : 6e-2,

    'style\_layers' : [0, 3, 5, 6],

    'style\_weights' : [300000, 1200, 10, 2],

    'tv\_weight' : 2e-2

}

style\_transfer(\*\*params3)

# Muse + Tubingen

params4 = {

    'content\_image' : '%s/styles/tubingen.jpg' % (FOLDERNAME),

    'style\_image' : '%s/styles/muse.jpg' % (FOLDERNAME),

    'image\_size' : 192,

    'style\_size' : 192,

    'content\_layer' : 2,

    'content\_weight' : 6e-2,

    'style\_layers' : [0, 3, 5, 6],

    'style\_weights' : [200000, 800, 12, 2],

    'tv\_weight' : 2e-2

}

style\_transfer(\*\*params4)